A Byte of Python

یک بایت پایتون

Book version 1.90
for Python version 3.0

www.swaroopch.com/notes/Python

Swaroop C H

مترجم: احمد صوفی محمودی

ویرایش فایل: رضا مشکسار

کتاب یک بایت از پایتون

کتاب A Byte of Python یکی از ساده‌ترین و بهترین کتاب‌های آموزشی زبان برنامه‌نویسی Python به زبان انگلیسی است. به دلیل نثر ساده این کتاب، خواندن آن به همه تازه‌کاران توصیه می‌شود. این کتاب توسط آقای Swaroop C H که اصلیتی هندی دارند، نوشته‌شده است. ایشان همچنین سابقه کار کردن برای شرکت‌های Yahoo! و Adobe را دارند. این کتاب به ۱۹ زبان ترجمه شده‌است. در این صفحه کار ترجمه‌ی این کتاب دنبال می‌شود. با ترجمه هر سه فصل این کتاب، نسخه PDF کتاب نیز به‌هنگام‌سازی می‌شود. این کتاب و ترجمه‌ی آن تحت مجوز Creative Commons Attribution-NonCommercial-ShareAlike License 2.0منتشر می‌شوند .

حدود چند ماه پیش بود که تصمیم گرفتم این کتاب را به فارسی ترجمه کنم و توانستم ۶ فصل اول این کتاب را ترجمه کنم. اما به دلایلی نتوانستم ترجمه این کتاب را تمام کنم. حالا برای این‌که ترجمه بهتر صورت بگیرد، ادامه ترجمه را از طریق این ویکی پی می‌گیریم. هر کسی می‌تواند در کار ترجمه این کتاب شرکت کند. اصلا مقدار زیاد ترجمه مورد نظر نیست. هرکس حتی با ترجمه یک جمله می‌تواند به پیش‌رفت ترجمه کتاب کمک کند. امیدوارم روزی بتوانیم این کتاب را در فهرست منابع زبان برنامه‌نویسی Python به زبان فارسی ببینیم .
جا دارد این‌جا از دوست خوبم saimazoon که مرا در ترجمه‌ی این کتاب خیلی یاری کرد، تشکر کنم

مترجم : احمد صوفی محمودی

فهرست مندرجات

دیباچه

فهرست مندرجات

دیباچه

Pythonاید یکی از معدود زبان های برنامه نویسی باشد که سادگی و قدرتمندی را با هم جمع کرده است. این برای تازه کاران هم به اندازه متخصصان خوب است، و مهم تر از آن اینکه برنامه نویسی با آن لذت بخش است. این کتاب می خواهد شما را برای یادگیری این زبان شگفت انگیز کمک کند و نشان دهد چگونه کارها را سریع و بی دردسر انجام دهید - یعنی "یک پادزهر کامل برای مشکلات برنامه نویسی شما ."

این کتاب برای چه کسانی است

این کتاب به عنوان یک راهنما یا آموزش برای زبان برنامه نویسی Python به کار می رود و اساسا تازه کاران را هدف قرار داده است ولی در عین حال برای برنامه نویسان با تجربه نیز سودمند است .اگر شما می دانید کامپیوتر چگونه فایل های متنی را ذخیره می کند، پس شما می توانید Python را از روی این کتاب یاد بگیرید. اگر شما تجربه قبلی برنامه نویسی دارید، شما نیز می توانید Python را از روی این کتاب یاد بگیرید . اگر شما تجربه قبلی برنامه نویسی داشته باشید، شما از تفاوت بین Python و زبان برنامه نویسی مورد علاقه خود، شگفت زده می شوید - من این قبیل تفاوت ها را مشخص کرده ام. یک هشدار گرچه کوچک، Python به زودی زبان برنامه نویسی مورد علاقه شما می شود !

تاریخچه درس

من اولین بار زمانی Python را شروع کردم که به نوشتن یک نصب کننده برای برنامه ام Diamond نیاز داشتم [۱] [از آن جا که می توانستم نصب کردن را آسان کنم. من باید بین Python و Perl یکی را برای ارتباط با کتاب خانه های Qt انتخاب می کردم. من کمی در اینترنت تحقیق کردم و به مقاله ای رسیدم که اریک اس. ریموند، هکر معروف و محترم، در مورد این که Python چگونه زبان برنامه نویسی مورد علاقه اش شده بود، سخن گفته بود. من هم چنین پی بردم که PyQt در مقایسه با Perl-Qt خیلی بهتر است. بنابراین، مصمم شدم که Python بهترین زبان برای من است. سپس من شروع به جست و جوی یک کتاب خوب برای Pythonکردم. هیچ چیزی پیدا نکردم! من تعدادی کتاب از (انتشارات O'Reilly (پیدا کردم، اما آن ها بسیار گران قیمت بودند یا بیش تر شبیه یک دستورالعمل ارجاعی بودند تا یک راهنما. بنابراین من با مستنداتی که همراه Python آمده بودند، بسنده کردم. هرچند که خیلی کوتاه و مختصر بود. این (مستندات) تصور خوبی را در مورد Python می داد، اما کامل نبود. من از آن جایی که تجربه قبلی برنامه نویسی داشتم، با همین خود را پیش بردم. اما این برای تازه کاران مناسب نبود .در حدود شش ماه بعد از اولین تجربه ام با Python ، آخرین نسخه Red Hat 9.0 Linux را نصب کردم و با KWord ور می رفتم. من در این مورد تحریک شدم و ناگهان فکر نوشتن چیزی در مورد Python به ذهنم رسید. من شروع به نوشتن تعداد کمی صفحه کردم، اما به سرعت به 30 صفحه رسید. سپس من در مورد ساختن آن به صورت مفید تر و در قالب یک کتاب جدی تر شدم. بعد از بازنویسی های فراوان، به مرحله ای رسید که به یک راهنمای مفید برای یادگیری زبان Python تبدیل شده بود. من باور داشتم که این کتاب به کمک و احترام من به جامعه متن باز تبدیل شود .این کتاب به عنوان یادداشت های شخصی من در Python شروع شد و من هنوز آن را در همان راه می بینم، اگرچه خیلی تلاش کرده ام تا به ذائقه دیگران خوش بیاید :) در روح واقعی متن باز، من پیشنهادهای فراوانی را دریافت کرده ام، انتقاد و بازخورد از خوانندگان علاقه مند که من را در بهبود این کتاب بسیار کمک کرده است .

وضعیت کتاب

این کتاب در حال ساخت است. فصل های زیادی دائما در حال تغییر و بهبود هستند. به هر حال این کتاب خیلی کامل شده است. شما باید بتوانید Python را از روی این کتاب یاد بگیرید. اگر قسمت هایی از کتاب نادرست یا غیر قابل درک است، لطفا به من بگویید .بیش تر فصل ها برای آینده طراحی شده است، مانند wxPython.

وب سایت رسمی

وب سایت رسمی این کتاب www.byteofpython.info است. از طریق وب سایت ، شما می توانید تمام کتاب را به صورت آنلاین بخوانید یا آخرین نسخه کتاب را دانلود کنید. و همچنین برای من بازخورد بفرستید .

شرایط استفاده

بازخورد

من برای این که این کتاب تا حد امکان جالب و دقیق باشد، تلاش زیادی کرده ام. در هر صورت، اگر شما چیزی که متناقض یا نادرست باشد پیدا کردید، و یا مطلبی به بهبود نیاز داشت، لطفا من را آگاه کنید. تا بتوانم اصلاحات مناسب را انجام دهم. شما می توانید به من در swaroop@byteofpython.info>دسترسی داشته باشید .

چیزی برای فکر کردن

دو راه برای ساختن یک نرم افزار وجود دارد، راه اول این است که آن را بسیار ساده بسازید که روشن است هیچ عیبی وجود ندارد; راه دیگر آن است که آن را بسیار پیچیده بسازید که در این حالت هیچ نقص آشکاری وجود ندارد .سی. ای. آر. هور موفقیت در زندگی یک چیز است که مثل پشتکار و استقامت، خیلی استعداد و فرصت لازم ندارد .سی. دبلیو. وندت

معرفی

فهرست مندرجات

معرفی

Pythonیکی از معدود زبان هایی است که می تواند ادعای قدرتمند بودن و در عین حال ساده بودن را داشته باشد. شما پی خواهید برد و شگفت زده خواهید شد از این که تمرکز کردن روی راه حل مشکل آسان است ، سریع تر از ترکیب و ساختار زبانی که دارید در آن برنامه نویسی می کنید .معرفی رسمی Python این است : Pythonیک زبان برنامه نویسی با یادگیری آسان و قدرتمند است Python .یک ساختمان داده سطح بالای کارامد و یک روش ساده اما موثر را برای برنامه نویسی شیء گرایی دارد .ترکیب زیبا و ماشین نویسی پویای Python، به همراه ذات مفسر گونه اش، آن را تبدیل به زبانی ایده آل برای اسکریپت نویسی و توسعه سریع نرم افزارها در مناطق بسیاری در اکثر پلتفورم ها کرده است .من بیشتر این قابلیت ها را با جزئیات بیشتری در بخش بعدی ذکر خواهم کرد .**توجه** گیدو ون روسوم، سازنده زبان Python ،نام این زبان را بعد از نمایش " Monty Python's Flying Circus"بی بی سی گذاشت. او مخصوصا مارهایی که با پیچش بدنشان دور حیوانات و فشار دادن آنها، آن ها را برای غذا می کشتند، دوست نداشت .

قابلیت های Python

سادگی

Pythonیک زبان ساده و ساده گرا است. خواندن متن یک برنامه خوب که با Python نوشته شده است، مثل خواندن انگلیسی است، هرچند یک انگلیسی سخت !این سرشت کد کاذب Python ، یکی از بزرگترین نقاط قوت آن است. این به شما اجازه می دهد که روی راه حل مشکل سریعتر از خود برنامه تمرکز کنید .

سادگی یادگیری

همان طور که می دانید، شروع به کار کردن با Python بسیار آسان است. همان طور که قبلا گفته شد، Pythonیک ترکیب فوق العاده آسان دارد .

آزاد و متن باز بودن

Pythonیک نمونه از FLOSS یا Free/Libré and Open Source Software است. به عبارت ساده تر، شما به طور آزادانه می توانید کپی هایی از این نرم افزار را توزیع کنید، متن آن را بخوانید، تغییراتی را در آن ایجاد کنید، قطعه هایی از آن را در برنامه های آزاد جدید به کار ببرید، و همان طور که می دانید، می توانید این کارها را انجام دهید FLOSS .بر پایه اندیشه یک اجتماع است که اطلاعات خودشان را به اشتراک می گذارند. این یکی از دلایل خیلی خوب بودن Python است - این ساخته شده است و دائما توسط یک اجتماع که فقط می خواهند یک Python بهتر را ببینند، بهبود می یابد .

زبان سطح بالا بودن

وقتی شما در حال برنامه نویسی با Python هستید، شما هرگز نیازی به نگرانی در مورد جزئیات سطح پایین بودن مثل مدیریت جافظه اشغال شده توسط برنامه ندارید .

قابلیت حمل

به علت طبیعت متن باز آن، Pythonبه پلتفورم های بسیاری برده شده است (یعنی تغییراتی برای کار کردن روی پلتفورم های در آن داده شده است .(تمام برنامه های شما، می توانند روی هر کدام از این پلتفورم ها بدون لازم داشتن هیچ تغییری کار کنند، اگر شما به اندازه کافی مراقب باشید که برخی قابلیت های سیستم های بخصوصی را به کار نبرید .شما می توانید Python را درLinux ، Windows، FreeBSD، Macintosh، Solaris، OS/2، Amiga، AROS، AS/400، BeOS، OS/390، z/OS، Palm OS، QNX، VMS، Psion، Acorn RISC OS، VxWorks، PlayStation، Sharp Zaurus، Windows CEو حتی Pocket PCهم استفاده کنید !

تفسیر شده بودن

این کمی توضیح لازم دارد .یک برنامه که در زبان های کامپایل شده مانند C و C++ نوشته شده باشد، از زبان اصلی یعنی C یا C++ به زبان کامپیوتر (کدهای باینری یعنی 0ها و 1ها (به وسیله یک کامپایلر همراه چندین

انتخاب و پرچم تبدیل می شود. وقتی شما برنامه را اجرا می کنید، برنامه پیوند دهنده/(linker)بارگذار (loader)برنامه را از هارددیسک به حافظه کپی می کند و اجرای آن را شروع می کند .از سویی دیگر، Python، تبدیل شدن به باینری را لازم ندارد. شما تنها مستقیما برنامه را از سورس کد آن اجرا می کنید. از درون، Pythonسورس کد را در داخل یک شکل متوسط به نام bytecodes تبدیل می کند و سپس این را به زبان کامپیوتر شما ترجمه می کند و سپس آن را اجرا می کند. تمام این ها، در حقیقت استفاده از Python را آسان می کند، چون لزومی ندارد که در مورد کامپایل کردن برنامه نگران باشید و مطمئن شوید که کتابخانه های مناسب لینک شده و لود شده باشند. همچنین این برنامه شما را خیلی بیشتر قابل حمل خواهد کرد، از آن جایی که شما می توانید فقط برنامه خود را به کامپیوتر دیگری منتقل کنید و کار خواهد کرد !

شیء گرایی

Pythonپردازه گرایی (procedure-oriented)را به خوبی شیء گرایی (object-oriented)پشتیبانی می کند. در زبان های پردازه گرا، پیرامون پروسه ها و توابعی ساخته می شود که هیچ چیز نیستند، اما برای قسمت هایی از برنامه قابل استفاده مجدد هستند. در زبان های شیء گرا، برنامه پیرامون شیء هایی ساخته می شود که داده ها و عملکرد سیستم را ترکیب می کند Python .یک راه قدرتمند، اما ساده را برای انجام OOP دارد. مخصوصا وقتی که با زبان هایی بزرگی مثل Java و C++ مقایسه شود .

توسعه پذیری

اگر شما به یک قطعه کد خطرناک برای اجرای بسیار سریع نیاز دارید یا می خواهید چند قطعه الگوریتم که باز نشوند داشته باشید، می توانید آن قسمت از برنامه تان را در C یا C++ برنامه نویسی کنید و سپس آن ها را در برنامه خود استفاده کنید .

قابلیت جادادن

شما می توانید Python را در داخل برنامه هایی که با C/C++ نوشته اید، جا بدهید تا بتوانید قابلیت اسکریپت نویسی را به کاربرانتان بدهید .

کتابخانه های گسترده

کتابخانه استاندارد Python به راستی بزرگ است. این به شما کمک می کند که چیزهای گوناگونی را شامل واردکردن عبارات منظم، تولید مستندات، آزمایش دستگاه، رشته کشی (threading)، پایگاه داده، مرورگر وب

GUI(رابط کاربری گرافیکی(، Tkو بقیه چیزهای وابسته به سیستم را انجام دهید. به یاد داشته باشید که تمام این ها همیشه درهرجا که Python نصب شده باشد، قابل دسترسی است. این به عنوان "قوه های توگذاشته" فلسفه Pythonیاد می شود .گذشته از این کتابخانه استاندارد، کتابخانه های باکیفیت گوناگونی مانند wxPython(به نشانی http://wxpython.org)و Twisted(به نشانی http://www.twistedmatrix.com/products/twisted)و Python Imaging Library(به نشانی http://www.pythonware.com/products/pil/index.htm)و کتابخانه های خیلی زیادتری نیز وجود دارند . CGI، Email،FTP، XML، XML-RPC، HTML، فایل های WAV ، رمزنگاری کردن،

خلاصه

Pythonواقعا یک زبان مهیج و قدرتمند است Python .ترکیب کارائی و ویژگی درستی دارد که نوشتن برنامه ها در Python را مفرح و آسان می کند .

چرا پرل (Perl)نه؟

اگر قبلا نمی دانستید، Perlیکی دیگر از زبان های برنامه نویسی به شدت محبوب متن باز تفسیر شده است . اگر تا به حال به نوشتن یک برنامه بزرگ در Perl تلاش کرده باشید، شما این سوال را از خود پرسیده اید! به عبارتی دیگر، برنامه های Perl تا وقتی آسان هستند که کوچک باشند و برای انجام دادن کارها، بر بهبودهای فنی کوچک و اسکریپت ها برتری دارد. هرچند از زمانی که شما شروع به نوشتن برنامه های بزرگ تر نمایید، آنها سریع سنگین می شوند و من در مورد سابقه ام در مورد نوشتن برنامه های Perl بزرگ برای یاهو صحبت می کنم !هنگامی که Perl و Python با هم مقایسه می شوند، برنامه های نوشته شده با Pythonقطعا ساده تر و واضح تر هستند و نوشتن آن ها آسان تر است و از این رو قابل فهم تر هستند و نگهداری از آنان اسان تر است. من Perl را تحسین می کنم و از آن برای پایه ای روزانه برای چیزهای دیگر استفاده می کنم. اما هرگاه که برنامه ای را می نویسم، من همیشه به فکر استفاده کردن از Python می افتم، زیرا برای من طبیعی تر است Perl .دچار تغییرات و دستکاری های زیادی شده است که به نظر می رسد یک بهبود فنی بزرگ است (اما یک جهنم برای بهبود است). متاسفانه به نظر نمی رسد که Perl 6 که در آینده خواهد آمد، هیچ بهبودی در این باره داشته باشد .تنها و مهمترین فایده Perl که احساس می کنم آن را دارد، کتابخانه بزرگ CPAN (the Comprehensive Perl Archive Network) آن است[۱]. همان طوری که از نام آن پیداست، این یک مجموعه بسیار بزرگ از ماژول های Perlاست و واقعا به دلیل حجم خالص و عمقش شگفت انگیز است - شما واقعا هرکاری را با کامپیوتری که این ماژول ها را داشته باشد، می توانید انجام دهید. یکی از دلایل اینکه Perlاز Python کتابخانه های بیش تری دارد این است که Perl زودتر از Python ساخته شده است. شاید

لازم باشد روشی برای انتقال ماژول های Perl به Python را از comp.lang.python پیشهاد کنم :) (http://groups.google.com/groups?q=comp.lang.python)همچنین، ماشین مجازی جدید Parrot برای اجرا در Perl 6 و بقیه زبان های تفسیر شده مانند Ruby و PHP و Tcl به خوبی Pythonطراحی شده است. این برای شما چه معنایی دارد که شاید بتوانید تمام ماژول های Perl را در آینده در Pythonبه کار ببرید؟ بنابراین، شما می توانید بهترین هردو دنیا را داشته باشید - کتابخانه قدرتمند CPANبه همراه زبان قدرتمند .Python در هر صورت، ما مجبور هستیم که فقط صبر کنیم و ببینیم چه اتفاقی رخ خواهد داد .

برنامه نویسان چه می گویند؟

شاید برای شما جالب باشد بخوانید که هکرهایی مثل ESR مجبور شده اند چه چیز در مورد Python بگویند : اریک. اس. ریموند، نویسند کتاب "کلیسای جامع و بازار " است و همچنین کسی است که کلمه متن باز را تجاری کرد. او می گوید که Python زبان برنامه نویسی محبوبش شده است [۲]. [این مقاله یک محرک واقعی برای اولین تجربه من با Python بود .بروس اکل، نویسنده کتاب های مشهور "تفکر در Java" و "تفکر در C++"است. او می گوید هیچ زبانی به اندازه Python او را تولیدکننده تر نکرده است. او می گوید که شاید Pythonتنها زبانی است که تمرکزش بر روی آسان تر کردن کارها برای برنامه نویس است. برای جزئیات بیش تر این مصاحبه را بخوانید[۳]. [پیتر نورویگ یک نویسنده مشهور زبان برنامه نویسی Lisp و مدیر کیفیت جست و جو در Google است(باتشکر از گایدو ون روسوم برای نشان دادن آن). او می گوید که Python همیشه یک قسمت کامل از Google بوده است. شما واقعا می توانید این جمله را با نگاه کردن به صفحه Google Jobs بررسی کنید[۴] [که دانستن Python را به عنوان یک نیاز برای برنامه نویسان فهرست می کند .بروس پرنس بنیان گذار OpenSource.org و پروژه LinuxUser است LinuxUser .قصد دارد که یک توزیع Linux استاندارد شده که چندین شرکت از آن پشتیبانی کنند را بسازد Python .مدعیانی همچون Perl و Ruby را شکست داده است تا خود زبان برنامه نویسی اصلی که توسط LinuxUser پشتیبانی خواهد شد، باشد .

نصب کردنPython

فهرست مندرجات

نصب کردن Python

برای کاربران Linux/BSD

اگر شما از یک توزیع Linux مانند Fedora و Mandriva یا {انتخابتان را اینجا قرار دهید}، یا یک سیستم BSDمانند FreeBSD استفاده می کنید، احتمالا Pythonقبلا روی سیستم شما نصب شده باشد .برای اینکه امتحان کنید Python روی سیستم Linux شما قبلا نصب شده است، یک برنامه shell را باز کنید(مانند Konsole یا (Gnome-Terminal و دستور python -V را، همان طور که در زیر نشان داده شده است، وارد کنید .

```
$ python -V
Python 2.3.4
```

توجه علامت $ اعلان shell است. این اعلان برای شما بسته به تنظیمات سیستم عاملتان مختلف خواهد بود، بنابراین من اعلان را فقط با علامت $ نشان خواهم داد .اگر شما اطلاعات نسخه را مانند چیزی که در بالا نشان داده شده است می بینید، شما از قبل Python را نصب کرده اید .در هر صورت اگر شما یک پیام مانند این پیام دریافت کردید :

```
$ python -V
bash: python: command not found
```

آن وقت شما Python را به صورت نصب شده ندارید. این بسیار بعید است، اما ممکن است .در این مورد، شما دو راه برای نصب کردن Python روی سیستم تان دارید .بسته های binary را با استفاده از نرم افزار مدیریت بسته ای که همراه سیستم عامل تان است، نصب کنید، مانند yum درFedora ، urmpiدر Mandriva

Linux، apt-getدر Debian GNU/Linux ، pkg_addدر FreeBSD و غیره. توجه داشته باشید که برای استفاده از این روش به اتصال اینترنتی نیازمند هستید .متناوبا، شما می توانید بسته های binary را از جایی دیگر تهیه کنید و به کامپیوتر شخصی تان منتقل کنید و آن را نصب کنید .شما می توانید Python را از روی کد منبع آن compile کنید[۱] [و آن را نصب کنید. راهنمای compile کردن در وب سایت تهیه شده است .

برای کاربران Windows

به آدرس http://www.python.org/download/بروید و آخرین نسخه را از این وب سایت download کنید(که 2.3.4 به هنگام تهیه این نوشته بود). این فقط 9 مگابایت حجم دارد که نسبت به اکثر زبان های دیگر کم حجم تر است. نصب کردن آن دقیقا مثل بقیه نرم افزارهای ویندوزی است .

