پایتون برای دانش‌آموزان

یک آشنایی سرگرم‌کننده با برنامه‌نویسی

ترجمه و انتشار

خانه ریاضیات اصفهان

پایلی

خانه ریاضیات اصفهان

با حمایت دکتر ایرج هیرمن‌پور

فهرست مطالب

مقدمه ۱

بخش اول: یادگیری برنامه نویسی

فصل ۱: همه مارها نمی خزند ۶

فصل ۲: محاسبات و متغیرها ۲۲

فصل ۳: رشته ها، لیست ها، چندتایی ها و نقشه ها ۳۰

فصل ۴: نقاشی با لاک پشت ۴۷

فصل ۵: طرح سئوالات با if و else ۵۶

فصل ۶: بریم برای حلقه زدن ۶۹

فصل ۷: با توابع و ماژول ها کد خود را بازیابی کنید ۸۱

فصل ۸: چگونه از کلاس ها و شیء ها استفاده کنیم ۹۱

فصل ۹: توابع ذاتی پایتون ۱۰۷

فصل ۱۰: ماژول های سودمند پایتون ۱۲۷

فصل ۱۱: تعداد بیشتری گرافیک های لاک پشت ۱۴۳

فصل ۱۲: استفاده از tkinter برای گرافیک های بهتر ۱۵۹

بخش دوم: پرش!

فصل ۱۳: اولین بازی خود را آغاز کنید: پرش! ۱۹۰

فصل ۱۴: تکمیل اولین بازی: بالا و پایین پریدن! ۲۰۲

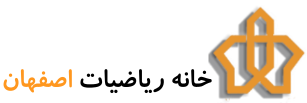
بخش سوم: مسابقه آقای خطی برای رسیدن به درب خروج

فصل ۱۵: خلق گرافیک هایی برای بازی مرد خطی.. ۲۱۶

فصل ۱۶: توسعه بازی Mr. Stick Man ۲۲۷

فصل ۱۷: خلق Mr. Stick Man...۲۴۵

فصل ۱۸: تکمیل بازی Mr. Stick Man..۲۵۳



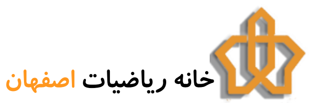
«این کتاب یک منبع آموزشی باز است. استفاده و انتشار آن در صورت ذکر نام "خانه ریاضیات اصفهان" و "پایلی" بلامانع می‌باشد.»

برای دسترسی به جدیدترین نسخه این کتاب به آدرس های زیر مراجعه نمایید:

خانه ریاضیات اصفهان

https://pylie.com/teach

پاییز ۹۶



پایتون برای بچه‌ها

معرفی برنامه‌نویسی بصورت بازی

Jason R.Briggs

مقدمه

به چه دلیل برنامه‌نویسی یاد می‌گیریم؟

برنامه‌نویسی باعث پرورش و تقویت خلاقیت، قوه استدلال و حل مسئله می‌شود. برنامه‌نویس این فرصت را پیدا می‌کند تا از هیچ چیز، چیزی را بوجود آورده و از منطق برای تبدیل سازه‌های برنامه-نویسی به فرم قابل اجرا در یک رایانه استفاده کرده و زمانیکه همه چیز طبق انتظار پیش‌نمی‌رود، از حل مسئله برای یافتن مشکل استفاده کند. برنامه‌نویسی یک سرگرمی و برخی اوقات یک فعالیت چالش-برانگیز(و گهگاه ناامیدکننده) است و مهارت‌های آموخته شده از آن هم در مدرسه و هم در محل کار مفید خواهند بود... حتی اگر در حرفه شما کاری با رایانه انجام نشود.

اگر همه این موارد را کنار بگذاریم، وقتی هوای بیرون منزل دلگیر است میتوانیم از برنامه‌نویسی برای گذراندن یک بعد از ظهر استفاده کنیم.

چرا پایتون ؟

یادگیری زبان برنامه‌نویسی پایتون بسیارساده بوده از ویژگی‌های واقعاً مفید و جذابی برای یک برنامه‌نویس مبتدی برخوردار می‌باشد. در مقایسه با دیگر زبان‌های برنامه‌نویسی، خواندن کد بسیار آسان بوده و دارای یک محیط تعاملی است که در آنجا میتوانید برنامه‌های خود را وارد کرده و اجرای آنها را مشاهده کنید. پایتون علاوه بر ساختار ساده زبانی و محیط محاوره‌ای (تعاملی) برای آزمایش، از ویژگی-های دیگری نیز برخوردار است که بطور قابل‌توجهی فرآیند یادگیری را تقویت کرده و به شما اجازه

می‌دهد تا برای خلق بازی‌های دلخواهتان، پویانمایی‌های ساده را در کنار هم قرار دهید. یکی از این ویژگی‌ها، ماژول turtle است که الهام گرفته از گرافیک Turtle (که در زبان برنامه‌نویسی Logo در دهه ۱۹۶۰ مورداستفاده بوده است) و طراحی شده با هدف آموزش بوده است. ویژگی دیگر، ماژول tkinter است؛ یک رابط[1] برای Tk GUI toolkit که شیوه‌ای ساده را برای خلق برنامههایی با پویانمایی و گرافیک‌های نسبتاً پیشرفته‌تر ارائه می‌کند.

چگونه کدنویسی را بیاموزیم؟

همانند هر چیز دیگری که برای اولین بار امتحان می‌کنید، بهترین کار شروع کردن با مبانی است پس با فصل‌های اول کار را آغاز کرده و برای رفتن به فصل‌های بعد عجله نمی‌کنیم. هیچ کس قادر نیست برای اولین بار و تنها بواسطه استفاده از وسایل، یک ارکستر سمفونی را رهبری کند. دانشجویان خلبانی تا قبل از درک کنترل‌های پایه، پرواز با هواپیما را امتحان نمی‌کنند. ژیمانست‌ها معمولاً نمیتوانند اولین بار پشتک بزنند. اگر در این کار عجله کنید، نه تنها ایده‌های اولیه و پایه در ذهن شما جا نمی‌افتند بلکه محتوای فصل‌های بعد نیز برای شما سنگین و غیرقابل درک خواهند بود.

همچنان که در این کتاب جلو می‌روید، به مثال‌ها توجه کرده و ببینید چگونه کار می‌کنند. همچنین در انتهای هر فصل، چیستان‌های برنامه‌نویسی نیز ارائه شده است تا انجام دهید زیرا به شما کمک می‌کند تا مهارت‌های برنامه‌نویسی خود را بهبود بخشید. بخاطر داشته باشید که هر چه بهتر مبانی و اصول را درک کنید، درک ایده‌های پیپده‌تر، آسانتر خواهد بود.

اگر از نظر شما چیزی ناامیدکننده یا بسیار چالش‌برانگیز است، در اینجا مواردی مطرح شده است که میتواند در تسهیل آن به شما کمک کند:

۱- مسئله را به بخش‌های کوچکتری تقسیم کنید. سعی کنید بفهمید هر تکه کد چه کاری انجام می‌دهد یا فقط روی بخش کوچکی از یک ایده دشوار تمرکز کنید (به جای اینکه سعی کنید همه چیز را به یکباره درک نمایید)

۲- اگر این کار هم کمک نکرد، برخی اوقات فقط بهتر است مدتی کاری با آن نداشته باشید و بعداً به سراغش بیایید. این راهکار برای حل بسیاری از مسائل جواب داده و بویژه برای برنامه‌نویسان رایانه بسیار مفید واقع شده است.

[1] interface

این کتاب برای چه کسانی است؟

این کتاب برای علاقمندان به برنامه‌نویسی رایانه است صرفنظر از اینکه با چه سن و سالی بـرای اولین بار به سراغ برنامه‌نویسی آمده باشد. اگر میخواهید یادبگیرید چطور نرم‌افزار دلخواهتان را خودتـان خلق کنید، به جای این که از برنامه‌های توسعه یافته توسط دیگران استفاده کنیـد، پـایتون بـرای بچـه‌هـا میتواند نقطه آغاز بسیار خوبی باشد.

در فصل‌های بعد، اطلاعاتی دریافت خواهید کرد که میتواند به شما در نصب پایتون، بـالا آوردن محیط پایتون و انجام محاسبات پایه، چاپ متن روی صفحه نمایش و ایجاد لیست، و کنترل در حد ساده روی جریان عملیات‌ها با استفاده از دستورات if و حلقه‌هایfor (همچنین یاد می‌گیرید که دستوراتif و حلقه‌هایfor چه چیزی هستند)کمک کند.خواهید آموخت که چگونه با استفاده از توابع، مبانی کلاس‌هـا[1] و شی‌ها[2] و توصیف برخی از چندین تابع و ماژول ذاتی پایتون مجدداً از کد استفاده کنید.

فصل‌هایی نیز به گرافیک‌های ساده و پیچیده لاک‌پشت و استفاده از ماژولtkinter بـرای ترسـیم روی صفحه رایانه اختصاص داده شده‌اند. در انتهای هر فصل نیز چیستان‌های برنامـه‌نویسـی بـا سـطوح مختلف پیچیدگی ارائه می‌شوند که به خواننده کمک می‌کنند تا دانش تازه کسب کرده خود را با نوشـتن برنامه‌های کوچک تحکیم کند.

بعد از برپا کردن دانش زیرساختی برنامه‌نویسی، خواهید آموخت که چگونه میتوانید برنامه‌هـای دلخواهتان را بنویسید. دو بازی گرافیکی را توسعه داده و درباره تشـخیص برخـورد(تلاقی)[3]، رخـدادها و تکنیک‌های مختلف پویانمایی چیزهایی می‌آموزید.

در اکثر مثال‌های این کتاب از محیطIDLE (محیط توسعه یکپارچه[4])پایتون اسـتفاده شـده اسـت. IDLE ترکیبی[5]را ارائه می‌کند که روی کارکرد کپی-درج[6](مشـابه آنچـه در برنامـه‌هـای کـاربردی دیگـر انجام می‌دهید)و یک پنجره ویرایشگر تأکید دارد که در آنجا میتوانید کد خود را برای استفاده آتی، ذخیره کنید، به این معنی کهIDLE هم بعنوان یک محیط محاوره‌ای برای آزمایش و هم بصـورت چیـزی شـبیه ویرایشگر متن عمل می‌کند. مثال‌ها منطبق با کنسول استاندارد و یک ویرایشـگر مـتن عـادی بـوده ولـی

[1] class
[2] object
[3] Collision detection
[4] Integrated DeveLopment Environment
[5] Syntax
[6] Copy-paste

ترکیب و محیط کاربرپسند[1]نسبتاً متفاوت IDLE میتواند به درک کمک کند بنابراین در فصل اول به شما نشان می‌دهیم که چگونه بایستی آن را برپا کنید.

این کتاب حاوی چه مطالبی است؟

در اینجا بطور مختصر محتوای هر فصل معرفی شده است:

فصل اول مقدمه‌ای است بر برنامه‌نویسی با دستورات نصب پایتون برای اولین بار.

فصل دوم به معرفی متغیرها و محاسبات پایه می‌پردازد و

فصل سوم برخی نوع‌های اصلی در پایتون از قبیل رشته‌ها[2] لیست‌ها و چندتایی[3]را شرح می-دهد.

در فصل چهارم برای اولین بار با ماژول turtle آشنا می‌شویم. از برنامه‌نویسی پایه جدا شده و یک لاک‌پشت را در صفحه نمایش(به شکل یک فلش)حرکت می‌دهیم.

در فصل پنجم، انواع شرطها و دستورات if ارائه شده و در فصل ششم به سراغ حلقه‌های for و حلقه‌های while می‌رویم.

فصل هفتم جایی است که استفاده و خلق توابع را آغاز کرده و در فصل هشتم، کلاس‌ها و شیءها معرفی می‌شوند. همچنین به حد کافی به ایده‌های اصلی برای پشتیبانی از برخی تکنیک‌های برنامه‌نویسی موردنیاز در فصل‌های بعد(مربوط به توسعه بازی)خواهیم پرداخت. در اینجا، موضوعات کم کم پیچیده‌تر می‌شوند.

فصل نهم به مرور اکثر توابع جزء ساختار پایتون پرداخته فصل دهم به معرفی چند ماژول می-پردازد که (اساساً مجموعه‌ای از کارکردهای سودمند)بصورت پیش‌فرض همراه با پایتون نصب می‌شوند.

فصل یازدهم مجدداً به سراغ ماژول turtle رفته تا خواننده شکل‌های پیچیده‌تری را تجربه کند.

فصل دوزادهم، به سراغ استفاده از ماژول tkinter برای خلق گرافیک‌های پیشرفته‌تر می‌رود.

در فصل‌های سیزدهم و چهاردهم، اولین بازی خودمان به نام "Bounce!," را می‌سازیم که براساس دانش بدست آمده از فصل‌های قبل بنا شده است.

در فصل‌های پانزدهم تا هجدهم، بازی دیگری به نام "Mr. Stick Man Races for the Exit." را می‌سازیم. فصل‌های مربوط به توسعه بازی جایی هستند که ممکن است اشتباهات بسیاری رخ دهد. اگر

---

[1] userfriendly

[2] String

[3] tuple

همه چیز خراب شد میتوانید کد را از وب سایت به آدرس *http://python-for-kids.com/* دانلود کـرده و کد خود را با مثال‌های درست مقایسه کنید.

در بخش نتیجه‌گیری، با نگاهی بهPyGame و برخی دیگر از زبان‌هیا برنامه‌نویسی مشـهور، کـار را خاتمه می‌دهیم.

درنهایت، در ضمیمه[1] اطلاعات مفصلی درباره کلید واژه‌های پایتون دراختیار شما قرار گرفتـه و در واژه‌نامه[2] تعاریف اصطلاحات برنامه‌نویسی بکاررفته در سراسر این کتاب ارائه خواهد شد.

وب سایت راهنما

اگـر در حـین خوانـدن مطالـب بـه کمـک نیـاز پیـدا کردیـد میتوانیـد بـه وب سـایت راهنمـا *http://python-for-kids.com/,* سر بزنید جاییکه امکان دانلود تمامی مثال‌های این کتاب و تعداد بیشتری چیستان برنامه‌نویسی وجود دارد. همچنین اگر در پاسخ به سئوالات درمانـده‌ایـد، میتوانیـد بـه ایـن وب سایت مراجعه کرده و پاسخ تمامی چیستان‌های برنامه‌نویسی این کتاب را مشاهده کنید.

خوش بگذره!

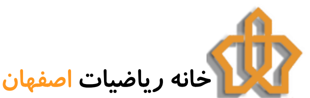
همچنان که با این کتاب پیش می‌روید بخاطر داشته باشید که برنامه‌نویسی میتواند سرگرم‌کننـده باشد. به برنامه‌نویسی بعنوان یک کار نگاه نکنید. برنامه‌نویسی را بعنوان راهـی بـرای خلـق برنامـه‌هـای کاربردی بازی‌های سرگرم‌کننده‌ای درنظر بگیرید که میتواند آن را با دوستان خود یا دیگران بـه اشـتراک بگذارید.

یادگیری نوشتن برنامه یک تمرین ذهنی جالب بوده و نتایج بسیار رضایتبخش و ارزشمند خواهند بود. ولی اساساً هر کاری که انجام دهید، سرگرم خواهید شد!

---

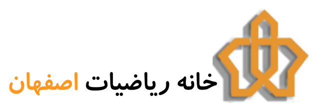
[1] Appendix

[2] glossary



بخش اول

یادگیری نوشتن برنامه



فصل اول

همه مارها نمی‌خزند

یک برنامه رایانه‌ای مجموعه‌ای از دستورات است که باعث می‌شوند رایانه کاری را انجام دهد.برنامه، بخش‌های فیزیکی یک رایانه نیست– مثلاً سیم‌ها، ریزتراشه‌ها[1] کارت‌ها، دیسک سخت[2]و .... – بلکه چیزی است که بطور پنهان روی آن سخت‌افزار اجرا می‌گردد. یک برنامه رایانه‌ای، که معمولاً بطور مختصر تحت عنوان یک برنامه[3]شناخته می‌شوند، مجموعه‌ای از فرمان‌ها می‌باشد که به سخت‌افزار زبان بسته می‌گوید چه کاری انجام دهد. نرم‌افزار[4]مجموعه‌ای از برنامه‌های رایانه‌ای است.

بدون برنامه‌های رایانه‌ای، تقریباً هر وسیله‌ای که بطور روزمره استفاده می‌کنید، بلااستفاده بوده یا کارایی کمتری خواهد داشت. برنامه‌های رایانه‌ای به هر فرمی که باشند نه فقط رایانه شخصی شما بلکه سیستم‌های بازی ویدئویی، تلفن‌های همراه و دستگاه‌هایGPS درون خودرو را نیز کنترل می‌کنند. نرم-افزار همچنین آیتم‌های نه چندان مشهود همچون تلوزیون‌های ال سی دی و دستگاه کنترل از راه دور آنها و همچنین برخی از جدیدترین رادیوها، دستگاه‌های پخش دی وی دی، اجاق گاز، و برخی یخچال-ها را کنترل می‌کنند. حتی موتور ماشین‌ها، چراغ‌های راهنمایی، روشنایی خیابان، علامت‌های قطار، تابلوهای اعلانات الکترونیکی و آسانسورها نیز توسط برنامه‌ها کنترل شده‌اند.

---

[1] microchip

[2] Hard drive

[3] program

[4] software

برنامه‌ها تاحدودی شبیه فکر کردن هستند. اگر تفکر نکنید، فقط روی زمین نشسته، بیکار به جایی خیره شده و آب دهانتان روی لباستان سرازیر می‌شود. تفکر شما مبنی بر«بلند شدن از روی زمین» یک دستور یا فرمان است که به بدن شما می‌گوید از جا بلند شود. به همین ترتیب برنامه‌های رایانه‌ای نیز به رایانه می‌گویند که چه کاری را انجام دهد.

اگر نوشتن برنامه‌های رایانه‌ای را بلد باشید میتوانید هر نوع کار سودمندی را انجام دهید. مطمئناً نمیتوانید برنامه‌هایی برای کنترل خودروها، چراغ‌های راهنمایی یا یخچال بنویسید(حداقل نه الان)بلکه میتوانید صفحات وب شخصی، بازی‌ها خودتان یا حتی برنامه‌ای برای کمک کردن به انجام تکالیف‌تان را بنویسید.

چند کلامی درباره زبان

رایانه‌ها نیز همانند انسان‌ها از چندین زبان برای برقراری ارتباط استفاده می‌کنند- در این مورد میتوان به زبان‌های برنامه‌نویسی اشاره نمود. یک زبان برنامه‌نویسی یک شیوه خاص برای صحبت کردن با یک رایانه است- راهی برای استفاده از دستورات که هم برای انسان و هم برای رایانه قابل درک باشد.

برخی زبان‌های برنامه‌نویسی به نام انسان‌ها(مثلاًAda وPascal)، برخی با نام‌های مخفف(مثلاًBASIC وFORTRAN) و حتی برخی به نام نمایش‌های تلویزیونی از جملهPython نامگذاری شده‌اند. بله زبان برنامه‌نویسی پایتون نام برنامه تلویزیونی*Monty Python's Flying Circus* است نه مار پایتون.

نکته: Monty Python's Flying Circus یک نمایش کمدی انگلیسی است که اولین بار در دهه ۱۹۷۰ پخش شد و امروزه در میان مخاطبین خاصی جایگاه ویژه‌ای دارد. این نمایش شامل انگاره-هایی همچون*"The Fish-Slapping Dance," "The Ministry of Silly Walks,"* و*"The Cheese Shop"* بود(که هیچ پنیری نمی‌فروخت).

زبان برنامه‌نویسی پایتون از ویژگی‌هایی برخوردار است که باعث شده تا برای مبتدیان فوق‌العاده سودمند باشد. مهمتر از همه، میتوانید از پایتون استفاده کرده و به سرعت برنامه‌های ساده و مؤثری بنویسید. پایتون نمادهای پیچیده‌ای ندارد، مثلاً کروشه({ })، هشتک(#)[1]، و علامت دلار($) که باعث می-شوند خواندن دیگر زبان‌های برنامه‌نویسی دشوار شده و بنابراین چندان موردتوجه مبتدیان قرار نگیرند.

[1] hashes

نصب پایتون

نصب پایتون نسبتاً ساده و سرراست است. دراینجا مراحل نصب آن در ویندوز۷ (Windows 7)، مکینتاش(Mac OS X) و اوبونتو(Ubuntu) را مرور می‌کنیم. هنگام نصب پایتون، میانبری نیز برای برنامهIDLE ایجاد خواهید کرد جاییکه محیط توسعه یکپارچه(IDLE)[۱]به شما اجازه می‌دهد برنامه‌هایی را برای پایتون بنویسید. اگر پایتون روی رایانه شما نصب شده است، مستقیماً به صفحه۱۰ بروید «بعد از اینکه پایتون را نصب کردید»

نصب پایتون در ویندوز۷

برای نصب پایتون در مایکروسافت ویندوز۷، به آدرس *http://www.python.org/* رفته و آخرین نسخه برنامه نصبPython3 تحت ویندوز را دانلود کنید. در لیست بدنبال عنوان**Quick Links** باشید:

نکته: نسخه دقیق پایتون که دانلود می‌کنید تا مادامیکه با عدد۳ آغاز می‌شود،اهمیتی ندارد.

بعد از اینکه برنامه نصب تحت ویندوز را دانلود کردید، روی آیکون آن دوبار کلیک کرده و دستورات را برای نصب پایتون در محل پیش‌فرض دنبال کنید:

۱- **Install for All Users** را انتخاب کرده و رویNext کلیک کنید

۲- کاری با دایرکتوری پیش‌فرض نداشته باشید ولی به نام دایرکتوری نصب توجه کنید(احتمالاً *C:\Python31*یا*C:\Python32*).رویNext کلیک نمایید.

۳- درطی نصب از بخشCustomize Python صرفنظر کرده و روی Next کلیک کنید

در انتهای این فرآیند، بایستی یک ورودیPython3 در منوی شروع(Start menu) داشته باشید:

---

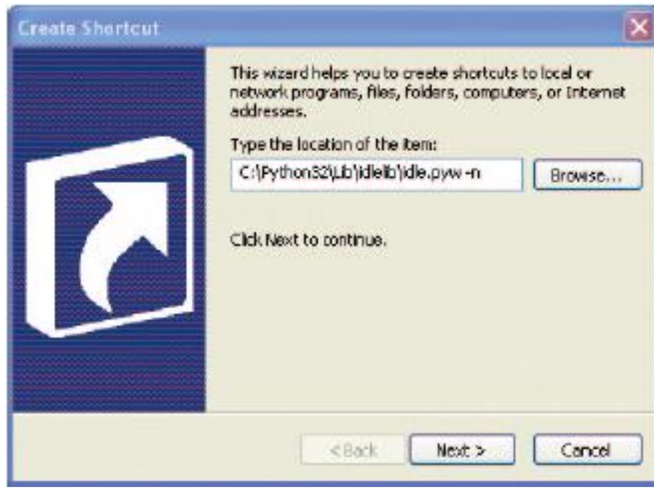
[۱] *I*ntegrated *D*eve*L*opment *E*nvironment

سپس مراحل زیر را برای افزودن میانبر Python3 به دسکتاپ(desktop) دنبال کنید:

۱- روی دسکتاپ راست کلیک کرده و **New ▸ Shortcut** را از منوی گزینه‌ها[1] انتخاب کنید. درجاییکه از شما خواسته می‌شود **Type the location of the item**(محل آیتم را تایپ کنید) (اطمینان حاصل کنید که دایرکتوری که وارد می‌کنید همان دایرکتوری باشد که قبلاً ذکر کرده‌اید) موارد زیر را وارد کنید:

c:₩Python32₩Lib₩idlelib₩idle.pyw –n

دیالوگ(کادر) شما بایستی بصورت زیر باشد



۳- روی Next کلیک کرده و به دیالوگ بعدی بروید.

٤- نام را بصورت IDLE وارد کرده و روی Finish کلیک کنید تا میانبر ایجاد شود.

[1] Pop-up

حال میتوانید به بخش«بعد از اینکه پایتون را نصب کردید» در صفحه ۱۰ رفته و کار با پایتون را آغاز کنید.

نصب پایتون درMAC OS X

اگر از یکMac استفاده می‌کنید باید نسخه پیش‌نصب شده پایتون را پیدا کنید ولی احتمالاً این نسخه یک نسخه قدیمی از این زبان خواهد بود. برای اینکه مطمئن شوید جدیدترین نسخه را اجرا می-کنید، به آدرس *http://www.python.org/getit/* رفته تا آخرین برنامه نصب را برایMac دانلود کنید.

دو برنامه نصب متفاوت در آنجا مشاهده خواهید کرد. نسخه‌ای که باید دانلود کنید به نسخه سیستم‌عاملMac OS X شما وابسته است(برای اینکه بفهمید از چه نسخه سیستم‌عاملی استفاده می‌کنید روی آیکونApple در نوار منوی بالا کلیک کرده و About this Mac را انتخاب کنید). بصورت زیر یک برنامه نصب انتخاب نمایید:

- اگر نسخه سیستم‌عاملMac OS X شما بین10.3 و10.6 است،نسخه ۳۲بیتی Python3 را برای i386/PPC دانلود کنید.
- اگر نسخه سیستم‌عاملMac OS X شما 10.6 یا بالاتر است،نسخه ٦٤بیتی/۳۲بیتی Python3 را برای x86-64 دانلود کنید.

بعد از اینکه فایل دانلود شد(باید با پسوندdmg. باشد)روی آن دوبار کلیک کنید.پنجره‌ای نمایش داده خواهد شد که آن محتوای فایل را می‌بینید.

در این پنجره دوبار روی Python.mpkg کلیک کرده و سپس دستورات را برای نصب نرم‌افزار دنبال کنید. برای نصب پایتون از شما کلمه‌عبور مدیر سیستمMac پرسیده می‌شود (اگر کلمه‌عبور مدیر را دراختیار ندارید، از والدین خود بخواهید آن را وارد کنند).

سپس بایستی یک اسکریپت به دسکتاپ اضافه کرده تا برنامه‌کاربردیIDLE مربوط به پایتون اجرا شود:

۱- روی آیکون Spotlight کلیک کنید، یک ذره‌بین[1] کوچک در گوشه سمت راست بالای صفحه نمایش.

۲- در کادر(جعبه)ی که ظاهر می‌شود، Automator را وارد کنید.

۳- با ظاهر شدن یک برنامه‌کاربردی شبیه ربات در منو، روی آن کلیک کنید. ممکن است در بخشی با عنوان Top Hit یا در applications باشد.

٤- با شروع Automator ، الگوی Application را انتخاب کنید.

۵- Choose را برای ادامه انتخاب کنید.

٦- در فهرست اقدامات، Run Shell Script را پیدا کرده و آن را به صفحه خالی در سمت راست بکشید:

۷- در جعبه متن، کلمه cat را مشاهده خواهید کرد. کلمه را انتخاب کرده و متن زیر را جایگزین آن کنید(هر چیزی از open تا n-):

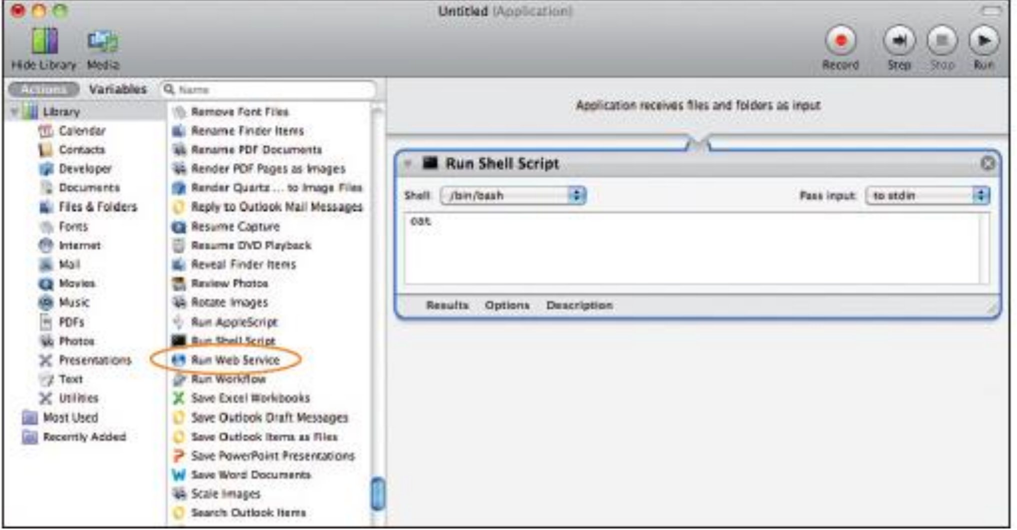
```
open -a "/Applications/Python 3.2/IDLE.app" --args -n
```

با توجه به نسخه پایتون ی که نصب کرده‌اید شاید لازم باشد دایرکتوری را تغییر دهید.

۸- **File ▸ Save** را انتخاب کرده و IDLE را بعنوان نام وارد کنید.

۹- Desktop را از دیالوگ(کادر) Where انتخاب کرده و روی Save کلیک کنید.

[1] Magnifying Glass



حال میتوانید به بخش«بعد از اینکه پایتون را نصب کردید» در صفحه ۱۰ رفته و کار با پـایتون را آغاز کنید.

نصب پایتون در اوبونتو

پایتون بصورت پیش‌فرض روی لینوکس اوبونتو نصب شده است ولی ممکـن اسـت نسـخه آن قدیمی باشد. برای نصبPython3 روی Ubuntu 12.x مراحل زیر را دنبال کنید:

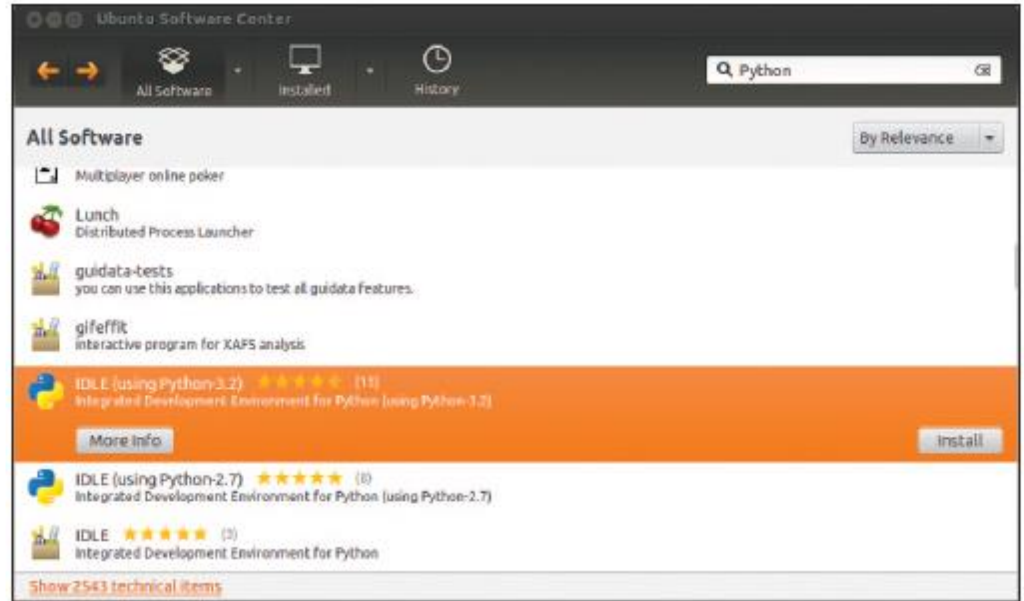
۱- روی دکمهUbuntu Software Center در نوار کناری کلیک کنید(آیکونی که شبیه یک کیـف نارنجی است- اگر آن را مشاهده نکردید میتوانید روی آیکونDash Home کیک کـرده و *Software* را در کادر وارد کنید)

۲- در کادر جستجو در گوشه بالا سمت راستSoftware Center، *Python* را وارد کنید.

۳- در لیست نرم‌افزار نمایش داده شده، آخرین نسخهIDLE را انتخاب کنید کـه در ایـن مثـال، *IDLE (using Python 3.2)* می‌باشد:

٤- رویInstall کلیک کنید.

۵- برای نصب نرم‌افزار باید کلمه عبور مدیر را وارد کنید و سـپس روی**Authenticate** کلیـک نمایید. (اگر کلمه عبور مدیر را در اختیار ندارید میتوانید از والـدین خـود خـواهش کنیـد تـا آن را وارد کنند).



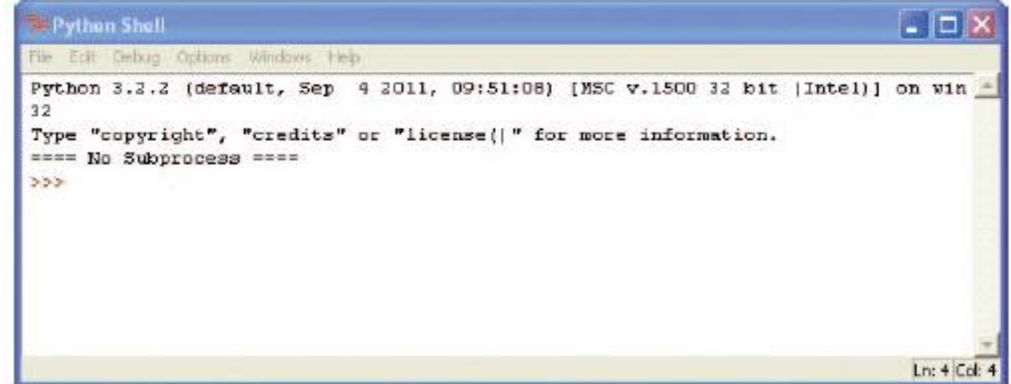
نکته: در برخی نسخه‌های اوبونتو در منوی اصلی، Python (v3.2) را (به جای IDLE) مشاهده کرده که میتوانید آن را نصب کنید.

حال که آخرین نسخه پایتون را نصب کردید، آن را امتحان می‌کنیم.

بعد از نصب پایتون

حال باید یک آیکون روی دسکتاپ ویندوز یا Mac OS X شما با عنوان IDLE قرار گرفته باشد. اگر از اوبونتو استفاده می‌کنید، در منوی Applications بایستی گروه جدیدی به نام **Programming** را برنامه‌کاربردی **IDLE (using Python 3.2)** (یا نسخه دیگری) را مشاهده کنید.

روی آیکون دوبار کلیک کرده یا گزینه منو را انتخاب کنید، درنتیجه بایستی پنجره زیر را مشاهده کنید:



این رابط پایتون است که بخشی از محیط توسعه یکپارچه پایتون می‌باشد. سه تا علامت بزرگتر(>>>)،prompt(اعلان آمادگی) نامیده می‌شود.

برای اینکه فرمان‌هایی را درPrompt وارد کنیم با کد زیر شروع می‌کنیم:

```
>>> print("Hello World")
```

مطمئن شوید که ازعلامت نقل‌قول دوتایی(" ")استفاده کرده‌اید. هنگامی که تایپ خط اول تمام شد، دکمهENTER را روی صفحه کلید فشار دهید. اگر فرمان را درست وارد کرده باشید، بایستی خروجی زیر را مشاهده کنید:

```
>>> print("Hello World")
Hello World
>>>
```

Prompt بایستی مجدداً ظاهر شود تا بفهمید که رابط پایتون آماده پذیرش فرمان‌های بیشتری است.

تبریک می‌گوییم! شما اولین برنامه پایتون خود را نوشتید. کلمهprint یک نوع از فرمان پایتون است که یک تابع نامیده می‌شود و هرآنچه درون پرانتز قرارداشته باشد را روی صفحه نمایش چاپ می‌کند. اساساً، شما به رایانه دستور می‌دهید که کلمات"Hello Word" را نمایش دهد– دستوری که هم شما و هم رایانه آن را درک می‌کنید.

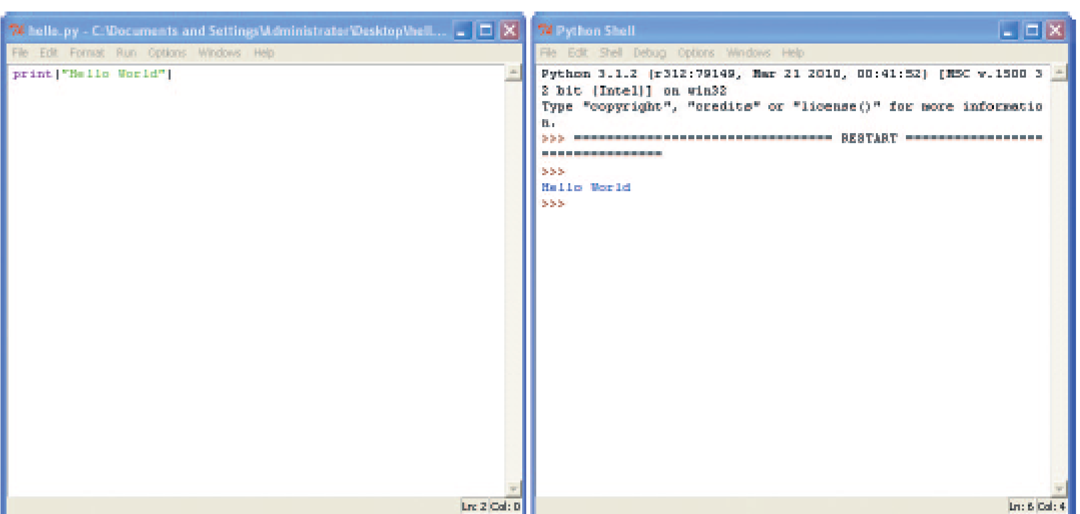
ذخیره برنامه‌های پایتون

اگر هربار نیاز به بازنویسی برنامه‌های پایتون داشته باشید،این برنامه‌ها چندان مفید نخواهند بود بنابراین اهمیت ندهید، میتوانید آنها را چاپ کنید و هربار که خواستید به آنها رجوع نمایید.مطمئناً، بازنویسی برنامه‌های کوتاه کار سختی نیست ولی برنامه بزرگی همچون پردازشگر متن، میتواند حاوی میلیون‌ها خط کد باشد. اگر آنها را چاپ کنید، ده‌ها هزار صفحه کد خواهید داشت. تصور کنید بخواهید این حجم کاغذ را با خود به خانه ببرید. پس بهتر است آرزو کنید هیچگاه در چنین وضعیتی گرفتار نشوید.

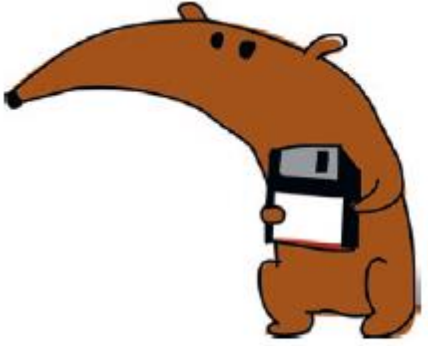
خوشبختانه میتوانیم برنامه‌ها را برای استفاده‌های آتی ذخیره کنیم. برای ذخیره کرده یک برنامه جدید، IDLE را باز کرده و **File ▸ New Window** را انتخاب کنید. یک پنجره خالی با عنوان ***Untitled*** در نوار منو ظاهر خواهد شد. کد زیر را در پنجره رابط جدید وارد کنید:

```
print("Hello World")
```

حال **File ▸ Save** را انتخاب کنید. زمانیکه از شما نام فایل خواسته می‌شود،*hello.py* را وارد کرده و فایل را روی دسکتاپ ذخیره کنید. سپس **Run ▸ Run Module** را انتخاب کنید. بدون نیاز به شانس، برنامه ذخیره شده شما اجرا خواهد شد:



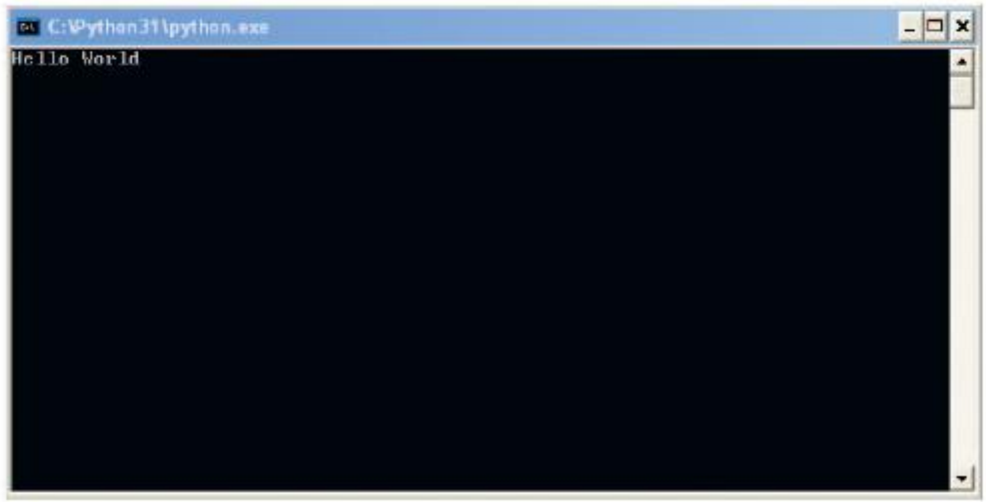
حال اگر پنجره رابط را بسته ولی پنجره hello.py همچنان باز باشد و **Run ▸ Run Module** را انتخاب کنید، رابط پایتون مجدداً باید ظاهر شود و برنامه شما دومرتبه اجرا گردد. (برای اینکه رابطه پایتون بدون اجرای برنامه دوباره باز شود، **Run ▸ Python Shell** را انتخاب کنید)



بعد از اجرای کد، آیکون جدیدی روی دسکتاپ با

عنوانhello.py ظاهر خواهد شد. اگر دوبار روی آیکون کلیک کنید، یک پنجره سیاه برای لحظاتی ظاهر و سپس محو می‌گردد. چه اتفاقی افتاده است؟

شما شاهد بالا آمدن کنسول خط–فرمان پایتون (مشابه رابط(shell))، چاپ"Hello Word" و سپس خروج هستید. اگر دید مافوق بشری داشته باشید میتوانید قبل از بسته شدن پنجره، آن را ببینید:



علاوه بر منوها، میتوانید از میانبرهای صفحه‌کلید نیز برای ساختن یک پنجره رابط جدید، ذخیره یک فایل و اجرای یک برنامه استفاده کنید.

- در ویندوز و اوبونتو، ازctrl-N برای ایجاد یک پنجره رابط جدید، ازctrl-S برای ذخیره فایل بعد از خاتمه ویرایش استفاده کرده، واز فشار کلیدF5 برای اجرای برنامه استفاده کنید.
- درMac OS X از⌘-N برای ساختن یک پنجره رابط جدید، از⌘-S برای ذخیره فایل استفاده کرده و با نگهداشتن کلیدFN و فشارF5 برنامه خود را اجرا کنید.

نصب برنامه پایچارم (اختیاری)

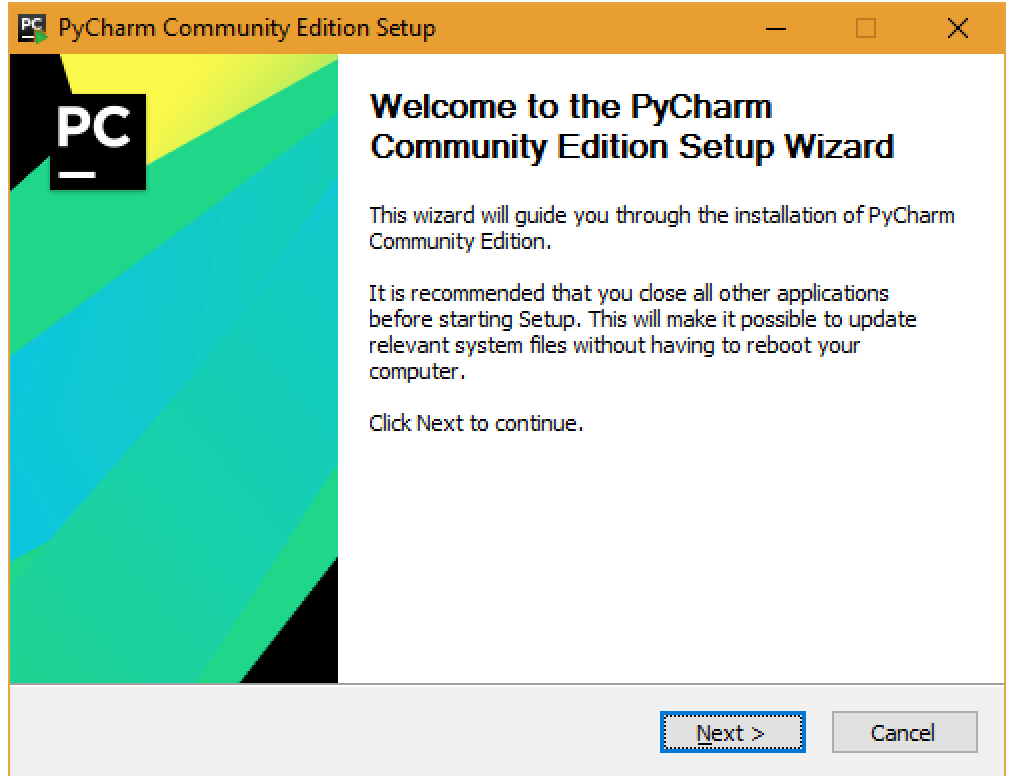
برای داشتن یک محیط گرافیکی با امکانات بیشتر می‌توان از برنامه پایچارم هم بهره برد. برنامه پایچارم یکی از قدرتمندترین فضاهای برنامه‌نویسی برای پایتون هست. استفاده از آن برای این کتاب اما کاملا اختیاری است.

اینجا نصب آن بر روی کامپیوتر با سیستم عامل ویندوز را توضیح می‌دهیم.

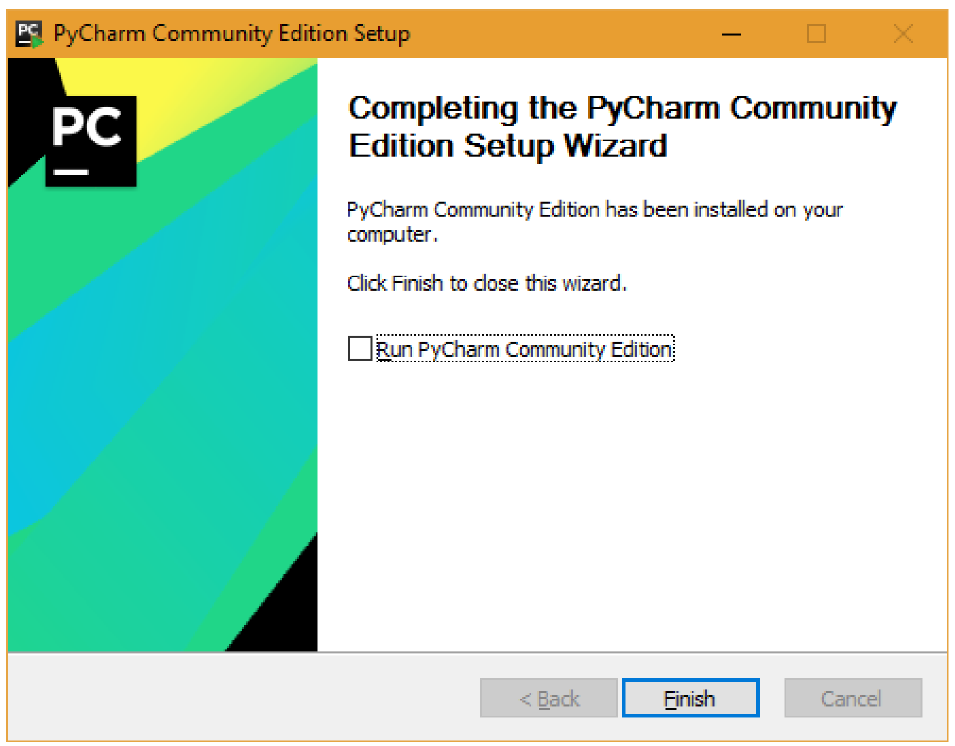
برای دانلود اینجا را کلیک کن:

https://www.jetbrains.com/pycharm/download/#section=windows

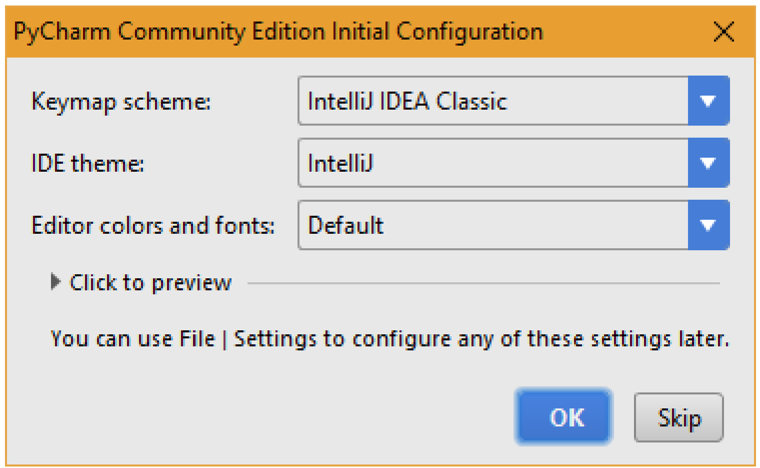
پس از دانلود نسخه رایگان Community، فایل را اجرا کن. پنجره زیر باز می‌شود:



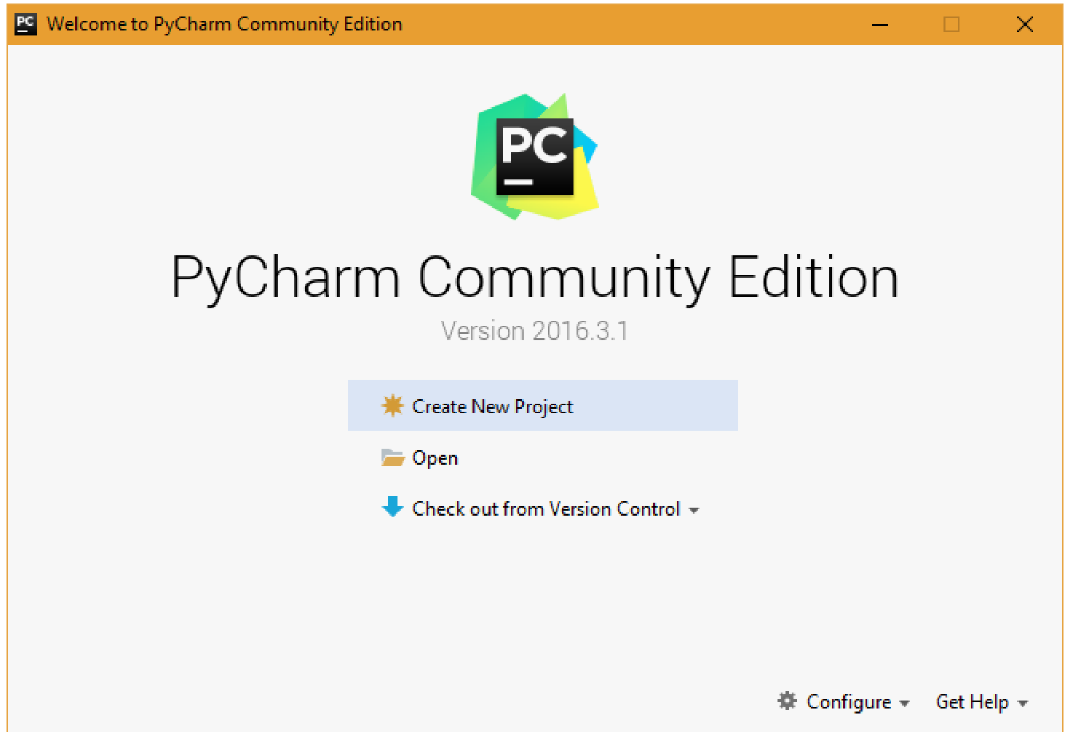
بر روی Next کلیک کن و مرحله‌های بعدی نصب را انجام بده. در صورت نصب درست، در پایان این پنجره را می‌بینی:



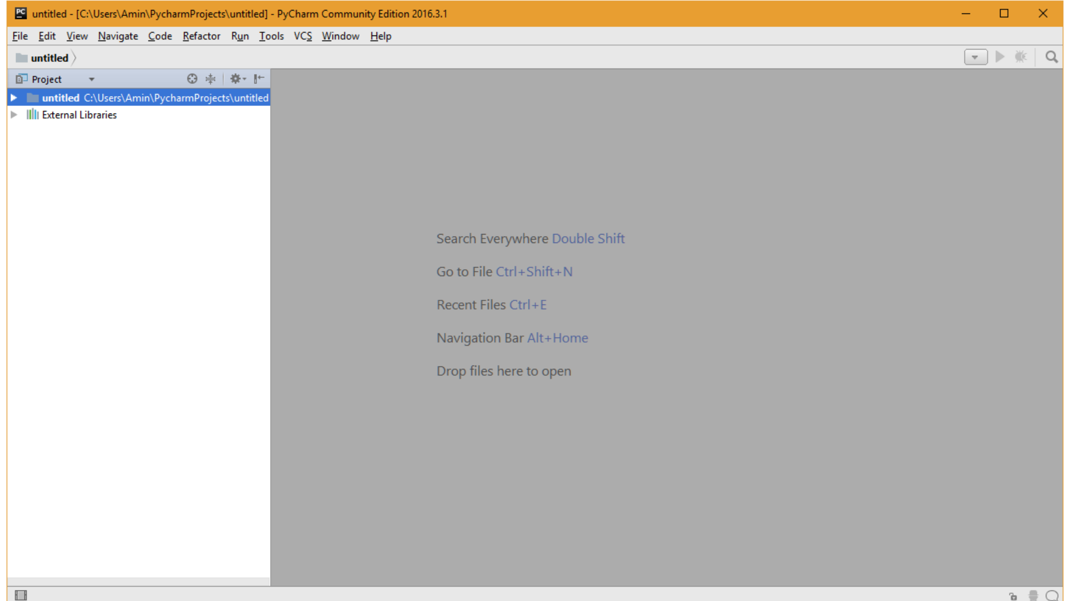
پایچارم را از منوی استارت ویندوز باز کن. پنجره‌های زیر باز می‌شوند:



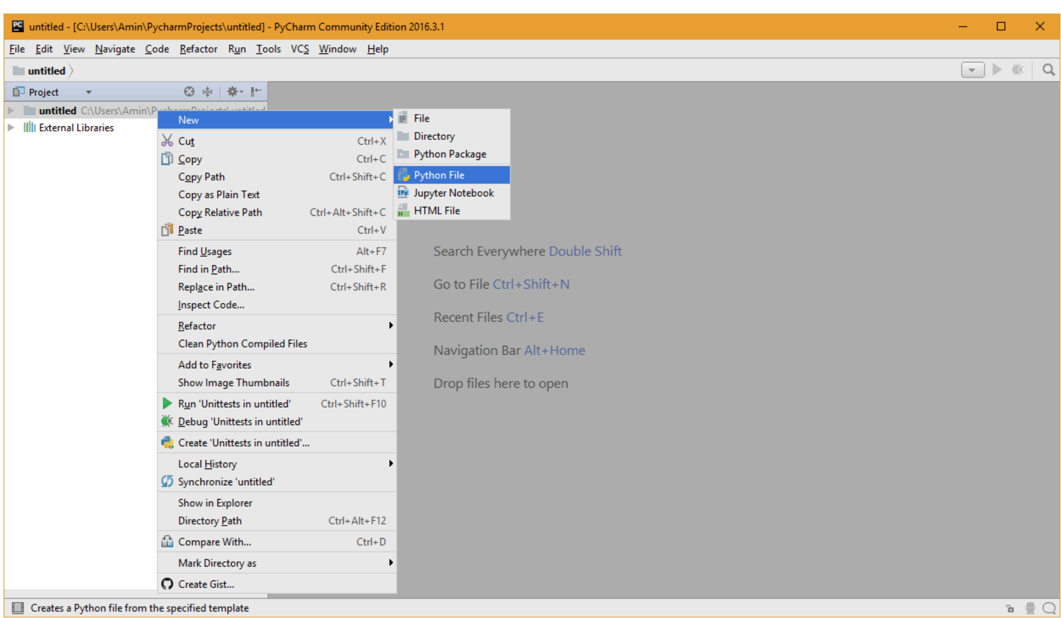
روی OK کلیک کن.



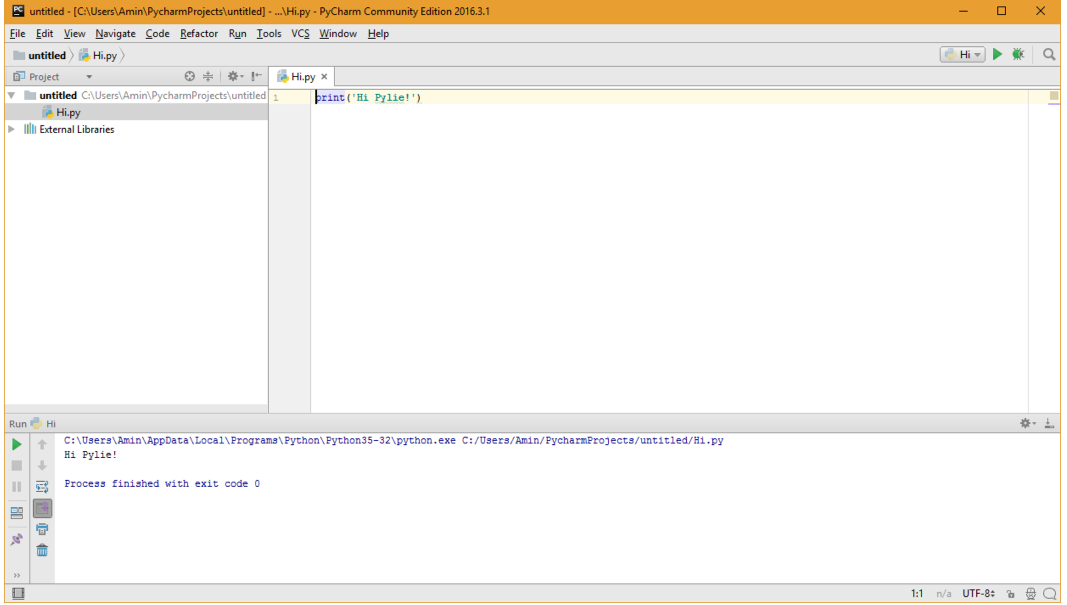
روی Create New Project و سپس Create کلیک کن. پایچارم باز می‌شود:



حالا سمت چپ و بالا روی پروژه untitled راست-کلیک کن و مثل شکل زیر یک فایل جدید پایتون بساز:



در پایان یک خط کد مثل شکل زیر را بنویس و از منو روی Run کلیک کن تا برنامه اجرا شود:



استفاده از پایلی‌باکس (اختیاری)

پایلی‌باکس یک محیط یکپارچه لینوکس هست که پایتون، پایچارم و خیلی از ابزارهای لازم برای برنامه‌نویسی‌های پیشرفته مانند ساخت اپلیکیشن موبایل روی آن از پیش نصب و آماده شده است. پایلی‌باکس را می‌توان به هر کامپیوتری هم جابجا و با آن کار کرد.

برای این کتاب اما استفاده از پایلی‌باکس الزامی نیست و استفاده از آن اختیاری است. در صورت تمایل، برای آشنایی و استفاده از آن به اینجا مراجعه کن:

https://pylie.com/course/pyliebox_intro

آنچه آموختید

در این فصل کار را با برنامه‌کاربردیHello Word آغاز کردیم- برنامه‌ای که تقریباً هرکسی برای آموختن برنامه‌نویسی رایانه، با آن شروع می‌کند. در فصل بعد کارهای مفیدتری را با رابط پایتون انجام خواهیم داد.

فصل دوم

محاسبات و متغیرها[۱]

حال که پایتون را نصب کردید و فهمیدید که چطور باید برنامه پایتون را اجرا کنید، آماده هستید تا کاری با آن انجام دهید. کار را با برخی محاسباتی ساده آغاز کرده و بعد به سراغ متغیرها می-رویم. متغیرها راهی برای روایت چیزها در یک برنامه رایانه‌ای هستند و میتوانند به شما در نوشتن برنامه‌های سودمند کمک کنند.

محاسبه با پایتون

بطور معمول زمانی که از شما خواسته می‌شود ضرب دو عدد مثلاً ۸×۳,۷۵ را پیدا کنید، از یک ماشین‌حساب یا یک قلم و کاغذ استفاده خواهید کرد. خوب، نظرتان درباره استفاده از برنامه[۲] پایتون برای انجام محاسبات‌تان چیست؟ اجازه دهید امتحان کنیم.

با دوبار کلیک کردن روی آیکون IDLE روی بخش رومیزی[۳]، برنامه پایتون را اجرا کرده یا درصورتیکه از Ubuntu استفاده می‌کنید، روی آیکون IDLE در منوی Application کلیک کنید. در prompt[۴] معادله[۵] (تساوی) زیر را وارد کنید:

```
>>> 8 * 3.57
28.56
```

---

۱ Variable

۲ Shell برنامه واسط

۳ desktop

۴ پنجره ورود اطلاعات

۵ equation

توجه داشته باشید که هنگام ورود محاسبه ضرب در پایتون، از نماد ستاره(*)به جای علامت ضرب(×)استفاده می‌شود.

اگر بخواهیم یک رابطه سودمندتر بنویسیم باید چه کار کنیم؟ فرض کنید در حال کندن باغچه حیاط پشتی هستید که ناگهان یک کیف با ۲۰ سکه طلا پیدا می‌کنید. روز بعد،یواشکی به زیرزمین رفته و سکه‌ها را در اختراع تکراری موتور بخار پدربزرگتان جاسازی می‌کنید(خیلی خوش‌شانس هستید که اندازه ۲۰ سکه جا دارد). صدای غژغژ و ترکیدنی را می‌شنوید و چند ساعت بعد ۱۰ سکه دیگر سوسو می‌زنند.

اگر هر روز سال این کار را تکرار کنید در صندوقچه خود چقدر سکه خواهید داشت؟ روی کاغذ، معادلات اینگونه خواهند بود:

$$10 \times 365 = 3650$$
$$20 + 3650 = 3670$$

مطمئناً انجام این محاسبات روی کاغذ یا ماشین حساب بسیار ساده است ولی میتوانید تمامی این محاسبات را با برنامه پایتون هم انجام دهید. ابتدا، ۱۰ سکه را در ۳٦۵ روز یک سال ضرب کرده تا عدد ۳٦۵۰ بدست آید. سپس با۲۰ سکه اصلی جمع کرده تا عدد ۳٦۷۰ بدست آید.

```
>>> 10 * 365
3650
>>> 20 + 3650
3670
```

حال، اگر یک کلاغ برای طلاهایی درخشان شما منقار تیز کرده، هر هفته داخل اتاق خواب شما بشود و سه سکه بدزدد، قضیه چگونه خواهد بود؟ در پایان سال چند سکه برای شما باقی خواهد ماند؟ ظاهر محاسبه در برنامه پایتون اینگونه خواهد بود:

```
>>> 3 * 52
156
>>> 3670 - 156
3514
```

ابتدا،۳ سکه را در ۵۲ هفته در سال ضرب می‌کنیم. نتیجه عدد ۱۵٦ خواهد بود. این عدد را از کل سکه‌ها(یعنی ۳٦۷۰)کسر می‌کنیم درنتیجه خواهیم دید که در پایان سال ۳۵۱٤ سکه باقی می‌ماند.

این یک برنامه بسیار ساده است. در این کتاب، خواهید آموخت که چگونه میتوانید این ایده‌ها را برای نوشتن برنامه‌های سودمندتر بسط و گسترش دهید.

عملگرهای[1] پایتون

شما میتوانید ضرب[2]، جمع[3]، تفریق[4] و تقسیم[5] را در کنار دیگر عملیات‌های ریاضی که اکنون کاری با آنها نداریم، در برنامه پایتون انجام دهید. نمادهای اصلی مورداستفاده در پایتون برای انجام عملیات‌های ریاضی، عملگرها نامیده می‌شوند که در جدول ۲,۱ نشان داده شده‌اند.

جدول ۲,۱ : عملگرهای اصلی پایتون

عملیات	نماد
جمع	+
تفریق	−
ضرب	*
تقسیم	/

از اسلش(/)[6] برای نشان دادن تقسیم استفاده می‌شود زیرا مشابه همان خط تقسیم است که از آن در هنگام نوشتن یک کسر استفاده می‌کنید. بعنوان مثال اگر ۱۰۰ دزددریایی[7] و ۲۰ بشکه[8] بزرگ داشته باشید و بخواهید حساب کنید که چند دزددریایی میتوانند در هر بشکه پنهان شوند، با وارد کردن 100/20 در برنامه پایتون میتوانید ۱۰۰ دزد دریایی را بر ۲۰ بشکه تقسیم کنید(20 ÷ 100).فقط بخاطر داشته باشید که اسلش(/) به سمت راست زاویه دارد.

---

[1] operator
[2] multiplication
[3] addition
[4] subtraction
[5] division
[6] Slash(/)
[7] Pirate
[8] barrel

ترتیب عملیات‌ها

در یک زبان برنامه‌نویسی از پرانتزها برای کنترل ترتیب عملیات‌ها استفاده می‌شود. یک عملیات[1] هر چیزی است که از یک عملگر استفاده می‌کند. ضـرب و تقسـیم نسـبت بـه جمـع و تفریـق در رتبـه بالاتری قرار دارند به این معنی که ابتدا اجرا می‌شوند. به بیان دیگر،اگر معادله‌ای را در پایتون وارد کنیـد، ضرب یا تقسیم قبل از جمع یا تفریق اجرا می‌شوند.

بعنوان مثال، در معادله زیر،ابتدا اعداد ۳۰ و۲۰ در هم ضرب شده و سپس عدد۵ به حاصل آنهـا اضافه می‌شود.

```
>>> 5 + 30 * 20
605
```

این معادله راه دیگری برای بیان «ضرب ۳۰ در ۲۰ و سپس جمع کردن۵ با نتیجه» است. حاصل ٦۰۵ خواهد بود. با اضافه کردن پرانتز دور دو عدد اول، میتوانیم ترتیب عملیات را تغییر دهیم:

```
>>> (5 + 30) * 20
700
```

نتیجه این معادله ۷۰۰(نه ٦۰۵) خواهد بود زیرا پرانتزهـا بـه پایتون میگویند که ابتدا عملیات‌هـای درون پرانتـز را انجـام دهـد و سپس به سراغ عملیات بیرون پرانتز برود. این مثال می‌گوید: «۵ را با ۳۰ جمع کند و سپس نتیجه را در۲۰ ضرب کن»

پرانتزها میتوانند تودرتو[2]باشند به این معنی که ممکـن اسـت پرانتزهایی درون پرانتزهای دیگری قرار داشته باشند، مثلاً

```
>>> ((5 + 30) * 20) / 10
70.0
```

در این مورد پایتون ابتدا داخلی‌ترین پرانتزها را ارزیابی کرده و سپس به سراغ پرانتزهای بعـدی می‌رود و درنهایت عملگر تقسیم را اجرا می‌کند. به بیان دیگر،این معادله می‌گوید: «۵ را با ۳۰ جمع کن، سپس حاصل را در ۲۰ ضرب کرده و نتیجه را بر ۱۰ تقسیم کن». این اتفاقی است که می‌افتد:

- از جمع ۵ با۳۰ داریم ۳۵
- از ضرب ۳۵ در ۲۰ داریم ۷۰۰
- از تقسیم ۷۰۰ بر ۱۰ داریم ۷۰ که جواب نهایی است

[1] operation
[2] nested

اگر از پرانتز استفاده نکرده باشید، نتیجه اندکی متفاوت خواهد بود:

```
>>> 5 + 30 * 20 / 10
65.0
```

در این مورد ، ابتدا ۳۰ در۲۰ ضرب شده(نتیجه ٦٠٠ خواهد بود) و سپس ٦٠٠ بر ١٠ تقسیم می‌شود (نتیجه ٦٠ خواهد بود). درنهایت،٥ به جواب اضافه شده تا ٦٥ بدست آید.

تذکر:

به خاطر داشته باشید که ضرب و تقسیم همواره قبل از جمع و تفریق انجام می‌شوند مگراینکه از پرانتزها برای کنترل ترتیب عملیات‌ها استفاده شده باشد.

متغیرها مانند برچسب‌ها هستند

در برنامه‌نویسی، واژه «متغیر» محل ذخیره اطلاعاتی مانند اعداد، متن، فهرست اعداد و متن و .... را توصیف می‌کند. از جنبه دیگر میتوان یک متغیر را بعنوان یک برچسب درنظر گرفت.

بعنوان مثال، برای ایجاد یک متغیر به نامfred، از علامت مساوی(=) استفاده کرده و سپس به پایتون می‌گوییم که متغیر بایستی برچسب چه اطلاعاتی باشد. در اینجا متغیرfred را ایجاد کرده و به پایتون می‌گوییم که عدد ١٠٠ را برچسب می‌زند(توجه داشته باشید به این معنی نیست که متغیر دیگری نمیتواند همین مقدار را داشته باشد):

```
>>> fred = 100
```

برای اینکه بفهمیم یک متغیر چه عددی را برچسب می‌زند، print و بدنبال آن نام متغیر را در داخل پرانتز، در برنامه وارد می‌کنیم:

```
>>> print(fred)
100
```

همچنین میتوانیم به پایتون بگوییم که متغیرfred را طوری تغییر دهد که چیز دیگری را برچسب بزند. بعنوان مثال، در اینجا نشان داده شده است که چگونه میتوانfred را به عدد ٢٠٠ تغییر داد:

```
>>> fred = 200
>>> print(fred)
200
```

در خط اول، میتوان گفت که fred عدد ٢٠٠ را برچسب می‌زند. در خط دوم، برای تأیید تغییر این سئوال را مطرح می‌کنیم که fred چه چیزی را برچسب می‌زند. پایتون نتیجه را در خط آخر چاپ می‌کند.

همچنین میتوان از یک برچسب(بیش از یک متغیر)برای یک آیتم استفاده نمود:

```
>>> fred = 200
>>> john = fred
>>> print(john)
200
```

در این مثال، به پایتون می‌گوییم که میخواهیم با علامت مساوی(=)میان john و fred ، نام(یا متغیر)john همان چیزی را برچسب بزند که که fred برچسب می‌زند.

البته، fred احتمالاً نام مناسبی برای یک متغیر نیست زیرا اطلاعاتی را درباره مورد مصرف متغیر در اختیار ما قرار نمی‌دهد. متغیر number_of_coins را به جای fred فراخوانی می‌کنیم:

```
>>> number_of_coins = 200
>>> print(number_of_coins)
200
```

درنتیجه روشن می‌شود که درباره ۲۰۰ سکه صحبت می‌کنیم. نام‌های متغیر میتوانند از حروف، اعداد و کاراکتر زیرخط(_) تشکیل شده باشند ولی نمیتوانند با عدد شروع شوند. میتوان از هر چیزی از یک حرف(مثلاً a) تا جملات طولانی برای نام‌های متغیر استفاده نمود. (یک متغیر نمیتواند حاوی جای خالی باشد بنابراین از زیرخط برای جدا کردن واژه‌ها استفاده می‌شود). برخی اوقات اگر میخواهید کاری را سریعاً انجام دهید، یک نام متغیر کوتاه بهترین گزینه خواهد بود. نامی که انتخاب می‌کنید به خواست شما درباره میزان معناداری نام متغیر وابسته است.

حال که میدانید چگونه باید یک متغیر را ساخت، به سراغ چگونگی استفاده از آنها می‌رویم.

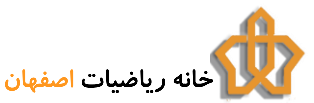
استفاده از متغیرها

معادله ما را برای حل این مسئله بخاطر بیاورید که در پایان سال چند سکه خواهید داشت اگر بطور خیالی بتوانید سکه‌های جدیدی را با کمک اختراع دیوانه‌وار پدربزرگتان در زیرزمین بسازید؟این معادله را داریم:

```
>>> 20 + 10 * 365
3670
>>> 3 * 52
156
>>> 3670 - 156
3514
```

میتوانیم آن را به یک خط کد تبدیل کنیم:

```
>>> 20 + 10 * 365 – 3 * 52
3514
```



حال با تبدیل اعداد به متغیرها چه اتفاقی خواهد افتاد؟ کد زیر را وارد کنید:

```
>>> found_coins = 20
>>> magic_coins = 10
>>> stolen_coins = 3
```

این ورودی‌ها میتوانند متغیرهای found_coins,magic_coins و stolen_coins را ایجاد کنند.

حال میتوانیم مجدداً معادله را وارد کنیم:

```
>>> found_coins + magic_coins * 365 - stolen_coins * 52
3514
```

میتوان دید که همگی جواب یکسانی دارند. آیا کسی اهمیت می‌دهد؟ بله، ولی این جا جادوی متغیرها وجود دارد. اگر یک مترسک کنار پنجره بگذارید، و کلاغ فقط دو سکه به جای سه سکه سرقت کند، مسئله چگونه خواهد شد؟ زمانی که از یک متغیر استفاده کنیم میتوانیم متغیر را به گونه‌ای تغییر دهیم که عدد جدید را حفظ کرده و هر کجا در هر معادله‌ای استفاده شود،تغییر کند. با کد زیر میتوانیم stolen_coins را به2 تغییر دهیم:

```
>>> stolen_coins = 2
```

سپس برای محاسبه مجدد پاسخ میتوانیم معادله را کپی و درج[1] کنیم:

۱- با موس روی متنی که میخواهید کپی کنید کلیک کرده و از ابتدا تا انتهای خط را انتخاب کنید.

```
*Python Shell*
File  Edit  Debug  Options  Windows  Help
Python 3.2.2 (default, Sep  4 2011, 09:51:08) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>> found_coins = 20
>>> magic_coins = 10
>>> stolen_coins = 3
>>> found_coins + magic_coins * 365 - stolen_coins * 52
3514
>>> stolen_coins = 2
Ln: 7 Col: 55
```

۲- برای کپی کردن متن برگزیده شده، کلیدctrl (یا اگر از سیستم‌عامل Mac استفاده می‌کنید، کلید ⌘)را نگهداشته و کلید C را فشار دهید. (از این به بعد بصورتCTRL-C نمایش داده خواهد شد).

۳- روی آخرین خط prompt (بعد از stolen_coins=2)کلیک کنید.

[1] paste

٤- برای درج متن برگزیده، همزمان کلیدCTRL را نگهداشته و کلیدV را فشار دهید(از این بـه بعد بصورتCTRL-V نمایش داده خواهد شد)

٥- برای دیدن نتیجه جدید،ENTER را فشار دهید

```
Python Shell
File Edit Debug Options Windows Help
Python 3.2.2 (default, Sep  9 2011, 09:51:08) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>> found_coins = 20
>>> magic_coins = 10
>>> stolen_coins = 3
>>> found_coins + magic_coins * 365 - stolen_coins * 52
3514
>>> stolen_coins = 2
>>> found_coins + magic_coins * 365 - stolen_coins * 52
3566
>>> |
Ln: 12 Col: 4
```

آیا آسان‌تر از تایپ مجدد کل معادله نیست؟ البته که هست.

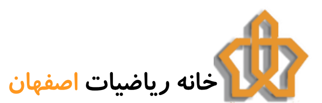
میتوانید متغیرهای دیگر را تغییر داده و معادله را کپی(CTRL-C) و درج(CTRL-V) کنید و اثـر تغییرات خود را مشاهده نمایید. بعنوان مثال اگر در لحظه مناسب به دوطرف اختراع پـدربزرگتان ضـربه بزنید، و هر بار ۳ سکه دیگر از آن خارج شود،درمی‌یابید که تا پایان سال ٤٦٦١ سکه خواهید داشت.

```
>>> magic_coins = 13
>>> found_coins + magic_coins * 365 - stolen_coins * 52
4661
```

البته استفاده از متغیرها برای چنین معادله ساده‌ای، خیلی مفید نیست. اکنون فقط به خاطر داشتـه باشید که متغیرها راهی برای برچسب زدن چیزها هستند بطوریکه بتوانید بعداً از آنها استفاده کنید.

آنچه آموختید

در این فصل یاد گرفتید که چگونه معادلات ساده را با استفاده از عملگرهای پایتون حل کـرده و چگونه از پرانتزها برای کنترل ترتیب عملیات‌ها ساتفاده کنید(ترتیب ارزیابی بخش‌های معادلات توسـط پایتون). سپس متغیرهایی را برای برچسب زدن مقادیر ایجاد کـرده و از ایـن متغیرهـا در محاسـبات‌مـان استفاده کردیم.



فصل۳
رشته‌ها[1]، لیست‌ها، چندتایی‌ها[2] و نقشه‌ها[3]

در فصل۲ برخی محاسبات پایه را با پایتون انجام داده و چیزهایی درباره متغیرها آموختیم. در این فصل، با آیتم‌های دیگری در برنامه‌های پایتون سروکار خواهیم داشت: رشته‌ها، لیست‌ها، چندتایی‌ها و نقشه‌ها. از رشته‌ها برای نمایش پیام‌ها در برنامه‌ها(مثلاً پیام‌های "Get Ready"و"Game Over" در یک بازی)استفاده می‌کنید. همچنین درخواهید یافت که چگونه از لیست‌ها، چندتایی‌ها و نقشه‌ها برای ذخیره کردن مجموعه چیزها استفاده می‌شود.

رشته‌ها

در اصطلاحات برنامه‌نویسی معمولاً متن را یک رشته(string) می‌نامیم. اگر یک رشته را بصورت یک مجموعه از حروف درنظر بگیرید، این اصطلاح منطقی خواهد بود. تمامی حروف، اعداد و نمادها در این کتاب میتوانند یک رشته باشند. درنتیجه نام و آدرس شما هم یک رشته است. درواقع، در اولین برنامه پایتون ی که در فصل۱ ساختیم، از رشته"Hello World" استفاده شده است.

ایجاد رشته‌ها

[1] string
[2] tuple
[3] map

در پایتون، با قراردادن علامت نقل قول("")اطراف متن، یک رشته را ایجاد می‌کنیم. بعنوان مثال، از متغیر بلااستفاده fred در فصل۲ برای برچسب‌گذاری یک رشته استفاده می‌کنیم:

```
fred = "Why do gorillas have big nostrils? Big fingers!!"
```

سپس، برای اینکه ببینیم چه چیزی داخل fred وجود دارد میتوانیم print(fred) را وارد کنیم:

```
>>> print(fred)
Why do gorillas have big nostrils? Big fingers!!
```

همچنین میتوانید از تک نقل‌قول (' ') نیز برای ایجاد یک رشته استفاده کنید

```
>>> fred = 'What is pink and fluffy? Pink fluff!!'
>>> print(fred)
What is pink and fluffy? Pink fluff!!
```

هرچند اگر سعی کنید بیش از یک خط متن را تنها با استفاده از نقل قول تکی(') یا نقل قول دوتایی(")برای رشته وارد کنید یا با یک نوع نقل قول شروع کرده و با نقل قول دیگری خاتمه دهید، پیام خطا را از جانب برنامه پایتون مشاهده خواهید کرد. بعنوان مثال اگر خط زیر را وارد کنید:

```
>>> fred = "How do dinosaurs pay their bills?
```

نتیجه زیر را مشاهده می‌کنید:

```
SyntaxError: EOL while scanning string literal
```

این یک پیام خطا درباره نحو کد[1] است زیرا از قوانین ختم یک رشته با نقل قول تکی یا دوتایی تبعیت نکرده‌اید.