اخطار

اگر به شما امکان داده شد که هر جزء انتخابی را بدون علامت کنید، هیچ کدام را بدون علامت نکنید! برخی از این اجزا مخصوصا IDLE می توانند برای شما مفید باشند .یک واقعیت جالب این است که حدود 70٪ از downloadهای Python مربوط به کاربران Windows است. البته این چهره واقعی موضوع را ازآنجاییکه تقریبا همه کاربران لینوکس از قبل به صورت پیش فرض Python روی سیستم شان نصب شده است، نشان نمی دهد .

به کارگیری Python در خط فرمان Windows

اگر شما می خواهید در خط فرمان Windows از Python استفاده کنید، شما نیاز دارید که مسیر متغیر را به طور مناسب مرتب کنید .برای Windows های 2000 و XP و 2003، بر روی Control Panel -> System -> Advanced -> Environment Variablesکلیک کنید. برروی متغیر PATH در قسمت "System Variables"کلیک کنید، سپس گزینه Edit را انتخاب کرده و "C:\Python23;"را به انتهای هرچیزی که از قبل آنجا بوده، اضافه کنید(بدون نشانه نقل قول). البته نام پوشه مناسب را به کار ببرید .برای نسخه های قدیمی تر Windows ، خطوط زیر را به فایل C:\AUTOEXEC.BAT اضافه کنید : ("PATH=%PATH%;C:\Python23"بدون نشانه نقل قول) و سیستم را دوباره راه اندازی کنید. برای Windows NT، از فایل AUTOEXEC.NT استفاده کنید .

خلاصه

برای یک سیستم Linux ، به احتمال قوی، Pythonاز قبل روی سیستم شما نصب شده است. در غیر این صورت، می توانید با استفاده از نرم افزار مدیریت بسته هایی که همراه سیستم شما هستند، آن را نصب کنید. برای یک سیستم Windows ، نصب کردن Python به اندازه download کردن آن و دوبار کلیک کردن روی آن آسان است. از این پس، ما فرض می کنیم که Python روی سیستم شما نصب است .در ادامه، ما اولین برنامه Python مان را می نویسیم .

اولین قدم ها

فهرست مندرجات

اولین قدم ها

مقدمه

ما حالا مشاهده خواهیم کرد که چگونه برنامه سنتی "Hello World" را در Pythonاجرا کنیم. این به شما خواهد آموخت که چگونه برنامه های Python را بنویسید، ذخیره و اجرا کنید .در اینجا دوراه برای استفاده از

Pythonبرای اجرای برنامه ها وجود دارد - استفاده از اعلان فعل و انفعالی مفسر یا استفاده از یک فایل منبع. ما خواهیم دید که چگونه از هر دو روش استفاده کنیم .

استفاده از اعلان مفسر

مفسر را در خط فرمان با وارد کردن python در اعلان shell آغاز کنید .حالا عبارت 'print 'Hello World را وارد کنید و کلید Enter را بزنید. شما باید کلمات Hello World را به عنوان خروجی مشاهده کنید. برای کاربران ویندوز، شما می توانید مفسر را به شرط اینکه مسیر متغیر را به درستی تعیین کرده باشید، در خط فرمان اجرا کنید. متناوبا، شما می توانید از برنامه IDLE استفاده کنید .IDLE مخفف Integrated DeveLopment Environmentاست. بر روی Start -> Programs -> Python 2.x -> IDLE (Python GUI)کلیک کنید .کاربران Linux نیز می توانند از IDLE استفاده کنند .توجه کنید که علامت های >>>اعلان وارد کردن عبارات Python هستند .

مثال 3.1. به کارگیری اعلان مفسر Python

```
$ python
Python 2.3.4 (#1, Oct 26 2004, 16:42:40)
[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> print 'hello world'
hello world
>>>
```

توجه داشته باشید که Python بی درنگ خروجی را به شما می دهد! چیزی که شما وارد کرده اید، یک عبارت Pythonاست. ما print را برای نشان دادن هر مقداری که شما می خواهید به کار می بریم. اکنون ما متن Hello Worldرا آماده کرده ایم و این بی درنگ روی صفحه نمایش نشان داده می شود .

چگونگی خارج شدن از اعلان Python

برای خارج شدن از اعلان، در صورتی که از IDLE و خط فرمان Linux/BSD استفاده می کنید، Ctrl-dرا فشار دهید. در مورد خط فرمان ویندوز، Ctrl-zو به دنبال آن Enter را فشار دهید .

انتخاب یک ویرایشگر

قبل از اینکه ما در راه نوشتن برنامه های Python در فایل های منبع قدم برداریم، ما به یک ویرایشگر برای نوشتن فایل های منبع نیازمندیم. انتخاب یک ویرایشگر به راستی بسیار سخت است. شما مجبور هستید که یک ویرایشگر را انتخاب کنید همانطوری که شما تمایل داشته باشید خودروی را که می خواهید، بخرید. یک ویرایشگر خوب شما را کمک می کند که برنامه های Python را به سادگی انجام دهید، کار شما را راحت تر کند و شما را در رسیدن به مقصدتان در یک راه سریع تر و امن تر کمک کند(به هدفتان دست پیدا کنید .(یکی از پایه ای ترین لزومات پررنگ کردن ترکیب زبان است که در آن تمام اجزای متفاوت برنامه Python شما رنگی می شوند. بنابراین می توانید برنامه خود را ببینید و کارکرد را تصور کنید .اگر شما از Windows استفاده می کنید، من به شما IDLE را پیشنهاد می کنم IDLE .برجسته کننده ترکیب زبان را و چیزهای بسیار دیگری از جمله امکان اجرای برنامه را درون IDLE در میان دیگر چیزها را داراست. یک توجه مخصوص: از Notepadاستفاده نکنید - این یک انتخاب خوب نیست، زیرا برجسته کردن ترکیب را انجام نمی دهد و به طور مهم، از دندانه گذاری متن پشتیبانی نمی کند که همان طور بعدا می بینیم، در مورد ما خیلی مهم است. ویرایشگرهای خوب مانند) IDLEو همچنین (VIM به صورت خودکار به شما کمک می کنند این کار را انجام دهید .اگر شما از Linux/FreeBSD استفاده می کنید، شما گزینه های زیادی برای انتخاب ویرایشگر دارید. اگر شما یک برنامه نویس باتجربه هستید، باید از قبل از VIM یا Emacs استفاده کرده باشید. نیازی به گفتن نیست که این دو، دو برنامه از قدرتمندترین ویرایشگرها هستند و شما از استفاده از آنان برای نوشتن برنامه های Python تان ... خواهید شد. من شخصا از VIM برای اکثر برنامه هایم استفاده می کنم. اگر شما یک برنامه نویس تازه کار هستید، می توانید از Kate استفاده کنید که یکی از برنامه های مورد علاقه من است. در صورتی که مایل هستید زمانی را برای یادگیری VIM و Emacs اختصاص دهید، در این صورت من خیلی پیشنهاد می کنم که استفاده از هرکدام را یاد بگیرید، زیرا در اجرای طولانی برای شما بسیار مفید خواهد بود . اگر شما هنوز می خواهید گزینه های دیگری از یک ویرایشگر را جست و جو کنید، لیست جامع ویرایشگرهای Pythonرا ببینید[۱] [و انتخاب تان را انجام دهید. شما همچنین می توانید یک IDE را برای Python انتخاب کنید .(Integrated Development Environment)لیست جامع IDE هایی را که از Python پشتیبانی می کنند، برای جزییات بیش تر ببینید [۲] .[یک وقت که شروع به نوشتن برنامه های بزرگ Pythonکنید، IDEها می توانند واقعا خیلی مفید باشند .یک بار دیگر تکرار می کنم، لطفا یک ویرایشگر مناسب را انتخاب کنید - این می تواند نوشتن برنامه های Python را جذاب تر و ساده کند .

استفاده از یک فایل منبع

اکنون بیایید به برنامه نویسی برگردیم. یک رسم وجود دارد که هرگاه یک زبان برنامه نویسی جدید را یاد می گیرید، اولین برنامه ای که می نویسید و اجرا می کنید، برنامه "Hello World" است - تمام آن چه که انجام

می دهد این است که هنگامی که آن را اجرا می کنید، 'Hello World'را بگوید. همانطوری که سیمون کوزنز آن را قرار داده است، این "طلسم باستانی خدایان برنامه نویسی برای کمک به شما برای یادگیری بهتر زبان برنامه نویسی است . :) "ویرایشگر برگزیده تان را باز کنید، برنامه زیر را وارد کنید و آن را بانام helloworld.pyذخیره کنید

```
Example 3.2. Using a Source File
#!/usr/bin/python
# Filename : helloworld.py
print 'Hello World'
```

)فایل منبع (code/helloworld.py :

این برنامه را به وسیله باز کردن پوسته خط فرمان (Linux terminal)یا اعلان (DOS و وارد کردن دستور python helloworld.pyاجرا کنید. اگر شما از IDLEاستفاده می کنید، از منوی Edit -> Run Scriptیا میانبر صفحه کلید Ctrl-F5استفاده کنید. خروجی در زیر نشان داده شده است .

خروجی

```
$ python helloworld.py
Hello World
```

اگر شما خروجی را مثل خروجی نشان داده شده بالا دریافت کردید، تبریک می گویم! - شما با موفقیت اولین برنامه Python تان را اجرا کرده اید .در صورتی که شما یک خطا دریافت کردید، برنامه بالا را همانطور که نشان داده شده است، تایپ کنید و دوباره برنامه را اجرا کنید. توجه داشته باشید که Pythonبه بزرگی یا کوچکی حروف حساس است یعنی print با Print یکی نیست - توجه کنید به حرف کوچک p در اولی و حرف بزرگ P در دومی. همچنین مطمئن شوید که قبل از اولین حرف هر خط هیچ فاصله یا Tab ای وجود نداشته باشد - ما بعدا خواهیم دید که چرا این مهم است .

این چگونه کار می کند؟

اجازه دهید که دو خط اول برنامه را بررسی کنیم. به این ها توضیح (comment)می گویند - هرچیزی در سمت راست علامت # یک comment است و اساسا به عنوان یادداشت هایی برای خواننده برنامه مفید است . Pythonتوضیح ها را به جز در موارد خاصی از اولین خط به کار نمی برد. این را)shebang line خط تعبیه)

می گویند - هر وقتی که اولین حروف فایل منبع #! و به دنبال آن محل یک برنامه باشد، این به سیستم Linux/Unixشما می گوید که هنگامی که برنامه را اجرا کردید، باید با این مفسر اجرا شود. این به صورت مفصل در بخش بعدی شرح داده خواهد شد. توجه کنید که شما همیشه می توانید برنامه را بر روی هر پلتفورمی به وسیله مشخص کردن مستقیم مفسر در خط های دستوری مانند python helloworld.py اجرا کنید .

مهم

از comment ها به صورت نمایان برای توضیح جزئیات مهم برنامه تان استفاده کنید - این برای خوانندگان برنامه تان مفید است، زیرا آن ها به سادگی می توانند درک کنند که برنامه چه کاری را انجام می دهد. به یاد داشته باشید که این شخص می تواند خود شما بعد از شش ماه باشد !به دنبال توضیحات، یک دستور پایتون می آید - این فقط متن 'hello world' را نشان می دهد. در حقیقت print یک)operator عملگر) و 'hello world'به عنوان یک رشته نشان داده می شود - نگران نباشید، ما بعدا این اصطلاحات فنی را با جزئیات بیش تر بررسی می کنیم .

برنامه‌های Python قابل‌اجرا

این تنها توسط کاربران Linux/Unix قابل اجراست، اما شاید کاربران Windowsدر مورد اولین خط برنامه کنجکاو باشند. ابتدا ما مجبوریم که به وسیله دستور chmod به برنامه مجوز اجرا شدن را بدهیم و سپس آن را اجرا کنیم .

```
$ chmod a+x helloworld.py
$ ./helloworld.py
Hello World
```

دستور chmod در اینجا برای تغییر روش فایل به وسیله دادن اجازه اجرا کردن به تمام کاربران سیستم به کار برده شده است. سپس ما برنامه را مستقیما به وسیله مشخص کردن مسیر فایل منبع اجرا می کنیم. ما از ./ برای نشان دادن اینکه برنامه در پوشه کنونی است، استفاده می کنیم .برای اینکه چیزها را جالب تر کنید، می توانید نام فایل را تنها به helloworldتغییر دهید و آن را به صورت ./helloworld. اجرا کنید و این از آن جاییکه سیستم می داند باید به وسیله مفسری که محل آن در اولین خط فایل منبع نوشته شده است آن را اجرا کند، کار خواهد کرد. شما می توانید تا زمانی که محل برنامه را بدانید، آن را اجرا کنید - اما اگر بخواهید برنامه را در هر جایی اجرا کنید، چه؟ شما می توانید این کار را با ذخیره برنامه در یکی از پوشه هایی که در محیط متغییر PATH لیست شده اند، انجام دهید. هرگاه شما برنامه ای را اجرا می کنید، سیستم به دنبال آن برنامه

در هر پوشه ای که در محیط متغییر PATH است، می گردد و سپس آن برنامه را اجرا می کند. شما می توانید این برنامه را در هرجا با کپی ساده فایل منبع به یکی از پوشه های لیست شده در PATH ، قابل دسترسی کنید .

```
$ echo $PATH
/opt/mono/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/b
in:/home/swaroop/bin
$ cp helloworld.py /home/swaroop/bin/helloworld
$ helloworld
Hello World
```

ما می توانیم توسط دستور echo و پیشوند $ متغییر PATH را نشان دهیم تا به shell بفهمانیم که ما مقدار این متغییر را می خواهیم. ما می بینیم که /home/swaroop/binیکی از پوشه هایی است که در متغییر PATHوجود دارد که swaroopنام کاربری است که من در حال استفاده از آن در سیستمم هستم. معمولا یک پوشه مشابه برای نام کاربریتان روی سیستم وجود خواهد داشت. متناوبا، شما می توانید یک پوشه دولخواه را به متغییر PATH اضافه کنید - این می تواند به وسیله اجرای دستور PATH=$PATH:/home/swaroop/mydirانجام گیرد که '/home/swaroop/mydir'پوشه ای است که می خواهم به متغییر PATH اضافه کنم .در صورتی که بخواهید اسکریپت های مفید بنویسید و بخواهید هر وقت و هرجا آن را اجرا کنید، این روش خیلی مفید خواهد بود. این به ساختن دستور خودتان مثل cdیا هر دستور دیگری که آن را در خط فرمان Linux و اعلان DOS استفاده می کنید، شبیه است .

توجه کنید که W.r.t. Pythonیا یک برنامه یا یک اسکریپت یا نرم افزار همه یک معنی دارند .

کمک گرفتن

اگر شما به اطلاعات فوری در مورد هر تابع یا دستوری در Python نیاز دارید، می توانید از دستور اصلی help استفاده کنید. این مخصوصا وقتی که از اعلان مفسر استفاده می کنید، خیلی سودمند است. برای مثال، help(str)را اجرا کنید - این راهنمایی را برای کلاس str نشان می دهد که این برای ذخیره کردن تمام متونی (رشته هایی) که در برنامه به کار برده اید، استفاده می شود. کلاس ها در فصل برنامه نویسی شیء گرایی مفصلا شرح داده خواهد شد .

توجه

برای خارج شدن از راهنما، کلید q را بزنید .به همین ترتیب، می توانید درباره تقریبا هرچیزی در Python اطلاعات کسب کنید. برای یادگیری بیشتر در مورد استفاده کردن از خود help ، از help() استفاده کنید !در صورتی که شما به راهنمایی در مورد عملگرهایی مانند print نیاز داشته باشید، آن گاه شما به تنظیم محیط متغییر PYTHONDOCS به صورت مناسب نیاز دارید. این در Linux/Unix به وسیله استفاده از دستور envبه سادگی قابل انجام است .

```
$ env PYTHONDOCS=/usr/share/doc/python-docs-2.3.4/html/
python
Python 2.3.4 (#1, Oct 26 2004, 16:42:40)
[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> help('print')
```

شما توجه خواهید کرد که من برای معلوم کردن 'print' از علامت نقل قول استفاده کرده ام، چنان که Pythonمی تواندبفهمد که من می خواهم در مورد 'print'کمک به دست آورم و من از آن نمی خواهم که چیزی را چاپ دهد print)کند .(توجه کنید محلی که از آن استفاده کرده ام، محل در لینوکس Fedora Core 3است -این شاید برای توزیع ها و نسخه های مختلف، متفاوت باشد .

خلاصه

اکنون شما باید بتوانید برنامه های Python را به آسانی بنویسید، ذخیره و اجرا کنید. حالا که شما یک کاربر Pythonهستید، بیاید چند مفهوم بیشتر Pythonرا یاد بگیریم .

پایه ها

فهرست مندرجات

پایه‌ها

فقط چاپ کردن 'Hello World' کافی نیست، این طور نیست؟ شما می خواهید چیزهایی بیش از آن انجام دهید - شما می خواهید چند ورودی بگیرید، آن را دستکاری کنید و چیزی را به عنوان خروجی بگیرید. می می توانیم این را در Pythonبه وسیله ثابت ها و متغییرها انجام دهیم .

ثابت‌های لفظی

یک مثال برای ثابت ها لفظی یک عدد مانند 5, 1.23, 9.25 e-3یا یک رشته مانند 'This is a string' یا "It's a string!"است. این یک لفظ نامیده می شود، زیرا لفظی است - شما ارزش آن را بطور لفظی به کار می برید. عدد 2 همیشه خودش را نشان می دهد، نه هیچ چیز دیگری - این یک ثابت است، زیرا ارزش آن را نمی توان تغییر داد. از این رو، تمام این ها به عنوان ثابت های لفظی تلقی می شوند .

اعداد

اعداد در Python چهار گونه اند - اعداد صحیح، اعداد صحیح طویل، اعداد ممیز شناور و اعداد مرکب .یک مثال برای اعداد صحیح 2 است که به تنهایی یک عدد صحیح است .اعداد صحیح طویل، تنها اعداد صحیح بزرگ تر هستند .مثال هایی برای اعداد ممیز شناور 3.23 و 52.3 E-4است. نشانه E توان های 10 را نشان می دهد. در این مورد، 52.3 E-4به معنای 52.3 * 10-4 است .مثال هایی برای اعداد مرکب، (- 5+4 (jو (2.3 - 4.6 (jاست .

رشته‌ها

یک رشته، یک توالی از حروف است. رشته ها اساسا تنها یک گروه از کلمات اند .من تقریبا می توانم تضمین کنم که شما احتمالا در همه برنامه های Python ای که می نویسید، از رشته ها استفاده خواهید کرد. بنابراین، به بخش زیر توجه کنید. چگونگی استفاده از رشته در Python اینجاست :

- استفاده از نشان نقل قول تکی (')

شما می توانید با استفاده از نشان نقل قول تکی مانند'Quote me on this' ، رشته ها را معین کنید .

- استفاده از نشان نقل قول دوتایی (")

رشته های درون نشان نقل قول دوتایی دقیقا مثل رشته های دورن نشان نقل قول تکی عمل می کنند. یک مثال "What's your name?" است .

- استفاده از نشان نقل قول سه تایی) یا (**"""**

شما می توانید با استفاده از نشان نقل قول سه تایی رشته های چند خطی را معین کنید. شما می توانید به طور آزادانه از نشان های نقل قول تکی و دوتایی درون نشان های نقل قول سه تایی استفاده کنید. یک مثال:

```
This is a multi-line string. This is the first line.
This is the second line.
"What's your name?," I asked.
He said "Bond, James Bond."
```

- توالی‌های گریز

فرض کنید که می خواهید رشته ای داشته باشید که دارای نشانه نقل قول تکی (')باشد، چگونه این رشته را معین خواهید کرد؟ مثلا رشته What's your name? است. شما نمی توانید 'What's your name?' را مشخص کنید، زیرا Python در مورد محل آغاز و پایان رشته اشتباه خواهد کرد. بنابراین شما باید نشان دهید که این نشان نقل قول تکی پایان رشته را نشان نمی دهد. این به وسیله کمک چیزی به نام توالی های گریز قابل انجام است. شما نشان نقل قول تکی را مانند '\مشخص می کنید - به خط مورب (backslash)توجه شود. حالا شما می توانید رشته را مانند 'What\'s your name?' مشخص کنید .یک راه دیگر برای نشان دادن این رشته مخصوص، می تواند "What's your name?" یعنی استفاده از نشان نقل قول دوتایی باشد. به همین ترتیب، شما مجبورید که از یک توالی گریز برای استفاده از خود نشان نقل قول دوتایی در یک رشته نشان نقل قول دوتایی استفاده کنید. همچنین شما باید خود خط مورب (backslash)را با استفاده از توالی

گریز \\ نشان دهید .اگر شما بخواهید یک رشته دو خطی را معین کنید چه؟ همانطور که در بالا نشان داده شده است، یک راه استفاده از رشته ای با نشان نقل قول سه تایی است و یا از توالی گریز برای حرف اول خط جدید استفاده کنید - استفاده از \n برای نشان دادن شروع خط جدید. یک مثال : This is the first line\nThis is the second line. یک توالی گریز مفید دیگر برای معلوم کردن این که یک tab است، \t است. در اینجا توالی های گریز بسیار بیشتری وجود دارند، اما من تنها مفیدترین توالی های گریز را در اینجا نام برده ام .یک چیز برای تذکر دادن این است که یک خط مورب (backslash)در انتهای خط نشان می دهد که رشته در خط بعدی ادامه دارد، اما هیچ خط جدیدی اضافه نشده است .مثلا :

```
"This is the first sentence.\
This is the second sentence."
```

معادل

```
"This is the first sentence. This is the second
sentence."
```

است .

- رشته‌های خام

اگر شما لازم دارید تعدادی رشته را معین کنید که هیچ پردازش مخصوصی مانند توالی های گریز در آن استفاده نشده است، چیزی که می خواهید مشخص کردن یک رشته خام به وسیله پیشوند r یا R است. یک مثال :

```
r"Newlines are indicated by \n"
```

- رشته‌های یونیکد

Unicodeیک راه استاندارد برای نوشتن متون بین المللی است. اگر شما می خواهید متن را در زبان محلی خود مانند هندی و عربی بنویسید، آنگاه شما به یک ویرایشگر متن با قابلیت پشتیبانی از Unicode نیازمندید. به همین ترتیب، Pythonبه شما اجازه می دهد که متن Unicode را استفاده کنید - تمام چیزهایی که لازم است انجام دهید، استفاده از پیشوند u یا U است. مثلا

```
u"This is a Unicode string."
```

به یاد داشته باشید که وقتی با فایل های متنی سروکار دارید از رشته های Unicodeاستفاده کنید، مخصوصا وقتی که می دانید که فایل، متن نوشته شده به غیر از انگلیسی را دربر خواهد داشت .