نحو(Syntax) به معنای چیدمان و ترتیب واژگان در یک جمله یا(در این مورد)چیدمان و ترکیب واژگان و نمادها در یک برنامه است. SyntaxError به این معنی است که کاری را به ترتیبی انجام داده‌اید که برای پایتون غیرمنتظره بوده است. یا پایتون انتظار چیزی را دارد که فراموش کرده‌اید. EOL به معنای پایان خط(end of line) است بنابراین بقیه پیام خطا به شما می‌گوید پایتون به سراغ پایان خط رفته و نقل قول دوتایی را برای بستن رشته مشاهده نمی‌کند.

بری استفاده از بیش از یک خط متن در رشته(تحت عنوان رشته چندخطی)از سه نقل قول تکی(''') استفاده کرده و سپس بین خطوط ENTER بزنید:

```
>>> fred = '''How do dinosaurs pay their bills?
With tyrannosaurus checks!'''
```

حال محتوای fred را چاپ کرده تا ببینیم درست کار می‌کند یا نه:

```
>>> print(fred)
How do dinosaurs pay their bills?
```

[1] Syntax ترکیب

With tyrannosaurus checks!

مدیریت مسائل با رشته‌ها

این مثال رشته را درنظر می‌گیریم که باعث می‌شود پایتون یک پیام خطا را نشان دهد.

```
>>> silly_string = 'He said, "Aren't can't shouldn't wouldn't."'
SyntaxError: invalid syntax
```

در خط اول، سعی می‌کنیم یک رشته را ایجاد کنیم(بصورت متغیر silly_string تعریف شده است)که در تک نقل قول(') قرار گرفته است و حاوی ترکیبی از تک نقل قول‌ها در واژگان can't, shouldn't و wouldn't و نقل قول‌های دوتایی است.

بخاطر داشته باشید که پایتون باندازه انسان هوشمند نیست بنابراین تمامی آن چیزی که می‌بینند یک رشته حاوی He said, "Aren, و بدنبال آن یک دسته از کاراکترهایی است که انتظار ندارد. زمانیکه پایتون با علامت نقل قول مواجه می‌شود(تک نقل قول یا دوتایی)، انتظار دارد که رشته بعد از اولین علامت آغاز شده و بعد از علامت نقل قول(تکی یا دوتایی) بعدی(از همان نوع) در آن خط خاتمه یابد. در این مورد، شروع رشته با علامت نقل قول تکی قبل از He و انتهای رشته همانطوریکه پایتون انتظار دارد، یک نقل قول تکی بعد از n درAren می‌باشد. IDLE روی جایی تأکید دارد که اشتباه رخ داده است:

```
Python Shell
File  Edit  Debug  Options  Windows  Help
Python 3.1.2 (r312:79149) on win32
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>> silly_string = 'He said, "Aren't can't shouldn't wouldn't."'
SyntaxError: invalid syntax
>>> |
Ln: 6 Col: 4
```

آخرین خط IDLE به ما نشان می‌دهد که چه نوع خطایی رخ داده است– در این مورد، یک خطای نحوی.

استفاده از نقل قول دوتایی به جای تکی هنوزهم خطا تولید می‌کند.

```
>>> silly_string = "He said, "Aren't can't shouldn't wouldn't.""
SyntaxError: invalid syntax
```

در اینجا پایتون با یک رشته مواجه است که در میان نقل قول دوتایی قرار گرفته است و حاوی حروف He said, (و یک فضای خالی) است. هر چیزی بعد از این رشته(ازAren't به بعد)خطا ایجاد می‌کند.

```
Python Shell
File Edit Debug Options Windows Help
Python 3.1.2 (r312:79149) on win32
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>> silly_string = 'He said, "Aren't can't shouldn't wouldn't."'
SyntaxError: invalid syntax
>>> silly_string = "He said, "Aren't can't shouldn't wouldn't.""
SyntaxError: invalid syntax
>>> |
Ln: 8 Col: 4
```

زیرا از دیدگاه پایتون، این چیز اضافی نباید اینجا باشد. پایتون بدنبال نقل قول منطبق(هم نـوع) بعدی است و نمیداند میخواهید با چیزهایی که بعد از نقل قول در آن خط آمده‌اند، چه کار کند.

راه‌حل ایـن مسـئله یـک رشـته چنـدخطی اسـت کـه قـبلاً چیزهایی درباره آن آموختیم، از سه علامـت نقـل قـول(' ' ') اسـتفاده کرده و میتوانیم نقل قول تکی و دوتایی را با هم و بدون تولیـد خطـا، در رشته ترکیب کنیم. درواقع، اگر از نقل قول سه‌تایی اسـتفاده کنـیم، میتوانیم هر ترکیبی از نقل قول‌های تکی و دوتـایی را در رشـته قـرار دهیم(تا زمانی که سعی نکنیم سه علامت نقل قول تکـی را در آنجـا قرار دهیم). در زیر نمونه‌ای از نسخه بدون خطای رشته را مشاهده می‌کنید:

```
silly_string = '''He said, "Aren't can't shouldn't wouldn't."'''
```

ولی صبر کنید، چیزهای بیشتری وجود دارد. اگر واقعاً میخواهید در پایتون، یک رشته را در نقل قول‌های تکی یا دوتایی به جای سه‌تایی قرار دهید، میتوانید قبل از هر علامت نقل قـول در رشـته، یـک اسلش معکوس(\)قرار دهید. به این کار فرار[1]می‌گویند. این راهی است بـرای اینکـه بـه پـایتون بگـوییم «بله،میدانم نقل قول‌هایی در رشته‌ام دارم و میخواهم تا رسیدن به نقل قول پایانی، آنها را نادیده بگیری».

خواندن رشته‌های escaping دشوارتر است بنابراین بهتر است از رشته‌هـای چنـدخطی اسـتفاده کنیم. هنوزهم ممکن است با تکه کدهایی روبرو شوید که از escaping استفاده مـی‌کننـد بنـابراین بهتـر است بفهمید چرا بک‌اسلش‌ها(\) آنجا هستند.

در اینجا چند مثال از چگونگی کار کردن escaping ارائه شده است:

```
❶ >>> single_quote_str = 'He said, "Aren₩'t can₩'t shouldn₩'t wouldn₩'t."'
```

[1] escaping

```
❷ >>> double_quote_str = "He said, ₩"Aren't can't shouldn't wouldn't.₩""
>>> print(single_quote_str)
He said, "Aren't can't shouldn't wouldn't."
>>> print(double_quote_str)
He said, "Aren't can't shouldn't wouldn't."
```

ابتدا در❶، یک رشته با نقل قول‌های تکی را با استفاده از بک‌اسلش در جلوی نقل قول‌های تکی در آن رشته می‌سازیم. در ❷ یک رشته با نقل قول‌های دوتایی ساخته و از بک‌اسلش در جلوی این نقل قول‌ها در آن رشته استفاده می‌کنیم. در خط‌های بعد، متغیرهایی که ایجاد کرده‌ایم را چاپ می‌کنیم. توجه داشته باشید بعد از چاپ، کاراکتر بک‌اسلش دررشته‌ها ظاهر نمی‌شود.

تعبیه مقادیر در رشته‌ها

اگر میخواهید با استفاده از محتویات یک متغیر پیامی را نمایش دهید، با استفاده از s% که مشابه علامت مقداری است که میخواهید بعداً اضافه کنید، میتوانید مقادیر را در یک رشته تعبیه کنید (برنامه-نویس به جای «اضافه کردن مقادیر» از «تعبیه مقادیر» استفاده می‌کند). بعنوان مثال، برای اینکه پایتون تعداد امتیازات گرفته شده در یک بازی را محاسبه و ذخیره کرده و سپس آن را به عبارتی مانند «من... امتیاز گرفتم» اضافه کند، در عبارت از s% به جای مقدار استفاده کرده و سپس آن مقدار را به پایتون اعلام می‌کند مثلاً:

```
>>> myscore = 1000
>>> message = 'I scored %s points'
>>> print(message % myscore)
I scored 1000 points
```

در اینجا متغیر myscore را با مقدار ۱۰۰۰ و متغیر message را با یک رشته حاوی واژگان «من s% امتیاز گرفتم» ایجاد می‌کنیم، که در آن s% یک متغیر جایگزین[1] برای تعداد امتیازات می‌باشد. در خط بعد، با نماد %، print(message) را فراخوانی می‌کنیم تا به پایتون بگوییم که s% را با جایگزین مقدار ذخیره شده در متغیر myscore نماید. در نتیجه این پیام چاپ خواهد شد I scored 1000 points. نیازی به استفاده از یک متغیر برای مقدار نداریم. همین مثال را تنها با استفاده از print(message % 1000) میتوانیم انجام دهیم.

همچنین میتوانیم از مقادیر مختلفی برای متغیر جایگزین s% با استفاده از متغیرهای مختلف استفاده کنیم مثلاً در این مثال:

[1] placeholder

```
>>> joke_text = '%s: a device for finding furniture in the dark'
>>> bodypart1 = 'Knee'
>>> bodypart2 = 'Shin'
>>> print(joke_text % bodypart1)
Knee: a device for finding furniture in the dark
>>> print(joke_text % bodypart2)
Shin: a device for finding furniture in the dark
```

دراینجا سه متغیر را ایجاد می‌کنیم. متغیرا ول،joke_text شامل رشته‌ای با علامت%s است. دو متغیر دیگر عبارتند ازbodypart1 وbodypart2 . میتوانیم متغیرjoke_txt را چاپ کرده و یکبار دیگر از عملگر % برای جایگزین کردن آن با محتویات متغیرهای bodypart1 وbodypart2 برای تولید پیام‌های مختلف استفاده کنیم.

همچنین میتوان از بیش از یک متغیر جایگزین در یک رشته استفاده نمود. مثلاً:

```
>>> nums = 'What did the number %s say to the number %s? Nice belt!!'
>>> print(nums % (0, 8))
What did the number 0 say to the number 8? Nice belt!!
```

زمانیکه از بیش از یک متغیر جایگزین استفاده می‌کنید،اطمینان حاصل کنید که مقادیر چایگزین را درون پرانتز قرار داده‌اید(همانند مثال).مقادیر به همان ترتیبی قرار می‌گیرند که در یک رشته بکارگرفته خواهند شد.

ضرب رشته‌ها

جواب ۱۰ ضربدر۵ چقدر می‌شود؟ البته که پاسخ ۵۰ است. ولی ۱۰ ضربدرa چقدر می‌شود؟ در اینجا پایتون پاسخ می‌دهد.

```
>>> print(10 * 'a')
aaaaaaaaaa
```

مثلاً برنامه‌نویسان پایتون از این رویکرد برای به خط کردن رشته‌ها با یک تعداد فضای خالی معین در هنگام نمایش پیام‌ها در برنامه(shell) استفاده می‌کنند. درباره چاپ یک حرف در برنامه(shell)نظرتان چیست (File ▸ New Window را انتخاب کرده و کد زیر را وارد کنید):

```
spaces = ' ' * 25
print('%s 12 Butts Wynd' % spaces)
print('%s Twinklebottom Heath' % spaces)
print('%s West Snoring' % spaces)
```

```
print()
print()
print('Dear Sir')
print()
print('I wish to report that tiles are missing from the')
print('outside toilet roof.')
print('I think it was bad wind the other night that blew them away.')
print()
print('Regards')
print('Malcolm Dithering')
```

به مجرد این که کد را در پنجره برنامه تایپ کردید، **File ‣ Save As** را انتخاب کرده و نام فایل خود را *myletter.py* قرار دهید.

نکته: از حالا به بعد زمانی که بالای یک تکه کد، Save As: somefilename.py را می‌بینید میدانید که باید ***File ‣ New Window*** را انتخاب کرده، کد را در پنجره‌ای که ظاهر می‌شود وارد کرده و سپس آن را همانند مثال، ذخیره کنید.

در اولین خط این مثال، متغیر space را با ضریب یک کاراکتر جای خالی (space) در ۲۵ ایجاد کردیم. سپس از این متغیر در سه خط بعد استفاده کرده تا متن را در برنامه راست‌چین کنیم. حال میتوانید نتیجه این دستورات print را در زیر ببینیم:

علاوه بر استفاده از ضرب برای همترازی[۱] میتوانید از آن برای پر کردن صفحه با پیام‌های خسته‌کننده استفاده کنید. این مثال را خودتان انجام دهید:

```
>>> print(1000 * 'snirt')
```

لیست‌ها بسیار قدرتمندتر از رشته‌ها هستند

«پاهای عنکبوت، پنجه‌های قورباغه، چشم سمندر،[1] بال‌های خفاش، روغن حلزون[2] و پوست خشک شده مار» فقط یک فهرست خرید ساده نیستند (مگراینکه یک جادوگر باشید) ولی میتوانید از آن بعنوان اولین مثال برای درک تفاوت میان لیست‌ها و رشته‌ها استفاده کنید.

میتوانیم این فهرست آیتم‌ها با استفاده از رشته‌ای مانند زیر در متغیر wizard_list ذخیره کنیم:

```
>>> wizard_list = 'spider legs, toe of frog, eye of newt, bat wing,
slug butter, snake dandruff'
>>> print(wizard_list)
spider legs, toe of frog, eye of newt, bat wing, slug butter, snake
dandruff
```

ولی میتوانیم یک list هم درست کنیم که تاحدودی یک نوع جادویی از شیء پایتون و قابل دستکاری است. در اینجا میبینیم که وقتی این آیتم‌ها بصورت یک لیست نوشته می‌شوند به چه صورتی خواهند بود:

```
>>> wizard_list = ['spider legs', 'toe of frog', 'eye of newt',
'bat wing', 'slug butter', 'snake dandruff']
>>> print(wizard_list)
['spider legs', 'toe of frog', 'eye of newt', 'bat wing', 'slug
butter', 'snake dandruff']
```

در مقایسه با ایجاد یک رشته، ایجاد یک لیست به تایپ بیشتری نیاز دارد ولی یک لیست سودمندتر از یک رشته است زیرا میتوان آن را دستکاری کرد. بعنوان مثال میتوانید سومین آیتم wizard_list (چشم سمندر)را با وارد کردن موقعیت آن درلیست(تحت عنوان موقعیت اندیس) در داخل کروشه([ ]) چاپ کنید:

```
>>> print(wizard_list[2])
eye of newt
```

خوب! آیا این سومین آیتم در لیست نیست؟ بله، ولی لیست در موقعیت اندیس 0 آغاز می‌شود بنابراین اولین آیتم یک لیست، 0، دومین آیتم، 1 و آیتم سوم، 2 می‌باشد. این موضوع چندان از نظر ما منطقی به نظر نمی‌رسد ولی از نظر رایانه‌ها منطقی است.

---

[1] newt

[2] slug

همچنین میتوانیم بسیار ساده‌تر از کاری در یک رشته انجام می‌دهیم، یک آیتم را در یک لیست تغییر دهیم. شاید به جای چشم سمندر به زبان مار نیاز داشته باشیم. دراینجا نشان خواهیم داد که چگونه این کار را با لیست خودمان انجام می‌دهیم.

```
>>> wizard_list[2] = 'snail tongue'
>>> print(wizard_list)
['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug
butter', 'snake dandruff']
```

در اینجا آیتم در موقعیت اندیس2 ، قبل از چشم سمندر در زبان مار قرار می‌گیرد.

گزینه دیگر، نمایش زیرمجموعه‌ای از آیتم‌ها در لیست است. ما اینکار را با استفاده از یک علامت دونقطه (کلون)(:) داخل کروشه انجام می‌دهیم. بعنوان مثال، عبارت زیر را وارد کرده تا آیتم‌های سوم و چهارم لیست را مشاهده کنید(مجوعه فوق‌العاده‌ای از عناصر تشکیل دهنده برای یک ساندویچ دلچسب):

```
>>> print(wizard_list[2:5])
['snail tongue', 'bat wing', 'slug butter']
```

با نوشتن[2:5] میخواهیم بگوییم «آیتم‌های موقعیت اندیس2 تا موقعیت اندیس5 (بدون خود موقعیت5) را نمایش بده» یا به عبارت دیگر آیتم‌های2,3,4.

میتوان از لیست‌ها برای ذخیره هر گونه آیتمی مثلاً اعداد استفاده نمود:

```
>>> some_numbers = [1, 2, 5, 10, 20]
```

همچنین میتوانند حاوی رشته‌ها باشند:

```
>>> some_strings = ['Which', 'Witch', 'Is', 'Which']
```

یا ترکیبی از اعداد و رشته‌ها:

```
>>> numbers_and_strings = ['Why', 'was', 6, 'afraid', 'of', 7,
'because', 7, 8, 9]
>>> print(numbers_and_strings)
['Why', 'was', 6, 'afraid', 'of', 7, 'because', 7, 8, 9]
```

حتی میتوانند شامل لیست‌های دیگری باشند:

```
>>> numbers = [1, 2, 3, 4]
>>> strings = ['I', 'kicked', 'my', 'toe', 'and', 'it', 'is', 'sore']
>>> mylist = [numbers, strings]
>>> print(mylist)
[[1, 2, 3, 4], ['I', 'kicked', 'my', 'toe', 'and', 'it', 'is', 'sore']]
```

این مثال لیست در لیست، سه متغیر را ایجاد می‌کند: numbers با چهار عدد، strings بـا هشـت رشته و mylist با استفاده از numbers و strings. لیست سوم (mylist) تنها دو عنصر دارد زیـرا فهرسـت نام‌های متغیر است نه محتویات متغیرها.

افزودن آیتم‌ها به یک لیست

برای افزودن آیتم‌ها به یک لیست از تابع append استفاده می‌کنیم. یک تابع[1] یک تکه از کد اسـت که به پایتون می‌گوید کاری را انجام دهد. در این مورد، append یک آیتم را به انتهای یک لیست اضـافه می‌کند.

بعنوان مثال، برای اضافه کردن یک آروغ خرس به لیست خرید جادگر،این کار را انجام می‌دهیم:

```
>>> wizard_list.append('bear burp')
>>> print(wizard_list)
['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug
butter', 'snake dandruff', 'bear burp']
```

به همین ترتیب میتوانید به اضافه کردن آیتم‌های جادویی بیشتر به لیست جادوگر ادامه دهید:

```
>>> wizard_list.append('mandrake')
>>> wizard_list.append('hemlock')
>>> wizard_list.append('swamp gas')
```

درنتیجه لیست جادوگر به این شکل خواهد بود

```
>>> print(wizard_list)
['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug
butter', 'snake dandruff', 'bear burp', 'mandrake', 'hemlock', 'swamp
gas']
```

حالا جادوگر آماده انجام یک جادوی واقعی است.

حذف آیتم‌ها از یک لیست

برای حذف آیتم‌ها از یک لیست، از فرمان del (کوتاه شده delete) اسـتفاده کنیـد. بعنـوان مثـال، برای حذف ششمین آیتم در لیست جادوگر(پوست خشکیده مار) به شیوه زیر عمل کنید:

```
>>> del wizard_list[5]
>>> print(wizard_list)
['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug
```

---

[1] function

```
butter', 'bear burp', 'mandrake', 'hemlock', 'swamp gas']
```

نکته: یادآور می‌شویم که موقعیت‌ها از 0 شروع می‌شوند، بنابراین wizard_list[5] درواقع به ششمین آیتم لیست اشاره دارد.

این نشان می‌دهیم که چگونه میتوانیم آیتم‌های افزوده شده به لیست را حذف کنیم(مهرگیاه[1]،شوکران[2] و گاز مرداب[3]):

```
>>> del wizard_list[8]
>>> del wizard_list[7]
>>> del wizard_list[6]
>>> print(wizard_list)
['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug
butter', 'bear burp']
```

حسابگان لیست

با اضافه کردن لیست‌ها به هم میتوانیم آنها را به هم پیوند بزنیم دقیقاً همانند جمع کردن اعداد با استفاده از علامت جمع(+).بعنوان مثال فرض کنید دو لیست داریم: List1 حاوی اعداد ۱تا۴ و List2 حاوی چند کلمه. با استفاده از print و علامت+ میتوانیم آنها را با هم جمع کنیم:

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = ['I', 'tripped', 'over', 'and', 'hit', 'the', 'floor']
>>> print(list1 + list2)
[1, 2, 3, 4, 'I', 'tripped', 'over', 'and', 'hit', 'the', 'floor']
```

همچنین میتوانیم دو لیست را با هم جمع کرده و نتیجه را برابر با متغیر دیگری قرار دهیم.

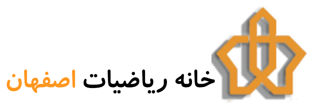
```
>>> list1 = [1, 2, 3, 4]
>>> list2 = ['I', 'ate', 'chocolate', 'and', 'I', 'want', 'more']
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4, 'I', 'ate', 'chocolate', 'and', 'I', 'want', 'more']
```

و میتوانیم یک لیست را در یک عدد ضرب کنیم. بعنوان مثال، برای ضرب list1 در ۵ می‌نویسیم list1*5.

```
>>> list1 = [1, 2]
```

---

[1] mandrake
[2] hemlock
[3] swamp




```
>>> print(list1 * 5)
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

درو اقع با این کار به پایتون می‌گوییم list1 را پنج بار تکرار کن درنتیجه داریم 1, 2, 1, 2, 1, 2, 1, 2, 1, 2.

از سوی دیگر تقسیم(/) و تفریق(–) فقط خطا تولید می‌کنند:

```
>>> list1 / 20
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
list1 / 20
TypeError: unsupported operand type(s) for /: 'list' and 'int'
>>> list1 - 20
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
list1 - 20
TypeError: unsupported operand type(s) for -: 'list' and 'int'
```

ولی چرا؟ خوب، پیوند دادن لیست‌ها با علامت+ و تکرار لیست‌ها بـا علامـت* عملیـات‌هـای ساده و بی واسطه‌ای هستند. این مفاهیم در دنیای واقعی نیز منطقی می‌باشند. بعنوان مثال اگـر دو لیسـت خرید به شما بدهم و بگویم «این دو لیست را با هم جمع کن»، شما تمامی آیتم‌ها را به ترتیب از ابتدا تا به انتها روی برگه کاغذ دیگری خواهید نوشت. همین کار در این مورد نیز صادق است اگر بگویم «ایـن لیست‌ها را در ۳ ضرب کن»، شما میتوانید تصور کنید لیستی از تمامی آیتم‌های لیسـت را سـه مرتبـه در برگه کاغذ دیگری می‌نویسید.

ولی چگونه میتوانیم یک لیست را تقسیم کنیم؟بعنوان مثال، یک لیست از شش عدد(۱ تا٦)را بـه دو تقسیم کنید. در این جا سه راه متفاوت وجود دارد:

```
[1, 2, 3]      [4, 5, 6]
[1]            [2, 3, 4, 5, 6]
 [5, 6]        [1, 2, 3, 4]
```

آیا لیست را از وسط بعد از آیتم اول دو قسمت کرده یا فقـط یـک محـل را بطور تصادفی انتخاب و تقسیم را در آنجا انجام می‌دهید؟ پاسخ به ایـن سـئوال سـاده نیست و زمانی که از پایتون میخواهید یک لیست را تقسیم کند، نمیداند چگونـه بایـد اینکار را انجام دهد. به همین دلیل با پیام خطا پاسخ می‌دهد.

همین قضیه در خصوص اضافه کردن هر چیزی به جز یک لیست به یک لیست نیز صادق است. شما نمیتوانید اینکار را انجام دهید. بعنوان مثال در اینجا میبینیم که اگر سعی کنیم عدد50 را به list1 اضافه کنیم چه اتفاقی خواهد افتاد:

```
>>> list1 + 50
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
list1 + 50
TypeError: can only concatenate list (not "int") to list
```

به چه دلیل با پیام خطا مواجه می‌شویم؟ اضافه کردن50 به یک لیست به چه معنی است؟ آیا به معنای اضافه کردن 50 به هر آیتم است؟ ولی اگر آیتم‌ها عدد نباشند چه می‌شود؟ آیا به معنای اضافه کردن عدد50 به انتها یا ابتدای لیست است؟

در برنامه‌نویسی رایانه‌ای، فرمان‌ها بایستی به همان شیوه‌ای که آنها را وارد می‌کنیم عمل نمایند. رایانه زبان بسته همه چیز را سیاه و سفید می‌بینید. اگر از او بخواهید کارهای پیچیده‌ای را انجام دهد، یک مشت پیام خطا به شما تحویل خواهد داد.

چندتایی‌ها

یک چندتایی، لیستی است که از پرانتزها استفاده می‌کند، بعنوان مثال:

```
>>> fibs = (0, 1, 1, 2, 3)
>>> print(fibs[3])
2
```

در اینجا متغیر fibs را بصورت اعداد0,1,1,2,3 تعریف می‌کنیم. سپس همانند کاری که در یک لیست انجام می‌دادیم، با استفاده از print(fibs[3]) ، آیتمی را که در موقعیت اندیس3 در چندتایی قرار دارد چاپ می‌کنیم.

عمده تفاوت میان یک چندتایی و یک لیست در این است که وقتی یک چندتایی را ایجاد کردید، بلافاصله قابل تغییر نیست. بعنوان مثال اگر سعی کنید عدد4 را جایگزین اولین مقدار در چندتایی fibs نمایید (همانند جایگزینی مقادیر در wizard_list) آنگاه با پیام خطای زیر روبرو خواهید شد:

```
>>> fibs[0] = 4
Traceback (most recent call last):
File "<pyshell>", line 1, in <module>
fibs[0] = 4
```

```
TypeError: 'tuple' object does not support item assignment
```

چرا از چندتایی به جای یک لیست استفاده می‌کنید؟ اساساً به این دلیل که برخی اوقات استفاده از چیزی که میدانید هیچ وقت تغییر نمی‌کند، سودمند خواهد بود. اگر یک چندتایی حاوی دو عنصر را ایجاد کنید، همواره این دو عنصر درون آن وجود خواهند داشت.

نقشه‌های پایتون نمیتوانند به شما در پیدا کردن راه‌تان کمک کنند

در پایتون یک نقشه (که تحت عنوان dict مخفف دیکشنری نیز شناخته می‌شود) مجموعه‌ای از چیزها می‌باشد مثلاً لیست‌ها و چندتایی‌ها. تفاوت بین نقشه‌ها و لیست‌ها یا چندتایی‌ها در این است که هر آیتم در یک نقشه دارای یک کلید[1] و یک مقدار متناظر با آن می‌باشد.

بعنوان مثال، لیستی از مردم و ورزش‌های موردعلاقه آنها را دراختیار داریم. میتوانیم این اطلاعات را در یک لیست پایتون قرار دهیم بطوریکه نام فرد و بدنبال آن ورزش مورد علاقه‌ای قرار گیرد:

```
>>> favorite_sports = ['Ralph Williams, Football',
                       'Michael Tippett, Basketball',
                       'Edward Elgar, Baseball',
                       'Rebecca Clarke, Netball',
                       'Ethel Smyth, Badminton',
                       'Frank Bridge, Rugby']
```

اگر از شما بپرسم ورزش موردعلاقه Rebecca Clarke چیست، میتوانید در این لیست جستجو کرده و پاسخ را بیابید: نت بال.ولی اگر در این لیست ۱۰۰ نفر وجود داشته باشند (یا بیشتر) وضعیت چگونه خواهد بود؟

حالا اگر همان اطلاعات را در یک نقشه ذخیره کنیم، جاییکه نام فرد بعنوان کلید و ورزش مورد علاقه‌اش بعنوان مقدار می‌باشد،کد پایتون بصورت زیر خواهد بود:

```
>>> favorite_sports = {'Ralph Williams' : 'Football',
                       'Michael Tippett' : 'Basketball',
                       'Edward Elgar' : 'Basebal',
                       'Rebecca Clarke' : 'Netball',
                       'Ethel Smyth' : 'Badminton',
                       'Frank Bridge' : 'Rugby'}
```

[1] key

از علامت دونقطه(:) برای جدا کردن هر کلید از مقدارش استفاده کرده و هر کلید و مقدارش در یک نقل قول تکی قرار می‌گیرند. همچنین توجه داشته باشید که آیتم‌های یـک نقشــه درون آکـولاد({}) قرار می‌گیرند نه پرانتز یا براکت.

درنتیجه یک نقشه (هر کلید به یک مقدار خاص نگاشت[1]می‌شود) بصورت جـدول ۳,۱ خواهیـد داشت.

جدول ۳,۱: کلیدها به مقادیر در یک نقشه از ورزش‌های موردعلاقه اشاره دارند

کلید	مقدار
Ralph Williams	فوتبال
Michael Tippett	بسکتبال
Edward Elgar	بیس‌بال
Rebecca Clarke	نت بال
Ethel Smyth	بدمینتون
Frank Bridge	راگبی

حالا که فهمیدیم چه ورزشی موردعلاقه Rebecca Clarke است، با استفاده از نام او بعنوان کلید میتوانیم به نقشه favorite_sports دسترسی داشته باشیم:

```
>>> print(favorite_sports['Rebecca Clarke'])
Netball
```

و پاسخ نت‌بال خواهد بود.

برای حذف یک مقدار در یک نقشه، از کلید آن استفاده می‌کنیم. بعنوان مثال در اینجا نشان می-دهیم که چگونه Ethel Smyth حذف می‌شود.

```
>>> del favorite_sports['Ethel Smyth']
>>> print(favorite_sports)
{'Rebecca Clarke': 'Netball', 'Michael Tippett': 'Basketball', 'Ralph
Williams': 'Football', 'Edward Elgar': 'Baseball', 'Frank Bridge':
'Rugby'}
```

برای جایگزینی یک مقدار در یک نقشه از کلید آن استفاده می‌کنیم.

```
>>> favorite_sports['Ralph Williams'] = 'Ice Hockey'
>>> print(favorite_sports)
{'Rebecca Clarke': 'Netball', 'Michael Tippett': 'Basketball', 'Ralph
```

[1] map

Williams': 'Ice Hockey', 'Edward Elgar': 'Baseball', 'Frank Bridge':
'Rugby'}

با استفاده از کلیــدRalph Williams، Ice Hokey را جـایگزین Football favorite sport of مــی-کنیم.

همانطوریکه مشاهده می‌کنید، کار با نقشه همانند کار کردن با لیست‌ها و چندتایی‌ها می‌باشد بـا این تفاوت که نمیتوانید نقشه‌ها را با عملیات جمع(+)به هم پیوند بدهید. اگر سعی کنید این کار را انجام دهید، با پیام خطا مواجه خواهید شد.

```
>>> favorite_sports = {'Rebecca Clarke': 'Netball',
'Michael Tippett': 'Basketball',
'Ralph Williams': 'Ice Hockey',
'Edward Elgar': 'Baseball',
'Frank Bridge': 'Rugby'}
>>> favorite_colors = {'Malcolm Warner' : 'Pink polka dots',
'James Baxter' : 'Orange stripes',
'Sue Lee' : 'Purple paisley'}
>>> favorite_sports + favorite_colors
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

پیوند دادن نقشه‌ها برای پایتون منطقی نیست بنابراین از این کار منصرف می‌شود (تسلیم مــی-شود).

آنچه آموختید

در این فصل، آموختید که چگونه پایتون از رشته‌ها برای ذخیره متن استفاده کـرده و چگونـه از لیست‌ها و چندتایی‌ها برای مدیریت چندین آیتم استفاده می‌کند. همچنین دیدید که آیتم‌های یک لیســت را میتوان تغییر داد و میتوانید یک لیست را به دیگری پیوند دهید ولی مقادیر یــک چنـدتایی قابـل تغییــر نیستند. همچنین آموختید که چگونه میتوانید از نقشه برای ذخیره مقادیر همراه با کلیدهای شناسایی آنهـا استفاده کنید.

(چیستان)معماهای برنامه‌نویسی

در زیر چند آزمایش ارائه شده است که میتوانید خودتان آنها را انجام دهید.پاسخ‌هـا را میتوانیـد در .http://python-for-kids.com/ مشاهده کنید.

۱– علاقمندی‌ها

لیستی از سرگرمی‌های موردعلاقه‌تان تهیه کرده و نام متغیر راgames بگذارید. حالا لیستی از غذاهای موردعلاقه‌تان تهیه کرده و نام متغیر راfood قرار دهید. این دو لیست را به هم پیوند داده و نتیجه راFavorites بنامید. درنهایت متغیرfavorites را چاپ کنید.

۲- شمارش جنگنده‌ها

اگر سه ساختمان داشته باشیم که در هر طبقه ۲۵ نینجا مخفی شده و ۲ تونل که ٤۰ سامورایی در هر تونل پنهان شده است، چند نینجا و سامورایی در این نبرد شرکت دارند؟ (میتوانید این کار را با یک رابطه در برنامه پایتون انجام دهید)

۳- پیام‌های تبریک!

دو متغیر ایجاد کنید: یک متغیر به نام اول و متغیر دیگر به فامیل شما اشاره کند. حال یک رشته ایجاد کرده و از متغیرهای جایگزین برای چاپ نام و یک پیام با استفاده از این دو متغیر استفاده کنید مثلاً "Hi there, Brando Ickett!".

فصل ٤
نقاشی با لاک پشت ها

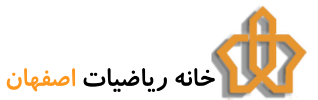
یک turtle در پایتون گونه‌ای از یک لاک‌پشت در دنیای واقعی است. ما لاک‌پشت را بعنوان گونه‌ای از خزندگان می‌شناسیم که بسیار کند حرکت کرده و خانه‌شان در پشت آنها قرار دارد. در دنیای پایتون،یک لاک‌پشت یک فلش کوچک و سیاه است که به آرامی در صفحه حرکت می‌کند. درواقع، فرض کنید لاک‌پشت پایتون در حین حرکت در صفحه اثری را از خود به جا می‌گذارد که باعث می‌شود بیشتر شبیه به یک حلزون یا نرم‌تن صدفدار باشد تا یک لاک‌پشت.

لاک‌پشت راه مناسبی برای یادگیری برخی از مبانی گرافیک‌های رایانه‌ای است بنابراین در این فصل از لاک‌پشت پایتون برای ترسیم برخی اشکال و خطوط ساده استفاده می‌کنیم.

استفاده از ماژول turtle در پایتون

یک ماژول[1] در پایتون راهی است برای تأمین یک کد سودمند برای اینکه برنامه دیگری بتواند از آن استفاده کند(علاوه بر چیزهای دیگر توابعی در ماژول وجود دارند که میتوانیم از آنها استفاده کنیم). در فصل ۷ بیشتر درباره ماژول‌ها خواهیم آموخت. پایتون یک ماژول بسیار خاص به نام turtle دارد که میتوانیم از آن استفاده کرده و یاد بگیریم که چگونه رایانه‌ها تصاویر گرافیکی را روی صفحه می‌کشند. ماژول turtle راهی است برای برنامه‌نویسی گرافیک‌های برداری که اساساً ترسیم با خطوط

[1] module



ساده، نقاط و منحنی‌ها می‌باشد.

اجازه دهید به این موضوع بپردازیم که turtle چگونه کار می‌کند. ابتدا،با کلیک روی آیکون رومیزی (اگر از Ubuntu استفاده می‌کنید،**Applications ‣ Programming ‣ IDLE**) برنامه پایتون را اجرا می‌کنیم. سپس به پایتون می‌گوییم که با وارد کردن ماژول turtle بصورت زیر از turtle استفاده کند:

```
>>> import turtle
```

وارد کردن یک ماژول به پایتون می‌گوید که شما می‌خواهید از آن استفاده کنید.

نکته:

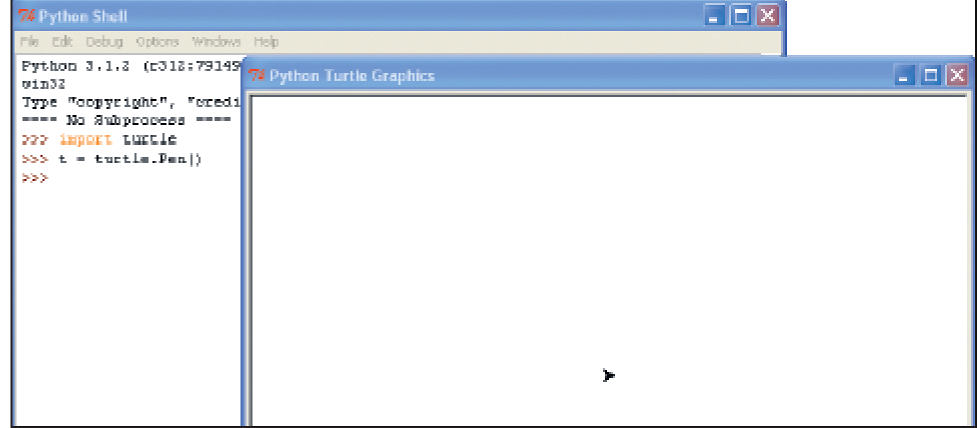
اگر از Ubuntu استفاده می‌کنید و در این نقطه با پیام خطا روبرو شدید، بایستی tkinter را نصب کنید. برای اینکار، *Ubuntu Software Center* را بازکرده و در کادر جستجو، python-tk را وارد کنید. بایستی *"Tkinter – Writing Tk Applicaitions with Python"* در پنجره ظاهر شود. برای نصب این مجموعه، روی ***Install*** کلیک کنید.

ایجاد یک بوم[1]

حالکه ماژول turtle را وارد کردیم، بایستی یک بوم ایجاد کینم- یک فضای خالی برای نقاشی روی آن مانند بوم هنرمندان. برای اینکار، تابع pen را از ماژول turtle فراخوانی می‌کنیم که بطور خودکار یک بوم ایجاد خواهد کرد. دستور زیر را در برنامه پایتون وارد کنید:

```
>>> t = turtle.Pen()
```

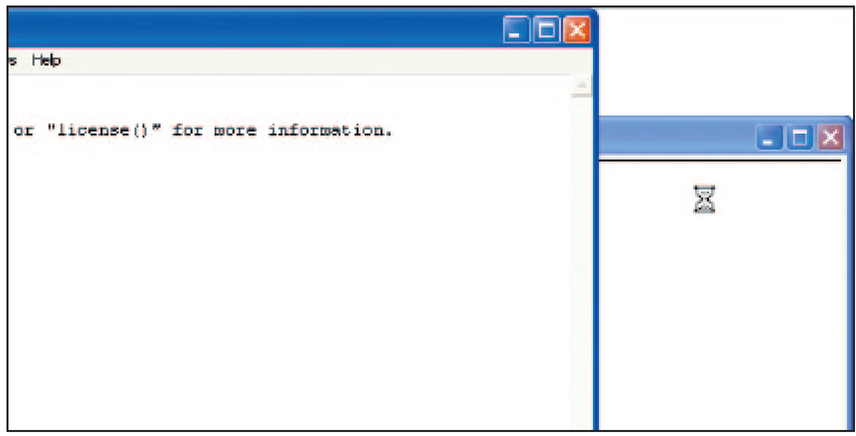
بایستی یک کادر خالی(بوم) ببینید که در آن یک فلش در مرکز قرار گرفته است:



فلش در مرکز صفحه، لاک‌پشت(turtle)است و شما کار را درست انجام داده‌اید- زیاد شبیه لاک‌پشت نیست.

[1] canvas

اگر پنجره Turtle پشت پنجره برنامه پایتون ظاهر شود، درخواهید یافت که بدرستی کار نمی‌کند. زمانی که موشواره[1] خود را روی پنجره Turtle جابجا می‌کنید، نشانگر[2] به یک ساعت شنی تبدیل می‌شود:



این اتفاق ممکن است به دلایل مختلفی رخ دهد: شما روی برنامه را از طریق آیکون رومیزی اجرا نکرده‌اید (اگر از ویندوز یا Mac استفاده می‌کنید)، روی IDLE (Python GUI) در منوی شروع ویندوز کلیک کردید؛ یا IDLE بدرستی نصب نشده است. از برنامه خارج شوید و برنامه را از روی آیکون رومیزی مجدداً اجرا کنید. اگر اینکار موفقیت‌آمیز نبود، سعی کنید از کنسول پایتون به جای برنامه(shell) استفاده کنید:

۱- در ویندوز، **Start ‣ All Programs** را انتخاب کرده و سپس در گروه **Python 3.2** روی **Python (command line)** کلیک کنید.

۲- در Mac OS X روی آیکون Spotlight در گوشه سمت راست- بالای صفحه کلیک کرده و در کادر ورودی، *Terminal* را وارد کنید. بعد از باز شدن ترمینال، *python* را وارد کنید.

۳- در Ubuntu ، ترمینال را از منوی **Applications** باز کرده و *python* را وارد کنید.

حرکت دادن لاک‌پشت

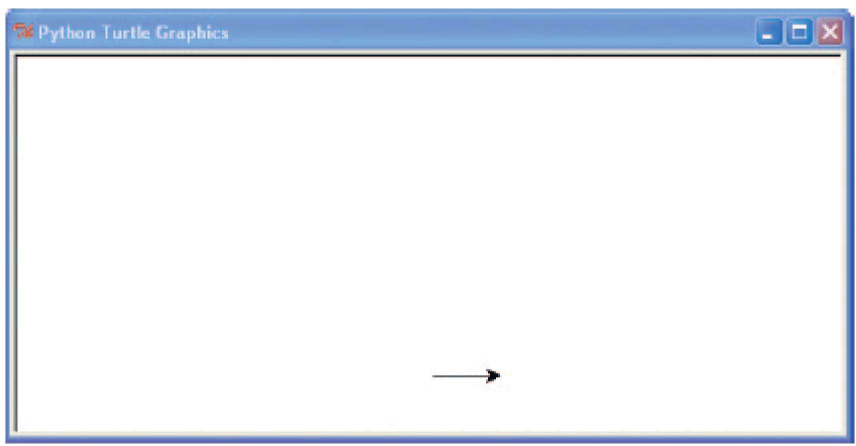
با استفاده از توابع دردسترس در متغیر t که همانند استفاده از تابع pen در ماژول turtle ایجاد کردیم، دستورات را برای turtle می‌فرستیم. بعنوان مثال، دستور forward به لاک‌پشت می‌گوید که به جلو

[1] Mouse موس
[2] cursor

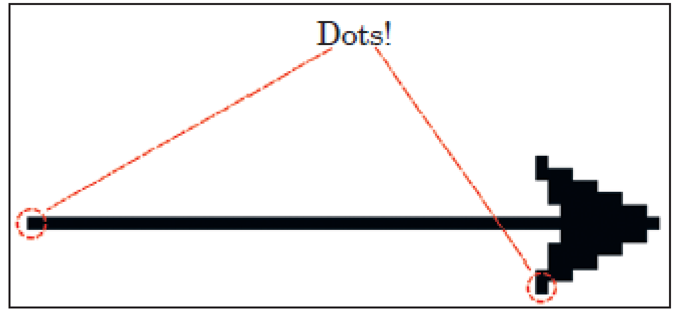
حرکت کند. برای اینکه لاک‌پشت ۵۰ پیکسل جلو برود، فرمان زیر را وارد کنید:

```
>>> t.forward(50)
```

بعد از وارد کردن این دستور باید چیزی شبیه به این روی صفحه ظاهر شود:



لاک‌پشت ۵۰ پیکسل به جلو رفته است. یک پیکسل[1] یک نقطه روی صفحه است–کوچکترین عنصری که قابل نمایش است. هرچیزی که روی صفحه نمایش رایانه خود می‌بینید از پیکسل‌ها ساخته شده است که نقاط مربعی‌شکل بسیار کوچک هستند. اگر بتوانید بوم و خط ترسیم شده توسط لاک‌پشت را بزرگنمایی کنید خواهید دید که فلش نشان‌دهنده مسیر لاک‌پشت، مجموعه‌ای از پیکسل‌ها می‌باشد. این گرافیک‌های ساده رایانه‌ای است.



حال، با فرمان زیر به لاک‌پشت می‌گوییم ۹۰ درجه به چپ بچرخد:

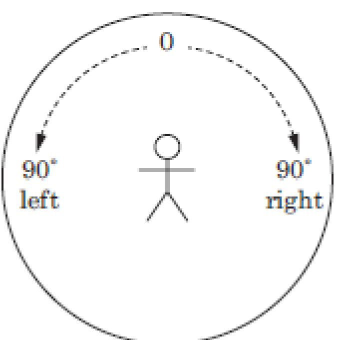
```
>>> t.left(90)
```

اگر هنوز درجه را بلد نیستید، در اینجا میتوانید تجسمی از آن داشته باشید. تصور کنید در مرکز یک دایره ایستاده‌اید.

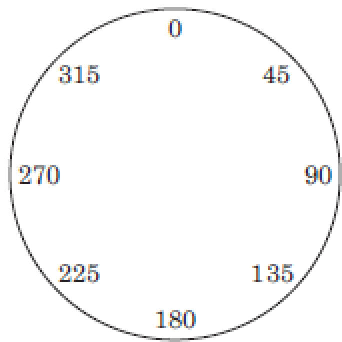
- جهت روبروی شما صفر درجه است
- اگر دست چپ خود را از پهلو بلند کنید، این ۹۰درجه به چپ است
- اگر دسته راست خود را از پهلو بلند کنید،۹۰ درجه به راست است

[1] pixel

در اینجا میتوانید ۹۰درجه به چپ یا راست را ببینید:

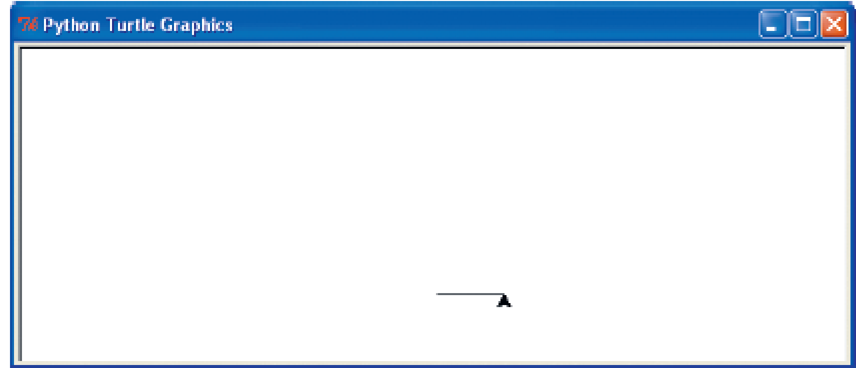


اگر به سمت راست از همان جاییکه دست راست شما به آن اشاره می‌کند، دور دایره بچرخید، ۱۸۰درجه درست پشت سر شما خواهد بود و ۲۷۰ درجه جهتی است که دست چپ شما به آن اشاره می‌کند، و۳۶۰ درجه پشت نقطه شروع چرخش است؛ درجه‌ها از صفر تا ۳٦۰ می‌باشند. درجه‌ها در یک دایره کامل زمانیکه به راست می چرخید، بصورت ٤٥ درجه، ٤٥ درجه (تقسیمات٤٥ درجه‌ای) نمایش داده شده‌اند.



زمانیکه لاک‌پشت پایتون به چپ می‌چرخد، دور دایره چرخیده تا رو به جهت جدید قرار گیرد(مانند زمانیکه بدن خود را تابانده تا رو به جایی قرار گیرید که دست شما به چپ ۹۰ درجه اشاره می‌کند).

فرمان(90)t.left فلش را رو به بالا نشانه می‌رود (زیرا با اشاره به راست شروع شده است):



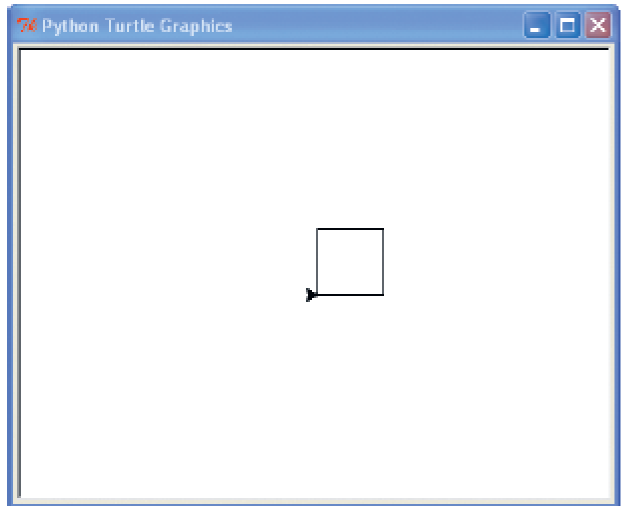
نکته:

زمانیکه *t.left(90)* را فراخوانی میکنید همانند فراخوانی *t.right(270)* است. این مورد برای *calling* *t.right(90)* نیز صادقی است که همانند *t.left(270)* می‌باشد. فقط این دایره را تجسم کرده و زاویه‌ها را دنبال کنید.

حالا یک مربع کشیده‌ایم. کد زیر را به خطوطی که وارد کرده‌اید اضافه کنید:

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
```

لاک‌پشت شما بایستی یک مربع را ترسیم کرده باشد و در همان جهتی قرار بگیرد که حرکت را شروع کرده.



برای پاک کردن بوم، reset را وارد کنید. درنتیجه بوم پاک شده و لاک‌پشت به موقعیت شروع بازمی‌گردد.

```
>>> t.reset()
```

همچنین میتوانید از clear استفاده کنید که صفحه را پاک کرده و لاک‌پشت را همان جایی که هست رها می‌کند.

```
>>> t.clear()
```

همچنین میتوانیم لاک‌پشت را به راست چرانده یا به عقب برگردانیم. میتوانیم از up برای بلند کردن قلم از روی صفحه (به بیان دیگر به لاک‌پشت بگوییم که ترسیم را متوقف کند) و از down برای شروع ترسیم استفاده کنیم. این توابع به شیوه‌ای که از بقیه توابع استفاده می‌شود، نوشته می‌شوند.

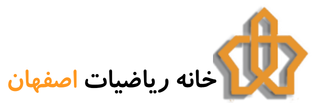
با استفاده از برخی از این توابع ترسیم‌های دیگری را انجام می‌دهیم. این بار، لاک‌پشت دو تا خط می‌کشد. کد زیر را وارد کنید:

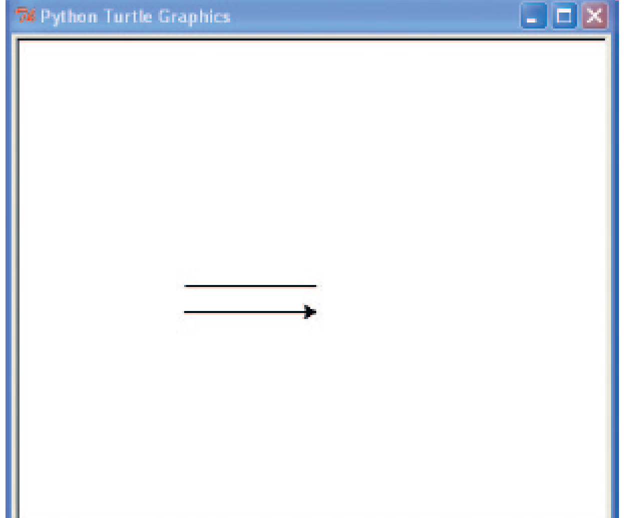
```
>>> t.reset()
>>> t.backward(100)
>>> t.up()
>>> t.right(90)
>>> t.forward(20)
>>> t.left(90)
>>> t.down()
>>> t.forward(100)
```

ابتدا، با دستور t.reset() بوم را از بازنشانی[1] کرده و لاک‌پشت را به موقعیت شروع برمی‌گردانیم. سپس با دستور t.backward(100) لاک‌پشت را ۲۰۰ پیکسل به عقب برمی‌گردانیم و سپس از t.up() برای برداشتن قلم و توقف ترسیم استفاده می‌کنیم.

در ادامه، با فرمان t.right(90) لاک‌پشت را ۹۰ درجه به پایین به سمت انتهای صفحه می‌چرخانیم و با فرمان t.forward(20) ۲۰ پیکسل جلو می‌رویم. هیچ چیزی ترسیم نمی‌شود زیرا در خط سوم از فرمان up استفاده شده است. با دستور t.left(90) لاک‌پشت ۹۰ درجه به چپ چرخانده تا رو به سمت راست قرار گیرد و سپس با فرمان down به لاک‌پشت می‌گوییم قلم را مجدداً پایین آورده و ترسیم را از سر بگیرد. درنهایت با فرمان t.forward(100) یک خط مستقیم به موازات اولین خط می‌کشیم. دو خط موازی که کشیده‌ایم به این صورت خواهند بود:

[1] reset





آنچه آموختید

در این فصل، یاد گرفتید که چگونه باید از ماژول turtle در پایتون استفاده کنید. با استفاده از فرمان‌های left، right، forward و backward چند خط ساده ترسیم کردیم. فهمیدید که چگونه میتوانید با فرمان up لاک‌پشت را از ترسیم کردن باز دارید و مجدداً با فرمان down ترسیم کردن را از سر بگیرید. همچنین دریافتید که لاک‌پشت چند درجه می‌چرخد.

(چیستان)معماهای برنامه‌نویسی

برخی از شکل‌های زیر را با لاک‌پشت ترسیم کنید. پاسخ‌ها در *http://python-for-kids.com/.* ارائه شده‌اند.

۱- یک مستطیل[1]

با استفاده از تابع pen ماژول turtle یک بوم جدید ایجاد کرده و یک مستطیل ترسیم کنید.

۲- یک مثلث[2]

بوم دیگری ایجاد کرده و این بار یک مثلث ترسیم کنید. مجدداً به سراغ دیاگرام دایره مدرج بروید («حرکت دادن لاک‌پشت» در صفحه٤٩) تا بخاطر بیاورید که با استفاده از درجه‌ها میتوانید لاک‌پشت را به چه جهاتی بچرخانید.

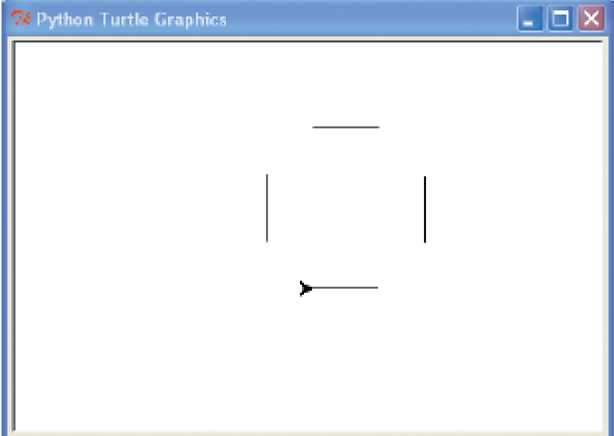
۳- یک کادر بدون گوشه

---

[1] Rectangle

[2] triangle

برنامه‌ای بنویسید تا چهار خط زیر را ترسیم کند (سایز اهمیتی ندارد، فقط شکل مهم است)



فصل ۵

طرح سئوالات با IF و ELSE

دربرنامه‌نویسی، سئوالات بلی یا خیر را مطرح کرده و تصمیم داریم براساس این پاسخ کاری را انجام دهیم. بعنوان مثال، ممکن است بپرسیم «آیا سن شما بیشتر از ۲۰ سال است؟» و اگر پاسخ مثبت بود، با «شما خیلی مسن هستی» پاسخ دهیم.

این دسته سئوالات، شرایط[1] نامیده شده و این شرایط و پاسخ‌ها را در دستورات if با هم ترکیب می‌کنیم. شرایط میتوانند پیچیده‌تر از یک سئوال باشند و دستورات if را میتوان با چندین سئوال و جواب براساس پاسخ به هر سئوال، ترکیب نمود.

در این فصل یاد می‌گیریم که چگونه از دستورات if برای ساختن برنامه‌های استفاده کنید.

دستورات IF

یک دستور if[2] در پایتون به فرم زیر است:

```
>>> age = 13
>>> if age > 20:
print('You are too old!')
```

[1] condition

[2] Statement جمله

یـک دسـتور if از کلیـدواژه if، بـدنبال آن یـک شـرط و یـک دونقطه(:) تشکیل شده است، مثلاً if age > 20:. خطوط بعد از دو نقطه بایستی در یک بلوک قرار گیرنـد و اگر پاسخ به سئوال مثبت بود(یاtrue، در برنامه‌نویسی پایتون) آنگاه فرمان‌های واقع در بلوک اجرا می-شوند. حال به بررسی این موضوع می‌پردازیم که چگونه بلوک‌ها و شرط‌ها را بنویسیم.

یک بلوک[1]یک دسته از دستورات برنامه‌نویسی است.

یک بلوک از کد یک مجموعه گروهبندی شده از دستورات برنامه‌نویسی است. بعنوان مثال وقتی if age > 20:، true (برقرار) است میخواهید کاری بیش از صرفاً چاپ «چند سالته؟» انجـام دهیـد. شـاید میخواهید دستورات دیگری را نیز چاپ کنید مثلاً:

```
>>> age = 25
>>> if age > 20:
print('You are too old!')
print('Why are you here?')
print('Why aren₩'t you mowing a lawn or sorting papers?')
```

این بلوک کد از سه دستورprint تشکیل شده است که فقط درصورتی اجرا می‌شوند کـه شـرط age > 20، true (برقرار) باشد. در مقایسه با دستور if فوق، در ابتدای هـر خـط در بلـوک، چهـار فاصـله[2] قرار دارد. در اینجا مجدداً به کد نگاه می‌کنیم با این تفاوت که این بار فضاهای خالی مشخص هستند:

```
>>> age = 25
>>> if age > 20:
▯ ▯ ▯ ▯ print('You are too old!')
▯ ▯ ▯ ▯ print('Why are you here?')
▯ ▯ ▯ ▯ print('Why aren₩'t you mowing a lawn or sorting papers?')
```

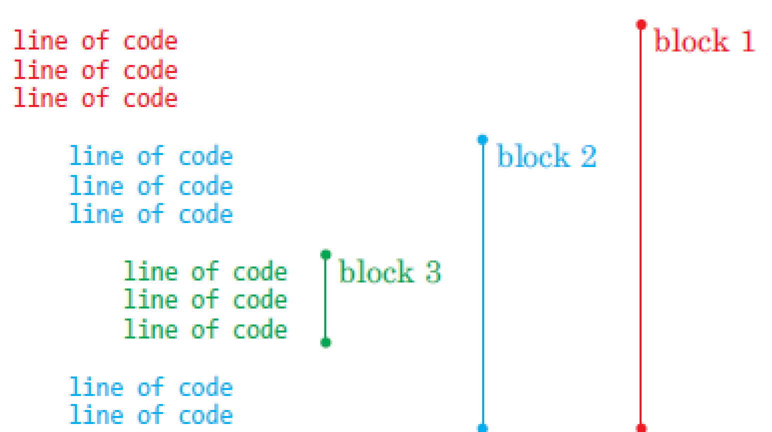
در پایتون،whitespace[1] مثلاً یک tab (زمانی اضافه می‌شود که کلیدTAB را فشار دهید) یا یـک فاصله (زمانی اضافه می‌شوند که spacebar را فشار دهید) معنادار است. کدی که در همان موقعیت قرار

---

[1] block

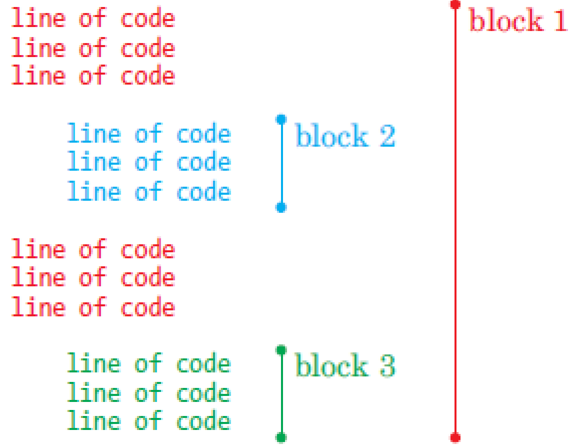
[2] Space فضای خالی

دارد (به همان تعداد فاصله از حاشیه چپ تورفتگی[2] دارد) در یک بلوک گروهبندی شده است و هرگاه خط جدیدی را با فضاهای خالی بیشتری درمقایسه با خط قبل شروع کنید، بلوک جدیدی را آغاز کرده‌اید که بخشی از بلوک قبلی است:



دستورات را با هم در یک گروه در بلوک‌ها قرار می‌دهیم زیرا به هم ربط دارند. دستورات بایستی با هم اجرا شوند.

وقتی تورفتگی‌ها را تغییر می‌دهید، معمولاً در حال خلق بلوک‌های جدید هستید. مثال زیر سه بلوک مجزا را نشان می‌دهد که فقط با تغییر تورفتگی ایجاد شده‌اند.



در اینجا حتی با فرض این که بلوک‌های۲و۳ یک تورفتگی دارند، متفاوت درنظر گرفته می‌شوند زیرا یک بلوک با تورفتگی کمتر بین آنها وجود دارد (فضاهای خالی کمتر).

[1] کاراکتری که در سند چاپ شده ظاهر نمی‌شود
[2] indent

به این منظور، زمانی که یک بلوک با چهار فاصله در هر خط و شش فاصله در خط بعدی را اجرا می‌کنید، یک پیام خطای تورفتگی را نمایش می‌دهد زیرا پایتون انتظار دارد از تعداد فاصله‌های یکسانی برای تمامی خطوط در یک بلوک استفاده کنید. بنابراین اگر یک بلوک با چهار فاصله را شروع می‌کنید، بایستی بطور ثابت از چهار فاصله برای آن بلوک استفاده نمایید. بعنوان مثال:

```
>>> if age > 20:
□ □ □ □ print('You are too old!')
□ □ □ □ □ □ print('Why are you here?')
```

در اینجا فضاهای خالی نمایش داده شده‌اند بطوری که بتوانید تفاوت‌ها را مشاهده کنید. توجه داشته باشید که در خط سوم، به جای چهار فاصله، شش فاصله وجود دارد.

زمانیکه می‌خواهیم این کد را اجرا کنیم،IDLE یک خط را پررنگ می‌کند جاییکه مسئله‌ای را در بلوک قرمز رنگ مشاهده کرده و یک پیام توضیحی Syntaxerror را نمایش می‌دهد:

```
>>> age = 25
>>> if age > 20:
        print('You are too old!')
          print('Why are you here?')
SyntaxError: unexpected indent
```

پایتون انتظار ندارد دو فاصله اضافی در ابتدای دومین خط print مشاهده کند.

نکته:

از فاصله‌گذاری سازگار استفاده کرده تا قرائت کد شما ساده‌تر گردد. اگر با نوشتن برنامه‌ای شروع کنید که در آنجا چهار فاصله در ابتدای بلوک گذاشته شده است، در ابتدای دیگر بلوک‌های برنامه نیز از چهار فاصله استفاده کنید. همچنین مطمئن شوید که تورفتگی‌های هر خط در یک بلوک با تعداد فاصله‌های برابری انجام می‌شود.

شرط‌ها به ما کمک می‌کنند تا چیزها را مقایسه کنیم.

یک شرط[۱] یک دستور برنامه‌نویسی است که چیزها را مقایسه کرده و True (بلی) یا False (خیر) بودن معیار تعیین شده در مقایسه را به ما نشان می‌دهد. بعنوان مثال age > یک شرط است که

[۱] condition

اینگونه نیز میتوان آن را بیان کرد «آیا مقدار age بزرگتر از 10 است؟». این هم یک شرط است: hair_color == 'mauve' یا به عبارت دیگر «آیا mauve مقدار متغیر hair_color است؟»

از نمادها در پایتون (که عملگرها نامیده می‌شوند) برای ایجاد شـرطهـا اسـتفاده مـی‌کـنیم مـثلاً 'مساوی با'، 'بزرگتر از'، و 'کوچکتر از'. جدول ۵،۱ برخی از نمادهای شرطها را نشان می‌دهد.

جدول ۵،۱) نمادهای شرطها

نماد	تعریف
==	مساوی با
!=	مساوی نیست با
>	بزرگتر از
<	کوچکتر از
>=	بزرگتر یا مساوی
<=	کوچکتر یا مساوی

بعنـوان مثـال، اگـر شـما ۱۰سـال داریـد، شـرط your_age == 10، True را برمـی‌گردانـد؛ درغیراینصورت، False را برخواهد گرداند. اگـر ۱۲سـال داریـد، شـرط your_age > 10، True را برمـی-گرداند.

هشدار:

مطمئن شوید هنگام تعریف شرط 'مساوی با'، از علامت مساوی دوتایی(==) استفاده می‌کنید.

حال اجازه دهید تا چند مثال دیگر را عنوان کنیم. دراینجا سن خود را برابر با ۱۰ گرفته و سپس یک دستور شرطی می‌نویسیم که اگر سن شما بیشتر از ۱۰ بود، «جوک من مناسب سـن شـما نیسـت! (You are too old for my jokes!)» را چاپ کند.

```
>>> age = 10
>>> if age > 10:
    print('You are too old for my jokes!')
```

وقتی این عبارت را در IDLE تایپ کرده و ENTER را فشار می‌دهید، چه اتفاقی خواهد افتاد؟

هیچ اتفاقی رخ نمی‌دهد.

چون مقـدار بازگشـتی از age بزرگتـر از ۱۰ نیسـت درنتیجـه پـایتون بلوک print را اجرا نمی‌کند. هرچند اگر متغیر age را برابر با ۲۰ درنظر بگیـریم،

پیام چاپ خواهد شد.

حال مثال قبل را تغییر داده تا از شرط 'بزرگتر یا مساوی'(>=) استفاده کنیم:

```
>>> age = 10
>>> if age >= 10:
      print('You are too old for my jokes!')
```

باید پیام"!You are too old for my jokes" روی صفحه نمایش چاپ شده باشد.

در مرحله بعد از شرط'مساوی با' (==) استفاده کنید:

```
>>> age = 10
>>> if age == 10:
        print('What₩'s brown and sticky? A stick!!')
```

باید پیام (چه چیزی قهوه‌ای و سفت است؟عصا!!)"!!What's brown and sticky? A stick"
روی صفحه چاپ شود.

دستورات IF-THEN-ELSE

علاوه بر این که وقتی شرطی برقرار است (True) از دستورات if برای انجام کاری استفاده می-کنیم، میتوانیم وقتی شرطی برقرار نیست (False) نیز از دستوراتif استفاده کنیم. بعنوان مثال، اگر سن شما ۱۲ سال باشد یک پیام و اگر ۱۲ نباشد (False) پیام دیگری را روی صفحه نمایش چاپ کنیم.

در اینجا رمز کار استفاده از یک دستورif-then-else است که به معنای «اگر چیزیtrue بود این کار را انجام بده؛ یا آن کار را انجام بده».

حال یک دستورif-then-else ایجاد می‌کنیم. عبارت زیر را در برنامه وارد کنید:

```
>>> print("Want to hear a dirty joke?")
Want to hear a dirty joke?
>>> age = 12
>>> if age == 12:
        print("A pig fell in the mud!")
else:
        print("Shh. It's a secret.")
A pig fell in the mud!
```

چون متغیرage را ۱۲ درنظر گرفته‌ایم و شرط این است که سن برابر با۱۲ باشد، بایستی اولینprint را روی صفحه مشاهده کنید. حال مقدارage را به عددی بزرگتر از ۱۲ تغییر دهید:

```
>>> print("Want to hear a dirty joke?")
Want to hear a dirty joke?
```

```
>>> age = 8
>>> if age == 12:
        print("A pig fell in the mud!")
else:
        print("Shh. It's a secret.")
Shh. It's a secret.
```

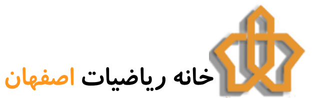
این بار، بایستی پیام دومین print را مشاهده کنید.

دستورات IF و ELIF

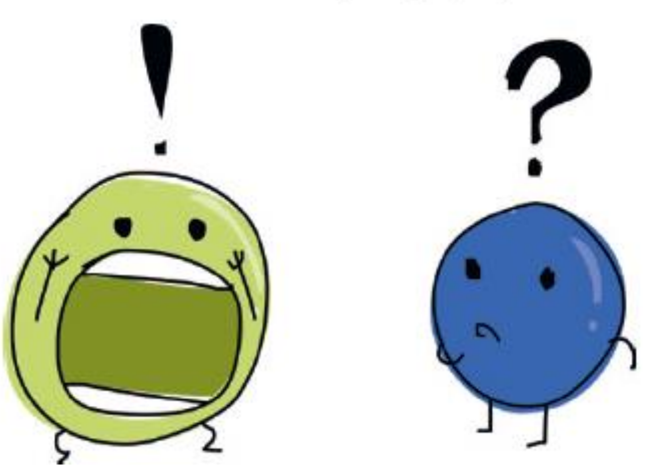
با elif (مخففelse-if) میتوانیم دستورif را بسط دهیم. بعنوان مثال میتوانیم ۱۰،۱۱ یا ۱۲ سال (یا بیشتر) بودن سن یک نفر را بررسی کرده و از برنامه بخواهیم براساس پاسخ کار متفاوتی را انجام دهد. وجه تمایز این دستورات با دستورات if-then-else در این است که بیش از یکelif میتواند در یک دستور وجود داشته باشد:

```
>>> age = 12
❶ >>> if age == 10:
❷       print("What do you call an unhappy cranberry?")
        print("A blueberry!")
❸ elif age == 11:
        print("What did the green grape say to the blue grape?")
        print("Breathe! Breathe!")
❹ elif age == 12:
❺       print("What did 0 say to 8?")
        print("Hi guys!")
elif age == 13:
        print("Why wasn't 10 afraid of 7?")
        print("Because rather than eating 9, 7 8 pi.")
else:
        print("Huh?")
What did 0 say to 8? Hi guys!
```

در این مثال، دستورif در خط دوم، مساوی بودن متغیرage با۱۰ را در ❶ وارسی می‌کند. دستورprint در❷ درصورتی اجرا خواهد شد که age برابر ۱۰ باشد. هرچند، چونage را برابر با۱۲ درنظر گرفته‌ایم، رایانه به دستورif بعدی در ❸ پریده و مساوی بودنage با ۱۱ را وارسی می‌کند. تساوی برقرار نیست بنابراین رایانه به سراغ دستورif بعدی در ❹ رفته تا برابر بودنage با۱۲ را وارسی کند. تساوی برقرار است بنابراین این بار فرمانprint در ❺اجرا می‌شود.



زمانی که این کد را درIDLE وارد می‌کنید، بطور خودکار تورفتگی پیدا می‌کند بنابراین مطمئن باشید که بعد از تایپ هر دستورprint، کلیدBACKSPACE یاDELETE را می‌زنید بطوریکه دستوراتif، elif وelse از منتهی علیه سمت چپ آغاز شوند. این همان جایگاهی است که اگر Prompt(>>>) وجود نداشت، دستورif در آن قرار می‌گیرد.



ترکیب شرطها

با استفاده از کلیدواژه‌هایand وor که کد ساده‌تر و کوتاهتری را تولید می‌کنند، شرطها را با هم ترکیب کنید. در اینجا مثالی از کاربردor وجود دارد:

```
>>> if age == 10 or age == 11 or age == 12 or age == 13:
        print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
        print('Huh?')
```

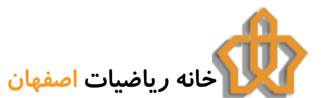
در این کد، اگر هریک از شرطهای خط اول true باشند(صدق کنند)(به بیان دیگر اگرage ۱۰،۱۱،۱۲ یا۱۳ باشد) بلوک کد در خط بعدی که باPrint آغاز می‌شود اجرا خواهد شد.

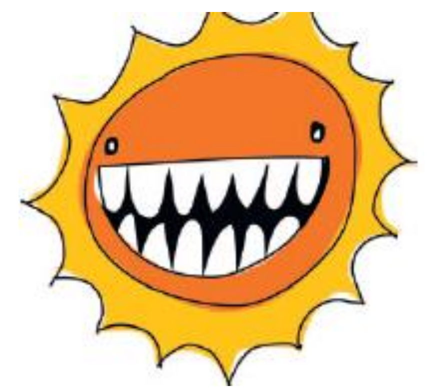
اگر شرطهای خط اول true نباشند(else) پایتون به سراغ بلوک آخرین خط می‌رود و Huh? را روی صفحه نمایش می‌دهد.

برای این که این مثال بازهم مختصرتر شود، میتوانیم از کلیدواژهand در کنار عملکرد بزرگتر یا مساوی(>=) و کوچکتر یا مساوی(<=) استفاده کنیم:

```
>>> if age >= 10 and age <= 13:
        print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
        print('Huh?')
```

[1] keyword





در اینجا اگر age بزرگتر یا مساوی ۱۰ و کوچتر یا مساوی ۱۳ باشد، طبق تعریف خط اول با if age >= 10 and age <= 13: ، بلوک کدی که در خط بعدی باprint شروع شده است اجرا خواهد شد. بعنوان مثال اگر مقدارage،۱۲ باشد آنگاه What is 13 + 49 + 84 + 155 + 97? A headache! روی صفحه چاپ می‌شود زیرا ۱۲ بزرگتر از ۱۰ و کوچکتر از ۱۳ می‌باشد.

متغیرهای بدون مقدار–None

همانطوریکه میتوانیم اعداد، رشته‌ها و لیست‌ها را به یک متغیر نسبت[1]دهیم، میتوانیم هیچ چیز یا یک مقدار تهی را نیز به یک متغیر نسبت دهیم. در پایتون یک مقدار تهی باNone نشان داده شده و فاقد مقدار است. لازم به ذکر است که مقدارNone با مقدار0 متفاوت است زیرا به جای عددی با مقدار0، فاقد مقدار است. زمانیکه به یک متغیر مقدار تهیNone را نسبت می‌دهیم، تنها مقداری که آن متغیرخواهد داشت، هیچ چیز خواهد بود(آن متغیر هیچ مقداری نخواهد داشت). بعنوان مثال:

```
>>> myval = None
>>> print(myval)
None
```

نسبت داده یک مقدارNone به یک متغیر یکی از راه‌های بازنشانی آن به حالت تهی اصلی‌اش است. مقداردهی یک متغیر باNone نیز یکی از راه‌های تعریف یک متغیر بدون تعیین مقدار آن است. اینکار زمانی انجام می‌شود که بعداً در برنامه به یک متغیر نیاز خواهید داشت ولی قصد دارید در ابتدای برنامه تمامی متغیرها را تعریف کنید. برنامه‌نویسان غالباً متغیرهای خود را در ابتدای یک برنامه تعریف می‌کنند زیرا قراردادن آنها در آنها، مشاهده نام‌های تمامی متغیرهای مورداستفاده در یک تکه کد را آسان-تر می‌کند.

همچنین میتوانید وجودNone را در یک دستورif وارسی کنید:

```
>>> myval = None
>>> if myval == None:
        print("The variable myval doesn't have a value")
The variable myval doesn't have a value
```

[1] assign

اینکار زمانی سودمند است که فقط قصد داشته باشید یک مقدار را برای متغیری که هنوز محاسبه نشده است، محاسبه کنید.

تفاوت میان رشته‌ها و اعداد

ورودی کاربر چیزی است که یک نفر روی صفحه‌کلید وارد می‌کند- میتواند یک کاراکتر، یک کلید پیکان[1] یاENTER فشرده شده یا هر چیز دیگری باشد. ورودی کاربر در پایتون بصورت یک رشته دریافت می‌شود به این معنی که وقتی عدد۱۰ را روی صفحه‌کلید تایپ می‌کنید، پایتون عدد۱۰ را در یک متغیر بصورت یک رشته ذخیره می‌کند نه یک عدد.

چه تفاوتی میان عدد۱۰ و رشته'۱۰' وجود دارد؟ این دو در نظر ما یکسان هستند با این تفاوت که یکی از آنها بین نقل قول تکی قرار گرفته است. ولی برای رایانه این دو با هم متفاوت هستند.

بعنوان مثال فرض کنید مقدار متغیرage را با یک عدد در دستورif مقایسه می‌کنیم:

```
>>> if age == 10:
        print("What's the best way to speak to a monster?")
        print("From as far away as possible!")
```

سپس متغیرage را برابر با عدد ۱۰ قرار می‌دهیم:

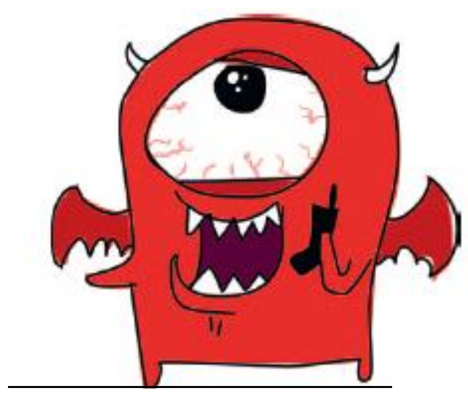
```
>>> age = 10
>>> if age == 10:
        print("What's the best way to speak to a monster?")
        print("From as far away as possible!")
What's the best way to speak to a monster?
From as far away as possible!
```

همانطوریکه مشاهده می‌کنید دستورprint اجرا می‌شود.

سپس age را برابر با رشته '10' قرار می‌دهیم (با علامت نقل قول):

```
>>> age = '10'
>>> if age == 10:
        print("What's the best way to speak to a monster?")
        print("From as far away as possible!")
```

در اینجا کد دستورprint اجرا نمی‌شود زیرا پایتون عدد داخل نقل قول(یک رشته) را بعنوان یک عدد درنظر نمی‌گیرد.



[1] arrow

خوشبختانه، پایتون توابع جادویی در اختیار دارد که میتوانند رشته‌ها را به اعداد و اعداد را به رشته‌ها تبدیل کنند. بعنوان مثال، با کمک int میتوانید رشته'10' را به یک عدد تبدیل کنید:

```
>>> age = '10'
>>> converted_age = int(age)
```

اکنون متغیر converted_age دارای عدد10 خواهد بود.

برای تبدیل یک عدد به یک رشته از str استفاده می‌کنیم:

```
>>> age = 10
>>> converted_age = str(age)
```

در این مورد converted_age، به جای عدد10، رشته10 را نگه میدارد.

یادآور می‌شویم زمانی که متغیر یک رشته('10' = age)درنظر گرفته شده بود، دستور if age == 10 هیچ چیزی را چاپ نمی‌کرد. اگر ابتدا متغیر را تبدیل کنیم، نتیجه‌ای کاملاً متفاوت بدست خواهیم آورد:

```
>>> age = '10'
>>> converted_age = int(age)
>>> if converted_age == 10:
        print("What's the best way to speak to a monster?")
        print("From as far away as possible!")
What's the best way to speak to a monster?
From as far away as possible!
```

حالا به این نکته توجه کنید:اگر بخواهید یک عدد اعشاری را تبدیل کنید، با پیام خطا روبرو خواهید شد زیرا تابع int انتظار دارد با یک عدد صحیح[1] مواجه شود.

```
>>> age = '10.5'
>>> converted_age = int(age)
Traceback (most recent call last):
        File "<pyshell#35>", line 1, in <module>
        converted_age = int(age)
ValueError: invalid literal for int() with base 10: '10.5'
```

یک ValueError چیزی است که پایتون از آن استفاده می‌کند تا به شما بگوید از مقدار مناسبی استفاده نمی‌کنید. برای اصلاح این خطا، از تابع float به جای int استفاده کنید. تابع flaot با اعداد غیرصحیح سروکار دارد.

```
>>> age = '10.5'
>>> converted_age = float(age)
>>> print(converted_age)
```

[1] Integer

10.5

اگر سعی کنید یک رشته فاقد عدد را به ارقام تبدیل کنید با پیام خطا روبرو خواهید شد:

```
>>> age = 'ten'
>>> converted_age = int(age)
Traceback (most recent call last):
        File "<pyshell#1>", line 1, in <module>
        converted_age = int(age)
ValueError: invalid literal for int() with base 10: 'ten'
```

آنچه آموختید

در این فصل یاد گرفتید که چگونه با دستورات if برای ساخت بلوک‌های کدی استفاده کنید که فقط زمانی اجرا می‌شوند که شرط(شروط)خاصی برقرار باشند. دیدید که چگونه با استفاده از elif میتوانید دستورات if را بسط دهید بطوریکه بخش‌های مختلف کد تحت شرایط مختلف اجرا شوند و چگونه از کلیدواژه else برای اجرای کد استفاده کنید اگر هیچیک از شرط‌ها برقرار(true) نباشند. همچنین یاد گرفتید که چگونه میتوانید شرطها را با استفاده از کلیدواژه‌های and و or ترکیب کنید بطوریکه قراردادشتن اعداد در یک رنج را وارسی کنید و چگونه با کمک int,str و float رشته‌ها را به اعداد تبدیل نمایید. دریافتید که هیچ‌چیز(None)برای پایتون معنا دارد و میتوان از آن برای بازنشانی متغیرها به حالت تهی اولیه‌شان استفاده نمود.

(چیستان)معماهای برنامه‌نویسی

معماهای زیر را با استفاده از شرط‌ها و دستور if حل کنید.پاسخ‌ها در *http://python-for-kids.com/* ارائه شده‌اند.

۱- آیا ثروتمند هستید؟

فکر می‌کنید کد زیر چه کاری انجام می‌دهد؟بدون تایپ کردن در محیط برنامه،پاسخ را پیدا کنید و سپس پاسخ خود را وارسی نمایید.

```
>>> money = 2000
>>> if money > 1000:
        print("I'm rich!!")
else:
        print("I'm not rich!!")
```

```
print("But I might be later…")
```

۲- twinkies

یک دستور if بنویسید که کمتر بودن تعداد Twinkies(در متغیر twinkies) از ۱۰۰ یا بیشتر بودن آن از ۵۰۰ را وارسی کند. برنامه شما بایستی در صورت برقرار بودن(true) شرط، پیام "Too few or too many" (خیلی کم یا خیلی زیاد) را چاپ کند.

۳- فقط عدد درست

یک دستور if بنویسید که تعیین کند مقدار پول در متغیر money بین ۱۰۰ و ۵۰۰ است یا بین ۱۰۰۰ و ۵۰۰۰.

٤- من میتوانم با این نینجاها مبارزه کنم

یک دستور if بنویسید که اگر متغیر ninja جاوی عددی کمتر از ۵۰ بوده، رشته "That's too many" (تعداد آنها خیلی زیاد است)را چاپ کند و اگر متغیر ninja حاوی عددی کمتر از ۳۰ بود، "It'll be a struggle, but I can take 'em" (این یک مبارزه است ولی میتوانم آن ها را شکست دهم)را چاپ کند و اگر این عدد کمتر از ۱۰ بود، "I can fight those ninjas!" (من میتوانم با این نینجاها مبارزه کمک) را چاپ نماید. کد خود را مقدار زیر امتحان کنید:

```
>>> ninjas = 5
```

فصل٦

Going loopy (بریم برای حلقه زدن)

هیچ چیز بدتر این است که یک کار را بارها و بارها تکـرار کنـیم. ایـن کـه برخـی مـردم بـرای خوابیدن مجبور هستند تعداد گوسفندان را بشمارند، دلیل خاص خود را داشته و هیچ ارتباطی به قـدرت القایی گوسفندان، این پستانداران پشمالو ندارد. دلیل این موضوع این است که تکرار بی‌پایـان یـک چیـز خسته‌کننده است و اگر روی چیز جالبی تمرکز نکرده‌اید، ذهن(مغز) شما راحـت‌تـر بـه خـواب خواهـد رفت.

برنامه‌نویسان بویژه خودشان تمایلی به تکرار کردن ندارند مگـر ایـن کـه بخواهند خواب عمیقی داشته باشند. بسیاری از زبان‌های برنامه‌نویسی از چیزی به نام حلقهfor برخودار هستند که چیزهای دیگری مانند دستورات برنامـه‌نویسـی و بلوک‌های کد را بطور خودکار تکرار می‌کنند.

در این فصل نگاهی داریم به حلقه‌هایfor و نـوع دیگـری از حلقـه کـه پایتون ارائه می‌کند، یعنی حلقهWhile.

استفاده از حلقه‌های for

برای اینکه در پایتون، hello پنج‌بار چاپ شود میتوانید کار زیر را انجام دهید:

```
>>> print("hello")
hello
>>> print("hello")
```

```
hello
>>> print("hello")
hello
>>> print("hello")
hello
>>> print("hello")
hello
```

ولی اینکار کسل‌کننده می‌باشد. درعوض میتوانید از یک حلقه‌for برای کاهش تعداد تایپ‌کردن-ها و تکرارها استفاده کنید:

```
❶ >>> for x in range(0, 5):
❷         print('hello')
        hello
        hello
        hello
        hello
        hello
```

میتوان از تابع‌range در ❶ برای ساختن یک لیست از اعداد از عدد شروع تا یک عدد قبل از عدد پایانی استفاده کرد. اینکار کمی سردرگم‌کننده است. اجازه دهید تابع‌range را با تابع‌list ترکیب کرده و ببینیم چگونه کار می‌کند. تابع‌range درواقع یک لیست از اعداد را درست نمی‌کند؛ بلکه یک تکرارکننده[1]را برمی‌گرداند که نوعی از شیء پایتون است و بویژه برای کار با حلقه‌ها طراحی شده است. هرچند، اگرrange را باlist ترکیب کنیم به لیستی از اعداد دست خواهیم یافت:

```
>>> print(list(range(10, 20)))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

در مورد حلقه‌for، کد خط ❶ به پایتون می‌گوید که کار زیر را انجام دهد:

- آغاز شمارش از0 و توقف قبل از رسیدن به5
- برای هر تعدادی که می‌شماریم، ذخیره کردن مقدار در متغیرx

سپس پایتون بلوک کد خط❷ را اجرا می‌کند. توجه داشته باشید که چهار فاصله اضافی در ابتدای خط ❷ وجود دارد(درمقایسه با خط❶). IDLE بطور خودکار این تورفتگی را برای شما انجام می‌دهد.

وقتی بعد از خط دومENTER را می‌زنیم، پایتون پنج بار"hello" را چاپ می‌کند.

همچنین میتوانیم برای شمارش helloها از x در دستورprint استفاده کنیم.

[1] iterator

```
>>> for x in range(0, 5):
        print('hello %s' % x)
hello 0
hello 1
hello 2
hello 3
hello 4
```

اگر بازهم میخواهیم از حلقهfor استفاده نکنیم، کد ما اینگونه خواهد بود:

```
>>> x = 0
>>> print('hello %s' % x)
hello 0
>>> x = 1
>>> print('hello %s' % x)
hello 1
>>> x = 2
>>> print('hello %s' % x)
hello 2
>>> x = 3
>>> print('hello %s' % x)
hello 3
>>> x = 4
>>> print('hello %s' % x)
hello 4
```

بنابراین استفاده از loop ما را از دست نوشتن هشت خط اضافی کد خلاص می‌کند. برنامه-نویسان ماهر از این که بیش از یکبار کاری را انجام دهند متنفر هستند بنابراین حلقهfor یکی از محبوب-ترین دستورات در یک زبان برنامه‌نویسی به شمار می‌آید.

نیازی نیست هنگام نوشتن حلقه‌هایfor، از توابع range و list استفاده کنید. همچنین میتوانید از لیستی که ایجد کرده‌اید استفاده کنید مثلاً لیست خرید فصل ۳:

```
>>> wizard_list = ['spider legs', 'toe of frog', 'snail tongue',
                   'bat wing', 'slug butter', 'bear burp']
>>> for i in wizard_list:
        print(i)
spider legs
toe of frog
snail tongue
bat wing
```

slug butter
bear burp

این کد راهی است برای بیان «برای هر آیتم در wizard_list مقدار را در متغیر i ذخیره کرده و سپس محتویات متغیر را چاپ کنید».

بازهم اگر میخواهیم از حلقه for استفاده نکنیم، بایستی کار زیر را انجام دهیم:

```
>>> wizard_list = ['spider legs', 'toe of frog', 'snail tongue',
                   'bat wing', 'slug butter', 'bear burp']
>>> print(wizard_list[0])
spider legs
>>> print(wizard_list[1])
toe of frog
>>> print(wizard_list[2])
snail tongue
>>> print(wizard_list[3])
bat wing
>>> print(wizard_list[4])
slug butter
>>> print(wizard_list[5])
bear burp
```

بازهم، حلقه ما را از دست تایپ کردن خلاص کرد. اجازه دهید حلقه دیگری بنویسیم. کد زیر را در محیط برنامه نویسید. بایستی بطور خودکار تورفتگی را در کد ایجاد کنید:

```
❶>>> hugehairypants = ['huge', 'hairy', 'pants']
❷>>> for i in hugehairypants:
❸        print(i)
❹        print(i)
❺
❻huge
        huge
        hairy
        hairy
        pants
        pants
```

در خط اول❶، یک لیست حاوی 'huge', 'hairy' و'pants' را ایجاد می‌کنیم. در خط بعد❷، در آیتم‌های این لیست حلقه می‌زنیم و سپس هر آیتم به متغیر i نسبت داده می-

شود. محتویات متغیر را در دو خط بعد دوبار چاپ می‌کنیم(❸و❹). در خط خالی بعد ❺، دکمه ENTER را فشار داده و به پایتون می‌گوییم که بلوک را خاتمه دهد و سپس کد را اجرا و هر مؤلفه از لیست ❻را دوبار چاپ کند.

یادآور می‌شویم که اگر تعداد فاصله‌ها را بدرستی وارد نکنید، با پیام خطا روبرو خواهید شد. اگر کد قبلی را با یک فاصله اضافی در خط چهارم ❹ وارد کنید، پایتون خطای تورفتگی را نمایش خواهد داد.

```
>>> hugehairypants = ['huge', 'hairy', 'pants']
>>> for i in hugehairypants:
        print(i)
         print(i)
SyntaxError: unexpected indent
```

همانطوریکه در فصل ۵ آموختید، پایتون انتظار دارد تعداد فاصله‌ها در یک بلوک همسان باشد. تا مادامیکه از یک عدد(تعداد) برای هر خط جدید استفاده می‌کنید (بعلاوه این که باعث می‌شود خوانن کد برای انسان آسان‌تر شود) اهمیتی ندارد چند فاصله اضافه کرده باشید.

در اینجا مثال پیچیده‌تری از یک حلقه for با دو بلوک کد وجود دارد:

```
>>> hugehairypants = ['huge', 'hairy', 'pants']
>>> for i in hugehairypants:
        print(i)
        for j in hugehairypants:
            print(j)
```

بلوک‌های در کجاین این کد قرار دارند؟ اولین بلوک،اولین حلقه for است:

```
hugehairypants = ['huge', 'hairy', 'pants']
for i in hugehairypants:
        print(i) #
        for j in hugehairypants: # These lines are the FIRST block.
            print(j) #
```

دومین بلوک یک خط print در دومین حلقه for است:

```
❶ hugehairypants = ['huge', 'hairy', 'pants']
for i in hugehairypants:
        print(i)
❷       for j in hugehairypants:
❸       print(j) # This line is also the SECOND block.
```

آیا میتوانید بفهمید این تکه کد چه کاری انجام می‌دهد؟ بعـد از ایـن کـه یـک لیسـت بـه نـام hugehairypants در ❶ ایجاد شد، میتوانید آن را از دو خط بعدی که فراخوانی کنید که تک تک آیتم‌هـای لیست را پردازش کرده و هر یک را چاپ می‌کند. هرچند در ❷ بازهم لیست را جستجو کرده و ایـن بـار مقدار را به متغیر j نسبت می‌دهد و در ❸ مجدداً هر آیتم را چاپ می‌کند. کد در ❷ و ❸ هنوزهم بخشی از حلقه for است به این معنی که با گذر کردن حلقه for از سراسر لیست، برای هر آیتم اجرا خواهند شد.

پس زمانی که این کد اجرا می‌شود، بایستی داشته باشیم huge و بعد از آن huge, hairy, pants, و سپس hairy و بدنبال آن huge, hairy, pants, و به همین ترتیب.

کد را در برنامه پایتون وارد کرده و خواهید داشت:

```
>>> hugehairypants = ['huge', 'hairy', 'pants']
>>> for i in hugehairypants:
❶       print(i)
        for j in hugehairypants:
❷               print(j)
◈huge
huge
hairy
pants
◈hairy
huge
hairy
pants
◈pants
huge
hairy
pants
```

پایتون حلقه او را وارد کرده و یک آیتم را از لیست ❶ چاپ می‌کند. سـپس، حلقـه دوم را وارد کرده و تمامی آیتم‌های لیست ❷ را چاپ می‌کند. در ادامه با فرمان print(j) ادامه می‌دهد، آیـتم بعـدی را در لیست اجرا کرده و لیست کامل را مجدداً با print(j) چـاپ مـی‌کنـد. در خروجی،خطـوطی کـه بـا ◈ مشخص شده‌اند با دستور print(i) چاپ می‌شوند. خطوط بدون علامت با دستور print(j) چاپ خواهنـد شد.

آیا میتـوان کـاری بـه غیـر از چـاپ واژه‌هـا را انجـام داد؟ محاسبه انجام شده در فصل ۲ را یادآور می‌شویم جاییکه میخواسـتیم بفهمیم اگر از اختراع پدربزرگتان برای دوبرابر کردن سکه‌ها استفاده

کنید، آخر سال چند سکه طلا خواهید داشت؟ کد برنامه اینگونه خواهد بود:

```
>>> 20 + 10 * 365 – 3 * 52
```

به این معنی: ۲۰ سکه پیدا شده بعلاوه ۱۰ سکه جادویی ضربدر۳۶۵ روز سال، منهای سه سکه در یک هفته توسط کلاغ سرقت شده است.

جالب خواهد بود وقتی می‌بینید هر هفته چقدر به انبوه سکه‌های شما اضافه می‌شود. میتوانیم اینکار را با حلقهfor دیگری اجام دهیم ولی این بار بایستی مقدار متغیر magic_coins را تغییر دهیم بنابراین تعداد کل سکه‌های جادویی هرهفته را نشان می‌دهد.۱۰ سکه جادویی برای هر روز از هفت روز هفته وجود دارد بنابراین magic_coins بایستی ۷۰ باشد:

```
>>> found_coins = 20
>>> magic_coins = 70
>>> stolen_coins = 3
```

میبینیم که گنج ما هر هفته با ایجاد متغیر دیگری به نامcoins و استفاده از یک حلقه افزایش می-یابد:

```
>>> found_coins = 20
>>> magic_coins = 70
>>> stolen_coins = 3
❶>>> coins = found_coins
❷ >>> for week in range(1, 53):
❸       coins = coins + magic_coins - stolen_coins
❹       print('Week %s = %s' % (week, coins))
```

در ❶ متغیر coins با مقدار متغیر found_coins بارگذاری(پر) شده است؛ این عدد آغازین ما است. خط بعدی در ❷ حلقهfor را برپا می‌کند که فرمان‌ها را در بلوک اجرا خواهد کرد (بلوک از خطوط ❸و ❹ تشکیل شده است). هر بار که حلقه می‌زند، متغیر week با عدد بعدی در رنج ۱تا۵۲ بارگذاری(پر) می‌گردد.

خط ❸ اندکی پیچیده‌تر است. اساساً هر هفته میخواهیم تعداد سکه‌های جادویی را اضافه کرده و تعداد سکه‌های سرقت شده را کسر کنیم. متغیر coin را چیزی مانند یک صندوق گنج تصور کنید. هر هفته سکه جدیدی به این صندوق اضافه می‌شود. بنابراین این خط به این معنی است: «جایگزین کردن تعداد سکه‌های جاری با محتویات متغیر coin و جمع زدن آن با چیزی که این هفته خلق کرده‌ام». اساساً علامت(=) یک تکه برجسته از کدی است که می‌گوید «محاسبه سمت راست، ذخیره آن برای بعدی و استفاده از نام در سمت چپ».

خط ❹ یک دستور print است که از متغیرهای جایگزین(placeholder)[1] استفاده می‌کند و تعداد هفته و تعداد کل سکه‌ها (تا اینجا) را روی صفحه نمایش می‌دهد. (اگر اینکار از نظر شما منطقی به نظر نمی‌آید، مجدداً به صفحه ۳۰ با عنوان «گنجاندن(تعبیه) مقادیر در رشته‌ها»مراجعه کنید). بنابراین اگر این برنامه را اجرا کنید، چیزی شبیه به پنجره زیر را مشاهده خواهید کرد:

```
Python Shell
File Edit Debug Options Windows Help
Python 3.2.2 (default, Sep  4 2011, 09:51:08) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>> found_coins = 20
>>> magic_coins = 70
>>> stolen_coins = 3
>>> coins = found_coins
>>> for week in range(1, 53):
        coins = coins + magic_coins - stolen_coins
        print('Week %s = %s' % (week, coins))


Week 1 = 87
Week 2 = 154
Week 3 = 221
Week 4 = 288
Week 5 = 355
Week 6 = 422
Week 7 = 489
Week 8 = 556
Week 9 = 623
Week 10 = 690
Week 11 = 757
Week 12 = 824
Week 13 = 891
Week 14 = 958
Week 15 = 1025
Week 16 = 1092
Week 17 = 1159
Week 18 = 1226
Ln: 43 Col: 0
```

وقتی درباره حلقه‌زنی صحبت می‌کنیم...

یک حلقه for فقط یک نوع حلقه نیست که میتوانید آن را در پایتون بسازید. حلقه while هم وجود دارد. یک حلقه for یک حلقه با طول معین است درحالیکه زمانی از حلقه while استفاده می‌شود که از قبل درباره زمان توقف حلقه‌زنی آگاه نیستید.

یک راه‌پله با ۲۰ پله را تجسم کنید. راه‌پله داخل ساختمان بوده و میدانیم که به آسانی میتوانید از ۲۰ پله بالا بروید. یک حلقه for اینگونه است

```
>>> for step in range(0, 20):
print(step)
```

[1] جانگهدار

حال یک راه‌پله را تصور کنید که در دامنه واقع است. دامنه واقعاً مرتفع است و قبل از این که به بالای آن برسید انرژی شما تمام می‌شود یا این که شرایط جوی ممکن است بد شود و شما مجبور شوید توقف کنید. حلقه while اینگونه است.

```
step = 0
while step < 10000:
        print(step)
        if tired == True:
                break
        elif badweather == True:
                break
        else:
                step = step + 1
```

اگر سعی کنید این کد را وارد و آن را اجرا کنید، با پیام خطا مواجه خواهید شد. چرا؟ زیرا متغیر tired و badweather را نساخته‌ایم. هرچند کد ما قابل اجرای یک برنامه را ندارد ولی یک نمونه اساسی از یک حلقه while را به تصویر می‌کشد.

با ایجاد متغیری به نام step=0 شروع می‌کنیم. سپس، حلقه while را می‌سازیم که بررسی می‌نماید آیا متغیر step که تعداد کل پله‌ها از پایین تا بالای دامنه است، از ۱۰۰۰۰ (step<10000) کوچکتر می‌باشد یا خیر. تا مادامیکه step<10000 باشد، پایتون بقیه کد را اجرا می‌کند.

با(print(step مقدار متغیر را چاپ کرده و True بودن متغیر tired را با if tired= =True بررسی می‌کنیم(True یک مقدار بولین[1] نامیده می‌شود که در فصل ۸ درباره آن خواهیم آموخت). دراینصورت، از کلیدواژه break برای خروج از حلقه استفاده می‌کنیم. کلیدواژه break راهی است برای(بلافاصله) بیرون پریدن از حلقه(به بیان دیگر،متوقف کردن آن) که هم در while و هم در loop جواب می‌دهد. در اینجا شاهد اثر بیرون پریدن از بلوک و وارد شدن به حلقه step=step+1 هستیم.

خط: elif badweather == True، True بودن متغیر badweather را بررسی می‌کند. دراینصورت، کلیدواژه break ازحلقه خارج می‌شود. اگر نه tired و نه badweather، (True(else نباشند، 1 را به متغیر step اضافه کرده(step=step+1) و حلقه ادامه می‌یابد.

بنابراین گام‌های یک حلقه loop بصورت زیر هستند:

[1] Boolean

۱- وارسی شرط

۲- اجرای کد در بلوک

۳- تکرار

بطور متداول، یک حلقهwhile را میتوان با دو شرط به جای یک شرط ساخت:

```
❶>>> x = 45
❷>>> y = 80
❸>>> while x < 50 and y < 100:
        x = x + 1
        y = y + 1
        print(x, y)
```

در ❶ متغیر x را با مقدار45 و در ❷ متغیر y را با مقدار80 می‌سازیم. حلقه دو شرط را در ❸ وارسی می‌کند: x کمتر از 50 و y کمتر از100. درحالیکه هر دو شرط برقرار هستند، خطوط بعدی اجرا شده و 1 به هر دو متغیر اضافه می‌گردد و سپس آنها چاپ می‌شوند. خروجی کد بصورت زیر خواهد بود:

```
46 81
47 82
48 83
49 84
50 85
```

آیا می‌فهمید چگونه عمل می‌کند؟

شمار را در45 برای متغیر x و در80 برای متغیر y شروع کرده و سپس هربار که کد در حلقه اجرا می‌شود، افزایش می‌یابد(افزودن 1 به هر متغیر)تا مادامیکه x<50 و y<100 باشد، حلقه اجرا خواهد شد. بعد از ۵بار حلقه‌زنی(هر بار 1 به هر متغیر اضافه می‌شود)مقدارx به50 می‌رسد. حالا شرط اول(x<50) دیگر برقرار نیست، بنابراین پایتون میداند که باید حلقه را متوقف کند.

کاربرد متداول دیگر حلقهwhile، ساختن حلقه‌های نیمه-بی‌پایان است. این نوعی حلقه است که میتواند تا بینهایت ادامه داشته باشد ولی درواقع تا زمانی ادامه می‌یابد که اتفاقی برای خروج از حلقه اتفاق بیافتد. بعنوان مثال:

```
while True:
        lots of code here
        lots of code here
        lots of code here
        if some_value == True:
```

```
break
```

شرط حلقه while،True است که همواره برقرا است بنابراین کد بلوک همواره اجرا می-شود(بنابراین حلقه بی‌پایان[1]است). تنها درصورتیکه متغیر some_value صادق باشد، پایتون از حلقه خارج خواهد شد. در بخش «استفاده از randint برای انتخاب یک عدد تصادفی» در صفحه ۱۳٤ مثال بهتری برای درک این مورد ارائه شده است ولی ابتدا باید فصل هفت را بطور کامل درک کنید.

آنچه آموختید

در این فصل، از حلقه‌ها برای انجام وظایف تکراری بدون تکرار همه آنها استفاده کردیم. با نوشتن وظایف درون بلوک‌های کد، که درنو حلقه قرار دادیم، به پایتون گفتیم میخواهیم چه چیزی تکرار شود. از دو نوع حلقه استفاده کردیم: حلقه‌های for و حلقه‌های while که مشابه بوده ولی به شیوه‌های متفاوتی بکار برده می‌شوند. همچنین از کلیدواژه break برای توقف حلقه‌زنی استفاده کردیم- یعنی خروج از حلقه.

(چیستان)معماهای برنامه‌نویسی

در اینجا مثال‌هایی از حلقه‌ها ارائه شده است که میتوانید خودتان امتحان کنید. پاسخ‌ها را میتوان در *http://python-for-kids.com/.* مشاهده کرد.

۱- حلقه Hello

فکر می‌کنید کد زیر چه کار می‌کند؟ ابتدا، حدس بزنید چه اتفاقی می‌افتد و سپس کد را در پایتون اجرا کرده تا ببینید حق با شما بوده است یا نه.

```
>>> for x in range(0, 20):
        print('hello %s' % x)
        if x < 9:
            break
```

۲- اعداد زوج[2]

حلقه‌ای بسازید که اعداد زوج را آنقدر چاپ کند تا به سن شما برسد یا درصورتیکه سن شما یک عدد فرد[1]باشد، اعداد فرد را آنقدر چاپ کند تا به سن شما برسد. بعنوان مثال، میتواند چیزی مشابه خروجی زیر را چاپ کند:

---

[1] eternal

[2] even

```
2
4
6
8
10
12
14
```

۳- پنج ماده موردعلاقه من

لیستی حاوی ۵ ماده مختلف ساندویچ بسازید:

```
>>> ingredients = ['snails', 'leeches', 'gorilla belly-button lint',
        'caterpillar eyebrows', 'centipede toes']
```

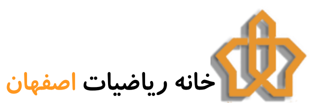
حال حلقه‌ای درست کنید که لیست را چاپ کند(شامل اعداد):

```
1 snails
2 leeches
3 gorilla belly-button lint
4 caterpillar eyebrows
5 centipede toes
```

٤- وزن شما روی ماه

اگر روی ماه ایستاده باشید، وزن شما ١٦,٥٪ وزن شما روی زمین خواهد بود. میتوانید این عدد را با ضرب وزن زمین در٠,١٦٥ محاسبه کنید.

اگر اگر تا ١٥ سال دیگر، هر سال یک کیلو اضافه کنید، هر سال که به ماه می‌روید وزن شما چقدر خواهد بود و بعد از ١٥سال چه وزنی خواهید داشت؟ با یک حلقهfor که وزن هر ساله شما روی ماه را چاپ می‌کند، برنامه‌ای بنویسید.



فصل۷

با توابع و ماژول‌ها کد خود را بازیابی کنید

درباره این موضوع فکر کنید که هر روز چه چیزهایی را دور می‌ریزید: بطری‌های آب، قوطی-های سودا، پاکت‌های چیپس سیب‌زمینی، کاغذ پلاستیکی ساندویچ، پاکت‌های حاوی زردک یا تکه‌های سیب، ساک خرید، روزنامه، مجله، و .... حال تصور کنید چه اتفاقی خواهد افتاد اگر تمامی این زباله‌ها در یک در انتهای مسیر رانندگی شما روی هم تل‌انبار شده ولی کاغذ، پلاستیک و قوطه‌های حلبی از آنها تفکیک نشده باشند.

البته، احتمالاً شما تا حد امکان بازیافت کرده‌اید زیرا هیچ کس دوست ندارد ربای رفتن به مدرسه از کوهی از زباله بالا برود. به جای نشستن در تلی از زباله، بطری‌های شیشه‌ای که بازیافت کرده‌اید ذوب شده و به بطری‌ها و کوزه‌های جدید تبدیل شده‌اند؛ کاغذ برایت بدی لبه کاغذ بازیافتی خمیر می‌شود؛ پلاستیک به کالاهای پلاستیکی سنگین‌تر تبدیل می‌شود. بنابراین اگر از چیزها مجدداً استفاده نکنیم، باید آنها را دور بریزیم.

در دنیای برنامه‌نویسی، استفاده مجدد اهمیت فوق‌العاده‌ای دارد. پرواضح است که برنامه شما زیر کوهی از زباله مدفون نمی‌گردد ولی اگر از برخی کارهای خود مجدداً استفاده نکنید، درنهایت در مسیر تکرار تایپ ریزبرنامه‌ها(stub) انگشتان شما از پار درخواهند آمد. استفاده مجدد میتواند باعث کوتاهتر شدن کد و سهولت قرائت آن شود.

همانطوریکه در این فصل خواهید آموخت، پایتون راه‌های مختلفی را برای استفاده مجدد از کد ارائه خواهد کرد.

استفاده از توابع

یکی از راه‌های بازیابی کد پایتون را شاهد بودید. در فصل قبل از توابع range و list برای شمارش پایتون استفاده کردیم.

```
>>> list(range(0, 5))
[0,1,2,3,4]
```

اگر شمارش بلد هستید، ساخت یک لیست از اعداد متوالی با تایپ این اعداد چندان دشوار نبوده ولی هر چه لیست طولانی‌تر باشد، تایپ بیشتری باید انجام دهید. هرچند اگر از توابع استفاده کنید، بسادگی میتوانید یک لیست با هزاران عدد بسازید.

در اینجا مثالی ارائه شده است که از توابع list و range برای تولید یک لیست از اعداد استفاده می‌کند:

```
>>> list(range(0, 1000))
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16...,997,998,999]
```

توابع[1] تکه‌هایی[2] از کد هستند که به پایتون می‌گویند کاری را انجام دهد. یک راه برای استفاده مجدد از کد وجود دارد- میتوانید از توابع در برنامه‌های خود بارها و بارها استفاده کنید.

زمانیکه برنامه‌های ساده‌ای می‌نویسید، استفاده از توابع بسیار آسان خواهد بود. زمانیکه نوشتن برنامه‌های طولانی و پیچیده‌تر را آغاز می‌کنید، مثلاً بازی، توابع ضرورت پیدا می‌کنند (فرض کنید میخواهید نوشتن برنامه را در این قرن به پایان ببرید)

بخش‌هایی از یک تابع

یک تابع سه بخش دارد: نام[3]، پارامترها[4] و بدنه[5]. در اینجا مثالی از یک تابع ساده ارائه شده است:

```
>>> def testfunc(myname):
print('hello %s' % myname)
```

---

[1] functions
[2] chunk
[3] name
[4] parameters
[5] body

نام این تابع testfunc است. این تابع یک پارامتر به نام myname داشته و بدنه آن، بلوکی از کد و بدنبال آن خطی است که باdef (مخففdefine) آغاز می‌شود. یک پارامتر یک متغیر است کـه تنهـا در زمانی که از تابع استفاده می‌شود،وجود خارجی دارد.

با فراخوانی نام تابع، با استفاده از پرانتز دور مقدار پارامتر، تابع را اجرا کنید:

```
>>> testfunc('Mary')
hello Mary
```

یک تابع میتواند دو، سه یا چندین پارامتر داشته باشد:

```
>>> def testfunc(fname, lname):
print('Hello %s %s' % (fname, lname))
```

دو مقدار برای این پارامترهای، با کاما(,) از هم جدا می‌شوند:

```
>>> testfunc('Mary', 'Smith')
Hello Mary Smith
```

میتوانیم ابتدا متغیرها را ایجاد و سپس فراخوانی تابع را با آنها انجام دهیم:

```
>>> firstname = 'Joe'
>>> lastname = 'Robertson'
>>> testfunc(firstname, lastname)
Hello Joe Robertson
```

غالباً از تابع برای برگرداندن یک مقدار، با استفاده از دستور**return** استفاده می‌شود. بعنوان مثال، میتوانید برای محاسبه مقدار پولی که پس‌انداز کرده‌اید، تابعی بنویسید:

```
>>> def savings(pocket_money, paper_route, spending):
return pocket_money + paper_route – spending
```

این تابع سـه پـارامتر دارد. ایـن تـابع دو پـارامتر اول را بـا هـم جمـع کـرده(pocket_money‌و spending)و پارامتر آخر را از آن کسر می‌کند(spending). نتیجه بازگردانده شده و میتواند به یـک متغیـر نسبت داده شده(همانطوریکه بقیه مقادیر به متغیرها نسبت داده شدند)یا چاپ گردد.

```
>>> print(savings(10, 10, 5))
15
```

متغیرها و حوزه[1]

زمانیکه تابع به اجرا خاتمه می‌دهد، نمیتوان از متغیری که درون بدن یک تابع قرار دارد، مجـدداً استفاده نمود زیرا فقط درون تابع وجود دارد. در دنیای برنامه‌نویسی، (حوزه)scope نامیده می‌شود.

[1] scope

در اینجا نگاهی به یک تابع ساده خواهیم داشت که از دو متغیر استفاده کرده ولی هیچ پارامتری ندارد:

```
❶ >>> def variable_test():
        first_variable = 10
        second_variable = 20
❷       return first_variable * second_variable
```

در این مثال، تابعی به نام variable_test را در ❶ می‌سازیم که دو متغیر (first_variable و second_variable) را درهم ضرب کرده و نتیجه را در ❷ برمی‌گرداند.

```
>>> print(variable_test())
200
```

اگر این تابع را با استفاده از Print چاپ کنیم، نتیجه: 200 بدست خواهد آمد. هرچند اگر بخواهیم محتوای first_variable را (یا **second_variable** برای این موضوع) بیرون بلوک کد در تابع چاپ کنیم، با پیام خطا روبرو خواهیم شد:

```
>>> print(first_variable)
Traceback (most recent call last):
    File "<pyshell#50>", line 1, in <module>
        print(first_variable)
NameError: name 'first_variable' is not defined
```

اگر یک متغیر در بیرون از تابع تعریف شده باشد، حوزه متفاوتی خواهد داشت. بعنوان مثال، قبل از این که تابع‌مان را بسازیم، یک متغیر را تعریف کرده و سپس از آن درون تابع استفاده می‌کنیم:

```
❶ >>> another_variable = 100
  >>> def variable_test2():
        first_variable = 10
        second_variable = 20
❷        return first_variable * second_variable * another_variable
```

در این کد، حتی اگر نتوان از متغیرهای first_variable **variables** و second_variable بیرون تابع استفاده نمود، میتوان از متغیر another_variable (که بیرون تابع در ❶ ساخته شده است) درون آن در ❷ استفاده نمود.

در اینجا نتیجه فراخوانی این تابع نشان داده شده است:

```
>>> print(variable_test2())
20000
```

حال فرض کنید یک سفینه فضایی را با استفاده از یک چیز مقرون به صرفه مثلاً یک قوطی(کنسرو) حلبی مستعمل می‌سازید. فرض کنید هر هفته میتوانید دو قوطی را مسطح کرده تا دیواره-

های منحنی سفینه فضایی شما ساخته شود ولی برای تکمیل دماغه[۱] حدود ۵۰۰ قوطی نیاز خواهید داشت. بسادگی میتوان تابعی نوشت تا در برآورد زمان لازم برای مسطح کرده ۵۰۰ قوطی با فرض دو قوطی در هفته، به ما کمک کند.

اجازه دهید تابعی را بسازیم تا نشان دهیم هر هفته تا پایان سال چند قوطی مسطح شده خواهیم داشت.تابع ما تعداد قوطی‌ها را بعنوان یک پارامتر درنظر می‌گیریم.

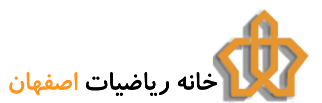
```
>>> def spaceship_building(cans):
        total_cans = 0
        for week in range(1, 53):
            total_cans = total_cans + cans
            print('Week %s = %s cans' % (week, total_cans))
```

در اولین خط تابع، متغیری به نام total_cans ساخته و به آن مقدار 0 می‌دهیم. یعنی یک حلقه برای هسته‌های سال ساخته و تعداد قوطی‌های مسطح شده در هر هفته را به آن اضافه می‌کنیم. این بلوک کد، محتوای تابع ما را می‌سازد. ولی بلوک کد دیگری نیز در این تابع وجود دارد: دو خط آخر که بلوک حلقه for را شکل می‌دهند.

این تابع را در برنامه وارد کرده و با مقادیر متفاوتی برای تعداد cans آن را فراخوانی می‌کنیم:

```
>>> spaceship_building(2)
Week 1 = 2 cans
Week 2 = 4 cans
Week 3 = 6 cans
Week 4 = 8 cans
Week 5 = 10 cans
Week 6 = 12 cans
Week 7 = 14 cans
Week 8 = 16 cans
Week 9 = 18 cans
Week 10 = 20 cans
(continues on...)
>>> spaceship_building(13)
Week 1 = 13 cans
Week 2 = 26 cans
```

[۱] fuselage



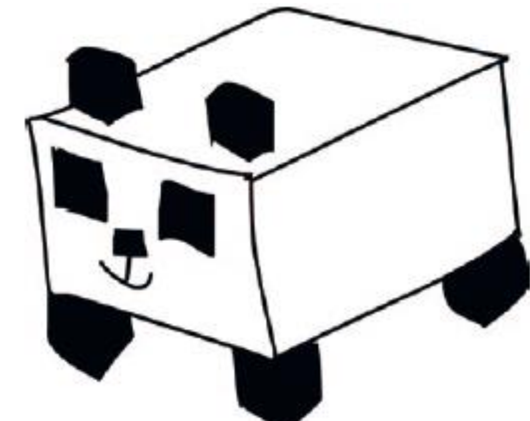

```
Week 3 = 39 cans
Week 4 = 52 cans
Week 5 = 65 cans
(continues on...)
```

میتوان مجدداً از این تابع با مقادیر مختلف برای تعداد قوطی‌های هر هفته نیز استفاده نمود که اندکی مؤثرتر از هر بار تایپ مجدد حلقه for برای امتحان کردن اعداد مختلف است.

همچنین میتوان توابع را در ماژول‌ها گروهبندی نمود که این همان جایی است که مزیت پایتون به چشم می‌آید.

استفاده از ماژول‌ها

از ماژول‌ها برای گروهبندی توابع، متغیرها و چیزهای دیگر به برنامههای بزرگتر و قدرتمندتر استفاده می‌شود. برخی از ماژول‌ها جزو پایتون هستند و شما میتوانید جداگانه ماژول‌های دیگر را دانلود کنید. ماژول‌ها به شما برای نوشتن برنامهها کمک می‌کنند(مثلاً tkinter که جزو پایتون است و PyGame که اینگونه نیست)، ماژول‌هایی برای دستکاری تصاویر(مثلاً PIL، کتابخانه تصویر پایتون) و ماژول-هایی برای ترسیم گرافیک‌های سه-بعدی(مثلاً Panda3D).



میتوان از ماژول‌ها برای انجام انواع کارهای سودمند استفاده نمود. بعنوان مثال اگر درحال طراحی یک بازی شبیه‌سازی هستید و میخواهید دنیای بازی واقعاً تغییر کند، میتوانید با استفاده از ماژول ذاتی پایتون به نام time، تاریخ و ساعت فعلی را محاسبه کنید:

```
>>> import time
```

در اینجا از فرمان import استفاده شده است تا به پایتون بگوییم میخواهیم از ماژول time استفاده کنیم.

سپس با استفاده از نماد نقطه[1] میتوانیم توابع دردسترس در این ماژول را فراخوانی کنیم(یادآور می‌شویم که از چنین توابعی برای کار با ماژول turtle در فصل ٤ استفاده کردیم مثلاً t.forward(50)).

بعنوان مثال، در اینجا نشان داده شده است که چگونه تابع asctime را با ماژول time فراخوانی می‌کنیم:

```
>>> print(time.asctime())
'Mon Nov 5 12:40:27 2012'
```

[1] dot

تابع asctime بخشی از ماژول time است که تاریخ و زمان فعلی را بصورت یک رشته برمی‌گرداند.

حال فرض کنید از کسی میخواهید از برنامه شما برای وارد کردن یک مقدار، شاید تاریخ تولد یا سن خود استفاده کند. میتوانید اینکار را با استفاده از دستور print برای نمایش پیام و ماژول sys (مخففsystem) که حاوی امکاناتی برای تعامل با خود سیستم پایتون است، انجام دهید. ابتدا، ماژول sys را وارد می-کنیم:

```
>>> import sys
```

درون ماژول sys یک شیء ویژه به نام stdin (برای ورودی استاندارد) است که تابع سودمندی به نام readline را تأمین می‌کند. از تابع readline برای خواندن یک خط از متن تایپ شده روی صفحه کلید تا فشرده شدن کلید ENTER استفاده می‌شود. (در فصل ۸ به کارکرد شیءها می‌پردازیم). برای تست readline کد زیر را در برنامه وارد می‌کنیم:

```
>>> import sys
>>> print(sys.stdin.readline())
```

سپس واژگانی را وارد کرده و ENTER را فشار دهید، واژگانی که در برنامه(shell) چاپ خواهند شد.

مجدداً به سراغ کدی می‌رویم که در فصل ۵ با استفاده از یک دستور if نوشتیم:

```
>>> if age >= 10 and age <= 13:
print('What is 13 + 49 + 84 + 155 + 97? A headache!')
else:
print('Huh?')
```

به جای ساختن متغیر age و نسبت دادن یک مقدار معین قبل از دستور if، میتوانیم از کسی بخواهیم مقدار را وارد کند. ولی ابتدا،باید کد را به یک تابع تبدیل کنیم:

```
>>> def silly_age_joke(age):
        if age >= 10 and age <= 13:
                print('What is 13 + 49 + 84 + 155 + 97? A headache!')
        else:
                print('Huh?')
```

حال میتوانید با وارد کردن نام تابع آن را فراخوانی کرده و با وارد کردن عدد در پرانتز، به او بگویید از چه عددی استفاده کند. آیا جواب می‌دهد؟

```
>>> silly_age_joke(9)
Huh?
```

```
>>> silly_age_joke(10)
What is 13 + 49 + 84 + 155 + 97? A headache!
```

بله جواب می‌دهد! حال تابع ask را برای سن یک فرد می‌نویسیم (میتوانید تابع را هر چندبار که میخواهید اضافه کرده یا تغییر دهید).

```
>>> def silly_age_joke():
        print('How old are you?')
❶       age = int(sys.stdin.readline())
❷       if age >= 10 and age <= 13:
            print('What is 13 + 49 + 84 + 155 + 97? A headache!')
        else:
            print('Huh?')
```

آیا تابع int را در ❶ تشخیص دادید که یک رشته را به یک عدد تبدیل می‌کند؟ به این دلیل از این تابع استفاده شده است که readline() هرآنچه را بصورت یک رشته وارد می‌شود برمی‌گرداند ولی ما بدنبال عدد هستیم تا بتوانیم اعداد 10 و 13 را در ❷ مقایسه کنیم.

برای اینکه این کار را خودتان انجام دهید، تابع را بدون هیچ پارامتری فراخوانی کرده و با ظاهر شدن How old are you? عددی را تایپ کنید.

```
>>> silly_age_joke()
How old are you?
10
What is 13 + 49 + 84 + 155 + 97? A headache!
>>> silly_age_joke()
How old are you?
15
Huh?
```

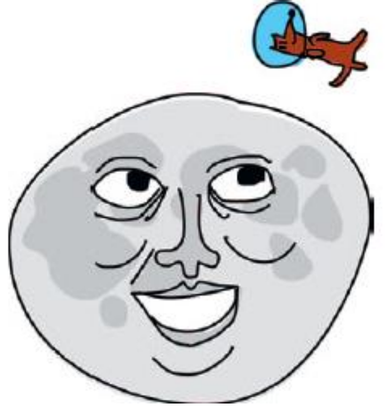
آنچه آموختید

در این فصل دیدید که چگونه میتوان با استفاده از توابع، تکه کدهایی را در پایتون نوشت که مجدداً قابل استفاده هستند و چگونه از توابع تأمین شده توسط ماژول‌ها استفاده نمود. آموختید که چگونه حوزه متغیرها مشاهده شدن آن‌ها درون یا بیرون توابع را کنترل می‌کند و چگونه با استفاده از کلیدواژه def توابع را تعریف کنید. همچنین دریافتیم که چگونه ماژول‌ها را وارد کنیم تا بتوانیم از محتوای آنها استفاده کنیم.

(چیستان)معماهای برنامه‌نویسی

مثال‌های زیر را بعنوان تمرین درنظر بگیرید و توابع خودتان را بسازید. پاسخ‌ها در *http:// python-for-kids.com/.* ارائه شده‌اند.

۱- تابع اصلی وزن ماه



در فصل٦ یک معمای برنامه‌نویسی برای ساختن یک حلقهfor برای تعیین وزن‌تان در ماه در یک دوره۱۵ساله مطرح گردید. این حلقهfor را بسادگی میتوان به یک تابع تبدیل کرد. تابعی بسازید که وزن شروع را گرفته و هر سال مقدار وزن را افزایش دهد. تابع جدید را با کد زیر فراخوانی کنید:

```
>>> moon_weight(30, 0.25)
```

۲- تابع وزن ماه و سال‌ها

تابعی که ساختید را درنظر بگیرید و برای محاسبه وزن در دوره‌های مختلف تغییر دهید مثلاً ۵سال یا ۲۰سال. مطمئن شوید که تابع را طوری تغییر می‌دهید که سه آرگومان داشته باشد: وزن ابتدایی، وزن بدست آمده در هرسال، و تعداد سال‌ها.

```
>>> moon_weight(90, 0.25, 5)
```

۳- برنامه وزن ماه

به جای یک تابع ساده، جاییکه مقادیر را بصورت پارامتر درنظر گرفته‌اید میتوانید یک برنامه کوچک نوشته که با استفاده از sys.stdin.readline() مقادیر را بپذیرد. در این مورد تابع را بدون هیچ پارامتری فراخوانی کنید:

```
>>> moon_weight()
```

تابع یک پیام را نمایش می‌دهد که وزن شروع را خواسته و پیام دوم، مقدار وزنی را میپرسد که هر سال اضافه می‌شود و درنهایت در یک پیام تعداد سال‌ها پرسیده می‌شود. چیزی شبیه به کد زیر را مشاهده خواهید کرد:

```
Please enter your current Earth weight
45
Please enter the amount your weight might increase each year
0.4
Please enter the number of years
12
```

بخاطر داشته باشید قبل از ساختن تابع‌تان، ابتدا ماژول sys را وارد کنید:

```
>>> import sys
```

فصل ۸

چگونه از کلاس‌ها و شیء‌ها[1] استفاده کنیم

چرا یک زرافه شبیه یک پیاده‌رو است؟ چون هم زرافه و هم پیاده‌رو «موجود»[2] هستند، و در زبان انگلیسی تحت عنوان «نام»[3] و در پایتون تحت عنوان «شیء» شناخته می‌شوند.

ایده‌ی اشیاء یک ایده‌ی بسیار مهم در دنیای رایانه محسوب می‌شود. شیء‌ها راهی برای سازماندهی کد در یک برنامه و شکستن موجودات برای تسهیل تفکر درباره ایده‌های پیچیده می‌باشند. (در فصل ٤ زمانی که با قلم-لاک‌پشت کار می‌کنیم از یک شیء استفاده خواهیم کرد).

برای درک کردن شیء‌ها در پایتون، بایستی درباره انواع شیء فکر کنیم. اجازه دهید با زرافه و پیاده‌رو کار را شروع کنیم.

یک زرافه یک نوع پستاندار است که گونه‌ای از یک حیوان نیز محسوب می‌شود- زنده است.

حال یک پیاده‌رو را درنظر می‌گیریم. چیز زیادی نمی‌توان درباره‌ی پیاده‌رو بیان کرد به جز این که یک موجود زنده نیست. اجازه دهید آن را شیء بی‌جان (یا به بیان دیگر، غیرزنده) بنامیم.

اصطلاحات پستاندار، حیوان، جاندار و بی‌جان، همگی راه‌هایی برای طبقه‌بندی موجودات هستند.

---

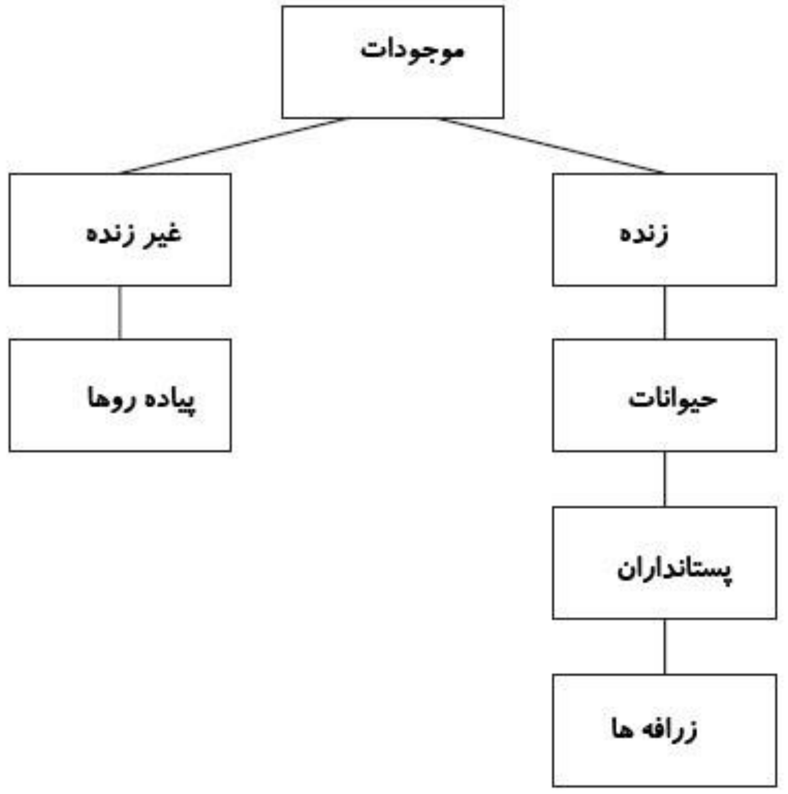
[1] object

[2] Thing

[3] noun

شکستن موجودات به کلاس‌ها

در پایتون، شیءها با کلاس‌ها[1] تعریف می‌شوند که میتوان آنها را راهی برای طبقه‌بندی شیءها به گروه‌ها درنظر گرفت. در این‌جا یک نمودار درختی از کلاس‌ها ارائه شده است که در آنجا زرافه‌ها و پیاده‌روها براساس تعاریف قبلی، جای داده شده‌اند.



Things کلاس اصلی است. در زیر کلاس Things، Inanimate و Animate وجود دارند. که بازهم به Sidewalks برای Inanimate و Animals, Mammals و Giraffes برای Animate شکسته می‌شوند.

می‌توانیم از کلاس‌ها برای سازماندهی بیت‌های کد پایتون استفاده کنیم. بعنوان مثال، ماژول turtle را درنظر بگیرید. هر موجودی که ماژول turtle پایتون بتواند انجام دهد- مثلاً حرکت رو به جلو، حرکت رو به عقب، چرخیدن به چپ و چرخیدن به راست- توابع در کلاس Pen هستند. یک شیء را می‌توان بعنوان یک عضو از یک کلاس درنظر گرفت و هر تعداد شیء که بخواهیم برای یک کلاس بنویسیم- به اختصار به آن خواهیم پرداخت.

حال اجازه دهید همان مجموعه کلاس‌های نمودار درختی‌مان را از بالا به پایین بسازیم. با استفاده از کلیدواژه‌ی class و سپس یک نام، کلاس‌ها را تعریف می‌کنیم. چون Things گسترده‌ترین کلاس است، ابتدا آن را خواهیم ساخت:

[1] class

```
>>> class Things:
        Pass
```

این کلاس راThings نامیده و از عبارتpass استفاده می‌کنیم تا بـه پـایتون بفهمـانیم کـه قصـد نداریم اطلاعاتی بیشتری را ارائه کنیم. ازpass زمانی استفاده می‌شود که می‌خـواهیم کـلاس یـا تـابعی را ایجاد کنیم ولی در آن زمان قصد نداریم جزئیات زیادی را ارائه نماییم.

سپس، کلاس‌های دیگر را اضافه کرده و روابطی را میان آنها ایجاد می‌کنیم.

فرزندان و والدین

اگر یک کلاس بخشی از کلاس دیگری باشد، آن را فرزند[1] آن کـلاس نامیـده و کلـس دیگـر را والد[2] آن می‌نامیم. کلاس‌ها، هم میتوانند فرزندان و هم، والدین کلاس‌های دیگر باشند. در نمودار درختـی ما، کلاسی که بالای کلاس دیگری قرار دارد، والد آن و کلاسی که زیـر آن قـرار دارد، فرزنـد آن اسـت. بعنوان مثال، Inanimate و Animate هردو فرزندان کلاسThings هستند به این معنی که Things والد آنها می‌باشد.

برای اینکه به پایتون بگوییم یک کلاس، فرزند کلاس دیگری است، نام کلاس والد را در پرانتز، بعد از نام کلاس جدید قرار می‌دهیم:

```
>>> class Inanimate(Things):
        pass
>>> class Animate(Things):
        pass
```

در اینجا، یک کلاس به نامInanimate ساخته و به پایتون می‌گوییم که کلاس والـد آنThings بـا کدclass Inanimate(Things) اسـت. سـپس یـک کـلاس بـه نـامAnimate سـاخته و بـا اسـتفاده از class Animate(Things) به پایتون می‌گوییم که کلاس والد آن نیزThings است.

اجازه دهید همین مورد را درخصوص کـلاس Sidewalks امتحـان کنـیم. کـلاسSidewalks را بـا کلاس والدInanimate می‌سازیم:

```
>>> class Sidewalks(Inanimate):
pass
```

و می‌تـوانیم کـلاس‌هـایAnimals, Mammals و Giraffes را بـا اسـتفاده از کـلاس‌هـای والدشـان بصورت زیر مرتب کنیم:

[1] child
[2] parent

```
>>> class Animals(Animate):
        pass
>>> class Mammals(Animals):
        pass
>>> class Giraffes(Mammals):
        pass
```

افزودن شیءها به کلاس‌ها

حالا یک مشت کلاس داریم ولی بایستی چه چیزی در این کلاس‌ها بگذاریم؟ مثلاً یک زرافه به نام Reginald داریم. می‌دانیم که متعلق به کلاس Giraffes است ولی در زبان برنامه‌نویسی بایستی از چه چیزی برای شرح و توصیف یک زرافه به نام Reginald استفاده کنیم؟ Reginald را یک شیء از کلاس Giraffes می‌نامیم (شاید با اصطلاح نمونه[1] یک کلاس نیز روبرو شوید). برای معرفی Reginald به پایتون، از تکه کد زیر استفاده می‌کنیم:

```
>>> reginald = Giraffes()
```

این کد به پایتون می‌گوید که یک شیء را در کلاس Giraffes ایجاد کرده و آن را به متغیر reginald نسبت دهد. نام کلاس نیز همانند یک تابع، درون پرانتز قرار می‌گیرد. در ادامه‌ی این فصل می‌بینیم که چگونه می‌توان شیءها را ساخت و از پارامترها درون پرانتز استفاده نمود.

ولی شیء reginald چه کاری انجام می‌دهد؟ خوب، در واقع در این لحظه کاری انجام نمی‌دهد. برای اینکه این شیء به درد بخورد، هنگامی که کلاس‌هایمان را می‌سازیم بایستی توابعی را تعریف کنیم که در کنار شیءهای آن کلاس قابل استفاده باشند. به جای اینکه فقط از کلیدواژه‌ی pass بعد از تعریف کلاس استفاده کنیم، می‌توانیم تعاریف تابع را اضافه کنیم.

تعریف توابع کلاس‌ها

در فصل ۷، توابع بعنوان راهی برای استفاده مجدد از کد معرفی گردیدند. زمانی که یک تابع مربوط به یک کلاس را تعریف می‌کنیم، این کار را برای تمامی توابع بطور یکسان انجام می‌دهیم مگر این که آن را نیز تعریف کلاس قرار داده باشیم. بعنوان مثال، در این جا تابع نرمالی وجود دارد که به یک کلاس مربوط نمی‌شود:

```
>>> def this_is_a_normal_function():
        print('I am a normal function')
```

[1] *instance*

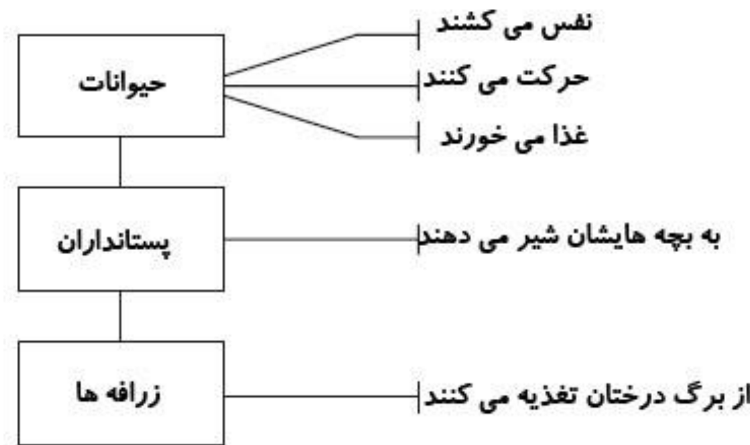
و در این جا دو تابع وجود دارد که به یک کلاس تعلق دارند:

```
>>> class ThisIsMySillyClass:
        def this_is_a_class_function():
                print('I am a class function')
        def this_is_also_a_class_function():
                print('I am also a class function. See?')
```

افزودن مشخصه‌های کلاس بصورت توابع

کلاس‌های فرزند کلاس Animate را که قبلاً تعریف کردیم، درنظر بگیرد. برای اینکه نشان دهیم هر کلاس چیست و چه کاری می‌تواند انجام دهد،می‌توانیم به هر کلاس، مشخصه‌هایی[1] را اضافه کنیم. یک مشخصه یک ویژگی[2] مشترک برای تمامی اعضای یک کلاس(و فرزندان آن) می‌باشد.

بعنوان مثال، وجه مشترک تمامی حیوانات چیست؟ همه نفس می‌کشند، حرکت می‌کنند و غذا می‌خورند. پستانداران چه وجه مشترکی دارند؟ پستانداران به فرزندان خود شیر می‌دهند، نفس می‌کشند، حرکت می‌کنند و غذا می‌خورند. می‌دانیم که زرافه از برگ‌های بالای درختان تغذیه می‌کند و همانند هر پستانداری، به فرزند خود شیر می‌دهد، نفس می‌کشد، حرکت می‌کند و غذا می‌خورد. زمانیکه این مشخصه‌ها را به نمودار درختی‌مان اضافه می‌کنیم، چیزی شبیه به نمودار زیر بدست خواهیم آورد:



این مشخصه‌ها را می‌توان بعنوان فعالیت یا توابع درنظر گرفت- چیزهایی که یک شیء از آن دسته میتواند انجام دهد.

برای افزودن یک تابع به یک کلاس، از کلیدواژه‌ی def استفاده می‌کنیم. بنابراین کلاس Animals اینگونه خواهد بود:

---

[1] characteristics

[2] trait

```
>>> class Animals(Animate):
        def breathe(self):
                pass
        def move(self):
                pass
        def eat_food(self):
                pass
```

در خط اول این فهرست، کلاس را همانند قبل تعریف می‌کنیم ولی به جای اینکه در خط بعد از کلیدواژه‌ی pass استفاده کنیم، تابعی به نام breathe را تعریف کرده و یک پارامتر به آن می‌دهیم: self .

پارامتر self راهی است برای این که یک تابع در یک کلاس بتواند تابع دیگری در یک کلاس(و در کلاس والد) را صدا بزند. در ادامه، شاهد یکی از کاربردهای این پارامتر خواهیم بود.

در خط بعد، کلیدواژه‌ی pass به پایتون اعلام می‌کند که قصد نداریم اطلاعات بیشتری را درباره‌ی تابع breath ارائه کنیم زیرا الآن به درد ما نمی‌خورد. سپس توابع move و eat_food را اضافه می‌کنیم که الآن کاری انجام نمی‌دهند. کلاس‌ها را مجدداً ساخته و کد مناسبی را در توابع قرار می‌دهیم. این یک راه متداول برای توسعه برنامه‌ها می‌باشد. غالباً، برنامه‌نویس‌ها برای این که دریابند کلاس چه کاری باید انجام دهد، و قبل از اینکه وارد جزئیات توابع خاص شوند، با کمک توابعی که کاری انجام نمی‌دهند، کلاس‌هایی را ایجاد می‌کنند.

همچنین توابعی را به دو کلاس دیگر یعنی Mammals و Giraffes اضافه می‌کنیم. هر کلاس می-تواند از مشخصه‌های(توابع) والد خود استفاده کند. به این معنی که نیازی نیست کلاس پیچیده‌ای را بسازید: می‌توانید توابع خود ر در بالاترین والد قرار دهید یعنی همان جایی که مشخصه‌ها بکارگرفته می‌شوند. (این یک شیوه مناسب برای ساده‌تر و قابل‌درک‌تر کردن کلاس‌ها می‌باشد.)

```
>>> class Mammals(Animals):
        def feed_young_with_milk(self):
                pass
>>> class Giraffes(Mammals):
        def eat_leaves_from_trees(self):
                pass
```

چرا از کلاس‌ها وشیءها استفاده می‌کنیم؟

توابع را به کلاس‌هایمان اضافه کردیم ولی وقتی می‌توانیم توابـع نرمـالی ماننـد breathe, move, eat_food و... را بنویسیم پس به چه دلیل از کلاس‌ها و شیء‌ها استفاده می‌کنیم؟

برای پاسخ به این پرسش، از زرافه خود به نام Reginald استفاده می‌کنیم که قبلاً آن را بصـورت یک شیء از کلاس Giraffes ساختیم:

```
>>> reginald = Giraffes()
```

از آنجاییکه reginald یک شیء است، می‌توانیم توابعی که توسط این کلاس(کـلاس Giraffes)و کلاس‌های والد آن ارائه شده‌اند را فراخوانی(اجرا) کنیم. با استفاده از عملوند نقطه(.) و نام تابع، توابع را صدا می‌زنیم. برای اینکه به زرافه‌ی Reginald بگوییم حرکت کند یا غذا بخورد، می‌تـوانیم توابـع زیـر را صدا بزنیم:

```
>>> reginald = Giraffes()
>>> reginald.move()
>>> reginald.eat_leaves_from_trees()
```

فرض کنید Reginald یک دوست زرافه به نام Harld دارد. شیء Giraffes دیگری بـه نـام harold می‌سازیم:

```
>>> harold = Giraffes()
```

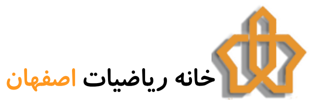
از آنجاییکه از شیء‌ها و کلاس‌ها استفاده می‌کنیم، زمانی که می‌خواهیم تابع move را اجرا کنـیم، دقیقاً می‌توانیم به پایتون بگوییم کـه دربـاره‌ی کـدام زرافـه صـحبت مـی‌کنـیم. بعنـوان مثـال اگـر مـی-خواهیم Harold حرکت کند ولی Reginald سر جـایش بمانـد، مـی‌تـوانیم تـابع move را بـا اسـتفاده از شیء harold صدا بزنیم:

```
>>> harold.move()
```

در این مورد فقط Harold حرکت خواهد کرد.

حال کلاس‌هایمان را اندکی تغییر داده تا این موضوع اندکی واضح‌تـر شـود. بـه جـای اسـتفاده از pass، دستور print را به هر تابع اضافه می‌کنیم:

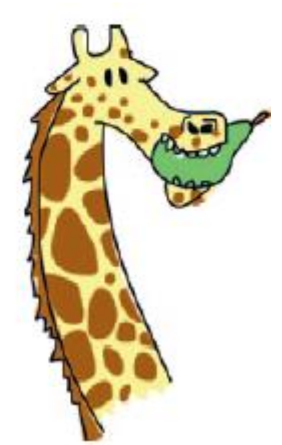
```
>>> class Animals(Animate):
        def breathe(self):
                print('breathing')
        def move(self):
                print('moving')
        def eat_food(self):
                print('eating food')
>>> class Mammals(Animals):
        def feed_young_with_milk(self):
```




```
                print('feeding young')
>>> class Giraffes(Mammals):
        def eat_leaves_from_trees(self):
                print('eating leaves')
```

حال، زمانی که شیءهای reginald و harold ر ساخته و توابع را برای آنها صدا می‌زنیم، می‌توانیم شاهد اتفاق زیر باشیم:

```
>>> reginald = Giraffes()
>>> harold = Giraffes()
>>> reginald.move()
moving
>>> harold.eat_leaves_from_trees()
eating leaves
```



در دو خط اول، متغیرهای reginald و harold را می‌سازیم که شیءهای کلاس Giraffes هستند. سپس تابع move را برای reginald صدا می‌زنیم و پایتون، moving را در خط زیر چاپ می‌کند. به همین ترتیب، تابع eat_leaves_from_trees را برای harold صدا می‌زنیم و پایتون، eating leaves را چاپ می‌کند. اگر این‌ها زرافه‌های واقعی بودند، نه شیءهای یک رایانه، یک زرافه راه رفته و زرافه دیگر برگ خواهد خورد.

شیءها و کلاس‌ها در تصاویر

نظرتان درباره استفاده از یک رویکرد گرافیکی‌تر برای شیءها و کلاس‌ها چیست؟

اجازه دهید به سراغ ماژول turtle برویم که در فصل ٤ با آن سرگرم بودیم. زمانیکه از turtle.Pen() استفاده می‌کنیم، پایتون یک شیء از کلاس Pen می‌سازد که توسط ماژول turtle تأمین شده است (مشابه شیءهای reginald و harold در بخش قبل). همانند اتفاقی که برای دو زرافه افتاد، می‌توانیم دو شیء turtle بسازیم (به نام Avery و Kate):

```
>>> import turtle
>>> avery = turtle.Pen()
>>> kate = turtle.Pen()
```

هر شیء turtle (avery و kate) یک عضو از کلاس Pen است.

این جا همان جایی است که قدرت شیءها رو به افزایش می‌رود. بعد از این که شیءهایturtle مان را ساختیم می‌توانیم توابعی را برای هر کدام از آنها صدا بزنیم و آنها نیز مستقلاً ترسیم می‌کنند. کد زیر را امتحان کنید.

```
>>> avery.forward(50)
>>> avery.right(90)
>>> avery.forward(20)
```

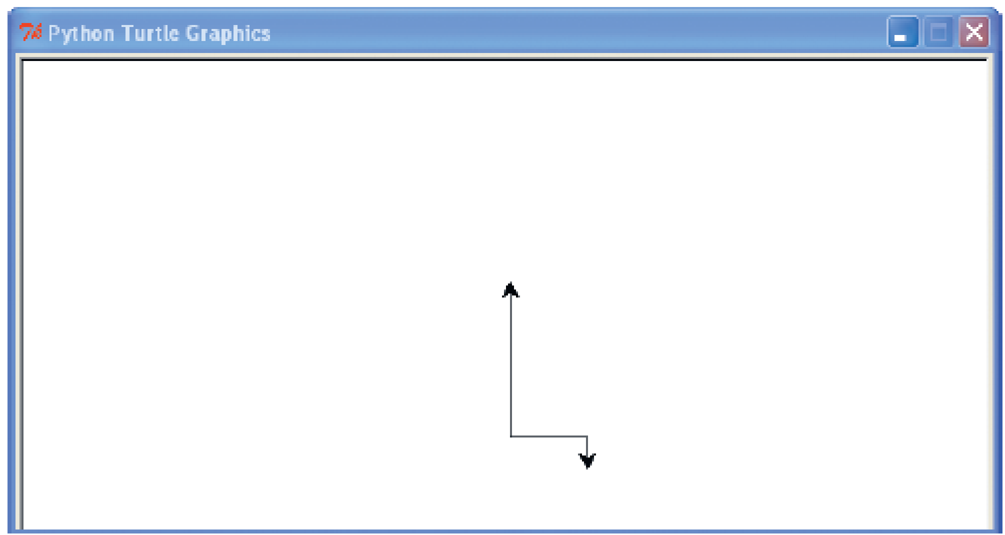
با این سری دستورات، بهAvery می‌گوییم که ۵۰ پیکسل به جلو حرکت کرده، ۹۰ درجه به سمت راست چرخیده و ۲۰ پیکسل به جلو برود بطوریکه رو به پایین توقف کند. به خطر داشته باشید که لاک‌پشت‌ها همواره رو به سمت راست حرکت را شروع می‌کنند.

حالا نوبت حرکت دادنKate است.

```
>>> kate.left(90)
>>> kate.forward(100)
```

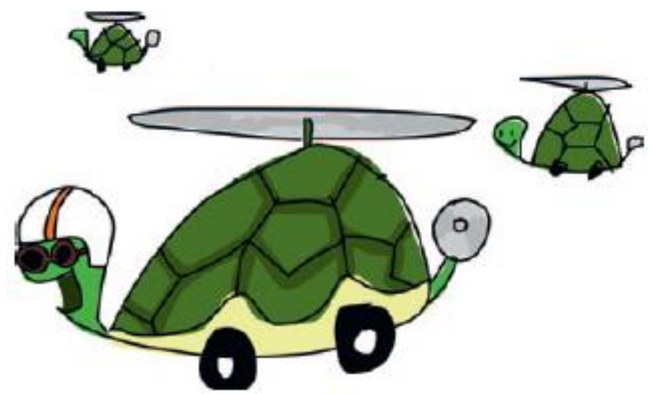
بهKate می‌گوییم که ۹۰درجه به چپ چرخیده و سپس ۱۰۰ پیکسل به جلو حرکت کند بطوریکه رو به بالا توقف نماید.

تا این‌جا، یک خط با فلش‌های در دو جهت مختلف داشتیم که سر هر کدام از آنها شیءturtle متفاوتی را نشان می‌دهد: Avery رو به پایین و Kate رو به بالا.

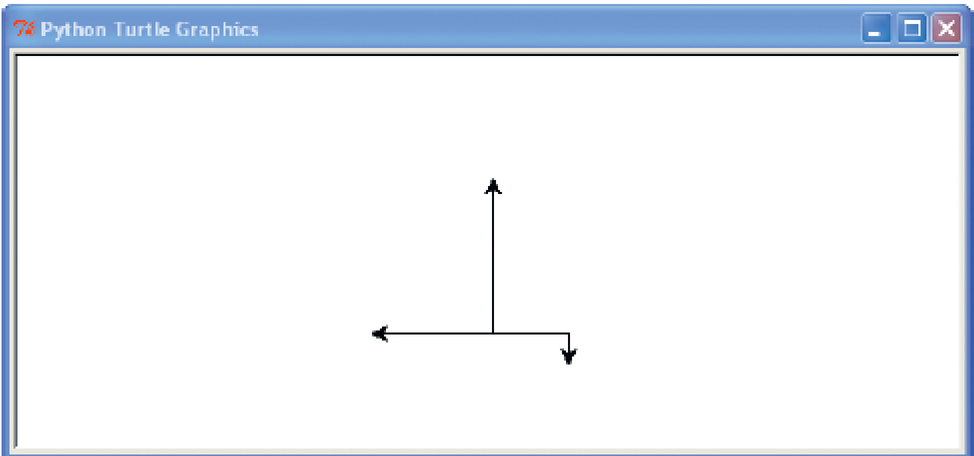


حالا لاک‌پشت(turtle) دیگری به نامJacob را اضافه کرده و آن را حرکت می‌دهیم بدون این که برایKate یاAvery مزاحمتی ایجاد کند.

```
>>> jacob = turtle.Pen()
>>> jacob.left(180)
>>> jacob.forward(80)
```



ابتدا یک شیءPen جدید به نامjacob را ایجاد کرده، سپس ۱۸۰ درجه آن را چرخانده و ۸۰ پیکسل به جلو می‌بریم. شکل ما با سه لاک‌پشت بصورت زیر خواهد بود:



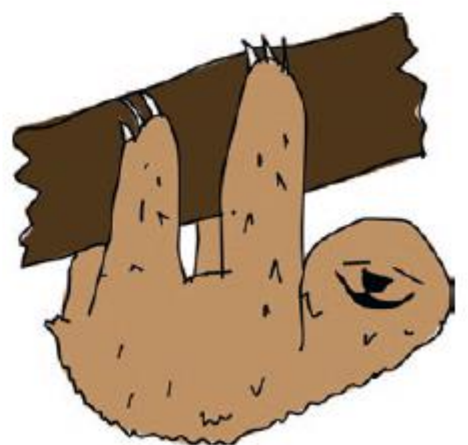
به خاطر داشته باشید هر بار که ()turtle.Pen را برای ساخت یکturtle صدا می‌زنیم، یک شیء جدید مستقل را اضافه می‌کنیم. هر شیء یک نمونه از کلاسPen است و می‌توانیم از همان توابع برای هر شیء استفاده کنیم ولی از آنجایی که از شیءها استفاده می‌کنیم، می‌توانیم هرturtle(لاک‌پشت) را بطور مستقل حرکت دهیم. همانند شیءهایgiraffe مستقل (Reginald وHarold)، Avery وKate وJacob نیز شیءهایturtle مستقلی هستند. اگر شیء جدیدی را با همان نام متغیر شیء‌ای که قبلاً ساخته‌ایم، ایجاد کنیم، شیء قدیمی لزوماً از بین نخواهد رفت. این کار را خودتان امتحان کنید: لاک-پشتKate دیگری ساخته و سعی کنید آن را حرکت دهید.

دیگر خصیصه‌های سودمند شیء‌ها و کلاس‌ها

کلاس‌ها و شیء‌ها کار گروه‌بندی توابع را آسان‌تر می‌کنند. همچنین زمانی که می‌خواهیم برنامه را در تکه‌های کوچکتری تصور کنیم، به ما کمک خواهند نمود.

بعنوان مثال، یک اپلیکیشن نرم‌افزاری بسیار بزرگ مثلاً یک پروازشگر واژه یا یک بازی رایانه‌ی ۳-بعدی را درنظر بگیرید. درک چنین برنامه‌های بزرگی تقریباً برای اکثر مردم غیرممکن است زیرا حجم کد بسیار زیاد است. ولی با شکستن این برنامه‌های غول‌آسا به تکه‌های کوچکتر و البته با آگاهی از زبان، درک هر تکه قابل درک خواهد بود.

هنگام نوشتن یک برنامه‌ی بزرگ، چند پاره کردن آن به شما اجازه می‌دهد تا کار را بین دیگر برنامه‌نویسان تقسیم کنید. پیچیده‌ترین برنامه‌هایی که از آنها استفاده می‌کنید(مثلاً مرورگر وب) توسط چندین نفر یا تیمی از افراد که در نقاط مختلف جهان و بطور همزمان روی بخش‌های مختلف کار می‌کنند، نوشته شده‌اند.

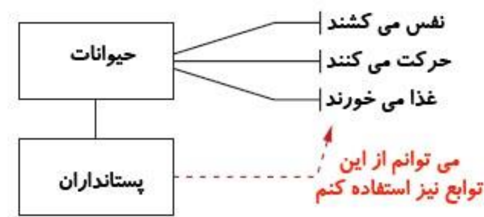


حال تصور کنید که می‌خواهیم برخی کلاس‌هایی که در این فصل ساخته‌ایم(Animals, Mammals, Giraffes) را بسط دهیم، ولی چون باید حجم زیادی از کار را انجام دهید، خواهان کمک دوستان خود هستیم. می‌توانیم کار نوشتن کد را به گونه‌ای تقسیم کنیم تا یک نفر روی کلاس Animals ، یک نفر دیگر روی کلاس Mammals و فرد دیگری روی کلاش Griaffes کار کند.

توابع موروژی[1]

کسانی که دقت و توجه کافی را به کار برده‌اند درک می‌کنند که هر کسی کار روی کلاس Giraffes را تمام کند خوش‌شانس است زیرا می‌توان از توابع خلق شده توسط کسانی که روی کلاس Animals و کلاس Mammals کار می‌کنند درکلاس Giraffes نیز استفاده نمود. کلاس Giraffes توابع را از کلاس Mammals و به همین ترتیب از کلاس Animals به اثر می‌برد. به بیان دیگر زمانی که یک شیء زرافه را ایجاد می‌کنیم، می‌توانیم از توابع تعریف شده در کلاس Giraffes و همچنین توابع تعریف شده در کلاس‌های Animals و Mammals استفاده کنیم. و به همین ترتیب، اگر یک شیء mammal (پستاندار) بسازیم می‌توانیم از توابع تعریف شده در کلاس Mammals و کلاس والد آن یعنی Animals نیز استفاده کنیم.

[1] inherited



مجدداً نگاهی به رابطه‌ی میان کلاس‌های Animals, Mammals و Giraffes می‌اندازیم. کلاس Animals والد کلاس Mammals است. و Mammals والد Giraffes می‌باشد.

حتی با فرض این که Reginald یک شیء از Giraffes است، بازهم می‌توانیم تابع move را که در کلاس Animal ایجاد کرده‌ایم، صدا بزنیم زیرا توابع تعریف شده در هر کلاس والد در دسترس کلاس‌های فرزند نیز می‌باشند:

```
>>> reginald = Giraffes()
>>> reginald.move()
moving
```

درواقع، می‌توانیم از شیء Reginald تمامی توابعی را که در هر دو کلاس Animals و Mammals تعریف کردیم، صدا بزنیم زیرا این توابع موروثی هستند:

```
>>> reginald = Giraffes()
>>> reginald.breathe()
breathing
>>> reginald.eat_food()
eating food
>>> reginald.feed_young_with_milk()
feeding young
```

توابعی که توابع دیگر را صدا می‌زنند

زمانی که توابع را روی یک شیء صدا می‌زنیم، از نام متغیر شیء استفاده می‌کنیم.بعنوان مثال، در اینجا نشان می‌دهیم که giraffe چگونه تابع move را روی Reginald صدا می‌زند:

```
>>> reginald.move()
```

برای این که تابعی در کلس Giraffes داشته باشیم که تابع move را صدا بزند، از پارامتر self استفاده می‌کنیم. پارامتر self راهی است تا یک تابع در یک کلاس بتواند تابع دیگری را صدا بزند. بعنوان مثال، فرض کنید تابعی به نام find_food را به کلاس Giraffes اضافه می‌کنیم:

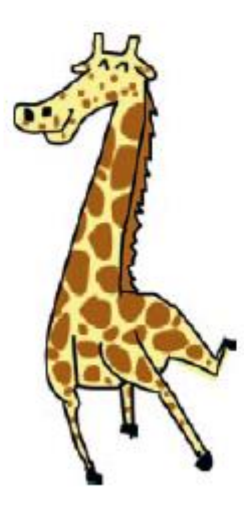
```
>>> class Giraffes(Mammals):
        def find_food(self):
            self.move()
            print("I've found food!")
```

```
self.eat_food()
```

در این‌جا تابعی را می‌سازیم که ترکیبی از دو تابع دیگر است و کاری متداول در برنامه‌نویسی می‌باشد. غالباً تابعی را می‌نویسیم که کار مفیدی انجام می‌دهد و بعداً می‌توانیم از آن درون تابع دیگری استفاده کنیم (این کار در فصل ۱۳ انجام خواهد شد، جایی که توابع پیچیده‌تری را برای ساختن یک بازی خواهیم نوشت).

اجازه دهید از self برای اضافه کردن چند تابع به کلاس Giraffes استفاده کنیم:

```
>>> class Giraffes(Mammals):
def find_food(self):
        self.move()
        print("I've found food!")
        self.eat_food()
def eat_leaves_from_trees(self):
        self.eat_food()
def dance_a_jig(self):
        self.move()
        self.move()
        self.move()
        self.move()
```



از توابع eat_food و move کلاس والد Animals برای تعریف eat_leaves_from_trees و dance_a_jig برای کلاس Giraffes استفاده می‌کنیم زیرا این‌ها توابع موروثی می‌باشند. زمانیکه شیءهای این کلاس‌ها را ایجاد می‌کنیم، با افزودن توابعی که به این ترتیب توابع دیگر را صدا می‌زنند، می‌توانیم تابعی را صدا بزنیم که بیش از یک کار را انجام می‌دهد. می‌توان مشاهده نمود که وقتی تابع dance_a_jig را صدا می‌زنیم چه اتفاقی می‌افتد– زرافه ٤ بار به جلو حرکت می‌کند(یعنی متن "moving" چهار بار چاپ می‌شود).