- رشته‌ها تغییر ناپذیرند

این به این معنی است که وقتی رشته ای را می سازید، نمی توانید آن را تغییر دهید.اگرچه شاید این یک چیز بد به نظرآید، اما واقعا این چنین نیست .ما خواهیم دید که چرا این یک محدودیت برای برنامه های بعدی که ما می نویسیم، نیست .

- رشته لفظی الحاقی

اگر شما دو رشته لفظی را کنار هم قرار دهید، آن ها به صورت خودکار توسط Pythonادغام می شوند. مثلا 'What\'s' 'your name?'به صورت خودکار به "What's your name?"تبدیل می شود .

تذکر برای برنامه نویسان C/C++

در Python هیچ نوع داده‌ٔ char جداگانه ای وجود ندارد. هیچ نیاز واقعی برای آن وجود ندارد و من مطمئن هستم که شما کمبود آن را حس نخواهید کرد .

تذکر برای برنامه نویسان Perl/PHP

به یاد داشته باشید که رشته های دارای نشان نقل قول تکی و رشته های نقل قول دوتایی یکی هستند - در هر صورت آن ها متفاوت نیستند .

تذکر برای کاربران عادی

همیشه وقتی که با عبارات عادی سروکار دارید از رشته های خام استفاده کنید. در غیر این صورت شاید تعداد زیادی لازم باشد .

متغییرها

استفاده محض از ثابت های لفظی می تواند خیلی زود خسته کننده شود - ما به چند راه ذخیره هر اطلاعاتی و دستکاری صحیح آن نیازمندیم. این جا مکانی است که متغییرها وارد صحنه می شوند. متغییرها کاملا چیزی هستند که معنی می دهند - ارزش آن ها می تواند تغییر پیدا کند یعنی هر چیزی را با استفاده از متغییرها می

توانید ذخیره کنید. متغییرها فقط قسمت هایی از حافظه کامپیوتر شما هستند که مقداری اطلاعات را در آن ذخیره می کنید.بر خلاف ثابت های لفظی، شما به روش هایی برای دسترسی به این متغییرها نیازمندید و از این رو شما آن ها را نام گذاری می کنید .

نام گذاری شناسه

متغییرها مثال هایی از شناسه ها هستند. شناسه ها نام هایی هستند که برای شناختن چیزی به آن داده می شود. اینجا تعدادی قانون وجود دارد که برای نام گذاری شناسه ها مجبورید از آن پیروی کنید :اولین حرف شناسه باید یکی از حروف الفبا(بزرگ یا کوچک) یا یک خط زیرین ('_')باشد .بقیه نام شناسه می تواند شامل حرف الفبا(بزرگ یا کوچک)، خطوط زیرین('_') یا اعداد(0-9) باشد .نام های نشانه به بزرگی و کوچکی حروف حساس هستند. برای مثال، myname وmyNameیکی نیستند. به حرف کوچک n در اولی و حرف بزرگ Nدر دومی توجه کنید i, __my_name, name_23 .و a1b2_c3 مثال هایی از نام های شناسه معتبر هستند 2things, this is spaced out .و my-name مثال هایی از نام های شناسه نامعتبر هستند .

انواع داده‌ها

متغییرها می توانند ارزش هایی از دستورات گوناگون را که انواع داده (data types)نامیده می شوند، نگه داری کنند. دستورات اصلی اعداد و رشته های هستند که ما در مورد آن ها بحث کرده ایم. در فصل های بعد، خواهیم دید که چگونه با استفاده از کلاس ها دستورات خودمان را بسازیم .

اشیا

به یاد داشته باشید که Python هرچیزی که در برنامه به کار گرفته شود را به عنوان شیء تلقی می کند. در عوض اینکه بگوییم چیزی، ما می گوییم شیئی .

تذکر برای کاربران برنامه نویسی شیءگرایی

Pythonشدیدا در دریافتن اینکه همه چیز شامل اعداد، رشته ها و حتی توابع شیء هستند، شیءگرا است .ما حالا می بینیم چگونه از متغییرها همراه با ثابت های لفظی استفاده کنیم. مثال پیش رو را ذخیره کنید و برنامه را اجرا کنید .

چگونگی نوشتن برنامه های Python

از این پس، روش استاندارد برای ذخیره و اجرای یک برنامه Python از این قرار است -1 :ویرایشگر مورد علاقه خود را باز کنید -2 .کدهای برنامه را که در مثال داده شده است را وارد کنید -3 .آن را به عنوان یک فایل با همان نام فایلی که در کامنت به آن اشاره شده است، ذخیره کنید. من از این قرارداد که تمام برنامه های Pythonرا با پسوند .pyذخیره کنم، پیروی می کنم -4 .مفسر را با دستور python program.py اجرا کنید یا از IDLE برای اجرای برنامه ها استفاده کنید. شما همچنین می توانید از روش قابل اجرا کردن استفاده کنید که قبلا در مورد آن بحث شده است .مثال 4.1. استفاده از متغییرها و ثابت های لفظی

```
# Filename : var.py
i = 5
print i
i = i + 1
print i
s = '''This is a multi-line string.
This is the second line.'''
print s
```

خروجی

```
$ python var.py
5
6
This is a multi-line string.
This is the second line.
```

این چگونه کار می کند؟

اینجا چگونگی کارکردن برنامه مطرح است. ابتدا ما ارزش ثابت لفظی 5 را به متغییر i به وسیله عملگر واگذاری(=) می دهیم. این خط یک دستور نامیده می شود، زیرا مشخص می کنید که چیزی باید این گونه انجام شود. ما اسم متغییر i را به ارزش 5 پیوند می دهیم. سپس ما ارزش i را با استفاده از دستور print چاپ می کنیم که به طور از پیش تعیین شده، فقط ارزش متغییر را روی صفحه نمایش چاپ می کند .سپس ما 1 را به ارزشی که i دارد اضافه می کنیم و آن را دوباره ذخیره می کند. سپس ما آن را چاپ می کنیم و همان طور که انتظار داشتیم، ما مقدار 6 را می گیریم .به همین ترتیب، ما رشته لفظی را به عنوان متغییر s تعیین می کنیم و سپس آن را چاپ می کنیم .

تذکر برای برنامه نویسان C/C++

متغییر ها تنها با تعیین یک ارزش برای آن ها به کار می روند. هیچ بیانیه یا تعریف نوع داده ای لازم نیست/به کار نمی رود .

خط های منطقی و فیزیکی

یک خط فیزیکی چیزی است که در حین نوشتن برنامه می بینید. یک خط منطقی چیزی است که Python به عنوان یک دستور تکی می بیند Python .ضمنا فرض می کند که هر خط فیزیکی مانند یک خط منطقی است .یک مثال برای یک خط منطقی یک دستور مانند 'print 'Hello World است - اگر این تنها روی یک خط بود(همانطوری که در یک ویرایشگر می بینید)، آن گاه این شبیه یک خط فیزیکی است .ضمنا Python کاربرد یک دستور تکی را بر طبق هر خط تقویت می کند که کد را خواناتر می کند .اگر می خواهید بیش تر از یک خط را در یک خط فیزیکی مشخص کنید، آنگاه مجبورید که این را به وسیله یک نقطه و ویرگول(;) مشخص کنید که انتهای خط/دستور منطقی را نشان می دهد. مثلا i = 5 print i امثل i = 5; print i; موثر است و همان چیز می تواند این گونه نوشته شود i = 5; print i; یا حتی i = 5; print i به هر حال، من شدیدا توصیه می کنم که به نوشتن تنها یک خط منطقی درون یک خط فیزیکی عادت کنید. از بیش از یک خط فیزیکی برای یک خط منطقی در صورتی استفاده کنید که خط منطقی واقعا طولانی باشد. هدف این است که در صورت امکان از نقطه و ویرگول به مقدار زیاد استفاده نکنید، از آنجایی که این موجب خواناتر شدن کد می گردد. در حقیقت، من هیچ وقت یک علامت نقطه و ویرگول را در یک برنامه Python به کار نبرده ام و حتی ندیده ام .یک مثال نوشتن یک خط منطقی تعداد زیادی خط فیزیکی را می پوشاند، در زیر آمده است. این به الحاق خط واضح (explicit line joining)اشاره دارد .

```
s = 'This is a string. \
This continues the string.'
print s
```

این، خروجی

```
This is a string. This continues the string.
```

می دهد .به همین ترتیب،

```
print \
i
```

با

```
print i
```

یکسان است .گاهی اوقات یک فرض مطلق وجود دارد که شما به استفاده از backslash نیاز ندارید. این حالتی است که خط منطقی از پرانتز، کروشه یا آکولاد استفاده می کند. این الحاق خط واضح (explicit line joining)نامیده می شود. شما می توانید این را هنگامی که ما برنامه ها را با استفاده از لیست ها در فصل های بعدی می نویسیم، در عمل ببینید .

Indentation

فضای سفید در پایتون مهم است. در حقیقت، فضای سفید اول خط مهم است. این Indentation(دندانه گذاری) نامیده می شود. فضای سفید (space)ها و tab ها) در ابتدای خط منطقی برای تعیین سطح خط منطقی به کار می رود که در عمل برای تعیین گروه بندی دستورها به کار می رود .این یعنی دستور هایی که با هم به کار می روند، باید indentation یکسانی داشته باشند. هر گروه این چنینی از عبارات یک بلاک (block)نام دارد. ما در فصل های بعدی خواهیم دید که چگونه بلاک ها مهم هستند .چیزی که باید به خاطر داشته باشید این است که چگونه indentation غلط می تواند باعث زیاد شدن خطاها شود. برای مثال :

```
i = 5
 print 'Value is', i # Error! Notice a single space at
the start of the line
print 'I repeat, the value is', i
```

وقتی که این را اجرا می کنید، شما خطای زیر را دریافت می کنید :

```
  File "whitespace.py", line 4
    print 'Value is', i # Error! Notice a single space
at the start of the line
    ^
SyntaxError: invalid syntax
```

توجه کنید که یک فاصله در ابتدای خط دوم وجود دارد. خطای اشاره شده توسط Pythonبه ما می گوید که ترکیب برنامه نامعتبر است، یعنی برنامه به درستی نوشته نشده است. آنچه به شما نشان می دهد این است که شما نمی توانید به صورت دلخواه بلاک های جدیدی از دستور را آغاز کنید(البته به جز بلاک اصلی که تمام آن را استفاده کرده اید). مواردی که شما می توانید از بلاک های جدید استفاده کنید، در فصل های بعدی مانند فصل control flow مفصلا شرح داده خواهند شد .

چگونگی دندانه گذاری

از یک ترکیب از tab ها و space ها برای دندانه گذاری به دلیل کارنکردن درست در تمام platform های مختلف استفاده نکنید. من شدیدا توصیه می کنم که شما از یک tab تکی یا دو یا چهار space برای هر سطح دندانه گذاری استفاده کنید .یکی از این سه شیوه دندانه گذاری را انتخاب کنید. مهم تر آن که یکی را انتخاب کنید و به صورت پایدار از آن استفاده کنید، یعنی تنها از آن شیوه دندانه گذاری استفاده کنید .

خلاصه

اکنون ما از جزئیات زیادی گذشته ایم، ما می توانیم به چیزهای جالب تری هم چون عبارات گردش کنترل تغییر دهیم. مطمئن باشید که با چیزهایی که در این فصل خوانده اید راحت باشید .

عملگرها و عبارات

فهرست مندرجات

عملگرها و عبارات

مقدمه

بیشتر دستورهایی(خط های منطقی) که می نویسید، شامل عبارات هستند. یک مثال ساده برای عبارات 2 + 3 است. یک عبارت می تواند درون عملگرها و عملوندها شکسته شوند .عملگرها عامل هایی هستند که چیزی را انجام می دهند و می توانند توسط نمادهایی مانند + یا واژه های کلیدی مخصوص نشان داده شوند. عملگرها برای کارکردن به مقداری داده و داده هایی که)operands عملوند) نامیده می شوند، نیازمندند. در این مورد، 2 و 3 عملوند هستند .

عملگرها

ما به صورت مختصر نگاهی به عملگرها و کاربردشان می اندازیم :

نکته

شما می توانید عباراتی که در مثال ها هستند را با استفاده از مفسر به صورت تعاملی بررسی کنید. مثلا، برای تست عبارت 2 + 3، از اعلان مفسر تعاملی Pythonاستفاده کنید .

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

جدول 5.1. عملگرها و کاربردشان

اولویت عملگرها

اگر شما یک عبارت مانند 2 + 3 * 4 داشتید، عمل جمع اول انجام شده است یا عمل ضرب؟ ریاضیات دبیرستان ما به ما می گوید که عمل ضرب ابتدا باید انجام شود - این یعنی عملگر ضرب اولویت بیشتری نسبت به عملگر جمع دارد .جدول زیر(با جدول موجود در راهنمای مرجع Python یکسان است (عملگرهایی که تا کنون به آن برنخورده ایم، در فصل های آتی شرح داده خواهند شد .عملگرهای با حق تقدم یکسان، در جدول بالا در یک ردیف لیست شده اند. مثلا + و -حق تقدم یکسان دارند .

ترتیب ارزیابی

به صورت پیش فرض، جدول حق تقدم عملگر تصمیم می گیرد که کدام عملگرها قبل از دیگر عملگرها ارزیابی می شوند. اما اگر بخواهید ترتیب محاسبه را تغییر دهید، می توانید از پرانتزها استفاده کنید. برای مثال، اگر می خواهید اضافه کردن قبل از ضرب در یک عبارت انجام شود، می توانید چیزی مثل (2 + 3) * 4 استفاده کنید .

شرکت پذیری

عملگرها معمولا از چپ به راست به هم می پیوندند، یعنی عملگرهایی با حق تقدم یکسان از چپ به راست انجام می شوند. مثلا 2 + 3 + 4 مثل (2 + 3) + 4 محاسبه می شود. بعضی از عملگرها مثل عملگرهای واگذاری شرکت پذیری راست به چپ دارند. یعنی a = b = c مثل a = (b = c) رفتار می کند .

عبارت ها

مثال 5.1. به کار بردن عبارت ها

```
#!/usr/bin/python
# Filename: expression.py
length = 5
breadth = 2
area = length * breadth
print 'Area is', area
print 'Perimeter is', 2 * (length + breadth)
```

خروجی

```
$ python expression.py
Area is 10
Perimeter is 14
```

این چگونه کار می کند؟

درازا و پهنای مستطیل در متغییزهایی با همین نام ذخیره می شود. ما این ها را برای محاسبه مساحت و محیط مستطیل با کمک عبارات به کار می بریم. ما نتیجه عبارت length * breadth را در متغییر area ذخیره می کنیم و سپس با استفاده از دستور print آن را چاپ می کنیم. در مورد دومی، ما مستقیما ارزش عبارت 2 * (length + breadth)را در دستور print به کار می بریم .همچنین توجه کنید چگونه Python خروجی را زیبا چاپ می کند. حتی با وجود این که فضایی بین 'Area is' و متغییر area مشخص نکرده ایم Python . آن را برای ما قرار می دهد، چنان که ما یک خروجی تمیز زیبا می گیریم و برنامه این چنین خیلی بیشتر قابل خواندن می شود(از آنجایی که ما نگران فاصله در خروجی نیستیم). این مثال است از این که چگونه Python زندگی را برای برنامه نویس آسان تر می کند .

خلاصه

ما چگونگی استفاده از عملگرها، عملوندها و عبارات را دیده ایم - این ها بلوک های اساسی ساختمان هر برنامه ای هستند. سپس ما خواهیم دید که چگونه از این ها در برنامه مان با استفاده از دستورات استفاده کنیم .

روند کنترل

فهرست مندرجات

فصل 6. روند کنترل

مقدمه

در برنامه هایی که تاکنون دیده ایم، یک سری از دستورات وجود داشته است و Pythonبه درستی آن ها را در جای خودش اجرا کرده است. اگر شما خواستید که روند چگونگی انجام آن را تغییر دهید چه؟ برای مثال، شما می خواهید که برنامه مقداری تصمیم بگیرد و چیزهای متفاوتی را بسته به موقعیت انجام دهد .مانند چاپ کردن 'Good Morning' یا 'Good Evening' بسته وقت روز؟

همان طوری که احتمالا حدس زده اید، این با استفاده از دستورات روند کنترل قابل انجام است. سه دستور روند کنترل در Python وجود دارندif - ، forو .while

دستورif

دستور if برای بررسی کردن وضعیت به کار می رود و اگر وضعیت درست باشد، ما یک مقدار از دستورات را اجرا می کنیم if-block)نامیده می شود)، دیگر اینکه ما مقدار دستورات دیگری را پردازش می کنیم -else) blockنامیده می شود). عبارت else اختیاری است .

استفاده از دستورif

مثال 6.1. استفاده از دستورif

```
#!/usr/bin/python
# Filename: if.py
number = 23
guess = int(raw_input('Enter an integer : '))
if guess == number:
        print 'Congratulations, you guessed it.' # New
block starts here
        print "(but you do not win any prizes!)" # New
block ends here
elif guess < number:
        print 'No, it is a little higher than that' #
Another block
        # You can do whatever you want in a block ...
else:
        print 'No, it is a little lower than that'
        # you must have guess > number to reach here
print 'Done'
# This last statement is always executed, after the if
statement is executed
```

خروجی

```
$ python if.py
Enter an integer : 50
No, it is a little lower than that
Done
$ python if.py
Enter an integer : 22
```

```
No, it is a little higher than that
Done
$ python if.py
Enter an integer : 23
Congratulations, you guessed it.
(but you do not win any prizes!)
Done
```

این چگونه کار می کند؟

در این برنامه، ما حدس هایی را از کاربر می گیریم و آن را بررسی می کنیم اگر عددی باشد که ما داریم. ما متغییر number را به هر عدد صحیحی که بخواهیم نسبت می دهیم، مانند 23. سپس، حدس کاربر را با استفاده از تابع raw_input()می گیریم. تابع ها تنها قسمت های قابل استفاده دوباره برنامه ها هستند. ما درباره آن ها در فصل بعد بیشتر خواهیم خواند .

ما یک رشته را برای تابع raw_input در نظر می گیریم که آن را روی صفحه نشان می دهد و برای دریافت ورودی از طرف کاربر صبر می کند. بار اولی که چیزی را وارد می کنیم و enter را فشار می دهیم، تابع، ورودی را پس می فرستد که در مورد raw_input یک رشته است. سپس ما با استفاده از int رشته را به عدد صحیح تبدیل می کنیم. سپس، آن را متغییر guess ذخیره می کنیم. در حقیقت int یک کلاس است، اما تمام چیزهایی که در حال حاضر لازم است بدانید این است که شما می توانید آن را برای تبدیل رشته به عدد صحیح به کار ببرید(...شامل یک عدد صحیح معتبر در متن است .(

بعد ما حدس کاربر را با عددی که انتخاب کرده ایم، مقایسه می کنیم .اگر با هم برابر بودند، ما یک پیام موفقیت را نشان می دهیم. توجه کنید که برای اینکه به Python بگوییم کدام دستور به کدام بلاک تعلق دارد، از سطوح دندانه گذاری استفاده می کنیم. این دلیل با اهمیت بودن دندانه گذاری در Pythonاست. امیدوارم شما قاعده 'یک tab به ازای هر سطح دندانه گذاری ' را تحمل کنید. آیا این چنین هستید؟ توجه کنید که دستور if شامل یک دو نقطه در انتهایش است - ما به Python نشان می دهیم که یک بلاک از عبارات در زیر هستند .

آن گاه، ما بررسی می کنیم که اگر حدس از عدد کمتر باشد، و اگر این چنین باشد، ما به کاربر اطلاع می دهیم که کمی بیش تر از آن را حدس بزند .چیزی که اینجا از آن استفاده کرده ایم، دستور elif است که در حقیقت دو دستور وابسته if else-if else را درون یک دستور ترکیب شده if-elif-else ترکیب می کند. این برنامه را آسان تر می کند و مستلزم ساده کردن مقدار دندانه گذاری است .

دستور های elif و else همچنین باید یک دو نقطه در انتهای خط منطقی که به دنبال بلاک متناظر دستور ها آمده است، داشته باشند(البته با دندانه گذاری مناسب .(

شما می توانید یک دستور if دیگر درون if-block یک دستور if دیگر داشته باشید - این یک دستور if تو در تو نامیده می شود .

به یاد داشته باشید که قسمت های else و elif اختیاری هستند. یک دستور if کوچک معتبر این چنین است :

```
if True:
        print 'Yes, it is true'
```

بعد از اینکه Python اجرا کردن دستور کامل if را همراه با عبارات elif و elseتمام کرد، آن را به دستور بعدی در بلاک همراه دستور if منتقل می کند .در این مورد، بلاک اصلی است که اجرای برنامه شروع می شود و دستور بعدی print 'Done'است. بعد از آن، Pythonانتهای برنامه را می بیند و به سادگی آن را تمام می کند .اگر چه این یک برنامه خیلی ساده است، من به بسیاری از چیزهایی که همواره در این برنامه ساده باید به آن توجه کنید، اشاره کرده ام. همه این ها خیلی آسان اند(و به طور شگفت آور آسان برای آن هایی از شما که سابقه C/C++ دارید) و مستلزم این است که شما در ابتدا با همه این ها آشنا شوید. اما بعد از آن، شما با آن راحت خواهید بود و برای شما طبیعی خواهد بود .

تذکر برای برنامه نویسانC/C++

هیچ دستور switch ای در Python وجود ندارد. شما می توانید از دستور if..elif..elseبرای انجام همان چیز استفاده کنید(و در بعضی موارد، از یک واژه نامه برای انجام سریع آن استفاده کنید .(

دستورwhile

دستور while به شما این اجازه را می دهد که یک بلاک از دستور را مکررا تا زمانی که شرط صحیح باشد، اجرا کنید. یک دستور while یک مثال از چیزی است که دستور looping نامیده می شود. یک دستور while می تواند یک عبارت else اختیاری را داشته باشد .