```
>>> reginald = Giraffes()
>>> reginald.dance_a_jig()
moving
moving
moving
moving
```

مقداردهی[1] یک شیء

برخی مواقع در هنگام ایجاد یک شیء، می‌خواهیم برخی مقادیر (که ویژگی‌ها[2] نامیده می‌شوند) را برای مصارف بعدی تعیین کنیم. هنگام مقداردهی یک شیء، آن را برای استفاده آماده می‌کنیم.

بعنوان مثال، فرض کنید می‌خواهیم هنگام ساخته شدن شیء‌های زرافه(giraffe) تعداد نقاط را روی آنها تعیین کنیم– یعنی زمانی که مقداردهی شده‌اند. برای این کار، یک تابع__init__ را می‌سازیم(توجه داشته باشید که دو کاراکتر زیرخط‌دار در هر طرف وجود دارند و در مجموعه چهار کاراکتر داریم).

این یک نوع خاص از تابع در کلاس‌های پایتون است و بایستی این نام را داشته باشد. تابع init راهی است برای تعیین ویژگی‌ها برای یک شیء در زمانی است که شیء ابتدا خلق می‌شود و زمانی که شیء جدیدی را می‌سازیم، پایتون بطور خودکار این تابع را صدا می‌زند. در اینجا نشان می‌دهیم که چگونه از آن استفاده می‌کنیم:

```
>>> class Giraffes:
        def __init__(self, spots):
                self.giraffe_spots = spots
```

ابتدا، با کد :def __init__(self, spots) تابع init را با دو پارامتر self و spot تعریف می‌کنیم. دقیقاً همانند توابع دیگری که در کلاس تعریف کرده‌ایم، در تابع init نیز بایستی self بعنوان پارامتر اول درنظر گرفته شود. سپس با استفاده از پارامتر self با کد self.giraffe_spots = spots، پارامتر spots را با یک متغیر شیء (ویژگی آن)به نام giraffe_spots مقدردهی می‌کنیم. این خط کد به این معنی است که«مقدار پارامتر spots را گرفته و بری مصرف بعدی ذخیره کن (با استفاده از متغیر شیءgiraffe_spots)». چون با استفاده از پارامتر self فقط یک تابع در یک کلاس می‌تواند تابع دیگری را صدا بزند، متغیرهای کلاس نیز با استفاده از self دردسترس خواهند بود.

سپس اگر دو شیءgiraffe جدید را ساخته(Ozwald and Gertrude) و تعداد نقاط آنها را نمایش دهیم، می‌توانیم تابع مقداردهی زیر را داشته باشیم:

```
>>> ozwald = Giraffes(100)
>>> gertrude = Giraffes(150)
>>> print(ozwald.giraffe_spots)
100
>>> print(gertrude.giraffe_spots)
```

---
[1] Initializing
[2] Properties

150

ابتدا، یک نمونه از کلاس Giraffes را با استفاده از مقدار پارامتر 100 ایجاد می‌کنیم. این کار هم-اثر با فراخوانی تابع __init__ و استفاده از 100 برای مقدار پرامتر spots است. سپس، نمونه‌ی دیگری از کلاس Giraffes، ولی این بار با مقدار 150 می‌سازیم. درنهایت، متغیر شیء giraffe_spots را برای هریک از شیءهای giraffe چاپ کرده و خواهیم دید که نتایج 100 و 150 می‌باشند.پس جواب می‌دهد!

بخاطر داشته باشید زمانی که یک شیء از یک کلاس را می‌سازیم، مثلاً ozwald، با استفاده از عملگر dot (نقطه) و نام متغیر یا تابعی که می‌خواهیم از آن استفاده کنیم (بعنوان مثال ozwald.giraffe_spots)، می‌توانیم به متغیرها یا توابع آن اشاره نماییم. ولی زمانی که توابعی را درون یک شیء ایجاد می‌کنیم، با استفاده از پارامتر self(self.giraffe_spots) به همان متغیرها (و توابع دیگر) اشاره می‌کنیم.

آنچه آموختید

در این فصل، از کلاس‌ها برای ایجاد دسته‌هایی از اشیاء(چیزها) و ساخت اشیاء(نمونه‌های) این کلاس‌ها استفاده کردیم. آموختید که چگونه فرزند یک کلاس از توابع والدش ارث می‌برد و حتی درصورتیکه دو شیء از یک کلاس باشند، لزوماً همزاد نیستند.

بعنوان مثال، یگ شیء زرافه می‌تواند تعداد نقاط خودش را داشته باشد. آموختید که چگونه می-توانید توابع را روی یک شیء صدا زده(یا اجرا کرده) و این که متغیرهای شیء راهی برای ذخیره کردن مقادیر در این شیءها می‌باشند. درنهایت، برای اشاره به متغیرها و توابع دیگر، از پارامتر self در توابع استفاده کردیم. این مفاهیم، مفاهیم اساسی در پایتون بوده و در ادامه این کتاب، بارها و بارها با آنها مواجه خواهید شد.

چیستان‌های برنامه‌نویسی

برخی از ایده‌های مطرح شده در این کتاب، در طی بکارگیری ملموس‌تر خواهند شد. مثال‌های زیر را انجام داده و پاسخ‌های خود را در *http://python-for-kids.com/.* مشاهده کنید.

۱) گام برداشتن زرافه

تابعی را به کلاس Giraffes اضافه کرده تا پای چپ و راست زرافه به عقب و جلو حرکت کند. تابع حرکت دادن پای چپ بصورت زیر است:

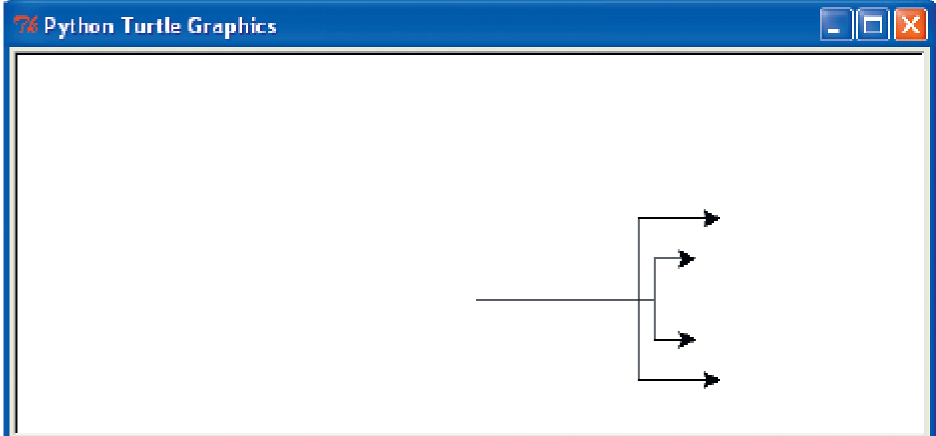
```
>>> def left_Foot_Forward(self):
        print('left foot forward')
```

سپس تابعی به نام dance را برای آموزش رقص به Reginald بسازید(این تابع، چهار تـابع foot را که قبلاً ساخته‌اید، صدا می‌زند). نتیجه‌ی فراخوان این تابع جدید یک رقص ساده خواهد بود:

```
>>> reginald = Giraffes()
>>> reginald.dance()
left foot forward
left foot back
right foot forward
right foot back
left foot back
right foot back
right foot forward
left foot forward
```

۲) پنجه‌ی لاک‌پشت

با استفاده از چهار شیء turtle pen، تصویر زیـر را از پنجـه‌ی کنـاری ایجـاد کنیـد(طول دقیـق خطوط اهمیتی ندارد). به خاطر داشته باشید که ابتدا بایستی ماژول turtle را وارد کنید.



فصل ۹

توابع ذاتی پایتون

پایتون یک مجموعه کافی از ابزارهای برنامه‌نویسی از جمله تعداد زیادی از توابع و ماژول‌هایی را در اختیار دارد که آماده‌ی استفاده شما می‌باشند. همانند یک چکش قابل‌اعتماد یا یک آچار دوچرخه، این ابزارهای ذاتی- در واقع تکه‌هایی از کد- کار برنامه‌نویسی را بسیار آسان‌تر می‌کنند.

همانطوریکه در فصل۷ آموختید، قبل از استفاده از ماژول‌ها بایستی آنها را وارد کرد. ولی نیازی نیست که همان ابتدا توابع ذاتی پایتون را وارد کنیم؛ با بالا آمدن پایتون، تمامی آنها در دسترس خواهند بود. در این فصل به برخی از توابع سودمند ذاتی پایتون نگاهی خواهیم انداخت و سپس روی یکی از آنها تمرکز می‌کنیم که به شما اجازه می‌دهد فایل‌ها را برای خواندن و نوشتن، باز کنید.

استفاده از توابع ذاتی

به ۱۲تابع ذاتی می‌پردازیم که بطور معمول برنامه‌نویسان از آن‌ها استفاده می‌کنند. همچنین شرح خواهم داد که چه کاری انجام می‌دهند و چگونه بایستی از آنها استفاده کرد، همچنین مثال‌های ارائه می-گردند که نشان می‌دهند چگونه می‌توانند به برنامه‌های شما کمک کنند.

تابع ABS

تابع abs قدر مطلق[۱] یک عدد را برمی‌گرداند که همان مقدار یک عدد بدون علامت آن است. بعنوان مثال، قدرمطلق 10 همان10 است و قدرمطلق 10-، 10 می‌باشد.

---

۱ Absolute value

برای استفاده از تابع abs، آن را با یک عدد یک متغیر بعنوان پارامتر آن صدا می‌زنیم:

```
>>> print(abs(10))
10
>>> print(abs(-10))
10
```

می‌توان از تابع abs برای کار دیگری مثلاً محاسبه‌ی مقدار دقیق جابجایی یک کاراکتر در یک بازی، صرفنظر از جهت حرکت این کاراکتر، استفاده نمود. بعنوان مثال، کاراکتر ۳ گام به راست(+۳) و سپس ۱۰ گام به چپ(-۱۰) برمی‌دارد. اگر جهت اهمیتی نداشته باشد(مثبت یا منفی)، آنگاه قدرمطلق این اعداد،۳ و ۱۰ خواهد بود. می‌توان از این روند در بازی تخته نیز استفاده کرد، یعنی جایی که دو تاس را ریخته و سپس کاراکتر خود را براساس مجموع تاس‌ها، به اندازه‌ی ماکسیمم گام در هر جهت حرکت می‌دهید. حال، اگر تعداد گام‌ها را در یک متغیر ذخیره کنیم می‌توانیم حرکت کردن/نکردن کاراکتر را با کد زیر تعیین کنیم. شاید زمانیکه بازیکن تصمیم به حرکت می‌گیرد، بخواهیم اطلاعاتی را نمایش دهیم(در این مورد فقط «کاراکتر در حال حرکت است» را نمایش می‌دهیم):

```
>>> steps = -3
>>> if abs(steps) > 0:
        print('Character is moving')
```

اگر از abs استفاده نکرده باشیم، دستور if بصورت زیر خواهد بود:

```
>>> steps = -3
>>> if steps < 0 or steps > 0:
        print('Character is moving')
```

همانطوریکه مشاهده می‌کنید، استفاده از abs باعث می‌شود تا دستور if اندکی کوتاهتر و درک آن ساده‌تر شود.

تابع bool

نام Bool مختصر شده‌ی Boolean است، واژه‌ای که برنامه‌نویسان از آن برای توصیف نوع داده-ای استفاده میکنند که می‌تواند یک از دو مقدار احتمالی: (درست)true یا (نادرست)false را داشته باشد.
تابع bool یک پارامتر را گرفته و با توجه به مقدار آن، True یا False را برمی‌گرداند. در هنگام استفاده از Bool برای اعداد، 0،False را برمی‌گرداند، و هر عدد دیگری True را برمی‌گرداند. در اینجا مورد استفاده‌ی bool با اعداد مختلف نشان داده شده است:

```
>>> print(bool(0))
False
```

```
>>> print(bool(1))
True
>>> print(bool(1123.23))
True
>>> print(bool(-500))
True
```

زمانی که از bool برای مقادیر دیگر استفاده می‌کنید، مثلاً رشته‌ها[1]، اگر هیچ مقداری بـرای رشـته (به بیان دیگر، کلیدواژه‌ی None یا یک رشته‌ی خـالی) وجـود نداشـته باشـد، False و درغیراینصـورت True را برمی‌گرداند:

```
>>> print(bool(None))
False
>>> print(bool('a'))
True
>>> print(bool(' '))
True
>>> print(bool('What do you call a pig doing karate? Pork Chop!'))
True
```

تابع bool برای لیست‌ها، چندتایی‌ها و نگاشت‌هایی که حاوی هیچ مقداری نیستند نیـز False را برگردانده و برای مورد زیر True را برمی‌گرداند:

```
>>> my_silly_list = []
>>> print(bool(my_silly_list))
False
>>> my_silly_list = ['s', 'i', 'l', 'l', 'y']
>>> print(bool(my_silly_list))
True
```

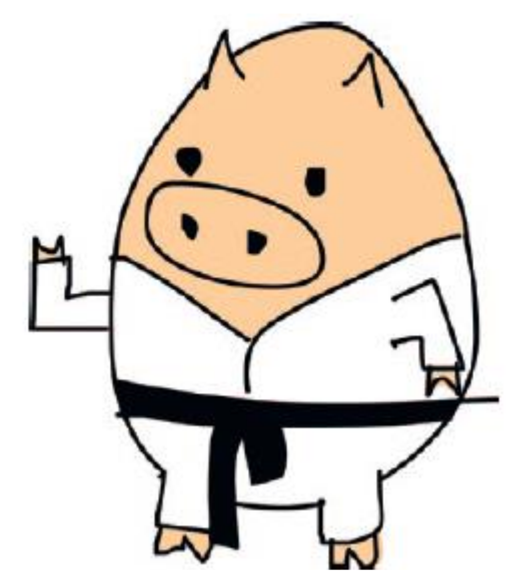
همچنین زمانی که می‌خواهید درمورد تعیین یک مقدار تصـمیم‌گیـری کنیـد، مـی‌توانیـد از bool استفاده نمایید. بعنوان مثال، اگر از دیگران بخواهیم از برنامه‌ی ما برای وارد کرن سال تولدشـان اسـتفاده کنند، می‌توانیم برای آزمودن مقداری که وارد کرده‌اند، از bool. در دستور if استفاده کنیم:

```
>>> year = input('Year of birth: ')
Year of birth:
>>> if not bool(year.rstrip()):
        print('You need to enter a value for your year of birth')
You need to enter a value for your year of birth
```

---

[1] string

در اولین خط این مثال از input برای ذخیره‌ی چیزی استفاده شده است که توسط صفحه کلید بعنوان متغیر year وارد شده است. با زدن کلید ENTER در خط بعد (بدون اینکه چیزی تایپ شود) مقدار کلید ENTER در متغیر ذخیره می‌گردد. (از sys.stdin.readline() –فصل ۷– استفاده می‌کنیم که راه دیگری برای انجام این کار است).



در خط بعد، دستور if مقدار Boolean متغیر را بعد از استفاده از تابع rstrip بررسی می‌کند (تمامی فاصله‌ها[1] و کاراکترهای ENTER را از انتهای رشته حذف می‌کند). چون در این مثال، کاربر چیزی را وارد نکرده است، تابع bool، false را برمی‌گرداند. چون در دستور if از کلیدواژه‌ی not استفاده شده است، به این معنی است که «اگر تابع true را برنگرداند، این کار را انجام بده»، درنتیجه کد در خط بعدی، این را چاپ می‌کند « You need to enter a value for your year of birth» (باید مقداری را برای سال تولدتان وارد کنید)

تابع DIR

تابع dir (مخفف Directory) اطلاعاتی را درباره‌ی هر مقداری برمی‌گرداند. اساساً، توابعی که با آن مقدار قابل استفاده هستند را به ترتیب حروف الفبا به شما نشان می‌دهد.

بعنوان مثال، برای نمایش توابعی که برای یک مقدار لیست دردسترس هستند، کد زیر را وارد کنید:

```
>>> dir(['a', 'short', 'list'])
['__add__', '__class__', '__contains__', '__delattr__',
'__delitem__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__',
'__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
'__sizeof__', '__str__', '__subclasshook__', 'append', 'count',
'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

[1] space

تابع dir تقریباً برای هر چیزی جواب می‌دهد از جمله، رشته‌ها، اعداد، توابع، ماژول‌ها، شیءها و کلاس‌ها. ولی برخی اوقات اطلاعاتی که برمی‌گرداند چندان سودمند نیستند. بعنوان مثال، اگر dir را برای عدد1 صدا بزنید، تعداد توابع ویژه (توابعی که با زیرخط شروع شده و خاتمه می‌یابند) مـورد اسـتفاده‌ی خود پایتون را نمایش خواهد داد که به درد نمی‌خورد(غالباً اکثر آنها را کنار می‌گذاریم).

```
>>> dir(1)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__',
'__class__', '__delattr__', '__divmod__', '__doc__', '__eq__',
'__float__', '__floor__', '__floordiv__', '__format__', '__ge__',
'__getattribute__', '__getnewargs__', '__gt__', '__hash__',
'__index__', '__init__', '__int__', '__invert__', '__le__',
'__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__',
'__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__',
'__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__',
'__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__',
'__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__',
'__sub__', '__subclasshook__', '__truediv__', '__trunc__',
'__xor__', 'bit_length', 'conjugate', 'denominator', 'imag',
'numerator', 'real']
```

زمانی که یک متغیر دراختیار دارید و به سرعت می‌خواهید بدانید چه کاری مـی‌توانیـد روی آن انجام دهید، می‌توانید از تابع dir استفاده کنید. بعنوان مثال، با استفاده از متغیر popcorn حاوی یک مقـدار رشته، تابع dir را اجرا کرده و لیستی از توابع تولید شده توسط کلاس string را مشاهده نماییـد (تمـامی رشته‌ها، اعضای کلاس String هستند):

```
>>> popcorn = 'I love popcorn!'
>>> dir(popcorn)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__',
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
'__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index',
'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle',
```

```
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

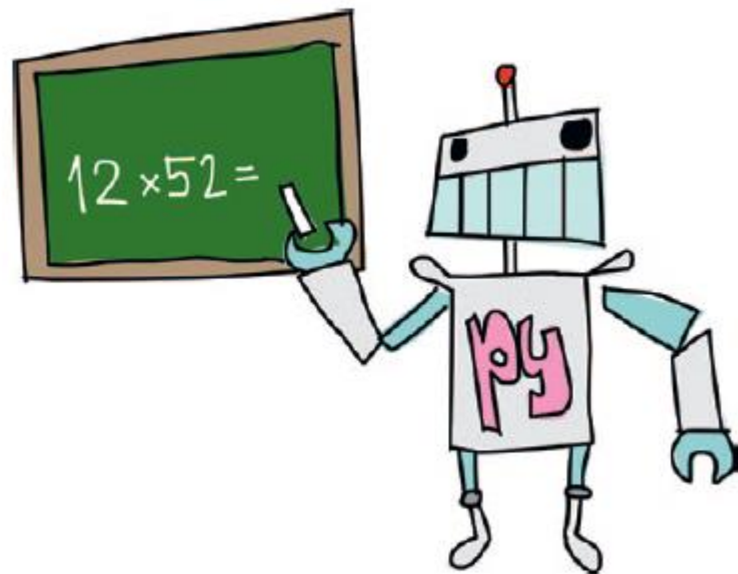
در این جا، می‌توانید از help برای دریافت توضیح مختصری درباره هر یک از توابع موجود در لیست استفاده کنید. در اینجا مثالی از اجرای help برای تابع upper وجود دارد:

```
>>> help(popcorn.upper)
Help on built-in function upper:
upper(...)
        S.upper() -> str ·
        Return a copy of S converted to uppercase.
```

اطلاعات نمایش داده شده، کمی سردرگم کننده خواهد بود بنابراین نگاهی دقیق‌تر می‌اندازیم.(...) به این معنی است که upper یک تابع ذاتی در کلاس string است و دراین مورد، هیچ پارامتری ندارد. (->) در خط بعد به این معنی است که این تابع یک رشته (str) را برمی‌گرداند. خط آخر نیز شرح مختصری از عملکرد تابع را ارائه می‌نماید.

تابع EVAL

تابع eval (مختصر evaluate) یک رشته را بعنوان یک پارامتر درنظر گرفته و آن را بصورت یک عبارت پایتون اجرا می‌کند. بعنوان مثال،('eval('print("wow") دستور ("print("wow را اجرا می‌کند.



تابع eval فقط با عبارات[1] سادهای مانند نمونه زیر کار می‌کند:

```
>>> eval('10*5')
50
```

عبارات چندخطی (مثلاً دستورات if) معمولاً ارزیابی نمی‌شوند:

```
>>> eval('''if True:
        print("this won't work at all")''')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1
    if True: print("this won't work at all')
     ^
SyntaxError: invalid syntax
```

غالباً از تابع eval برای تبدیل ورودی کاربر به عبارات پایتون استفاده می‌شود. بعنوان مثال، می‌توانید برنامه‌ی ساده‌ی ماشین‌حسابی را بنویسید که روابط وارد شده به پایتون را خوانده و سپس جواب‌ها را محاسبه(ارزیابی) کند.

چون ورودی کاربر بعنوان یک رشته خوانده می‌شود، پایتون بایستی قبل از انجام هرگونه محاسبه‌ی آن را به عدد و عملگر تبدیل نماید. تابع eval این تبدیل را بسیار آسان می‌کند:

```
>>> your_calculation = input('Enter a calculation: ')
Enter a calculation: 12*52
>>> eval(your_calculation)
624
```

در این مثال، از input برای خواندن چیزی که کاربر در متغیر your_calculation وارد کرده است، استفاده می‌کنیم. در خط بعد، عبارت 12*52 را وارد می‌کنیم (شاید سن شما در تعداد هفته‌های یک سال ضرب شده باشد). از eval برای اجرای این محاسبه استفاده کرده و نتیجه در خط آخر چاپ می‌شود.

تابع EXEC

تابع exec مانند eval است با این تفاوت که می‌توانید از آن برای اجرای برنامه‌های پیچیده‌تری استفاده نمایید. تفاوت میان این دو در این است که eval یک مقدار را برمی‌گرداند (چیزی که می‌توانید در یک متغیر ذخیره نمایید) درحالیکه exec قادر به انجام آن نیست. بعوان مثال:

```
>>> my_small_program = '''print('ham')
print('sandwich')'''
```

[1] expression

```
>>> exec(my_small_program)
ham
sandwich
```

در دو خط اول، متغیری را با یک رشته‌ی چندخطی حاوی دو دستور print ایجاد کرده و سپس از exec برای اجرای رشته استفاده می‌کنیم.

می‌توان از exec برای اجرای برنامه‌های کوچکی که برنامه‌ی پایتون از فایل‌ها خوانده است، استفاده کرد– درواقع، برنامه‌های درون برنامه‌ها! این کاردر هنگام نوشتن اپلیکیشن‌های طولانی و پیچیده به درد می‌خورد.

بعنوان مثال می‌توانید بازی دوئل ربات‌ها (Dueling Robots)را بسازید، جایی که دو ربات در صفحه حرکت کرده و سعی می‌کنند به هم حمله کنند. بازیکنان با برنامه‌های کوچک پایتون، به ربات-هایشان دستور می‌دهند بازی Dueling Robots این اسکریپت‌ها را خوانده از exec برای اجرا استفاده می‌کنیم.

تابع FLOAT

تابع float یک رشته یا یک عدد را به یک عدد ممیز شناور[1] تبدیل می‌کند که یک عدد اعشاری است (عدد حقیقی[2] نیز نامیده می‌شود). بعنوان مثال، عدد 10 یک عدد صحیح(کامل)[3] است (که عدد صحیح[4] نیز نامیده می‌شود) ولی 10.0، 10.1 یا 10.253 اعداد ممیزشناور هستند (عدد شناور[5] نیز نامیده می‌شود)

بسادگی با فراخوانی float، می‌توانید یک رشته را به عدد شناور تبدیل کنید:

```
>>> float('12')
12.0
```

می‌توانید از یک اعشار در یک رشته استفاده کنید:

---

[1] Floating point
[2] real
[3] integer
[4] Whole Number
[5] float

```
>>> float('123.456789')
123.456789
```

می‌توانید از float برای تبدیل مقادیر وارد شده به برنامه به اعداد مناسب استفاده کنید که برای مقایسه‌ی مقادیر مختلفی که وارد شده‌اند، بسیار سودمند خواهد بود. بعنوان مثال، برای این که ببینیم سن یک نفر بزرگتر از یک عدد معین است یا نه، می‌توانیم کار زیر را انجام دهیم:

```
>>> your_age = input('Enter your age: ')
Enter your age: 20
>>> age = float(your_age)
>>> if age > 13:
        print('You are %s years too old' % (age - 13))
You are 7.0 years too old
```

تابع INT

تابع int یک رشته یا عدد را به یک عدد صحیح تبدیل می‌کند که، به این معنی که هر چیزی بعد از ممیز حذف خواهد شد. بعنوان مثال، در اینجا نشان داده می‌شود که چگونه یک عدد ممیزشناور را به یک عددصحیح ساده تبدیل کنیم:

```
>>> int(123.456)
123
```

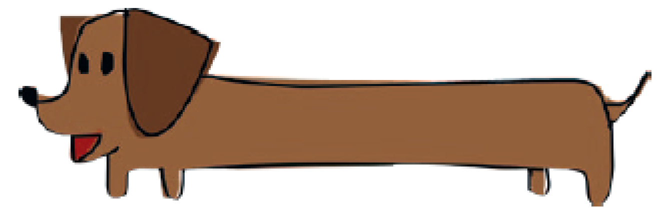
این مثال یک رشته را به یک عددصحیح تبدیل می‌کند:

```
>>> int('123')
123
```

ولی اگر سعی کنید یک رشته حاوی یک عدد ممیزشناور را به یک عدد صحیح تبدیل کنید، یک پیام خطا نمایش داده خواهد شد. بعنوان مثال، در این‌جا سعی داریم با استفاده از تابع Int، یک رشته حاوی یک عدد ممیزشناور تبدیل کنیم:

```
>>> int('123.456')
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
      int('123.456')
ValueError: invalid literal for int() with base 10: '123.456'
```

همانطوریکه مشاهده می‌کنید، نتیجه یک پیام ValueError می‌باشد.



تابع LEN

تابع len طول طول یک شیء یا در مورد یک رشته، تعداد کاراکترها در یک رشته را را برمی‌گرداند. بعنوان مثال، برای بدست آوردن طول رشته‌ی this is a test، کار زیر را انجام می‌دهیم:

```
>>> len('this is a test string')
21
```

اگر از len در یک لیست یا چندتایی استفاده شود، تعداد آیتم‌های آن لیست یا چندتایی را برمی-گرداند.

```
>>> creature_list = ['unicorn', 'cyclops', 'fairy', 'elf', 'dragon',
                'troll']
>>> print(len(creature_list))
6
```

اگر از len در یک نگاشت استفاده گردد، تعداد آیتم‌های نگاشت را برمی‌گرداند:

```
>>> enemies_map = {'Batman' : 'Joker',
                'Superman' : 'Lex Luthor',
                'Spiderman' : 'Green Goblin'}
>>> print(len(enemies_map))
3
```

بویژه زمانی که با حلقه‌ها کار می‌کنید، تابع len سودمند خواهد بود.بعنوان مثال، می‌توان از آن برای نمایش موقعیت اندیس عناصر در فهرستی مانند زیر استفاده کنید:

```
>>> fruit = ['apple', 'banana', 'clementine', 'dragon fruit']
❶ >>> length = len(fruit)
❷ >>> for x in range(0, length):
❸         print('the fruit at index %s is %s' % (x, fruit[x]))
    the fruit at index 0 is apple
    the fruit at index 1 is banana
    the fruit at index 2 is clementine
    the fruit at index 3 is dragon fruit
```

در اینجا، طول لیست در متغیر length در ❶ ذخیره گردیده و سپس در❷ از این متغیر در تابع range برای خلق حلقه استفاده می‌کنیم. در ❸ زمانی که هر آیتم از لیست را چک کردیم، پیامی را چاپ می‌کنیم که موقعیت و مقدار اندیس آیتم را نشان می‌دهد. همچنین درصورتیکه یک لیست از

رشته‌ها را در اختیار داشته و بخواهیم آیتم دوم یا سوم لیست را چاپ کنیم باید از تابع len نیز استفاده نماییم.

توابع MAX و MIN

تابع max بزرگترین آیتم در یک لیست،چندتایی یا رشته را برمی‌گرداند. بعنوان مثال، در اینجا نشان داده شده است که چگونه از آن برای یک لیست از اعداد استفاده کنیم:

```
>>> numbers = [5, 4, 10, 30, 22]
>>> print(max(numbers))
30
```

در مورد یک رشته با کاراکترهایی که با کاما یا فاصله از هم جدا شده‌اند نیز صادق است:

```
>>> strings = 's,t,r,i,n,g,S,T,R,I,N,G'
>>> print(max(strings))
t
```

همانطوریکه این مثال نشان می‌دهد، حروف به ترتیب الفبا مرتب شده‌اند و حروف کوچک بعد از حروف بزرگ قرار می‌گیرند بنابراین t بزرگتر از T است.

ولی لزومی ندارد از لیست، چندتایی یا رشته‌ها استفاده کنیم. همچنین می‌توانید مستقیماً تابع max را صدا زده و آیتم‌هایی که می‌خواهید مقایسه کنید را بعنوان پارامتر، در پرانتز قرار دهید:

```
>>> print(max(10, 300, 450, 50, 90))
450
```

تابع min نیز همانند تابع max عمل می‌کند با این تفاوت که کوچکترین آیتم در لیست، چندتایی یا رشته را برمی‌گرداند. در اینجا مثالی از یک لیست از اعداد وجود دارد که در آنجا از min به جای max استفاده شده است:

```
>>> numbers = [5, 4, 10, 30, 22]
>>> print(min(numbers))
4
```

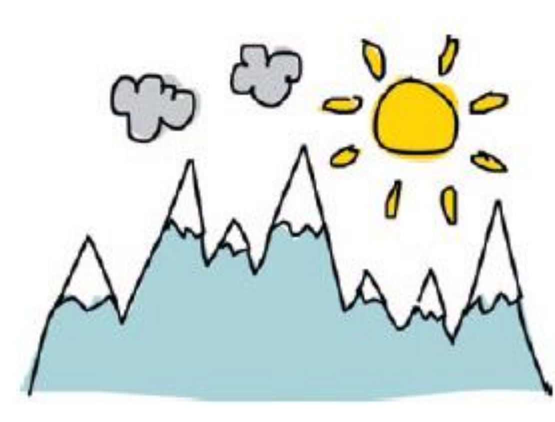
فرض کنید بازی حدس‌بزن را در یک تیم چهار نفره بازی می‌کنید و هر یک از افراد بایستی عددی را کوچکتر از عدد مدنظر شما حدس بزنند. اگر حدس تمامی بازیکنان بزرگتر از عدد موردنظر شما باشد، تمامی بازیکنان بازنده هستند ولی اگر حدس تمامی آنها کمتر از عدد موردنظرتان باشد، برنده

خواهند بود. برای اینکه سریعاً بفهمیم اعداد حدس زده شده، کمتر از عدد موردنظر هستند می‌توانیم از max استفاده کنیم:

```
>>> guess_this_number = 61
>>> player_guesses = [12, 15, 70, 45]
>>> if max(player_guesses) > guess_this_number:
        print('Boom! You all lose')
else:
        print('You win')
Boom! You all lose
```

در این مثال، عددی که بایستی حدس زده شود را با استفاده از متغایر guess this number ذخیره می‌کنیم. حدس‌های اعضای تیم در player_guesses list ذخیره شده‌اند. دستور if حدس ماکسیمم را با عدد guess_this_number مقایسه کرده و اگر حدس بازیکنن بیشتر از عدد موردنظر باشد، "Boom! You all lose." را چاپ می‌کنیم.

تابع RANGE



اساساً از تابع range (همانطوریکه قبلاًدیدیم) در حلقه‌های for برای گردش در یک بخش از کد به دفعات معین استفاده می‌کنیم. دو پارامتر اول range؛ start و stop نامیده می‌شوند. در مثال قبلی که از تابع Len برای کار با یک حلقه استفاده شد، شاهد استفاده از range با این دو پارامتر بودیم.

اعدادی که range تولید می‌کند با عددی آغاز می‌گردد که بعنوان پارامتر اول شناخته می‌شود و به عددی ختم می‌شو که یکی کمتر از پارامتر دوم است. بعنوان مثال، در ادامه نشان داده می‌شود که هنگام چاپ اعداد بین 0 و 5 که توسط range تولید شده‌اند، چه اتفاقی خواهد افتاد:

```
>>> for x in range(0, 5):
        print(x)
0
1
2
3
4
```

درواقع تابع range یک شیء خاص به نام iterator را برمی‌گرداند که یک عمـل را چنـدین بـار تکرار می‌کند. در این مورد، هر بار که فراخوانی می‌شود، بزرگترین عدد بعدی را برمی‌گرداند.

می‌توان تکرارگر[1] را به یک لیست تبدیل کرد (با استفاده از تابع list). اگر بدنبال فراخـوانی range ، مقدار بازگشتی را چاپ کنید، اعدادی که در آن وجود دارد را خواهید دید:

```
>>> print(list(range(0, 5)))
[0, 1, 2, 3, 4]
```

همچنین می‌توانید پارامتر سومی را به نام step را به range اضافه کنید. اگر مقدار Step قـرارداده نشده باشد، از عدد 1 بصورت پیش‌فرض بعنوان step (گام) استفاده شده است. ولی زمانی کـه عـدد 2 را بعنوان step درنظر بگیریم چه اتفاقی خواهد افتاد؟ نتیجه را می‌بینیم:

```
>>> count_by_twos = list(range(0, 30, 2))
>>> print(count_by_twos)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

هر عدد در لیست دو تا بیشتر از عدد قبلی است و لیست با عدد 28 خاتمه می‌یابد که 2 تا کمتـر از 30 می‌باشد. ازگام‌های منفی نیز می‌توانید استفاده کنید:

```
م>>> count_down_by_twos = list(range(40, 10, -2))
>>> print(count_down_by_twos)
[40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12]
```

تابع SUM

تابع sum آیتم‌های یک لیست را با هم جمع کرده و نتیجه‌ی نهایی را برمی‌گرداند. بعنوان مثال:

```
>>> my_list_of_numbers = list(range(0, 500, 50))
>>> print(my_list_of_numbers)
[0, 50, 100, 150, 200, 250, 300, 350, 400, 450]
>>> print(sum(my_list_of_numbers))
2250
```

در خط اول، با استفاده از range و یک گام 50تایی لیستی از اعداد بین 0 و 500 را ایجاد می‌کـنیم. سپس برای مشاهده‌ی نتیجه، لیست را چاپ(print) می‌نماییم. درنهایت، متغیر my_list_of_numbers را در تابع sum قرار داده و با print(sum(my_list_of_numbers)) تمامی آیتم‌های لیست را با هـم جمع کـرده تـا مجموع 2250 بدست آید.

[1] iterator

کار با فایل‌ها

فایل‌های پایتون همانند دیگر فایل‌های رایانه‌ی شما هستند: اسناد، تصاویر، موسیقی، بازی‌ها و.... درواقع، هرچیزی که رایانه شما بعنوان فایل ذخیره کرده است.

در اینجا خواهیم دید که چگونه در پایتون، با استفاده از تابع ذاتی open، میتوان فایل‌ها را باز نمود و با آنها کار کرد. ولی ابتد بایستی یک فایل جدید بسازیم.

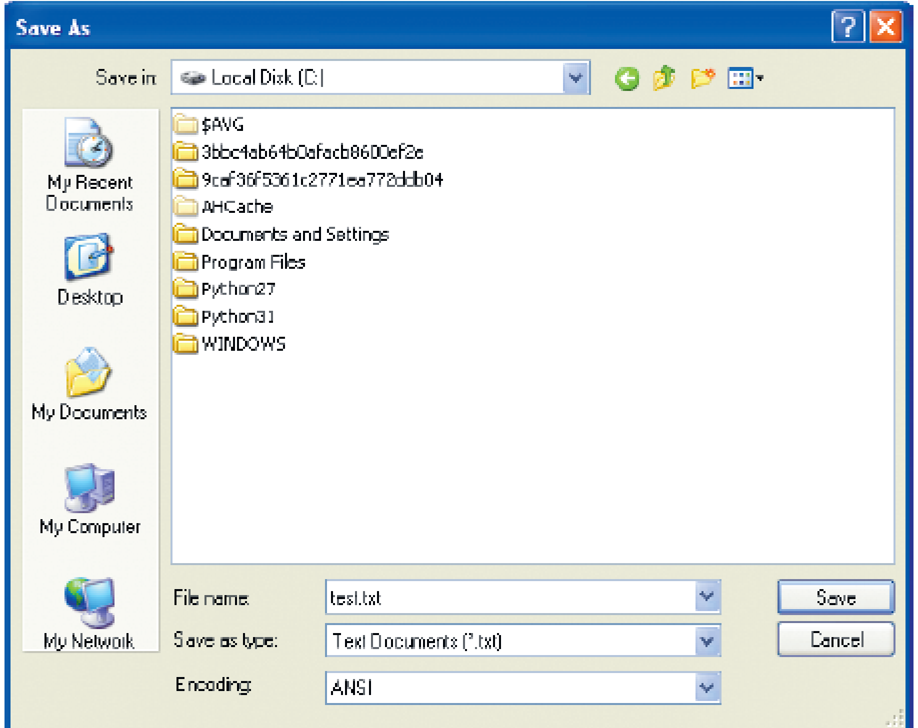
ایجاد یک فایل تست

با یک فایل متنی کار می‌کنیم و آن را test.txt می‌نامیم. مراحلی را دنبال کنید که برای سیستم-عامل مورداستفاده‌ی شما معین شده است.

ایجاد یک فایل جدید در ویندوز

اگر از ویندوز استفاده می‌کنید، مراحل زیر را برای ایجاد یک test.txt دنبال کنید:

۱. **Start ▸ All Programs ▸ Accessories ▸ Notepad** را انتخاب کنید
۲. یک خط جدید در فایل خالی وارد نمایید
۳. **File ▸ Save** را انتخاب کنید
٤. بعد از ظاهر شدن پنجره، انتخاب درایو:C با دوبار کلیک روی **My Computer** و سپس دوبار کلیک روی **Local Disk (C:)**
٥. test.txt را در بخش **File name** در پایین کادر نمایش داده شده، وارد کنید
٦. درنهایت روی دکمه‌ی Save کلیک نمایید



ایجاد یک فایل جدید در Mac OS X

اگر از یک Mac استفاده می‌کنید، مراحل زیر را برای ایجاد یک text.txt دنبال نمایید:

۱. کلیک روی آیکون **Spotlight** در نوار منوی در بالای صفحه

۲. وارد کردن *TextEdit* در کادر جستجویی که ظاهر می‌شود

۳. *TextEdit* بایستی در بخش Applications ظاهر گردد. روی آن کلیک کرده تا ادیتور[1] باز شود (می‌توانید TextEdit را در پوشه‌ی Applications در Finder پیدا کنید)

٤. چند خط متن در فایل خالی خود تایپ کنید

٥. انتخاب **Format ‣ Make Plain Text**

٦. انتخاب **File ‣ Save**

۷. در کادر **Save As**، test.txt را وارد کنید

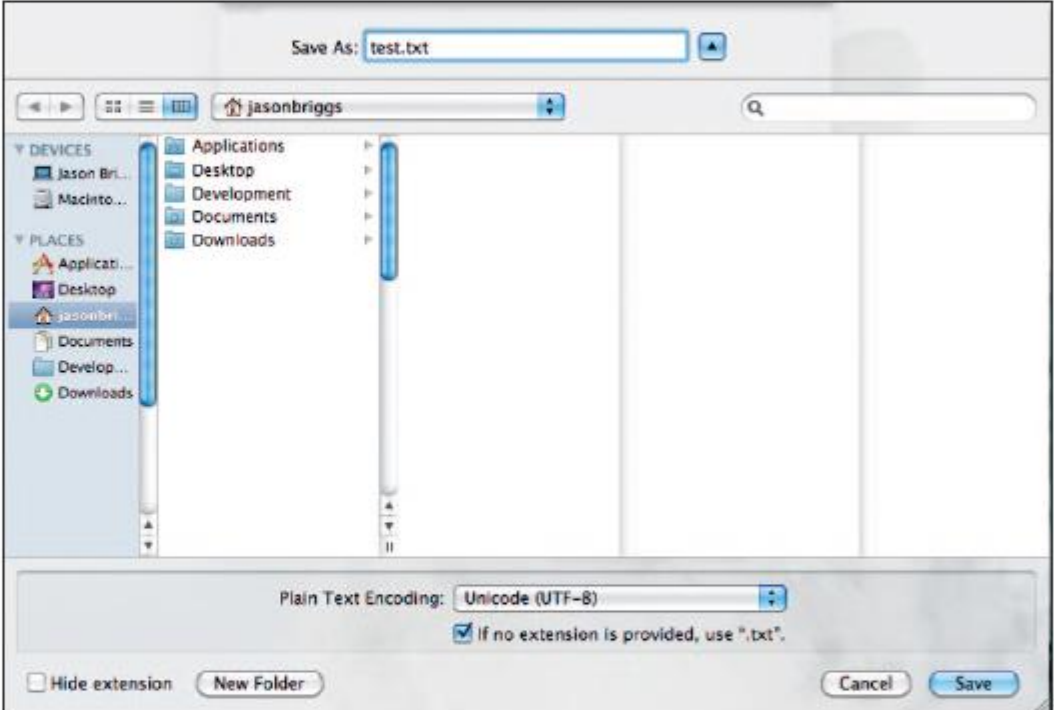
۸. در Places list روی نام کاربری[2] کلیک کرده- نامی که با آن وارد سیستم شده‌اید یا نام صاحب رایانه‌ای که از آن استفاده می‌کنید.

۹. در نهایت روی دکمه‌ی **Save** کلیک کنید.

---

[1] Editor ویرایشگر، ویراستار

[2] username



ایجاد یک فایل جدید در UBUNTU

اگر از UBUNTU استفاده می‌کنید، بایستی مراحل زیر را برای ایجاد یک text.txt دنبال نمایید:

۱. ادیتور خود را که معمولاً Text Editor نام دارد، بازکنید.

۲. چند خط متن در ادیتور وارد کنید.

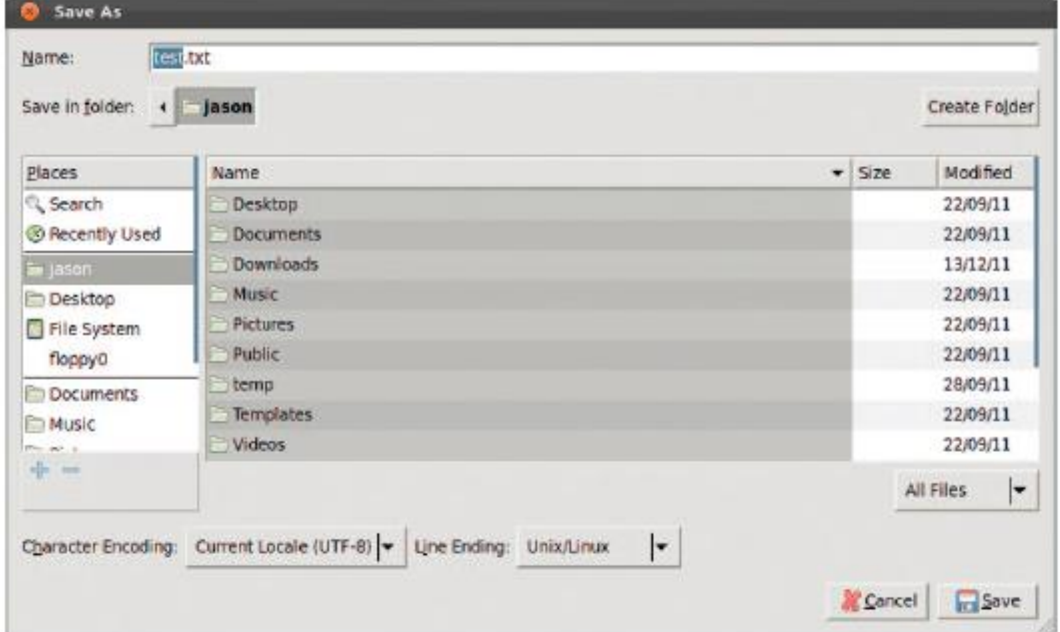
۳. **File ‣ Save** را انتخاب نمایید.

٤. در کادردر Name، text.txt ر بعنوان filename (نام فایل) وارد کنید. دایرکتوری اصلی مشار در این کادر تحت عنوان **Save in Folder** مشخص شده است، درغیراینصورت، در Places list روی آن کلیک نمایید (دایرکتوری اصلی[۱] شما، نام‌کاربری است که با آن وارد سیستم شده‌اید).

۵. روی دکمه‌ی Save کلیک کنید.

---

[۱] Home directory



باز کردن یک فایل در پایتون

تابع ذاتی open در پایتون، یک فایل را در پنجره‌ی پایتون باز کرده و محتویات آن را نمایش می‌دهد. این که به این تابع بگوید کدام فایل را باز کند، به سیستم‌عامل شما بستگی دارد. به مثال‌های مربوط به یک فایل ویندوز دقت کرده و اگر از سیستم‌عامل دیگری استفاده می‌کنید، به بخش ویژه‌ی Mac یا UBUNTU مراجعه نمایید.

باز کردن یک فایل ویندوز

اگر از ویندوز استفاده می‌کنید، برای باز کردن text.txt، کد زیر را وارد نمایید:

```
>>> test_file = open('c:₩₩test.txt')
>>> text = test_file.read()
>>> print(text)
There once was a boy named Marcelo
Who dreamed he ate a marshmallow
He awoke with a start
As his bed fell apart
And he found he was a much rounder fellow
```

در خط اول، Open را باز می‌کنیم که یک شیء فایل را همراه با توابعی برای کار کردن با فایل‌ها برمی‌گرداند. از پارامتر string همراه با تابع open استفاده می‌کنیم که به پایتون می‌گوید که در کجا می-

تواند فایل را پیدا کند. اگر از ویندوز استفاده می‌کنید،test.txt را در دیسک محلی روی درایو:C ذخیره کرده‌اید بنابراین مکان فایل را بصورتc:\\test.txt تعریف می‌کنید.

دو بک‌اسلش(\\) در نام‌فایل ویندوز به پایتون می‌گوید که بک‌اسلکش جزو فرمان نیست. (همانطوریکه در فصل۳ آموختید، بک‌اسلکش‌ها، بویژه در رشته(string)ها، در پایتون معنا دارند). شیء فایل را را در متغیرtest_fileذخیره می‌کنیم.

در خط دوم، برای خواندن محتویات فایل و ذخیره‌ی آن در متغیرtext از تابعread استفاده می-کنیم که توسط شیء فایل درست شده است. در خط آخر، بری نمایش محتویات فایل، متغیر را چاپ می‌کنیم.

باز کردن یک فایلMac OS X

اگر ازMac OS X استفاده می‌کنید، برای باز کردن test.txt ، بایستی مکان متفاوتی را برای اولین خط مثال ویندوز وارد کنید. هنگام ذخیره‌ی فایل متنی در رشته، از نام‌کاربری که روی آن کلیک کرده‌اید استفاده نمایید. بعنوان مثال، اگر *sarahwinters* نام‌کاربری باشد، پارامترopen بایستی به شکل زیر باشد:

```
>>> test_file = open('/Users/sarahwinters/test.txt')
```

باز کردن یک فایلUBUNTU

اگر ازUbuntu استفاده می‌کنید، بایستی برای باز کردن test.txt، بایستی مکان متفاوتی را برای اولین خط مثال ویندوز وارد نمایید. هنگام ذخیره‌ی فایل متنی در رشته، از نام‌کاربری که روی آن کلیک کرده‌اید استفاده کنید. بعنوان مثال، اگر *jacob* نام‌کاربری باشد، پارامترopen بایستی به شکل زیر باشد:

```
>>> test_file = open('/home/jacob/test.txt')
```

نوشتن در فایل‌ها

شیء فایلی که توسطopen بازگردانده شده است، علاوه بر read توابع دیگری نیز دارد. با استفاده از پارامتر دوم، رشته‌ی'w'، هنگام فراخوانی تابع زیر، می‌توانیم یک فایل خالی جدید را ایجاد کنیم:

```
>>> test_file = open('c:\\myfile.txt', 'w')
```

پارامتر'w' به پایتون می‌گوید که به جای خواندن از شیء فایل،می‌خواهیم در شیء فایل بنویسیم.

حال با استفاده از تابعwrite می‌توانیم اطلاعاتی را به این فایل جدید اضافه کنیم:

```
>>> test_file = open('c:₩₩myfile.txt', 'w')
>>> test_file.write('this is my test file')
```

درنهایت با استفاده از تابع close زیر، بایستی به پایتون بگوییم که چه زمان نوشتن در فایل را خاتمه داده‌ایم

```
>>> test_file = open('c:₩₩myfile.txt', 'w')
>>> test_file.write('What is green and loud? A froghorn!')
>>> test_file.close()
```

حال اگر فایل را با استفاده از ادیتور متن خود باز کنید،بایستی مشاهده کنید که حاوی متن "What is green and loud? A froghorn!" است یا این که از پایتون برای خواندن مجدد آن استفاده کنید:

```
>>> test_file = open('myfile.txt')
>>> print(test_file.read())
What is green and loud? A froghorn!
```

آنچه آموختید

در این فصل چیزهایی درباره توابع ذاتی پایتون، مثلاً float و int آموختید که می‌توانند اعداد اعشاری را به اعداد صحیح و بالعکس تبدیل کنند. همچنین دیدید که چگونه تابع Len قادر است ایجاد حلقه‌ی تکرار[1]را آسان‌تر کند و اینکه چگونه می‌توان برای خواندن از و نوشتن در فایل‌ها، از پایتون استفاده نمود.

چیستان‌های برنامه‌نویسی

برای درک برخی از توابع ذاتی پایتون، مثال‌های زیر را دنبال کنید. برای مشاهده‌ی پاسخ‌ها می‌توانید به آدرس *http://python-for-kids.com/* مراجعه کنید.

۱) کد اسرارآمیز

نتیجه‌ی اجرای کد زیر چیست؟ حدس زده و برای ارزیابی درستی حدس خود، کد را اجرا کنید.

[1] looping

```
>>> a = abs(10) + abs(-10)
>>> print(a)
>>> b = abs(-10) + -10
>>> print(b)
```

۲) یک پیام مخفی

از dir و help استفاده کرده تا بفهمیم چطور باید یک رشته را به واژه‌ها تقسیم کنیم و سپس یک برنامه‌ی کوچک برای چاپ بقیه‌ی واژه‌ها در رشته‌ی زیر می‌سازیم که با اولین واژه(this) شروع می‌شود:

```
"this if is you not are a reading very this good then way you to have
hide done a it message wrong"
```

۳) کپی کردن یک فایل

برای کپی کردن یک فایل، یک برنامه‌ی پایتون بنویسید. (تذکر: بایستی فایلی را که میخواهیم کپی کنید، باز کرده، آن را خوانده و سپس یک فایل جدید- کپی- ایجاد نمایید). با چاپ محتویات فایل جدید روی صفحه، بررسی کنید که برنامه کار می‌کند.

فصل ۱۰
ماژول‌های سودمند پایتون

همانطوریکه در فصل۷ آموختید، یک ماژول پایتون ترکیبی از توابع، کلاس‌ها و متغیرها می‌باشد. پایتون از ماژول‌ها برای گروه‌بندی توابع و کلاس‌ها استفاده کرده تا استفاده از آنها ساده‌تر شـود. بعنـوان مثال، ماژولturtle که در فصول قبل از آن استفاده کردیم، توابع و کلاس‌هایی را گروه‌بندی می‌کند کـه از آنها برای خلق یک بوم برای یک لاک‌پشت برای ترسیم روی صفحه استفاده شده است.

زمانی که یک ماژول را به یک برنامه وارد می‌کنید، مـی‌توانیـد از تمـامی محتویـات آن اسـتفاده نمایید. بعنوان مثال، زمانی که از ماژولturtle در فصل ٤ استفاده می‌کنیم، به کلاسpen دسترسی خـواهیم داشت که می‌توانیم از آن برای خلق یک شیء برای نمایش بوم لاک‌پشت استفاده نماییم:

```
>>> import turtle
>>> t = turtle.Pen()
```

پایتون برای انجام وظایف مختلف، ماژول‌های متنوع و متعددی را در اختیار دارد. در این فصل، نگاهی خواهیم داشت به برخی از سودمندترین نمونه ماژول‌ها و سعی می‌کنـیم از برخـی از توابـع آنهـا استفاده کنیم.

کپی برداری با استفاده از ماژولcopy

ماژولcopy حاوی توابعی برای خلق چندین کپی از شیءها می‌باشد. معمولاً، هنگام نوشتن یک برنامه، شیءهای جدیدی را خلق می‌کنید ولی برخی اوقات ایجـاد یـک کپـی از یـک شـیء و سـپس

استفاده از آن برای ایجاد یک شیء جدید سودمند خواهد بود، بویژه زمانی که پروسه‌ی خلق یک شیء چند مرحله‌ای باشد.

بعنوان مثال، فرض کنید کلاس Animal با تابع _init__ و پارامترهای name, number_of_legs و color را در اختیار داریم.

```
>>> class Animal:
def __init__(self, species, number_of_legs, color):
self.species = species
self.number_of_legs = number_of_legs
self.color = color
```

با استفاده از کد زیر می‌توانیم شیء جدیدی را در کلاس Animal ایجاد کنیم. اجازه دهید یک اسب-گریفین[1] صورتی شش پا به نام harry را خلق کنیم.

```
>>> harry = Animal('hippogriff', 6, 'pink')
```

فرض کنید بدنبال یک گله اسب-گریفین شش پا هستیم. می‌توانیم کد قبلی را بارها و بارها تکرار کرده یا از copy استفاده کنیم که در ماژول copy پیدا می‌شود:

```
>>> import copy
>>> harry = Animal('hippogriff', 6, 'pink')
>>> harriet = copy.copy(harry)
>>> print(harry.species)
hippogriff
>>> print(harriet.species)
hippogriff
```

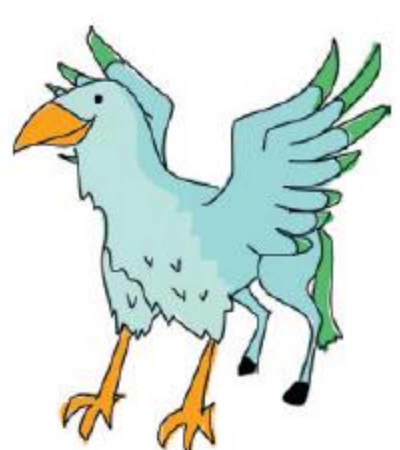
در این مثال، یک شیء ایجاد کرده و آن را با متغیر harry مشخص کردیم، سپس یک کپی از آن شیء ایجاد نموده و آن را harriet نامیدیم. این دو شیء کاملاً متفاوت هستند حتی با فرض این که ویژگی‌های یکسانی دارند. با این کار اندکی در تایپ صرفه‌جویی می‌شود، ولی زمانی که شیءها پیچیده-تر می‌شوند، کپی کردن آن‌ها مفید خواهد بود.

همچنین می‌توانیم یک لیست از شیءهای Animal را ایجاد و کپی کنیم.

```
>>> harry = Animal('hippogriff', 6, 'pink')
>>> carrie = Animal('chimera', 4, 'green polka dots')
>>> billy = Animal('bogill', 0, 'paisley')
>>> my_animals = [harry, carrie, billy]
>>> more_animals = copy.copy(my_animals)
>>> print(more_animals[0].species)
```

[1] hippogriff

```
hippogriff
>>> print(more_animals[1].species)
chimera
```



در سه خط اول، سه شیءAnimal را ایجاد کرده و آن‌ها را در harry, carrie وbilly ذخیره می‌کنیم. در خط چهارم، این شیءها را به لیستmy_animals اضافه می‌کنیم. سپس از copy برای ایجاد لیست جدید more_animals استفاده می‌نماییم. درنهایت، گونه‌های دو شیء اول([0] و[1]) در لیستmore_animals را چاپ کرده و مشاهده می‌کنیم که همانند لیست اصلی هستند: hippogriff وchimera. یک کپی از لیست می‌گیریم بدون این که نیازی باشد مجدداً تمامی شیءها را ایجاد کنیم.

ولی حالا می‌بینیم که اگر گونه‌های یکی از چهار شیءAnimal را در لیست اصلیmy_animals (ازhippogriff تاghoul) تغییر دهیم چه اتفاقی خواهد افتاد.

پایتون گونه‌ها را درmore_animals نیز تغییر می‌دهد.

```
>>> my_animals[0].species = 'ghoul'
>>> print(my_animals[0].species)
ghoul
>>> print(more_animals[0].species)
ghoul
```

خیلی عجیب است. مگر ما فقط گونه‌ها را در لیستmy_animals تغییر ندادیم؟ پس چرا گونه‌ها در هر دو لیست تغییر کردند؟

چونcopy واقعاً یک کپی سطحی می‌گیرد، به این معنی که شیءهای داخل شیءهایی که کپی کردیم را کپی‌برداری نمی‌کند، گونه‌ها تغییر خواهند کرد. در این‌جا شیء List اصلی را کپی کرده است نه شیءهای داخلی آن را.بنابراین لیست جدیدی در اختیار داریم که فاقد شیءهای جدید متعلق به آن می-باشد– لیستmore_animals دارای همان سه شیء درون آن می‌باشد.

به همین ترتیب اگر حیوان جدیدی را به لیست اول اضافه کنیم(my_animals)، درکپی(more_animals) ظاهر نخواهد شد. برای اثبات این موضوع، بعد از اضافه کردن هر حیوان، طول هر لیست را چاپ می‌کنیم:

```
>>> sally = Animal('sphinx', 4, 'sand')
>>> my_animals.append(sally)
>>> print(len(my_animals))
```

```
4
>>> print(len(more_animals))
3
```

همانطوریکه مشاهده می‌گردد، زمانی که حیوان جدیدی را به لیست اول یعنی my_animals اضافه می‌کنیم، به کپی آن لیست یعنی more_animals اضافه نخواهد شد. زمانی که از len استفاده کرده و نتایج را چاپ می‌کنیم، لیست اول دارای چهار المان و لیست دوم، سه المان خواهد داشت.

تابع دیگری که در ماژول copy وجود دارد؛ یعنی deepcopy، چندین کپی از تمامی شیءهای درون شیء کپی شده را ایجاد می‌کند. زمانی که از deepcopy برای کپی my_animals استفاده می‌کنیم، لیست جدیدی حاوی کپی‌های تمامی شیءهای آن در اختیار خواهیم داشت. درنتیجه، تغییر به یکی از شیءهای اصلی Animal ما اثری روی شیءهای در لیست جدید نخواهد داشت.بعنوان مثال:

```
>>> more_animals = copy.deepcopy(my_animals)
>>> my_animals[0].species = 'wyrm'
>>> print(my_animals[0].species)
Wyrm
>>> print(more_animals[0].species)
ghoul
```

زمانی که گونه‌های شیء اول در لیست اصلی را از ghoul به wyrm تغییر می‌دهیم، همانطوریکه هنگام چاپ گونه‌های شیء اول در هر لیست مشاهده می‌شود، لیست کپی شده تغییر نمی‌کند.

حفظ اثر کلیدواژه‌ها با ماژول KEYWORD

یک کلیدواژه در پایتون، به هر واژه‌ای در پایتون اتلاق می‌شود که بخشی از خود زبان است مثلاً If، else و for. ماژول keyword حاوی تابعی به نام iskeyword و متغیری به نام kwlist است. تابع iskeyword درصورتی true برمی‌گرداند که هر رشته یک کلیدواژه‌ی پایتون باشد. متغیر kwlist لیستی از تمامی کلیدواژه‌های پایتون را برمی‌گرداند.

در کد زیر دقت کنید که تابع iskeyword برای رشته‌ی if، true و برای رشته‌ی Ozwald، false را برمی‌گرداند. زمانی که محتویات متغیر را چاپ می‌کنیم، می‌توانید لیست کاملی از کلیدواژه‌ها را مشاهده کنید. این کار از این جهت سودمند است که کلیدواژه‌ها همواره یکسان باقی نمی‌مانند. در نسخه‌های جدید(یا نسخه‌های قدیمی‌تر) پایتون ممکن است کلیدواژه‌های متفاوتی وجود داشته باشد.

```
>>> import keyword
>>> print(keyword.iskeyword('if'))
True
```

```
>>> print(keyword.iskeyword('ozwald'))
False
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally',
'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while',
'with', 'yield']
```

در ضمیمه، شرحی از هر کلیدواژه ارائه شده است.

تولید اعداد تصادفی با استفاده از ماژول RANDOM

ماژول random حاوی چندین تابع است که برای تولید اعداد تصادفی مفید هستند– مثلاً از رایانه بخواهیم «عددی را انتخاب کن». سودمندترین توابع در ماژول random عبارتند از randint, choice و shuffle.

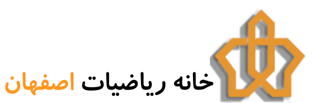
استفاده از random برای انتخاب یک عدد تصادفی

تابع randint یک عدد تصادفی را در رنجی از اعداد مثلاً بین 1 تا 100، بین 100 تا 1000 یا بین 1000 و 5000 انتخاب می‌کند. بعنوان مثال:

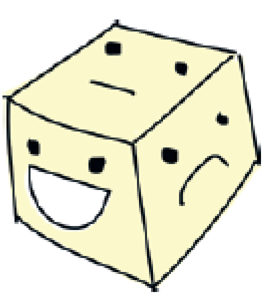
```
>>> import random
>>> print(random.randint(1, 100))
58
>>> print(random.randint(100, 1000))
861
>>> print(random.randint(1000, 5000))
3795
```

می‌توانید از randint برای ایجاد یک بازی حدس‌بزن ساده(و خسته‌کننده) با استفاده از حلقه-ی while استفاده کنید. مثلاً:

```
>>> import random
>>> num = random.randint(1, 100)
❶ >>> while True:
❷        print('Guess a number between 1 and 100')
❸        guess = input()
❹        i = int(guess)
❺        if i == num:
```




```
                print('You guessed right')
❻               break
❼       { elif i < num:
                print('Try higher')
❽       | elif i > num:
                print('Try lower')
```



ابتدا ماژول random را وارد می‌کنیم و سپس با استفاده از رنج 1 تا 100، متغیر num را برابر با یک عدد تصادفی قرار می‌دهیم. سپس یک حلقه‌ی while را در ❶ می‌سازیم که تا ابد تکرار خواهد شد (یا حداقل تا زمانی که بازیکن عدد را حدس بزند).

سپس، پیامی را در ❷ چاپ کرده و آنگاه از input برای گرفتن ورودی از کاربر استفاده کرده و در ❸ آن را در متغیر guess ذخیره می‌کنیم. با استفاده از int ورودی را به یک عدد تبدیل کرده و در ❹ آن را در متغیر i ذخیره می‌کنیم. سپس در ❺، آن را با عددی که به تصادف انتخاب شده است مقایسه می‌کنیم.

درصورت مساوی بودن ورودی و عددی که بصورت تصادفی تولید شده است، "You guessed right," (شما درست حدس زدید) را چاپ کرده و در ❻ از حلقه خارج می‌شویم. درغیراینصورت، در ❼ بزرگتر بودن عدد حدس زده شده نسبت به عدد تصادفی و در ❽ کوچکتر بودن آن نسبت به عدد تصادفی را بررسی کرده و براین اساس یک پیام تذکر را چاپ می‌کنیم.

این کد یک بیتی است بنابراین شاید بخواهید آن را در یک پنجره‌ی فرمان جدید تایپ کرده یا یک سند متنی ایجاد نموده، آن را ذخیره کرده و سپس در IDLE اجرا کنید. در اینجا یک یادآوری از شیوه‌ی باز کردن و اجرای یک برنامه‌ی ذخیره شده ارائه گردیده است:

۱. IDLE را آغاز کرده و **File ▸ Open** را انتخاب کنید

۲. همان دایرکتوری که فایل را در آن ذخیره کرده‌اید، باز کنید و برای انتخاب آن روی نام فایل کلیک کنید.

۳. روی Open کلیک کرده

٤. بعد از باز شدن پنجره‌ی جدید، **Run ▸ Run Module** را انتخاب کنید

در اینجا نشان داده شده است که بعد از اجرای برنامه چه اتفاقی می‌افتد

```
Python Shell
File  Edit  Debug  Options  Windows  Help
Python 3.1.2 (r312:79149, Mar 21 2010, 00:41:52) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>>
Guess a number between 1 and 100
50
Try higher
Guess a number between 1 and 100
70
Try lower
Guess a number between 1 and 100
60
Try higher
Guess a number between 1 and 100
65
Try higher
Guess a number between 1 and 100
68
Try higher
Guess a number between 1 and 100
69
You guessed right
>>>
                                                              Ln: 23 Col: 4
```

استفاده از CHOICE برای انتخاب یک آیتم تصادفی از یک لیست

اگر می‌خواهید به جای یک عدد تصادفی از یک رنج، یک آیتم تصادفی را از یک لیست انتخاب کنید، می‌توانید از choice استفاده کنید. بعنوان مثال، می‌خواهید پایتون دسر را برای شما انتخاب کند.

```
>>> import random
>>> desserts = ['ice cream', 'pancakes', 'brownies', 'cookies',
                'candy']
>>> print(random.choice(desserts))
brownies
```

احتمالاً شما نان خامه‌ی[1] خواهید داشت– انتخاب بدی نیست

استفاده از SHUFFLE برای بُر زدن[2] یک لیست

[1] brownies
[2] Shuffle به هم ریختن

تابع shuffle یک لیست را بُر زده و آیتم‌ها را به هم می‌ریزد. اگر در IDLE کار می‌کنید و با استفاده از random لیست دسرهای مثال قبل را ساخته‌اید، می‌توانید به سراغ فرمان random.shuffle در کد زیر بروید:

```
>>> import random
>>> desserts = ['ice cream', 'pancakes', 'brownies', 'cookies',
                'candy']
>>> random.shuffle(desserts)
>>> print(desserts)
['pancakes', 'ice cream', 'candy', 'brownies', 'cookies']
```

بعد از چاپ لیست می‌توانید نتایج را مشاهده کنید- ترتیب کاملاً متفاوت است. اگر یک بازی کارتی نوشته‌اید، می‌توانید از این تابع برای بُر زدن یک لیست معرف یک دسته کارت استفاده کنید.

کنترل کردن صفحه پایتون با ماژول SYS

ماژول sys حاوی تابعی از سیستم است که می‌توانید از آنها برای کنترل خود صفحه‌ی پایتون استفاده کنید. در اینجا خواهیم دید که چگونه از تابع exit، شیء‌های stdin و stdout و متغیر version استفاده می‌گردد.

خروج از صحفه‌ی پایتون با استفاده از تابع EXIT

تابع exit راهی است برای متوقف کردن کنسول و صفحه‌ی پایتون. با وارد کردن کد زیر صفحه- ای باز شده و در آنجا از شما خواسته می‌شود تا خروج از برنامه را تأیید کنید. روی Yes کلیک کرده و درنتیجه برنامه بسته خواهد شد.

```
>>> import sys
>>> sys.exit()
```

اگر از نسخه‌ی اصلاح شده‌ی IDLE که در فصل ۱ معرفی شد استفاده نکنید، این کار جواب نخواهد داد. درعوض، با پیام خطا مواجه خواهید شد:

```
>>> import sys
>>> sys.exit()
Traceback (most recent call last):
        File "<pyshell#1>", line 1, in <module>
          sys.exit()
SystemExit
```

خواندن با شیءSTDIN

شیءstdin (مخفف ورودی استانداردstandard inpput) در ماژولsys از کاربر می‌خواهد تا اطلاعاتی را وارد کند که در صفحه خوانده شده و برنامه از آنها استفاده می‌کند. همانطوریکه در فصل۷ آموختید، این شیء دارای یک تابعreadln است که تا قبل از فشرده شدن کلیدENTER توسط کاربر، از آن برای خواندن یک خط از متنی که با صفحه کلید تایپ شده است، استفاده می‌گردد. این تابع همانند تابعinput کار می‌کند که از آن برای بازی حدس عدد تصادفی استفاده کردیم. بعنوان مثال کد زیر را وارد کنید:

```
>>> import sys
>>> v = sys.stdin.readline()
He who laughs last thinks slowest
```

پایتون رشته‌یHe who laughs last thinks slowest را در متغیرv ذخیره می‌کند. برای تصدیق این موضوع، محتوایv را چاپ می‌کنیم:

```
>>> print(v)
He who laughs last thinks slowest
```

یکی از تفاوت‌های بین تابعreadline و input در این است که با تابعreadline می‌توانیم تعداد کاراکترهایی را که بعنوان یک پارامتر خوانده می‌شوند، تعیین کنیم. بعنوان مثال:

```
>>> v = sys.stdin.readline(13)
He who laughs last thinks slowest
>>> print(v)
He who laughs
```

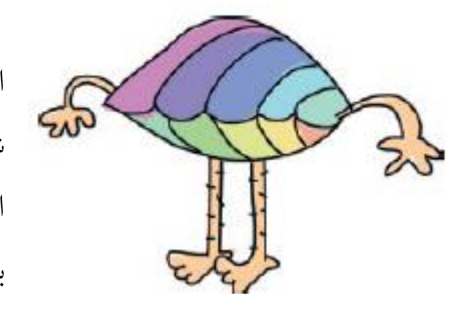
نوشتن با شیءSTOUT

برخلافstdin، می‌توان از شیءstdout (مخففstandard output) برای نوشتن پیام‌هایی در شِل(یا کنسول)، به جای خواندن آنها، استفاده کرد. از جنبه‌هایی مانندprint است ولی با این تفاوت که stdout یک شیء فایل است و بنابراین از همان توابعی برخوردار است که در فصل۹ بیان نمودیم، مثلاً .write بعنوان مثال:

```
>>> import sys
>>> sys.stdout.write("What does a fish say when it swims into a wall?
Dam.")
What does a fish say when it swims into a wall? Dam.52
```

توجه داشته باشید که به مجرد خاتمه‌یwrite، تعداد کاراکترهایی را که نوشته است، برمی‌گرداند. در پایان پیام می‌توان مشاهده نمود که52 در صفحه‌ی برنامه چاپ شده است. می‌توانیم این مقدار را در یک متغیر ذخیره کرده تا درطی زمان بفهمیم چند کاراکتر روی صفحه نوشته‌ایم.

من چطور بهفمم که از کدام نسخه‌ی پایتون استفاده می‌کنم؟



متغیر version نسخه‌ی پایتون شما را نمایش می‌دهد. زمانی این کار مفید است که بخواهید از به روز بودن نسخه‌ی خود مطمئن شوید. برخی از برنامه‌نویسان تمایل دارند با بالا آمدن برنامه‌هایشان، اطلاعات چاپ شود. بعنوان مثال، شاید بخواهید نسخه‌ی پایتون–بصورت زیر– در یک پنجره‌یAbout در برنامه‌تان ظاهر شود:

```
>>> import sys
>>> print(sys.version)
3.1.2 (r312:79149, Mar 21 2013, 00:41:52) [MSC v.1500 32 bit (Intel)]
```

زمان انجام کار با استفاده از ماژولTIME

ماژولtime پایتون شامل توابعی برای نمایش زمان است هرچند ضرورتی در انجام این کار نیست. کد زیر را امتحان کنید:

```
>>> import time
>>> print(time.time())
1300139149.34
```

عددی که با فراخوانtime() برگردانده می‌شود، درواقع تعداد ثانیه‌ها از ۱ ژانویه‌ی ۱۹۷۰ درAM 00:00:00 است. این نقطه ارجاع غیرمعمول در نگاه اول سودمند نخواهد بود ولی واقعیت چیز دیگری است. بعنوان مثال، برای این که بفهمیم اجرای بخش‌های برنامه چقدر طول می‌کشد، می‌توانیم زمان شروع و خاتمه را ثبت کرده و مقادیر را با هم مقایسه کنیم. همان کار را برای تعیین زمان لازم برای چاپ تمامی اعداد از0 تا999 انجام دهید.

ابتدا، تابعی مانند تابع زیر را ایجاد می‌کنیم:

```
>>> def lots_of_numbers(max):
        for x in range(0, max):
            print(x)
```

سپس تابع را با max=1000 صدا می‌زنیم:

```
>>> lots_of_numbers(1000)
```

در ادامه با اصلاح برنامه با ماژول time، زمان موردنیاز تابع را محاسبه می‌کنیم:

```
>>> def lots_of_numbers(max):
❶       t1 = time.time()
❷       for x in range(0, max):
            print(x)
❸       t2 = time.time()
❹       print('it took %s seconds' % (t2-t1))
```

مجدداً برنامه را فراخوانی کرده و نتیجه‌ی زیر بدست خواهد آمد (با توجه به سرعت سیستم نتیجه متفاوت خواهد بود):

```
>>> lots_of_numbers(1000)
0
1
2
3
.
.
.
997
998
999
it took 50.159196853637695 seconds
```

روند کار اینگونه است: اولین باری که تابع time() را صدا می‌زنیم، مقداری که در ❶ مقدار بازگشتی را به متغیر t1 نسبت می‌دهیم. سپس در ❷ حلقه‌ای ایجاد کرده و تمامی اعداد در خطوط سوم و چهارم را چاپ می‌کنیم. بعد از حلقه، مجدداً تابع time() را فراخوانی کرده و در ❸ مقدار بازگشتی را به متغیر t2 نسبت می‌دهیم. چون تکمیل حلقه چند ثانیه طول می‌کشد، مقدار t2 بزرگتر از t1 خواهد بود زیرا از ۱ ژانویه‌ی ۱۹۹۷ ثانیه‌های بیشتری سپری شده است. در ❹ از تفاضل t1 و t2، تعداد ثانیه‌های لازم برای چپ تمامی این اعداد بدست خواهد آمد.

تبدیل یک تاریخ با کمک ASCTIME

تابع asctime یک تاریخ را بصورت یک چندتایی درنظر گرفته و آن را به چیزی قابل خواندن تبدیل می‌کند. (بخاطر داشته باشید که یک چندتایی ماند یک لیست با آیتم‌های غیرقابل تغییر است).

همانطوریکه در فصل ۷ دیدیم، اگر asctime را بدون هیچ پارامتری فراخوانی کنیم، تاریخ و زمان جاری به فرمی قابل قرائت نمایش داده خواهد شد.

```
>>> import time
>>> print(time.asctime())
Mon Mar 11 22:03:41 2013
```

برای فراخوانی asctime با یک پارامتر، ابتدا یک چندتایی با مقادیری برای تاریخ و زمان ایجاد می‌کنیم. بعنوان مثال، چندتایی را به متغیر t نسبت می‌دهیم:

```
>>> t = (2007, 5, 27, 10, 30, 48, 6, 0, 0)
```

مقادیر در این دنباله عبارتند از سال، ماه، روز، ساعت، دقیقه، ثانیه، روز هفته(0 دوشنبه، 1 سه‌شنبه و به همین ترتیب)، روز سال(0 بعنوان مکان‌نما) و روز بودن یا نبودن(0 نباشد و 1 باشد). asctime را با چندتای مشابهی فراخوانی کرده و خواهیم داشت:

```
>>> import time
>>> t = (2020, 2, 23, 10, 30, 48, 6, 0, 0)
>>> print(time.asctime(t))
Sun Feb 23 10:30:48 2020
```

بدست آوردن تاریخ و زمان با تابع LOCALTIME

برخلاف asctime، تابع localtime تاریخ و زمان جاری را بصورت یک شیء برمی‌گرداند و مقادیر آن تقریباً به همان ترتیب ورودی asctime هستند. اگر شیء را چاپ کنید، نام کلاس را مشاهده کرده و هر یک از مقادیر با tm_year، tm_mon(برای ماه) tm_mday (برای روز ماه)، tm_hour و ... برچسب‌گذاری شده‌اند.

```
>>> import time
>>> print(time.localtime())
time.struct_time(tm_year=2020, tm_mon=2, tm_mday=23, tm_hour=22,
tm_min=18, tm_sec=39, tm_wday=0, tm_yday=73, tm_isdst=0)
```

برای چاپ سال و ماه جاری می‌توانید از جایگاه اندیس آنها (همانند چندتایی که با asctime استفاده کردیم) استفاده کنید. براساس این مثال، می‌دانیم که year در جایگاه اول(موقعیت0) و month در جایگاه دوم (1) قرار دارد. بنابراین از year = t[0] و month = t[1] استفاده می‌کنیم:

```
>>> t = time.localtime()
>>> year = t[0]
>>> month = t[1]
>>> print(year)
```

```
2020
>>> print(month)
2
```

و می‌بینیم که در ماه دوم ۲۰۲۰ هستیم.

ایجاد وقفه با استفاده از تابع SLEEP

از تابع sleep زمانی استفاده می‌شود که می‌خواهید در برنامه تأخیر ایجاد کرده یا اجرای آن را کند کنید. بعنوان مثال برای چاپ هر ثانیه از 1 تا 60 می‌توانیم از حلقه‌ی زیر استفاده کنیم:

```
>>> for x in range(1, 61):
        print(x)
```

این کد به سرعت تمامی اعداد از 1 تا 60 را چاپ می‌کند. هرچند، می‌توانیم به پایتون بگوییم که یک ثانیه بین هر دستور print کاری انجام ندهد:

```
>>> for x in range(1, 61):
        print(x)
        time.sleep(1)
```

در نتیجه، تأخیری بین نمایش هر عدد ایجاد خواهد شد. در فصل ۱۲، از تابع sleep برای واقعی‌تر جلوه دادن یک انیمیشن استفاده خواهیم کرد.

استفاده از ماژول PICKLE برای ذخیره‌ی اطلاعات

از ماژول pickle برای تبدیل شیءهای پایتون به چیزی استفاده شده است که در یک فایل قابل نوشتن و بسادگی قابل خواندن باشد. ماژول pickle هنگام نوشتن یک بازی یا زمانی که بخواهیم اطلاعاتی را درباره‌ی پیشرفت یک بازیکن ذخیره کنیم، به درد می‌خورد. بعنوان مثال در اینجا نشان می‌دهیم که چگونه می‌توانیم ویژگی ذخیره را به یک بازی اضافه کنیم:

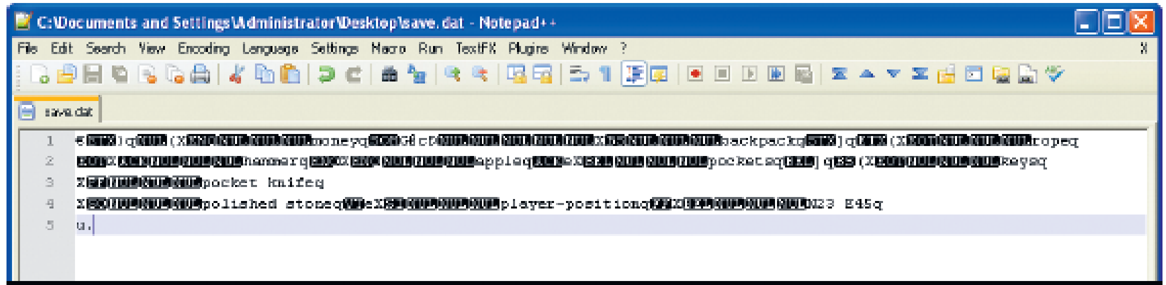
```
>>> game_data = {
        'player-position' : 'N23 E45',
        'pockets' : ['keys', 'pocket knife', 'polished stone'],
        'backpack' : ['rope', 'hammer', 'apple'],
        'money' : 158.50
}
```

در اینجا، یک نقشه‌ی پایتون حاوی موقعیت جاری بازیکن را در بازی خیالی خودمان، فهرستی از آیتم‌ها در جیب جلویی و عقبی بازیکن و مقدار پولی که بازیکن با خود حمل می‌کند را خلق می‌کنیم. با باز کردن یک فایل برای نوشتن و سپس فراخوانی تابع pickle's dump می‌توانیم این نقشه را در یک فایل ذخیره کنیم:

```
❶ >>> import pickle
❷>>> game_data = {
'player-position' : 'N23 E45',
'pockets' : ['keys', 'pocket knife', 'polished stone'],
'backpack' : ['rope', 'hammer', 'apple'],
'money' : 158.50
        }
❸>>> save_file = open('save.dat', 'wb')
❹>>> pickle.dump(game_data, save_file)
❺ >>> save_file.close()
```

ابتدا در ❶ ماژول pickle را وارد کرده و در ❷ نقشه‌ای از داده‌های بازی را می‌سازیم. در ❸ فایل save.dat را با پارامتر wb باز می‌کنیم که به پایتون می‌گوید فایلی را در مد باینری بنویسد(شاید لازم باشد آن را در یک دایرکتوری مثلاً */home/susanb/* ،*/Users/malcolmozwald* یا *C:\\Users\JimmyIpswich* ذخیره کنید همانطوریکه در فصل ۹ انجام شد). در ❹ از dump برای حرکت در نقشه و متغیر file بعنوان دو پارامتر استفاده می‌کنیم. درنهایت در ❺، فایل را می‌بندیم چون کارمان با آن تمام شده است.

نکته: فایل‌های ساده‌ی متنی فقط حاوی کاراکترهایی هستند که توسط انسان قابل خواندن هستند. تصاویر، فایل‌های صوتی، فیلم‌ها و شیءهای pickle شده‌ی پایتون حاوی اطلاعاتی هستند که همواره توسط انسان قابل خواندن نیستند بنابراین تحت عنوان فایل‌های باینری شناخته می‌شوند. اگر می‌خواهید فایل dat. را ذخیره کنید، خواهید دید که شبیه یک فایل متنی نیست؛ بلکه شبیه مخلوطی درهم-آمیخته از متن ساده و کاراکترهای ویژه است.



می‌توانیم شیءهایی که با استفاده از تابع pickle's load در فایل نوشته شده‌اند را از حالت pickle خارج کنیم. زمانیکه چیزی را از حالت pickle خارج می‌کنیم، معکوس فرآیند pickle را انجام مـی‌دهـیم: اطلاعات نوشته شده در فایل را گرفته و آن را به مقادیری برمی‌گردانیم کـه بـرای برنامـه قابـل اسـتفاده باشند. این فرآیند مشابه استفاده از تابع dump است.

```
>>> load_file = open('save.dat', 'rb')
>>> loaded_game_data = pickle.load(load_file)
>>> load_file.close()
```

ابتدا با استفاده از rb (به معنای خوندن باینری) بعنوان یک پارامتر، فایل را باز مـی‌کـنیم. سـپس فایل را به load انتقال داده و مقدار بازگشت را برابر با متغیر loaded_game_data قرار می‌دهـیم. در پایـان فایل را می‌بندیم.

برای اینکه ثابت کنیم داده‌های ذخیره شده بدرستی بارگذاری شده‌اند؛ متغیر را چاپ می‌کنیم:

```
>>> print(loaded_game_data)
{'money': 158.5, 'backpack': ['rope', 'hammer', 'apple'],
'player-position': 'N23 E45', 'pockets': ['keys', 'pocket knife',
'polished stone']}
```

آنچه آموختید

در این فصل آموختید که چگونه ماژول‌های پایتون، توابع، کلاس‌ها و متغیرها را گروه‌بندی کرده و چگونه با کمک وارد کردن ماژول، از این توابع استفاده می‌شود. دیدید که چگونه مـی‌تـوان شـیء‌ها را کپی کرد، اعداد تصادفی تولید نمود و به تصادف فهرستی از شیء‌ها را بُر زد و همچنین چگونه با زمـان در پایتون کار کرد. درنهایت، آموختید که چگونه با استفاده از pickle، اطلاعات را در یک فایل ذخیره و از یک فایل فراخوانی کنید.

چیستان‌های برنامه‌نویسی

کد زیر برای یادگیری ماژول‌های پایتون، امتحان کنید. می‌توانید پاسخ‌های خود را در آدرس *http://python-for-kids.com* بررسی نمایید.

۱) ماشین‌های کپی‌برداری شده

کد زیر چه چیزی را چاپ می‌کند؟

```
>>> import copy
>>> class Car:
        pass
>>> car1 = Car()
>>> car1.wheels = 4
>>> car2 = car1
>>> car2.wheels = 3
>>> print(car1.wheels)
```

چه چیزی در اینجا چاپ می‌شود؟→

```
>>> car3 = copy.copy(car1)
>>> car3.wheels = 6
>>> print(car1.wheels)
```

چه چیزی در اینجا چاپ می‌شود؟→

۲) برگزیدگان pickle شده

فهرستی از موارد موردعلاقه‌ی خود را ایجاد کرده و سپس از pickle برای ذخیره‌ی آنها در فایلی به نام *favorites.dat* استفاده کنید. شِل پایتون را ببندید و مجدداً آن را باز کنید و بت بارگذاری فایل، فهرست موارد برگزیده‌ی خود را نمایش دهید.

فصل ۱۱

تعداد بیشتری گرافیک‌های لاک‌پشت

اجازه دهید نگاهی دوباره داشته باشیم به ماژول turtle که در فصل ٤ از آن استفاده کردیم. همانطوریکه در این فصل خواهیم دید، در پایتون، لاک‌پشت‌ها کاری بیش از ترسیم خطوط سیاه ساده انجام می‌دهند. بعنوان مثال، میتوانید از آنها برای ترسیم اشکال گرافیکی پیشرفته‌تر استفاده کرده، رنگ-های مختلفی را ایجاد نموده و حتی شکل‌های خود را رنگ‌آمیزی کنید.

شروع کار با مربع

آموختیم که چگونه کاری کنیم تا لاک‌پشت شکل‌های ساده‌ای را ترسیم کند. قبل از استفاده از لاک‌پشت، بایستی ماژول turtle را وارد کرده و شیء pen را درست کنیم:

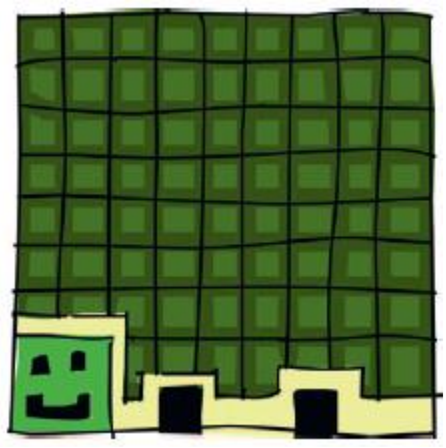
```
>>> import turtle
>>> t = turtle.Pen()
```

در اینجا کدی وجود دارد که از آن برای درست کردن یک مربع در فصل ٤ استفاده کردیم:

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
```

در فصل٦، حلقه‌های for را یاد گرفتیم. با دانشی که جدیداً بدست آوریم میتوانیم با استفاده از یک حلقه for، کد نسبتاً خسته کننده یک مربع را ساده کنیم:

```
>>> t.reset()
>>> for x in range(1, 5):
t.forward(50)
t.left(90)
```



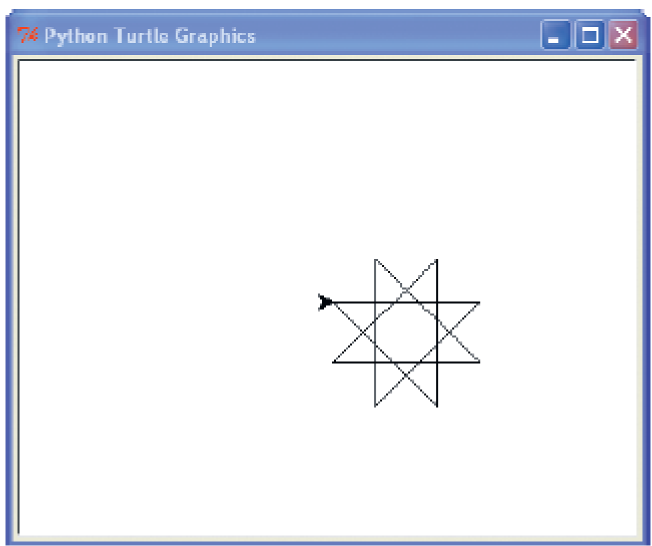
در خط اول،به شیء pen میگوییم خود را بازنشانی(reset ری‌ست)کند. سپس حلقه for ای را شروع می-کنیم که از ۱ تا٤ را با کد(range(1, 5 می‌شمارد. سپس، با خطوط زیر در هر اجرای حلقه، ٥٠ پیکسل به جلو رفته و ٩٠درجه به چپ می‌چرخیم. چون از حلقه for استفاده کردیم، کد اندکی کوتاهتر از نسخه قبلی است– از خط reset صرفنظر کرده و کد را از شش خط به سه خط کاهش می‌دهیم.

ترسیم ستاره

حال، با چند تغییر ساده در حلقه for، میتوانیم چیز جالب‌تری بسازیم. کد زیر را تایپ کنید:

```
>>> t.reset()
>>> for x in range(1, 9):
t.forward(100)
t.left(225)
```

این کد یک ستاره هشت پر را می‌سازد.



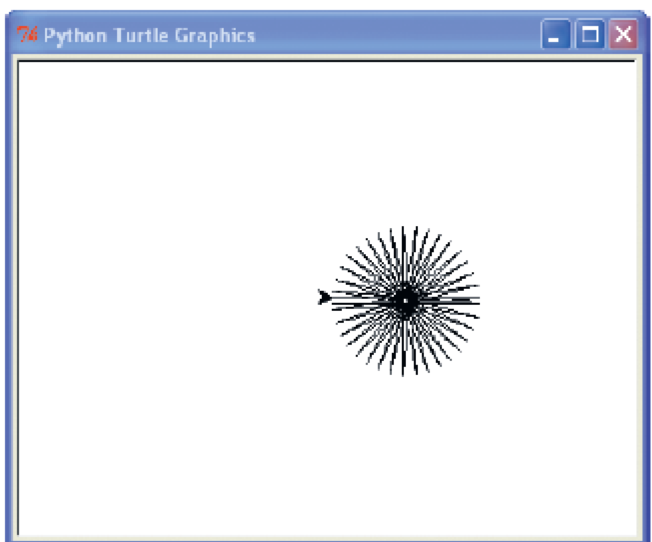
خود کد بسیار شبیه به کدی است که برای ترسیم مربع استفاده شده است ولی با این تفاوت که:

- به جای این که با(range(1, 5 چهار بار حلقه بزنیم، با(range(1, 9 هشت بار حلقه می‌زنیم.
- به جای اینکه ۵۰ پیکسل جلو برویم؛ ۱۰۰ پیکسل جلو می‌رویم.
- به جای اینکه ۹۰درجه بچرخیم، ۲۲۵ درجه به چپ می‌چرخیم.

حال ستاره را اندکی بیشتر جلو می‌بریم. با استفاده از کد زیر، با استفاده از یک زاویه ۱۷۵درجه و ۳۷ بار حلقه زدن، میتوانیم ستاره‌ای با تعداد نقاط(رئوس) بیشتر بسازیم:

```
>>> t.reset()
>>> for x in range(1, 38):
        t.forward(100)
        t.left(175)
```

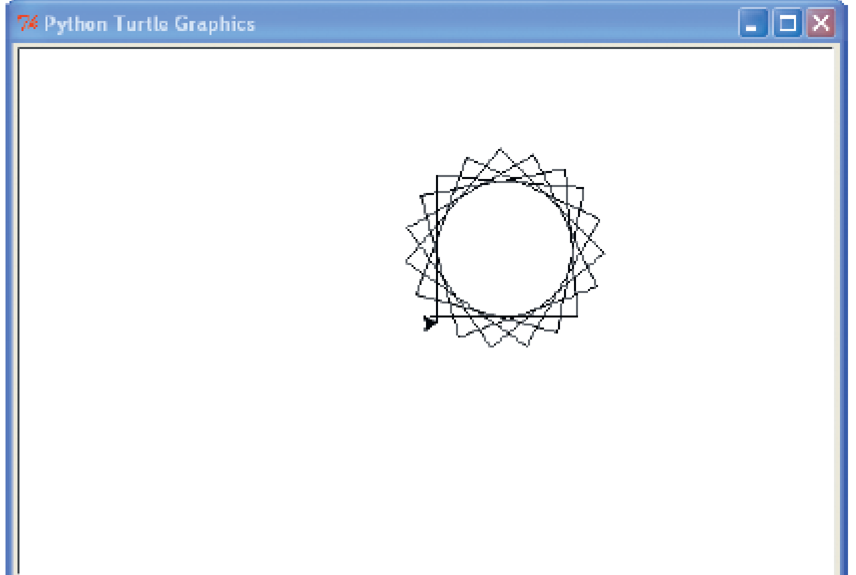
نتیجه اجرای این کد بصورت زیر خواهد بود:

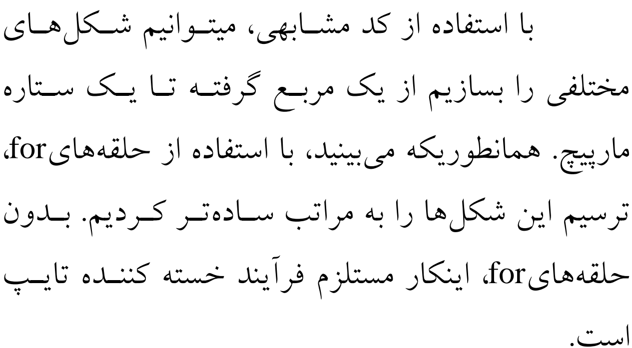


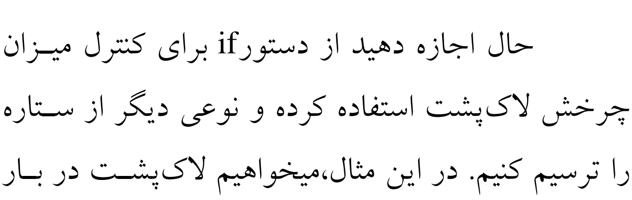
درحالیکه با ستاره‌ها بازی می‌کنیم، در ادامه کدی برای ساختن یک ستاره مارپیچ(حلزونی) ارائه گردیده است:

```
>>> t.reset()
>>> for x in range(1, 20):
        t.forward(100)
        t.left(95)
```

با تغییر درجه چرخش و کاهش تعداد حلقه‌ها، لاک‌پشت ترسیم سبک متفاوتی از ستاره را به پایان می‌برد:




با استفاده از کد مشابهی، میتوانیم شکل‌های مختلفی را بسازیم از یک مربع گرفته تا یک ستاره مارپیچ. همانطوریکه می‌بینید، با استفاده از حلقه‌هایfor، ترسیم این شکل‌ها را به مراتب ساده‌تر کردیم. بدون حلقه‌هایfor، اینکار مستلزم فرآیند خسته کننده تایپ است.

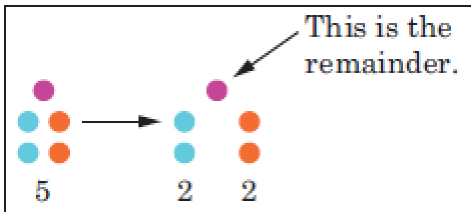

حال اجازه دهید از دستورif برای کنترل میزان چرخش لاک‌پشت استفاده کرده و نوعی دیگر از ستاره را ترسیم کنیم. در این مثال،میخواهیم لاک‌پشت در بار اول یک درجه چرخیده و بار بعد یک درجه دیگر بچرخد.

```
>>> t.reset()
>>> for x in range(1, 19):
        t.forward(100)
        if x % 2 == 0:
           t.left(175)
        else:
           t.left(225)
```

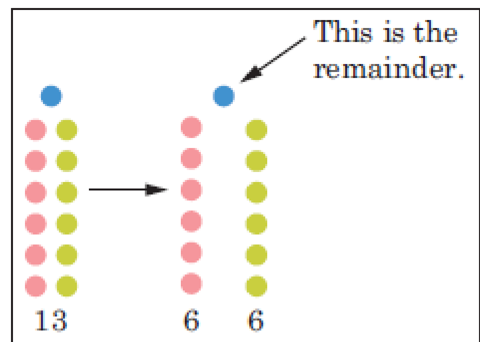
در اینجا یک حلقه درست می‌کنیم که ۱۸بار((range(1, 19) را اجرا کرده و به لاک‌پشت می‌گوید ۱۰۰ پیکسل به جلو برود((t.forward(100). اتفاق جدیدی که در اینجا رخ داده است، دستورif می‌باشد(if x

0:) == 2 %. این دستور با استفاده از عملگر modulo بررسی می‌کند که متغیر x حاوی یک عدد زوج باشد و % در عبارت 0 == 2 % x راهی است برای گفتن این که«x mod2 برابر با 0 است»

عبارت x%2 اساساً بیان می‌کند «زمانی که عدد متغیر x را به دو تقسیم می‌کنیم، چقدر باقی می-ماند؟» بعنوان مثال،اگر ۵ توپ را به ۲ تقسیم کنیم، دو دسته ۲تایی توپ (مجموعاً ٤توپ) و باقیمانده •مقدار باقیمانده)۱ توپ خواهد بود.



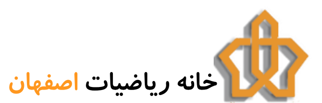
اگر ۱۳ توپ را به ۲ تقسیم کنیم، دو دسته ٦تایی توپ و ۱ توپ باقیمانده خواهیم داشت.

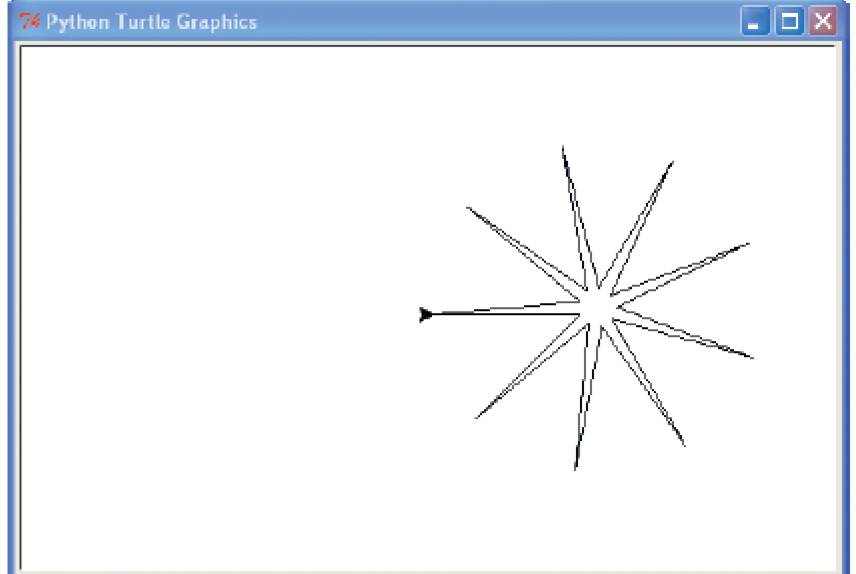


زمانیکه بررسی می‌کنیم تقسیم x بر ۲، هیچ باقیمانده‌ای نداشته باشد(باقیمانده صفر) درواقع بدنبال این هستیم که آیا میتوان آن را بدون هیچ باقیمانده‌ای به دو تقسیم کرد. این شیوه یک راه بسیار مطلوب برای این است که بفهمیم عدد یک متغیر زوج است یا خیر زیرا اعداد زوج همواره به دو بخش‌پذیر هستند(به دو بخش مساوی تقسیم می‌شوند).

در خط پنجم این کد به لاک‌پشت می‌گوییم که اگر عدد x زوج است(if x % 2 == 0:) ، ۱۷۵درجه به چپ بچرخد((t.left(175)) ؛ درغیراینصورت(else) ، در خط نهایی، می‌گوییم ۲۲۵ درجه به چپ بچرخد (t.left(225)).

در انیجا نتیجه اجرای کد نشان داده شده است:





ترسیم یک ماشین

لاک‌پشت به غیر از ترسیم ستاره و اشکال ساده هندسی میتواند ترسیمات دیگری را نیز انجام دهد. بعنوان مثال بعدی،یک ماشین خوش–ظاهر می‌کشیم. ابتدا بدنه ماشین را ترسیم می‌کنیم. درIDLE ، **File ▸ New Window** را انتخاب کرده و سپس کد زیر را در پنجره نمایش داده شده وارد می‌کنیم:

```
t.reset()
t.color(1,0,0)
t.begin_fill()
t.forward(100)
t.left(90)
t.forward(20)
t.left(90)
t.forward(20)
t.right(90)
t.forward(20)
t.left(90)
t.forward(60)
t.left(90)
t.forward(20)
t.right(90)
t.forward(20)
t.left(90)
t.forward(20)
```

```
t.end_fill()
```

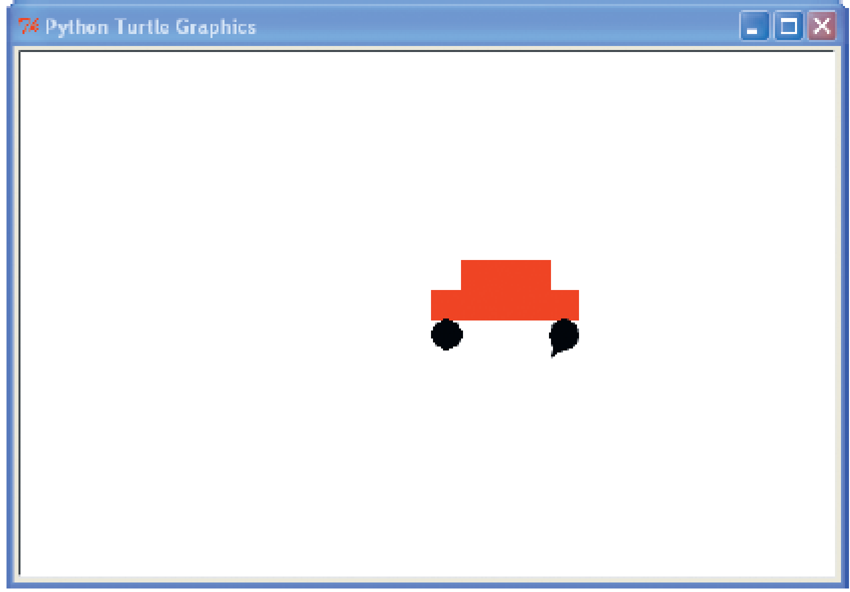
سپس چرخ اول را می‌کشیم:

```
t.color(0,0,0)
t.up()
t.forward(10)
t.down()
t.begin_fill()
t.circle(10)
t.end_fill()
```

درنهایت، چرخ دوم را خواهیم کشید.

```
t.setheading(0)
t.up()
t.forward(90)
t.right(90)
t.forward(10)
t.setheading(0)
t.begin_fill()
t.down()
t.circle(10)
t.end_fill()
```

File ▸ Save As را انتخاب می‌کنیم. یک نام برای فایل درنظر می‌گیریم مثلاً car.py.
Run ▸ Run Module را انتخاب کرده و کد را تست می‌کنیم. در نتیجه ماشین ما آماده است:



اگر دقت کرده باشید، چند تابع جدید لاک‌پشت در این کد آشکار می‌شوند:

- از color برای تغییر رنگ قلم استفاده می‌شود
- از begin_fill و begin_fill برای پر کردن (رنگ کردن) سطح بوم با رنگ استفاده می‌شود
- Circle یک دایره با سایز مشخص را می‌کشد
- setheading لاک‌پشت را به جهت مشخصی می‌چرخاند

حال می‌بینیم که چگونه میتوانیم از این توابع برای اضافه کردن رنگ به نقاشی‌مان استفاده کنیم.

رنگ کردن چیزها

تابع color سه پارامتر دارد. پارامتر اول مقدار رنگ قرمز، پارامتر دوم مقدار رنگ سبز و سومین پارامتر، مقدار رنگ آبی را تعیین می‌کند. بعنوان مثال، برای رنگ قرمز روشن(براق) ماشین، از color(1,0,0) استفاده می‌کنیم که به لاک‌پشت می‌گوید از قلم قرمز ۱۰۰٪ استفاده کند.

این دستورالعمل رنگ قرمز، سبز و آبی، RGB نامیده می‌شود. به این شیوه رنگ‌ها روی صفحه نمایش رایانه شما ظاهر می‌شوند و رنگ‌های دیگر از ترکیب نسبی این رنگ‌های اصلی(اولیه) بدست می‌آیند، دقیقاً مانند زمانی که برای بدست آوردن رنگ ارغوانی[1]،رنگ آبی و قرمز را با هم ترکیب می‌کنید یا برای بدست آوردن رنگ نارنجی، دو رنگ زرد و قرمز را ترکیب می‌کنید. رنگ‌های قرمز، سبز و آبی رنگ‌های اصلی(اولیه) نامیده می‌شوند زیرا از ترکیب رنگ‌های دیگر نمیتوانید به این رنگ‌ها دست پیدا کنید.

هرچند زمانیکه رنگ‌ها را روی صفحه نمایش رایانه می‌سازیم (از نور استفاده می‌کنیم)، درواقع از رنگ استفاده نمی‌کنیم، ولی برای این که این دستورالعمل RGB را بهتر درک کنیم، سه قوطی رنگ را درنظر می‌گیریم: یک قوطی رنگ قرمز، یک سبز و یک قوطی رنگ آبی. قوطی‌ها پر بوده و مقدار ۱(یا ۱۰۰٪) را برای هر قوطی پر درنظر می‌گیریم. سپس در یک ظرف بزرگتر، کل قوطی رنگ قرمز را با کل رنگ سبز مخلوط کرده تا رنگ زرد بدست آید(۱و۱ از هر کدام یا ۱۰۰٪ از هر رنگ).

حال دوباره به دنیای کد بازمی‌گردیم. برای اینکه با استفاده از لاک‌پشت یک دایره زرد بکشیم، از ۱۰۰٪ دو رنگ سبز و قرمز(نه آبی)استفاده می‌کنیم:

```
>>> t.color(1,1,0)
>>> t.begin_fill()
>>> t.circle(50)
>>> t.end_fill()
```

[1] purple

1,1,0 در خط اول نشان دهنده ۱۰۰٪قرمز،۱۰۰٪سبز و ۰٪ رنگ آبی است. در خط بعد به لاک-پشت می‌گوییم شکل‌هایی که ترسیم کرده است را با این رنگ RGB پر کند(t.begin_fill) و سپس از آن میخواهیم تا با دستور (t.circle) دایره‌ای بکشد. در خط آخر،(t.circle) به لاک‌پشت می‌گوید تا دایره را با رنگ RGB پر کند(رنگ‌آمیزی).

تابعی برای ترسیم یک دایره توپر

برای اینکه راحت‌تر با رنگ‌های مختلف کار کنیم، تابعی را از این کد می‌سازیم و از آن برای ترسیم یک دایره توپر استفاده می‌کنیم.

```
>>> def mycircle(red, green, blue):
        t.color(red, green, blue)
        t.begin_fill()
        t.circle(50)
        t.end_fill()
```

فقط با نیمی از رنگ سبز(0.5) میتوانیم یک دایره سبز پر رنگ بکشیم:

```
>>> mycircle(0, 0.5, 0)
```

برای بازی با رنگ‌های RGB روی صفحه نمایش، ابتدا دایره‌ای با رنگ قرمز، سبز نصف قرمز(1و0.5)، در ادامه کامل با رنگ آبی و درنهایت با نصف رنگ آبی می‌کشیم:

```
>>> mycircle(1, 0, 0)
>>> mycircle(0.5, 0, 0)
>>> mycircle(0, 0, 1)
>>> mycircle(0, 0, 0.5)
```

نکته: اگر بوم شما درهم و برهم شد، از*t.reset()* برای حذف ترسیم‌های قبلی استفاده کنید. همچنین بخاطر داشته باشید که بدون ترسیم خطوط با استفاده از*t.up()* برای برداشتن قلم و(*use t.down()* برای برگردان قلم، میتوانید لاک‌پشت را حرکت دهید.

ترکیب‌های مختلف قرمز، سبز و آبی دامنه گسترده‌ای از رنگ‌ها را بوجود می‌آورند، مثلاً طلایی:

```
>>> mycircle(0.9, 0.75, 0)
```

در اینجا رنگ صورتی کم رنگ را داریم

```
>>> mycircle(1, 0.7, 0.75)
```

و در اینجا دو نسخه برای رنگ نارجی را خواهیم داشت

```
>>> mycircle(1, 0.5, 0)
>>> mycircle(0.9, 0.5, 0.15)
```

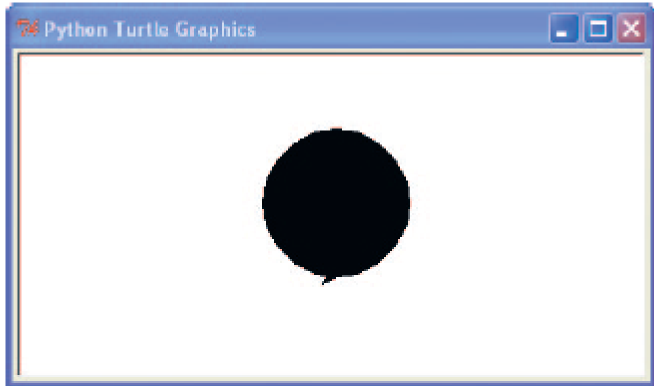
خودتان یک سری رنگ را با هم ترکیب کنید!

ایجاد سیاه و سفید خالص

وقتی تمامی چراغ‌ها را در شب خاموش کنید، چه اتفاقی می‌افتد؟ همه چیز سیاه می‌شود. تقریباً چنین اتفاقی برای رنگ‌ها در یک رایانه می‌افتد. نبودن نور یعنی نبودن رنگ، بنابراین یک دایره که تمامی رنگ‌ها اولیه‌اش 0 باشند، سیاه را تولید می‌کند:

```
>>> mycircle(0, 0, 0)
```

درنتیجه داریم:



اگر از ۱۰۰ تمامی رنگ‌های اولیه استفاده کنید، موضوع کاملاً عکس خواهد بود. در این مورد به رنگ سفید خواهید رسید. کد زیر را وارد کرده تا دایره سیاه خود را تمیز کنید:

```
>>> mycircle(1, 1, 1)
```

تابع رسم مربع

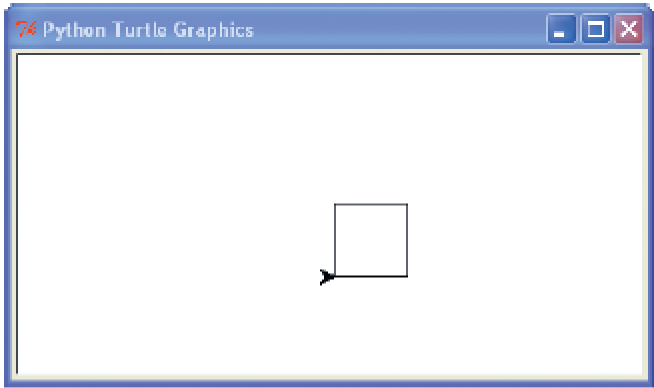
دیدید که با اعلام به لاک‌پشت برای شروع پر کردن با استفاده از begin_fill شکل‌ها را با رنگ پر کرده و به مجرد پر شدن شکل‌ها از تابع end_fill استفاده می‌کنیم.حال سعی می‌کنیم آزمایشات بیشتری را با شکل‌ها و رنگ‌آمیزی انجام دهیم. از تابع رسم مربع ابتدای فصل استفاده کرده و سایز مربع را بعنوان یک پارامتر درنظر می‌گیریم.

```
>>> def mysquare(size):
for x in range(1, 5):
t.forward(size)
t.left(90)
```

با فراخوانی تابع با سایز50 آن را تست کنید:

```
>>> mysquare(50)
```

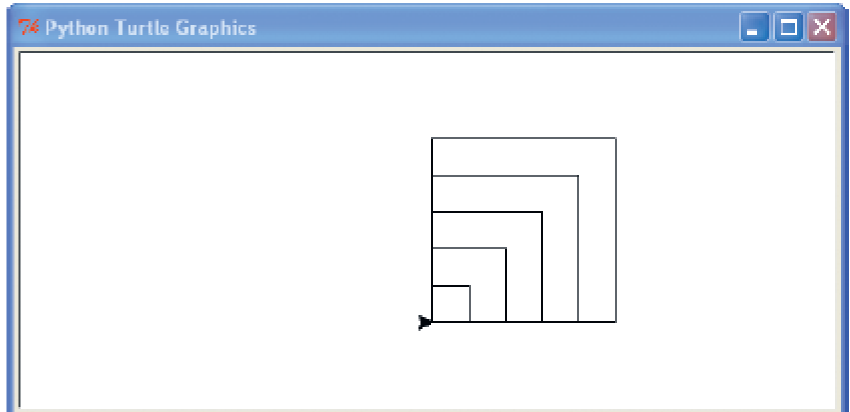
درنتیجه یک مربع کوچک خواهیم داشت



حال بیایید تابع را با سایزهای مختلف امتحان کنیم. کد زیر پنج مربع متوالی با سایز ۱۰۰،۷۵،۵۰،۲۵ و ۱۲۵ را تولید می‌کند:

```
>>> t.reset()
>>> mysquare(25)
>>> mysquare(50)
>>> mysquare(75)
>>> mysquare(100)
>>> mysquare(125)
```

درنتیجه خواهیم داشت



رسم مربع‌های توپر

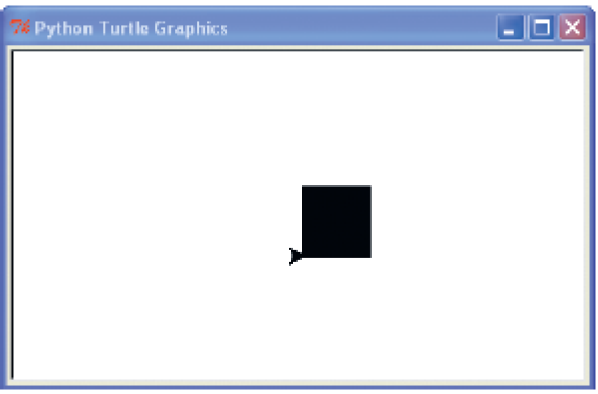
برای ترسیم یک مربع توپر،ابتدا باید بوم را بازنشانی(ری‌ست)کرده، پر کردن را آغاز نموده و سپس مجدداً تابع square را با استفاده از کد زیر فراخوانی کنیم:

```
>>> t.reset()
>>> t.begin_fill()
>>> mysquare(50)
```

تا قبل از تکمیل روند پر شدن، بایستی یک مربع توخالی مشاهده کنید:

```
>>> t.end_fill()
```

درنتیجه خواهیم داشت:



حال این تابع را به گونه‌ای تغییر می‌دهیم که بتوانیم مربع توپر یا توخالی دیگری رسم کنیم. برای این کار، به پارامتری دیگر و کدی که اندکی متفاوت است نیاز داریم:

```
>>> def mysquare(size, filled):
        if filled == True:
          t.begin_fill()
        for x in range(1, 5):
          t.forward(size)
          t.left(90)
        if filled == True:
          t.end_fill()
```

در خط اول، تعریف تابع را طوری تغییر داده تا بتواند دو پارامتر بگیرد: size و filled. سپس با if filled == True بررسی می‌کنیم که آیا مقدار filled برابر با True قرار داده شده است یا خیر. دراینصورت، begin_fill را فراخوانی کرده تا به لاک‌پشت بگوییم که باید شکلی که رسم کرده‌ایم، پر(رنگ) کند. سپس برای رسم چهار وجه مستطیل[1] چهار مرتبه حلقه می‌زنیم ((for x in range(0, 4)) (به جلو و چپ حرکت می‌کنیم) قبل از این که بازهم با if filled == True، True بودن filled را وراسی کنیم. دراینصورت، با دستور t.end_fill روند پر کردن را متوقف کرده و لاک‌پشت مربع[2] را با رنگ پر می‌کند.

با خط کد زیر میتوانیم یک مربع توپر رسم کنیم:

```
>>> mysquare(50, True)
```

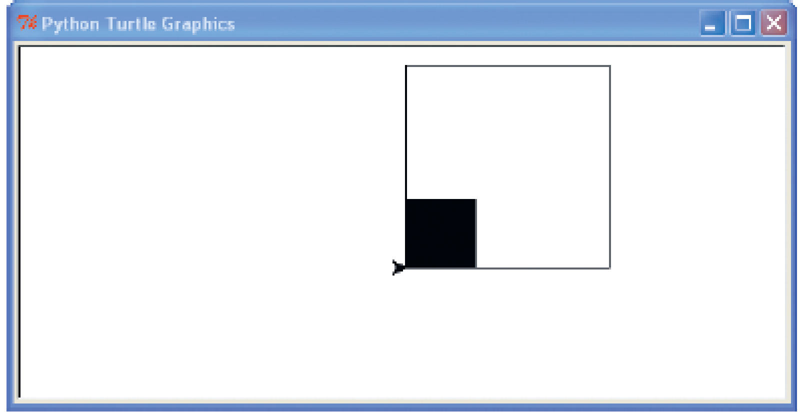
یا اینکه با خط کد زیر،یک مربع توخالی بکشیم:

---

[1] rectangle
[2] square

```
>>> mysquare(150, False)
```

بعد از این دو فراخوان تابع mysquare، تصویر زیر بدست خواهد آمد که اندکی شبیه یک چشم مربعی است.



ولی در اینجا کار متوقف نمی‌شود. میتوانید تمامی انواع شکل‌ها را کشیده و آنها را بـا زنـگ پـر کنید.

رسم ستاره‌های توپر

بعنوان آخرین مثال، ستاره‌ای که قبلاً کشیدیم را با رنگ پر می‌کنیم. کد اصلی بصورت زیر است:

```
for x in range(1, 19):
        t.forward(100)
        if x % 2 == 0:
            t.left(175)
        else:
            t.left(225)
```

اکنون یک تابع mystar می‌سازیم. از دستورات if تابع mystar استفاده کرده و پارامتر size را اضافه می‌کنیم.

```
>>> def mystar(size, filled):
        if filled == True:
            t.begin_fill()
        for x in range(1, 19):
            t.forward(size)
            if x % 2 == 0:
                t.left(175)
            else:
                t.left(225)
        if filled == True:
```

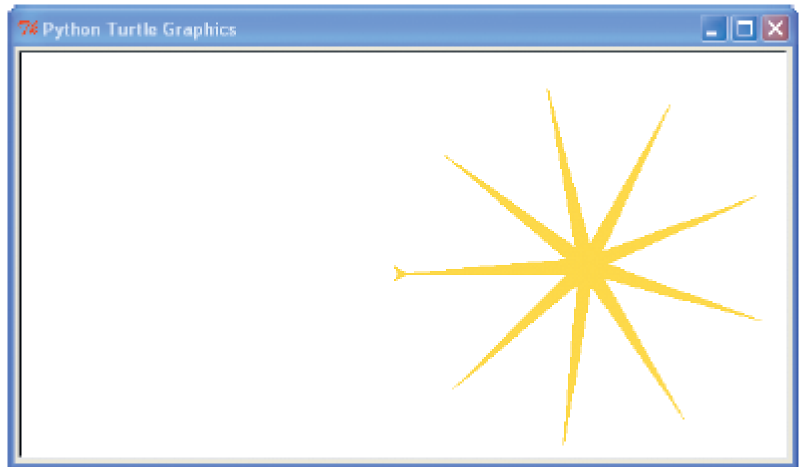
```
        t.end_fill()
```

در دو خط ابتدایی این تابع، True بودن filled را وارسی کرده و دراینصورت با پر کردن شکل کار را شروع می‌کنیم. بازهم در دو خط آخر همین وارسی را انجام داده و درصورت true بودن filled ، پر کردن را متوقف می‌کنیم. همانند تابع mysquare سایز ستاره را در پارامتر size قرار می‌دهیم و در هنگام فراخوانی t.forward از مقدار آن استفاده می‌کنیم.

در اینجا رنگ طلایی را انتخاب کرده(٪۹۰قرمز،٪۷۵سبز و ٪۰ آبی)و مجدداً تابع را صدا می‌زنیم:

```
>>> t.color(0.9, 0.75, 0)
>>> mystar(120, True)
```

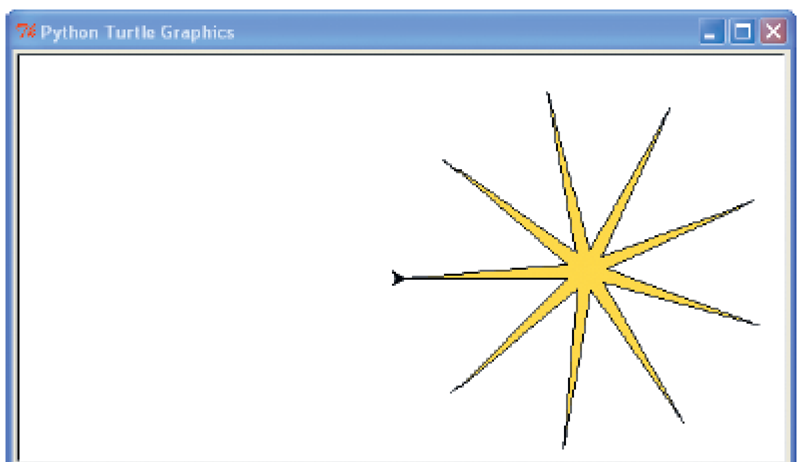
لاک‌پشت ستاره توپر زیر را رسم می‌کند:



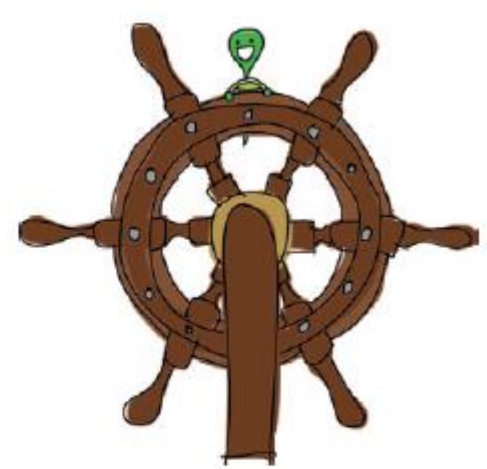
برای اینکه طرحی به ستاره بدهیم، رنگ را به سیاه تغییر داده و ستاره را توخالی رسم می‌کنیم:

```
>>> t.color(0,0,0)
>>> mystar(120, False)
```

اکنون یک ستاره طلایی با خطوط سیاه داریم:



آنچه آموختید



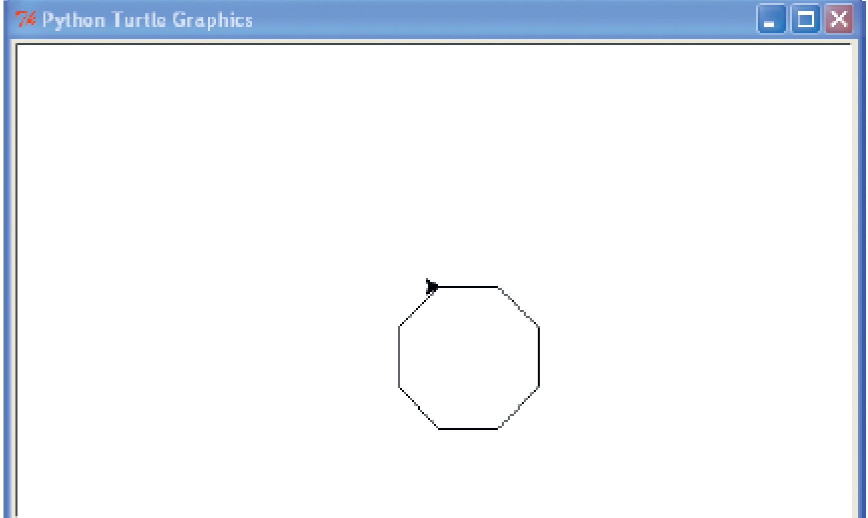
در این فصل، یاد گرفتید که چگونه از ماژول turtle برای رسم چند شکل هندسی پایه استفاده کرده و از حلقه‌های for و دستورات if برای کنترل کاری که لاک‌پشت روی صفحه انجام می‌دهد، بهره بگیرید. رنگ قلم لاک‌پشت را عوض کرده و شکل‌هایی که رسم می‌کند را پر می‌کنیم. همچنین از کد رسم در برخی توابع مجدداً استفاده کرده تا تنها با فراخوانی یک تابع، شکل‌هایی با رنگ‌های مختلف را ساده‌تر ترسیم کنیم.

(چیستان)معماهای برنامه‌نویسی

در آزمایشات زیر، شکل‌های دلخواهتان را با لاک‌پشت بکشید. همانند همیشه، جواب‌ها در *http:// python-for-kids.com/* دردسترس می‌باشند.

۱- رسم یک هشت‌وجهی[1]

ستاره، مربع و مشتطیل را در این فصل کشیدیم. نظرتان درباره رسم یک شکل هشت وجهی مانند یک هشت‌پر چیست؟ (تذکر: سعی کنید لاک‌پشت را ۴۵درجه بچرخانید)

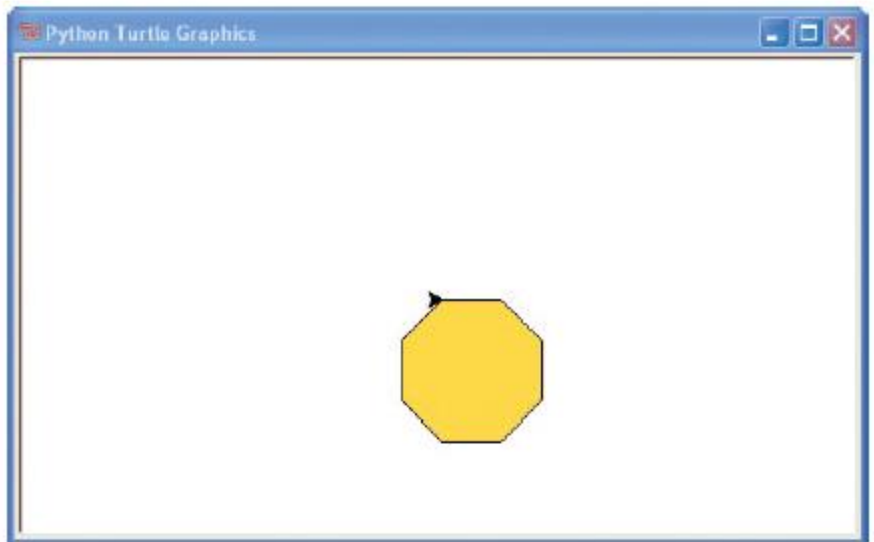


---

[1] Octagon هشت پر

۲- رسم یک هشت‌وجهی توپر

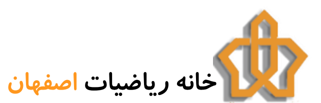
حالا که تابعی برای رسم یک هشت‌وجهی دارید، آن را طوری تغییر دهید تا یک هشت‌وجهی توپر بکشد. سعی کنید یک هشت‌وجهی با خطوط دور رسم کنید، همانطوریکه در خصوص ستاره انجام شد.



۳- یک تابع دیگر برای رسم ستاره

تابعی با دو پارامتر سایز و تعداد نقاط، برای رسم یک ستاره بنویسید. شروع تابع چیزی شبیه زیر خواهد بود:

```
def draw_star(size, points):
```



فصل ۱۲

استفاده از TKINTER برای گرافیک های بهتر

مشکل استفاده از یک لاک‌پشت برای ترسیم این است که لاک‌پشت‌ها خیلی کند هستند. حتی زمانی که لاک‌پشت با سرعت حرکت می‌کند، بازهم خیلی سریع نیست. این اتفاق برای لاک‌پشت‌ها مسئله‌ای نیست ولی برای نگاره‌سازی رایانه‌ای مسئله‌ساز می‌باشد.

نگاره‌سازی[1] رایانه‌ای بویژه در بازی‌ها، معمولاً بایستی به سرعت انجام شود. اگر یک کنسول بازی در اختیار دارید یا بازی رایانه‌ای انجام می‌دهید، لحظاتی به ترسیماتی که روی صفحه‌ی نمایش می‌بینید، فکر کنید. گرافیک‌های ۲بعدی(2D) مسطح هستند- کاراکترها معمولاً فقط بالا و پایین یا چپ و راست حرکت می‌کنند- همانند بسیاری از بازی‌های Nintendo DS, PlayStation Portable (PSP) و بازی‌های تلفن همراه. در بازی‌های شبه ۳بعدی(3D)- بازی‌هایی که تقریباً ۳بعدی هستند- تصاویر واقعی‌تر به نظر رسیده ولی کاراکترها فقط نسبت به صفحه‌ی صاف حرکت می‌کنند(این مورد تحت عنوان نگاره‌های ایزومتریک شناخته می‌شود). و در نهایت، بازی‌های ۳بعدی هستند که در آنها در تلاش برای تقلید از واقعیت، تصاویر روی صفحه نمایش کشیده شده‌اند. خواه در بازی از نگاره‌های ۲بعدی، شبه-۳بعدی یا ۳بعدی استفاده شده باشد، همگی یک وجه مشترک دارند: بایستی به سرعت روی صفحه‌ی نمایش کشیده شوند.

اگر هیچ وقت سعی نکردید انیمیشنی بسازید، این پروژه‌ی ساده را امتحان کنید:

[1] graphics

۱. یک دفترچه یادداشت خالی پیدا کرده و در گوشه‌ی پایین صفحه‌ی اول آن چیزی بکشید (مثلاً عکس یک عصا)

۲. درگوشه‌ی صفحه‌ی بعد، همان شکل عصا را مجدداً بکشید ولی این بار اندکی پایین آن را جابجا کنید

۳. در صفحه‌ی بعد همین کار را تکرار کرده و این بار بازهم اندکی پایین آن را جابجا کنید

٤. همین کار را در چندین صفحه تکرار کرده و هر بار پای عصا را کمی بیشتر جابجا کنید

بعد از خاتمه‌ی کار، صفحات را به سرعت ورق بزنید و با این کار بایستی شاهد حرکت کردن عصا باشید. این شیوه‌ای اصلی مورداستفاده در تمامی انیمیشن(پویانمایی)ها می‌باشد، چه تلویزیون، کارتون یا بازی رایانه‌ای و کنسول. تصویری کشیده شده است و سپس با اندکی تغییر تصویر دیگری کشید شده تا حرکت مجسم گردد. برای این که یک تصویر متحرک به نظر برسد، بایستی به سرعت هر فریم[1]یا بخشی از انیمیشن را نمایش دهید.

پایتون راه‌های متفاوتی را برای خلق نگاره‌ها ارائه می‌کند. شما علاوه بر ماژول turtle، می‌توانید از ماژول‌های خارجی (که بایستی بطور جداگانه نصب شوند) و ماژول tkinter که بایستی در نصب استاندرد پایتون دراختیار داشته باشید، استفاده کنید. می‌توانید از tkinter برای ساخت اپلیکیشن‌های کاملی از جمله واژه‌پرداز[2]و ترسیمات ساده استفاده نمایید. در این فصل، به کاوش این موضوع می-پردازیم که چگونه می‌توان از tkinter برای ساخت نگاره‌ها استفاده نمود.

خلق یک دکمه‌ی قابل کلیک

برای اولین مثال، از tkinter برای ساخت یک اپلیکیشن پایه با یک دکمه استفاده می‌کنیم. کد زیر را وارد کنید:

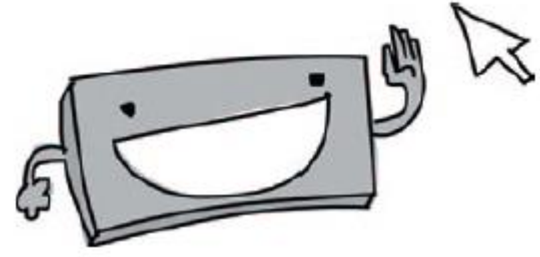
```
>>> from tkinter import *
>>> tk = Tk()
>>> btn = Button(tk, text="click me")
>>> btn.pack()
```

در خط اول، محتوای ماژول tkinter را وارد می‌کنیم. استفاده از* from *module-name* import به ما اجازه می‌دهد تا از محتویات یک ماژول استفاده کنیم بدون این که از نام آن استفاده نماییم. درمقابل،

[1] frame
[2] Word-processor

زمانی که در مثال‌های قبل از import turtle استفاده می‌کنیم، برای دسترسی به محتوای آن بایستی نام ماژول را نیز بکار ببریم.

```
import turtle
t = turtle.Pen()
```



هنگام استفاده از * import ، همانند کاری که در فصول ٤و ١١ انجام دادیم، نیازی به فراخوانی turtle.Pen نیست. درخصوص ماژول **turtle** چندان کاربردی ندارد ولی زمانی که از ماژول‌هایی با چندین کلاس و تابع استفاده می-کنید، بدلیل این که کمتر باید تایپ کنید، مفید خواهد بود.

```
from turtle import *
t = Pen()
```

در خط بعد در مثال زیر، دقیقاً همانند ایجاد یک شیء **Pen** برای **turtle**، متغیری حاوی یک شیء از کلاس **tk** را با tk = Tk() ایجاد می‌کنیم. شیء **tk** یک پنجره‌ی اصلی را می‌سازد که می‌توانیم چیزهای دیگری از قبیل دکمه‌ها، کادرهای ورودی یا یک بوم برای ترسیم را به آن اضافه کنیم. این همان کلاس اصلی است که توسط ماژول **tkinter** ساخته شده است– بدون ساختن شیء‌ای از کلاس **Tk** قادر نخواهید بود ترسیمات یا پویانمایی‌هایی را خلق کنید.

در خط سوم، با btn = Button دکمه‌ای را ایجاد کرده و متغیر **tk** را بعنوان پارامتر اول و "click me" را بعنوان متنی که توسط دکمه و با (tk, text="click me") نمایش داده خواهد شد، درنظر می‌گیریم.

هرچند این دکمه را به پنجره اضافه کرده‌ایم ولی تا قبل از این که خط btn.pack() را وارد نکنیم، نمایش داده نخواهد شد زیرا این خط به دکمه فرمان می‌دهد تا ظاهر شود. نتیجه بایستی چیزی شبیه به این باشد:

دکمه‌ی *click me* کار زیادی انجام نمی‌دهد. شما هر روز می‌توانید روی آن کلیک کنید ولی تا زمانی که کد را اندکی تغییر ندهیم، اتاقی نخواهد افتاد. (از بسته بودن پنجره‌ای که قبلاً ساخته‌اید، مطمئن شوید!)

ابتدا، برای چاپ یک متن، تابعی را ایجاد می‌کنیم:

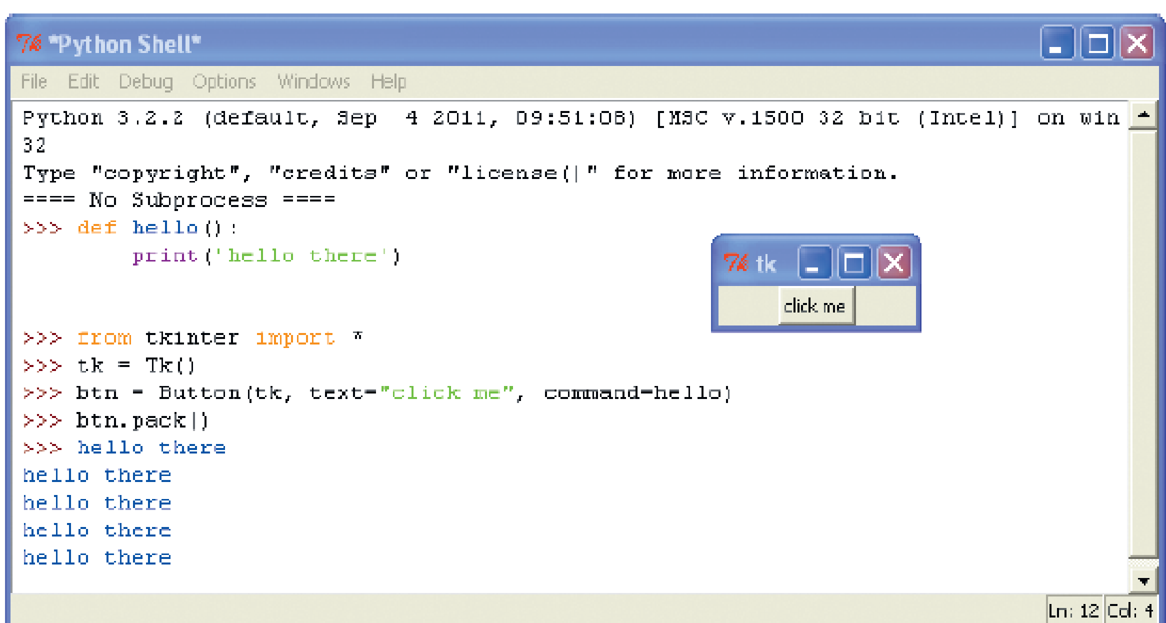
```
>>> def hello():
```

```
print('hello there')
```

سپس برای استفاده از این تابع جدید، مثال را تغییر می‌دهیم:

```
>>> from tkinter import *
>>> tk = Tk()
>>> btn = Button(tk, text="click me", command=hello)
>>> btn.pack()
```

توجه داشته باشید که تغییر بسیار کمی در نسخه‌ی قبلی این کد ایجاد کرده‌ایم: پارامتر command را اضافه کرده‌ایم که به پایتون می‌گوید به مجرد کلیک شدن دکمه، از تابع hello استفاده کند. حال، زمانی که روی دکمه کلیک می‌کنید، مشاهده خواهید کرد که "hello there" در شِل نوشته شده است. در مثال زیر، ۵ مرتبه روی دکمه کلیک کرده‌ایم:



این اولین باری است که از پارامترهای بانام در نمونه کدها استفاده کرده‌ایم بنابراین اجازه دهید قبل از ادامه‌ی ترسیمات خود، بیشتر درباره‌ی آن توضیح دهیم.

استفاده از پارامترهای بانام[1]

پارامترهای بانام همانند پارامترهای معمول هستند با این تفاوت که به جای استفاده از ترتیب خاصی از مقادیر تابعی که تعیین می‌کند کدام مقدار مربوط به کدام پارامتر است (مقدار اول، اولین

[1] Named مشخص،مشهور

پارامتر؛ مقدار دوم، دومین پارامتر؛ مقدرا سوم، سومین پارامترو ....)، مقادیر را نامگذاری کرده و درنتیجه می‌توانند به هر ترتیبی ظاهر شوند.

برخی اوقات، توابع پارامترهای زیادی دارند و همواره نیازی نیست برای هر پارامتر مقداری را تعیین کنیم. پارامترهای با نام، راهی است برای تعیین مقادیر برای پارامترهایی که بایستی به آنها مقدار بدهیم.

بعنوان مثال فرض کنید تابعی به نام person با دو پارامتر width و height داریم.

```
>>> def person(width, height):
        print('I am %s feet wide, %s feet high' % (width, height))
```

بطور معمول، تابع را اینگونه صدا می‌زنیم:

```
>>> person(4, 3)
I am 4 feet wide, 3 feet high
```

با استفاده از پارامترهای بانام می‌توانیم این تابع را فراخوانده و نام پارامتر را با هر مقدار تعیین کنیم.

```
>>> person(height=3, width=4)
I am 4 feet wide, 3 feet high
```

پارامترهای بانام در هنگام کار با ماژول tkinter بسیار مفید هستند.

ساخت یک بوم برای ترسیم

دکمه‌های ابزارهای سودمندی هستند ولی زمانی که می‌خواهیم چیزی را روی صفحه ترسیم کنیم، به درد نمی‌خورند. وقتی واقعاً می‌خواهیم چیزی بکشیم، به جزء دیگری نیاز داریم: یک شیءcanvas که یک شیء از کلاس Canvas می‌باشد (توسط ماژول tkinter تولید شده است).

زمانی که یک بوم می‌سازیم، طول و عرض بوم(برحسب پیکسل)را به پایتون می‌دهیم. درغیراینصورت، کد مشابه کد دکمه خواهد بود. بعنوان مثال:

```
>>> from tkinter import *
>>> tk = Tk()
>>> canvas = Canvas(tk, width=500, height=500)
>>> canvas.pack()
```

همانند مثال دکمه، زمانی که tk = Tk() را وارد می‌کنید پنجره‌ای ظاهر می‌گردد. در خط آخر، بوم را با دستور canvas.pack() فشرده می‌کنیم جایی که سایز بوم به طول ۵۰۰ و

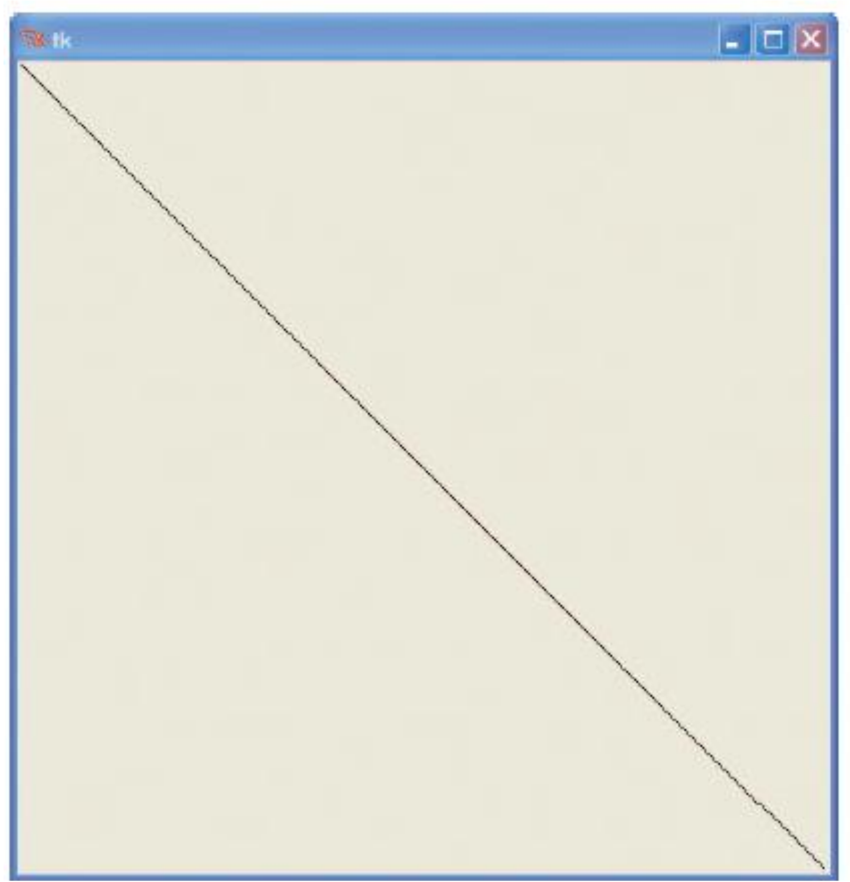
عرض ۵۰۰ پیکسلی که در سومین خط کد تعریف شده است، تغییر خواهد کرد.

همانند مثال دکمه، تابع pack به بوم دستور می‌دهد تا خود را در موقعیت صحیحی در پنجره نمایش دهد. درصورتی که این تابع فراخوانی نشده باشد، چیزی بدرستی نمایش داده نخواهد شد.

ترسیم خطوط

برای کشیدن یک خط روی بوم، از مختصات پیکسل استفاده می‌کنیم. *Coordinates* موقعیت پیکسل‌ها روی یک سطح را تعیین می‌کند. در بوم tkinter، مختصات نشان می‌دهند که پیکسل در چه موقعیتی در عرض بوم (از چپ به راست) و در چه موقعیتی در طول بوم (از بالا به پایین) قرار گرفته است.

بعنوان مثال چون بوم ما ۵۰۰×۵۰۰ پیکسل است، مختصات گوشه‌ی پایین-راست صفحه (500,500) می‌باشد. برای کشیدن خطی که در تصویر زیر نشان داده شده است، از مختصات شروع (0,0) و مختصات پایان (500,500) استفاده خواهیم کرد.



مختصات را با استفاده از تابع create_line بصورت زیر تعریف می‌کنیم:

```
>>> from tkinter import *
```

```
>>> tk = Tk()
>>> canvas = Canvas(tk, width=500, height=500)
>>> canvas.pack()
>>> canvas.create_line(0, 0, 500, 500)
1
```

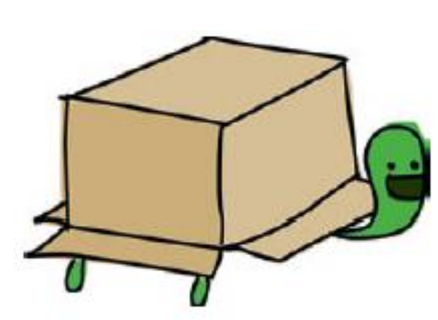
تابع create_line، 1 را برمی‌گرداند که یک شناسه است و بعداً بیشتر درباره‌ی آن خواهیم آموخت.

اگر همین کار را با ماژول turtle انجام داده باشیم، به کد زیر نیاز خواهیم داشت

```
>>> import turtle
>>> turtle.setup(width=500, height=500)
>>> t = turtle.Pen()
>>> t.up()
>>> t.goto(-250, 250)
>>> t.down()
>>> t.goto(500, -500)
```

بنابراین کد tkinter یک اصلاح محسوب می‌شود. این کد کوتاهتر و اندکی ساده‌تر است.

حال به برخی از توابع دردسترس در شیء ccanvas نگاهی می‌اندازیم که می‌توانیم برای ترسیمات جالب‌تر از آن‌ها استفاده کنیم.



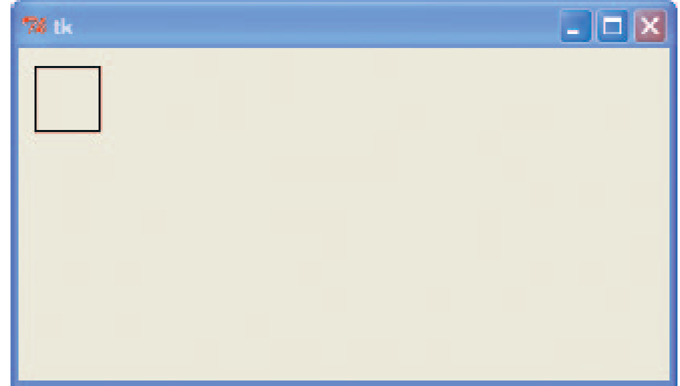
کشیدن جعبه

با کمک ماژول turtle به این صورت یک جعبه می‌کشیم: حرکت به جلو، برگشتن، رفتن به جلو، بازهم دورزدن و به همین ترتیب تکرار می‌کنیم. درنهایت با تغییر میزان حرکت خود می‌توانیم یک جعبه مربعی یا مستطیلی بکشیم.

ماژول tkinter ترسیم یک مربع یا مستطیل را ساده‌تر می‌نماید. تنها چیزی که باید بدانید، مختصات گوشه‌ها می‌باشد. در اینجا مثالی ارائه شده است(اکنون می‌توانید بقیه‌ی پنجره‌ها را ببندید):

```
>>> from tkinter import *
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> canvas.create_rectangle(10, 10, 50, 50)
```

در این کد، از tkinter برای ساخت یک بوم به ابعاد ٤٠٠×٤٠٠ پیکسل استفاده کرده و سپس در گوشه‌ی بالا–چپ پنجره یک مربع می‌کشیم:

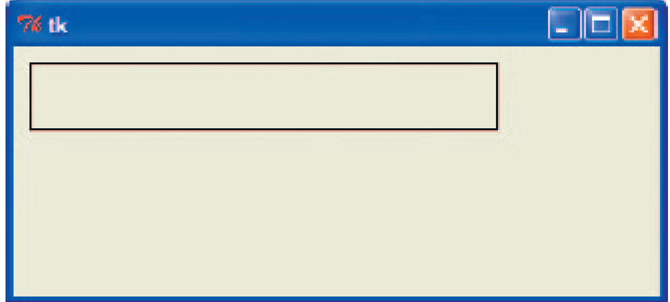


پارامترهایی که در آخرین خط کد در canvas.create_rectangle قرار می‌دهیم، مختصات گوشه‌های بالا-چپ و پایین-راست مربع می‌باشند. این مختصات را بعنوان فاصله از وجه دست چپ بوم و فاصله تا بالای بوم ارائه می‌کنیم.در این مورد، دو مختصات اول(گوشه‌ی بالا چپ) ۱۰پیکسل از چپ و ۱۰ پیکسل از بالا(اعداد اول:10,10) می‌باشند. گوشه‌ی پایین-راست مربع، ۵۰ پیکسل از چپ و ۵۰پیکسل از پایین فاصله دارد (دومین اعداد:50,50).

این دو مجموعه مختصات را با x1,y1 و x2,y2 نشان می‌دهیم. برای رسم یک مستطیل، می‌توانیم فاصله‌ی گوشه‌ی دوم از لبه‌ی بوم را افزیش دهیم (افزایش مقدار پارامتر x2):

```
>>> from tkinter import *
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> canvas.create_rectangle(10, 10, 300, 50)
```

در این مثال، مختصات بالا-چپ مستطیل(موقعیت آن در صفحه)،(10,10) و مختصات پایین-راست،(300,50) می‌باشند. درنتیجه یک مستطیل داریم که عرض آن با مربع اصلی برابر است(۵۰یکسل) ولی طول آن خیلی بیشتر است.



همچنین با افزایش اصله‌ی گوشه‌ی دوم از بالای بوم نیز می‌توان یک مستطیل رسم نمود (افزایش مقدار پارامتر y2):

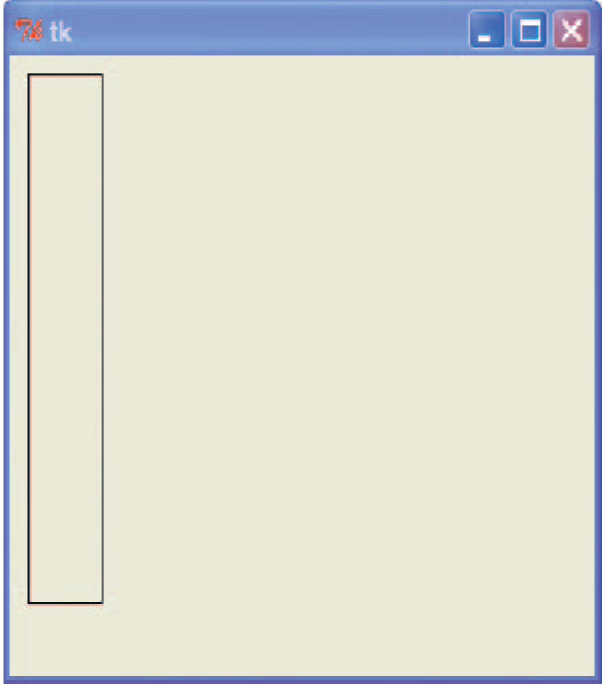
```
>>> from tkinter import *
```

```
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> canvas.create_rectangle(10, 10, 50, 300)
```

در این فراخوان تابع create_rectangle، به ترتیب فرامین زیر داده می‌شود:

- ۱۰ پیکسل در عرض بوم حرکت کن(از بالا چپ)
- ۱۰ پیکسل به سمت پایین بوم برو. این گوشه‌ی شروع مستطیل است.
- مستطیل را تا۵۰ پیکسل در عرض بکش
- تا ۳۰۰ پیکسل به پایین رسم کن

نتیجه‌ی نهایی بایستی چیزی شبیه به این باشد



کشیدن چند مستطیل بیشتر

نظرتون درباره‌ی این که بوم را پر از مستطیل‌هایی با سایزهای مختلف بکنیم، چیست؟

ما می‌توانیم این کار را با وارد کردن ماژول random و سپس ایجاد تابعی که از یک عدد تصادفی برای مختصات در گوشه‌ی بالا–چپ و پایین–راست مستطیل استفاده می‌کند، انجام دهیم.

از تابع randrange که توسط ماژول random بدست آمده است، استفاده خواهیم کرد. زمانی که یک عدد به این تابع می‌دهیم، یک عدد تصادفی بین 0 و عددی که به آن داده بودیم، برمی‌گرداند. بعنوان

مثال، فراخواندن randrange(10) یک عدد بین 0 و 9 را برگردانده و randrange(100) عددی بین 0 و 99 را برگردانده و به همین ترتیب.

در اینجا می‌بینیم که چگونه از randrange در یک تابع استفاده خواهیم کرد. با انتخاب **File ‣ New Window** یک پنجره‌ی جدید ساخته و کد زیر را وارد می‌کنیم:

```
from tkinter import *
import random
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
def random_rectangle(width, height):
        x1 = random.randrange(width)
        y1 = random.randrange(height)
        x2 = x1 + random.randrange(width)
        y2 = y1 + random.randrange(height)
        canvas.create_rectangle(x1, y1, x2, y2)
```

ابتدا تابع خود (def random_rectangle) را با دو پارامتر width و height تعریف می‌کنیم. سپس، با استفاده از تابع randrange متغیرهایی را برای گوشه‌ی بالا–چپ مستطیل ایجاد کرده و طول و عرض را به ترتیب بصورت پارامترهایی با x1 = random.randrange(width) و y1 = random.randrange(height) تعریف می‌کنیم. درعمل، در خط دوم این تابع می‌گوییم «متغیری به نام x1 را ساخته و مقادیر آن را برابر با عدد تصادفی بین 0 و مقدار پارامتر width قرار بده».

دو خط بعد، با درنظر گرفتن مختصات بالا–چپ (x1 یا y1) و افزودن عدد تصادفی به این مقادیر، متغیرهایی را برای گوشه‌ی پایین–راست مستطیل می‌سازند. خط سوم، تابع می‌گوید «متغیر x2 را با افزودن عدد تصادفی به مقداری که برای x1 محاسبه کرده‌ایم،ایجاد کن»

درنهایت با canvas.create_rectangle از متغیر x1, y1, x2, و y2 برای ترسیم مستطیل روی بوم استفاده می‌کنیم.

برای امتحان کردن تابع random_rectangle، طول و عرض بوم را به آن می‌دهیم. کد زیر را بعد از تابعی که وارد کرده‌اید، اضافه کنید:

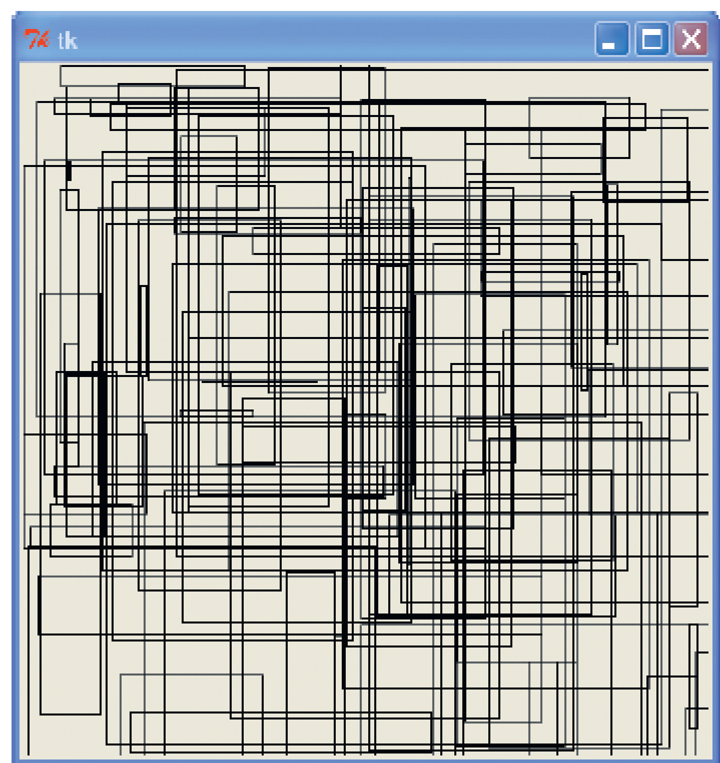
```
random_rectangle(400, 400)
```

کدی که وارد کرده‌اید را ذخیره کنید(**File ‣ Save** را انتخاب کرده و نام فایل را وارد کنید مثلاً (*randomrect.py* و سپس **Run ‣ Run Module** را انتخاب نمایید. اگر دید که تابع جواب می‌دهد، با ایجاد یک حلقه برای چندین بار فراخوانی random_rectangle ، صفحه را با مستطیل‌ها پر کنید. یک

حلقه‌ی for را برای ۱۰۰ مستطیل تصادفی امتحان کنید. کد زیر را وارد کرده، کار خود را ذخیره کرده و مجدداً آن را اجرا کنید:

```
for x in range(0, 100):
random_rectangle(400, 400)
```

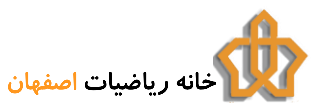
این کد یک تصویر شلوغ و درهم ایجاد می‌کند ولی نباید آن را با هنر مدرن اشتباه گرفت:



تنظیم رنگ

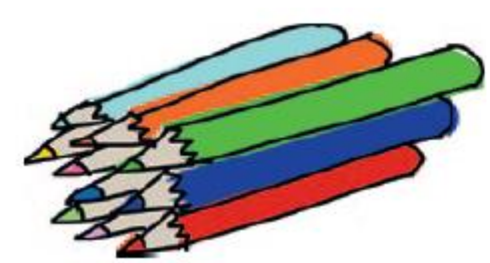
البته می‌خواهیم ترسیمات خود را رنگ کنیم. اجازه دهید با تغییر تابع random_rectangle یک رنگ را بعنوان یک پارامتر اضافی (fill_color) برای مستطیل تعیین کنیم. این کد را در پنجره‌ی جدیدی وارد کرده و زمانی که آن را ذخیره کردید، فایل *colorrect.py* را فراخوانی کنید:

```
from tkinter import *
import random
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
def random_rectangle(width, height, fill_color):
    x1 = random.randrange(width)
    y1 = random.randrange(height)
    x2 = random.randrange(x1 + random.randrange(width))
```




```
y2 = random.randrange(y1 + random.randrange(height))
canvas.create_rectangle(x1, y1, x2, y2, fill=fill_color)
```

اکنون، تابع create_rectangle پارامتر fill_color را می‌گیرد که رنگ را برای استفاده در هنگام ترسیم مستطیل تعیین می‌نماید.



برای ساختن چندین مستطیل با رنگ‌های مختلف، رنگ‌های نام-دار را در تابع قرار می‌دهیم (با استفاده از بوم ۴۰۰×۴۰۰ پیکسل). با انجام این مثال، برای این که کمتر تایپ کنید، از کپی و درج[1] استفاده خواهید کرد. برای این کار، متنی که میخواهید کپی کنید را انتخاب کرده، برای کپی کردن آن Ctrl+c را فشار دهید، روی یک خط خالی کلیک کرده و برای درج آن، Ctrl+v را فشار دهید. با استفاده از تابع زیر این کد را به *colorrect.py* اضافه کنید:

```
random_rectangle(400, 400, 'green')
random_rectangle(400, 400, 'red')
random_rectangle(400, 400, 'blue')
random_rectangle(400, 400, 'orange')
random_rectangle(400, 400, 'yellow')
random_rectangle(400, 400, 'pink')
random_rectangle(400, 400, 'purple')
random_rectangle(400, 400, 'violet')
random_rectangle(400, 400, 'magenta')
random_rectangle(400, 400, 'cyan')
```

بسیاری از این رنگ‌های نام‌دار(مشخص) رنگی را نمایش می‌دهند که موردانتظار شما می‌باشد ولی بقیه‌ی رنگ‌ها ممکن است پیام خطایی را تولید کنند (به سیستم عامل شما بستگی دارد: Windows, Mac OS X یا Linux).

ولی اگر رنگ دلخواهی به جز رنگ‌های نام‌دار(مشخص) استفاده شود چه اتفاقی خواهد افتاد؟ به فصل ۱۱ رجوع می‌کنیم، جایی که رنگ قلم لاک‌شت را با استفاده از درصد رنگ‌های قرمز، آبی و سبز تعیین کردیم. تعیین مقدار هر رنگ اولیه (قرمز،سبز،آبی) برای استفاده در یک ترکیب رنگی با tkinter اندکی پیچیده‌تر است ولی با آن کار خواهیم کرد:

زمانیکه با ماژول turtle کار می‌کنیم، با استفاده از ۹۰٪ قرمز، ۷۵٪ سبز و بدون رنگ آبی، رنگ طلایی را می‌سازیم. در tkinter می‌توانیم همان رنگ طلایی را با استفاده از خط زیر نیز بسازیم:

```
random_rectangle(400, 400, '#ffd800')
```

---

[1] Copy-paste

علامت هَش (#)قبل از مقدارffd800 به پایتون می‌گوید که یـک عـدد هگزادسـیمال[1]مـی‌سـازیم. هگزادسیمال راهی است برای نمایش اعدادی که بطورمعمول در برنامه‌نویسی رایانه‌ای استفاده می‌شـوند. در اینجا از مبنای۱٦(0 تا9 و سپس A تاF)به جای دسیمال[2]استفاده می‌شـود کـه مبنـای۱۰(از0 تـا9) دارد. اگر چیزی درباره‌ی مبناها در ریاضیات نمی‌دانید، فقط کافی است بدانید که بسادگی و با استفاده از یـک متغیرجایگزین قالب[3]در یک رشته می‌توانید یک عدد نرمال دسیمال را به هگزادسـیمال تبـدیل کنیـد: %x (به «جاسازی مقادیر در رشته‌ها» در صفحه‌ی ۳۰ رجوع کنید). بعنوان مثال، برای تبدیل عدد دسیمال15 به هگزادسیمال، می‌توانید این کار را انجام دهید:

```
>>> print('%x' % 15)
f
```

برای این که مطمئن شویم عددمان حداقل دو رقمی است، می‌توانیم اندکی متغیرجایگزین قالـب را تغییر دهیم، برای این کار:

```
>>> print('%02x' % 15)
0f
```

ماژولtkinter راه ساده‌ای را برای رسیدن به مقدار رنگ هگزادسیمال ارائه می‌نماید. کـد زیـر را به*colorrect.py* اضافه کنید(می‌توانید بقیه فراخوانی‌ها به تابع random_rectangle را حذف نمایید):

```
from tkinter import *
colorchooser.askcolor()
```

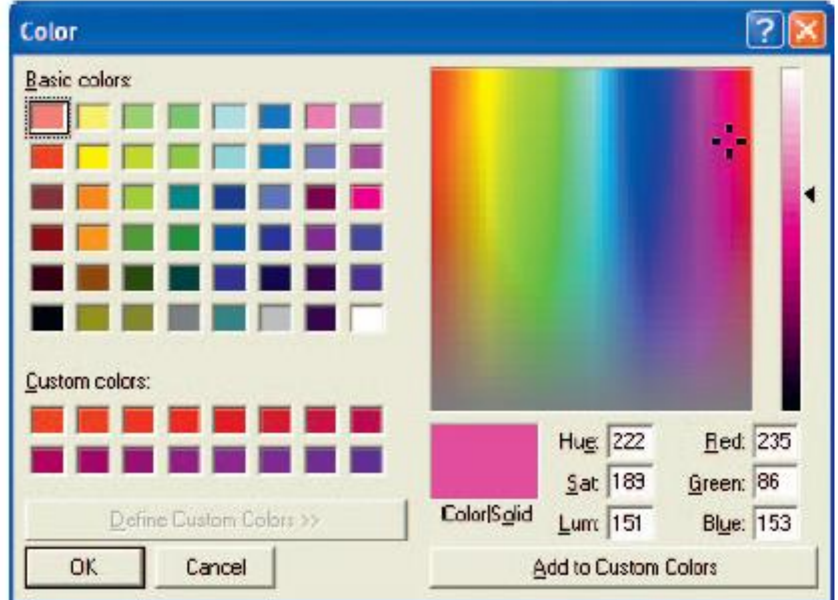
در نتیجه یک پنجره‌ی انتخاب رنگ برای شما ظاهر خواهد شد:

---

[1] Hexadecimal مبنای۱٦

[2] مبنای ۱۰

[3] Format placeholder



زمانی که رنگی را انتخاب کرده و روی **OK** کلیک می‌کنید، یک چندتایی نمایش داده خواهد شد. این چندتایی حاوی یک چندتایی دیگر با سه عدد و یک رشته است:

```
>>> colorchooser.askcolor()
((235.91796875, 86.3359375, 153.59765625), '#eb5699')
```

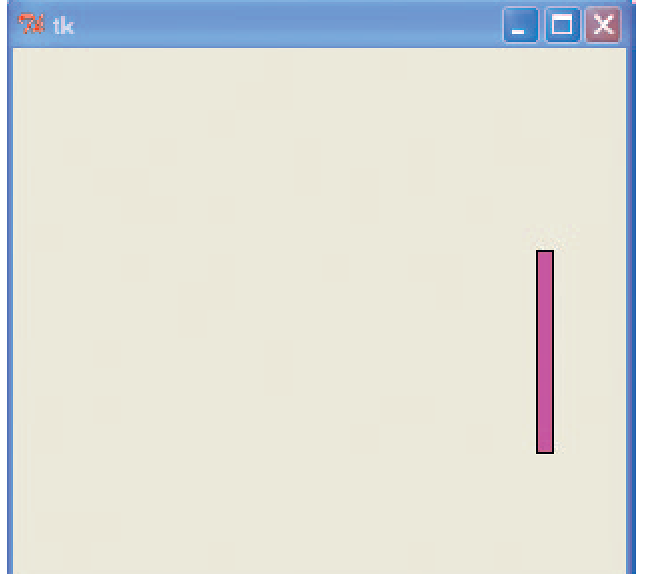
سه عدد معرف رنگ‌های قرمز، سبز و آبی هستند. در tkinter، مقدار رنگ اولیه برای این که در یک ترکیب رنگی استفاده شود، با عددی بین 0 و 255 نمایش داده می‌شود (که با استفاده از یک درصد برای هر رنگ اولیه در ماژول turtle فرق دارد). رشته در چندتایی حاوی نسخه‌ی هگزادسیمال این سه عدد است.

برای استفاده از یا ذخیره‌ی چندتایی بصورت یک متغیر، می‌توانید مقدار رشته را کپی و درج کرده و سپس از موقعیت اندیس مقدار هگزادسیمال استفاده کنید.

در اینجا از تابع random_rectangle برای مشاهده‌ی روند این کار استفاده می‌کنیم.