استفاده از دستورwhile

مثال 6.2. استفاده از دستورwhile

```
#!/usr/bin/python
# Filename: while.py
number = 23
running = True
while running:
        guess = int(raw_input('Enter an integer : '))
        if guess == number:
                print 'Congratulations, you guessed
it.'
                running = False # this causes the while
loop to stop
        elif guess < number:
                print 'No, it is a little higher than
that.'
        else:
                print 'No, it is a little lower than
that.'
else:
        print 'The while loop is over.'
        # Do anything else you want to do here
print 'Done'
```

خروجی

```
$ python while.py
Enter an integer : 50
No, it is a little lower than that.
Enter an integer : 22
No, it is a little higher than that.
Enter an integer : 23
Congratulations, you guessed it.
The while loop is over.
Done
```

این چگونه کار می کند؟

در این برنامه، ما هنوز بازی حدس زدن را انجام می دهیم، اما با این تفاوت که کاربر می تواند حدس زدن را تا زمانی که حدسش درست باشد، ادامه دهد -هیچ نیازی نیست که مکررا برنامه را برای هر حدس اجرا کنید همان طوری که قبلا آن را انجام داده ایم. این به طور مناسب کاربرد دستور while را نشان می دهد .

ما مکان دستورهای raw_input و if را به درون حلقه while تغییر می دهیم و قبل از حلقه while ،به متغییر running عبارت True را نسبت می دهیم .ابتدا ما بررسی می کنیم که اگر متغییر running برابر Trueباشد، برای اجرای while-blockمتناظر اقدام شود. بعد از اینکه این بلاک اجرا شد، شرط دوباره بررسی می شود که در اینجا متغییر running است. اگر این درست باشد، ما while-blockرا دوباره اجرا می کنیم. دیگر اینکه ما به اجرای else-block اختیاری ادامه می دهیم و سپس به دستور بعدی را ادامه می دهیم .

بلاک else زمانی که شرط حلقه while به False تبدیل شود، اجرا می شود -این حتی ممکن است اولین دفعه ای باشد که شرط بررسی شود. اگر یک عبارت elseبرای حلقه while وجود داشته باشد، همیشه اجرا می شود، مگر اینکه شما یک حلقه while داشته باشید که برای همیشه بدون هیچ گریزی حلقه بزند !

Trueو False نوع های بولی نامیده می شوند و شما می توانید آن ها را به ترتیب به عنوان 1 و 0 بشناسید. استفاده این جایی که موقعیت یا بررسی کردن مهم است و ارزش واقعی مثل 1 ندارد، مهم است .

else-blockدر حقیقت از آن جاییکه که شما می توانید این دستورات را در همان بلاک قرار دهید بعد از دستور while برای دریافت نتیجه یکسان، اضافی است(مانند دستور .(while

تذکر برای برنامه نویسان++C/C

به یاد داشته باشید که شما می توانید عبارت else را برای حلقه while داشته باشید .

حلقهfor

دستور for..in یک دستور ایجاد حلقه دیگر است که سراسر توالی یک شیء را تکرار می کند. یعنی سراسر هر بخش در یک توالی را می گردد. ما در مورد توالی ها در فصل های بعدی بیشتر خواهیم دید. چیزی که حالا لازم است بدانید این است که یک توالی تنها یک مجموعه منظم از اجزاست .

استفاده از دستورfor

مثال 6.2. استفاده از دستورfor

```
#!/usr/bin/python
# Filename: for.py
for i in range(1, 5):
        print i
else:
        print 'The for loop is over'
```

خروجی

```
$ python for.py
1
2
3
4
The for loop is over
```

این چگونه کار می کند؟

در این برنامه ما یک توالی از اعداد را چاپ می کنیم. ما این توالی اعداد را با استفاده از تابع توکار range تولید می کنیم .

چیزی که ما اینجا انجام می دهیم در نظر گرفتن دو عدد است و range یک توالی اعداد از عدد اول تا عدد دوم را گزارش می کند. مثلا، range(1,5)توالی [1, 2, 3, 4] را نشان می دهد. به صورت پیش فرض، rangeیک شمارش مرحله از 1 را می گیرد. اگر ما یک عدد سومی را برای range در نظر بگیریم، به شمارش مرحله ای تبدیل می شود. مثلا range(1,5,2) توالی [1,3] را نشان می دهد. به یاد داشته باشید که محدوده تا عدد دوم گسترش می یابد. یعنی عدد دوم را در بر ندارد .

حلقه for سپس تمام این محدوده را تکرار می کند (for i in range(1,5) - مساوی for i in [1, 2, 3, 4] است که مانند تعیین هر عدد(یا شیء (یکی یکی در توالی به i است و سپس اجرای هر بلاک از دستور برای همه ارزش های i است. در این مورد، ما تنها ارزشِ در بلاکِ دستور را چاپ می کنیم .

به یاد داشته باشید که بخش else اختیاری است. وقتی که وجود داشته باشد، همیشه یک بار بعد از اتمام حلقه forاجرا می شود، مگر اینکه یک دستور شکسته به آن بر بخورد .

به یاد داشته باشید که حلقه for..in برای هر توالی ای کار می کند .در اینجا، ما یک لیست از اعداد که توسط تابع توکار range تولید شده است را داریم، اما به طور کلی ما می توانیم از هر نوع از توالی های هر شیء ای استفاده کنیم! ما معنی این را در فصل های بعدی با جزئیات بررسی می کنیم .

تذکر برای برنامه نویسانC/C++/Java/C#

حلقه loop پایتون اساسا با حلقه for زبان های C/C++ متفاوت است. برنامه نویسان C# توجه خواهند کرد که حلقه for در Python شبیه حلقه foreach در C# است. برنامه نویسان Java توجه خواهند کرد که همان چیز شبیه for (int i : IntArray) در Java 1.5 است .

در C/C++ اگر می خواهید بنویسید for (int i = 0; i < 5; i++) ، آنگاه در Python تنها for i in range(0,5)را می نویسید. همان طوری که می توانید ببینید، حلقه loop ساده تر، گویا تر و مستعد خطای کمتری در Python است .

دستور شکسته

دستور break برای شکستن، بیرون از یک دستور حلقه ای به کار می رود. یعنی اجرای یک دستور حلقه زن را متوقف می کند، حتی اگر شرط حلقه False نباشد یا توالی اجزا به طور کامل تکرار شده باشد .یک تذکر مهم این است که اگر شما بیرون یک دستور for یا while بشکنید، هر حلقه بلاک else اجرا نمی شود .

استفاده از دستورbreak

مثال 6.4. استفاده از دستورbreak

```
#!/usr/bin/python
# Filename: break.py
while True:
```

```
        s = raw_input('Enter something : ')
        if s == 'quit':
                break
        print 'Length of the string is', len(s)
print 'Done'
```

خروجی

```
$ python break.py
Enter something : Programming is fun
Length of the string is 18
Enter something : When the work is done
Length of the string is 21
Enter something : if you wanna make your work also fun:
Length of the string is 37
Enter something :       use Python!
Length of the string is 12
Enter something : quit
Done
```

در این برنامه ما مکررا ورودی را از کاربر می گیریم و طول هر ورودی را هر بار چاپ می کنیم. ما یک شرط مخصوص را برای متوقف کردن برنامه با به وسیله بررسی کردن در نظر گرفته ایم، اگر ورودی 'quit' باشد. ما برنامه را به وسیله شکستن بیرون حلقه متوقف می کنیم و به انتهای برنامه می رسیم .طول رشته ورودی می تواند به وسیله تابع توکار len بررسی شود .به یاد داشته باشید که دستور break می تواند همراه حلقه for به خوبی مورد استفاده قرار گیرد .

G2's Poetic Python

ورودی که من اینجا استفاده کرده ام، یک شعر کوتاه است که نوشته ام و G2's Poetic Python نام دارد :

```
Programming is fun
When the work is done
if you wanna make your work also fun:
        use Python!
```

ترجمه

برنامه نویسی سرگرم کننده است هنگامی که کار انجام شده باشد اگر می خواهید کارتان را مفرح کنید :از Pythonاستفاده کنید !

دستورcontinue

دستور continue برای گفتن به Python برای رد کردن بقیه دستور در حلقه بلاک کنونی و ادامه از سرگیری بعدی حلقه به کار می رود .

استفاده از دستورcontinue

مثال 6.5. استفاده از دستور =continue

```
#!/usr/bin/python
# Filename: continue.py
while True:
        s = raw_input('Enter something : ')
        if s == 'quit':
                break
        if len(s) < 3:
                continue
        print 'Input is of sufficient length'
        # Do other kinds of processing here...
```

خروجی

```
$ python continue.py
 Enter something : a
Enter something : 12
Enter something : abc
Input is of sufficient length
Enter something : quit
```

این چگونه کار می کند؟

در این برنامه، ما خروجی را از کار می پذیریم، اما آن ها را تنها زمانی پردازش می کنیم که حداقل سه حرف داشته باشند. بنابراین، ما از تابع توکار lenبرای اندازه گیری طول استفاده می کنیم و اگر حرف ها کمتر از سه

باشند، ما بقیه دستور در بلاک را با استفاده از دستور continue رد می کنیم. در غیر این صورت، بقیه دستورات در حلقه اجرا می شوند و ما می توانیم هر نوع پردازشی را که می خواهیم اینجا انجام بدهیم، انجام دهیم .تذکر این که دستور continue با حلقه for به خوبی کار می کند .

خلاصه

ما چگونگی استفاده از سه دستور روند کنترل را دیدیم if - ، while و for همراه وابسته هایشان، دستورات break و continue. این ها تعدادی از قسمت هایی هستند که بیشترین استفاده را در Python دارند و از این رو، راحت شدن با آنان ضروری است .سپس ما چگونگی ساختن و استفاده کردن از توابع را می بینیم .

کتاب یک بایت از پایتون.فصل هفتم.توابع

فهرست مندرجات

فصل هفتم . توابع

معرفی

توابع بخش های قابل استفاده ی مجدد برنامه ها هستند . آنها به شما اجازه می دهند که نامی را بر روی یک بلاک از دستورات بگذارید و سپس آن بلوک را با استفاده از آن نام در هر کجایی از برنامه و به هر تعداد اجرا

کنید . این مطلب فراخوانی تابع نامیده می شود . ما تا کنون از بسیاری از توابع توکار مثل len و range استفاده کرده ایم .

توابع با استفاده از کلمه ی کلیدی def تعریف می شوند . در ادامه ی def، نام (شناسه) تابع می آید که با یک جفت پرانتز همراه است ، که در داخل جفت پرانتز ممکن است ، نام های متغیرهایی آورده شود ، در انتها این خط با کاراکتر کلون (:) ، تمام می شود . سپس در خطوط بعدی بلاکی از دستورات که بخشی از این تابع هستند ، می آیند . یک مثال به شما نشان خواهد داد که این کار بسیار ساده است :

تعریف تابع

مثال 7.1. تعریف کردن یک تابع

```
#!/usr/bin/python
# Filename: function1.py

def sayHello():
print 'Hello World!' # block belonging to the
function
# End of function

sayHello() # call the function
```

خروجی

```
$ python function1.py
Hello World!
```

نحوه ی عملکرد این مثال

ما تابعی را به نام sayHello با استفاده از نحوی که در بالا توضیح داده شد ، تعریف نمودیم . این تابع پارامتری نمی پذیرد ، از اینرو اعلان متغیری در جفت پرانتز تعریف تابع صورت نگرفته است . پارامترهای توابع ، ورودی

های توابع هستند ، بطوریکه ما می توانیم مقادیر متفاوتی را به آنها پاس دهیم و نتایجی منوط به آنها ، را دریافت کنیم .

پارامترهای توابع

یک تابع می تواند پارامترهایی بپذیرد که در واقع مقادیرشان را شما به تابع می دهید . از اینرو توابع می توانند این مقادیر را برای انجام کارهایی به کار گیرد . این پارامترها در واقع شبیه به متغیرها هستند ، به استثناء این که مقادیر این متغیرها هنگامیکه ما تابع را صدا می زنیم ، تعریف می شوند و در داخل خود تابع انتساب مقادیر صورت نگرفته است .

پارمترها در داخل یک جفت پرانتز در تعریف تابع مشخص می شوند ، و با کاراکتر (,) از هم تفکیک می شوند . و زمانیکه تابعی را صدا می زنیم ، ما مقادیر آن ها را به تابع می رسانیم . به اصطلاح به کاررفته در اینجا دقت کنید – نام هایی که در تعریف تابع ذکر می شوند ، پارامتر نامیده می شود . در حالیکه مقادیری که شما در حین فراخوانی تابع می آورید ، آرگومان نامیده می شود .

استفاده از پارمترهای تابع

مثال ۷.۲. بکارگیری پارامترهای تابع

```
#!/usr/bin/python
# Filename: func_param.py

def printMax(a, b):
if a > b:
print a, 'is maximum'
else:
print b, 'is maximum'

printMax(3, 4) # directly give literal values

x = 5
y = 7

printMax(x, y) # give variables as arguments
```

خروجی

```
$ python func_param.py
4 is maximum
7 is maximum
```

نحوه ی عملکرد این مثال

در اینجا ، ما تابعی به نام printMax را تعریف نمودیم که دو پارامتر به نام های aو b می پذیرد . در داخل تابع عدد بزرگ تر را با دستور ساده ی if .. else پیدا می کنیم و آن را چاپ می کنیم .

در اولین استفاده از تابع printMax ، ما اعداد)آرگومان ها) را بطور مستقیم به تابع رساندیم ، در دومین فراخوانی ، ما تابع را با استفاده از متغیرها صدا زدیم . دستور printMx(x,y) باعث می شود که مقدار آرگومان xبه پارامتر a و مقدار آرگومان y به پارامتر b منتسب شود . تابع printMax در دو حالت بدرستی کار می کند .

متغیرهای محلی

هنگامیکه متغبرهایی را در درون تعریف تابع اعلان می کنید ، آنها به هیچ وجه با متغیرهای هم نامی که درخارج از تابع هستند ، ارتباطی ندارند . بدین معنی که نام های متغیرها **محدود** به تابع هستند . این مطلب **حوزه ی متغیر** (scope of the variable)نامیده می شود . تمامی متغیرهایی که در حوزه ی یک بلوک هستند ، از محل تعریف نام شان اعلان می شوند .

استفاده از متغیرهای محلی

مثال 7.3. بکارگیری متغیرهای محلی

```
#!/usr/bin/python
# Filename: func_local.py

def func(x):
print 'x is', x
x = 2
```

```
print 'Changed local x to', x

x = 50
func(x)
print 'x is still', x
```

خروجی

```
$ python func_local.py
x is 50
Changed local x to 2
x is still 50
```

نحوه ی عملکرد این مثال

در تابع ، اولین باری که ما از مقدار x استفاده کردیم ، پایتون از مقدار پارامتری که درتابع اعلان شده استفاده می کند . سپس ، ما مقدار 2 را به x نسبت دادیم . نام x برای تابع ما محلی است . از اینرو وقتی ما مقدار xرا در تابع تغییر بدهیم ، متغیر x ایی که در بلاک اصلی برنامه تعریف شده است ، بدون تاثیر باقی می ماند .

در آخرین دستور print ، ما تائید می کنیم که مقدار متغیر x در بلاک اصلی برنامه ، بی تغییر مانده است .

استفاده از دستور global

در صورتی که بخواهید یک مقدار را به یک نام تعریف شده در خارج از تابع منتسب کنید . برای اینکار می بایست به پایتون بفهمانید که این نام محلی نیست ، بلکه سراسری است . ما این کار را با استفاده از دستور globalانجام می دهیم . این غیر ممکن است که یک مقدار را به یک متغیر تعریف شده درخارج از تابع بدون استفاده از دستور global نسبت داد .

شما می توانید از مقادیر متغیرهایی که در خارج از تایع تعریف شده اند (با فرض اینکه متغیرهای همنامی با آن ها درداخل تابع تعریف نشده اند (استفاده کنید . هر چند که این کار رواج ندارد و همچنین می بایست اجتناب

شود ، چرا که بدلیل اینکه تعریف متغیرها در جایی دیگر است ، باعث می شود که خوانایی و شفافیت برنامه از بین برود. اما استفاده از دستور global ، به طور وافری باعث شفافیت این مسئله می شود ، چرا که مشخص می کند که متغیر در بلوک بیرونی تعریف شده است .

مثال 7.4. استفاده از دستور global

```
#!/usr/bin/python
# Filename: func_global.py

def func():
global x

print 'x is', x
x = 2
print 'Changed global x to', x

x = 50
func()
print 'Value of x is', x
```

خروجی

```
$ python func_global.py
x is 50
Changed global x to 2
Value of x is 2
```

نحوه ی عملکرد این مثال

دستور global ، برای بیان اینکه xیک متغیر سراسری است ، به کار می رود. از اینرو هنگامیکه ما مقداری را در درون تابع به x نسبت می دهیم ، این تغییر زمانیکه در بلوک اصلی برنامه از x استفاده می کنیم ، نمایان می شود .

شما می توانید ، بیش از یک متغیر سراسری را با استفاده از یک دستور global ، مشخص کنید . برای مثال global x,y,z:

مقادیر پیش فرض آرگومان ها

برای برخی توابع ، ممکن است که شما بخواهید ، پارامترهایی اختیاری ایجاد کنید ، بطوریکه در صورتی که کاربر مقادیری را برای آن پارامترها ارسال نکرد ، از مقادیر پیش فرض خودشان استفاده کنند . این کار به کمک مقادیر پیش فرض آرگومان ها انجام می شود . شما می توانید مقادیر پیش فرض آرگومان ها ، برای پارامترها به این صورت مشخص می شود که بعد از نام پارامتر در تعریف تابع ، عملگر انتساب (=) قرار می گیرد و بدنبال آن مقدار پیش فرض می آید .

توجه کنید که مقدار پیش فرض آرگومان می بایست ثابت باشد . بصورت موشکافانه تر باید گفت که مقدار پیش فرض آرگومان می بایست تغییرناپذیر (immutable)باشد . این مطلب با جزئیات بیشتری در فصل های بعدی توضیح داده خواهد شد . فعلا فقط آن را بخاطر بسپارید .

استفاده از مقادیر پیش فرض آرگومان

مثال 7.5. بکارگیری مقادیر پیش فرض آرگومان

```
#!/usr/bin/python
# Filename: func_default.py

def say(message, times = 1):
print message * times

say('Hello')
say('World', 5)
```

خروجی

```
$ python func_default.py
Hello
WorldWorldWorldWorldWorld
```

نحوه ی عملکرد این مثال

تابع say ، برای چاپ یک رشته به دفعات درخواستی استفاده می شود . اگر ما مقداری را برای تعداد دفعات چاپ ارسال نکنیم ، بطور پیش فرض ، رشته تنها یک بار چاپ می شود . ما این کار را برای پارامتر times با تعیین مقدارپیش فرض آرگومانش بر روی 1 ، انجام می دهیم .

در اولین استفادها از تابع say ، ما تنها رشته را ارسال کردیم و درنتیجه ، تابع تنها یک بار آن را چاپ کرد . در دومین استفاده از say ، ما هم رشته و هم تعداد دفعات را برابر با 5 مشخص کردیم ، که در نتیجه باعث می شود ، رشته ما 5 مرتبه چاپ شود .

مهم

تنها ، آن پارامترهایی که در انتهای لیست پارامترها هستند ، می توانند دارای مقادیر پیش فرض آرگومان باشند ، یعنی در ترتیب پارامترهای اعلان شده در تعریف داده ، شما نمی توانید یک پارامتر با مقدار پیش فرض آرگومان را قبل از یک پارامتر بدون مقدار پیش فرض آرگومان بیاورید .

این به این دلیل است که مقادیر توسط موقعیت شان (position)به پارامترها منتسب می شوند . برای مثال ، def func(a,b=5)معتبر است ولی def func(a=5,b) نامعتبر است .

آرگومان های کلیدی

اگر شما توابعی داشته باشید که پارامترهای زیادی می پذیرد ، ولی شما می خواهید تنها بخشی از آن پارامترها را مشخص کنید ، می توانید به آن پارامترها توسط نام شان مقدار بدهید – این مطلب آرگومان های کلیدی نامیده می شود . ما از نام کلیدی (keyword) در عوض موقعیت (position) برای تعیین آرگومان های تابع استفاده می کنیم .

و دو مزیت وجود دارد – یک ، استفاده از تابع آسان تر است ، چرا که ما نیازی به نگرانی درباره ی ترتیب آرگومان ها نداریم . دو ، ما تنها به آن پارامترهایی که می خواهیم مقدار می دهیم ، مقادیر مابقی پارامترها از طریق آرگومان های پیش فرض ارسال می شود .

استفاده از آرگومان های کلیدی

مثال 7.6. بکارگیری آرگومان های کلیدی

```
#!/usr/bin/python
# Filename: func_key.py

def func(a, b=5, c=10):
print 'a is', a, 'and b is', b, 'and c is', c

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

خروجی

```
$ python func_key.py
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

نحوه ی عملکرد این مثال

تابع func ، دارای یک پارامتر بدون مقادیر پیش فرض آرگومان و دو پارامتر با مقادیر پیش فرض آرگومان است .

در اولین استفاده از این تابع ، بصورت func(3,7) ، پارامتر a مقدار 3 ، پارامتر b مقدار 5 و پارامتر c مقدار پیش فرض 10 را می گیرد .

در دومین استفاده از این تابع ، بصورت func(25,c=24) ، متغیر a مقدار 25 را براساس موقعیت آرگومان می گیرد . سپس ، پارامتر c مقدار 24 را براساس نام ، یعنی آرگومان های کلیدی می پذیرد . متغیر b هم مقدار پیش فرض 5 را می گیرد .

در سومین استفاده از این تابع ، بصورت func(c=50,a=100) ، ما بطورکامل از آرگومان های کلیدی برای تعیین مقادیر استفاده کردیم . توجه کنید که ما مقدار پارامتر c را قبل از a تعیین کردیم ، حتی با آنکه a در تعریف تابع قبل از c تعریف شده است .

دستور return

دستور return ، برای برگشتن ازیک تابع استفاده می شود . یعنی ، برای گریز از تابع به کار می رود . ما می توانیم بصورت اختیاری مقداری را نیز از تابع برگردانیم .

استفاده از دستور return

مثال 7.7. بکارگیری دستور return

```
#!/usr/bin/python
# Filename: func_return.py

def maximum(x, y):
if x > y:
return x
else:
return y

print maximum(2, 3)
```

خروجی

```
$ python func_return.py
3
```

نحوه ی عملکرد این مثال

تابع maximum ، ماکزیمم پارامترها را برمی گرداند ، در این مورد ، ماکزیمم اعدادی که به تابع ارسال شده است را برمی گرداند . این تابع از دستور if .. else برای یافتن عدد بزرگتر و سپس برگرداندن آن استفاده می کند .

توجه کنید که دستور return ، بدون مقدار ، برابر با دستور return None است None . یک نوع خاص در پاتون است که نمایانگر هیچ است . برای مثال ، برای تعیین اینکه یک متغیر دارای مقداری نیست اگر آن متغیر دارای مقدار None باشد ، به کار می رود .

هر تابعی بطور تلویحی شامل دستور return None در پایان خود می باشد ، مگر اینکه شما دستور return خود را بنویسید . شما می توانید این مسئله را با اجرای دستور print someFunction() ، بطوریکه تابع someFunctionاز دستور return استفاده نکرده باشد ، متوجه شوید ، به این صورت :

```
def someFunction():
pass
```

دستور *pass*در پایتون برای مشخص کردن یک بلوک خالی از دستورات به کار می رود.

DocStrings

پایتون دارای یک ویژگی بسیارعالی به نام documentation strings است ، که با نام کوتاه docstrings به آن اشاره می شود . رشته های مستندساز (DocStrings) ابزار مهمی هستند که شما می بایست از آن برای مستندسازی بهتر برنامه های تان برای درک بهتر برنامه تان کمک بگیرید ، با کمال تعجب باید گفت ، حتی ما می توانیم docstring یک تابع را در حین اجرای برنامه نیز بدست آوریم !

استفاده از DocStrings

مثال 7.8. بکارگیری DocStrings

```
#!/usr/bin/python
# Filename: func_doc.py

def printMax(x, y):
'''Prints the maximum of two numbers.

The two values must be integers.'''
```

```
x = int(x) # convert to integers, if possible
y = int(y)

if x > y:
print x, 'is maximum'
else:
print y, 'is maximum'

printMax(3, 5)
print printMax.__doc__
```

خروجی

```
$ python func_doc.py
5 is maximum
Prints the maximum of two numbers.

        The two values must be integers.
```

نحوه ی عملکرد این مثال

رشته ایی که در اولین خط منطقی تابع آمده است ، docstringآن تابع است . توجه کنید که DocStrings ها بروی ماژول ها و کلاس ها (که در فصل های آتی درباره ی آن ها خواهیم آموخت) هم اعمال می شوند.

رسم است که docstring ها در چندین خط نوشته شوند بطوریکه در خط اول اولین کلمه با حرف بزرگ شروع می شود و آخرین خط با نقطه پایان می یابد . همچنین دومین خط یک خالی است و هرگونه توضیحات جزئیات مورد نیازاز خط سوم شروع می شود .به شدت به شما توصیه می شود که این رسم را برای تمامی docstringهای توابع با اهمیت تان رعایت کنید .

شما می توانید به رشته مستندساز تابع printMax با استفاده از ویژگی __doc__ (به دوبار تکرار پی درپی زیرخط توجه کنید) از تابع دسترسی پیدا کنید . دقت کنید که پایتون به هرچیزی به چشم یک شیء نگاه می کند و این شامل توابع هم می شود . (درباره ی اشیاء در فصل کلاس ها خواهیم آموخت).

اگر تا به حال از تابع help() در پایتون استفاده کرده باشید ، می بایست نحوه ی استفاده از docstring ها را دیده باشید ! کاری که این تابع انجام می دهد ، اینست که ویژگی __doc__ از آن تابع را واکشی می کند و سپس آن را برای شما در یک قالب منظم نمایش می دهد . شما به این صورت help(printMax)، می توانید این کار را برای تابع تعریف شده در این مثال امتحان کنید .یادتان باشد برای خروج از help ، کاراکتر q را تایپ کنید .

به این طریق ، ابزارهای خودکار می توانند مستندات را از برنامه ی شما استخراج کنند . بنابراین ، من شدیدا به شما پیشنهاد می کنم که از رشته های مستند ساز برای توابع با اهمیتی که می نویسید ، استفاده کنید . دستور pydocکه در توزیع پایتون شما وجود دارد ، بطور مشابه مثل دستور help() از docstringها استفاده می کند .

خلاصه

شما جنبه های مختلفی از توابع را ملاحظه نمودید ، اما توجه کنید که ما در اینجا تمامی جنبه ها را پوشش ندادیم . به هرحال ، بیشتر آنچه که شما درباره ی توابع پایتون بطور کلی با آن روبه رو می شوید را مطرح کردیم .

در فصل بعدی ، می خواهیم نحوه ی ایجاد ماژول های پایتون را بیاموزیم .

کتاب یک بایت از پایتون.فصل هشتم.ماژول ها

فهرست مندرجات

ماژول ها

معرفی

شما ملاحظه نمودید که به چه صورت می توانیم در برنامه های مان با یکبار تعریف توابع ، از کدش استفاده ی مجدد ببریم . حال اگر بخواهید از شماری از توابع در دیگر برنامه هایی که می نویسید استفاده مجدد ببرید ، چطور ؟ همان طور که حدس زده اید ، پاسخ ماژول است . یک ماژول بطور کلی فایلی ست که شامل توابع و متغیرهایی ست که شما تعریف کرده اید . برای استفاده ی مجدد از ماژول در دیگر برنامه ها ، می بایست نام فایل ماژول دارای پسوند py. باشد .

یک ماژول می تواند بوسیله ی دیگر برنامه ها برای استفاده ی عملیاتی وارد (import) شود . و به همین ترتیب است که ما می توانیم از کتابخانه ی استاندارد پایتون به خوبی استفاده کنیم . در ادامه نحوه ی استفاده از ماژول های کتابخانه ی استاندارد پایتون را خواهیم دید .

استفاده از ماژول sys

مثال 8.1. استفاده از ماژول sys

```
#!/usr/bin/python
# Filename: using_sys.py

import sys

print 'The command line arguments are:'
for i in sys.argv:
print i

print '\n\nThe PYTHONPATH is', sys.path, '\n'
```

خروجی

```
$ python using_sys.py we are arguments
The command line arguments are:
using_sys.py
we
are
arguments

The PYTHONPATH is ['/home/swaroop/byte/code',
'/usr/lib/python23.zip',
'/usr/lib/python2.3', '/usr/lib/python2.3/plat-linux2',
'/usr/lib/python2.3/lib-tk', '/usr/lib/python2.3/lib-
dynload',
'/usr/lib/python2.3/site-packages',
'/usr/lib/python2.3/site-packages/gtk-2.0']
```

نحوه ی عملکرد این مثال

در ابتدا ، ما ماژول sys را با استفاده از دستور import وارد کردیم . بطورکلی این دستور برای ما به اینصورت تفسیر می شود که به پایتون می گوئیم که می خواهیم از این ماژول استفاده کنیم . ماژول sys شامل عملیاتی مرتبط با مفسر پایتون و محیط آن است .

زمانی که پایتون دستور import sys را اجرا می کند ، در یکی از دایرکتوری های لیست شده در متغیر sys.pathخودش ، به دنبال ماژول sys.py می گردد .اگر که فایل را در آنجا پیدا کند ، سپس دستورات بلوک اصلی این ماژول اجرا می شود و سپس این ماژول برای استفاده ، دردسترس ما قرار می گیرد .توجه کنید که کارهای اولیه ی ماژول (initialization) تنها اولین زمانی که ما یک ماژول را وارد می کنیم انجام می شود . همچنین ، 'sys'مختصری برای کلمه ی 'system' است .

متغیر argv در ماژول sys از طریق علامت یک نقطه برای استفاده مورد اشاره قرار می گیرد . یکی از مزیت های این شیوه ، اینست که این نام با دیگر متغیرهای argv در برنامه برخورد و تضاد پیدا نمی کند . همچنین ، به روشنی مشخص می کند که این نام بخشی از ماژول sys است .

متغیر sys.argvیک لیست از رشته هاست (لیست ها بطور جزئی تری در بخش بعدی توضیح داده می شوند(.بطور خاص ، sys.argvشامل لیستی از *آرگومان های خط فرمان* است ، بدین معنی که حاوی آرگومان هایی ست که از طریق خط فرمان به برنامه پاس داده شده اند .

اگر شما از یک IDE برای نوشتن و اجرای این برنامه ها استفاده می کنید ، از منوهای آن به دنبال راهی برای مشخص کردن آرگومان های خط فرمان به برنامه تان بگردید .

در اینجا ، وقتی که دستور python using_sys.py we are arguments را اجرا می کنیم ، در واقع ماژول using_sys.py را با برنامه ی python اجرا می کنیم و مابقی چیزهایی که بدنبال آن نوشتیم به برنامه پاس داده می شود . پایتون آنها را برای ما در متغیر sys.argv ذخیره می کند .

به خاطر بسپارید که نام اسکریپتی که اجرا می شود ، همیشه بعنوان اولین آرگومان در لیست sys.argv قرار می گیرد . از اینرو در این مثال ، ما رشته ی 'using_sys.py' را در sys.argv[0]، رشته ی 'we' را در sys.argv[1]، رشته ی 'are' را در sys.argv[2] و رشته ی 'arguments' را در sys.argv[3] خواهیم داشت . دقت کنید که ایندکس لیست در پایتون از 0 شروع می شود و نه از 1 .

sys.pathشامل لیستی از نام های دایرکتوری هایی ست که ماژول ها از آنجا وارد (import) می شوند . مشاهده می کنید که اولین رشته در sys.path خالی ست – این رشته ی خالی مشخص می کند که دایرکتوری جاری نیز بخشی از sys.pathاست ،همین طور متغیر محیطی PYTHONPATH هم بخشی از sys.pathاست .این بدین معنی ست که مستقیما می توانید ماژول هایی که در دایرکتوری جاری واقع شده اند را وارد کنید . در غیر اینصورت برای وارد کردن ماژول تان ، می بایست آن را در یکی از دایرکتوری هایی که در sys.pathفهرست شده اند ، قرار بدهید .

فایل های Byte-compiled ، .pyc

وارد کردن یک ماژول تا حدی امر پرهزینه ایی ست ، از اینرو پایتون ترفندهایی را برای سرعت بخشی به این قضیه اعمال می کند . یک راه آن ایجاد فایل های کامپایل شده ی بایتی (byte-compiled) با پسوند .pyc است که مربوط به بخش میانی ایی است که پایتون برنامه ها را به آن تبدیل می کند ، (بخش معرفی پایتون چگونه کار می کند ؟ را به یاد بیاورید) . و این عمل باعث سرعت بخشی می شود چرا که با این کار بخشی از فرآیند مورد نیاز در واردکردن ماژول انجام می شود . همچنین این فایل های کامپایل شده ی بایتی مستقل از platfromهستند . خب اکنون می دانید که فایل های .pyc برای چه منظوری استفاده می شوند .

دستور from … import

اگر بخواهید مستقیما متغیر argv در برنامه تان وارد کنید (برای پرهیز از این که در هر نوبت .sys را برای آن تایپ کنید) ، می توانید از دستور from sys import argvاستفاده کنید . اگر می خواهید تمامی نام های به کار رفته در ماژول sys را وارد کنید ، می توانید از دستور from sys import * استفاده کنید . این روش برای هر ماژولی کار می کند . بطور کلی ، از دستور from … importپرهیز کنید و به جای آن از دستور importاستفاده کنید ، چرا که به این طریق هم برنامه تان خوانایی بالایی پیدا می کند و هم از برخورد و تضاد نام ها اجتناب می شود .

ویژگی __name__ ماژول

هر ماژول دارای یک نام است و از اینرو دستوراتی که در یک ماژول هستند می توانند از طریق آن ، نام ماژول شان را پیدا کنند . این مطلب مخصوصا در شرایطی خاص باعث راحتی کار می شود . همین طور که قبلا هم اشاره شد ، وقتی یک ماژول برای اولین بار وارد می شود ، بلوک اصلی آن ماژول اجرا می شود . حال اگر بخواهیم که این بلوک تنها وقتی که برنامه توسط خودش به کار گرفته می شود و نه زمانی که توسط دیگر ماژول ها وارد می شود ، اجرا شود ، چه کاری باید انجام بدهیم ؟ این کار می تواند با استفاده از ویژگی __name__از ماژول بدست آید .

استفاده از ویژگی __name__ ماژول

مثال 8.2. به کارگیری ویژگی __name__ ماژول

```
#!/usr/bin/python
# Filename: using_name.py

if __name__ == '__main__':
print 'This program is being run by itself'
else:
print 'I am being imported from another module'
```

خروجی

```
$ python using_name.py
This program is being run by itself

$ python
>>> import using_name
I am being imported from another module
>>>
```

نحوه ی عملکرد این مثال

هر ماژول پایتون ، دارای ویژگی __name__ تعریف شده ی خودش است ، که اگر برابر با '__main__' باشد ، بر این موضوع دلالت دارد که ماژول به تنهایی توسط کاربر اجرا شده و ما می توانیم با توجه به این مطلب اعمال مناسبی را انجام بدهیم .

ساخت ماژول های دلخواه تان

ایجاد ماژول های دلخواه بسیار راحت است، شما این کار را بارها انجام داده اید ! هر برنامه ی پایتون یک ماژول نیز هست . شما تنها می بایست مطمئن باشید که آن برنامه دارای پسوند .py است . مثال زیر این مطلب را روشن خواهد کرد .

ایجاد ماژول های دلخواه

مثال 8.3. چگونگی ایجاد ماژول های دلخواه

```
#!/usr/bin/python
# Filename: mymodule.py

def sayhi():
print 'Hi, this is mymodule speaking.'

version = '0.1'

# End of mymodule.py
```

مثال بالا یک ماژول نمونه بود . همانطور که ملاحظه می نمائید ، این ماژول هیچ چیز خاص بخصوصی برای مقایسه با برنامه های پایتون معمولی ندارد . ما در ادامه خواهیم آموخت که به چه صورت این ماژول را در دیگر برنامه های مان وارد کنیم .

به یاد بیاورید که این ماژول می بایست در دایرکتوری یکسانی از برنامه ایی که در آن import می شود قرار گرفته باشد ، و یا می بایست این ماژول در یکی از دایرکتوری های لیست شده در sys.path قرار بگیرد .

```
#!/usr/bin/python
# Filename: mymodule_demo.py

import mymodule

mymodule.sayhi()
print 'Version', mymodule.version
```

خروجی

```
$ python mymodule_demo.py
Hi, this is mymodule speaking.
Version 0.1
```

نحوه ی عماکرد این مثال

دقت کنید که ما از همان علامت نقطه برای دسترسی به اعضای ماژول استفاده کردیم . پایتون به خوبی از همان علامت استفاده مجدد می برد تا 'Pythonic'را به نمایش بگذارد ، بطوری که ما مجبورنباشیم روش های جدیدی را برای انجام کارها یاد بگیریم .

from … import

در این جا مثالی هست که نحو from … import را به کار می گیرد .

```
#!/usr/bin/python
# Filename: mymodule_demo2.py

from mymodule import sayhi, version
# Alternative:
# from mymodule import *

sayhi()
print 'Version', version
```

خروجی برنامه ی mymodule_demo2.py همانند خروجی برنامه ی mymodule_demo.py است .

تابع dir()

شما می توانید از تابع توکار dirبرای فهرست گرفتن identifierهایی که در یک ماژول تعریف شده اند ، استفاده کنید identifer . ها شامل توابع ، کلاس ها و متغیرهای تعریف شده در ماژول هستند .

هنگامی که نام ماژولی را به تابع توکار dir می دهید ، این تابع لیستی از نام های تعریف شده در آن ماژول را برمی گرداند . هنگامی که آرگومانی را به این تابع نفرستید ، آن لیستی از نام های تعریف شده در ماژول فعلی را بر می گرداند .

استفاده از تابع dir

مثال 8.4. بکارگیری تابع dir

```
$ python
>>> import sys
>>> dir(sys) # get list of attributes for sys module
```

```
['__displayhook__', '__doc__', '__excepthook__',
'__name__', '__stderr__',
'__stdin__', '__stdout__', '_getframe', 'api_version',
'argv',
'builtin_module_names', 'byteorder', 'call_tracing',
'callstats',
'copyright', 'displayhook', 'exc_clear', 'exc_info',
'exc_type',
'excepthook', 'exec_prefix', 'executable', 'exit',
'getcheckinterval',
'getdefaultencoding', 'getdlopenflags',
'getfilesystemencoding',
'getrecursionlimit', 'getrefcount', 'hexversion',
'maxint', 'maxunicode',
'meta_path','modules', 'path', 'path_hooks',
'path_importer_cache',
'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval',
'setdlopenflags',
'setprofile', 'setrecursionlimit', 'settrace',
'stderr', 'stdin', 'stdout',
'version', 'version_info', 'warnoptions']
>>> dir() # get list of attributes for current module
['__builtins__', '__doc__', '__name__', 'sys']
>>>
>>> a = 5 # create a new variable 'a'
>>> dir()
['__builtins__', '__doc__', '__name__', 'a', 'sys']
>>>
>>> del a # delete/remove a name
>>>
>>> dir()
['__builtins__', '__doc__', '__name__', 'sys']
>>>
```

نحوه ی عملکرد این مثال

در ابتدا ، ما از تابع توکار dir برای ماژول sys وارد شده ، استفاده نمودیم ، که به این ترتیب می توانیم لیستی بزرگی از ویژگی هایی که در این ماژول هست را ملاحظه کنیم . سپس ، ما از تابع توکار dir ، بدون آرگومان

استفاده کردیم ، که بطور پیش فرض لیستی از ویژگی های مربوط به ماژول فعلی را بر می گرداند . توجه کنید که لیست ماژول های وارد شده قبلی هم در بخشی از این لیست است . برای این که dir را در عمل مشاهده کنیم ، ما یک متغیر جدید به نام a تعریف نمودیم و یک مقدار به آن نسبت دادیم و سپس خروجی dir را بررسی کردیم و درنتیجه متوجه مقداراضافه شده با همان نام a در لیست شدیم .سپس متغیر/ویژگی a از ماژول فعلی را با دستور del حذف می کنیم و این تغییر دوباره در خروجی تابع dir منعکس می شود .

یک نکته برای del اینست که این دستور برای حذف نام/متغیر به کار می رود و بعد از آن که این دستور اجرا می شود ، در این مورد del a ، نمی توانید دیگر به متغیر a دسترسی پیدا کنید – بطوری که گویا هیچ وقت پیش از این وجود نداشته است .

خلاصه

ماژول ها مفید هستند ، چرا که آن ها سرویس ها و عملیاتی را ارائه می دهند ، که می توانید دردیگر برنامه ها از آن ها استفاده ی مجدد ببرید . کتابخانه ی استاندارد پایتون که همراه با پایتون ارائه شده است ، مثالی از مجموعه ایی از این ماژول هاست . ما به خوبی ملاحظه نمودیم که چطور از این ماژول ها استفاده کنیم و چطور ماژول های خودمان را ایجاد کنیم .

در فصل بعدی ، درباره ی برخی دیگر از جنبه های جالب توجه پایتون که ساختمان داده ها نام دارد ، خواهیم آموخت .

کتاب یک بایت از پایتون.فصل نهم.ساختمان های داده

فهرست مندرجات

معرفی

ساختمان های داده بطور کلی عبارتست از ساختارهایی که می تواند داده هایی را با همدیگر و در کنار هم نگه دارد . بعبارتی دیگر ، برای ذخیره ی مجموعه ایی از داده های وابسته به کار می روند.

سه ساختمان داده ی توکار در پایتون هست ، که عبارتند از لیست (list)، چندتایی (tuple) و دیکشنری (dictionary)ما در ادامه نحوه ی استفاده از هر یک از آن ها و اینکه چطور زندگی را برای ما راحت تر می سازند را خواهیم دید .

لیست

یک لیست ، ساختمان داده ایی ست که مجموعه ایی ترتیبی از آیتم ها را نگه می دارد ، بدین معنی که می توانید یک دنباله از آیتم ها را در یک لیست ذخیره کنید . این مطلب می تواند به راحتی با در نظر گرفتن یک لیست خرید تصور شود ، چرا که آن جا هم لیستی از آیتم ها را برای خریدن داریم ، با این تفاوت که احتمالا در لیست خرید هر آیتم را در خطی مجزا داریم ، در حالی که در پایتون ، بین آیتم ها ویرگول می گذاریم.

لیست آیتم ها می بایست در یک جفت براکت قرار بگیرند ، تا پایتون بفهمد که شما یک لیست را مشخص کرده اید . یکبار که لیستی را تعریف می کنید ، می توانید آیتم هایی را در لیست اضافه ، حذف و یا جستجو نمائید ، و چون می توانیم آیتم هایی را به لیست اضافه و یا حذف کنیم ، از اینرو می گوئیم که لیست یک نوع داده ی تغییر پذیر (mutable data type) است ، یعنی این نوع می تواند دستکاری شود .

تعریف مختصری از اشیاء و کلاس

هر چند درادامه ی کتاب به بحث کلی درباره ی کلاس ها و اشیاء ها می پردازیم ، اما توضیح مختصری در اینجا برای درک بهتر لیست مناسب هست ، و بحث کلی این موضوع را در فصل خودش خواهیم داشت.

یک لیست مثالی از استفاده از اشیاء و کلاس است . وقتی که شما متغیر i را بکار می برید و یک مقداری را به آن نسبت می دهید (فرض کنید که عدد صحیح 5 را به آن نسبت داده اید) ، می توانید به این صورت فکر

کنید که یک شیء (نمونه) به نام i از کلاس (نوع int (ایجاد نموده اید . برای درک بهتر این مطلب خروجی help(int)را ملاحظه نمائید .

یک کلاس می تواند دارای **متدهایی** باشد . بدین معنی که ، توابعی که تعریف شده اند ، تنها در مورد آن کلاس به کار گرفته می شوند . و می توانید ازعملیات آن قسمت تنها هنگامی که یک شیء از آن کلاس داشته باشید ، استفاده کنید . برای مثال ، پایتون متد append را برای کلاس list ارائه نموده است که به شما اجازه می دهد آیتمی را در انتهای لیست اضافه کنید . برای مثال ، دستور mylist.append('an item')، آن رشته را به انتهای لیست mylistاضافه می کند . توجه کنید که از علامت نقطه برای دسترسی به متدهای یک شیء استفاده می کنید .

یک کلاس می تواند دارای **فیلدهایی** باشد که چیزی بیش از یک متغیر نیستند ولی متغیرهایی هستند که تنها در مورد کلاس به کار گرفته می شوند . می توانید از این متغیرها/نام ها هنگامی که شیء ایی از آن کلاس دارید ، استفاده کنید . فیلدها هم همچنین از طریق علامت نقطه دردسترس قرارمی گیرند ، برای مثال ، mylist.field .