```
>>> c = colorchooser.askcolor()
>>> random_rectangle(400, 400, c[1])
```

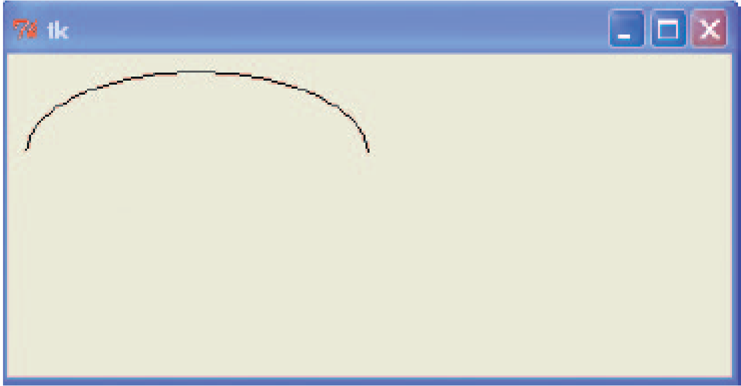
در اینجا نتیجه را می‌بینیم:



ترسیم کمان[1]

یک کمان، تکه‌ای از محیط یک دایره یا هر منحنی دیگری است ولی برای کشیدن یک کمان باtkinter، بایستی با استفاده از تابع create_arc و کدی مشابه کد زیر، آن را درون یک مستطیل بکشید:

```
canvas.create_arc(10, 10, 200, 100, extent=180, style=ARC)
```



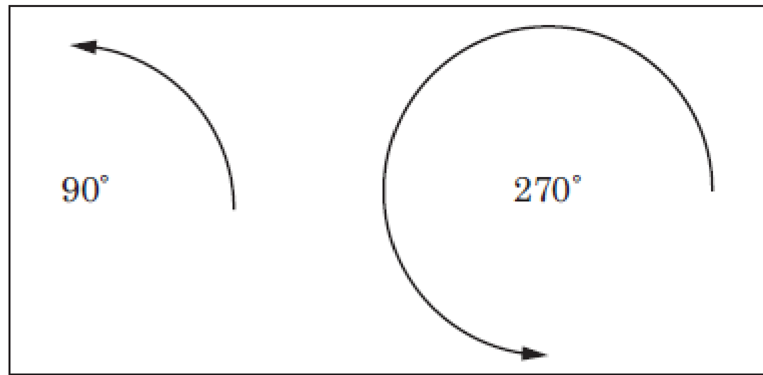
اگر تمامی پنجره‌هایtkinter را بسته یا IDLE را مجدداً اجرا کرده‌اید، مطمئن شوید که tkinter را مجدداً وارد(ایمپورت) کرده و بوم را با کد زیر مجدداً ساخته‌اید:

```
>>> from tkinter import *
```

[1] Arc

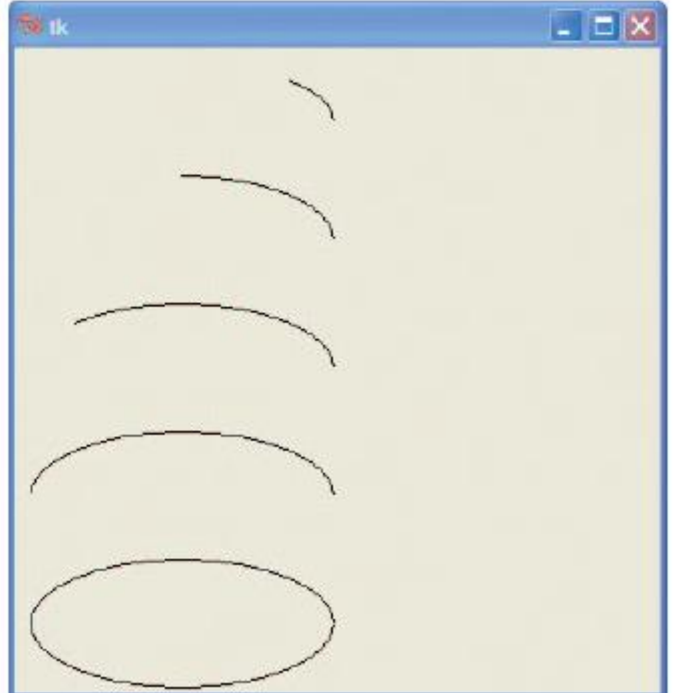
```
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> canvas.create_arc(10, 10, 200, 100, extent=180, style=ARC)
```

این کد، محل گوشه‌ی بالا-چپ مستطیل را تعیین می‌کند که حاوی یک کمان به مختصات(10,10) به فاصله‌ی ۱۰ پیکسل از پایین و گوشه‌ی پایین-راست به مختصات(200,100) یا ۲۰۰پیکسل از کنار و ۱۰۰ پیکسل از پایین می‌باشد. از پارامتر بعدی یعنی extent برای تعین درجه‌ی زاویه‌ی کمان استفاده می‌شود. با ارجاع به فصل ٤ خاطرنشان می‌کنیم که درجات راهی برای اندازه‌گیری فاصله برای حرکت دور یک دایره هستند. در اینجا مثال‌هایی از دو کمان ارائه شده است که در آنها ٤٥درجه و ۲۷۰ درجه دور یک دایره حرکت می‌کنیم:



کد زیر، چندین کمان مختلف را در پایین صفحه ترسیم می‌کند بطوریکه خواهید دید در هنگام استفاده از درجات مختف با تابع create_arc چه اتفاقی خواهد افتاد.

```
>>> from tkinter import *
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> canvas.create_arc(10, 10, 200, 80, extent=45, style=ARC)
>>> canvas.create_arc(10, 80, 200, 160, extent=90, style=ARC)
>>> canvas.create_arc(10, 160, 200, 240, extent=135, style=ARC)
>>> canvas.create_arc(10, 240, 200, 320, extent=180, style=ARC)
>>> canvas.create_arc(10, 320, 200, 400, extent=359, style=ARC)
```



نکته

به جای ۳۶۰ درجه، از ۳۵۹ درجه در دایره‌ی نهایی استفاده می‌کنیم زیرا tkinter، ۳۶۰درجه را همان صفر درجه درنظر می‌گیرد و اگر از ۳۶۰ استفاده کنیم، هیچ چیزی رسم نخواهد شد.

ترسیم چندضلعی[1]

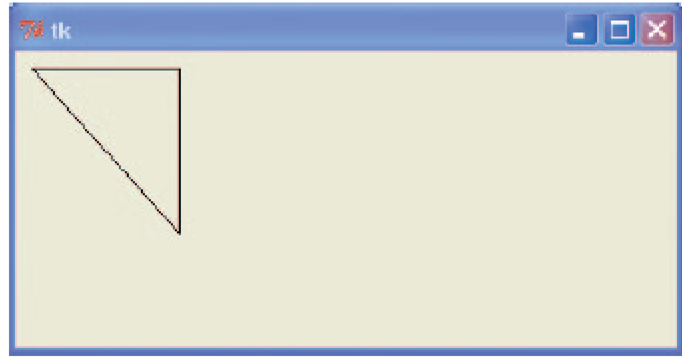
یک چندضلعی، شکلی است با سه یا چند وجه. چندضلعی‌های منظمی مانند مستطیل‌ها، مربع‌ها، مثلث‌ها، پنج‌ضلعی‌ها، هشت‌ضلعی‌ها و ... و چندضلعی‌های نامنظمی با اضلاع ناهمسان، وجوه زیاد و اشکال عجیب و غریب وجود دارند.

زمانیکه یک چندضلعی را با tkinter می‌کشید بایستی مختصات چند نقطه از چندضلعی را تعیین کنید. در اینجا نشان داده شده است که چگونه می‌توانید یک مثلث بکشید:

```
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
canvas.create_polygon(10, 10, 100, 10, 100, 110, fill="",
outline="black")
```

[1] Polygon چندوجهی

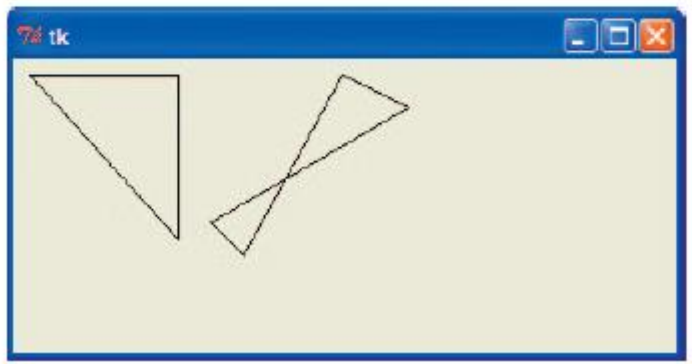
در این مثال، یک مثلث از مختصات x و y (10,10) آغاز شده، در عرض به(100,10) حرکت کرده و درنهایت در(100,110) خاتمه می‌یابد. در انیجا نتیجه را مشاهده می‌کنید:



با استفاده ازکد زیر می‌توانیم چندضلعی نامنظم دیگری را اضافه کنید:

```
canvas.create_polygon(200, 10, 240, 30, 120, 100, 140, 120, fill="",
outline="black")
```

این کد با مختصات(200,10) آغاز شده، به(240,30) رفته و سپس به(120,200) حرکت می‌کند و درنهایت به(100,140) می‌رود. Tkinter بطورخودکار خط را به مختصات اول متصل می‌کند و در اینجا نتیجه‌ی اجرای کد را می‌بینیم:



نمایش متن

علاوه بر کشیدن شکل‌ها، با استفاده از create_text می‌توانید روی بوم چیز بنویسید. این تابع تنها دو مختصات را (موقعیت x و y متن) را همراه را یک پارامتر با نام برای نمایش متن در نظر می‌گیرد. در کد زیر، بوم خود را همانند قبل ساخته و سپس جمله‌ای را نمایش می‌دهیم که در مختصات(150,100) قرار گرفته است. این کد را بصورت *text.py* ذخیره می‌کنی.

```
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
```

```
canvas.create_text(150, 100, text='There once was a man from Toulouse,')
```

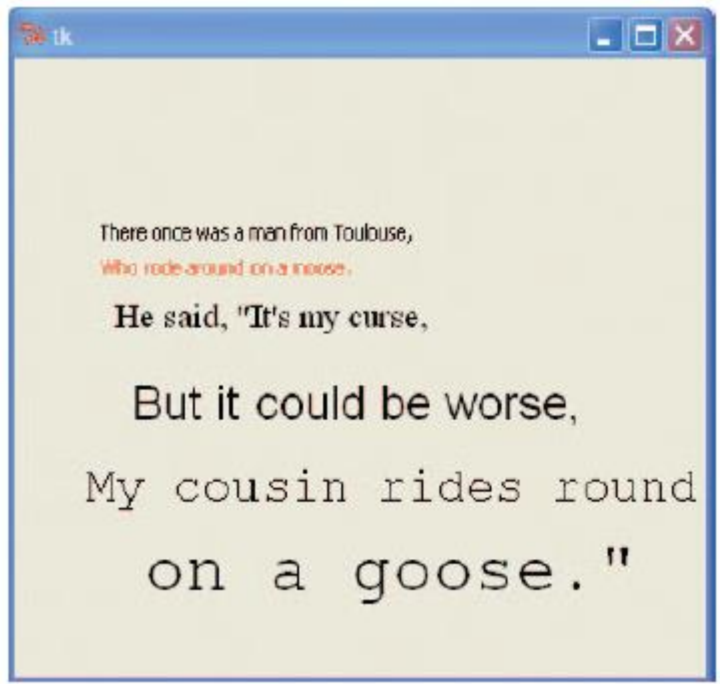
تابع create_text پارامترهای سودمند دیگری را نیز درنظر می‌گیرد از جمله یک متن که با رنگ پر شده است. در کد زیر، تابع create_text با مختصات(130,120)، متنی که می‌خواهیم نمایش داده شود و سطل رنگ قرمز را فراخوانی می‌کنیم.

```
canvas.create_text(130, 120, text='Who rode around on a moose.',
fill='red')
```

همچنین می‌توانیم فونت (ظاهر متن تایپ شده‌ای که می‌خواهیم نمایش داده شود) را بصورت یک چندتایی با نام فونت و سایز متن تعریف کنیم. بعنوان مثال، چندتایی برای فونتTimes با سایز ۲۰ عبارت است از ('Times', 20). در کد زیر، متن را با استفاده ازفونتTimes و سایز۱۵، فونت Helvetica با سایز ۲۰ و فونتCourier با سایز ۲۲و سپس ۳۰ نمایش می‌دهیم:

```
canvas.create_text(150, 150, text='He said, "It₩'s my curse,',
font=('Times', 15))
canvas.create_text(200, 200, text='But it could be worse,',
font=('Helvetica', 20))
canvas.create_text(220, 250, text='My cousin rides round',
font=('Courier', 22))
canvas.create_text(220, 300, text='on a goose."', font=('Courier', 30))
```

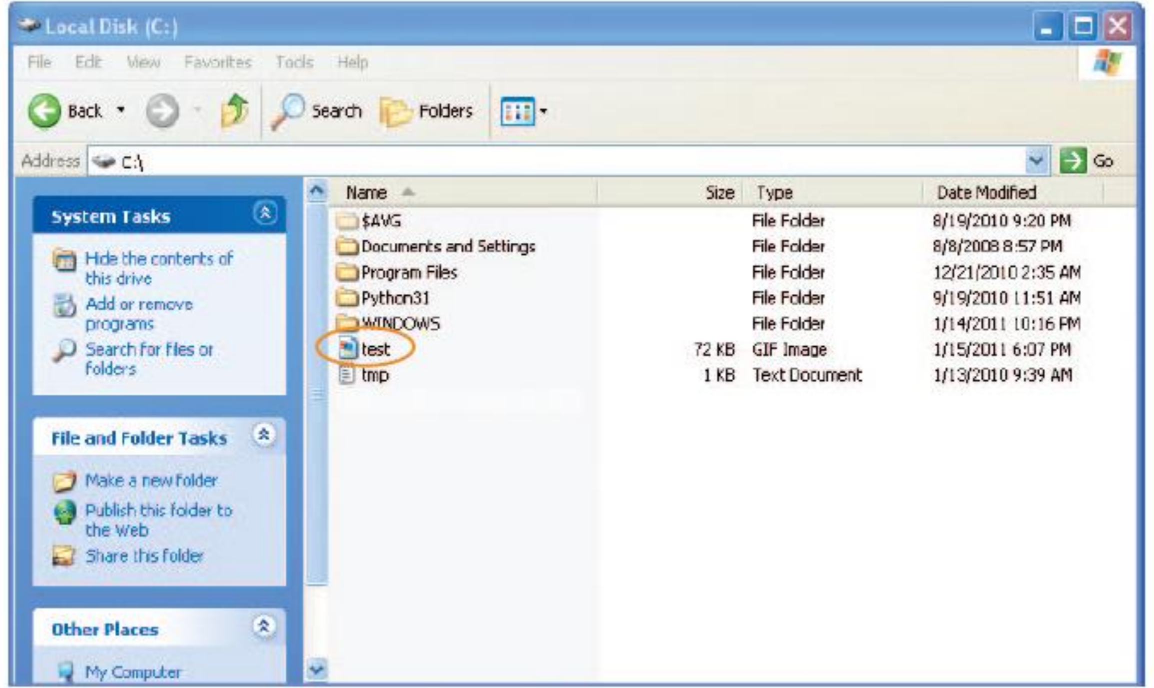
در اینجا نتیجه‌ی این توابع با استفاده از سه فونت تعریف شده با سایزهای مختلف را مشاهده خواهیم نمود:



نمایش تصاویر

برای نمایش یک تصویر روی یک بوم با استفاده از tkinter، ابتدا تصـویر را بارگـذاری کـرده و سپس از تابع create_image روی شیء canvas استفاده می‌کنیم.

هر تصویری که بارگذاری می‌کنید بایستی در دایرکتوری که پایتون بـه آن دسترسـی دارد، قـرار داشته باشد. برای این مثال تصویر خود *test.gif* را در دایرکتوری *C:* قرار می‌دهیم که دایرکتـوری ریشـه (دایرکتوری اصلی) درایو C: می‌باشد ولی شما می‌توانید آن را هرکجا که مایل هستید قرار دهید.



اگر از سیستم Mac یا لینوکس استفاده می‌کنید می‌توانید تصویر را در دایرکتوری Home قرار دهید. اگر نمی‌توانید فایل‌ها را در درایو :C قرار دهید، می‌توانید تصویر را روی دسک‌تاپ بگذارید.

نکته:

با کمک tkinter می‌توانید فقط می‌توانید تصاویر GIF را بارگذاری کنید یعنی فایل‌های تصویری با پسوند gif.. همچنین می‌توانید دیگر انواع تصویر از قبیل *PNG* (.png) و *JPG* (.jpg) را نیز نمایش دهید ولی بایستی از ماژول دیگری مثلاً *Python Imaging Library* استفاده کنید (http://www.pythonware.com/products/pil/).

می‌توانیم تصویر *test.gif* را بصورت زیر نمایش دهیم:

```
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
my_image = PhotoImage(file='c:\\test.gif')
canvas.create_image(0, 0, anchor=NW, image=myimage)
```

در چهار خط اول، همانند مثال‌های قبل، بوم را شکل می‌دهیم. در خط پنجم، تصویر در متغیر my_image بارگذاری شده است. ما PhotoImage با دایرکتوری 'c:\\test.gif' را می‌سازیم. اگر تصویر خود را روی دسک‌تاپ ذخیره کنید، بایستی PhotoImage را با آن دایرکتوری ایجاد نمایید:

```
my_image = PhotoImage(file='C:\\Users\\Joe Smith\\Desktop\\test.gif')
```

بعد از بارگذاری تصـویر در متغیـر،canvas.create_image(0, 0, anchor=NW, image=myimage)
آن را با استفاده از create_image تابع نمایش می‌دهد. مختصات(0,0) جایی است که تصویر نمـایش داده شده است وanchor=NW به تابع فرمان می‌دهد که از لبه‌ی بالا-چپ(NW بـرای شـمال غربـی) تصـویر بعنوان نقطه‌ی شروع استفاده کند (درغیراینصورت، بطور پیش‌فرض از مرکز تصویر بعنوان نقطه‌ی شروع استفاده می‌کند). آخرین پارامتر بانامimage به آن متغیر برای تصویر بارگذاری شده اشاره دارد. در اینجـا نتیجه نشان داده شده است:

ایجاد پویانمایی اصلی

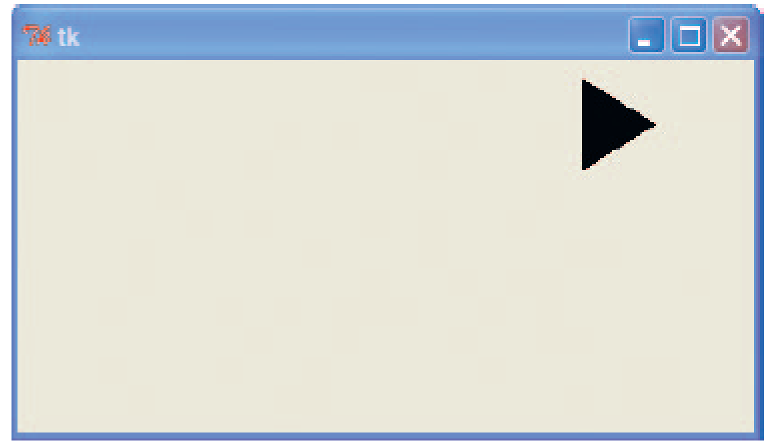
دریافتیم که چگونه می‌توانیم ترسیمات استاتیک بسازیم- تصاویری که حرکت نمـی‌کننـد. ولـی درباره‌ی ایجاد پویانمایی اوضاع چگونه است؟

پویانمایی لزوماً مختص ماژول tkinter نیست بلکه می‌تواند اصول را نیز مدیریت نماید. بعنـوان مثال، می‌توانیم یک مثلث توپر را ساخته و سپس با استفاده از این کـد آن را در عـرض صـفحه حرکـت دهـیم(فرامـوش نکنیـد،**File ▸ New Window** را انتخـاب کـرده، کارتـان را ذخیـره کـرده و سـپس با**Run ▸ Run Module** کد را اجرا نمایید):

```
import time
```

```
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=200)
canvas.pack()
canvas.create_polygon(10, 10, 10, 60, 50, 35)
for x in range(0, 60):
canvas.move(1, 5, 0)
tk.update()
time.sleep(0.05)
```

زمانی که این کد را اجرا می‌کنید، مثلث تا رسیدن به پایان این مسیر، حرکت در عرض صفحه را آغاز می‌کند:



چگونه کار می‌کند؟ همانند قبل، از سه خط اول بعد از وارد کردن(ایمپورت) tkinter برای انجام تنظیمات اصلی برای نمایش یک بوم استفاده کرده‌ایم. در خط چهارم، مثلث را با این تابع می‌سازیم:

```
canvas.create_polygon(10, 10, 10, 60, 50, 35)
```

نکته:

زمانی که این کد را وارد می‌کنید، یک عدد روی صفحه چاپ خواهد شد. این یک شناسه برای چندوجهی است. همانطوریکه در مثال زیر شرح داده شده است، بعداً می‌توانیم از آن برای اشاره به شکل استفاده کنیم.

سپس، یک حلقه‌ی for ساده را برای شماره از 0 تا 59 می‌سازیم که با `for x in range(0, 60):` آغاز می‌شود. بلوک کد درون حلقه، مثلث را در عرض صفحه حرکت می‌دهد. تابع `canvas.move` هر شیء ترسیم شده را با افزودن مقادیری به مختصات x و y آن حرکت می‌دهد. بعنوان مثال، با `canvas.move(1, 5, 0)` شیء دارای ID 1 (شناسه برای

مثلث)را 5 پیکسل در عرض و 0 پیکسل در طول حرکت می‌دهد. برای این که مجدداً آن را برگردانیم، می‌توانیم از فراخوان تابع canvas.move(1, -5, 0) استفاده کنیم.

تابع tk.update()، tkinter را مجبور می‌کند تا صفحه را آپدیت نماید(مجدداً آن را بکشد). اگر از update استفاده نکنیم، tkinter آنقدر منتظر خواهد ماند تا قبل از حرکت دادن مثلث، حلقه خاتمه یابد به این معنی که شاهد خواهید بود که به جای حرکت در عرض بوم، به جایگاه آخر می‌پرد. آخرین خط حلقه یعنی time.sleep(0.05) به پایتون می‌گوید که قبل از ادامه‌ی کار، به اندازه‌ی ۱/۲۰ ثانیه (0.05s) صبر کند.

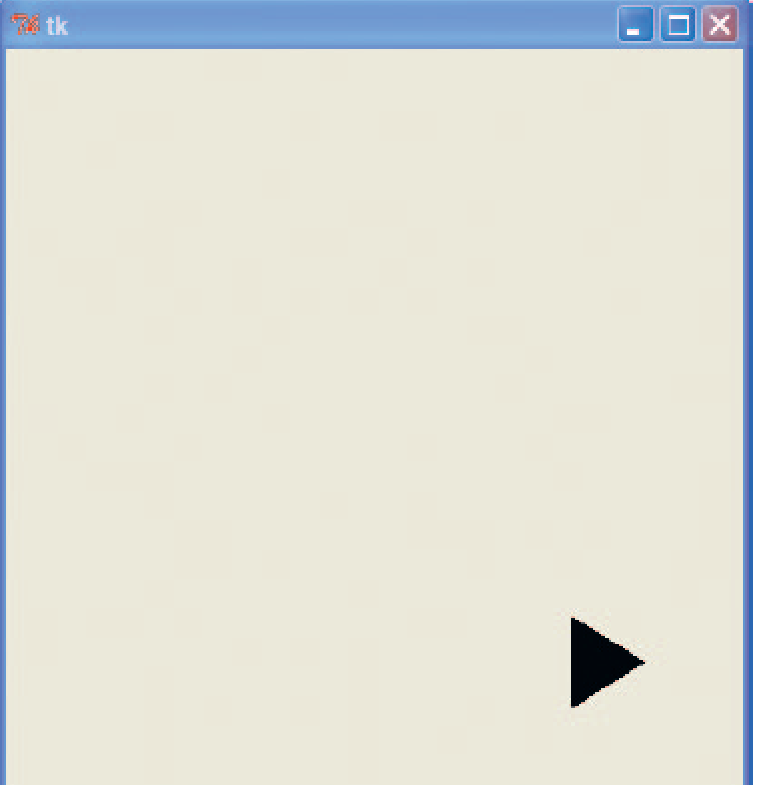
برای اینکه مثلث بصورت قطری به پایین صفحه حرکت کند، می‌توانیم با فراخوانی move(1, 5, 5) این کد را اصلاح کنیم. برای این کار، بوم را بسته و فایل جدیدی(**File ‣ New Window**) را برای کد زیر بسازید:

```
import time
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
canvas.create_polygon(10, 10, 10, 60, 50, 35)
for x in range(0, 60):
        canvas.move(1, 5, 5)
        tk.update()
        time.sleep(0.05)
```

این کد از دو جهت با کد اصلی تفاوت دارد:

- با دستور canvas = Canvas(tk, width=400, height=400) طول بوم را ٤٠٠ پیکسل قرار داده‌ایم نه ۲۰۰ پیکسل.
- با دستور canvas.move(1, 5, 5)، 5 واحد به مختصات x وy مثلث اضافه کرده‌ایم.

بعد از ذخیره و اجرای کد، موقعیت مثلث در انتهای حلقه اینگونه خواهد بود:



برای حرکت دادن قطری مثلث، با استفاده از 5- ,5- صفحه را به موقعیت شروع برمی‌گردانیم (این کد را به انتهای فایل اضافه کنید)

```
for x in range(0, 60):
        canvas.move(1, -5, -5)
        tk.update()
        time.sleep(0.05)
```

ساخت یک شیء به چیزی واکنش نشان دهد

می‌توانیم کاری کنیم که وقتی با استفاده از *event bindings* کلیدی فشار داده می‌شود، مثلث عکس‌العمل نشان دهد. Events(رویدادها) چیزهایی هستند که در حین اجرای برنامه رخ می‌دهد مثلاً کسی موشواره را تکان داده، کلیدی را فشرده یا پنجره‌ای را ببندد. می‌توانیم بهtkinter بگوییم مراقبت این رویدادها بوده و درپاسخ به آنها، کاری را انجام دهد.

برای آغاز مدیریت رویدادها (یعنی وقتی رویدادی رخ می‌دهد، پایتون کاری انجام دهد)، ابتدا یک تابع ایجاد می‌کنیم. بخش انقیاد[1] زمانی مطرح می‌شود که به tkinter می‌گوییم تابع خاصی به رویداد

[1] binding

خاصی مقید (یا ربط داده شده است) شده است؛ به بیان دیگر بطورخودکار برای مدیریت این رویداد، توسط tkinter فراخوانی خواهد شد.

بعنوان مثال، برای اینکه هنگام فشرده شدن کلید ENTER، مثلث حرکت کند، می‌توانیم تابع زیر را تعریف کنیم:

```
def movetriangle(event):
canvas.move(1, 5, 0)
```

این تابع یک پارامتر(event) می‌گیرد که tkinter از آن برای ارسال اطلاعات درباره‌ی رویداد، برای تابع استفاده می‌کند. حال با استفاده از تابع bind_all روی بوم، به tkinter می‌گوییم که این تابع بایستی برای یک رویداد خاص بکارگرفته شود. کد کامل اینگونه خواهد بود:

```
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
canvas.create_polygon(10, 10, 10, 60, 50, 35)
def movetriangle(event):
    canvas.move(1, 5, 0)
canvas.bind_all('<KeyPress-Return>', movetriangle)
```

اولین پارامتر در این تابع، رویدادی را توصیف می‌کند که میخواهیم tkinter مراقب آن باشد. در این مورد، <KeyPress-Return> را فراخوانی می‌کند که فشار دادن کلید ENTER یا RETURN است. به tkinter می‌گوییم که هرگاه رویداد KeyPress رخ می‌دهد، تابع movetriangle بایستی فراخوانده شود. این کد را اجرا کرده، با موشواره روی بوم کلیک کرده و کلید ENTER را روی صفحه‌کلید فشار دهید.

اگر بخواهیم جهت حرکت مثلث را با توجه به فشرده شدن کلیدهای مختلف مثلاً کلیدهای جهت‌نما[1] تغییر دهیم، چه کار باید بکنیم؟ مشکلی وجود ندارد. فقط بایستی تابع movetriangle را بصورت زیر تغییر دهیم:

```
def movetriangle(event):
```

[1] Arrow key

```
    if event.keysym == 'Up':
        canvas.move(1, 0, -3)
    elif event.keysym == 'Down':
        canvas.move(1, 0, 3)
    elif event.keysym == 'Left':
        canvas.move(1, -3, 0)
    else:
        canvas.move(1, 3, 0)
```

شیء رویداد که به movetriangle انتقال داده شده است حاوی چندین متغیر می‌باشد. یکی از این متغیرها keysym (مخفف نماد کلید) نامیده می‌شود که یک رشته است و مقدار کلید واقعی فشارداده شده را نگه میدارد. خط: if event.keysym == 'Up' بیان می‌نماید که اگر متغیر keysym حاوی رشته‌ی 'Up' باشد، آنگاه بایستی canvas.move را با پارامترهای (1, 0, -3) صدا بزنیم دقیقاً همان کاری که در خط زیر انجام می‌دهیم. اگر keysym حاوی 'Down' باشد مثلاً: elif event.keysym == 'Down' آن را با پارامترهای (1, 0, 3) صدا خواهیم زد و به همین ترتیب ادامه پیدا می‌کند.

یادآور می‌شویم که پارامتر اول، شماره شناسایی برای شکلی است که روی بوم ترسیم شده است، پارامتر دوم مقداری است که بایستی به مختصات x (افقی) اضافه شود و متغیر سوم مقداری است که به مختصات y (عمودی) اضافه می‌گردد.

سپس به tkinter می‌گوییم که باید از تابع movetriangle برای مدیریت رویدادها از چهار کلید مختلف (بالا،پایین، چپ و راست) استفاده شود. در ادامه نشان می‌دهیم که این کد چگونه خواهد بود. زمانی که این کد را وارد می‌کنیم، اگر با انتخاب **File ▸ New Window** یک پنجره شِل جدید بسازید، کار بازهم آسان‌تر خواهد شد. قبل از اجرای کد، آن را با نام فایل معناداری ذخیره کنید مثلاً *movingtriangle.py*.

```
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
canvas.create_polygon(10, 10, 10, 60, 50, 35)
def movetriangle(event):
❶ if event.keysym == 'Up':
❷ canvas.move(1, 0, -3)
❸ elif event.keysym == 'Down':
❹ canvas.move(1, 0, 3)
❺ elif event.keysym == 'Left':
```

```
❻ canvas.move(1, -3, 0)
❼ { else:
❽ canvas.move(1, 3, 0)
canvas.bind_all('<KeyPress-Up>', movetriangle)
canvas.bind_all('<KeyPress-Down>', movetriangle)
canvas.bind_all('<KeyPress-Left>', movetriangle)
canvas.bind_all('<KeyPress-Right>', movetriangle)
```

در اولین خط تابع movetriangle، در ❶ وجود 'up' در متغیـر keysym را بررسـی مـی‌کـنـیم. اگـر چنین بود، در ❷ با استفاده از تابع move با پارامترهای 3- ,0 ,1 به سمت بالا حرکت می‌دهیم. اولین پارامتر، شناسه‌ی مثلث است، پارامتر دوم، مقداری است که به راست حرکت میکنـد (چـون نمـی‌خـواهیم افقـی حرکت کند پس این مقدار را برابر با 0 قرار می‌دهیم) و پارامتر سوم، مقداری است که بـه پـایین حرکـت می‌نماید(3- پیکسل).

سپس ❸ در وجود 'Down' در keysym را بررسی مـی‌کـنـیم، در اینصـورت، در ❹ مثلـث را بـه سمت پایین(3 پیکسل) حرکت می‌دهیم. در ❺ وجود مقدار 'left' بررسی می‌شود، اگر وجود داشت، مثلث را به سمت چپ(3- پیکسل) حرکت می‌دهـیم. درصـورتیکه هیچیـک از مقـادیر تطـابق نداشـته باشـند، آخرین else در ❼، مثلث را به راست حرکت می‌دهد(در ❽).

حالا مثلث بایستی در جهت فشرده شدن کلیدهای جهت‌نما حرکت کند.

راه‌های دیگر برای استفاده از شناسه

هرگاه از تابع function _create بوم استفاده می‌کنید، مـثلاً create_polygon یـا create_rectangle ، یک شناسه برگردانده می‌شود. همانطوریکه قبلاً با تابع move مشاهده کردیم می‌توان از این عدد شناسایی در کنار دیگر توابع canvas استفاده نمود.

```
>>> from tkinter import *
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> canvas.create_polygon(10, 10, 10, 60, 50, 35)
1
>>> canvas.move(1, 5, 0)
```

مسئله‌ای که در این مثال وجود دارد این است کـه create_polygon همیشـه 1 را برنمـی‌گردانـد. بعنوان مثال، اگر شکل‌های دیگری ساخته باشید، ممکن است 2,3 یا حتی 100 را برای آن برگردانده باشد (با توجه به تعداد اشکالی که ساخته‌اید). اگر برای ذخیره‌ی مقداری بازگشتی بصورت یک متغیر، کـد را تغییر داده و سپس از متغیر استفاده کنیم (به جای اینکه فقط به عدد 1 اشاره کنیم)، آنگاه کد بدون توجـه به عدد بازگشتی کار خود را انجام خواهد داد:

```
>>> mytriangle = canvas.create_polygon(10, 10, 10, 60, 50, 35)
>>> canvas.move(mytriangle, 5, 0)
```

تابع move به ما اجازه می‌دهد تا با استفاده از شناسه‌ی شیءها، آنها را در صـفحه حرکـت دهـیم. ولی توابع canvas دیگری نیز وجود دارند که می‌توانند آنچه را ترسیم کرده‌ایم تغییر دهند. بعنـوان مثـال، می‌توان از تابع itemconfig بوم برای تغییر برخی از پارامترهای شکل استفاده نمود مثلاً رنگ دور شکل و رنگی که با آن پُر شده است.

می‌خواهیم یک مثلث قرمز رنگ بکشیم:

```
>>> from tkinter import *
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> mytriangle = canvas.create_polygon(10, 10, 10, 60, 50, 35,
fill='red')
```

با استفاده از itemconfig و استفاده از شناسه بعنوان پارامتر اول، می‌توانیم رنـگ مثلـث را تغییـر دهیم. کد زیر می‌گوید: «با توجه به عدد متغیر mytriangle، رنگ شیء شناسایی شده را آبی کن»

```
>>> canvas.itemconfig(mytriangle, fill='blue')
```

همچنین می‌توانیم رنگ دور مثلث را تغییر دهیم و ایـن بـار نیـز از شناسـه بعنـوان پـارامتر اول استفاده می‌کنیم.

```
>>> canvas.itemconfig(mytriangle, outline='red')
```

در ادامه خواهیم آموخت که چگونه می‌تـوانیم تغییـرات دیگـری را در ترسیمات خود اعمال کنیم از جمله پنهـان کـردن و ظـاهر کـردن مجـدد آن. در فصل بعد زمانی که نوشتن بازی را شروع می‌کنیم خواهید دید که امکـان تغییـر یک رسم زمانی که روی صفحه نمایش داده می‌شود، چقدر سودمند خواهد بود.

آنچه آموختید

در این فصل از ماژول tkinter برای ترسیم اشکال ساده‌ی هندسی روی بوم، نمایش تصاویر و اجرای پویانمایی اصلی استفاده کردیم. آموختید که چگونه از انقیادهای رویداد استفاده کرده تا کاری کنیم که اشکال ترسیم شده نسبت به فشرده شدن یک کلید توسط یک نفر واکنش نشان دهند، کاری که وقتی می‌خواهیم نوشتن بازی را آغاز کنیم بسیار سودمند خواهد بود. آموختید که چگونه توابعی را در tkinter بسازیم تا عدد شناسه‌ای را برگردانند که بعد از ترسیم شدن شکل‌ها، به درد اصلاح کردن آنها می‌خورند مثلاً حرکت دادن آنها در صفحه یا تغییر رنگ آن‌ها.

چیستان‌های برنامه‌نویسی

برای بازی با ماژول tkinter و پویانمایی پایه، مثال‌های زیر را امتحان کنید. برای مشاهده‌ی جواب‌ها به *http://python-for-kids.com/* مراجعه کنید.

۱) پر کردن صفحه با مثلث‌ها

با استفاده از tkinter برنامه‌ای بنویسید که صفحه را پر از مثلث کند. سپس کد را طوری تغییر دهید تا صفحه را با مثلث‌هایی با رنگ‌های مختلفی پر کند.

۲) حرکت دادن مثلث

دستکاری کد برای حرکت دادن مثلث (ایجاد پویانمایی پایه در صفحه‌ی۱۸۳) بطوری که در عرض صفحه به سمت راست، سپس پایین حرکت کرده، بعداً به چپ برگشته و به موقعیت شروع برگردد.

۳) عکس متحرک

سعی کنید با استفاده از tkinter تصویری از خودتان را روی بوم نمایش دهید. مطمئن شوید که یک تصویر GIF باشد! آیا می‌توانید آن را در صفحه جابجا کنید؟

بخش ۲

پرش

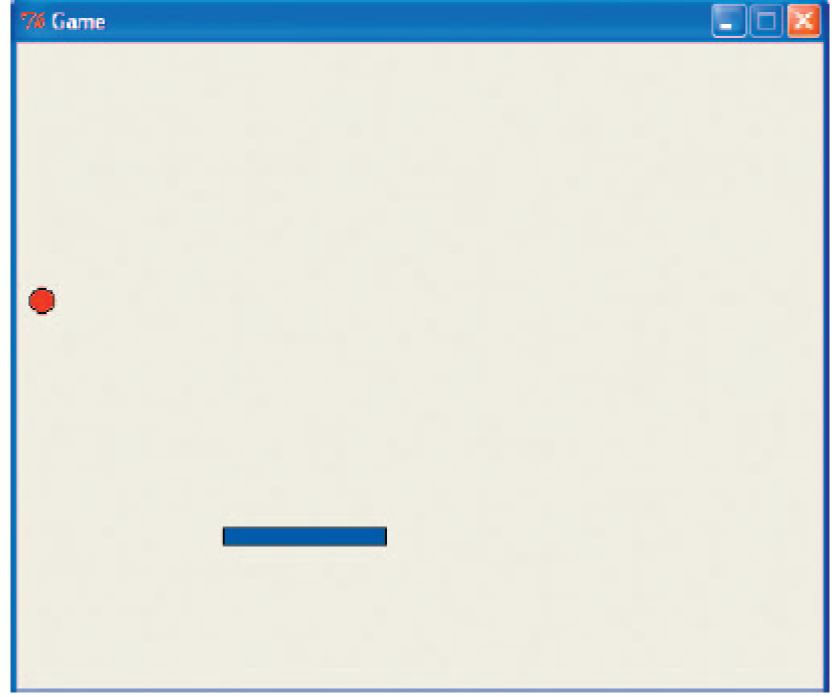
فصل۱۳

اولین بازی خود را آغاز کنید: پرش!

تا اینجا، مبانی برنامه‌نویسی رایانه‌ای را دریافتید. آموختید که چگونه از متغیرها بـرای ذخیـره‌ی اطلاعات، دستوراتif برای کد شرطی و حلقه‌هایfor برای تکرار کد استفاده کنید. می‌دانید کـه چگونـه می‌توان تابعی را برای استفاده مجدد از کد، ایجاد نمود و چگونه از کلاس‌ها و شیءها برای تقسیم کد به تکه‌های کوچکتر برای آسان‌تر شدن درک آن‌ها استفاده نمود. آموختید که چگونه با هر دو ماژولturtle و thinker می‌توانید گرافیک‌هایی را روی صفحه بکشید. حالا زمان آن رسیده تا از این دانـش بـرای خلـق اولین بازی خود استفاده کنید.

ضربه زدن به توپ درحال جهش

قصد داریم با یک توپ درحال پرش و یک راکت یک بازی بسازیم. توپ در صفحه پرواز کرده و بازیکن با راکت به آن ضربه می‌زند. اگر توپ به انتهای صفحه اصابت کند، بازی پایان می‌یابد. در این-جا پیش‌نمایشی از بازی نهایی نشان داده شده است.



بازی ما به نظر ساده می‌باشد ولی کد بسیار هوشمندانه‌تر از چیزی است که تا اینجا نوشته‌ایم زیرا چیزهای دیگری نیز وجود دارد که بایستی آنها را مدیریت نماییم. بعنوان مثال، بایستی راکت و توپ را پویانمایی نموده و دریافت که چه زمان توپ به راکت یا دیوار برخورد می‌کند.

در این فصل، با افزودن یک بوم بازی و یک توپ جهنده، خلق بازی را آغاز می‌کنیم. رد فصل بعد، با افزودن راکت بازی را تکمیل خواهیم نمود.

خلق بوم بازی

برای خلق بازی‌تان ابتدا بایستی فایل جدیدی را در شِل پایتون باز کنید (انتخاب **File ▸ New Window**). سپس tkinter را وارد (ایمپورت) کرده و بومی را برای ترسیم روی آن ایجاد می‌کنیم:

```
from tkinter import *
import random
import time
tk = Tk()
tk.title("Game")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()
```

این مثال اندکی با مثال‌های قبل متفاوت است. ابتدا، ماژول‌های **time** و **radom** با import random و import time، را برای استفاده‌ی آتی در کد، وارد(ایمپورت) می‌کنیم.

با ("tk.title("Game، از تابع **title** شیء **tk** که با ()tk = Tk ساخته‌ایم استفاده کرده تا عنوانی را برای پنجره تعیین کنیم. سپس از resizable استفاده کرده تا سایز ثابتی را برای پنجره تعیین نماییم. پارامترهای 0,0 به می‌گویند که «سایز پنجره نه بصورت افقی و نه عمودی تغییر نمی‌کند». سپس wm_attributes را صدا زده و به **tkinter** بگوییم که پنجره‌ی حاوی بوم ما را جلوتر از بقیه پنجره‌ها قرار دهد("topmost-").

توجه داشته باشید وقتی که با = canvas یک شیء **canvas** می‌سازیم، نسبت به مثال‌های قبل، چند پارامتر مشخص بیشتر را درنظر می‌گیریم. بعنوان مثال، هم bd=0 و هم highlightthickness=0 تضمین می‌نمایند که هیچ مرزی دور حاشیه‌ی خارجی بوم وجود ندارد و درنتیجه ظاهر بهتری در صفحه‌ی بازی ایجاد خواهد شد.

خط ()canvas.pack به بوم می‌گوید که منطبق با پارامترهای طول و عرض در خط قبل، سایز خود را تطبیق دهد. درنهایت، ()tk.update به **thinker** می‌گوید خود را برای پویانمایی در بازی ما مقداردهی اولیه نماید. بدون خط آخر، هیچ چیز طبق انتظار کار نخواهد کرد.

مطمئن شوید که درطی فرآیند کار، کد را ذخیره می‌کنید. اولین باری که آن را ذخیره می‌کنید، نام فایل معناداری برای آن درنظر بگیرید؛ مثلاً *paddleball.py*.

ایجاد یک کلاس ball

حال کلاسی را برای توپ خواهیم ساخت. با کدی آغاز می‌کنیم که بواسطه‌ی آن توپ خودش را روی بوم ترسیم می‌نماید. کاری که باید انجام دهیم به این ترتیب است:

- خلق یک کلاس به نام ball که پارامترهایی را برای بوم و رنگ توپی که می‌خواهیم ترسیم کنیم را می‌گیرد
- ذخیره‌ی بوم بصورت یک متغیر شیء زیرا توپ را روی آن ترسیم خواهیم نمود
- ترسیم یک دایره‌ی تو پُر روی بوم با استفاده از مقدار پارامتر رنگ بعنوان سطل رنگ.

- ذخیره‌ی شناسه‌ای که tkinter در هنگام ترسیم دایره (بیضی) برمی‌گرداند زیرا قصد داریم از آن برای جابجا کردن توپ در صفحه استفاده کنیم.
- حرکت دادن بیضی به وسط بوم.

این کد بایستی دقیقاً قبل از دو خط اول در فایل اضافه گردد (بعد از import time):

```
from tkinter import *
import random
import time
❶ class Ball:
❷ def __init__(self, canvas, color):
❸ self.canvas = canvas
❹ self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
❺ self.canvas.move(self.id, 245, 100)
def draw(self):
pass
```

ابتدا، در ❶ کلاس را Ball نامگذاری می‌کنیم. سپس، در ❷ تابع مقداردهی اولیه (همانطوریکه در فصل ۸ توضیح داده شد) ای را ایجاد می‌کنیم که پارامترهای canvas و color را می‌گیرد. در ❸ متغیر شیء canvas را برابر با مقدار پارامتر canvas قرار می‌دهیم.

در ❹ تابع create_oval را با پنج پارامتر صدا می‌زنیم: مختصات x و y برای گوشه‌ی بالا-چپ (10و10)، مختصات x و y برای گوشه‌ی پایین-راست (25و25) و درنهایت سطل رنگ برای بیضی.

تابع create_oval یک شناسه را برای شکل ترسیم شده برمی‌گرداند که آن را در متغیر شیء id ذخیره می‌کنیم. در ❺ بیضی را به وسط بوم آورده (موقعیت 245,100) و بوم می‌داند که چه چیزی جابجا شده است زیرا از شناسه شکل ذخیره شده (متغیر شیء id) برای شناسایی آن استفاده می‌کنیم.

در دو خط بعد از کلاس Ball، تابع draw را با def draw(self) تعریف کرده و بدنه‌ی تابع همان گذرواژه‌ی Pass می‌باشد. در این لحظه کاری انجام نمی‌دهد. ما چیزهایی را به این تابع اضافه خواهیم کرد.

حال که کلاس Ball را ساخته‌ایم، بایستی شیء‌ای از این کلاس را بسازیم (بخاطر داشته باشید که یک کلاس کاری را که می‌تواند انجام دهد توصیف می‌کند ولی شیء چیزی است که درواقع انجام می‌دهد). کد زیر را به انتهای برنامه اضافه کرده تا یک شیء توپ قرمز بسازیم:

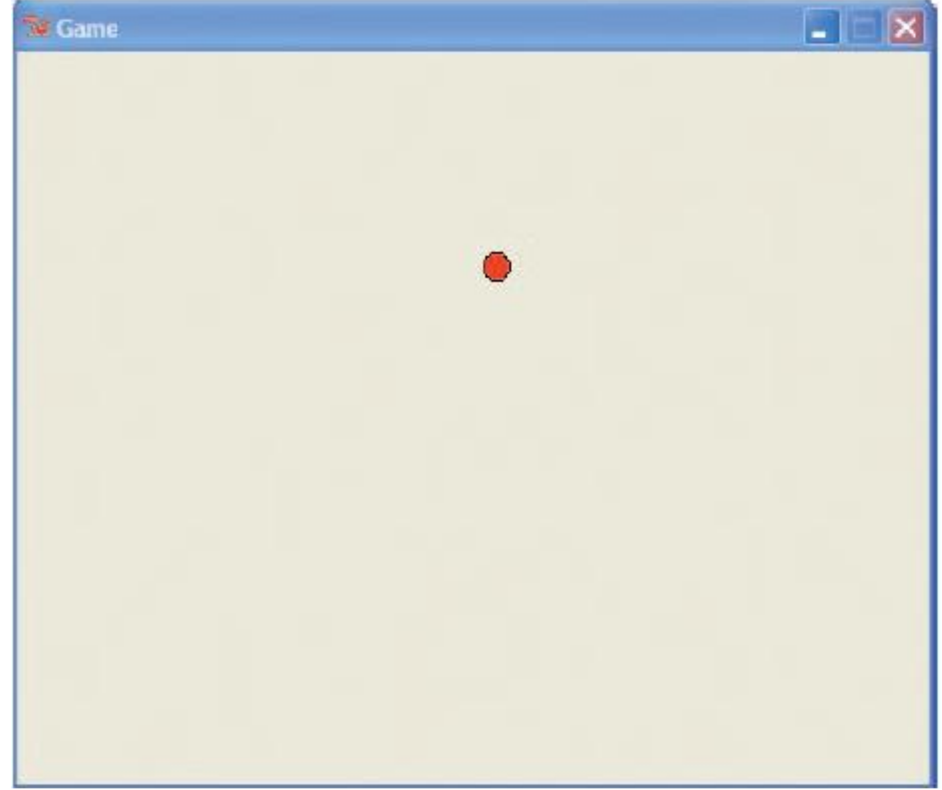
```
ball = Ball(canvas, 'red')
```

اگر این برنامه را بـا اسـتفاده از **Run ‣ Run Module** اجـرا کنیـد، بـوم بـرای کسـری از ثانیـه ظاهرشده و بلافاصله ناپدید می‌شود. برای این که پنجـره بلافاصـله بسـته نشـود، بایسـتی یـک حلقـه‌ی پویانمایی اضافه کنیم که main loop (حلقه‌ی اصلی) بازی ما نامیده می‌شود.

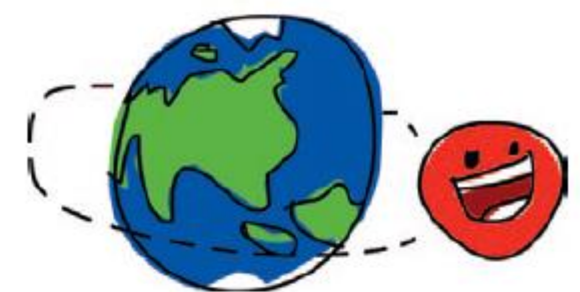
حلقه‌ی اصلی بخش مرکزی یک برنامه است که عموماً اکثـر کارهـای آن را مـدیریت مـی‌کنـد. حلقه‌ی اصلی ما برای لحظاتی به tkinter می‌گوید که صفحه را مجدداً ترسیم نماید. حلقه تـا ابـد اجـرا می‌شود(یا حداقل تا زمانی که پنجره بسته شود) و دائماً به tkinter می‌گوید که صفحه را مجـدداً ترسـیم کند و سپس برای یک‌صدم ثانیه به خواب می‌رود. این کد را به انتهای برنامه اضافه می‌کنیم.

```
ball = Ball(canvas, 'red')
while 1:
tk.update_idletasks()
tk.update()
time.sleep(0.01)
```

حال اگر کد را اجرا کنید، توپ بایستی در مرکز بوم ظاهر شود:



اضافه کردن یک فعالیت



بعد از ایجاد کلاس Ball، زمان آن رسیده تا توپ را بــه حالت پویانمایی درآوریم. کاری می‌کنیم تا توپ حرکت کــرده، پرش کند و جهتش تغییر نماید.

حرکت دادن توپ

برای حرکت دادن توپ، تابع draw را بصورت زیر تغییر می‌دهیم:

```
class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
    def draw(self):
        self.canvas.move(self.id, 0, -1)
```

از آنجاییکه __init__ پارامتر canvas رابصورت متغیر شیءcanvas ذخیره مــی‌کنـد، از ایــن متغیــر باself.canvas استفاده کرده و تابع move را روی بوم صدا می‌زنیم.

سه پارامتر را در move قرار می‌دهیم:id بیضی، و اعداد0 و1- . 0 مــی‌گویـد کـه بصــورت افقــی حرکت نکن و 1- می‌گوید که یک پیکسل به بالای صفحه برو.

چون ایده‌ی خوبی برای امتحان کردن کد درطی فرآیند کدنویسی می‌باشد، ایــن تغییــر انــدک را اعمال کرده‌ایم. تصور کنید اگر کل کد را به یکباره برای بازی می‌نوشتیم و سپس درمی‌یافتیم که جــواب نمی‌دهد، چه اتفاقی می‌افتاد. کجا باید دنبال اشکال برنامه می‌گشتیم؟

تغییر دیگری که اعمال شده است در حلقه‌ی اصلی در پایین برنامه مــی‌باشــد.در بلـوک حلقــه-ی while (حلقه‌ی اصلی ما!) یک فراخوان به تابع شیء توپ draw می‌زنیم:

```
while 1:
    ball.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

اگر این کد را اکنون اجرا کنید، توپ بایستی به طرف بالای بوم رفته و ناپدید شود زیرا کد به tkinter دستور می‌دهد به سرعت صفحه را مجدداً ترسیم کند – فرامین update_idletasks و update به tkinter دستور می‌دهند عجله کرده و چیزی که روی بوم است را بکشد.

فرمان time.sleep یک فراخوان به تابع sleep ماژول time است که به پایتون می‌گوید برای یک-صدم ثانیه کاری انجام ندهد. این کار تضمین می‌نماید که برنامه آنقدر سریع اجرا نمی‌شود که توپ قبل از این که ما آن را ببینیم، ناپدید گردد.

بنابراین حلقه اساساً می‌گوید: توپ را اندکی جابجا کن، مجدداً صفحه را با موقعیت جدید ترسیم کن، برای چند لحظه کاری نکن، مجدداً کار را از سر بگیر.

نکته:

ممکن است زمانیکه پنجره‌ی بازی را می‌بندید پیام‌های خطایی را روی شِل ببینید. به این دلیل است که وقتی پنجره را می‌بندید، کد از حلقه‌ی while خارج می‌شود و پایتون گیج می‌شود.

بازی شما بایستی چیزی شبیه به این باشد:

```
from tkinter import *
import random
import time

class Ball:
        def __init__(self, canvas, color):
                self.canvas = canvas
                self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
                self.canvas.move(self.id, 245, 100)
        def draw(self):
                self.canvas.move(self.id, 0, -1)
tk = Tk()
tk.title("Game")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()
```

```
ball = Ball(canvas, 'red')

while 1:
        ball.draw()
        tk.update_idletasks()
        tk.update()
        time.sleep(0.01)
```

جهش دادن توپ

توپی که در بالای صفحه ناپدید شود به درد بازی نمی‌خورد پس اجازه دهید کمی بپرد. ابتدا، چند متغیر شیء اضافی را در تابع مقداردهی کلاس Ball ذخیره می‌کنیم:

```
def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
        self.x = 0
        self.y = -1
        self.canvas_height = self.canvas.winfo_height()
```

سه خط دیگر به برنامه اضافه کرده‌ایم. با self.x = 0 متغیر شیءx را برابر با0 قرار داده و سپس باself.y = -1 متغیر y را برابر با 1- قرار می‌دهیم. درنهایت نیز با فراخوانی تابع بوم winfo_height متغیر شیءcanvas_height را مقداردهی می‌کنیم. این تابع طول جاری بوم را برمی‌گرداند.

بازهم تابع draw را تغییر می‌دهیم:

```
def draw(self):
   ❶    self.canvas.move(self.id, self.x, self.y)
        ❷pos = self.canvas.coords(self.id)
   ❸    if pos[1] <= 0:
self.y = 1
   ❹    if pos[3] >= self.canvas_height:
        self.y = -1
```

در ❶، با انتقال متغیرهای شیءx وy فراخوان را به تابع move بوم تغییر می‌دهیم. سپس، در ❷ با فراخوانی تابع coords بوم، متغیری به نام pos را می‌سازیم. این تابع مختصات جاری x وy هر چیزی که روی بوم ترسیم شده باشد را تا مادامیکه شماره‌ی شناسایی آن را می‌دانید،برمی‌گرداند. در این مورد، متغیر شیءid را که حاوی شناسه‌ی بیضی است در coords قرار می‌دهیم.

تابع coords مختصات را بصورت یک فهرست چهار عددی برمی‌گرداند. اگر نتایج فراخوان این تابع را چاپ کنیم، چیزی شبیه این را خواهیم دید:

```
print(self.canvas.coords(self.id))
[255.0, 29.0, 270.0, 44.0]
```

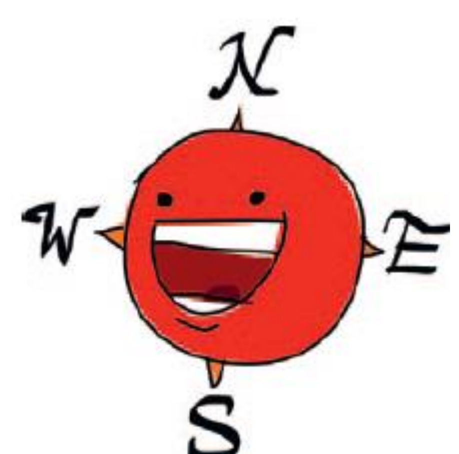
دو عدد اول در این فهرست(255.0 و 29.0) حاوس مختصات بالا-چپ بیضی(x1 و y1)؛ دو عدد بعدی(270.0 و 44.0) مختصات x2 و y2 پایین-راست می‌باشند. از این مقادیر در چند خط بعدی کد استفاده خواهیم کرد.

در ❸ می‌بینیم که مختصات y1 (نوک توپ!)کوچکتر یا مساوی 0 می‌باشد. دراینصورت، متغیر شیءy را برابر با 1 قرار می‌دهیم. درعمل، اگر به بالای صفحه ضربه بزنیم، دیگر از موقعیت عمودی یک واحد کسر نشده و درنتیجه حرکت به سمت بالا متوقف خواهد شد.

در ❹ اگر مختصات y2 (زیر توپ!) بزرگتر یا مساوی متغیر canvas_height است. دراینصورت متغیر شیءy را مجدداً برابر با 1- قرار خواهیم داد.

اکنون این کد را اجرا کرده و توپ بایستی تا زمانی که پنجره را نبسته‌ایم، بالا و پایین بپرد.

تغییر جهت شروع حرکت توپ



پرش دادن توپ به آرامی، بازی جالبی نیست پس اجازه دهید با تغییر جهت شروع حرکت توپ، اندکی هیجان بازی را اضافه کنیم- زاویه‌ای که با شروع بازی توپ به بالا می‌رود. در تابع __init__ خطوط زیر را

```
self.x = 0
self.y = -1
```

به صورت زیر تغییر می‌دهیم (مطمئن شوید که در ابتدای هر خط تعداد جاهای خالی درست هستند- هشت جای خالی):

```
❶ starts = [-3, -2, -1, 1, 2, 3]
❷ random.shuffle(starts)
❸ self.x = starts[0]
❹ self.y = -3
```

در ❶ متغیر start را با یک لیست شش عددی ایجاد کرده و در ❷ با فراخوانی random.shuffle، این فهرست را درهم می‌ریزیم. در ❸ مقدار x را برابر با اولین آیتم در لیست قرار داده بطوریکه x بتواند هر عددی در لیست از 3- تا 3 باشد.

سپس اگر در ❹، y را به 3- تغییر دهیم (برای افزایش سرعت توپ) برای این که مطمئن شویم توپ از صفحه محو نمی‌شود، بایستی بیشتر به آن اضافه کنیم. برای حفظ عرض بوم در متغیر شیء جدید canvas_width ، خط زیر را به انتهای تابع __init__ اضافه می‌کنیم:

```
self.canvas_width = self.canvas.winfo_width()
```

از این متغیر شیء جدید در تابع draw استفاده کرده تا ببینیم که توپ به بالای یا پایین بوم برخورد کرده است یا نه:

```
if pos[0] <= 0:
        self.x = 3
if pos[2] >= self.canvas_width:
        self.x = -3
```

از آنجایی که x را برابر با 3 و 3- درنظر گرفته‌ایم، همین کار را با y انجام می‌دهیم بطوریکه توپ با سرعت ثابت در تمامی جهات حرکت کند. تابع draw بایستی شبیه به این باشد:

```
def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
                self.y = 3
        if pos[3] >= self.canvas_height:
                self.y = -3
        if pos[0] <= 0:
                self.x =3
        if pos[2] >= self.canvas_width:
                self.x = -3
```

کد را ذخیره و سپس اجرا کنید، درنتیجه توپ بایستی بدون این که ناپدید شود در صفحه جهش کند. در اینجا ظاهر یک برنامه‌ی کامل را مشاهده می‌نمایید:

```
from tkinter import *
import random
import time
class Ball:
	def __init__(self, canvas, color):
		self.canvas = canvas
		self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
		self.canvas.move(self.id, 245, 100)
		starts = [-3, -2, -1, 1, 2, 3]
		random.shuffle(starts)
		self.x = starts[0]
		self.y = -3
		self.canvas_height = self.canvas.winfo_height()
		self.canvas_width = self.canvas.winfo_width()
	def draw(self):
		self.canvas.move(self.id, self.x, self.y)
		pos = self.canvas.coords(self.id)
		if pos[1] <= 0:
			self.y = 3
		if pos[3] >= self.canvas_height:
			self.y = -3
		if pos[0] <= 0:
			self.x = 3
		if pos[2] >= self.canvas_width:
			self.x = -3
tk = Tk()
tk.title("Game")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()
ball = Ball(canvas, 'red')
while 1:
	ball.draw()
	tk.update_idletasks()
	tk.update()
	time.sleep(0.01)
```

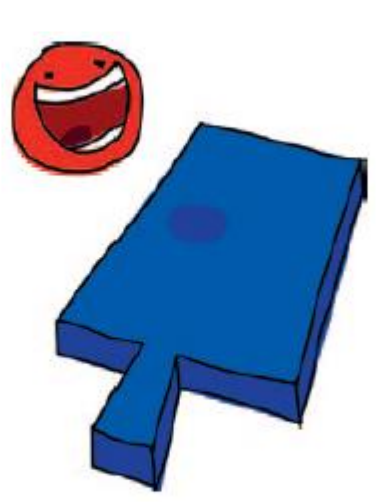
آنچه آموختید

در این فصل، با استفاده از ماژولtkinter نوشتن اولین بازی را آغاز کردیم. کلاسی را برای یک توپ ساخته و آن را به جرکت درآوردیم بطوریکه در صفحه جابجا شود. از مختصات استفاده کرده تا ببینیم چه زمان توپ به کناره‌های بوم برخورد می‌کند تا بتوانیم آن را به جهش درآوریم. همچنین از تابعShufle در ماژول randim استفاده کردیم تا توپ ما همیشه از یک جهت حرکت خود را آغاز نکند. در فصل بعد، با اضاه کردن راکت، بازی را تکمیل خواهیم نمود.

فصل ١٤

تکمیل اولین بازی: بالا و پایین پریدن!

در فصل قبل، ساخت اولین بازی را آغاز کردیم: بالا و پایین پریدن! یک بوم ایجاد کـرده و یـک توپ جهنده را به کد بازی اضافه کردیم. ولی توپ ما تا ابد در صفحه بالا و پایین می‌پرد (یا حـداقل تـا زمانی که رایانه را خاموش نکردایم) که بازی جالبی نخواهد بود. حال یک راکـت را در اختیـار بـازیکن قرار می‌دهیم. همچنین المانی از شانس را به بازی اضافه می‌کنیم که باعث مـی‌شـود بـازی جـداب‌تـر و چالش‌برانگیزتر گردد.



اضافه کردن راکت

هیچ چیز جذابی در بالا و پایین پریدن یک توپ وجود ندراد مگـر اینکه چیز دیگری در این میان وجود داشته باشد که به آن ضربه بزنند. زمان آن رسیده تا یک راکت بسازیم!

کار را با اضافه کردن کد زیر بعد از کلاس Ball آغاز کرده تـا یـک راکت بسازیم (این کار در یک خط جدید زیر تابع Ball draw انجام خواهد شد):

```
def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
                self.y = 3
        if pos[3] >= self.canvas_height:
```

```
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.canvas_width:
            self.x = -3
class Paddle:
        def __init__(self, canvas, color):
            self.canvas = canvas
            self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
            self.canvas.move(self.id, 200, 300)
def draw(self):
            pass
```

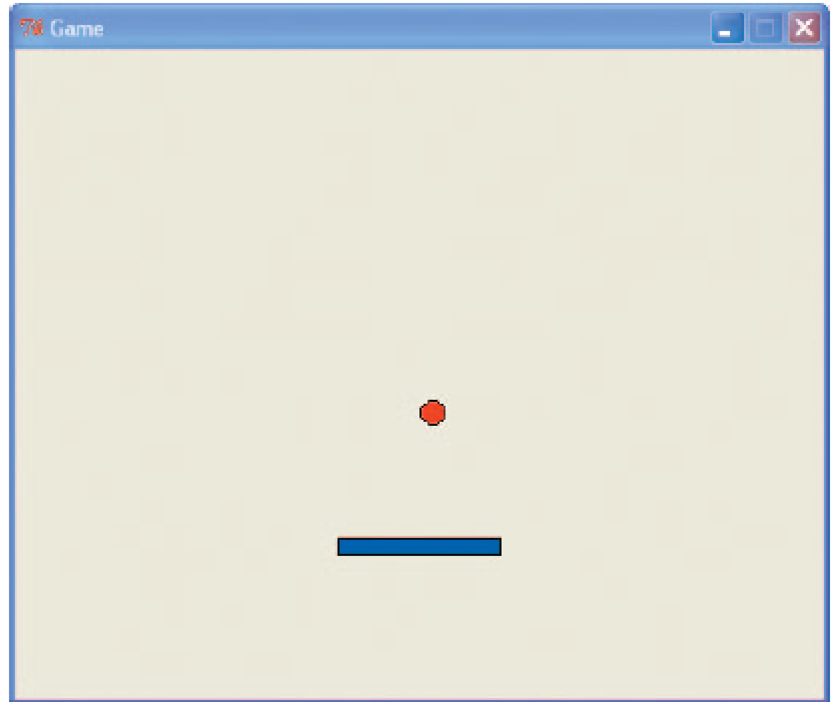
کدی که به تازگی اضافه شد تقریباً همان کلاس Ball است با این تفاوت کـه create_rectangle را (به جای create_oval) صدا زده و مستطیل را به موقعیـت 200,300 حرکـت مـی‌دهـیم (۲۰۰ پیکسـل در عرض و ۳۰۰پیکسل به پایین).

سپس در پایین فهرست‌بندی کد، شیءای از کلاس paddle ایجاد کرده و حلقه‌ی اصلی را بـرای فراخوانی تابع draw راکت تغییر می‌دهیم:

```
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, 'red')

while 1:
        ball.draw()
        paddle.draw()
        tk.update_idletasks()
        tk.update()
        time.sleep(0.01)
```

اگر الآن بازی را اجرا کنید، بایستی شاهد یک توپ که به بـالا و پـایین مـی‌پـرد و یـک راکـت بیحرکت مستطیلی شکل باشید:



حرکت دادن راکت

برای اینکه راکت به چپ و راست حرکت کند، از انقیادها[1]ی رویداد برای مقید کردن کلیدهای جهت‌نمای چپ و راست به توابع جدید در کلاس paddle استفاده می‌کنیم. زمانی که بازیکن جهت‌نمای چپ را فشار می‌دهد، متغیر x برابر با2- (حرکت به چپ) قرار داده می‌شود. فشار دادن کلید جهت‌نمای راست، متغیر x را برابر با2 (حرکت به راست)قرار خواهد داد.

اولین گام برای افزودن متغیر شیءx به تابع_int_ کلاس paddle و متغیری برای عرض بوم است همانگونه که برای کلاس Ball عمل کردیم:

```
def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
```

اکنون به توابعی برایتغییر جهت بین چپ(turn_left) و راست(turn_right) نیاز داریم. درنتیجه کد زیر را دقیقاً بعد از تابع draw اضافه می‌کنیم:

---

[1] binding

```
def turn_left(self, evt):
        self.x = -2
def turn_right(self, evt):
        self.x = 2
```

با دو خط زیر می‌توانیم این توابع را به کلید درستی در تابع_init_ کلاس، مقید کنیم. برای این که با فشرده شدن یک کلید، پایتون تابعی را فراخوانی کند، از انقیاد در بخش «کاری کنیم تا شیء نسبت به چیزی واکنش نشان دهد»استفاده می‌نماییم. در این مورد، با استفاده از نام رویداد'<KeyPress-Left>'، تابع turn_left کلاس Paddle را به کلید جهت‌نمای چپ مقید می‌نماییم. سپس با استفاده از نام رویداد'<KeyPress-Right>' تابع turn_right را کلید جهت‌نمای راست مقید می‌نماییم. درنتیجه تابع_init_ به شکل زیر خواهد بود:

```
def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
        self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
```

تابع draw برای کلاس Paddle مشابه کلاس Ball است:

```
def draw(self):
        self.canvas.move(self.id, self.x, 0)
        pos = self.canvas.coords(self.id)
        if pos[0] <= 0:
                self.x = 0
        elif pos[2] >= self.canvas_width:
                self.x = 0
```

با(self.canvas.move(self.id, self.x, 0 از تابع move بوم برای حرکت دادن راکت در جهت متغیر x استفاده می‌کنیم. سپس با استفاده از مقدار درpos، مختصات راکت را گرفته تا ببینیم که به وجه چپ صفحه برخورد کرده است یا راست.

راکت به جای بالا و پایین پریدن مثل توپ، بایستی حرکت نکند.بنابراین، زمانی که مختصات x چپ(pos[0]) کوچکتر یا مساوی(0 =>) 0 باشد آنگاه با self.x = 0 متغیر x را برابر با0 قرار می‌دهیم. به همین ترتیب، زمانی که مختصات x راست(pos[2]) بزرگتر یا مساوی عرض بوم است (=< self.canvas_width)، با self.x = 0 متغیر x را برابر با0 قرار می‌دهیم.

نکته:

اگر اکنون برنامه را اجرا کنید، بایستی قبل از این که بازی کنش‌های کلید جهت‌نمای راست و چپ را تشخیص دهد، روی بوم کلیک کنید. در نتیجه‌ی کلیک روی بوم، بوم متمرکز خواهد شد به این معنی که وقتی کسی یکی از کلیدهای صفحه کلید را فشار می‌دهد، فرمانی را دریافت می‌کند.

درک این که چه زمان توپ به راکت برخورد می‌کند

در این نقطه از کد، توپ به راکت برخورد نخواهد کرد؛ درواقع، توپ مستقیماً از روی راکت پرواز خواهد کرد. توپ بایستی بداند که چه زمان به راکت برخورد کرده است دقیقاً همانطوریک هباید بداند چه زمان به دیوار خورده است.

با اضاه کردن کد به تابع draw می‌توانیم این مسئله را حل کنیم (جایی که کدی را در اختیار داریم که دیوارها را وارسی می‌کند) ولی بهترین ایده این است که این نوع کد به توابع جدیدی انتقال داده شده تا به تکه‌های کوچکتری شکسته شود. اگر حجم زیادی از کد را یکجا داشته باشیم (بعنوان مثال، درون یک تابع) باعث می‌شویم تا درک کد دشوارتر گردد. اجازه دهید تغییرات لازم را اعمال کنیم.

ابتدا تابع_int_ توپ را تغییر داده بطوریکه بتوانیم شیءpaddle را بعنوان یک پارامتر انتقال دهیم

```
class Ball:
❶   def __init__(self, canvas, paddle, color):
        self.canvas = canvas
❷       self.paddle = paddle
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
```

توجه داشته باشید که در ❶ پارامتر_int_ را به گونه‌ای تغییر داده تا راکت را شامل شود. سپس در ❷ پارامتر paddle را به متغیر شیءpaddle نسبت می‌دهیم.