استفاده از لیست ها

مثال 9.1. بکارگیری لیست ها

```
#!/usr/bin/python
# Filename: using_list.py

# This is my shopping list
shoplist = ['apple', 'mango', 'carrot', 'banana']

print 'I have', len(shoplist), 'items to purchase.'

print 'These items are:', # Notice the comma at end of
the line
for item in shoplist:
print item,

print '\nI also have to buy rice.'
shoplist.append('rice')
print 'My shopping list is now', shoplist
```

```
print 'I will sort my list now'
shoplist.sort()
print 'Sorted shopping list is', shoplist

print 'The first item I will buy is', shoplist[0]
olditem = shoplist[0]
del shoplist[0]
print 'I bought the', olditem
print 'My shopping list is now', shoplist
```

خروجی

```
$ python using_list.py
I have 4 items to purchase.
These items are: apple mango carrot banana
I also have to buy rice.
My shopping list is now ['apple', 'mango', 'carrot',
'banana', 'rice']
I will sort my list now
Sorted shopping list is ['apple', 'banana', 'carrot',
'mango', 'rice']
The first item I will buy is apple
I bought the apple
My shopping list is now ['banana', 'carrot', 'mango',
'rice']
```

نحوه ی عملکرد این مثال

متغیر shoplist یک لیست خرید برای کسی ست که به فروشگاه می رود . در shoplist، ما فقط رشته هایی از نام های اقلام خرید را ذخیره کرده ایم ، اما یادتان باشد که می توانید هر نوعی از اشیاء را ، شامل اعداد و حتی دیگر لیست ها را به این لیست اضافه کنید .

ما همچنین از عبارت for .. in در حلقه برای حرکت بروی آیتم های لیست استفاده نمودیم . پس اکنون ، می بایست تشخیص داده باشید که همچین یک لیست یک دنباله (sequence)هم هست . ویژگی های دنباله ها در بخش بعدی بحث خواهد شد .

توجه کنید که ما از یک علامت کاما (,) در انتهای دستور print برای جلوگیری از چاپ خودکار کاراکتر line breakبعد از هر دستور print استفاده نمودیم . این روش کمی ناخوشایند هست ولی ساده ست و خواسته ی ما را برآورده می کند .

در ادامه ، آیتم هایی را به لیست با استفاده از دستور append از شیء لیست که قبلا بحث شد ، اضافه می کنیم ، سپس ، برای اطمینان از اینکه آیا واقعا آیتم به لیست اضافه شده است ، محتوای لیست را به سادگی با فرستادن لیست به دستور print چاپ می کنیم ، که در نتیجه به طرز مناسبی لیست را برای ما چاپ می کند.

سپس ما لیست را با متد sort از لیست مرتب می کنیم . توجه کنید که این متد بروی لیست خودش تاثیر می گذارد و لیست تغییر یافته ی جدیدی برنمی گرداند – و این برخلاف روشی ست که رشته ها کار می کنند . به همین دلیل است که ما می گوئیم که لیست ها تغییر پذیرهستند و رشته ها تغییرناپذیرند .

سپس وقتی که خرید یک آیتم را در فروشگاه به اتمام رساندیم ، می خواهیم که آن را از لیست حذف کنیم . که این کار را با استفاده از دستور del بدست می آوریم . در اینجا ما اشاره کردیم که آیتمی را از لیست می خواهیم حذف کنیم و دستور del آن را برای ما از لیست حذف می کند . و چون می خواهیم که اولین آیتم از لیست را حذف کنیم ، از اینرو از دستور del shoplist[0] ، استفاده می کنیم . (یادتان باشد که پایتون شمارش را از 0 شروع می کند .)

اگرمی خواهید تمامی متدهای تعریف شده در شیء لیست را بدانید ، خروجی دستور help(list) را برای جزئیات بیشتر ملاحظه نمائید .

چندتایی (Tuple)

چندتایی ها مشابه لیست ها هستند ، با این تفاوت که مانند رشته ها **تغییرناپذیرند** ، بدین معنی که نمی توانید چندتایی ها را تغییر بدهید ، چندتایی ها با استفاده از آیتم هایی که با کاما (,) از هم جدا شده و در درون یک جفت پرانتز احاطه شده اند ، تعریف می شوند . چندتایی ها معمولا در مواردی به کار گرفته می شوند که یک دستور یا یک تابع تعریف شده توسط کاربر با مجموعه ایی از مقادیر بدون آنکه آسیبی به محتویات آنها وارد بشود بتواند کار کند ، بدین معنی که چندتایی که استفاده شده است ، مقادیرش تغییر نخواهند کرد .

استفاده از چندتایی ها

مثال 9.2. بکارگیری چندتایی ها

```
#!/usr/bin/python
# Filename: using_tuple.py

zoo = ('wolf', 'elephant', 'penguin')
print 'Number of animals in the zoo is', len(zoo)

new_zoo = ('monkey', 'dolphin', zoo)
print 'Number of animals in the new zoo is',
len(new_zoo)
print 'All animals in new zoo are', new_zoo
print 'Animals brought from old zoo are', new_zoo[2]
print 'Last animal brought from old zoo is',
new_zoo[2][2]
```

خروجی

```
$ python using_tuple.py
Number of animals in the zoo is 3
Number of animals in the new zoo is 3
All animals in new zoo are ('monkey', 'dolphin',
('wolf', 'elephant', 'penguin'))
Animals brought from old zoo are ('wolf', 'elephant',
'penguin')
Last animal brought from old zoo is penguin
```

نحوی عملکرد این مثال

متغیر zoo به یک چندتایی از آیتم ها اشاره می کند ، ملاحظه می کنید که از تابع len برای گرفتن طول چندتایی استفاده شده است . این مسئله همچنین به خوبی نشان می دهد که یک چندتایی ، یک دنباله هم هست .

سپس ما حیوانات باغ وحش قبلی را به داخل باغ وحش جدیدی انتقال دادیم ، چرا که باغ وحش قبلی بسته شده است ! بنابراین چندتایی new_zoo حاوی هم تعدادی حیوان هست که در خودش بوده است و هم حاوی حیواناتی ست که از باغ وحش قبلی آمده اند . خب به واقعیت برگردیم ، توجه کنید که یک چندتایی که در داخل خودش یک چندتایی دیگر را نگه می دارد ، هویت چندتایی بودنش را از دست نمی دهد .

می توانیم به آیتم های داخل یک چندتایی با مشخص کردن موقعیت آیتم در داخل یک جفت براکت ([]) ، مشابه به کاری که برای لیست ها انجام می دهیم ، دسترسی پیدا کنیم . پس ما به سومین آیتم در چندتایی new_zooبا مشخص کردن new_zoo[2]دسترسی پیدا می کنیم و به سومین آیتم از سومین آیتم در چندتایی new_zooبا مشخص کردن new_zoo[2][2] دسترسی پیدا می کنیم . تنها اگر که یکبارشیوه ی آن را درک کنید ، این کار برای تان بسیار آسان خواهد شد .

چندتایی با صفر و یا یک آیتم

یک چندتایی خالی با یک جفت پرانتز همچون my_empty=() ساخته می شود . ولی ایجاد یک چندتایی با تک آیتم کار راحتی نیست . شما می بایست آن را با استفاده از یک کاما که به دنبال اولین (و تنها) آیتم چندتایی می آید ، مشخص کنید ، تا از این طریق پایتون بتواند در یک عبارت ، بین یک چندتایی و یک چفت پرانتز که اطراف یک شیء قرار می گیرد ، تفاوت قائل بشود ، یعنی می بایست یک چندتایی با تک آیتم را اگر می خواهید به فرض شامل آیتم 2 باشد ، به صورت singleton = (2,) مشخص کنید .

نکته ایی برای برنامه نویسان Perl

یک لیست که در داخلش یک لیست دیگرهست ، هویت لیست بودن خودش را از دست نمی دهد ، بدین معنی که لیست ها در پایتون همچون لیست های پرل مسطح (flattened)نیستند ، و همین طور در مورد چندتایی هایی که در داخل شان چندتایی های دیگری ست ، یا یک چندتایی که در داخلش لیست است ، یا یک لیستی که در داخلش چندتایی ست و غیره . تا آنجا که به پایتون مرتبط است ، آنها تنها اشیاء ایی هستند که با استفاده از اشیاء ی دیگر ذخیره می شوند . تمام مطلب همین هست .

کاربرد چندتایی ها در دستور print

یکی از بیشترین استفاده ی چندتایی ها در دستور print است . در اینجا مثالی آورده شده است .

مثال 9.3. خروجی با استفاده از چندتایی ها

```
#!/usr/bin/python
# Filename: print_tuple.py

age = 22
name = 'Swaroop'

print '%s is %d years old' % (name, age)
print 'Why is %s playing with that python?' % name
```

نحوه ی عملکرد این مثال

دستور print ، می تواند رشته ایی را بگیرد که این رشته از مشخصه هایی معین که علامت % به دنبال شان هست ، و بدنبال آن یک چندتایی از آیتم هایی که با آن مشخصه ها تطابق دارند تشکیل شده باشد. این مشخصه ها برای قالب بندی خروجی در یک روش معین به کار گرفته می شوند . این مشخصه ها می تواند مثل %s برای رشته ها و %d برای اعداد صحیح باشد . چندتایی که بدنبال آن ها می آید ، می بایست دارای آیتم هایی مرتبط با این مشخصه ها و در یک ترتیب یکسان با آن ها باشد .

روشن هست که اولین کاربرد آن ، در جایی ست که ما از اولین %s استفاده کردیم و این به متغیر name ایی مربوط هست که اولین آیتم در چندتایی است و دومین مشخصه %d می باشد که مربوط به age ایی ست که دومین آیتم در چندتایی ست .

کاری که پایتون در اینجا انجام می دهد ، اینست که هر آیتم در چندتایی را به رشته ایی تبدیل می کند و آن مقدار رشته را با مکان مربوطه اش جایگزین می کند . بنابراین %s با مقدار متغیر name جایگزین می شود و الی آخر .

اینگونه استفاده از دستور print ، نوشتن خروجی را به شدت آسان می سازد و از دستکاری بیشتررشته که برای رسیدن به همچین کاری انجام می شود جلوگیری می کند. همچین از استفاده از کاما را که تاکنون استفاده می کردیم جلوگیری می کند .

بیشتر اوقات ، شما کافیست که از مشخصه %s ، استفاده کنید و به پایتون اجازه بدهید که به جای شما مراقب باشد . این قضیه حتی برای اعداد هم کار می کند . هرچند ، شما ممکن است بخواهید مشخصه های درستی را قرار بدهید که در اینصورت یک مرحله ی چک کردن را به برنامه تان اضافه می شود .

در دستور دوم print ، ما از یک مشخصه مجرد که به دنبال علامت % آمده و به دنبالش یک آیتم تنها هست استفاده کردیم – در این حالت دیگرجفت علامت پرانتز نیامده است و این حالت فقط برای مواردی که یک مشخصه تنها در رشته است به کار می رود .

دیکشنری

یک دیکشنری مثل یک کتابچه –آدرس است . که شما می توانید آدرس یا جزئیات تماس یک شخص را با دانستن فقط نامش پیدا نمائید . بدین معنی که ، ما **کلیدهایی**)نام) را به **مقادیری)**جزئیات (مرتبط و وابسته می سازیم . توجه کنید که کلیدها می بایست یکه باشند ، چراکه در غیراینصورت برای مثال شما با داشتن دو شخص با نام های یکسان نمی توانید اطلاعات درستی را پیدا کنید .

توجه کنید که تنها می توانید از اشیاء تغییرناپذیر برای کلیدهای یک دیکشنری استفاده کنید (مثل رشته ها) ، ولی می توانید هم از اشیاء تغییرپذیر وهم از اشیاء تغییرناپذیربرای مقادیر دیکشنری استفاده کنید . این مطلب بطور کلی به این صورت تفسیرمی شود که می بایست تنها از اشیاء ی ساده برای کلیدها استفاده شود .

جفت های کلید و مقدار در یک دیکشنری با استفاده از نشان d = {key1: value1 , key2 : value2} معین می شوند . دقت کنید که جفت های کلید/مقدار با علامت : از هم جدا می شوند و خود جفت ها با علامت , از هم جدا می شوند و همگی در داخل یک جفت براکت {} قرار می گیرند .

دیکشنری هایی که استفاده خواهید کرد ، نمونه هایی/اشیایی از کلاس dict هستند .

استفاده از دیکشنری

مثال 9.4. بکار گیری دیکشنری

```
#!/usr/bin/python
# Filename: using_dict.py

# 'ab' is short for 'a'ddress'b'ook
```

```
'Swaroop'   :            ab = {
'swaroopch@byteofpython.info',
'Larry'     : 'larry@wall.org',
'Matsumoto' : 'matz@ruby-lang.org',
'Spammer'   : 'spammer@hotmail.com'
}

print "Swaroop's address is %s" % ab['Swaroop']

# Adding a key/value pair
ab['Guido'] = 'guido@python.org'

# Deleting a key/value pair
del ab['Spammer']

print '\nThere are %d contacts in the address-book\n' %
len(ab)

for name, address in ab.items():
print 'Contact %s at %s' % (name, address)

if 'Guido' in ab: # OR ab.has_key('Guido')
print "\nGuido's address is %s" % ab['Guido']
```

خروجی

```
$ python using_dict.py
Swaroop's address is swaroopch@byteofpython.info

There are 4 contacts in the address-book

Contact Swaroop at swaroopch@byteofpython.info
Contact Matsumoto at matz@ruby-lang.org
Contact Larry at larry@wall.org
Contact Guido at guido@python.org

Guido's address is guido@python.org
```

نحوه ی عملکرد این مثال

ما در این مثال ، دیکشنری ab را با همان نشانی که قبلا توضیح داده شد ، ایجاد نمودیم . سپس به جفت های کلید/مقدار آن ، با مشخص کردن کلیدی که درعملگر ایندکس به کار رفته است ، به همان صورتی که در مبحث لیست ها و چندتایی ها بحث شد ، دسترسی پیدا نمودیم . واضح است که نحوی (syntax) که برای دیکشنری به کار رفته است ، بسیار ساده است .

برای اضافه کردن جفت های کلید/مقدار جدید ، می توانیم این کار را به سادگی با استفاده از عملگر ایندکس ، برای دسترسی به یک کلید و مقداردهی به آن به کار ببریم . همان طوری که برای Guido در مثال بالا انجام دادیم .

ما می توانیم جفت های کلید/مقدار را با استفاده از دستور del حذف کنیم . برای این کار ، به سادگی دیکشنری و عملگر ایندکس را برای کلیدی که می خواهد حذف شود مشخص نموده و سپس آن را به دستور delمی فرستیم . برای این عمل ، نیازی نیست که مقدار مرتبط با کلید را بدانیم .

سپس ، ما به هریک از جفت های کلید/مقدار از دیکشنری با متد items از شیء دیکشنری دسترسی پیدا می کنیم ، این متد یک لیست از چندتایی هایی را که هر چندتایی حاوی یک جفت از آیتم های کلید و مقدار دیکشنری ست را برمی گرداند ، ما با استفاده از حلقه ی foor .. in هر جفت از این جفت ها را بازیابی نموده و به ترتیب به متغیرهایی به نام name و Addrees نسبت می دهیم و سپس این مقادیر را در یک بلوک for چاپ می کنیم .

می توانیم با استفاده از عملگر in یا حتی متد has_key از کلاس dict بررسی کنیم که آیا یک جفتِ کلید/مقدار در دیکشنری وجود دارد یا خیر . شما می توانید برای لیست کاملی از متدهای کلاس dict مستندات آن را با استفاده از دستور help(dict) ملاحظه نمائید .

دیکشنری و آرگومان های کلیدی . نکته ایی در حاشیه ، اگر شما از آرگومان های کلیدی در تابع تان استفاده کرده باشید ، در واقع از دیکشنری استفاده کرده اید ! کافیست در این مورد فکر کنید . جفت های کلید/مقدار توسط شما در لیست پارامترهای تعریف تابع مشخص می شوند ، و آن گاه هنگامی که در داخل تابع تان به متغیرها دسترسی پیدا می کنید ، این دقیقا همان دسترسی به کلیدهای یک دیکشنری ست . (که در اصطلاح طراحی کامپایلر symbol table نامیده می شود (

دنباله ها

لیست ها ، چندتایی ها و رشته ها ، مثال هایی از دنباله ها هستند ، اما دنباله ها چه هستند و چه ویژگی هایی دارند ؟ دو ویژگی اصلی از دنباله ها عبارتست از عملگر ایندکس گذاری **(indexing)**که باعث می شود بتوانیم مستقیما به یک آیتم خاص در دنباله دسترسی پیدا کنیم و عملگرتکه کردن **(slicing)**که باعث می شود بتوانیم یک تکه از رشته را بدست آوریم ، یعنی یک بخش از دنباله را .

استفاده از دنباله ها

مثال 9.5. بکارگیری دنباله ها

```
#!/usr/bin/python
# Filename: seq.py

shoplist = ['apple', 'mango', 'carrot', 'banana']

# Indexing or 'Subscription' operation
print 'Item 0 is', shoplist[0]
print 'Item 1 is', shoplist[1]
print 'Item 2 is', shoplist[2]
print 'Item 3 is', shoplist[3]
print 'Item -1 is', shoplist[-1]
print 'Item -2 is', shoplist[-2]

# Slicing on a list
print 'Item 1 to 3 is', shoplist[1:3]
print 'Item 2 to end is', shoplist[2:]
print 'Item 1 to -1 is', shoplist[1:-1]
print 'Item start to end is', shoplist[:]

# Slicing on a string
name = 'swaroop'
print 'characters 1 to 3 is', name[1:3]
print 'characters 2 to end is', name[2:]
print 'characters 1 to -1 is', name[1:-1]
print 'characters start to end is', name[:]
```

خروجی

```
$ python seq.py
Item 0 is apple
Item 1 is mango
Item 2 is carrot
Item 3 is banana
Item -1 is banana
Item -2 is carrot
Item 1 to 3 is ['mango', 'carrot']
Item 2 to end is ['carrot', 'banana']
Item 1 to -1 is ['mango', 'carrot']
Item start to end is ['apple', 'mango', 'carrot',
'banana']
characters 1 to 3 is wa
characters 2 to end is aroop
characters 1 to -1 is waroo
characters start to end is swaroop
```

نحوه ی عملکرد این مثال

در ابتدا ، چگونگی استفاده از ایندکس، برای گرفتن تک آیتم هایی از دنباله را ملاحظه می کنیم . که به این مطلب همچنین بعنوان عملگر subscriptionنیز اشاره می شود . هر گاه که شما یک عدد در داخل جفت براکت برای یک دنباله مشخص می کنید ، پایتون آیتم مربوط به آن موقعیت در دنباله را برای شما بدست می آورد . یادتان باشد که پایتون عدد شمارش را از 0شروع می کند . بنابراین ، shoplit[0]، اولین آیتم و shoplist[3]، چهارمین آیتم دردنباله ی shoplist را بدست می آورد .

همچنین ایندکس می تواند یک عدد منفی باشد ، که در این مورد ، موقعیتش از آخر دنباله محاسبه می شود . بنابراین shoplist[-1] به آخرین آیتم و shoplist[-2] به دومین آیتم از آخر، در دنباله اشاره می کند .

عملگر slicing ، با نام دنباله و بدنبال آن یک جفت از اعداد اختیاری که با علامت : درون جفت براکت از هم جدا شده اند ، مشخص می شود . دقت کنید که این مطلب بسیار زیاد شبیه به عملگر ایندکس گذاری هست که تا بدین جا از آن استفاده می کردیم . و یادتان باشد که اعداد اختیاری هستند ، اما علامت : این طور نیست .

اولین عدد (پیش از علامت :) در عملگر slicing به موقعیتی که تکه کردن از آنجا شروع می شود ، اشاره می کند و عدد دوم (بعد از علامت :) مشخص می کند که در کجا تکه کردن پایان یابد . اگر اولین عدد مشخص نگردد ، پایتون از ابتدای دنباله شروع خواهد کرد و اگر دومین عدد مشخص نشود ، در انتهای دنباله پایان می یابد . توجه کنید که slice ، از موقعیت شروع تا پیش از موقعیت پایان را بر می گرداند ، بدین معنی که در تکه ی دنباله ، موقعیت شروع را شامل می شود ولی موقعیت پایان را شامل نمی شود .

بنابراین ، دستور shoplist[1:3] یک تکه از دنباله را که شروعش از موقعیت 1 است ، (که دارای مکان 2 هست) ولی پایانش در مکان 3 است را بر می گرداند ، پس بنابراین یک تکه از دو آیتم را برمی گرداند و دستور shoplist[:]یک کپی از تمام دنباله را بر می گرداند .

ترکیب های گوناگونی از تکه کردن رشته را با اتکا به تعاملی بودن مفسرپایتون امتحان کنید ، چرا که نتیجه ی دستورات وارده شده را فورا در خط فرمان ملاحظه خواهید کرد . یک نکته با ارزش در مورد دنباله ها اینست که به یک روش یکسان می توانید به تمامی چندتایی ها ، لیست ها و رشته ها دسترسی یابید .

مرجع ها

وقتی شما یک شیء را ایجاد می کنید و یک متغیر را به آن نسبت می دهید ، متغیر فقط به شیء **رجوع** می کند و نشان دهنده ی خود شیء نیست ! که این یعنی ، نام متغیر به بخشی از حافظه ی کامپیوتر شما که آن شیء در آنجا ذخیره شده است اشاره می کند . و بعنوان bind شدن یک نام به یک شیء از آن نام برده می شود .

درکل ، لازم نیست که نگران این موضوع باشید ، اما تاثیرات ظریفی از مرجع ها (references) هست که می بایست از آن آگاه باشید ، این مطلب با مثال زیر روشن می شود .

اشیاء و مرجع ها

مثال 9.6. اشیاء و مرجع ها

```
#!/usr/bin/python
# Filename: reference.py

print 'Simple Assignment'
shoplist = ['apple', 'mango', 'carrot', 'banana']
```

```
mylist = shoplist # mylist is just another name
pointing to the same object!

del shoplist[0] # I purchased the first item, so I
remove it from the list

print 'shoplist is', shoplist
print 'mylist is', mylist
# notice that both shoplist and mylist both print the
same list without
# the 'apple' confirming that they point to the same
object

print 'Copy by making a full slice'
mylist = shoplist[:] # make a copy by doing a full
slice
del mylist[0] # remove first item

print 'shoplist is', shoplist
print 'mylist is', mylist
# notice that now the two lists are different
```

خروجی

```
$ python reference.py
Simple Assignment
shoplist is ['mango', 'carrot', 'banana']
mylist is ['mango', 'carrot', 'banana']
Copy by making a full slice
shoplist is ['mango', 'carrot', 'banana']
mylist is ['carrot', 'banana']
```

نحوه ی عملکرد این مثال

اکثر توضیحات این مثال در خود برنامه بصورت comment آمده است . چیزی که می بایست به خاطر بسپارید این است که اگر یک کپی از یک لیستی و یا دیگر انواع دنباله ها یا اشیاء ی complex ایجاد کنید (و نه اشیاء ساده مثل integerها) ، می بایست که از عملگر تکه سازی برای انجام کپی استفاده کنید . اگر که تنها نام

متغیری را به نام دیگری نسبت بدهید ، هر دوی آنها به یک شیء یکسان اشاره خواهند کرد و این در صورتیکه مراقب نباشید ، می تواند منجر به بوجود آمدن انواع مشکلاتی بشود .