بعد از اینکه شیءpaddle را ذخیره کردیم، بایستی کد را تغییر دهیم، یعنی همان جایی که شیء ballرا ایجاد می‌کنیم. این تغییر در ته برنامه و دقیقاً قبل از حلقه‌ی اصلی صورت می‌گیرد.

```
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, paddle, 'red')

while 1:
        ball.draw()
        paddle.draw()
        tk.update_idletasks()
        tk.update()
        time.sleep(0.01)
```

کدی که بایستی بعد از برخورد توپ به راکت مشاهده کنیم اندکی پیچیده‌تر از کدی است که برخورد به دیوار را وارسی می‌کند. تابع hit_paddle را صدا زده و آن را به تابع draw کلاس Ball اضافه می‌کنیم جایی که برخورد توپ به پایین صفحه را وارسی می‌کنیم:

```
def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
                self.y = 3
        if pos[3] >= self.canvas_height:
                self.y = -3
        if self.hit_paddle(pos) == True:
                self.y = -3
        if pos[0] <= 0:
                self.x = 3
        if pos[2] >= self.canvas_width:
                self.x = -3
```

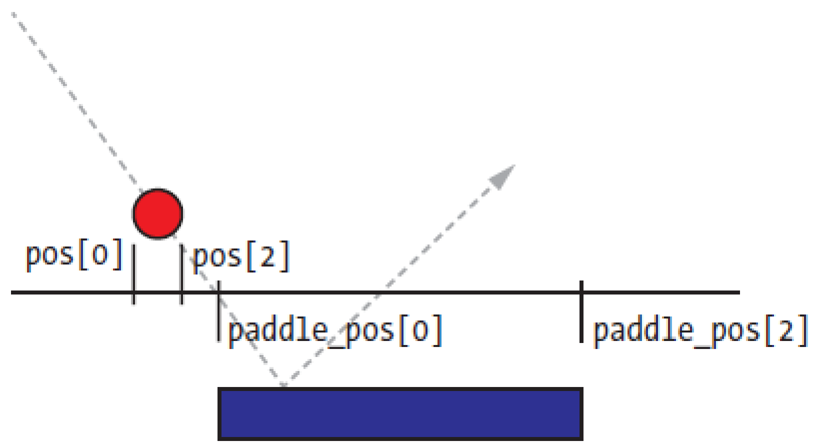
همانطوریکه در کد جدیدی که اضافه کردیم می‌بینیم اگر hit_paddle، True را برگرداند، با مقداردهی متغیر شیء با3- = self.y به3-، جهت توپ را تغییر می‌دهیم. ولی الآن نباید کد را اجرا کنید-هنوز تابع hit_paddle را نساخته‌ایم. اکنون این کار را انجام خواهیم داد.

تابع hit_paddle را دقیقاً قبل از تابع draw قرار دهید.

```
❶ def hit_paddle(self, pos):
❷ paddle_pos = self.canvas.coords(self.paddle.id)
❸ if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
❹ if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
```

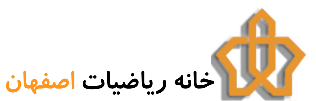
```
return True
return False
```

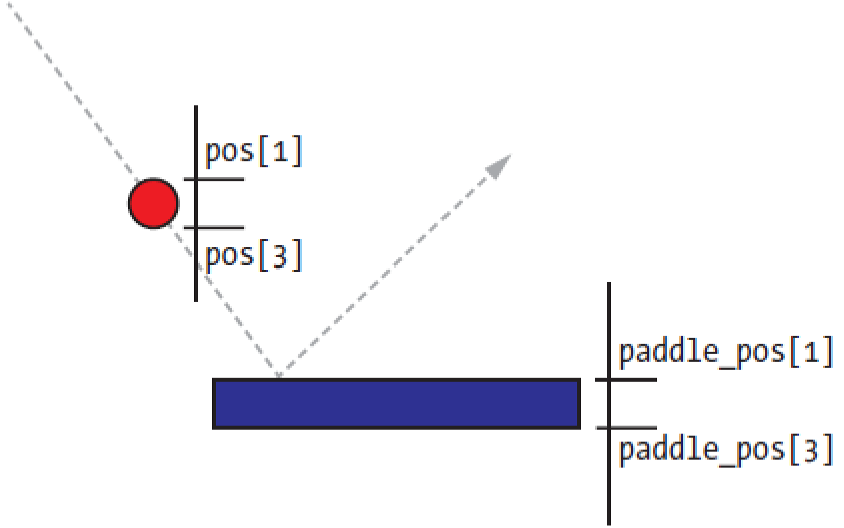
ابتدا، در ❶ تابعی با پارامتر pos را تعریف می‌کنیم.این خط حاوی مختصات جاری تـوپ اسـت. سپس در ❷ مختصات راکت را بدست آورده و آنها را در متغیر paddle_pos ذخیره می‌کنیم. در ❸ اولـین بخش از دستور if-then را در اختیار داریم و می‌گوییم «اگر سمت راست تـوپ بزرگتـر از سـمت چـپ راکت باشد و سـمت چـپ تـوپ کمتـر از سـمت راسـت راکـت باشـد... ». در اینجـا، pos[2] حـاوی مختصـات x بـرای سـمت راسـت تـوپ و pos[0] حـاوی مختصـات x بـرای سـمت چـپ آن اسـت. متغیر paddle_pos[0] حاوی مختصات x برای سمت چپ راکت و paddle_pos[2] حـاوی مختصـات x آن برای سمت راست است. نمودار زیر نشان می‌دهد که وقتی توپ در حال برخـورد کـردن بـه راکـت است، این مختصات‌ها چگونه خواهند بود:



توپ به طرف راکت می‌افتد ولی در این مورد می‌بینید که سمت راست توپ(pos[2]) هنـوز بـه سمت چپ راکت برخورد نکرده است(یعنی paddle_pos[0]).

در ❹ مـی‌بینـیم کـه تـه تـوپ(pos[3]) بـین روی راکـت(paddle_pos[1]) و تـه راکـت(paddle_pos[3]) اسـت. در نمـودار بعـدی مـی‌بینیـد کـه تـه تـوپ(pos[3]) هنـوز بـه روی راکت(padd;e_pos[1]) برخورد نکرده است.



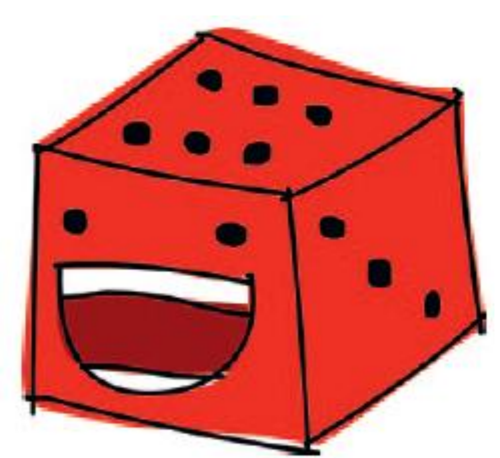


بنابراین براساس موقعیت فعلی توپ، تابع hit_paddle، false را برمی‌گرداند.

نکته:

به چه دلیل باید ببینیم آیا ته توپ بین بالا و زیر راکت قرار دارد یا نه؟ چرا اگر ته توپ به روی راکت برخورد کرده باشد، کفایت نمی‌کند؟ چون هر بار که توپ را روی بوم حرکت می‌دهیم، ۳ پیکسل می‌پریم. اگر فقط بررسی کنیم که توپ به روی راکت رسیده باشد(pos[1])،از روی آن موقعیت خواهیم پرید. در این مورد، توپ به حرکت خود ادامه خواهد داد و بدون توقف از درون راکت عبور خواهد کرد.

اضافه کردن المان شانس



حال زمان آن رسیده است که برنامه‌ی خود را به یک بازی تبدیل کنیم نه فقط یک توپ جهنده و یک راکت. بازی‌ها به المان شانس نیز نیاز دارند- راهی برای باختن بازیکن. در پایان بازی، توپ تا ابد بالا و پایین می‌پرد بنابراین باختی وجود ندارد. ما بازی را با اضافه کردن یک کد به پایان می‌بریم، کدی که می‌گوید اگر توپ به ته بوم برخورد کرد (به بیان دیگر اگر به زمین خورد)، بازی تمام است.

ابتدا متغیر شیء hit_bottom را به ته تابع _int_ کلاس Ball اضافه می‌کنیم:

```
self.canvas_height = self.canvas.winfo_height()
self.canvas_width = self.canvas.winfo_width()
self.hit_bottom = False
```

سپس، حلقه‌ی اصلی در ته برنامه را تغییر می‌دهیم:

```
while 1:
    if ball.hit_bottom == False:
        ball.draw()
        paddle.draw()
tk.update_idletasks()
tk.update()
time.sleep(0.01)
```

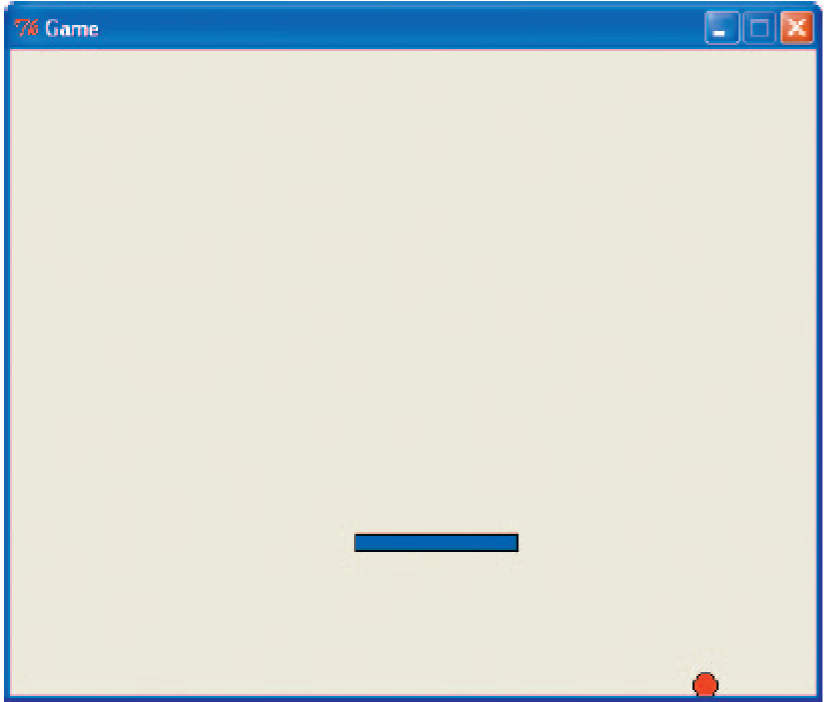
حال حلقه دائماً hit_bottom را بررسی کرده تا از برخورد توپ به ته صفحه نمایش آگاه شود. کد تنها زمانی باید به حرکت توپ و راکت ادامه دهد که توپ به ته صفحه نخورده باشد (همانطوریکه در دستور if دیدیم). بازی زمانی تمام می‌شود که توپ و راکت از حرکت بیافتند.

آخرین تغییر روی تابع draw کلاس Ball صورت می‌گیرد:

```
def draw(self):
    self.canvas.move(self.id, self.x, self.y)
    pos = self.canvas.coords(self.id)
    if pos[1] <= 0:
        self.y = 3
    if pos[3] >= self.canvas_height:
        self.hit_bottom = True
    if self.hit_paddle(pos) == True:
        self.y = -3
    if pos[0] <= 0:
        self.x = 3
    if pos[2] >= self.canvas_width:
        self.x = -3
```

دستور if را تغییر داده تا از برخورد توپ به ته صفحه آگاه شویم(یعنی اگر بزرگتر یا مساوی canvas_height باشد). دراینصورت، در خط زیر، hit_bottom را True می‌گیریم نه اینکه مقدار متغیر y را تغییر دهیم زیرا زمانی که توپ به ته صفحه برخورد می‌کند، نیازی به بالا و پایین پریدن آن نیست.

زمانی که برنامه را اجرا کرده و توپ به راکت برخورد نمی‌کند، لحظات متوقف شده و با برخورد توپ به ته بوم، بازی خاتمه می‌یابد:



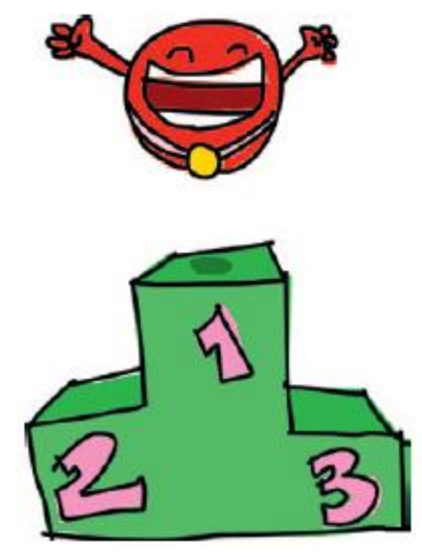
برنامه‌ی شما بایستی شبیه به کد زیر باشد.اگر در اجرای بـازی‌تـان مشـکل داشـتید، ببینیـد چـه چیزی در قبال این کد وارد کرده‌اید:

```
from tkinter import *
import random
import time
class Ball:
        def __init__(self, canvas, paddle, color):
                self.canvas = canvas
                self.paddle = paddle
                self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
                self.canvas.move(self.id, 245, 100)
                starts = [-3, -2, -1, 1, 2, 3]
                random.shuffle(starts)
                self.x = starts[0]
                self.y = -3
                self.canvas_height = self.canvas.winfo_height()
                self.canvas_width = self.canvas.winfo_width()
                self.hit_bottom = False
def hit_paddle(self, pos):
paddle_pos = self.canvas.coords(self.paddle.id)
if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
```

```
return True
return False
def draw(self):
self.canvas.move(self.id, self.x, self.y)
pos = self.canvas.coords(self.id)
if pos[1] <= 0:
self.y = 3
if pos[3] >= self.canvas_height:
self.hit_bottom = True
if self.hit_paddle(pos) == True:
self.y = -3
if pos[0] <= 0:
self.x = 3
if pos[2] >= self.canvas_width:
self.x = -3
class Paddle:
def __init__(self, canvas, color):
self.canvas = canvas
self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
self.canvas.move(self.id, 200, 300)
self.x = 0
self.canvas_width = self.canvas.winfo_width()
self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
def draw(self):
self.canvas.move(self.id, self.x, 0)
pos = self.canvas.coords(self.id)
if pos[0] <= 0:
self.x = 0
elif pos[2] >= self.canvas_width:
self.x = 0
def turn_left(self, evt):
self.x = -2
def turn_right(self, evt):
                    self.x = 2
tk = Tk()
tk.title("Game")
tk.resizable(0, 0)
```

```
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, paddle, 'red')
while 1:
if ball.hit_bottom == False:
ball.draw()
paddle.draw()
tk.update_idletasks()
tk.update()
        time.sleep(0.01)
```

آنچه آموختید



در این فصل، با استفاده از ماژول tkinter ساخت بازی را به پایان رساندیم. کلاس‌هایی را برای راکتی که در بازی از آن استفاده کردیم، ساختیم و از مختصات برای وارسی برخورد توپ به راکت یا دیواره‌های بوم بازی‌مان استفاده کردیم. از انقیاد رویداد برای مقید کردن کلیدهای جهت‌نمای راست و چپ به حرکت راکت و از حلقه‌ی اصلی برای فراخوانی تابع draw، برای متحرکت کردن آن استفاده کردیم. درنهایت نیز کد را تغییر داده تا المان شانس را به بازی اضافه کنیم بطوریکه وقتی بازیکن توپ را از دست می‌دهد، با برخورد کردن توپ به ته بوم، بازی تمام شود.

چیستان‌های برنامه‌نویسی

در این لحظه، بایز اندکی ساده‌تر شده است. برای خلق یک بازی حرفه‌ای تر، می‌توانید تغییرات بسیار بیشتری را اعمال کنید. سعی کنید کد خود را به شیوه‌های زیر ارتقاء دهید تا جالب‌تر شود و سپس پاسخ‌های خود را در *http://python-for-kids.com/* وارسی نمایید.

۱) تأخیر در شروع بازی

بازی اندکی سریع آغاز می‌شود و بایستی قبل از این که فشرده شدن کلیدهای جهت‌نمای چپ و راست روی صفحه‌ی کلید را تشخیص دهد، روی بوم کلیک کنید. آیا می‌توانید شروع بازی را به تأخیر

انداخته تا بازیکن زمان کافی برای کلیک روی بوم دراختیار داشته باشد؟ یا حتی بهتر، آیا می‌توانید انقیاد رویدادی را برای کلیک موس اضافه کنید، که بازی از آن به بعد شروع شود؟

نکته۱: درحال حاضر انقیادهای رویداد را به کلاس Paddle اضافه کرده‌اید بطوریکه مکان خوبی برای شروع خواهد بود.

نکته۲: انقیاد رویداد برای دکمه‌ی چپ موشواره، رشته‌ی '<Button-1>' است

۲) یک "Game Over" مناسب

با پایان یافتن بازی همه چیز از حرکت می‌افتد و این اتفاق اصلاً خوشآیند بازیکنان نیست. سعی کنید زمانی که توپ به ته صفحه برخورد می‌کند، متن "Game Over" را چاپ کنید. می‌توانید از تابع create_text استفاده کنید ولی شاید پارامتر بانام state نیز به درد شما بخورد (مقادیری همچون normal و hidden را می‌گیرد). نگاهی نیز به itemconfig در بخش «راه‌های بیشتر برای استفاده از شناسه» در صفحه‌ی ۱۸۸ بیاندازید. بعنوان یک چالش دیگر، تأخیر را طوری اضافه کنید که متن فوراً ظاهر نشود.

۳) سرعت بخشیدن به توپ

اگر تنیس بازی می‌کنید، می‌دانید وقتی که توپ به راکت شما برخورد می‌کند، برخی اوقات سریعتر از سرعتی که دریافت شده است، بسته به شدت ضربه‌ی شما، شلیک می‌شود. در این بازی، توپ با سرعت یکسان حرکت می‌کند صرفنظر از این که راکت حرکت کند یا نه. برنامه را طروی تغییر دهید که سرعت راکت بیشتر از سرعت توپ باشد.

۴) ثبت رکورد بازیکن

برای ثبت رکورد بازیکن چکار می‌کنید؟ هر بار که توپ به راکت برخورد می‌کند، امتیاز باید افزایش یابد. امتیاز را در گوشه‌ی بالا–راست بوم نمایش دهید. شاید بخواهید به itemconfig در بخش «راه‌های بیشتر برای استفاده از شناسه» در صفحه‌ی ۱۸۸ سری بزنید.

بخش۳

مسابقه‌ی آقای خطی برای رسیدن به درب خروج

فصل ۱۵

خلق گرافیک‌ها[1]یی برای بازی مرد خطی[2]

هنگام خلق یک بازی (یا هر برنامه‌ای) توسعه‌ی برنامه (طرح) ایده‌ی بسیار خوبی است. طرح شما بایستی چیزی را توصیف کند که بازی درباره‌ی آن نوشته شده است و همچنین المان‌ها و کاراکترهای اصلی بازی را توصیف می‌کند. وقتی زمان برنامه‌نویسی می‌رسد، توصیف شما به کمک خواهد کرد تا روی آنچه می‌خواهید توسعه دهید تمرکز نمایید. برنامه‌ی شما دقیقاً شبیه توصیف اصلی خواهد بود.

در این فصل یک بازی جالب به نام مسابقاتMr.Stick Man for Exit را خواهیم نوشت.

طرح بازیMr. Stick Man

در اینجا توصیفی از بازی جدید ارائه شده است:

- مأمور مخفیMr. Stick Man در مکان اسرارآمیزDr. Innocuous به دام افتاده است و شما می‌خواهید به او کمک کنید تا از طریق درب خروجی طبقه‌ی‌بالا فرار کند.
- بازی یک شکل خطی است که می‌تون از چپ به راست دیوده و بپرد. سکوهایی در ره طبقه وجود دارد که بایستی روی آن‌ها بپرد
- هدف بازی، رسیدن به درب خروجی است قبل از اینکه خیلی دیر شود و بازی پایان یابد.

براساس این توصیف می‌دانیم که به چندین تصویر شامل تصاویرMr. Stick Man، سکوها، و درب نیاز داریم. بوضوح مشخص است که برای کنار گذاشتن تمامی این‌ها به یک کد نیاز داریم ولی پیش از آن، درا ین فصل، گرافیک‌هایی را برای بازی خود می‌سازیم. بدین ترتیب، در فصل بعدی کاری هست که باید انجام دهیم.

در این بازی، المان‌های خود را چگونه می‌کشیم؟ می‌توانیم از همان گرافیک‌هایی که برای راکت و توپ جهنده در فصول قبل ایجاد کردیم استفاده کنیم ولی این گرافیک‌های برای این بازی بسیار ساده هستند. درعوض، می‌خواهیم یک بلوک پیکسل[1]بسازیم.

---

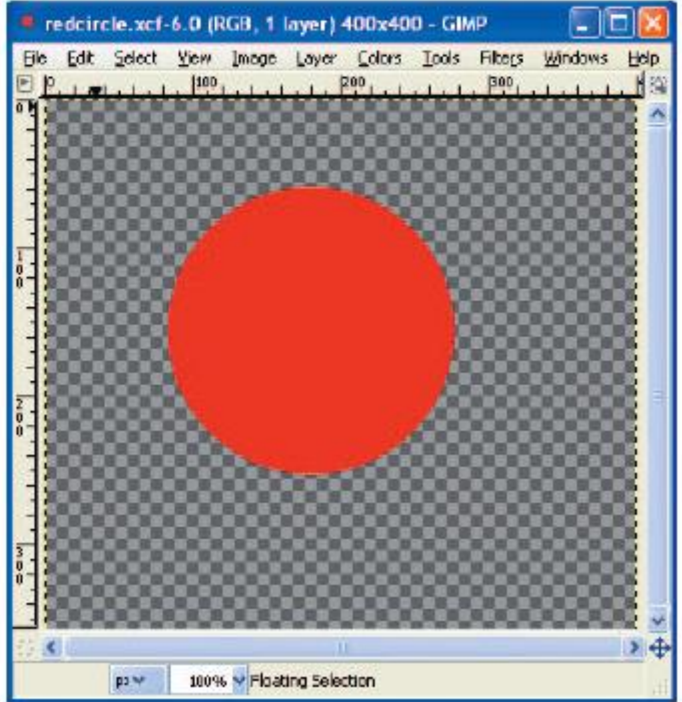
[1] Graphics نگاره

[2] MR. Stick Man مرد خطی

ارواح موجودات بازی هستند- معمولاً یک نوع شخصیت. بلوک پیکسل‌ها معمولاً از قبـل رنـدر شده‌اند به این معنی که از قبل ترسیم شده‌انـد (قبـل از اجـرای برنامـه) نـه اینکـه برنامـه بـا اسـتفاده از چندوجهی‌ها، مانند مورد بازی جهش توپ، خودش آن‌ها را خلق کند. یـک Mr. Stick Man یـک بلـوک پیکسل است و سکوها نیز همینطور.به مظنور خلق این تصاویر، بایستی یک برنامه‌ی نگاره‌سـازی[2] نصـب کنید.

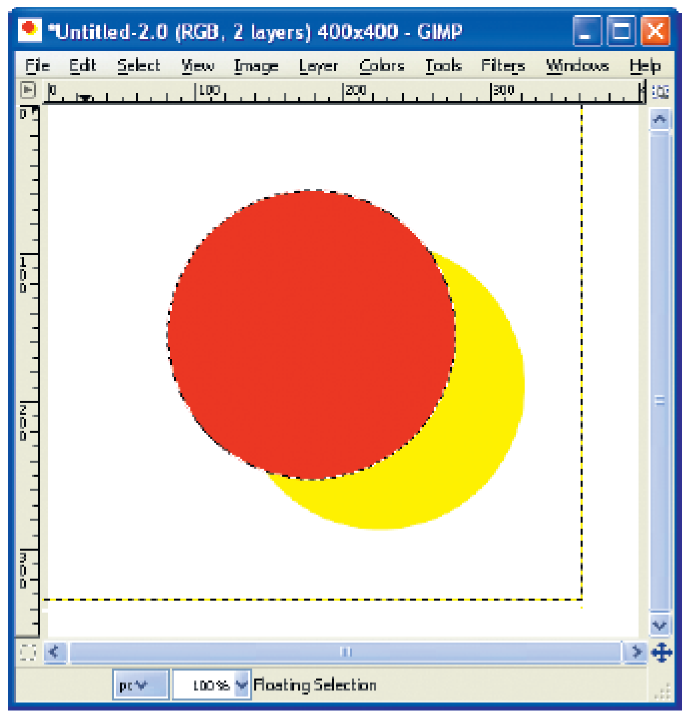
دستیابی به GIMP[3]

برنامه‌های نگاره‌سازی متعددی دردسترس هستند ولی برای این بازی، به برنامه‌ای نیاز داریم کـه از شفافیت[4] پشتیبانی کند(تحت عنوان کانال آلفا نیز شناخته می‌شود)درنتیجه تصاویر بخش‌هایی خواهنـد داشت که در آنجا هیچ رنگی درتصویر وجود ندارد. ما به تصاویری با بخش‌های شفاف نیاز داریـم زیـرا زمانیکه تصویری از روی تصویر دیگری عبور کرده یا به آن نزدیک می‌شود، نمی‌خواهیم پس‌زمینـه یـک تصویر بخشی از تصویر دیگر را بپوشاند. بعنوان مثال، در تصویر زیر، الگـوی شـطرنجی در پـس‌زمینـه، معرف ناحیه‌ی شفاف(ترانسپرنت) است:



[1] Sprite شبح
[2] graphics
[3] برنامه‌ی دستکاری عکس
[4] transparency

بنابراین اگر کل تصویر را کپی کرده و روی تصویر دیگری درج نماییم، پس‌زمینه هیچ چیزی را نمی‌پوشاند.



GIMP(*http://www.gimp.org/*) مخفف برنامه‌ی دستکاری تصویر GNU، یک برنامه نگاره-سازی رایگان برای Linux, Mac OS X, و Windows است که از تصاویر شفاف پشتیبانی می‌کند. آن را دانلود کرده و بصورت زیر نصب نمایید:

- اگر از ویندوز استفاده می‌کنید می‌توانید نصب‌کننده‌های ویندوزی را در صفحه‌ی پروژه GIMP-WIN در *http://gimp-win.sourceforge.net/stable.html* پیدا کنید.
- اگر از Ubuntu استفاده می‌کنید، GIMP را با باز کردن Ubuntu Software Center و وارد کردن gimp در کادر جستجو، نصب کنید. زمانی که GIMP Image Editor در نتایج جستجو ظاهر شد، روی دکمه‌ی Install کلیک کنید.
- اگر از Mac OS X استفاده می‌کنید، یک مجموعه اپلیکیشن را از *http://gimp.lisanet.de/Website/Download.html* دانلود نمایید.

همچنین بایستی یک دایرکتوری برای بازی بسازید. برای این کار، در فضای خالی روی دسک-تاپ کلیک راست نموده و **New ▸ Folder** را انتخاب نمایید(در Ubuntu ، گزینه **Create New Folder** است؛ در Mac OS X، گزینه **New Folder** است). در کادری که باز شده است، *stickman* را بعنوان نام فایل وارد کنید.

خلق المان‌های بازی

بعد از نصب برنامه‌ی نگاره‌سازی، آماده‌ی ترسیم شکل خواهید بود. این تصاویر را برای المان‌ها‌ی بازی خلق می‌کنیم:

- تصاویری برای یک پیکره‌ی خطی که می‌تواند به چپ و راست دویده و بپرد
- تصاویری برای سکو، در سه سایز مختلف
- تصاویری برای درب: یک درب باز و یک درب بسته
- تصویری برای پس‌زمینه‌ی بازی (چون پس‌زمینه‌ی سفید یا خاکستری ساده خسته‌کننده است)

قبل از شروع ترسیم، بایستی تصاویر را با پس‌زمینه‌های شفاف آماده کنیم.

تدارک یک تصویر شفاف

برای ایجاد یک تصویر شفاف- کانال آلفا- GIMP را آغاز کرده و مراحل زیر را دنبال می‌کنیم:

۱- **File ‣ New** را وارد کنید

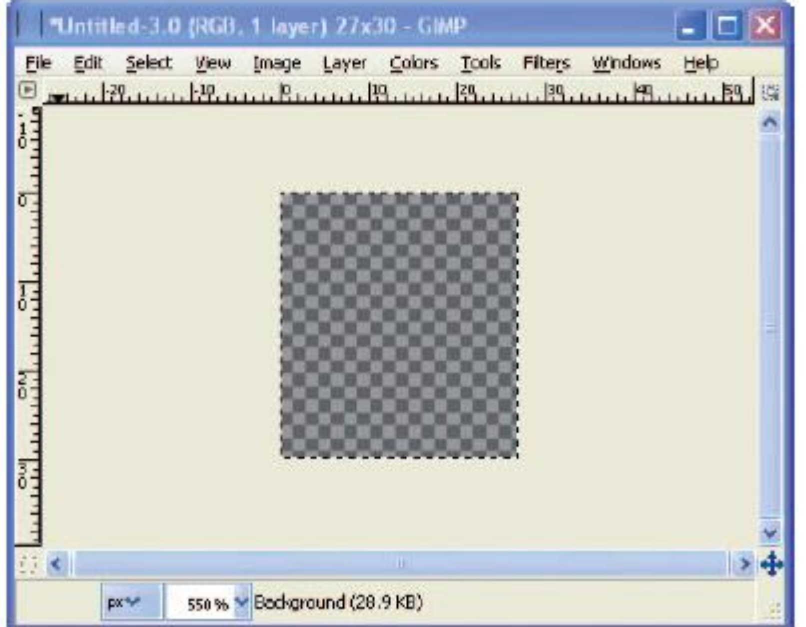
۲- در کادر محاوره، ۲۷پیکسل را برای عرض و ۳۰ پیکسل را برای طول تصویر وارد کنید

۳- **Layer ‣ Transparency ‣ Add Alpha Channel** را انتخاب کنید

٤- **Select ‣ All** را انتخاب کنید

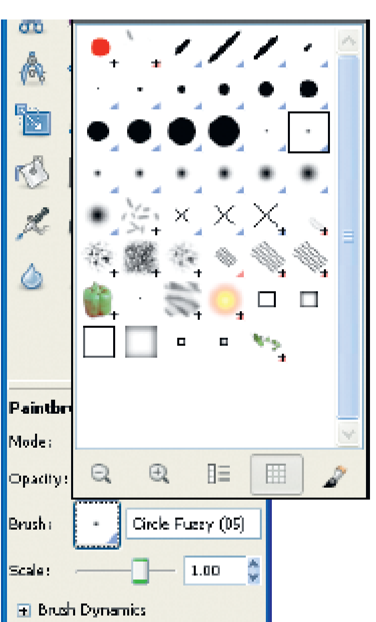
۵- **Edit ‣ Cut** را انتخاب کنید

نتیجه‌ی نهایی بایستی تصویری باشد که بصورت شطرنجی خاکستری تیره و روشن پر شده است(تصویر بزرگنمایی شده):



حال می‌توانیم خلق مأمور مخفی را شروع کنیم: Mr. Stick Man

کشیدن Mr. Stick Man



برای کشیدن اولین تصویر از پیکره‌ی خطی، روی ابزار Paintbrush[1] در GIMP Toolbox کلیک کرده و همانطوری که در سمت راست نشان داده شده است، در نوارابزار Brushes قلمویی را انتخاب کنید که شبیه یک نقطه‌ی کوچک است(معمولاً در پایین-راست صفحه).

سه تصویر مختلف (یا فریم[2]) را برای پیکره‌ی خطی خود ترسیم کرده تا دویدن و پریدن آن به سمت راست را نشان دهیم. از سه فریم برای متحرک ساختن Mr. Stick Man استفاده می‌کنیم، یعنی همان کاری که برای پویانمایی در فصل ۱۲ انجام دادیم.

اگر بزرگنمایی کرده و با دقت به این تصاویر نگاه کنیم، این چنین خواهد بود:

---

[1] قلموی نقاشی

[2] frame

تصاویر شما لزوماً نبایستی یکسان باشند بلکه بایستی شکل یک پیکره‌ی خطی با سه وضعیت حرکتی متفاوت باشند. به خاطر داشته باشید که هر کدام از آنها ۲۷پیکسل عرض و ۳۰ پیکسل ارتفاع دارند.

Mr. Stick Man به سمت راست می‌دود

ابتدا، یک سلسله فریم را برای دویدنMr. Stick Man به سمت راست می‌کشیم. اولین تصویر را بصورت زیر خلق می‌کنیم:

۱- کشیدن اولین تصویر (تصویر سمت چپ در نمایش قبلی)

۲- انتخاب**File ▸ Save As**

۳- در کادری که نمایش داده می‌شود، وارد کردن*figure-R1.gif* در محل نام. سپس کلیک روی دکمه‌ی علامت+ کوچک با برچسب**Select File Type**.

٤- انتخاب**GIF image** در لیستی که ظاهر می‌شود

۵- ذخیره‌ی فایل در دایرکتوری*stickman* که قبلاً ساخته بودید (کلیک روی **Browse for Other Folders** برای یافتن دایرکتوری صحیح)

برای خلق یک تصویر جدید ۳۰×۲۷ پیکسلی جدید مراحل قبل را دنبال کرده و سپس تصویرMr. Stick Man بعدی را می‌کشیم. این تصویر را با نام*figure-R2.gif* ذخیره می‌کنیم. این پروسه را برای تصویر نهایی نیز تکرار کرده و آن را با نام*figure-R3.gif* ذخیره می‌کنیم.

Mr. Stick Man به سمت چپ می‌دود

به جای این که نگاره‌ها را مجدداً برای حرکت پیکره‌ی خطی به سمت چپ، تکرار کنیم، می‌توانیم از GIMP برای چرخاندن فریم‌ها و حرکت Mr. Stick Man به سمت راست استفاده نماییم.

در GIMP، هر تصویر را به ترتیب باز کرده و سپس **Tools ▸ Transform Tools ▸ Flip** را انتخاب می‌کنیم. زمانی که روی تصویر کلیک می‌کنید، بایستی ببینید که از پهلو به پهلو می‌چرخد. تصویر را با نام‌های *figure-L1.gif*, *figure-L2.gif* و *figure-L3.gif* ذخیره کنید.

حال، شش تصویر را برای Mr. Stick Man خلق می‌کنیم ولی هنوز هم به تصاویری برای سکوها و درب خروج نیاز داریم.

ترسیم سکوها

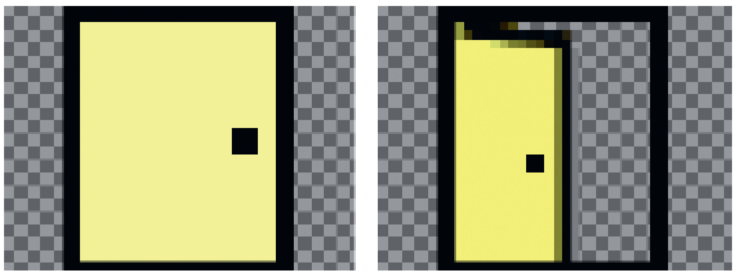
سه سکو با سه سایز مختلف می‌سازیم: ۱۰۰پیکسل عرض و ۱۰ پیکسل طول، ٦٠پیکسل عرض و ۱۰ پیکسل طول، و ۳۰ پیکسل عرض و ۱۰ پیکسل طول. هرطور که دوست دارید می‌توانید سکوها را ترسیم کنید ولی باید مطمئن شوید که پس‌زمینه‌ی آنها همانند تصاویر پیکره‌ی خطی، شفاف است.

در اینجا تصویر سه سکو نمایش داده شده است:

همانند تصاویر پیکره‌ی خطی، آن‌ها را در دایرکتوری *stickman* ذخیره می‌کنیم. کوچکترین سکو را *platform1.gif*، سکوی متوسط را *platform2.gif* و بزرگترین سکو را *platform3.gif* می‌نامیم.

ترسیم درب

سایز درب بایستی متناسب با سایز Mr. Stick Man باشد یعنی ۲۷پیکسل عرض در ۳۰ پیکسل ارتفاع، و به دو تصویر یکی برای درب بسته و دیگری برای درب باز نیاز داریم. درها شبیه تصویر زیر هستند(بزرگنمایی شده):



برای خلق این تصاویر، مراحل زیر بایستی دنبال شوند:

۱- روی جعبه رنگ پیش‌زمینه کلیک کرده (در پایین GIMP Toolbox) تا انتخابگر رنگ نمایش داده شود. رنگ موردنظرتان برای درب را انتخاب کنید. در سمت راست، مثالی با رنگ زرد نشان داده شده است.

۲- ابزار Bucket را انتخاب کرده (در Toolbox انتخاب شده است) و صفحه را با رنگ منتخب‌تان پر کنید.

۳- رنگ پیش‌زمینه را به سیاه تغییر دهید.

٤- ابزار Pencil یا Paintbrush را انتخاب کرده (در سمت راست ابزار Bucket) و خط سیاه رنگ دور درب و دستگیره‌ی در را بکشید.

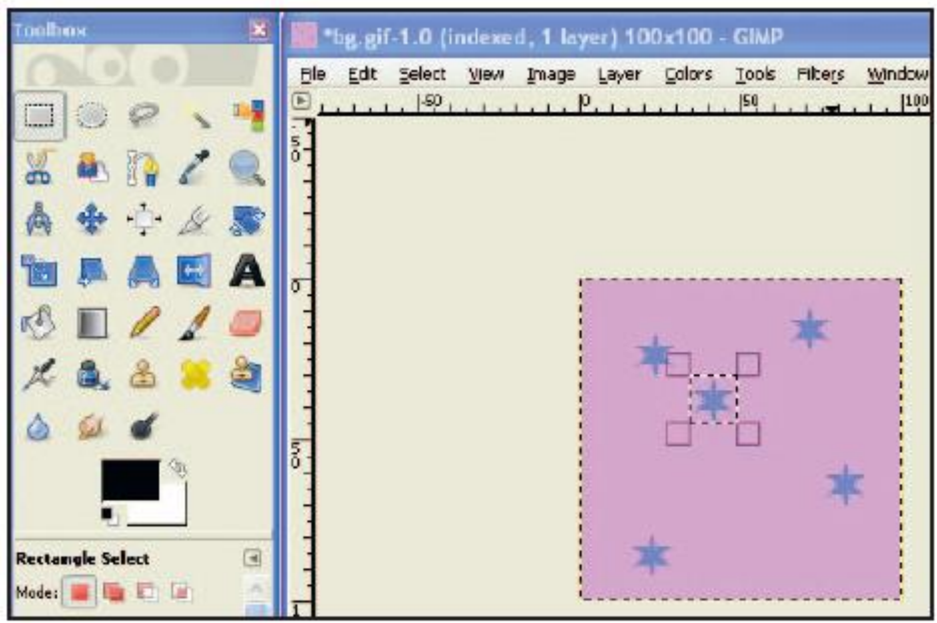
۵- آنها را در دایرکتوری *stickman* ذخیره کرده و آن‌ها را *door1.gif* و *door2.gif* نامگذاری می‌کنیم.

ترسیم پس‌زمینه

پس‌زمینه آخرین تصویری است که بایستی بسازیم. این تصویر را ۱۰۰پیکسل عرض در۱۰۰ پیکسل طول می‌سازیم. به پس‌زمینه‌ی شفاف نیازی نیست زیرا آن را با رنگی پر می‌کنیم که کاغذ دیواری پس‌زمینه پشت دیگر المان‌های بازی خواهد بود.

برای خلق یک پس‌زمینه، **File ▸ New** را انتخاب کرده و سایز تصویر را ۱۰۰×۱۰۰ پیکسل درنظر می‌گیریم. یک رنگ شیطانی برای کاغذ دیواری مخفی‌گاه villian استفاده می‌کنیم. صورتی تیره را بعنوان رنگ انتخاب می‌کنیم.

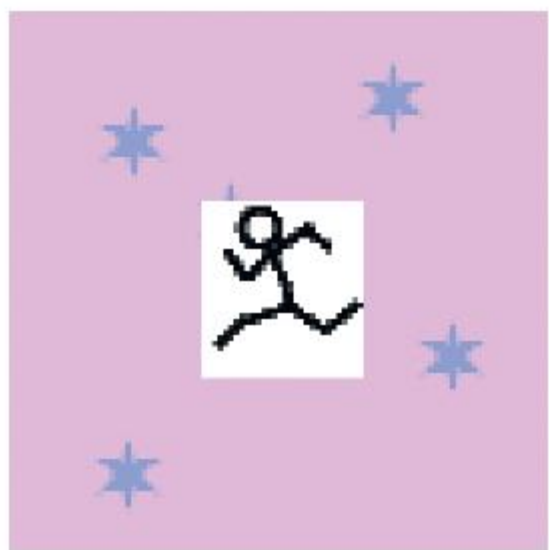
می‌توانید کاغذ دیواری را با گل، ستاره، راه‌راه و ... تزئین کنید-هر مدلی که فکر می‌کنید مناسب بازی شما است. بعنوان مثال، اگر می‌خواهید ستاره‌ها به کاغذدیواری اضافه کنید، رنگ دیگری را انتخاب نمایید، ابزارPencil را انتخاب کرده و اولین ستاره را بسازید. سپس از ابزارSelection برای انتخاب کادر دور ستاره استفاده کرده و آن را دور تصویر کپی و درج نمایید (ابتدا**Edit ▸ Copy** و سپس **Edit ▸ Paste** را انتخاب کنید). بایستی بتوانید با کلیک روی تصویر درج شده، آن را به صفحه انتقال دهید. در اینجا مثالی با چند ستاره و ابزارSelection انتخاب شده درToolbox وجود دارد.



هر وقت از ترسیمات خود راضی بودید، تصویر را با نام *background.gif* در دایرکتوری *stickman* ذخیره کنید.

شفافیت

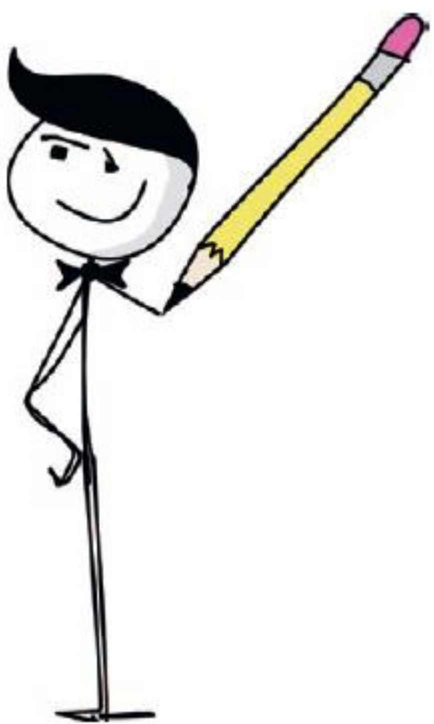
با گرافیک‌هایی که ساختیم، می‌توانید به ایده‌ی بهتری از نیاز به شفافیت در تصاویر (به جای پس‌زمینه) دست پیدا کنید. اگر Mr. Stick Man را جلوی کاغذ دیواری پس‌زمینه قرار داده و پس‌زمینه‌ی شفافی نداشته باشد، چه اتفاقی خواهد افتاد؟ پاسخ در اینجا است:



پس‌زمینه‌ی سفید Mr. Stick Man بخشی از کاغذ دیواری را می‌پوشاند. ولی اگر از تصویر شفاف استفاده کنیم، خواهیم داشت:

تصویر پیکره‌ی خطی هیچ چیزی را در پس‌زمینه محو نخواهد کرد مگر چیزی که خودش آن را پوشانده است. این کار خیلی حرفه‌ای‌تر است!

آنچه آموختید



در این فصل، یادگرفتید که چگونه یک طرح پایه برای یک بازی نوشته (در این مورد «مسابقه‌ی Mr. Stick برای رسیدن به درب خروج») و از کجا شروع کنید. از آنجاییکه قبل از خلق یک بازی به المان‌های گرافیک‌ها نیاز داریم، از برنامه‌ی نگاره‌سازی برای خلق نگاره‌های اصلی برای بازی‌مان استفاده کردیم. در این فرآیند، آموختید که چگونه پس‌زمینه‌ی این تصاویر را شفاف کرده و درنتیجه دیگر تصاویر موجود در صفحه را نپوشانند.

در فصل بعد، برخی کلاس‌ها را برای بازی‌مان خلق خواهیم کرد.

فصل۱٦

توسعه‌ی بازی Mr. Stick Man

حال که تصاویری را برای بازی «مسابقه‌ی آقای خطی برای رسیدن بـه درب خـروج» سـاختیم، می‌توانیم توسعه‌ی کد را آغاز کنیم. با توجه به توصیف بازی در فصل قبل، ایـده‌ای از آنچـه بـه آن نیـاز داریم بدست خواهیم آورد: یک پرکره‌ی خطی که می‌تواند بدود و بپرد و سکوهایی که بایـد روی آن‌هـا بپرد.

همچنین به کدی نیاز داریم که پیکره‌ی خطی را نمـایش دهـد، آن را در صـفحه جابجـا کنـد و سکوها را بکشد. ولی قبل از نوشتن این کد، بایستی بومی را برای نمایش تصویر پس‌زمینه خلق کنیم.

ایجاد کلاس GAME

ابتدا، کلاسی به نامGame را می‌سازیم که کنترلر اصلی ما می‌باشـد. کـلاسGame حـاوی یـک تابع_init_ برای تعیین مقادیر اولیه‌ی بازی و یک تابعmainloop برای انجام پویانمایی می‌باشد.

تعیین عنوان پنجره و ایجاد بوم

در اولین بخش تابع_init_، عنوان پنجره را مشـخص کـرده و بـوم را مـی‌سـازیم. همانطوریکـه مشاهده خواهید نمود، این بخش از کد مشابه کدی است کـه بـرای بـازی «جهـش تـوپ»! در فصـل۱۳ نوشتیم. ادیتور[1]را باز کرده و کد زیر را وارد کنید، سپس فایل خـود را بـا نـام*stickmangame.py* ذخیـره کنید. مطمئن شوید که آن را در دایرکتوری ذخیره کرده‌اید که در فصل۱۵ آن را ساختیم(به نام*stickman*).

[1] Editor ویرایشگر

```
from tkinter import *
import random
import time
class Game:
def __init__(self):
self.tk = Tk()
self.tk.title("Mr. Stick Man Races for the Exit")
self.tk.resizable(0, 0)
self.tk.wm_attributes("-topmost", 1)
self.canvas = Canvas(self.tk, width=500, height=500, ₩
highlightthickness=0)
self.canvas.pack()
self.tk.update()
self.canvas_height = 500
self.canvas_width = 500
```

در نیمه‌ی اول این برنامه (از * from tkinter import تا self.tk.wm_attributes)، شیء **tk** را ساخته و سپس با استفاده از self.tk.title عنوان پنجره را("Mr. Stick Man Races for the Exit") قرار می‌دهیم. با فراخوانی تابع resizable، پنجره را ثابت نگه میداریم (بنابراین امکان تغییرسایز آن وجود ندارد) و سپس با استفاده از تابع wm_attributes پنجره را جلوی تمامی پنجره‌ها قرار می‌دهیم.

در ادامه، در خط self.canvas = Canvas بوم را ساخته و توابع **pack** و **update** شیء **tk** را صدا می-زنیم. درنهایت نیز، دو متغیر **height** و **width** را برای کلاس **Game** می‌سازیم تا بتوانیم طول و عرض بوم را ذخیره کنیم.

نکته:

از بک‌اسلش(\)[1] در خط *self.canvas = Canvas* فقط برای جداسازی خط طولانی کد استفاده شده است. این کار ضرورتی ندارد ولی از آنجاییکه کل خط در یک صفحه جا نمی‌شود،از آن استفاده کرده-ایم.

خاتمه‌ی تابع _init_

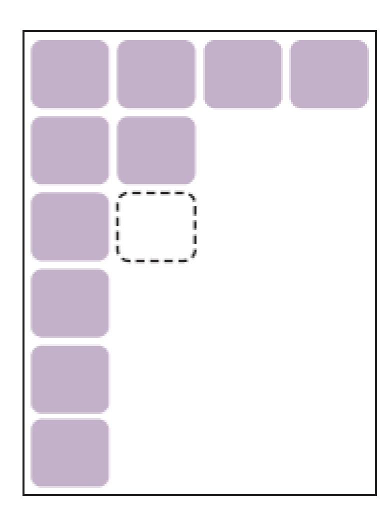
بقیه‌ی تابع _init_ را در فایل *stickfiguregame.py* قرار می‌دهیم که شما آن را ساخته‌اید. این کد، تصویر پس‌زمینه را بارگذاری کرده و سپس آن را روی بوم نمایش می‌دهد.

```
self.tk.update()
```

[1] Backslash

```
self.canvas_height = 500
self.canvas_width = 500
❶ self.bg = PhotoImage(file="background.gif")
❷ w = self.bg.width()
h = self.bg.height()
❸ for x in range(0, 5):
❹ for y in range(0, 5):
❺ self.canvas.create_image(x * w, y * h, ₩
image=self.bg, anchor='nw')
❻ self.sprites = []
self.running = True
```

در ❶، متغیر bg را می‌سازیم که حاوی شیء PhotoImage است- فایل تصویر پس‌زمینه به نام *background.gif* که‌آن را در فصل ۱۵ ساختیم. سپس در ❷، طول و عرض تصویر را در متغیرهای w و h ذخیره می‌کنیم. توابع width و height کلاس PhotoImage سایز تصویر را به مجرد بارگذاری، برمی-گردانند.



در ادامه شاهد دو حلقه در این تابع هستیم. باری درک کاری که انجام می‌دهند، تصور کنید یک مهر لاستیکی مربعی، یک صفحه‌ی مرکب و یک تکه‌ی بزرگ کاغذ در اختیار دارید. چگونه با استفاده از مهر، صفحه‌ی کاغذ را با مربع‌های رنگی پر می‌کنید؟ شما می‌توانید بصورت تصادفی صفحه را مهر زده تا صفحه پر شود. نتیجه‌ی کار آشفته بوده و پر کردن صفحه مدت زمان زیادی طول می‌کشد ولی در نهایت پر خواهد شد. یا این که، همانطوریکه در شکل سمت راست نشان داده شده است، می‌توانید بصورت ستونی از بالا به پایین صفحه را مهر زده و همین‌کار را تکرار کنید تا صفحه پر شود.

تصویر پس‌زمینه‌ای که در فصل قبل ساختیم، مهر ما خواهد بود. می‌دانیم که بوم ۵۰۰×۵۰۰ پیکسل است و تصویر پس‌زمینه‌ی یک مربع ۱۰۰ پیکسلی ساخته‌ایم. این به ما نشان می‌دهد که برای پر کردن صفحه با تصاویر، به ۵ ستون و ۵ سطر نیاز داریم. در ❸ از حلقه برای محاسبه‌ی ستون‌ها و در ❹ از حلقه برای محاسبه‌ی سطرها استفاده شده است.

در ❺ متغیر حلقه‌ی اول(x) را در عرض تصویر(w) ضرب کرده(x * w) تا ببینیم در عرض چقدر ترسیم کرده‌ایم و متغیر حلقه‌ی دوم(y) را در طول تصویر(h) ضرب کرده(y * h) تا ببینیم چقدر در طول

صفحه ترسیم نموده‌ایم. با از تابع create_image شیء canvas (self.canvas.create_image) برای ترسیم تصویر روی صفحه با این مختصات استفاده می‌کنیم.

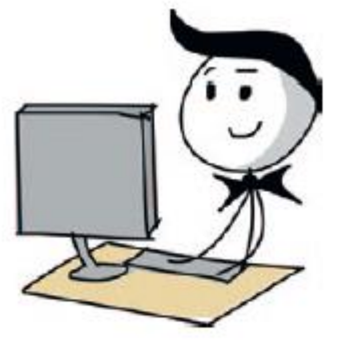
درنهایت، در ❻ متغیرهای sprites (حاوی که یک فهرست)و running (حاوی مقدار بولین Ture) را می‌سازیم. بعداً از این متغرها در کد برنامه استفاده خواهیم کرد.

ایجاد تابع MAINLOOP

از تابع mainloop در کلاس Game برای متحرک‌سازی بازی استفاده می‌کنیم. این تابع بسیار شبیه به حلقه‌ی اصلی(یا حلقه‌ی پویانمایی)است که برای بازی جهش توپ! در فصل ۱۳ استفاده کردیم:

```
for x in range(0, 5):
for y in range(0, 5):
self.canvas.create_image(x * w, y * h, ₩
image=self.bg, anchor='nw')
self.sprites = []
self.running = True
def mainloop(self):
❶ while 1:
❷ if self.running == True:
❸ for sprite in self.sprites:
❹ sprite.move()
❺ self.tk.update_idletasks()
self.tk.update()
time.sleep(0.01)
```

در ❶، حلقه‌ی loop ی را می‌سازیم که تا بسته شدن پنجره همچنان اجرا خواهد شد. سپس در ❷، رابطه‌ی running=True را بررسی می‌کنیم. دراینصورت، در ❸، در اسپریت‌های لیست اسپریت‌ها (self.sprites)حلقه می‌زنیم و در ❹ تابع move را برای هریک از آن‌ها صدا می‌زنیم. (البته هنوز اسپریتی درست نشده است بنابراین درصورت اجرای برنامه، این کد هیچ کاری انجام نخواهد داد، ولی بعداً به درد خواهد خورد).



سه خط آخر تابع که در ❺ شروع می‌شود، شیء tk را ملزم کرده تا مجدداً صفحه را ترسیم کرده و برای کسری از ثانیه غیرفعال شود، همانطوریکه در مورد بازی جهش توپ! در فصل ۱۳ دیدیم.

برای این که این کد اجرا شود دو خط زیر را اضافه کرده (توجه

داشته باشید که برای این دو خط نیازی به تورفتگی[1] نیست) و فایل را ذخیره نمایید.

```
g = Game()
g.mainloop()
```

نکته:

مطمئن شوید کد به ته فایل بازی شما اضافه شده است. همچنین مطمئن شوید که تصاویر شما در همان دایرکتوری قرار دارند که فایل پایتون قرار دارد. اگر دایرکتوری مردخطی را در فصل ۱۵ ایجاد کرده و تمامی تصاویرتان را در آنجا ذخیره نموده‌اید، فایل پایتون این بازی نیز بایستی همان جا باشد.

این کد یک شیء از کلاس Game ایجاد کرده و آن را بصورت متغیر g ذخیره می‌کند. سپس تابع mainloop را در شیء جدید فراخوانده تا صفحه را ترسیم نماید.

بعد از این که برنامه را ذخیره کردید، با انتخاب **Run ▸ Run Module** آن را در IDLE اجرا کنید.

شاهد هستید که پنجره‌ای ظاهر می‌شود که در آنجا بوم با تصویر پس‌زمینه پر شده است.

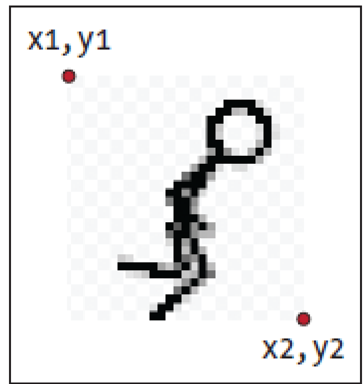
یک پس‌زمینه‌ی زیبا برای بازی‌مان انتخاب کرده‌ایم و حلقه‌ی پویانمایی را ساخته‌ایم که اسپریت‌ها را برای ما ترسیم خواهند کرد (بعد از اینکه آنها را خلق کردیم).

ایجاد کلاس‌های COORDS

[1] indentation

حال کلاسی را خلق می‌کنیم که از آن برای تعریف موقعیت چیزی در صفحه بازی استفاده خواهیم کرد. این کلاس می‌تواند مختصات بالا-چپ(x1وy1) وپایین-راست(x2 وy2)هر جزء از بازی را ذخیره نماید.

در اینجا نشان خواهیم داد که چگونه موقعیت این تصویر از پیکره‌ی خطی با استفاده از این مختصات ثبت می‌شود:



کلاسی که ساختیم راCoords می‌نامیم و فقط حاوی یک تابع_init_ است که در آن، چهار پارامتر وجود دارد(x1,y1,x2,y2). در اینجا کدی نشان داده شده است که باید اضافه شود (آن را در ابتدای فایل *stickmangame.py* قرار می‌دهیم):

```
class Coords:
def __init__(self, x1=0, y1=0, x2=0, y2=0):
self.x1 = x1
self.y1 = y1
self.x2 = x2
self.y2 = y2
```

توجه داشته باشید که هر پارامتر بصورت یک متغیر شیء با همان نام ذخیره شده است(x1,y1,x2,y2). از شیء‌های این کلاس به اختصار استفاده خواهیم کرد.

وارسی برخوردها

حالا که فهمیدید چگونه می‌توان موقعیت اسپریت‌های بازی را ذخیره کرد، به راهی برای بیان برخورد دو اسپریت با هم نیاز داریم، همانند زمانی کهMr. Stick Man در صفحه می‌پرد و به یکی از سکوها برخورد می‌کند. برای این که حل این مسئله ساده‌تر شود، می‌توانیم آن را به دو مسئله‌ی کوچکتر تقسیم کرده و برخورد افقی یا عمودی اسپریت‌ها را بررسی کنیم. سپس می‌توانیم دو راه‌حل کوچکتر را با هم ترکیب کرده تا ببینیم اسپریت‌ها در چه جهتی با هم برخورد کرده‌اند.

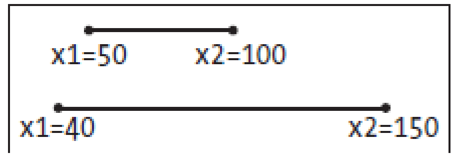
اسپریت‌هایی که افقی با هم برخورد می‌کنند

ابتدا، تابع within_x را ساخته تا ببینیم آیا یک مجموعه از مختصات x (x1 وx2) مجموعه مختصات دیگری از x (بازهم x1 و x2) را قطع کرده است یا خیر. چندین راه برای انجام این کار وجود دارد، ولی در اینجا رویکرد ساده‌ای ارائه شده است که می‌توانید آن را زیر کلاس Coords اضافه کنید:

```
class Coords:
def __init__(self, x1=0, y1=0, x2=0, y2=0):
self.x1 = x1
self.y1 = y1
self.x2 = x2
self.y2 = y2
def within_x(co1, co2):
❶ if co1.x1 > co2.x1 and co1.x1 < co2.x2:
❷ return True
❸ elif co1.x2 > co2.x1 and co1.x2 < co2.x2:
❹ return True
❺ elif co2.x1 > co1.x1 and co2.x1 < co1.x2:
return True
❻ elif co2.x2 > co1.x1 and co2.x2 < co1.x1:
return True
{ else:
| return False
```

تابع within_x پارامترهای co1 و co2 را می‌گیرد که هردو شیءهای Cooeds هستند. در ❶ بررسی می‌کنیم که آیا چپ‌ترین جایگاه اولین شیء مختصات (co1,x1) بین موقعیت منتهی علیه سمت چپ (co2.x1) و موقعیت منتهی علیه سمت راست (co2.x2) دومین شیء مختصات قرار دارد یا خیر. در اینصورت در ❷، True را برمی‌گردانیم.

اجازه دهید برای درک این روند، نگاهی به دو خطی بیندایم که در آن‌ها مختصات‌های x همپوشی دارند. هر خط در x1 آغاز شده و در x2 پایان می‌یابد.



اولین خط در این دیاگرام(co1)در موقعیت پیکسل ۵۰(x1) آغاز شده و در موقعیت ۱۰۰(x2) خاتمه می‌یابد. خط دوم(co2) در موقعیت ٤٠ آغاز و در موقعیت ۱۵۰ خاتمه می‌یابد. در این مورد، از آنجاییکه موقعیت x1 خط اول بین موقعیت‌های x1 و x2 خط دوم قرار دارد، اولین دستور if در تابع، برای این دو مجموعه مختصات، true خواهد بود.

با elif در ❸، می‌بینیم که موقعیت منتهی علیه سمت راست اولین خط(co1,x1) بین موقعیت منتهی علیه سمت چپ(co2,x1) و موقعیت منتهی علیه سمت راست(co2,x2) خط دوم قرار دارد. دراینصورت، در ❸، True را برمی‌گردانیم. دو دستور elif در ❺ و ❻ تقریباً یک کار را انجام می‌دهند: آنها موقعیت‌های منتهی علیه سمت راست و چپ خط دوم (co2) را با خط اول (co1) مقایسه می‌کنند.

درصورتیکه هیچکدام از دستورات if تطابق نداشته باشند، به else در ❼ رسیده و در ❽، False را برمی‌گردانیم. به این معنی که «نه، دو شیء مختصات بصورت افقی یکدیگر را قطع نمی‌کنند».

برای دیدن نمونه‌ای از عملکرد این تابع، به دیاگرامی رجوع کنید که خطوط اول و دوم را نشان می‌دهد. موقعیت‌های x1 و x2 اولین شیء مختصات، 100, 40 هستند و موقعیت‌های x1 و x2 ی دومین شیء مختصات 150, 50 می‌باشند. در اینجا می‌بینیم وقتی تابع within_x را صدا می‌زنیم چه اتفاقی خواهد افتاد:

```
>>> c1 = Coords(40, 40, 100, 100)
>>> c2 = Coords(50, 50, 150, 150)
>>> print(within_x(c1, c2))
True
```

تابع True را برمی‌گرداند. این اولین گام به سمت امکان تعیین برخورد دو اسپریت است. بعنوان مثال، زمانی که یک کلاس برای Mr. Stick Man و سکوها ساختیم خواهیم توانست برخورد مختصات-های x آن‌ها را نشان دهیم.

این شیوه‌ی خوبی برای برنامه‌نویسی نیست چون از دستورات متعدد if یا else استفاده شده است که یک مقدار را برمی‌گردانند. برای حل این مسئله، با قراردادن هر یک از شرط‌ها در پرانتز، و جدا کردن آن‌ها با کلیدواژه، می‌توانیم تابع within_x را کوتاه کنیم. اگر خواهان تابع شسته رفته‌تری، با خطوط کد کمتری هستید می‌توانید تابع را طوری تغییر دهید تا به شکل زیر درآید:

```
def within_x(co1, co2):
if (co1.x1 > co2.x1 and co1.x1 < co2.x2) ₩
or (co1.x2 > co2.x1 and co1.x2 < co2.x2) ₩
or (co2.x1 > co1.x1 and co2.x1 < co1.x2) ₩
or (co2.x2 > co1.x1 and co2.x2 < co1.x1):
return True
```

```
else:
return False
```

همانطوریکه در بالا نشان داده شده است، به جای اینکه دستور if در یک خط باشد و کار تمام شود، با استفاده از یک بک‌اسلاش(\) آن را در چندین خط بسط می‌دهیم.

برخورد عمودی اسپریت‌ها

همچنین بایستی از برخورد عمودی اسپریت‌ها آگاه شویم. تابع within_y بسیار شبیه به تابع within_x است. برای ایجاد آن، برخورد موقعیت y1 مختصات اول با موقعیت‌های y1,y2 مختصات دوم و بالعکس را بررسی می‌کنیم. در اینجا تابعی وجود دارد که آن را اضافه خواهیم کرد (آن را زیر تابع within_x قرار می‌دهیم)- این بار آن را با استفاده از نسخه‌ی کوتاهتری از کد خواهیم نوشت (به جای چندین دستور if):

```
def within_y(co1, co2):
        if (co1.y1 > co2.y1 and co1.y1 < co2.y2) ₩
                or (co1.y2 > co2.y1 and co1.y2 < co2.y2) ₩
                or (co2.y1 > co1.y1 and co2.y1 < co1.y2) ₩
                or (co2.y2 > co1.y1 and co2.y2 < co1.y1):
            return True
        else:
            return False
```

کنارهم گذاشتن تمامی این موارد: کد نهایی کشف برخورد

بعد از این که برخورد یک مجموعه از مختصات‌های x با مجموعه‌ی دیگری را تعیین کردیم، و همین کار را برای مختصات‌های y انجام دادیم، می‌توانیم تابعی برای تعیین برخورد اسپریت‌ها و جهت برخورد آنها بنویسیم. این کار را با توابع collided_left، collided_right، collided_top و collided_bottom انجام خواهیم داد.

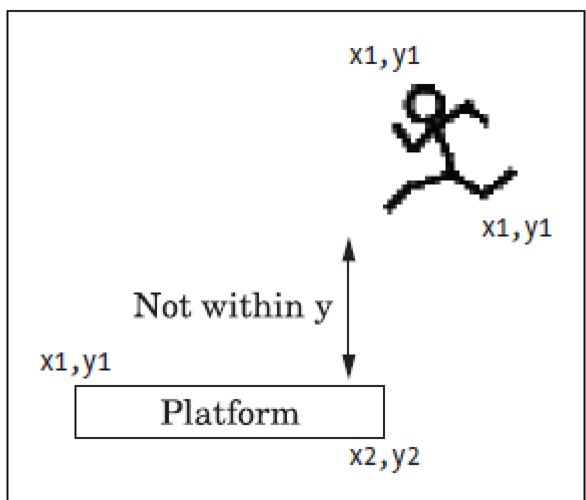
تابع collided_left

در اینجا کدی برای تابع collided_left ارائه شده است که می‌توانید آن را به ته دو تابع within اضافه کنید:

```
❶ def collided_left(co1, co2):
❷ if within_y(co1, co2):
❸ if co1.x1 <= co2.x2 and co1.x1 >= co2.x1:
❹ return True
❺ return False
```

این تابع به ما می‌گوید که طرف چپ (مقدار x1) شیء مختصات اول به شیء مختصات دیگری برخورد کرده است.

تابع دو پارامتر دارد: co1(شیء مختصات اول) و co2(شیء مختصات دوم). همانطوریکه در ❶می‌بینیم، برخورد عمودی دو شیء مختصات را با استفاده از تابع within_y در ❷ بررسی می‌کنیم. بعد از تمامی این کارها، اگر Mr. Stick Man has بالای سکو شناور باشد، نمی‌توان برخورد او با سکو را تعیین کرد:



در ❸، برخورد مقدار موقعیت منتهی علیه سمت چپ شیء مختصات اول(co1,x1) با موقعیت x2 دومین شیء مختصات (co2,x2) را تعیین می‌کنیم- یعنی کوچکتر یا مساوی موقعیت x2 باشد. همچنین بررسی کرده تا مطمئن شویم از موقعیت x1 گذر نکرده است. اگر به پهلو خورده باشد، True را در ❹ برمی‌گردانیم. اگر هیچیک از دستورات if، true نباشند، در ❺، False را برمی‌گردانیم.

تابع collided_right

تابع collided_right بسیار شبیه به collided_left است:

```
def collided_right(co1, co2):
❶ if within_y(co1, co2):
```

```
❷ if co1.x2 >= co2.x1 and co1.x2 <= co2.x2:
❸ return True
❹ return False
```

همانند collided_left، با استفاده از تابع within_y در ❶ ، برخورد مختصات‌های y با یکدیگر را بررسی می‌کنیم. سپس در ❷ بررسی می‌کنیم که آیا موقعیت x2 بین x1 و x2 دومین شیء مختصات قرار دارد یا نه. در اینصورت در ❸، True را برمی‌گردانیم و درغیراینصورت در ❹، False را برخواهیم گرداند.

تابع collided_top

تابع collided_top بسیار شبیه به دو تابعی است که قبلاً اضافه کردیم.

```
def collided_top(co1, co2):
❶ if within_x(co1, co2):
❷ if co1.y1 <= co2.y2 and co1.y1 >= co2.y1:
return True
return False
```

ولی با این تفاوت که این بار با استفاده از تابع within_x در ❶ برخورد افقی مختصات‌ها را بررسی می‌کنیم. سپس در ❷، برخورد موقعیت منتهی علیه بالای مختصات اول (co1,y1) با موقعیت y2 مختصات دوم را بررسی کمی‌کنیم نه موقعیت y1 آن را. دراینصورت True را برمی‌گردانیم (به این معنی که بله، بالای مختصات اول به مختصات دوم برخورد کرده است)

تابع collided_bottom

البته، می‌دانید که یکی از این چهرا تابع بایستی اندکی متفاوت‌تر باشد و این همان تابع است:

```
def collided_bottom(y, co1, co2):
❶ if within_x(co1, co2):
❷ y_calc = co1.y2 + y
❸ if y_calc >= co2.y1 and y_calc <= co2.y2:
❹ return True
❺ return False
```

این تابع از پارامتر اضافی y استفاده می‌کند، مقداری که آن را به موقعیت y مختصات اول اضافه می‌کنیم. در ❶، برخورد افقی مختصات‌ها را بررسی می‌کنیم (همان کاری که با collided_top انجام دادیم). سپس، مقدار پارامتر y را به موقعیت y2 مختصات اضافه کرده و نتیجه را در y_calc در ❷ ذخیره می‌کنیم. اگر در ❸ مقدار تازه محاسبه شده بین مقادیر y1 و y2 مختصات دوم باشد، در ❹، True را برمی‌گردانیم زیرا

ته مختصات co1 به بالای مختصات co2 برخورد کرده است. هرچند اگر هیچیک از دستورات if، True نباشند، در ❺، False را برمی‌گردانیم.

به یک پارامتر اضافی y نیز نیاز داریم زیرا Mr. Stick Man می‌توند از روی سکو سقوط کند. برخلاف دیگر توابع collided، بایستی بتوانیم برخورد احتمالی قسمت ته او را نیز بررسی کنیم. اگر از یک سکو گذر کرده و درهوا شناور باشد، بازی چندان واقعی نخواهد بود بنابراین در جین راه رفتن، باید ببینیم آیا با چیزی در سمت چپ یا راست برخورد کرده است یا نه. زمانی که زیر او را بررسی می‌کنیم، از برخورد او با سکو آگاه خواهیم شد. درصورتیکه برخوردی وجود نداشته باشد، بایستی سقوط کند!

ایجاد کلاس SPRITE

کلاس والد برای آیتم‌های بازی را Sprite می‌نامیم. این کلاس دو تابع تولید می‌کند: move برای حرکت دادن اسپریت و ccords برای برگرداندن موقعیت جاری sprie به صفحه. در اینجا کد کلاس Sprite وجود دارد:

```
class Sprite:
❶ def __init__(self, game):
❷ self.game = game
❸ self.endgame = False
❹ self.coordinates = None
❺ def move(self):
❻ pass
❼{ def coords(self):
        ❽ return self.coordinates
```

تابع _init_ کلاس Sprite که در ❶ تعریف شده است، یک پارامتر دارد: game. این پارامتر، شیء game خواهد بود. به آن نیاز داریم بطوری که هراسپریتی که می‌سازیم بتواند به لیست اسپریت‌های دیگر در بازی دسترسی داشته باشد. در ❷پارامتر game را بصورت یک متغیر شیء ذخیره می‌کنیم.

در ❸، متغیر شیء endgame را ذخیره می‌کنیم که از آن برای نشان دادن خاتمه‌ی بازی استفاده می‌شود (هنوز False نشده است). در ❹آخرین متغیر شیء یعنی coordiantes نشان داده شده است که مقدار آن None است.

تابع move که در ❺ تعریف شده است؛ در این کلاس والد کاری انجام نمی‌دهد بنابراین در ❻کلیدواژه‌ی Pass را در بدنه‌ی این تابع قرار می‌دهیم. تابع coords در ❼ همان متغیر شیء coordinates در ❽ را برمی‌گرداند.

بنابراین کلاس Sprite ما دارای تابع move ای است که کاری انجام نمی‌دهد و یک تابع coord دارد که هیچ مختصاتی را برنمی‌گرداند. به نظر به هیچ دردی نمی‌خورد، اینطور نیست؟ هرچند می‌دانیم که هر کلاسی که Sprite را بعنوان کلاس والد خود درنظر بگیرد، هواره دارای توابع move و coords خواهد بود. بنابراین در حلقه‌ی اصلی بازی، زمانی که در لیستی از اسپریت‌ها حلقه می‌زنیم، می‌توانیم تابع move را صدا بزنیم و هیچ خطایی هم تولید نشود. چرا نه؟ زیرا هر اسپریت آن تابع را دارد.

نکته:

در برنامه‌نویسی معمولاً از کلاس‌هایی استفاده می‌شود که توابع آن‌ها کار زیادی انجام نمی‌دهد. بدین ترتیب، نوعی توافق یا قرارداد هستند که تضمین می‌نمایند تمامی فرزندان یک کلاس، یک نوع کارکرد دارند حتی در مواردی که توابع در کلاس‌های فرزند، کاری انجام نمی‌دهند.

اضافه کردن سکوها

حا سکوها را اضافه خواهیم کرد. کلاس خود را برای شیءهای سکو، PlatformSprite نامیده و یک کلاس فرزند از Sprite خواهد بود. تابع_init_ برای این کلاس، دارای پارامتر game (همانند کلاس والد Srpite)، یک تصویر، موقعیت‌های x و y و همچنین طول و عرض تصویر. کد کلاس PlatformSprite به قرار زیر است:

```
❶ class PlatformSprite(Sprite):
❷ def __init__(self, game, photo_image, x, y, width, height):
❸ Sprite.__init__(self, game)
❹ self.photo_image = photo_image
❺ self.image = game.canvas.create_image(x, y, ₩
image=self.photo_image, anchor='nw')
❻ self.coordinates = Coords(x, y, x + width, y + height)
```

زمانی که در ❶، کلاس PlatformSprite را تعریف می‌کنیم، یک پارامتر است: نام کلاس والد(Sprite). تابع_init_ در ❷ دارای هفت پارامتر می‌باشد: self, game, photo_image, x, y, width وheight.

در ❸، تابع_init_ کلاس والدSprite را با استفاده از self وgame بعنوان مقادیر پارامتر صدا می‌زنیم زیرا تابع_init_ کلاسSprite به جای کلیدواژه‌یself، فقط یک پارامتر را اختیار می‌کند:game.

در این نقطه، اگر بخواهیم یک شیءPlatformSprite بسازیم، از تمامی متغیرهای شیء کلاس والد(game, endgame, and coordinates) برخوردار خواهد بود زیرا تابع_init_ را درSprite صدا زده‌ایم.

در ❹، پارامترphoto_image را بصورت یک متغیر شیء ذخیره می‌کنی و

در ❺،از متغیرcanvas شیءgame و با استفاده از create_image برای ترسیم تصویر روی صفحه استفاده می‌کنیم.

درنهایت، یک شیءCoords با پارامترهایxوy بعنوان دو آرگومان اول. سپس در ❻پارامترهایwidh وheight را بعنوان دو آرگومان دوم به این پارامترها اضافه می‌کنیم.

با فرض این که در کلاس والدSprite متغیرcoordinates برابر باNone قرار داده شده است، در کلاس فرزندPlatformSprite آن را به یک شیء حقیقیCoords تغییر داده‌ایم که حاوی مکان واقعی تصویر سکو روی صفحه است.

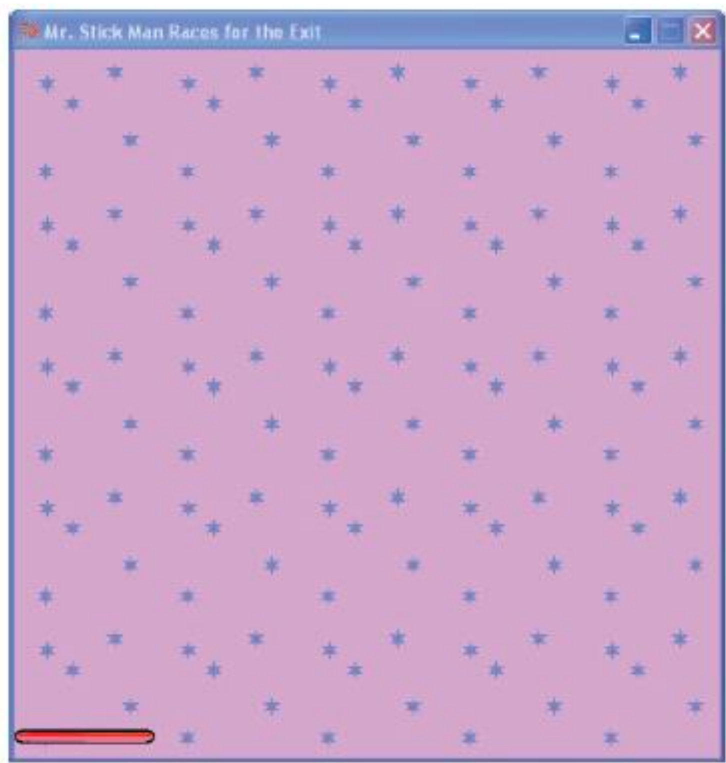
اضافه کردن شیءPlatform

سکو را به بازی اضافه کرده تا ببینیم چه ظاهری پیدا می‌کند. دو خط آخر فایل بازی(*stickmangame.py*) را بصورت زیر تغییر می‌دهیم:

```
❶ g = Game()
❷ platform1 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
0, 480, 100, 10)
❸ g.sprites.append(platform1)
❹ g.mainloop()
```

همانطوریکه مشاهده می‌کنید، خطوط ❶ و ❷ تغییر نکرده ولی در❷، شیء کلاسPlatformSprite را ساخته، درکنار شیءPhotoImage، متغیری را برای بازی(g) در آن قرار می‌دهیم (که از اولین تصاویر سکو استفاده می‌نمایدplatform1.gif). همچنین، موقعیت ترسیم سکو را (0 پیکسل از عرض و ۴۸۰ پیکسل به پایین، نزدیک انتهای بوم) همراه با طول و عرض تصویر(۱۰۰پیکسل عرض و ۱۰ پیکسل طول) در آن قرار می‌دهیم. در❸ این اسپریت را به لیست اسپریت‌های شیءgame اضافه می‌کنیم.

اگر اکنون بازی را اجرا کنید، بایستی سکویی را مشاهده کنید که در طرف پایین-چپ صفحه کشیده شده است:



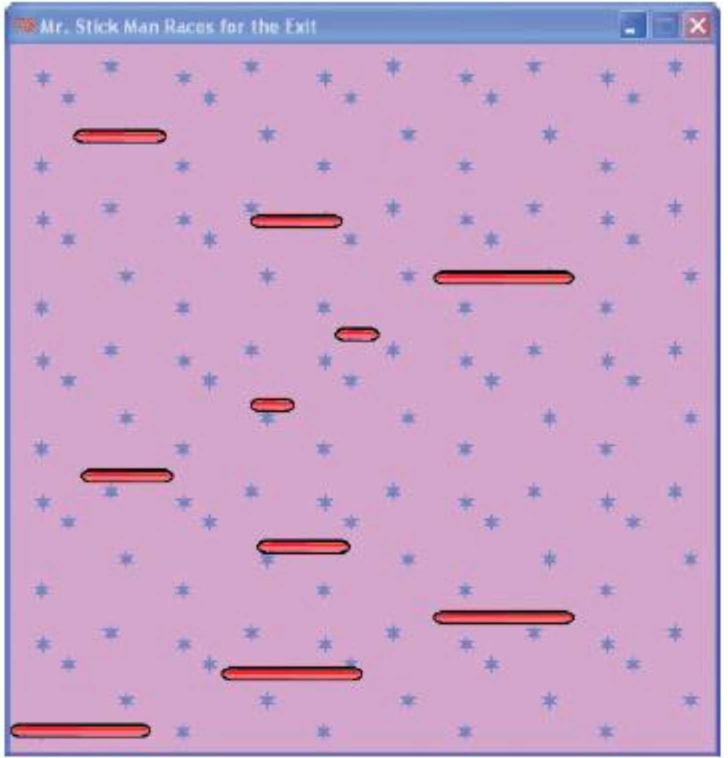
اضافه کردن یک دسته سکو

در اینجا کل سکوها را اضافه می‌کنیم. هر سکو دارای موقعیت x و y متفاوتی است بطوریکه در صفحه پراکنده خواهند بود. از کد زیر استفاده می‌کنیم:

```
g = Game()
platform1 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
0, 480, 100, 10)
platform2 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
150, 440, 100, 10)
platform3 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
300, 400, 100, 10)
platform4 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
        300, 160, 100, 10)
platform5 = PlatformSprite(g, PhotoImage(file="platform2.gif"), ₩
175, 350, 66, 10)
platform6 = PlatformSprite(g, PhotoImage(file="platform2.gif"), ₩
50, 300, 66, 10)
```

```
platform7 = PlatformSprite(g, PhotoImage(file="platform2.gif"), ₩
170, 120, 66, 10)
platform8 = PlatformSprite(g, PhotoImage(file="platform2.gif"), ₩
45, 60, 66, 10)
platform9 = PlatformSprite(g, PhotoImage(file="platform3.gif"), ₩
170, 250, 32, 10)
platform10 = PlatformSprite(g, PhotoImage(file="platform3.gif"), ₩
230, 200, 32, 10)
g.sprites.append(platform1)
g.sprites.append(platform2)
g.sprites.append(platform3)
g.sprites.append(platform4)
g.sprites.append(platform5)
g.sprites.append(platform6)
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
g.mainloop()
```

چندین شیء PlatformSprite را ساخته، آن‌ها را بصورت متغیرهای platform1, platform2, platform3 ... تا platform10 ذخیره می‌کنیم. سپس هر سکو را به متغیر sprites که در کلاس Game ساخته‌ایم، اضافه می‌کنیم. اگر اکنون بازی را اجرا کنید، اینگونه خواهد بود:



مبانی بازی را ایجاد کردیم! حال آماده‌ایم تا کاراکتر اصلی یعنی Mr. Stick Man را بـه آن اضـافه کنیم.

آنچه آموختید

در این فصل، کلاس Game را ساخته و تصویر پس‌زمینه را بصـورت یـک کاغـذ دیـواری روی صفحه می‌کشیم. آموختید که چگونه می‌توانید با خلق توابع within_x و within_y، قرار داشتن موقعیت افقی یا عمودی در مرزهای دو موقعیت افقی و عمودی دیگر را تعیین نمایید. سپس از این توابـع بـرای خلـق توابع جدید برای تعیین برخورد یک شیء مختصات با دیگری استفاده نمودید. از ایـن توابـع در فصـول بعد استفاده خواهیم کرد، یعنی وقتی که Mr. Stick Man را متحرک ساخته و هنگـام جابجـا شـدن او در بوم، بایستی از برخورد او با یک سکو آگاه شویم.

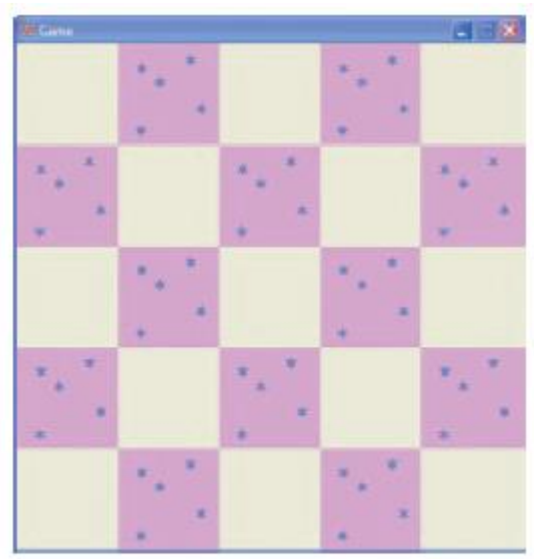
همچنین کلاس والد Sprite و اولین کلاس فرزند آن یعنی PlatformSprite را خلق کرده کـه از آن برای ترسیم سکوها روی بوم استفاه خواهیم کرد.

چیستان‌های برنامه‌نویسی

چیستان‌های کدنویسی زیر راهی برای تجربه کردن تصویر پس‌زمینه‌ی بازی می‌باشند. پاسخ خود را در *http://python-for-kids.com/* بررسی کنید.

۱) تخته‌ی چکرز[1]

کلاس Game را به گونه‌ای تغییر دهید تا تصویر پس‌زمینه همانند یک تخته‌ی چکرز کشیده شود:



۲) تخته چکرز دوتصویری:

بعد از اینکه دریافتید چگونه می‌توانید اثر تخته چکرز را ایجاد کنید، از دو تصویر دلخواه استفاده نمایید. از تصویر کاغذ دیواری دیگری استفاده کرده (با استفاده از برنامه‌ی نگاره‌سازی) و سپس کلاس Game را تغییر داده تا یک تخته چکرز با دو تصویر دلخواه به جای یک تصویر، و پس‌زمینه‌ی خالی را نمایش دهد.

۳) قفسه‌ی کتاب و فانوس(لامپ)

می‌توانید تصاویر کاغذدیواری متفاوتی را ایجاد کرده تا پس‌زمینه‌ی بازی جذاب‌تر به نظر آید. یک کپی از تصویر پس‌زمینه را ایجاد کرده و سپس یک قفسه کتاب ساده روی آن بکشید. یا این که می-توانید میزی با یک فانوس(لامپ) یا یک پنجره بکشید. سپس با تغییر کلاس Game این تصاویر را دو صفحه نقطه‌گذاری نمایید بطوریکه سه یا چهار تصویر کاغذدیواری متفاوت را بارگذاری نماید(و نمایش دهد).

[1] Checkerboard جنگ نادر

فصل ۱۷

خلق Mr. Stick Man

در این فصل، کاراکتر[1] اصلی بازی« مسابقه‌ی Mr. Stick Man برای پیدا کردن درب خروجی» را خواهیم ساخت. این کار نیازمند کدنویسی پیچیده‌ای است که تا این جا انجام داده‌ایم زیرا Mr. Stick Man بایستی به چپ و راست بدود، بپرد و وقتی روی سکو می‌پرد، بایستد و وقتی از لبه‌ی سکو رد می‌شود، پایین بیافتد. از انقیادهای رویداد برای کلیدهای جهت‌نمای چپ و راست استفاده کرده تا این پیکره‌ی خطی بتواند به چپ و راست بدود و زمانی که بازیکن کلید space را فشار می‌دهد، کاری می‌کنیم تا پرش کند.

مقداردهی اولیه‌ی پیکره‌ی خطی

تابع _init_ برای کلاس پیکره‌ی خطی جدید بسیار شبیه به چیزی است که تا اینجا در دیگر کلاس‌های بازی بوده است. با نامگذاری این کلاس:StickFigureSprite کار را شروع می‌کنیم. همانند کلاس‌های قبل، این کلاسیک کلاس والد دارد: Sprite.

```
class StickFigureSprite(Sprite):
def __init__(self, game):
Sprite.__init__(self, game)
```

این کد شبیه به کدی است که برای کلاس PlatformSprite در فصل ۱٦ نوشتیم با این تفاوت کمه از هیچ پارامتر اضافی (به غیر از self و game) استفاده نمی‌کنیم. دلیل آن این است که برخلاف این کلاس PlatformSprite، در این بازی فقط از یک شیء StickFigureSprite استفاده شده است.

[1] Character شخصیت

بارگذاری تصاویر پیکره‌ی خطی

از آنجاییکه چندین شیء سکو در صفحه داریم، که هرکدام می‌تواند از تصویری با سایز متفاوت استفاده کند، تصویر سکو را بصورت پارامتری از تابع_init_ مربوط به PlatformSprite درنظر می‌گیریم (به بیان دیگر «Platform Sprite، وقتی تصویر خودتان را روی صفحه می‌کشید از این تصویر استفاده کنید»). ولی چون فقط یک پیکره‌ی خطی روی صفحه وجود دارد، بارگذاری تصویر بیرون از اسپرایت و سپس بکارگرفتن آن بعنوان یک پارامتر،کار مطقی نیست. کلاس StickFigureSprite درخواهد یافت که چگونه تصاویرش را بارگذاری کند.

چند خط بعدی تابع_init_ همین کار را انجام می‌دهد: هریک از سه تصویر سمت چپ (از آن‌ها برای متحرک نمودن پیکره‌ی خطی برای دویدن به چپ استفاده می‌کنیم) و سه تصویر سمت راست (از آن‌ها برای متحرک نمودن پیکره‌ی خطی برای دویدن به راست استفاده می‌کنیم) را بارگذاری می‌نمایند. الان وقت آن رسیده تا تصاویر را بارگذرای کنیم زیرا نمی‌خواهیم هر بار که پیکره‌ی خطی را روی صفحه نمایش می‌دهیم، مجبور باشیم آنها را بارگذاری کنیم (این کار بسیار زمان‌بر بوده و روند اجرای بازی را کند خواهد کرد).

```
class StickFigureSprite(Sprite):
def __init__(self, game):
Sprite.__init__(self, game)
❶ self.images_left = [
PhotoImage(file="figure-L1.gif"),
PhotoImage(file="figure-L2.gif"),
PhotoImage(file="figure-L3.gif")
]
❷ self.images_right = [
PhotoImage(file="figure-R1.gif"),
PhotoImage(file="figure-R2.gif"),
PhotoImage(file="figure-R3.gif")
]
❸ self.image = game.canvas.create_image(200, 470, ₩
image=self.images_left[0], anchor='nw')
```

این کد، هریک از سه تصویر چپ را بارگذاری می‌کند که از آن‌ها برای متحرک نمودن پیکره‌ی خطی برای دویدن به چپ، و سه تصویر سمت راست را بارگذاری می‌کنند که از آن‌ها برای متحرک نمودن پیکره‌ی خطی برای دویدن به راست استفاده خواهیم کرد.

در ❶ و ❷، متغیرهای شیءimages_left و images_right را خلق می‌کنیم. هر یک از این متغیرها حاوی لیستی از شیءهایPhotoImage (که در فصل ۱۵ ساختیم) می‌باشد که پیکره‌ی خطی را نشان می-دهد که به چپ یا راست چرخیده است.

باimages_left[0] و با استفاده از تابع create_image بوم در موقعیت(200, 470) که پیکره‌ی خطی را در وسط صفحه‌ی بازی و ته بوم قرار می‌دهد، اولین تصویر را در ❸ می‌سازیم. تابع create_image عددی را برمی‌گرداند که تصویر را روی بوم شناسایی می‌کند. این شناسه را در متغیر شیءimage ذخیره کرده تا بعداً از آن استفاده کنیم.

تعیین متغیرها

بخش بعدی تابع_init_ متغیرهای بیشتری را ترتیب داده تا بعداً بتوانیم در این کد از آنها استفاده کنیم.

```
self.images_right = [
PhotoImage(file="figure-R1.gif"),
PhotoImage(file="figure-R2.gif"),
PhotoImage(file="figure-R3.gif")
]
self.image = game.canvas.create_image(200, 470, ₩
image=self.images_left[0], anchor='nw')
❶ self.x = -2
❷ self.y = 0
❸ self.current_image = 0
❹ self.current_image_add = 1
❺ self.jump_count = 0
❻ self.last_time = time.time()
❼ self.coordinates = Coords()
```

در❶و❷، متغیرهای شیءxوy مقداری را ذخیره می‌کند که وقتی پیکره‌ی خطی در صفحه حرکت می‌کند، آنها را به مختصات افقی(x1وx2) یا عمودی(y1وy2) پیکره‌ی خطی اضافه خواهیم کرد.

همانطوریکه در فصل۱۳ آموختیم، برای اینکه با استفاده از ماژول tkinter چیزی را متحرک نماییم، مقداری را به موقعیتxوy شیء اضافه کرده تا در بوم حرکت نماید. با تعیینx=-2 وy=0 ،

مقدار2 را از موقعیتx کم کرده و چیزی به موقعیت عمودی اضافه نمی‌کنیم تا پیکره‌ی خطی بـه سـمت چپ بدود.

نکته:

بخاطر بیاورید که عددx منفی به معنای حرکت به چپ در بوم و عددx مثبت به معنـای حرکـت به راست است. عددy منفی به معنای حرکت رو به بالا و عددy مثبت به معنای حرکت رو به پایین است.

در❸، برای ذخیره‌ی موقعیت شاخص(اندیس)[۱] تصویر بصورتی که در صفحه نمـایش داده شـده است، متغیر current_imageشیء را می‌سازیم. لیست تصاویر دست چپ ما،images_left حـاوی *figure-L1.gif*, *figure-L2.gif* و *figure-L3.gif* است. این‌ها موقعیت‌های شاخص0,1,2 هستند.

در❹، متغیر current_image_add حاوی عددی است که بـه بـه موقعیـت شـاخص ذخیـره شـده درcurrent_image اضافه کرده تا بتوانیم به موقعیت شاخص بعدی دست پیـدا کنـیم. بعنـوان مثـال، اگـر تصویر در موقعیت0 نمایش داده شده باشد،1 را اضافه کرده تا به تصویر بعـدی در موقعیـت شـاخص1 برسیم و سپس مجدداً1 را اضافه می‌کنیم تا به تصویر نهـایی در لیسـت در موقعیـت شـاخص2 برسـیم. (خواهید دید که چگونه از این متغیر برای پویانمایی در فصل بعد استفاده خواهیم کرد).

در❺ متغیرjump_count یک شمارنده است کـه در حـین پـرش پیکـره‌ی خطـی، از آن اسـتفاده خواهیم کرد. متغیرlast_time آخرین زمانی را ثبت می‌کند که هنگام متحرک‌سازی پیکره‌ی خطی، تصـویر را تغییر داده‌ایم. در❻ با استفاده از تابعtime ماژولtime زمان جاری را ثبت می‌کنیم.

در❼، متغیر شیءcoordinates را برابر بـا شـیء کـلاسCoords قـرار مـی‌دهـیم بـدون تعیـین پارامترهای مقداردهی (x1,y1,x2,y2 همگی برابر با0 هستند). برخلاف سکوها، مختصات پیکره‌ی خطی تغییر کرده و بنابراین، بعداً این مقادیر را تعیین خواهیم نمود.

انقیاد کلیدها

در آخرین بخش تابع_init_ ، توابعbind یک کلید را به چیزی در کد مقید می‌کنند که بایستی با فشرده شدن کلید، اجرا شود.

```
self.jump_count = 0
self.last_time = time.time()
self.coordinates = Coords()
game.canvas.bind_all('<KeyPress-Left>', self.turn_left)
```

[۱] index

```
game.canvas.bind_all('<KeyPress-Right>', self.turn_right)
game.canvas.bind_all('<space>', self.jump)
```

<KeyPress-Left> را به تابع turn_left، <KeyPress-Right>را به تابع turn_right و <space> را به تابع jump مقید می‌نماییم. حال برای حرکت دادن پیکره‌ی خطی، بایستی این توابع را بسازیم.

به چپ یا راست حرکت دادن پیکره‌ی خطی

توابع turn_left و turn_right تضمین می‌نمایند که پیکره‌ی خطی نمی‌پرد و سپس برای به چپ یا راست بردن آن، به متغیر شیءx مقدار می‌دهیم. (اگر کاراکتر ما می‌پرد، بازی به ما اجازه نمی‌دهد تا جهت او را در بین زمین و هوا تغییر دهیم)

```
game.canvas.bind_all('<KeyPress-Left>', self.turn_left)
game.canvas.bind_all('<KeyPress-Right>', self.turn_right)
game.canvas.bind_all('<space>', self.jump)
❶ def turn_left(self, evt):
❷ if self.y == 0:
❸ self.x = -2
❹ def turn_right(self, evt):
❺ if self.y == 0:
❻ self.x = 2
```

زمانی که بازیکن کلید جهت‌نمای چپ را فشار می‌دهد، پایتون تابع turn_left را صدا می‌زند و شیء‌ی با اطلاعاتی درباره‌ی کاری که بازیکن انجام داده است، بصورت یک پارامتر درنظر می‌گیرد. این شیء یک شیء رویداد نامیده شده و نام این پارامتر را evt می‌نامیم.

نکته:

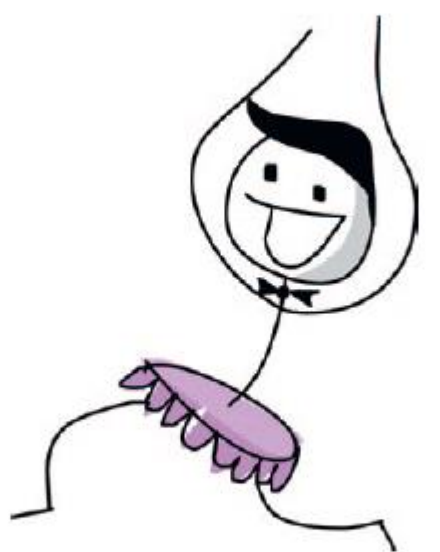
شیء رویداد برای ما چندان اهمیتی ندارد ولی بایستی ان را بعنوان پارامتر توابع خود درنظر گرفته(در ❶و❹) چون در غیراینصورت با پیام خطا مواجه خواهیم شد زیرا پایتون انتظار دارد که این شیء وجود داشته باشد. شیء رویداد حاوی چیزهایی مانند موقعیت‌هایx وy موشواره (رویداد موشواره)، کد شناسایی یک کلید خاص (رویداد صفحه‌کلید) و اطلاعات دیگر می‌باشد. برای این بازی هیچیک از این اطلاعات مفید نیست بنابراین با خیالت راحت می‌توانیم از آن چشم‌پوشی کنیم.

برای بررسی پرش پیکره‌ی خطی، مقدار متغیر شیءy را در❷ و ❺ وارسی می‌کنیم. اگر مقدار0 نباشد، پیکره‌ی خطی در حال پریدن است. در این مثال، اگر مقدارy=0 باشد،x=-2 قرار داده تا به چپ برود ❸یا x=2 قرار داده تا به راست بدود❻، زیرا اگر مقدار برابر با1 یا1- باشد پیکره‌ی خطی با سرعت کافی در صفحه حرکت نخواهد کرد (بعد از متحرک کردن پیکره‌ی خطی، این مقدار را تغییر داده تا تفاوت‌هایی که ایجاد می‌شود را مشاهده نمایید)

کاری کنیم تا پیکره‌ی خطی پرش کند

تابع jump بسیار شبیه به توابع turn_left و turn_right است.

```
def turn_right(self, evt):
if self.y == 0:
self.x = 2
def jump(self, evt):
❶ if self.y == 0:
❷ self.y = -4
❸ self.jump_count = 0
```



این تابع پارامترevt (شیء رویداد) را اختیار کرده که می‌توانیم از آن صرفنظر کنیم زیرا به اطلاعات زیادی درباره‌ی رویداد نیاز نداریم. با راخوانی این تابع، بدلیل فشرده شدن کلید space از آن آگاه خواهیم شد.

چون فقط بدنبال پریدن پیکره‌ی خطی هستیم، در❶، برابر بودن آن با0 را بررسی می‌کنیم. اگر پرش نکند، در❷، y=-4 قرار داده (برای اینکه بصورت عمودی به بالای صفحه حرکت کند) و در❸،jump_count=0 قرار می‌دهیم. برای تضمین اینکه پیکره‌ی خطی دائماً پرش نکند، ازjump_count استفاده خواهیم کرد. درعوض، اجازه می‌دهیم برای مدت زمان معینی پرش کرده و سپس مجدداً پایین بیاید زیرا هر جا که باشد نیروی جاذبه آن را به پایین می‌کشد. در فصل بعد این کد را اضافه خواهیم کرد.

کاری که تا اینجا انجام داده‌ایم

در اینجا تعاریف کلاس‌ها و توابعی که در بازی داریم را بازنگری می‌کنیم. دستورات Import بایستی در بالای برنامه بوده و بعد از آن، کلاس‌های Game و Coords قرار گیرند. از کلاس Game برای خلق شیء‌ای استفاده شده است که کنترلر اصلی بازی خواهد بود و از شیء‌های کلاس Coords برای حفظ موقعیت چیزها در بازی استفاده می‌شود (از جمله سکوها و Mr. Stick Man):

```
from tkinter import *
import random
import time
class Game:
...
class Coords:
...
```

سپس بایستی توابع within را قرار دهیم(که نشان می‌دهد مختصات یک اسپرایت در ناحیه‌ی اسپرایت دیگری قرار دارد). کلاس والد Sprite (که کلاس والد تمامی اسپرایت‌ها در بازی می‌باشد (کلاس والد تمایم اسپرایت‌ها در بازی)، کلاس PlatformSprite و ابتدای کلاس StickFigureSprite. از PlatformSprite برای خلق شیء‌های سکو استفاده شده است که پیکره‌ی خطی ما روی آن‌ها می‌پرد، و یک شیء از کلاس StickFigureSprite را برای نمایش کاراکتر اصلی در بازی خلق کرده‌ایم:

```
def within_x(co1, co2):
...
def within_y(co1, co2):
...
class Sprite:
...
class PlatformSprite(Sprite):
...
class StickFigureSprite(Sprite):
...
```

درنهایت، در پایان برنامه بایستی کدی را دراختیار داشته باشید که تمامی شیء‌های بازی شما را خلق نماید: خود شیء game و سکوها. آخرین خط جایی است که تابع mainloop را صدا می‌زنیم.

```
g = Game()
platform1 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
0, 480, 100, 10)
...
g.sprites.append(platform1)
```

```
...
g.mainloop()
```

اگر کد شما اندکی متفاوت‌تر است، یا اینکه در اجرای آن دچار مشکل هستید، می‌توانید به انتهای فصل۱۸ رجوع کرده و کد کامل[۱]بازی نهایی را مشاهده کنید.

آنچه آموختید

در این فصل، کار روی کلاس را برای پیکره‌ی خطی آغاز کردیم. در این جا، اگر یک شیء از این کلاس را خلق کرده‌ایم، کاری به جز بارگذاری تصاویر موردنیاز خود برای به حرکت درآوردن پیکره‌ی خطی و تعیین چند متغیر شیء برای مصارف آتی در کد، انجام نخواهد داد. این کلاس حاوی دو تابع است که براساس رویدادهای صفحه‌کلید، مقادیر این متغیرهای شیء را تغییر می‌دهد (زمانی که بازیکن کلیدهای جهت‌نمای راست یا چپ یاspacebar را فشار می‌دهد).

در فصل بعد، بازی را تکمیل خواهیم نمود. توابعی را برای کلاسStickFigureSprite خواهیم نوشت تا پیکره‌ی خطی را نمایش داده، به حرکت درآورده و در صفحه جابجا کنند. همچنین درب خروج را اضافه خواهیم کرد، همان چیزی کهMr. Stick Man تلاش می‌کند به آن برسد.

---

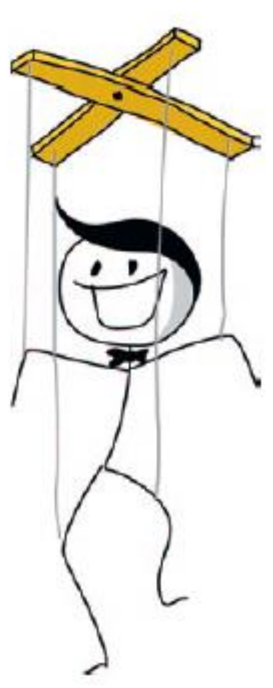
[۱] listing

فصل۱۸

تکمیل بازیMr. Stick Man

در سه فصل قبل، بازی خودمان را توسعه دادیم: مسابقه‌ی Mr. Stick Man برای رسیدن به درب خروج. نگاره‌ها را خلق کرده و کدی را برای اضافه کردن تصویر پس‌زمینه، سکوها و پیکره‌ی خطی ساختیم. در این فصل، تکه‌های گم‌شده را پر کرده تا پیکره‌ی خطی را به حرکت درآورده و درب را اضافه کنیم.

در پایان این فصل، کد کامل بازی تکمیل‌شده ارائه شده است. اگر هنگام نوشتن بخشی ازکد، سردرگم شدید، کد خود را با کد کامل نهایی مقایسه کرده تا به اشتباهات خود پی ببرید.



به حرکت درآوردن پیکره‌ی خطی

تا این جا، کلاس اصلی را برای پیکره‌ی خطی خلق کرده، تصاویری را که می‌خواهیم از آنها استفاده کنیم، بارگذاری نموده و کلیدها را به برخی توابع مقید ساختیم. ولی اگر در این نقطه، برنامه را اجرا کنید، کدنویسی ما هیچ کار جالب‌توجهی انجام نخواهد داد.

حال توابع باقیمانده را به کلاس StickFigureSpriteی که در فصل۱۷ ساختیم، اضافه خواهیم کرد:animate, move و coords. تابع animate تصاویر متفاوتی از پیکره‌ی خطی را ترسیم خواهد کرد، move جایی را تعیین می‌کند که کاراکتر بایستی به آنجا برود و coords موقعیت جاری پیکره‌ی خطی را

برمی‌گرداند. (برخلاف اسپرایت‌های سکو، بایستی موقعیت پیکره‌ی خطی را در حین جابجا شدن در صفحه، مجدداً محاسبه کنیم.)

خلق تابع پویانمایی

ابتدا، تابع animate را اضافه خواهیم کرد که برای وارسی حرکت و تغییر تصویر براساس آن، به آن نیاز داریم.

وارسی جابجایی

در پویانمایی قصد نداریم به سرعت تصویر پیکره‌ی خطی را تغییر داده یا این که کاری کنیم تا حرکت آن غیرواقعی به نظر برسد. تصور کنید یک تصویر متحرک چند مرحله‌ای را گوشه‌ی دفتریادداشت خود کشیده‌اید– اگر به سرعت ورق بزنید، اثر کامل آنچه ترسیم کرده‌اید را مشاهده نخواهید کرد.

نیمه‌ی اول تابع animate دویدن پیکره‌ی خطی به چپ یا راست را وارسی کرده و سپس از متغیر last_time استفاده کرده تا تصمیم بگیرد که چه زمان تصویر جاری را تغییر دهد. این متغیر به ما کمک خواهد کرد تا سرعت پویانمایی را کنترل کنیم. این تابع بعد از تابع jump که در فصل ۱۷ به کلاس StickFigureSprite اضافه کردیم، قرار می‌گیرد:

```
def jump(self, evt):
if self.y == 0:
self.y = -4
self.jump_count = 0
def animate(self):
❶ if self.x != 0 and self.y == 0:
❷ if time.time() - self.last_time > 0.1:
❸ self.last_time = time.time()
❹ self.current_image += self.current_image_add
❺ if self.current_image >= 2:
❻ self.current_image_add = -1
❼ if self.current_image <= 0:
❽ self.current_image_add = 1
```

در دستور if در❶، برای اینکه ببینیم پیکره‌ی خطی حرکت کرده است (به چپ یا راست) نامساوی $x \neq 0$ و برای اینکه ببینیم پیکره‌ی خطی پرش نکرده است، تساوی $y=0$ را وارسی می‌کنیم. در

صورت true بودن دستور if، بایستی پیکره‌ی خطی را حرکت دهیم؛ درغیراینصورت، همچنان بیحرکت مانده و بنابراین نیازی به ادامه‌ی ترسیم نیست. اگر پیکره‌ی خطی حرکت نکند، از تابع خارج شده و از بقیه‌ی کد در کد نهایی، صرفنظر خواهد شد.

در ❷، مقدار زمان از آخرین باری که تابع animate صدا زده شده است را، با استفاده از time.time() و با تفریق مقدار متغیر last_time از زمان جاری، محاسبه می‌کنیم. از این محاسبه برای تصمیم‌گیری درخصوص ترسیم تصویر بعدی در این دنباله استفاده می‌شود و درصورتیکه نتیجه بزرگتر از یک دهم ثانیه(0.1s) باشد، بلوک کد را در ❸ ادامه خواهیم داد. متغیر last_time را برابر با زمان جاری قرار می‌دهیم که اساساً برای اعمال تغییر بعدی در تصویر، زمان توقف را به زمان شروع بازنشانی می‌کند.

در ❹، مقدار متغیر شیء current_image_add را به متغیر current_image اضافه می‌کنیم که موقعیت شاخص(اندیس) تصویری که درحال حاضر نمایش داده شده است را ذخیره می‌کند. به خاطر داشته باشید که متغیر current_image_add در تابع _init_ پیکره‌ی خطی را در فصل ۱۷ ساختیم بنابراین زمانی که تابع animate اولین بار صدا زده می‌شود، مقدار متغیر برابر با 1 قرار داده می‌شود.

در ❺، بررسی می‌کنیم که آیا مقدار موقعیت شاخص در current_image بزرگتر یا مساوی 2 هست یا نه، دراینصورت، در ❻ مقدار current_image_add را به 1- تغییر می‌دهیم. این فرآیند در ❼ نیز تکرار می‌شود یعنی وقتی به 0 می‌رسیم در ❽ بایستی شمارش را مجدداً از سر بگیریم.

نکته:

اگر با درک شیوه‌ی استفاده از تورفتگی در کد مشکل دارید، این جا تذکری داده شده است: هشت جای خالی در ابتدای ❶ و ۲۰ جای خالی در ابتدای ❽.

برای این که بفهمیم تا اینجا چه اتفاقی در تابع افتاده است، تصور کنید یک ردیف از بلوک‌های رنگی در یک خط در یک طبقه دارید. انگشت خود را از یک بلوک به بلوک بعدی اشاره نموده و هر بلوکی که انگشت شما به آن اشاره کند(...,1,2,3,4) یک شماره خواهد داشت(متغیر current_image).شماره‌ی بلوکی که انگشت شما به طرف آن حرکت می‌کند(هر بار به یک بلوک اشاره می‌کند) عددی است که متغیر current_image_add ذخیره شده است. وقتی انگشت شما به یک خط بالاتر از ردیف بلوک‌ها اشاره می‌کند، هر بار یک واحد(1) را اضافه کرده و زمانی که به آخرین خط اشاره کرده و به پایین برمی‌گردد، یک واحد(1-) کم می‌کنید(یعنی 1- اضافه می‌کنید).

کدی که به تابع animate اضافه کرده‌ایم این فرآیند را انجام می‌دهد ولی به جای بلوک‌های رنگی، برای هر جهت، سه تصویر از پیکره‌ی خطی داریم که در یک لیست ذخیره شده است. موقعیت

شاخص این تصاویر عبارتند از0,1,2. با به حرکت درآوردن پیکره‌ی خطی، وقتی به آخرین تصویر می-رسیم، شمارش معکوس را شروع می‌کنیم و زمانی که به اولین تصویر می‌رسیم، بایستی مجدداً شمارش رو به بالا را آغاز کنیم. درنتیجه، اثر یک پیکره‌ی در حال دویدن را ایجاد میکنیم.

در ادامه نشان می‌دهیم که چگونه با استفاده از موقعیت‌های شاخص که در تابع animate محاسبه کردیم، در لیست تصاویر جستجو می‌کنیم.

Position 0	Position 1	Position 2	Position 1	Position 0	Position 1
Counting up	Counting up	Counting up	Counting down	Counting down	Counting up

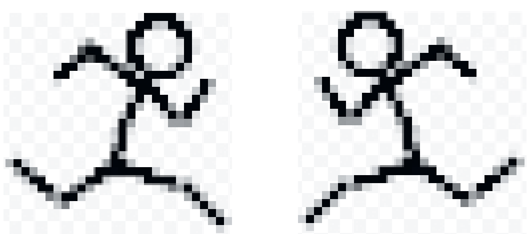
تغییر تصویر

در نیمه‌ی دوم تابع animate، با استفاده از موقعیت شاخص محاسبه شده، تصویری که به تازگی نمایش داده شده است را تغییر می‌دهیم.

```
def animate(self):
if self.x != 0 and self.y == 0:
if time.time() - self.last_time > 0.1:
self.last_time= time.time()
self.current_image += self.current_image_add
if self.current_image >= 2:
self.current_image_add = -1
if self.current_image <= 0:
self.current_image_add = 1
❶ if self.x < 0:
❷ if self.y != 0:
❸ self.game.canvas.itemconfig(self.image, ₩
image=self.images_left[2])
❹ else:
        self.game.canvas.itemconfig(self.image, ₩
image=self.images_left[self.current_image])
❻ elif self.x > 0:
❼ if self.y != 0:
❽ self.game.canvas.itemconfig(self.image, ₩
```

```
image=self.images_right[2])
❾ else:
❿ self.game.canvas.itemconfig(self.image, ₩
image=self.images_right[self.current_image])
```

در❶، اگر x<0 باشد پیکره‌ی خطی به چپ حرکت می‌کند بنابراین پایتون به بلوک کدی حرکت می‌کند که از❷ تا❺ نشان داده شده است و نابرابری y≠0 را وارسی می‌کند (به این معنی که پیکره‌ی خطی درحال پریدن است). درصورتیکه y≠0 باشد (پیکره‌ی خطی به بالا یا پایین می‌رود- یا به معنای دیگر، پریدن) آنگاه از تابع itemconfig بوم برای تغییر تصویر نمایش داده شده به تصویر نهایی در فهرست تصاویر رو به چپ در❸ استفاده می‌کنیم(images_left[2]). ازآنجایی که پیکره‌ی خطی در حال پریدن است، از تصویری استفاده خواهیم کرد که او را در حال گام برداشتن کامل نشان می‌دهد تا پویانمایی واقعی‌تر به نظر آید:



درصورتیکه پیکره‌ی خطی در حال پریدن نباشد(یعنی y=0) آنگاه دستور else که در❹ آغاز می-شود از itenconfig برای تغییر تصویر نمایش داده شده، به موقعیت شاخص در متغیر current_image استفاده می‌کند همانطوریکه در کد در❺ نشان داده شده است.

در❻، می‌بینیم که پیکره‌ی خطی در حال دویدن به سمت راست است(x>0) و پایتون به بلوک نشان داده شده در❼ تا❿ حرکت می‌کند. این کد بسیار شبیه به بلوک اول است و بازهم پریدن پیکره‌ی خطی را وارسی کرده و دراینصورت، تصویر صحیح را ترسیم خواهد نمود مگراین که از لیست images_right استفاده کند.

دستیابی به موقعیت پیکره‌ی خطی

از آنجایی که بایستی موقعیت پیکره‌ی خطی را در صفحه مشخص کنیم (چون در حال جابجا شده در صفحه است)، تابع coords با دیگر توابع کلاس Sprite متفاوت می‌باشد. از تابع coords بوم برای تعیین مکان پیکره‌ی خطی استفاده کرده و سپس از این مقادیر برای تعیین مقادیر x1,y1,x2,y2 متغیر Coordinates استفاده خواهیم نمود که در تابع_init_ در ابتدای فصل ۱۷ خلق کردیم. در اینجا کد نشان داده شده است و می‌توان آن را بعد از تابع animate اضافه نمود:

```
if self.x < 0:
if self.y != 0:
self.game.canvas.itemconfig(self.image, ₩
image=self.images_left[2])
else:
self.game.canvas.itemconfig(self.image, ₩
image=self.images_left[self.current_image])
elif self.x > 0:
if self.y != 0:
self.game.canvas.itemconfig(self.image, ₩
image=self.images_right[2])
else:
self.game.canvas.itemconfig(self.image, ₩
image=self.images_right[self.current_image])
def coords(self):
❶ xy = self.game.canvas.coords(self.image)
❷ self.coordinates.x1 = xy[0]
❸ self.coordinates.y1 = xy[1]
❹ self.coordinates.x2 = xy[0] + 27
❺ self.coordinates.y2 = xy[1] + 30
return self.coordinates
```

زمانی که کلاس Game را در فصل ۱٦ خلق کردیم، یکی از متغیرهای شیء، canvas بود. در ❶ ، از تابع coord این متغیر canvas با self.game.canvas.coords برای برگرداندن موقعیت‌های x و y تصویر جاری استفاده می‌کنیم. این تابع از عدد ذخیره شده در متغیر شیء image استفاده می‌کند که همان شناسه‌ی تصویری می‌باشد که روی بوم کشیده شده است.

لیست حاصل را در متغیر xy ذخیره می‌کنیم که حاوی دو مقدار است: موقعیت x بالا–چپ که بصورت متغیر x مربوط به coordinates در ❷ ذخیره شده است و موقعیت بالا–چپ که بصورت متغیر y1 مربوط به coordinates در ❸ ذخیره گردیده است.

چون تمامی تصاویر پیکره‌ی خطی که خلق کرده‌ایم، ۲۷×۳۰(طول در عرض) پیکسل هستند می‌توانیم با افزودن عرض در ❹ و ارتفاع در ❺ به اعداد x و y ، متغیرهای x و y را تعیین کنیم.

در نهایت، در آخرین خط تابع، متغیر شیء coordinates را برمی‌گردانیم.

به حرکت درآوردن پیکره‌ی خطی

Move آخرین تابع کلاس StickFigureSprite، کاراکتر بازی ما را در صفحه به حرکت در خواهد آورد. همچنین بایستی بتواند زمانی که کاراکتر به چیزی برخورد می‌کند، به ما اعلام نماید.

شروع تابع MOVE

در این جا کد اولین بخش تابع move نشان داده شده است که بعد از coords قرار می‌گیرد:

```
def coords(self):
xy = self.game.canvas.coords(self.image)
self.coordinates.x1 = xy[0]
self.coordinates.y1 = xy[1]
self.coordinates.x2 = xy[0] + 27
self.coordinates.y2 = xy[1] + 30
return self.coordinates
def move(self):
❶ self.animate()
❷ if self.y < 0:
❸ self.jump_count += 1
❹ if self.jump_count > 20:
❺ self.y = 4
❻ if self.y > 0:
❼ self.jump_count -= 1
```

در ❶ این بخش از تابع، تابع animate ی را صدا می‌زند که در ابتدای این فصل ساختیم و درصورت لزوم، تصویری که نمایش داده شده است را تغییر خواهد داد. در ❷، شرط y<0 را بررسی می‌کنیم. دراینصورت، می‌دانیم که پیکره‌ی خطی پرش کرده است زیرا مقدرا منفی آن را به بالای صفحه حرکت خواهد داد. (یادآور می‌شویم که 0 در بالای بوم است و پایین بوم، موقعیت پیکسل ۵۰۰ می‌باشد).

در ❸، 1 را به jump_count اضافه کرده و در ❹، می‌گوییم که اگر مقدار jump_count به ۲۰ برسد بایستی y را به ٤ تغییر داده تا پیکره‌ی خطی مجدداً شروع به پایین آمدن بکند(❺).

در ❻، اگر مقدار y>0 باشد (به این معنی که کاراکتر بایستی سقوط کند)آنگاه 1 را از jump_count کم می‌کنیم زیرا یکبار تا ۲۰ شمرده‌ایم، و بایستی دوباره شمارش معکوس کنیم. (درحین شمارش تا۲۰، دست خود را به آرامی بالا بیاورید و هنگام شمارش معکوس از۲۰، دستان خود را به آرامی پایین بیاورید، بدین ترتیب حس خواهید کرد که محاسبه‌ی بالا و پایین پردیدن پیکره‌ی خطی به چه صورت انجام می‌شود)

در چند خط بعدی تابع move، تابع coords را صدا می‌زنیم که محل کاراکتر در صفحه را نشان داده و نتیجه‌ی آن را در متغیر co ذخیره می‌کند. سپس متغیرهای left, right, top, bottom و falling را خلق می‌کنیم.از هر یک از این متغیرها در بقیه‌ی این تابع استفاده خواهد شد.

```
if self.y > 0:
self.jump_count -= 1
co = self.coords()
left = True
right = True
top = True
bottom = True
falling = True
```

توجه داشته باشید که هر متغیر یک مقدار بولین True است. از این مقادیر بعنوان شاخص‌هایی برای وارسی برخورد کاراکتر با چیزی در صفحه یا سقوط کردن کاراکتر، استفاده خواهیم کرد.

آیا پیکره‌ی خطی با چیزی در پایین یا بالای بوم برخورد کرده است؟

بخش بعدی تابع move، برخورد کاراکتر بازی با بالا یا پایین بوم را وارسی می‌کند:

```
bottom = True
falling = True
❶ if self.y > 0 and co.y2 >= self.game.canvas_height:
```

```
❷ self.y = 0
❸ bottom = False
❹ elif self.y < 0 and co.y1 <= 0:
❺ self.y = 0
❻ top = False
```

اگر کاراکتر به پایین صفحه سقوط کند، y بزرگتر از 0 خواهد بود بنابراین بایستی مطمئن شویم که هنوز به ته بوم برخورد نکرده است (یا در پایین صفحه محو خواهد شد). برای این کار در❶، شرط موقعیت y2 (ته پیکره‌ی خطی) بزرگتر یا مساوی متغیر canvas_height شیء game را بررسی می‌کنیم. دراینصورت، در❷، مقدار y را برابر با 0 قرار داده تا مانع سقوط پیکره‌ی خطی شویم و سپس در❸، متغیر bottom را False قرار می‌دهیم که به بقیه کد می‌گوید، دیگر نیازی به وارسی برخورد پیکره‌ی خطی با ته بوم نداریم.

فرآیند تعیین برخورد پیکره‌ی خطی با بالای صفحه بسیار شبیه به شیوه‌ی تعیین برخورد آن با ته صفحه است. برای این کار در❹، ابتدا شرط پرش پیکره‌ی خطی(y<0) و سپس شرط موقعیت y1 کوچکتر یا مساوی 0 را بررسی خواهیم کرد به این معنی که پیکره به بالای بوم برخورد کرده است. درصورت برقرار بودن هر دو شرط، در❺، y را برابر با 0 قرار داده تا جلوی حرکت را بگیریم. همچنین در❻، متغیر top را برابر با True قرار داده تا به بقیه‌ی کد بگوییم که دیگر نیازی به وارسی شرط برخورد پیکره‌ی خطی با بالای صفحه نداریم.

آیا پیکره‌ی خطی به کناره‌های بوم برخورد کرده است؟

در زیر، تقریباً همان فرآیند کد قبل برای تعیین برخورد پیکره‌ی خطی با کناره‌های چپ یا راست بوم دنبال می‌شود:

```
elif self.y < 0 and co.y1 <= 0:
self.y = 0
top = False
❶ if self.x > 0 and co.x2 >= self.game.canvas_width:
❷ self.x = 0
❸ right = False
❹ elif self.x < 0 and co.x1 <= 0:
❺ self.x = 0
❻ left = False
```

در❶، کد براساس این واقعیت است که می‌دانیم اگر x>0 باشد، پیکره‌ی خطی به طرف راست می‌دود. همچنین اگر موقعیت x2(co.x2)بزرگتر یا مساوی عرض بوم ذخیره شده درcanvas.width باشد،

پیکره‌ی خطی به سمت راست صفحه برخورد کرده است. درصورت برقرار بودن هر دو شـرط، x=0 در نظر گرفته شده (برای این که پیکره‌ی خطی دیگر ندود) و در ❸ متغیر right را برابر با False قرار خواهیم داد.

برخورد با دیگر اسپرایت‌ها

بعد از این که فهمیدیم پیکره‌ی خطی به کناره‌های صفحه برخورد کرده است، بایستی ببینـیم بـه جای دیگری نیز در صفحه برخورد کرده است یا نـه. از کـد زیـر بـرای جسـتجو در لیسـت شـیء‌های اسپرایت ذخیره شده در شیءgame استفاده کرده تا از برخورد پیکره‌ی طی با هریک از آنها آگاه شویم:

```
elif self.x < 0 and co.x1 <= 0:
self.x = 0
left = False
❶ for sprite in self.game.sprites:
❷ if sprite == self:
❸ continue
❹ sprite_co = sprite.coords()
❺ if top and self.y < 0 and collided_top(co, sprite_co):
❻ self.y = -self.y
❼ top = False
```

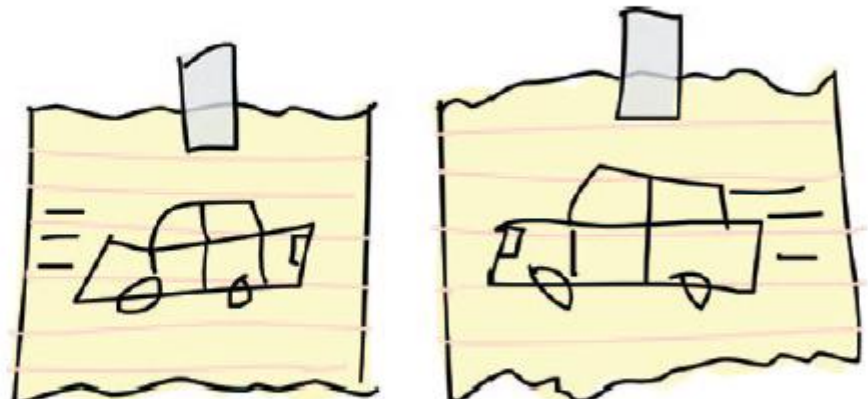
در❶، در لیست اسپرایت‌ها جستجو کرده و به نوبت هریک را به متغیر sprite نسـبت مـی‌دهـیم.

در❷، می‌گوییم اگر sprite=self (شیوه‌ی دیگری برای گفتن «اگر اسپرایت همان من باشد») آنگاه نیازی به تعیین برخورد پیکره‌ی خطی نیست زیرا فقط به خودش خورده است. اگر متغیر sprite= self باشـد از continue برای پریسدن به اسپرایت بعدی در لیست استفاده می‌کنیم.

سپس، با فراخوانی تابع coords در ❹ مختصات اسـپراتی جدیـد را بدسـت آورده و نتـایج را در متغیر sprite_co ذخیره می‌کنیم.

سپس کد ❺ را بدنبال موارد زیر وارسی می‌نماییم:

- پیکره‌ی خطی به بالای بوم برخورد نکرده است (متغیر top هنوز همtrue است)
- پیکره‌ی خطی پرش می‌کند(y<0)
- بالای پیکره‌ی خطی به اسـپرایت موجـود در لیسـت برخـورد کـرده اسـت (بـا اسـتفاده از تابع collide_top که در فصل١٦ ساختیم).



درصورت برقرار بودن تمامی شرایط فوق، میخواهیم اسپرایت مجدداً به پایین برگـردد بنـابراین در❻، با استفاده از علامت منفی(−) مقدارy را برعکس می‌کنیم. در❼، متغیـرtop=False اسـت زیـرا بـه مجرد برخورد کردن پیکره‌ی خطی با بالای صفحه، نیازی به وارسی برخورد نیست.

برخورد در پایین

بخش بعدی حلقه، به وارسی برخورد ته کاراکتر به یک شیء تخصیص یافته است:

```
if top and self.y < 0 and collided_top(co, sprite_co):
self.y = -self.y
top = False
❶ if bottom and self.y > 0 and collided_bottom(self.y, ₩
co, sprite_co):
❷ self.y = sprite_co.y1 - co.y2
❸ if self.y < 0:
❹ self.y = 0
❺ bottom = False
❻ top = False
```

سه وارسی مشابه در❶ صورت می‌گیرد: آیا هنوزهم متغیرbottom مشخص است، آیا کاراکتر در حال سقوط است(y>0) و آیا ته کاراکتر ما به اسپرایت برخورد کرده است. درصورت برقرار بودن هر سه شرط، در❷، مقدارy ته(y2) پیکره‌ی خطی را از مقدارy بالای اسـپرایت(y1) کـم مـی‌کـنیم. شـاید کمـی عجیب به نظر برسد پس دلیل آن را توضیح خواهیم داد.

تصـور کنیـد کـاراکتر بـازی مـا از روی یـک سـکو سـقوط کـرده اسـت. بـا هـر بـار اجـرای تابعmainloop، چهار پیکسل به پایین صفحه حرکت خواهد کرد و پای پیکره‌ی خطی ۳ پیکسـل بـالای سکوی دیگری قرار می‌گیرد. اگر ته پیکره‌ی خطی(y2) در موقعیت۵۷ و روی سکو(y1) در موقعیـت ٦٠

باشد، تابع collided_botom، true را برمی‌گرداند زیرا کد آن، مقدار y (y=4) را به متغیر y2 پیکره‌ی خطی اضافه می‌کند، که نتیجه ٦١ خواهد بود.

هرچند، نمی‌خواهیم Mr. Stick Man به سرعت متوقف شود انگار که به سکویی برخورد کـرده است یا به ته صفحه رسیده باشد، زیرا مثل این است که پرش خیلی بلندی انجام داده و در میان زمـین و هوا در فاصله‌ی یک اینچی زمین متوقف شده است. حقه‌ی شسته رفته‌ای اسـت ولـی در بـازی مـا کـار درستی نیست. درعوض، اگرم قدار y2 کاراکتر(٥٧)را از مقدار y1 سـکو(٦٠) کـم کـنیم، مقـدار٣ بدسـت خواهد آمد که همان مقداری است که پیکره‌ی خطی بایستی سقوط کرده تا بدرستی روی سکو قرار گیرد (فرود آید).

در ❸، تضمین می‌نماییم که حاصل محاسبه یک عدد منفی نخواهد بود؛ در اینصورت، در ❹، y=0 درنظرگرفته می‌شود. (اگر عدد منفی باشد، پیکره‌ی خطی مجدداً به هـوا خواهیـد پریـد و در ایـن بـازی نمی‌خواهیم چنین اتفاقی بیافتد.)

درنهایت، پرچم‌های top ❻و bottom ❼، را برابر با False قرار می‌دهیم بنابراین دیگر لازم نیست ببینیم پیکره‌ی خطی به بالا یا پایین اسپرایت دیگری خورده است یا نه.

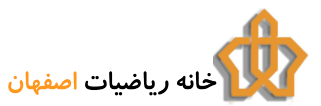
وارسی برخورد به انتهای دیگری را انجام داده تا ببینیم آیا پیکره‌ی خطی به لبه‌ی سـکو خـورده است یا نه. کد این دستور به قرار زیر است:

```
if self.y < 0:
self.y = 0
bottom = False
top = False
if bottom and falling and self.y == 0 ₩
and co.y2 < self.game.canvas_height ₩
and collided_bottom(1, co, sprite_co):
falling = False
```

هر پنج وارسی که در اینجا انجام می‌شود بایستی true باشند تا متغیر falling برابر با False باشد:

- هنوز هم بایستی True بودن پرچم bottom را وارسی کنیم
- بایستی سقوط کردن پیکره‌ی خطی را وارسی کنیم (پرچم falling همچنان True است)
- پیکره‌ی خطی درحال سقوط نیست(y=0)
- ته اسپرایت به پایین صحه برخورد نکرده است (کمتر از طول بوم نیست)
- پیکره‌ی خطی به بالای سکو برخورد کرده است (collided-bottom،True را برمی‌گرداند)

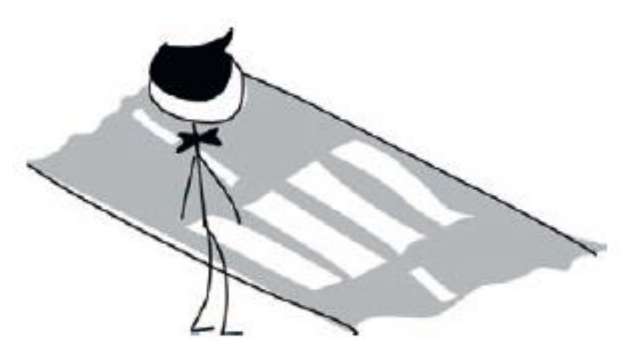
سپس متغیر falling را False قرار می‌دهیم.



وارسی چپ و راست

برخورد پیکره‌ی خطی به اسپرایت در پایین یا بالا را وارسی کردیم. حال بایستی برخورد پیکره-ی خطی به چپ یا راست را وارسی کنیم:

```
if bottom and falling and self.y == 0 ₩
and co.y2 < self.game.canvas_height ₩
and collided_bottom(1, co, sprite_co):
falling = False
❶ if left and self.x < 0 and collided_left(co, sprite_co):
❷ self.x = 0
❸ left = False
❹ if right and self.x > 0 and collided_right(co, sprite_co):
❺ self.x = 0
❻ right = False
```



در❶، همچنان بـدنبال برخوردهـا بـا چـپ(left-True). حرکـت پیکـره‌ی خطـی بـه چـپ(x<0) هسـتیم. همچنین با استفاده از تابع collided_left برخورد پیکـره‌ی خطی با یک اسپرایت را بررسی می‌کنیم. در صورت برقرار بودن این سه شرط، در❷،x را برابر با0 قرار می‌دهیم (برای اینکه پیکـره‌ی خطـی دیگـر حرکـت نکنـد)و در❸، Left راFalse قرار می‌دهیم بطوریکه دیگر بدنبال وارسی برخورد به چپ نباشیم.

کد مشابه کد برخوردها به راست است که در❹ نشان داده شده است. مجدداً در❺ x را برابر با0 قرار می‌دهیم و در❻، right=False تا دیگر بدنبال بررسی برخوردهای دست راست نباشیم.

حال با وارسی‌های برخورد در هر چهار جهت، حلقه‌یfor بایستی بصورت زیر باشد:

```
elif self.x < 0 and co.x1 <= 0:
self.x = 0
left = False
for sprite in self.game.sprites:
if sprite == self:
continue
sprite_co = sprite.coords()
if top and self.y < 0 and collided_top(co, sprite_co):
self.y = -self.y
top = False
```

```
if bottom and self.y > 0 and collided_bottom(self.y, ₩
co, sprite_co):
self.y = sprite_co.y1 - co.y2
if self.y < 0:
self.y = 0
bottom = False
top = False
if bottom and falling and self.y == 0 ₩
and co.y2 < self.game.canvas_height ₩
and collided_bottom(1, co, sprite_co):
        falling = False
if left and self.x < 0 and collided_left(co, sprite_co):
self.x = 0
left = False
if right and self.x > 0 and collided_right(co, sprite_co):
self.x = 0
        right = False
```

بایستی چند خط دیگر به تابع move اضافه کنیم:

```
if right and self.x > 0 and collided_right(co, sprite_co):
self.x = 0
right = False
❶ if falling and bottom and self.y == 0 ₩
and co.y2 < self.game.canvas_height:
❷ self.y = 4
❸ self.game.canvas.move(self.image, self.x, self.y)
```

در ❶، True بـودن هـر دو متغیـر falling و bottom را وارسـی مـی‌کنـیم. دراینصـورت، در هـر اسپرایت سکو در لیست جستجو می‌کنیم بدون این که به پایین صفحه برخورد کنیم.

وارسی نهایی در این خط تعیین می‌کند که آیا ته کاراکتر ما کمتر از طول بوم هست یا نه- یعنی بالای سطح زمین(ته بوم). اگر پیکره‌ی خطی به هیچ چیزی برخورد نکرده باشـد و بـالای سـطح زمـین باشد، همچنان میان زمین و هوا قرار دارد و بنابراین بایستی سقوط را آغاز کند (بـه بیـان دیگـر، از آخـر سکو افتاده است). برای این که از انتهای هر سکویی بیافتد، در❷،Y را برابر با٤ قرار می‌دهیم.

در❸، برطبق مقادیری که برای متغیرهایxو y تعیین می‌کنیم، تصویر را در صـفحه حرکـت مـی- دهیم. این واقعیت که بدنبال یافتن برخوردها در اسپرایت‌ها گشت می‌زنیم، به این معنی است که هـر دو

متغیر را برابر با0 قرار می‌دهیم زیرا پیکره‌ی خطی به چپ و پایین برخورد کرده است. در این مورد، صدا زدن تابع move بوم کاری انجام نخواهد داد.

در این مورد ممکن است Mr. Stick Man از لبه‌ی سکو جدا شده باشد. دراینصورت y=4 گردیده و Mr. Stick Man به پایین سقوط خواهد کرد.

لعنتی، چه تابع طولانی!

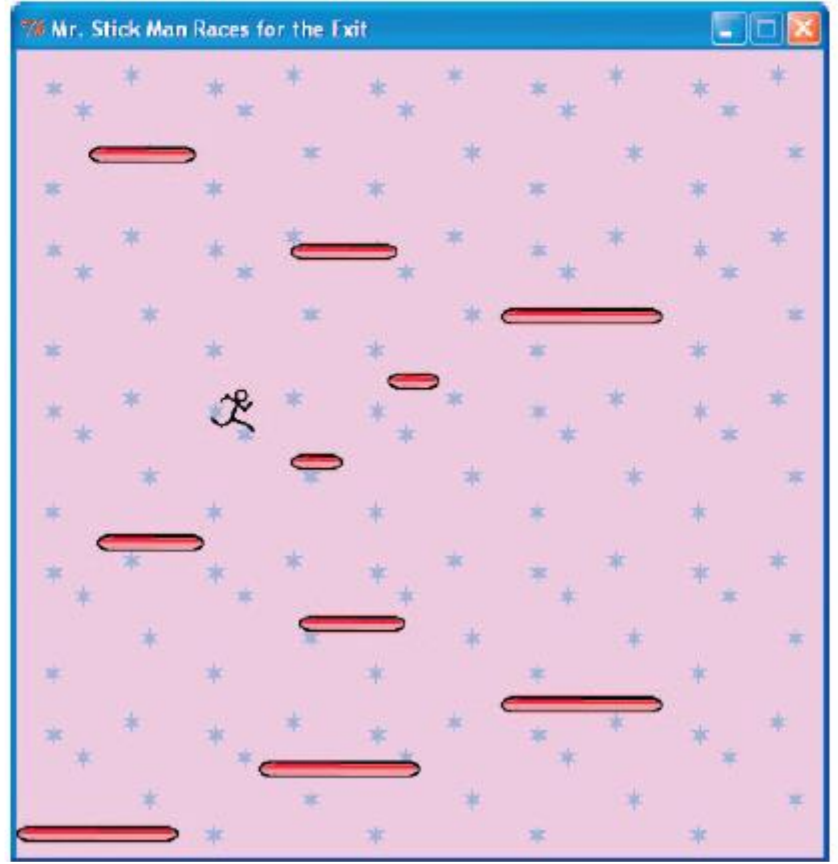
آزمودن اسپرایت پیکره‌ی ما

بعد از خلق کلاس StickFigureSprite، با اضافه کرده دو خط زیر قبل از فراخوان تابع mainloop، آن را می‌آزماییم:

```
❶ sf = StickFigureSprite(g)
❷ g.sprites.append(sf)
g.mainloop()
```

در ❶، شیء StickFigureSprite را ساخته و آن را برابر با متغیر sf قرار می‌دهیم. همانطوریکه با سکوها رفتار کردیم، در ❷ این متغیر جدید را به لیست اسپرایت‌های ذخیره شده در شیء game اضافه می‌کنیم.

حال برنامه را اجرا می‌کنیم. خواهید دید که Mr. Stick Man می‌تواند بدود، از روی سکویی به سکوی دیگر پریده و سقوط کند!



درب!

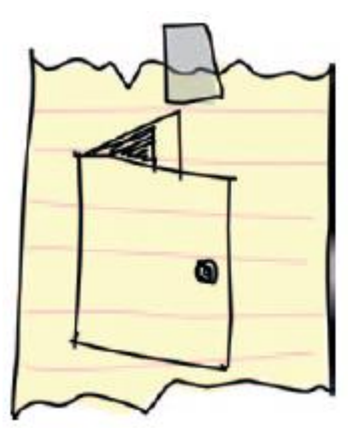
تنها چیزی که در این بازی غایب است، درب خروج می‌باشد. ما با خلق یک اسپرایت برای درب، کار را تمام می‌کنیم: این کار را با ضافه کردن کدی برای شناسایی درب انجام داده و یک شیءdoor به برنامه اضافه می‌کنیم.

ایجاد یک کلاس DOORSPRITE

درست حدس زدید- بایستی یک کلاس دیگر بسازید:DoorSprite. در اینجا شروع کد را مشاهده می‌کنید:

```
class DoorSprite(Sprite):
❶ def __init__(self, game, photo_image, x, y, width, height):
❷ Sprite.__init__(self, game)
❸ self.photo_image = photo_image
❹ self.image = game.canvas.create_image(x, y, ₩
image=self.photo_image, anchor='nw')
❺ self.coordinates = Coords(x, y, x + (width / 2), y + height)
```

```
❻ self.endgame = True
```



همانطوریکـــه در ❶ نشـــان داده شـــده اســـت، تـــابع _init_ کلاس DoorSprite دارای پارامترهایی برای self، یک شیءgame، یک شیء Photo_image، مختصات xو y و width و height تصویر است. در ❷، _init_ را همراه با دیگر کلاس‌های Sprite صدا می‌زنیم.

در ❸، بـا اسـتفاده از متغیـر شـیء بـا همـان نـام PlatformSprite، پارامتر photo_image را ذخیره می‌کنیم. با استفاده از تابع create_image بوم یـک نمـای تصـویر را سـاخته و در ❹ عـدد شناسـایی کـه توسـط آن تـابع برگردانده شده است را با استفاده از متغیر شیءimage ذخیره می‌کنیم.

در ❺، مختصات DoorSptripe را پارامترهای x و y قرار می‌دهیم (که موقعیـت‌هـای x1 و y1 درب هستند) و سپس موقعیت‌های x2 و y2 را محاسبه می‌کنیم. بـا اضـافه کـردن نیمـی از عرض(متغیـر width تقسیم بر۲)به پارامتر x موقعیت x2 را محاسبه می‌کنیم. بعنوان مثـال اگـر x=10 باشد(مختصـات x1 نیـز ۱۰ خواهد بود)و عرض ۴۰ باشد، مختصات x2، ۳۰ خواهد بود(۱۰ بعلاوه‌ی نیمی از ۴۰).

چرا از این محاسبه‌ی گیج‌کننده استفاده می‌کنیم؟ چون برخلاف سکوها، جایی که می‌خواهیم بـه مجرد برخورد Mr. Stick Man با کناره‌ی سکو، متوقف شده و به دویدن ادامه ندهد، در اینجا میخـواهیم جلوی در بایستد. (اتفاق جالبی نخواهد بود اگر Mr. Stick Man کنار در، به دویدن خاتمه دهد!). هنگـام بازی و نزدیک شدن به در، این اتفاق را در عمل مشاهده خواهید کرد.

برخلاف موقعیت x1، محاسبه‌ی موقعیت y1 بسیار ساده است. مقدار متغیر height را بـه پـارامتر y اضافه کرده و نتیجه بدست خواهد آمد.

درنهایت، در ❻، متغیر شیءedngame را برابر باTrue قرار می‌دهـیم. یعنـی زمـانی کـه پیکـره‌ی خطی به در رسید، بازی تمام شود.

کشف درب

حال بایستی کد کـلاس StickFigureSprite تـابع move را کـه برخـورد پیکـره‌ی خطـی بـا یـک اسپرایت در راست یا چپ را تعیین می‌کند، تغییر دهیم. در اینجا اولین تغییر نشان داده شده است:

```
if left and self.x < 0 and collided_left(co, sprite_co):
self.x = 0
left = False
if sprite.endgame:
self.game.running = False
```

در اینجا بررسی می‌کنیم که آیا اسپرایتی که پیکره‌ی خطی با آن برخورد کرده است دارای متغیر endgame=True هست یا نه. دراینصورت، متغیر running را برابر با False قرار خواهیم داد و همه چیز متوقف می‌شود– به انتهای بازی رسیده‌ایم.

همین خط‌ها را به کدی اضافه می‌کنیم که برخورد به راست را وارسی می‌کند:

```
if right and self.x > 0 and collided_right(co, sprite_co):
self.x = 0
right = False
if sprite.endgame:
self.game.running = False
```

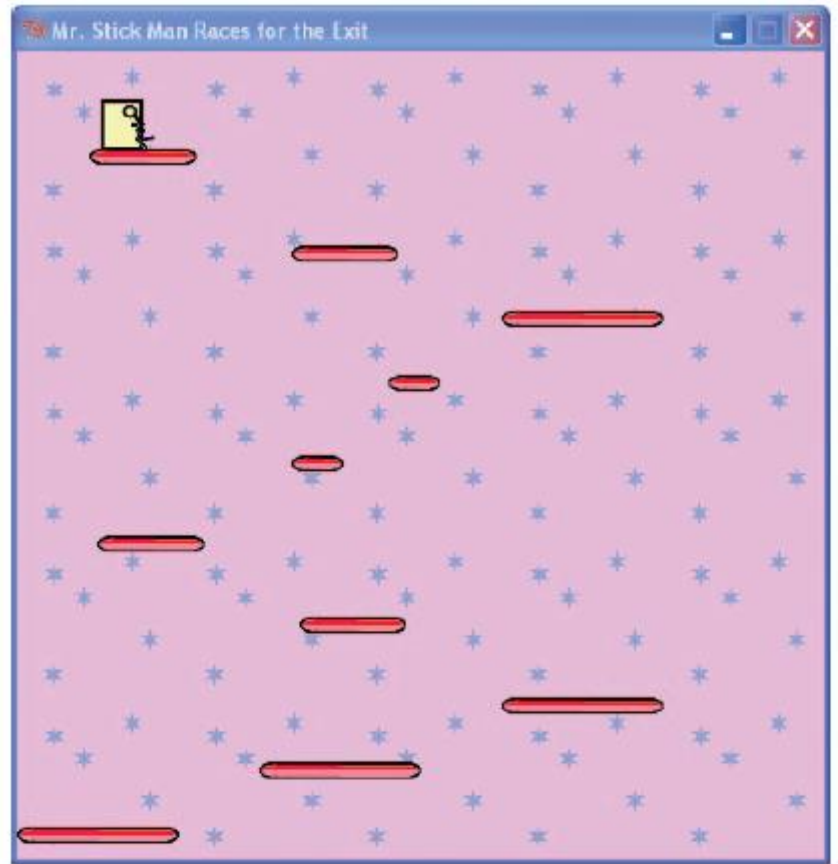
اضافه کردن شیء DOOR

آخرین چیزی که به کد بازی اضافه می‌کنیم، یک شیء برای درب است. این شیء قبل از حلقه-ی اصلی قرار می‌گیرد. دقیقاً قبل از ساختن شیء پیکره‌ی خطی، یک شیء door را خلق می‌کنیم و آن را به لیست اسپرایت‌ها اضافه خواهیم کرد. کد به قرار زیر است:

```
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
door = DoorSprite(g, PhotoImage(file="door1.gif"), 45, 30, 40, 35)
g.sprites.append(door)
sf = StickFigureSprite(g)
g.sprites.append(sf)
g.mainloop()
```

با استفاده متغیر g برای شیء game، یک شیء door و یک PhotoImage می‌سازیم (تصویر درب را در فصل ۱۵ ساخته‌ایم). پارامترهای x و y را به ترتیب ٤٥ و ۳۰ قرار داده تا درب روی سکویی نزدیک به بالای صفحه قرار گیرد و width و height را به ترتیب ٤۰ و ۳٥ درنظر می‌گیریم. شیء door را به لیست اسپرایت‌ها اضافه می‌کنیم، یعنی همان کاری که با بقیه‌ی اسپرایت‌ها در بازی انجام دادیم.

زمانی که Mr. Stick Man به در می‌رسد، نتیجه را مشاهده خواهید کرد. او جلوی درب متوقف می‌شود، نه کنار درب:



بازی نهایی

کد کامل بازی اندکی بیش از ۲۰۰ خط کد شده است. در ادامه، کد تکمیل شده‌ی بازی را خواهید دید. اگر برای اجرای بازی با مشکل روبرو شدید می‌توانید هر تابع(و هر کلاس) را با کد تکمیل شده مقایسه کرده و به اشتباهات خود پی ببرید:

```
from tkinter import *
import random
import time
class Game:
def __init__(self):
self.tk = Tk()
self.tk.title("Mr. Stick Man Races for the Exit")
        self.tk.resizable(0, 0)
self.tk.wm_attributes("-topmost", 1)
self.canvas = Canvas(self.tk, width=500, height=500, ₩
highlightthickness=0)
self.canvas.pack()
```

```
self.tk.update()
self.canvas_height = 500
self.canvas_width = 500
self.bg = PhotoImage(file="background.gif")
w = self.bg.width()
h = self.bg.height()
for x in range(0, 5):
for y in range(0, 5):
self.canvas.create_image(x * w, y * h, ₩
image=self.bg, anchor='nw')
self.sprites = []
self.running = True
def mainloop(self):
while 1:
if self.running == True:
for sprite in self.sprites:
sprite.move()
self.tk.update_idletasks()
self.tk.update()
time.sleep(0.01)
class Coords:
def __init__(self, x1=0, y1=0, x2=0, y2=0):
self.x1 = x1
self.y1 = y1
self.x2 = x2
self.y2 = y2
def within_x(co1, co2):
if (co1.x1 > co2.x1 and co1.x1 < co2.x2) ₩
or (co1.x2 > co2.x1 and co1.x2 < co2.x2) ₩
or (co2.x1 > co1.x1 and co2.x1 < co1.x2) ₩
or (co2.x2 > co1.x1 and co2.x2 < co1.x1):
return True
else:
return False
def within_y(co1, co2):
if (co1.y1 > co2.y1 and co1.y1 < co2.y2) ₩
or (co1.y2 > co2.y1 and co1.y2 < co2.y2) ₩
or (co2.y1 > co1.y1 and co2.y1 < co1.y2) ₩
```

```
or (co2.y2 > co1.y1 and co2.y2 < co1.y1):
          return True
else:
return False
def collided_left(co1, co2):
if within_y(co1, co2):
if co1.x1 <= co2.x2 and co1.x1 >= co2.x1:
return True
return False
def collided_right(co1, co2):
if within_y(co1, co2):
if co1.x2 >= co2.x1 and co1.x2 <= co2.x2:
return True
return False
def collided_top(co1, co2):
if within_x(co1, co2):
if co1.y1 <= co2.y2 and co1.y1 >= co2.y1:
return True
return False
def collided_bottom(y, co1, co2):
if within_x(co1, co2):
y_calc = co1.y2 + y
if y_calc >= co2.y1 and y_calc <= co2.y2:
return True
return False
class Sprite:
def __init__(self, game):
self.game = game
self.endgame = False
self.coordinates = None
def move(self):
pass
def coords(self):
return self.coordinates
class PlatformSprite(Sprite):
def __init__(self, game, photo_image, x, y, width, height):
Sprite.__init__(self, game)
self.photo_image = photo_image
```

```
self.image = game.canvas.create_image(x, y, ₩
image=self.photo_image, anchor='nw')
        self.coordinates = Coords(x, y, x + width, y + height)
class StickFigureSprite(Sprite):
def __init__(self, game):
Sprite.__init__(self, game)
self.images_left = [
PhotoImage(file="figure-L1.gif"),
PhotoImage(file="figure-L2.gif"),
PhotoImage(file="figure-L3.gif")
]
self.images_right = [
PhotoImage(file="figure-R1.gif"),
PhotoImage(file="figure-R2.gif"),
PhotoImage(file="figure-R3.gif")
]
self.image = game.canvas.create_image(200, 470, ₩
image=self.images_left[0], anchor='nw')
self.x = -2
self.y = 0
self.current_image = 0
self.current_image_add = 1
self.jump_count = 0
self.last_time = time.time()
self.coordinates = Coords()
game.canvas.bind_all('<KeyPress-Left>', self.turn_left)
game.canvas.bind_all('<KeyPress-Right>', self.turn_right)
game.canvas.bind_all('<space>', self.jump)
def turn_left(self, evt):
if self.y == 0:
self.x = -2
def turn_right(self, evt):
if self.y == 0:
self.x = 2
def jump(self, evt):
if self.y == 0:
self.y = -4
self.jump_count = 0
```

```
def animate(self):
if self.x != 0 and self.y == 0:
if time.time() - self.last_time > 0.1:
self.last_time= time.time()
        self.current_image += self.current_image_add

if self.current_image >= 2:
self.current_image_add = -1
if self.current_image <= 0:
self.current_image_add = 1
if self.x < 0:
if self.y != 0:
self.game.canvas.itemconfig(self.image, ₩
image=self.images_left[2])
else:
self.game.canvas.itemconfig(self.image, ₩
image=self.images_left[self.current_image])
elif self.x > 0:
if self.y != 0:
self.game.canvas.itemconfig(self.image, ₩
image=self.images_right[2])
else:
self.game.canvas.itemconfig(self.image, ₩
image=self.images_right[self.current_image])
def coords(self):
xy = self.game.canvas.coords(self.image)
self.coordinates.x1 = xy[0]
self.coordinates.y1 = xy[1]
self.coordinates.x2 = xy[0] + 27
self.coordinates.y2 = xy[1] + 30
return self.coordinates
def move(self):
self.animate()
if self.y < 0:
self.jump_count += 1
if self.jump_count > 20:
self.y = 4
if self.y > 0:
```

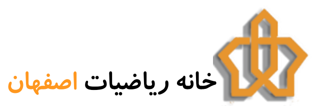
```
self.jump_count -= 1
co = self.coords()
left = True
right = True
top = True
bottom = True
falling = True
if self.y > 0 and co.y2 >= self.game.canvas_height:
self.y = 0
bottom = False
elif self.y < 0 and co.y1 <= 0:
self.y = 0
          top = False
if self.x > 0 and co.x2 >= self.game.canvas_width:
self.x = 0
right = False
elif self.x < 0 and co.x1 <= 0:
self.x = 0
left = False
for sprite in self.game.sprites:
if sprite == self:
continue
sprite_co = sprite.coords()
if top and self.y < 0 and collided_top(co, sprite_co):
self.y = -self.y
top = False
if bottom and self.y > 0 and collided_bottom(self.y, ₩
co, sprite_co):
self.y = sprite_co.y1 - co.y2
if self.y < 0:
self.y = 0
bottom = False
top = False
if bottom and falling and self.y == 0 ₩
and co.y2 < self.game.canvas_height ₩
and collided_bottom(1, co, sprite_co):
falling = False
if left and self.x < 0 and collided_left(co, sprite_co):
```

```
self.x = 0
left = False
if sprite.endgame:
self.game.running = False
if right and self.x > 0 and collided_right(co, sprite_co):
self.x = 0
right = False
if sprite.endgame:
self.game.running = False
if falling and bottom and self.y == 0 ₩
and co.y2 < self.game.canvas_height:
self.y = 4
self.game.canvas.move(self.image, self.x, self.y)
class DoorSprite(Sprite):
def __init__(self, game, photo_image, x, y, width, height):
Sprite.__init__(self, game)
self.photo_image = photo_image
self.image = game.canvas.create_image(x, y, ₩
image=self.photo_image, anchor='nw')
self.coordinates = Coords(x, y, x + (width / 2), y + height)
        self.endgame = True
g = Game()
platform1 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
0, 480, 100, 10)
platform2 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
150, 440, 100, 10)
platform3 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
300, 400, 100, 10)
platform4 = PlatformSprite(g, PhotoImage(file="platform1.gif"), ₩
300, 160, 100, 10)
platform5 = PlatformSprite(g, PhotoImage(file="platform2.gif"), ₩
175, 350, 66, 10)
platform6 = PlatformSprite(g, PhotoImage(file="platform2.gif"), ₩
50, 300, 66, 10)
platform7 = PlatformSprite(g, PhotoImage(file="platform2.gif"), ₩
170, 120, 66, 10)
platform8 = PlatformSprite(g, PhotoImage(file="platform2.gif"), ₩
45, 60, 66, 10)
```

```
platform9 = PlatformSprite(g, PhotoImage(file="platform3.gif"), ₩
170, 250, 32, 10)
platform10 = PlatformSprite(g, PhotoImage(file="platform3.gif"), ₩
230, 200, 32, 10)
g.sprites.append(platform1)
g.sprites.append(platform2)
g.sprites.append(platform3)
g.sprites.append(platform4)
g.sprites.append(platform5)
g.sprites.append(platform6)
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
door = DoorSprite(g, PhotoImage(file="door1.gif"), 45, 30, 40, 35)
g.sprites.append(door)
sf = StickFigureSprite(g)
g.sprites.append(sf)
        g.mainloop()
```

آنچه آموختید

در این فصل بازی خود را تکمیل کردیم: مسابقه‌ی Mr. Stick Man برای رسیدن به درب خروج. یک کلاس را برای پیکره‌ی خطی متحرک خود ساخته و توابعی را برای به حرکت درآوردن آن در صفحه و پویانمایی آن درحین حرکت نوشتیم (تغییر از یک تصویر به تصویر بعد برای ایجاد حس دویدن). از کشف برخورد استفاده کرده تا نشان دهیم چه زمان پیکره‌ی خطی به کناره‌های چپ یا راست بوم برخورد می‌کند و چه زمان به اسپرایت دیگری برخورد کرده است؛ مثلاً یک سکو یا یک درب. همچنین کد برخورد را اضافه کرده تا نشان دهیم په زمان به بالا یا پایین صفحه برخورد می‌کند و مطمئن شویم که وقتی از لبه‌ی سکو لیز می‌خورد، سقوط خواهد کرد. همچنین کدی را اضافه کرده تا نشان دهیم چه زمانMr. Stick Man به درب رسیده است و درنتیجه بازی پایان پیدا می‌کند.



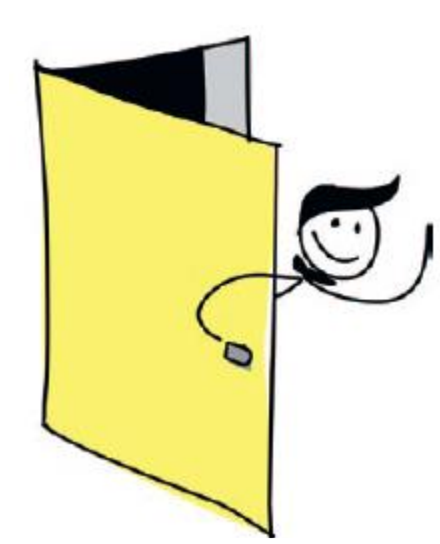
چیستان‌های برنامه‌نویسی

برای بهبود بازی کارهای بسیاری را می‌توانیم انجام دهیم. در اینجا، کار بسیار ساده است و بنابراین می‌توانیم کدی را اضافه کنیم تا بازی حرفه‌ای‌تر و جذاب‌تر شود. ویژگی‌های زیر را اضافه کرده و کد خود را در *http://python-for-kids.com/* وارسی نمایید.

۱) شما پیروز شدید!

همانند متن «پایان بازی» در بازی «بالا و پایین پریدن!» که در فصل ١٤ آن را تکمیل کردیم، زمانی که پیکره‌ی خطی به درب می‌رسد، متن «شما پیروز شدید!» را اضافه می‌کنیم و درنتیجه بازیکن متوجه می‌شود که پیروز شده است.

۲) پویانمایی درب



در فصل ١۵، دو تصویر برای درب ساختیم: یک درب باز و یک درب بسته. زمانی که Mr. Stick Man به درب می‌رسد، تصویر درب بایستی به «درب باز» تغییر پیدا کرده، Mr. Stick Man ناپدید شده و تصویر درب به «درب بسته» تغییر یابد. درنتیجه اینگونه به نظر خواهد رسید که Mr. Stick Man از درب خارج شده و آن را می‌بندد. شما می‌توانید این کار را با تغییر کلاس DoorSprite و کلاس StickFigureSprite انجام دهید.

۳) حرکت دادن سکوها

کلاس جدیدی به نام MovingPlatformSprite را اضافه کنید. این سکو بایستی از یک طرف به طرف دیگر حرکت کند و درنتیجه رسیدن MovingPlatformSprite به دری که در بالا قرار دارد، دشوارتر شود.