نکته ایی در مورد برنامه نویسان Perl

به خاطر بسپارید که دستور انتساب برای لیست ها یک کپی از آن ها را ایجاد نمی کند . شما می بایست از عملگر تکه سازی برای ایجاد یک کپی از دنباله استفاده کنید .

کمی بیشتر درباره ی رشته ها

ما درباره ی جرئیات رشته ها بحث کردیم . چه چیز بیشتر دیگری برای دانستن هست ؟ خب ، آیا می دانید که رشته ها هم اشیایی هستند و در نتیجه متدهایی دارند که بروی بخش هایی از رشته .

رشته هایی که شما در برنامه های تان استفاده می کنید ، همگی اشیایی از کلاس str هستند . برخی از متدهای مفید این کلاس در مثال بعدی نشان داده می شوند . برای لیست کاملی از متدهای آن خروجی دستور help(str)را ملاحظه نمائید .

متدهای رشته

مثال 9.7. متدهای رشته

```
#!/usr/bin/python
# Filename: str_methods.py

name = 'Swaroop' # This is a string object

if name.startswith('Swa'):
print 'Yes, the string starts with "Swa"'

if 'a' in name:
print 'Yes, it contains the string "a"'

if name.find('war') != -1:
```

```
print 'Yes, it contains the string "war"'

delimiter = '_*_'
mylist = ['Brazil', 'Russia', 'India', 'China']
print delimiter.join(mylist)
```

خروجی

```
$ python str_methods.py
Yes, the string starts with "Swa"
Yes, it contains the string "a"
Yes, it contains the string "war"
Brazil_*_Russia_*_India_*_China
```

نحوه ی عملکرد این مثال

دراین مثال ، در عمل بسیاری از متدهای رشته را می بینیم . متد startswithبرای مشخص کردن اینکه آیا رشته ی اصلی با رشته ی داده شده آغاز شده است یا خیر به کار می رود . عملگر in ، برای برای بررسی اینکه رشته داده شده بخشی از رشته اصلی می باشد یا خیر به کار می رود .

متد find برای پیداکردن موقعیت رشته داده شده در رشته اصلی بکار می رود ، و اگر موفق به پیداکردن این زیر رشته در رشته اصلی نشود ، مقدار - 1 را بر می گرداند . همچنین کلاس str دارای یک متد بسیار عالی به نام join می باشد که برای **الحاق** آیتم های یک ساختمان داده ی دنباله ایی به یکدیگر با اتکا به رشته ایی که نقش جداکننده ی مابین هر آیتم را در دنباله دارد به کار می رود ، و در نتیجه یک رشته ی بزرگ تری را که از آن تولید می شود را برمی گرداند .

خلاصه

ما جزئیات انواع مختلفی از ساختارهای داده ایی توکار را در پایتون بررسی کردیم . این ساختمان های داده برای نوشتن برنامه هایی با سایزهای معقول مناسب اند .

اکنون تا بدین جا ، ما با بسیاری از اصول پایه ایی پایتون آشنا شده ایم ، در فصل بعدی خواهیم دید که چطور یک برنامه ی واقعی به زبان پایتون را طراحی و پیاده سازی می نمائیم .

کتاب یک بایت از پایتون.فصل دهم.حل مسائل.نوشتن برنامه ایی به زبان پایتون

فهرست مندرجات

فصل دهم .حل مسائل . نوشتن برنامه ایی به زبان پایتون

ما بخش های متنوعی از زبان برنامه نویسی پایتون را تشریح کردیم و اکنون می خواهیم با طراحی و نوشتن یک برنامه که قرار است کارمفیدی را برای ما انجام بدهد ، ببینیم که چطور می توانیم تمامی این بخش ها را بدرستی درکنار هم قرار دهیم .

مسئله

مسئله ی مورد نظر ما عبارتست از : " من می خواهم برنامه ایی برای پشتیبان گیری از تمامی فایل های ضروری ام ایجاد کنم ".

هرچند ، این مسئله ی ساده ایی ست ، ولی اطلاعات مورد نیاز را برای شروع حل مسئله در اختیار ما قرار نمی دهد . در واقع **تجزیه و تحلیل** بیشتری مورد نیاز است . برای مثال ، به چه صورت باید مشخص کنیم که کدام فایل ها شامل پشتیبان گیری می شوند ؟ این فایل ها به چه صورت در فایل پشتیبان ذخیره می شوند ؟ و فایل پشتیبان تهیه شده در کجا ذخیره می شود ؟

بعد از تجزیه و تحلیل مناسب مسئله ، برنامه مان را **طراحی** می کنیم . ما لیستی از نحوه ی عملکرد برنامه ایجاد می کنیم . در این مورد ، من لیست زیر را بصورتی که می خواهم برنامه کار کند ، ایجاد کرده ام . اگر شما طراحی برنامه را بر عهده بگیرید ، ممکن است که با چنین موارد یکسانی که در لیست زیر آمده است روبه رو نشوید – هر شخصی روش خاص خودش را برای انجام این کاردارد ، و درست هم هست .

- نام فایل ها و دایرکتوری هایی که می بایست از آنها نسخه ی پشتیبان گرفته شود در یک لیست مشخص می شوند .
- فایل پشتیبان تهیه شده می بایست در یک دایرکتوری اصلی ذخیره شود .
- فایل های پشتیانی گرفته شده در داخل یک فایل zip قرار می گیرند .
- در نام فایل آرشیو (zip) ، تاریخ و ساعت فعلی را ذکر می کنیم .
- ما از دستوراستاندارد zip که در توزیع های استاندارد یونیکس / لینوکس ، قابل دسترس است ، برای ایجاد فایل پشتیبان کمک می گیریم . کاربران ویندوز می توانند از برنامه ی Info-Zip استفاده کنند . توجه کنید که شما می توانید از هر برنامه ی فشرده سازی که از دستورات خط فرمان پشتیبانی می کند ، استفاده کنید ، چرا که می توانیم از طریق برنامه مان دستورات را به آن برنامه ارسال کنیم .

حل مسئله

از آنجایی که طراحی برنامه ی ما به یک مرحله ی مناسب و پایدار رسید ، اکنون می توانیم کدنویسی را ، که در واقع یک پیاده سازی از حل مسئله مان می باشد را آغاز کنیم .

نسخه ی اول برنامه

مثال 10.1. برنامه ی پشتیبان گیری – اولین نسخه

```
#!/usr/bin/python
# Filename: backup_ver1.py

import os
import time

# 1. The files and directories to be backed up are
specified in a list.
source = ['/home/swaroop/byte', '/home/swaroop/bin']
```

```
# If you are using Windows, use source =
[r'C:\Documents', r'D:\Work'] or something like that

# 2. The backup must be stored in a main backup
directory
target_dir = '/mnt/e/backup/' # Remember to change this
to what you will be using

# 3. The files are backed up into a zip file.
# 4. The name of the zip archive is the current date
and time
target = target_dir + time.strftime('%Y%m%d%H%M%S') +
'.zip'

# 5. We use the zip command (in Unix/Linux) to put the
files in a zip archive
zip_command = "zip -qr '%s' %s" % (target, '
'.join(source))

# Run the backup
if os.system(zip_command) == 0:
print 'Successful backup to', target
else:
print 'Backup FAILED'
```

خروجی

```
$ python backup_ver1.py
Successful backup to /mnt/e/backup/20041208073244.zip
```

اکنون ، ما در مرحله ی (فاز) تست برنامه هستیم ، که صحت و درستی عملکرد برنامه مان را امتحان می کنیم . اگر دراین مرحله برنامه بدرستی کار نکند ، ما می بایست برنامه را اشکال زدایی (Debug) کنیم ، بدین معنی که خطاهای (bug)برنامه را رفع کنیم .

نحوه ی عملکرد این مثال

شما ملاحظه می نمائید که چطور ما در یک روش مرحله به مرحله طراحی مان را به سمت تولید کد سوق می دهیم . ما از ماژول های os و time در برنامه کمک خواهیم گرفت ، از اینرو آنها را در ابتدا وارد (import) برنامه کردیم . سپس ، در لیست source فایل ها و دایرکتوری هایی که قرار است از آن ها نسخه ی پشتیبان گرفته شود را مشخص کردیم . مسیر دایرکتوری مقصد نیز ، که تمامی فایل های پشتیانی در آن ذخیره می شود را در متغیر target_dict مشخص نموده ایم . نام فایل آرشیوی (zip) که قصد ایجاد آن را داریم ، بر اساس تاریخ و زمان فعلی ایی هست که از طریق تابع time.strftime() بدست می آوریم . این فایل آرشیو همچنین دارای پسوند .zipمی باشد و در دایرکتوری target_dir ذخیره خواهد شد .

تابع time.strftime() ، مشخصه ایی (specification) را بعنوان آرگومان ورودی بصورتی که ما در مثال بالا استفاده کردیم می پذیرد . به این صورت که مشخصه ی %y با سال جاری جایگزین خواهد شد . سال بدون سده می باشد و عددی صحیح از 00 تا 99 است . مشخصه ی %m ، با ماه جاری جایگزین خواهد شد ، ماه بصورت عددی صحیح از 01 تا 12 می باشد . لیست کاملی از این مشخصه ها را می توانید در مستندات پایتون ملاحظه نمائید . توجه کنید که این مطلب مشابه (ولی نه یکسان) با مشخصه های بکار رفته در دستور print)که با یک چندتایی همراه بود) است .

نام فایل zip مقصد را با استفاده از عملگر + که باعث اتصال رشته ها به هم می شود ، ایجاد می نمائیم .بدین معنی که این عملگر دو رشته را بکدیگر الحاق می کند و یک رشته جدید را بر می گرداند . سپس رشته ی zip_commandرا که حاوی دستوراتی ست که می خواهیم اجرا شود را ایجاد می کنیم . برای بررسی اینکه آیا این دستور به درستی اجرا می شود یا خیر می توانید آن را در) shellخط فرمان dos و یا ترمینال لینوکس)اجرا نمائید .

دستور zip ایی که ما دراینجا استفاده نموده ایم دارای چندین پارمترو سوئیچ ارسالی است . پارامتر q– برای مشخص کردن این است که دستور zipبه صورت ساکت (quietly) کار کند . گزینه ی r– معین می کند که دستور zipمی بایست بصورت بازگشتی بروی دایرکتوری ها عمل کند ، بدین معنی که می بایست دایرکتوری های داخلی و فایل های درون دایرکتوری های داخلی را نیز شامل شود . این دو گزینه به روش کوتاهتری بصورت qr– نوشته می شوند . بدنبال گزینه ها نام فایل آرشیوی (zip) که می خواهد ایجاد شود و لیست فایل ها و دایرکتوری هایی که قرار است نسخه ی پشتیبان از آنها گرفته شود ، می آید . ما لیست source را با روشی تازه که اکنون در اینجا ملاحظه می نمائید ، با کمک متد join که مربوط به رشته ها است ، به رشته تبدیل نمودیم .

در نهایت این دستور ایجاد شده را با استفاده از تابع os.system اجرا می کنیم ، در صورتیکه این دستور با موفقیت در سیستم اجرا شود ، منظور در shell – مقدار 0 را برمی گرداند و در غیراینصورت شماره ی خطای اتفاق افتاده را بر می گرداند .

و منوط به خروجی این دستور ، ما پیغام مناسب را که نسخه ی پشتیبان با موفقیت یا با شکست ایجاد شده است را نمایش می دهیم .

بله ، اکنون ما موفق شدیم برنامه ایی بنویسیم که از فایل های مهم ما نسخه ی پشتیبان می گیرد !

نکته ایی برای کاربران ویندوزی

شما می توانید لیست source و دایرکتوری target را بروی هر فایل و دایرکتوری دلخواهی تنظیم کنید ، اما در مورد ویندوز می بایست کمی مراقب باشید . دلیلش اینست که ویندوز از کاراکتر (\ ) برای جداکننده ی مسیر استفاده می کند ، اما پایتون از این کاراکتر (\ ) برای نمایش دنباله های گریز (escape sequences) استفاده می کند !

از اینرو ، برای نمایش کاراکتر (\ ) می بایست پیش از آن از یک کاراکتر (\ ) دیگربعنوان escape sequenceاستفاده کنید و یا اینکه از raw strings استفاده کنید . پس برای مثال ، باید یا به این صورت 'C:\\Douments'و یا بصورت r'C:\Documents\' بنویسد ، و نباید به شکل 'C:\Docments' بنویسید ، چرا که دراین صورت شما از یک escape sequence نامفهموم(\D) استفاده کرده اید !

اکنون که ما با این برنامه ی پشتیبان گیر کار کردیم ، می توانیم از آن در هر جایی که می خواهیم از فایل ها نسخه ی پشتیبان گرفته شود ، استفاده کنیم . کاربران Unix\Linux نیز از روش های قابل اجرایی که اخیرا توضیح داده شد برای اجرای برنامه ی پشتیبان گیر در هر زمان و هر مکانی می توانند استفاده کنند . این قضیه بعنوان فاز عملکرد (operation) و یا فاز توسعه (deploymet) از نرم افزار نام برده می شود .

برنامه ی بالا بدرستی کار می کند ، ولی (معمولا) اولین برنامه ها آنطور که انتظار داریم کار نمی کنند . برای مثال ، اگر بدرستی طراحی برنامه را انجام نداده باشید ، ممکن است مشکلاتی پیش بیاید ، یا اگر کدی را اشتباهی تایپ کرده باشید و یا چیزهای دیگر. در این شرایط می بایست که به فاز طراحی برگردید و یا شاید مجبور باشید برنامه تان را اشکال زدایی نمائید .

نسخه ی دوم برنامه

نسخه ی اول از برنامه ی ما کار کرد . اما ، می خواهیم آن را کمی بهینه و اصلاح کنیم تا بهتر کارکند . این قضیه فاز **نگهداری (maintenance)**نرم افزار نامیده می شود .

یکی از بهبودهایی که من احساس می کنم موثر است ، بهتر کردن مکانیسم نام دهی فایل است – به این صورت که از زمان فعلی بعنوان نام فایل استفاده شود و این فایل در داخل دایرکتوری قرار بگیرد که بر اساس تاریخ فعلی در دایرکتوری اصلی پشتیبان ایجاد می شود ، یکی از مزیت های این روش این است که پشتیبان های تهیه شده ی شما در یک روش سلسله مراتبی ذخیره می شوند و درنتیجه مدیریت آنها آسان تر می شود . دیگر مزیت این روش کوتاه شدن طول نام فایل هاست . و مزیت دیگر اینست که جداسازی دایرکتوری ها به شما کمک می کند که به آسانی نسخه های پشتیبان تهیه شده را به ازای هر روز بررسی کنید ، چرا که هر دایرکتوری تنها درصورتی که نسخه ی پشتیبانی برای آن روز گرفته شود ، ایجاد می شود.

مثال 10.2. برنامه ی پشتیبان – نسخه ی دوم

```
#!/usr/bin/python
# Filename: backup_ver2.py

import os
import time

# 1. The files and directories to be backed up are
specified in a list.
source = ['/home/swaroop/byte', '/home/swaroop/bin']
# If you are using Windows, use source =
[r'C:\Documents', r'D:\Work'] or something like that

# 2. The backup must be stored in a main backup
directory
target_dir = '/mnt/e/backup/' # Remember to change this
to what you will be using

# 3. The files are backed up into a zip file.
# 4. The current day is the name of the subdirectory in
the main directory
today = target_dir + time.strftime('%Y%m%d')
```

```
# The current time is the name of the zip archive
now = time.strftime('%H%M%S')

# Create the subdirectory if it isn't already there
if not os.path.exists(today):
os.mkdir(today) # make directory
print 'Successfully created directory', today

# The name of the zip file
target = today + os.sep + now + '.zip'

# 5. We use the zip command (in Unix/Linux) to put the
files in a zip archive
zip_command = "zip -qr '%s' %s" % (target, '
'.join(source))

# Run the backup
if os.system(zip_command) == 0:
print 'Successful backup to', target
else:
print 'Backup FAILED'
```

خروجی

```
$ python backup_ver2.py
Successfully created directory /mnt/e/backup/20041208
Successful backup to /mnt/e/backup/20041208/080020.zip

$ python backup_ver2.py
Successful backup to /mnt/e/backup/20041208/080428.zip
```

نحوه ی عملکرد برنامه

بیشتر کدهای این نسخه از برنامه همانند ، نسخه ی قبلی است . تغییری که در برنامه رخ داده است ، این است که با استفاده از تابع os.exists بررسی می کنیم که آیا یک دایرکتوری به اسم تاریخ امروز درون دایرکتوری

پشتیبان اصلی وجود دارد یا خیر. اگر چنین دایرکتوری موجود نباشد ، آن را با استفاده از تابع os.mkdir ایجاد می کنیم .

به کاربرد متغیر os.sep توجه کنید – این متغییر حاوی کاراکتر جداکننده ی دایرکتوری مطابق با نوع سیستم عامل شما است . بدین معنی که ، در سیستم عامل لینوکس/یونیکس برابر با '/' است و در ویندوز برابر با '\\' و در مکینتاش برابر با کاراکتر ':' است . استفاده از متغیر os.sep به جای استفاده ی مستقیم از این کاراکتر ها باعث می شود ، برنامه ی شما قابل حمل باشد و بروی سیستم عامل های مختلف به خوبی کار کند .

نسخه ی سوم برنامه

نسخه ی دوم برنامه هم وقتی که من به دفعات برای پشتیبان گیری از آن استفاده کردم به خوبی کار می کند ، اما وقتی که فایل های پشتیبان زیاد می شوند ، فهمیدم که پیداکردن این که هر فایل پشتیبان مربوط به چه چیزی است سخت می شود . برای مثال ، من یک سری تغییراتی را در یک برنامه یا یک فایل مربوط به کنفرانس ایجاد می کنم ، سپس می خواهم این چنین تغییراتی را در نام فایل آرشیوی (zip) که گرفته می شود انعکاس بدهم ، این کار می تواند به آسانی با استفاده از گرفتن توضیحات کاربر در حین ایجاد نسخه ی پشتیبان و درج آن در نام فایل آرشیو بدست آید .

مثال 10.3 . برنامه ی تهیه ی نسخه ی پشتیبان – نسخه ی سوم (که البته کار نمی کند (!

```
#!/usr/bin/python
# Filename: backup_ver2.py

import os
import time

# 1. The files and directories to be backed up are
specified in a list.
source = ['/home/swaroop/byte', '/home/swaroop/bin']
# If you are using Windows, use source =
[r'C:\Documents', r'D:\Work'] or something like that

# 2. The backup must be stored in a main backup
directory
target_dir = '/mnt/e/backup/' # Remember to change this
to what you will be using
```

```
# 3. The files are backed up into a zip file.
# 4. The current day is the name of the subdirectory in
the main directory
today = target_dir + time.strftime('%Y%m%d')
# The current time is the name of the zip archive
now = time.strftime('%H%M%S')

# Take a comment from the user to create the name of
the zip file
comment = raw_input('Enter a comment --> ')
if len(comment) == 0: # check if a comment was entered
target = today + os.sep + now + '.zip'
else:
target = today + os.sep + now + '_' +
comment.replace(' ', '_') + '.zip'

# Create the subdirectory if it isn't already there
if not os.path.exists(today):
os.mkdir(today) # make directory
print 'Successfully created directory', today

# 5. We use the zip command (in Unix/Linux) to put the
files in a zip archive
zip_command = "zip -qr '%s' %s" % (target, '
'.join(source))

# Run the backup
if os.system(zip_command) == 0:
print 'Successful backup to', target
else:
print 'Backup FAILED'
```

خروجی

```
$ python backup_ver3.py
File "backup_ver3.py", line 25
target = today + os.sep + now + '_' +
^
```

```
SyntaxError: invalid syntax
```

نحوه ی عملکرد (نادرست) این برنامه

این برنامه بدرستی کار نمی کند ! مفسر پایتون اعلام می کند که خطای نحوی وجود دارد ، بدین معنی که ساختار صحیحی که پایتون انتظار دیدن آن را داشته است رعایت نشده است . وقتی که ما متوجه ی خطایی از سوی مفسر پایتون می شویم ، این خطا ، مکانی که خطا در آن رخ داده شده است را نیز بخوبی اعلام می کند . درنتیجه اشکال زدایی برنامه را از همان خط شروع می کنیم .

اگر کمی بیشتر دقت کنیم ، ملاحظه می کنیم که یک خط منطقی از کد برنامه در دو خط مجزا شکسته شده است ، اما ما مشخص نکرده ایم که این دوخط درادامه و متعلق به یکدیگر هستند . بطور کلی مفسر پایتون در انتهای خط اول با عملگر (+) رو به رو می شود ، بدون آنکه عملوندی بعد ازاین عملگر بیابد و از اینرو نمی داند که چطور باید ادامه بدهد . برای این منظور برای بیان مرتبط بودن این دو خط به یکدیگر می بایست در خط اول از کاراکتر (\ ) ، استفاده کنیم .پس به این روش برنامه مان را تصحیح می کنیم . به این کاری که اکنون انجام دادیم ، رفع اشکال برنامه می گویند .

نسخه ی چهارم

مثال 10.4 .برنامه ی تهیه نسخه ی پشتیبان – نسخه ی چهارم

```
#!/usr/bin/python
# Filename: backup_ver2.py

import os, time

# 1. The files and directories to be backed up are
specified in a list.
source = ['/home/swaroop/byte', '/home/swaroop/bin']
# If you are using Windows, use source =
[r'C:\Documents', r'D:\Work'] or something like that

# 2. The backup must be stored in a main backup
directory
```

```
target_dir = '/mnt/e/backup/' # Remember to change this
to what you will be using

# 3. The files are backed up into a zip file.
# 4. The current day is the name of the subdirectory in
the main directory
today = target_dir + time.strftime('%Y%m%d')
# The current time is the name of the zip archive
now = time.strftime('%H%M%S')

# Take a comment from the user to create the name of
the zip file
comment = raw_input('Enter a comment --> ')
if len(comment) == 0: # check if a comment was entered
target = today + os.sep + now + '.zip'
else:
target = today + os.sep + now + '_' + \
comment.replace(' ', '_') + '.zip'
# Notice the backslash!

# Create the subdirectory if it isn't already there
if not os.path.exists(today):
os.mkdir(today) # make directory
print 'Successfully created directory', today

# 5. We use the zip command (in Unix/Linux) to put the
files in a zip archive
zip_command = "zip -qr '%s' %s" % (target, '
'.join(source))

# Run the backup
if os.system(zip_command) == 0:
print 'Successful backup to', target
else:
print 'Backup FAILED'
```

خروجی

```
$ python backup_ver4.py
Enter a comment --> added new examples
Successful backup to
/mnt/e/backup/20041208/082156_added_new_examples.zip

$ python backup_ver4.py
Enter a comment -->
Successful backup to /mnt/e/backup/20041208/082316.zip
```

نحوه ی عملکرد این مثال

اکنون این برنامه به درستی کار می کند ! اجازه بدهید ، بهبودهایی را که در این برنامه نسبت به نسخه ی سوم اعمال کردیم ، را بررسی کنیم . ما توضیحات کاربر را با استفاده از تابع raw_input ، دریافت می کنیم . سپس طول رشته را با تابع len بررسی می کنیم تا متوجه شویم که آیا کاربر واقعا چیزی وارد کرده است یا خیر . اگر که کاربر بنا به دلایلی (ممکن است یک پشتیبان گیری عادی باشد و یا تغییرات خاصی ایجاد نشده باشد.) تنها کلید Enterرا فشار داده باشد ، سپس ما بصورتی که قبلا رفتار می کردیم ، عمل می کنیم .

و اگر توضیحی توسط کاربر وارد شده بود ، این توضیح در نام فایل آرشیو (zip) پیش از پسوند zip. درج می شود ، توجه کنید که ما فضاهای خالی موجود در رشته ی توضیحات کاربر را با کارکتر (_) ، جایگزین می کنیم ، چرا که این کار مدیریت نام های فایل ها را آسان تر می کند .

اعمال بهبود واصلاح بیشتر در برنامه

برنامه ی چهارم ، برای اکثر کاربران به خوبی کار می کند ، اما همیشه همچنان جا برای بهبود برنامه وجود خواهد داشت . برای مثال ، می توانید سطح پاسخگویی رابه برنامه تان اضافه کنید ، به این صورت که گزینه ی v–را برای پاسخگویی کاملتر برنامه مشخص کنید .

بهبود ممکن دیگراینست که اجازه بدهید که دایرکتوری ها و فایل های اضافی از طریق خط فرمان به برنامه ارسال شوند . ما در برنامه اسامی آنها را از طریق لیست sys.argv دریافت می کنیم و می توانیم لیست آن ها را با متد extendاز کلاس list به لیست source اضافه کنیم .

و یکی دیگراز بهبودهایی که من ترجیح می دهم انجام شود ، اینست که از دستور tar به جای دستور zip استفاده کنیم . یکی از مزیت های این کار اینست که هنگامی که از دستور tar به همراه gzip استفاده می کنید ، فایل پشتیبان هم سریعتر و هم خیلی کوچک تر ایجاد می شود . و همچنین اگر من به این فایل آرشیو

در ویندوز احتیاج داشته باشم ، WinZipبه خوبی فایل هایی همچون tar.gz. را اداره می کند . دستور tar بطور پیش فرض در اغلب سیستم های Unix/Linuxدردسترس می باشد . کاربران ویندوز هم می توانند آن را از آدرس][۱] [دانلود و نصب کنند .

درنتیجه برای استفاده از دستور tar رشته ی دستوری به صورت زیر در می آید :

```
tar = 'tar -cvzf %s %s -X /home/swaroop/excludes.txt' %
(target, ' '.join(srcdir))
```

شرح سوئیچ های برنامه در ادامه آمده است :

- c-، ایجاد فایل آرشیو را مشخص می کند .
- v-، verboseرا مشخص می کند ، بدین معنی که دستورات می بایست پاسخگویی کاملتری داشته باشند .
- z-، مشخص می کند که فیلتر gzip می بایست اعمال شود .
- f-، اجبار در ایجاد فایل آرشیو را مشخص می کند ، بدین معنی که اگر فایل هم نامی از قبل موجود است ، می بایست فایل آرشیو جدید جایگزین آن شود .
- X-، لیستی از فایل هایی که نباید در فایل آرشیو قرار بگیرند را مشخص می کند ، برای مثال ، معنی ~*اینست که فایل آرشیو شامل فایل هایی که اسم شان به ~ ختم می شود ، نمی باشد .

مهم

یک روش بسیار عالی برای بهبود برنامه اینست که از خود ماژول های پایتون به اسم tarfile و zipfile برای ایجاد فایل آرشیو استفاده کنیم . این ماژول ها جزء کتابخانه ی استاندارد پایتون هستند و در نتیجه هم اکنون برای استفاده در دسترس شما هستند . استفاده ازاین کتابخانه ها از بکارگیری دستور os.system که بطورکلی برای استفاده توصیه نمی شود ، (چرا که این دستور براحتی ممکن است اشتباهات پرهزینه ایی را بوجود بیاورد.) نیز جلوگیری می کند .

هر چند که من در اینجا صرفا به جهت آموزش از روش os.system برای ایجاد فایل پشتیبان استفاده کردم .

فرآیند توسعه ی نرم افزار

ما در این فصل در حین فرآیند نوشتن نرم افزار از میان فازهای مختلفی گذر کردیم . این فاز ها می توانند بصورت زیر خلاصه شوند :

- چه چیزی)آنالیز (
- چطور)طراحی (
- انجام دادن (پیاده سازی (
- تست (تست و اشکال زدایی (
- استفاده) عملکرد یا توسعه (
- نگهداری (بهبود (

مهم

روش پیشنهاد شده برای نوشتن برنامه ها ، عبارتست از شیوه ایی که ما در نوشتن برنامه ی تهیه ی نسخه ی پشتیبان آن را دنبال نمودیم :

آنالیز و طراحی را انجام دادیم . پیاده سازی را با یک نسخه ی ساده شروع کردیم . آن را تست و اشکال زادیی کردیم . آن را بکارگرفتیم و استفاده کردیم تا مطمئن شویم که همانند انتظارمان عمل می کند .

اکنون هر ویژگی که می خواهید را به برنامه اضافه کنید و چرخه ی انجام –تست – استفاده را برای چندین بار تا جایی که نیاز هست تکرار کنید ، به یاد داشته باشید که نرم افزار رشد می کنند ، نه اینکه ساخته شود .

خلاصه

ما ملاحظه نمودیم که به چه شکل برنامه های پایتون دلخواه خودمان را ایجاد کنیم و دراین حین درگیر فازهای مختلفی درنوشتن برنامه شدیم . و همان طور که در این فصل دیدیم ، شما هم این فاز ها را در ایجاد برنامه های تان مفید خواهید یافت ، و از اینرو در پایتون و به همان اندازه در حل مسائل آسوده خواهید شد .

در فصل بعدی ، ما برنامه نویسی شیء گرا را بحث خواهیم کرد .

کتاب یک بایت از پایتون.فصل یازدهم.برنامه نویسی شیء گرا

فهرست مندرجات

برنامه نویسی شیء گرا

معرفی

تا بدین جا در تمامی برنامه ها ، برای طراحی برنامه های مان ، از توابع و یا بلاک هایی از دستوراتی که داده ها را دستکاری می کردند استفاده می کردیم ، این روش در برنامه نویسی بعنوان برنامه نویسی رویه گرا (procedure-oriented)شناخته می شود . شیوه ی دیگری از ساماندهی برنامه ها وجود دارد که داده و عملیات بروی داده را در کنار هم ترکیب می کند و این دو را در داخل آن چیزی قرار می دهد که شیء نامیده می شود . این شیوه از برنامه نویسی بعنوان الگوی برنامه نویسی شیء گرا شناخته می شود . اکثر مواقع ممکن است که شما از برنامه نویسی ساخت یافته استفاده کنید ولی زمانی که قصد نوشتن برنامه های بزرگ و یا راه حلی که بیان اش در این روش مناسب تر است را دارید ، می توانید از شیوه برنامه نویسی شیء گرا استفاده کنید .

کلاس ها و اشیاء دو مفهوم اساسی در برنامه نویسی شیء گرا هستند . یک **کلاس** یک نوع جدید را ایجاد می کند ، در حالیکه **اشیاء** نمونه هایی از آن کلاس هستند . بدین معنی که ، اینکه شما می توانید متغیرهایی از نوع int داشته باشید ، به تعبیری دیگر مثل آنست که بگوئید ، متغیرهایی که اعداد صحیح را ذخیره می کنند ، متغیرهایی هستند که نمونه هایی)اشیاء ایی) از کلاس int هستند .

نکاتی برای برنامه نویسان C/C++/Java/C#

توجه داشته باشید که حتی اعداد صحیح هم به مانند اشیاء (اشیایی از کلاس (intرفتار می کنند . این برخلاف C++ و جاوا است (البته پیش از نسخه (1.5که اعداد صحیح از نوع های اولیه اصلی هستند . برای جزئیات بیشتر در مورد این کلاس help(int) را ملاحظه کنید .

برنامه نویسان C# و Java 1.5 با این مفهوم آشنا تر هستند ، چرا که شبیه به مفهوم boxing و unboxing در این زبان ها هست .

اشیاء می توانند داده ها را با استفاده از متغیرهای معمولی که به شیء تعلق دارد ، ذخیره کنند . متغیرهایی که به یک شیء یا یک کلاس تعلق دارند ، **فیلد** نامیده می شوند . همچنین اشیاء می توانند عملیاتی را با استفاده از توابعی که متعلق به کلاس هستند ، انجام دهند . این چنین توابعی method هایی از کلاس نامیده می شوند . این اصطلاحات بسیار مهم هستند ، چرا که به ما کمک می کند تا بین توابع و متغیرهایی که بصورت جدا از هم تعریف می شوند با توابع و متغیرهایی که متعلق به یک کلاس و یا شیء هستند ، تفاوت قائل بشویم .بطور مشترک ، فیلد ها و متدها می توانند بعنوان مشخصه/ویژگی هایی (attributes) از کلاس اشاره شوند .

فیلدها بر دو قسم اند : یک قسم آنهایی که می توانند به هر شیء /نمونه ایی از کلاس تعلق بگیرند ، و یک قسم دیگر آنهایی که می توانند به کلاس خودشان تعلق بگیرند . قسم اول **instance variables**و قسم دوم **class variables**نامیده می شوند .

کلاس با استفاده از کلمه کلیدی class ایجاد می شود ، فیلدها و متدهای کلاس در یک بلاک تورفته فهرست می شوند .

Self

متدهای کلاس تنها یک تفاوت مشخص با توابع معمولی دارند ، آنها می بایست دارای یک اسم اضافی باشند که بعنوان اولین آرگومان به فهرست پارامترهای متد اضافه شود ، اما شما هنگام فراخوانی متد نباید مقداری به آن ارسال کنید ،چرا که پایتون خود این کار را انجام می دهد . این متغیر خاص به شیء خودش رجوع می کند ، و رسم هست که نام self به آن داده شود .

هرچند که می توانید هر اسمی به این پارامتر بدهید ، اما به شدت پشنهاد می شود که از نام self استفاده کنید – هر نام دیگری بطور معلوم ناخوشایند است . مزیت های زیادی هست که از یک نام استاندارد استفاده بشود –

هر خواننده ی برنامه های شما فورا آن را تشخیص می دهد و حتی اگر از نام self استفاده کنید IDE های (محیط های توسعه یکپارچه) مخصوص نیز می توانند به شما کمک کنند .

نکاتی برای برنامه نویسان C++/Java/C#

self در پایتون همان اشاره گر self در C++ و مرجع this در جاوا و C# است .

حتما کنجکاو هستید که پایتون به چه صورت مقدار self را ارسال می کند و چرا شما نیازی به مقداردهی به آن ندارید . با یک مثال این موضوع را شفاف می کنیم ، فرض کنید کلاسی به اسم MyClass و نمونه ایی از این کلاس به نام MyObject دارید . هنگامی که متدی از این کلاس را بعنوان نمونه MyObject.method(arg1,arg2) صدا می زنید . این کد بصورت خودکار توسط پایتون به MyClass.method(Myobject,arg1,arg2) تبدیل می شود -این تمام آن چیزی بود که self بطور خاص داشت . این مطلب همچنین بدین معنی ست که اگر متدی داشته باشید که هیچ آرگومانی بعنوان ورودی نمی پذیرد ، باز شما می بایست آرگومان self را برای آن تعریف نمائید .

کلاس ها

یک کلاسِ تا حد ممکن ساده در مثال زیر آورده شده است .

ایجاد یک کلاس : مثال 11.1 . ایجاد یک کلاس :

```
#!/usr/bin/python
# Filename: simplestclass.py

class Person:
pass # An empty block

p = Person()
print p
```

خروجی :

```
$ python simplestclass.py
```

```
<__main__.Person instance at 0xf6fcb18c>
```

نحوه ی عملکرد این مثال

ما یک کلاس جدید را با استفاده از دستور class و بدنبال آن با آوردن نام کلاس و متعاقبا با یک بلاک تو رفته از دستورات که بدنه کلاس را شکل می دهد ایجاد کردیم . در این مثال ، ما بلاک خالی تو رفته را تنها با یک دستور pass نشان دادیم . سپس ، یک شیء/نمونه از این کلاس را با استفاده از نام کلاس همراه با یک جفت پرانتز باز وبسته ، ایجاد کردیم . (در بخش بعدی درباره نمونه سازی بیشتر خواهیم آموخت . (وسپس برای اثبات صحت کارمان به سادگی با چاپ کردن متغیر، از نوع آن مطمئن می شویم ، چرا که به ما نشان می دهد که یک نمونه از کلاس Person ، در ماژول __main__ داریم . توجه داشته باشید که آدرس حافظه کامپیوتری که شیء در آن ذخیره شده است نیز چاپ می شود . این آدرس در کامپیوترهر کسی متفاوت هست ، چرا که پایتون می تواند شیء را در هر کجایی از حافظه که فضای خالی یافت ذخیره کند.

متدهای شیء

ما درباره اشیاء / کلاس ها صحبت کردیم و گفتیم که می توانند متدهایی مشابه به توابع داشته باشند ، تنها با این استثناء که یک آرگومان اضافی به نام self دارند . حال می خواهیم مثال هایی دراین رابطه داشته باشیم .

استفاده از متدهای اشیاء

مثال 11.2 . به کارگیری متدهای اشیاء

```
#!/usr/bin/python
# Filename: method.py

class Person:
def sayHi(self):
print 'Hello, how are you?'

p = Person()
p.sayHi()

# This short example can also be written as
Person().sayHi()
```

خروجی

```
$ python method.py
Hello, how are you?
```

نحوه ی عملکرد این مثال

در این مثال در عمل با self آشنا شدیم . دقت کنید که متد SayHi پارامتری نمی گیرد ولی با این حال self را در تعریف تابع خود دارد .

متد __init__

تعدادی از نام های متدها هستند که برای کلاس های پایتون معنی و مفهوم خاصی دارند . یکی از آن نام ها __init__هست ، که اکنون می خواهیم به مفهوم آن پی ببریم .متد __init__ ، به مجرد اینکه یک شیء از کلاس نمونه سازی می شود ، اجرا می شود . این متد برای هر کار اولیه ایی که می خواهید با شیء تان انجام دهید ، مناسب است . به دو زیرخطی که در ابتدا و انتهای نام این متد هست دقت داشته باشید .

به کارگیری متد __init__

مثال 11.3 . استفاده از متد __init__

```
#!/usr/bin/python
# Filename: class_init.py

class Person:
def __init__(self, name):
self.name = name
def sayHi(self):
print 'Hello, my name is', self.name

p = Person('Swaroop')
p.sayHi()

# This short example can also be written as
Person('Swaroop').sayHi()
```

خروجی :

```
$ python class_init.py
Hello, my name is Swaroop
```

نحوه ی عملکرد این مثال

در اینجا ، ما متد __init__ را طوری تعریف کردیم که یک پارامتر به اسم name بپذیرد (در کنارپارامتر معمول و همیشگی self) . سپس تنها یک فیلد جدید به نام name ایجاد کردیم . دقت کنید که این ها متغیرهایی متفاوت هستند ، هر چند که نام های شان مثل هم است . علامت نقطه گذاشته شده باعث می شود که بین آنها تمایز قائل شویم . نکته حائز اهمیت این است که ما بطور صریح متد __init__ را فراخوانی نکردیم ، اما آرگومان های آن را در حین ایجاد یک نمونه از کلاس ، در بین پرانتزهای نام کلاس به آن فرستادیم . این مفهوم خاص این متد هست . اکنون ما می توانیم از فیلد self.name در تمامی متدهای مان استفاده کنیم ، که این کار در متد SayHi مشخص هست .

نکته ایی برای برنامه نویسان C++/Java/C#

متد __init__ در قیاس می تواند مثل سازندها در C++ ، C#و جاوا باشد .

متغیرهای شیء و متغیرهای کلاس

تا بدین جا درباره عملیاتی که بخشی از اشیاء و کلاس هستند بحث شد ، اکنون ما می خواهیم درباره ی داده که بخش دیگری از اشیاء و کلاس هستند بحث کنیم . در واقع ، داده ها چیز خاصی به جز متغیرهای معمولی که به فضای نامی اشیاء و کلاس ها bound شده اند ، نیستند ، بدین معنی که نام های این متغیرها تنها در داخل این کلاس ها و اشیاء معتبر هستند .

دو نوع ازفیلد ها وجود دارند – متغیرهای کلاس و متغیرهای شیء ، که طبقه بندی شان به ترتیب ، منوط براین است که آیا متغیرها متعلق به کلاس هستند و یا به شیء .

متغیرهای کلاس به این جهت به اشتراک گذاشته شده اند تا برای تمامی اشیاء ی (نمونه ها) کلاس قابل دسترس باشند . تنها یک کپی از متغیر کلاس وجود دارد و هنگامی که هر شیء ایی تغییری در یک متغیر کلاس بدهد ، این تغییر به درستی در تمامی نمونه های دیگر منعکس می شود .

متغیرهای شیء ،متعلق به هر شیء/نمونه منحصر به فرد از یک کلاس هستند .در این حالت ، هر شیء کپی مخصوص به خودش را از فیلد دارد ، بدین معنی که آنها اشتراک گذارده نمی شوند و رابطه ایی بین نام های

یکسان آن ها در کلاس های یکسان از نمونه های متفاوت وجود ندارد . یک مثال باعث فهم آسان این مسئله خواهد شد .

استفاده از متغیرهای شیء و کلاس

مثال 11.4 . به کارگیری متغیرهای شیء و کلاس

```
#!/usr/bin/python
# Filename: objvar.py

class Person:
'''Represents a person.'''
population = 0

def __init__(self, name):
'''Initializes the person's data.'''
self.name = name
print '(Initializing %s)' % self.name

# When this person is created, he/she
# adds to the population
Person.population += 1

def __del__(self):
'''I am dying.'''
print '%s says bye.' % self.name

Person.population -= 1

if Person.population == 0:
print 'I am the last one.'
else:
print 'There are still %d people
left.' % Person.population

def sayHi(self):
'''Greeting by the person.
```

```
Really, that's all it does.'''
print 'Hi, my name is %s.' % self.name

def howMany(self):
'''Prints the current population.'''
if Person.population == 1:
print 'I am the only person here.'
else:
print 'We have %d persons here.' %
Person.population

swaroop = Person('Swaroop')
swaroop.sayHi()
swaroop.howMany()

kalam = Person('Abdul Kalam')
kalam.sayHi()
kalam.howMany()

swaroop.sayHi()
swaroop.howMany()
```

خروجی

```
$ python objvar.py
(Initializing Swaroop)
Hi, my name is Swaroop.
I am the only person here.
(Initializing Abdul Kalam)
Hi, my name is Abdul Kalam.
We have 2 persons here.
Hi, my name is Swaroop.
We have 2 persons here.
Abdul Kalam says bye.
There are still 1 people left.
Swaroop says bye.
I am the last one.
```

نحوه ی عملکرد این مثال

این مثال نسبتا بزرگی ست ولی ماهیت متغیرهای کلاس و شیء را برای ما روشن می کند . در اینجا ، population متعلق به کلاس Person هست و بنابراین یک متغیر کلاس هست . متغیر name متعلق به شیء (که با استفاده از self نسبت دهی صورت گرفته است) هست و از اینرو یک متغیر شیء است .

بنابراین ما به متغیر کلاس population بصورت Person.population رجوع می کنیم و نه به صورت self.population . توجه داشته باشید در صورتیکه یک متغیر شیء ، نام یکسانی با متغیر کلاس داشته باشد ، متغیر کلاس را خواهد پوشاند ! در یک متد از شیء ، با استفاده از نشان self.name به متغیر شیء name رجوع می کنیم . تفاوت ساده بین متغیرهای کلاس و متغیرهای شیء را به خاطر بسپارید .

مشخص هست که متد __init__ برای ایجاد کارهای مقدماتی و اولیه ی نمونه ی Person به کار گرفته شده است . در این متد ، ما مقدار population را به اندازه یک واحد افزایش می دهیم ، چرا که یک شخص جدید به جمعیت ما اضافه می شود . همچنین واضح هست که مقدار self.name مختص هر شیء است و ماهیت متغیر شیء را برای ما روشن می کند .

به یاد داشته باشید که ، می بایست به متغیرها و متدهای یک شیء یکسان تنها با استفاده از متغیر self رجوع کنید . که این مطلب ، ارجاع به ویژگی (attribute reference) نامیده می شود .

همچنین در این مثال ، ما می بینیم که از docstring ها، هم برای کلاس و هم برای متدها استفاده کردیم . می توانیم در زمان اجرا به docstring مربوط به کلاس با استفاده از Person.__doc__ و به docstring مربوط به متد از طریق Person.sayHi.__doc__ دسترسی داشته باشیم .

به مانند متد خاص __init__، متد ویژه دیگری به نام __del__ نیز وجود دارد که هنگامیکه یک شیء در حال از بین رفتن هست ، فراخوانی می شود ، بدین معنی که آن شیء از این پس مورد استفاده قرار نمی گیرد و برای استفاده مجدد از آن بخش از حافظه ، به سیستم برگردانده می شود . ما در این متد به سادگی مقدار Person.populationرا یک واحد کاهش دادیم .

متد __del__ وقتی که شیء ایی بیش ازاین نیازی به استفاده ازش نیست اجرا می شود و وقتی که این متد اجرا شود دیگر ضمانتی به آن شیء نیست ، اگر که شما می خواهید بطور صریح این کار را انجام بدهید ، می بایست از دستور delبصورتی که در مثال های قبلی آورده شده بود ، استفاده کنید .

نکته ایی برای برنامه نویسان C++/Java/C# در پایتون تمامی اعضای یک کلاس (شامل اعضای داده) عمومی (public) هستند و تمامی متدها ، مجازی ((virtual هستند .

یک استثناء : اگر نام اعضای داده را بصورت دو زیرخط چسپبیده به نام تعریف کنید ، مثل __privatevar پایتون از دستکاری نام (name-mangling) بهره می برد تا بطور موثری آن متغیر را خصوصی (private) سازد .

]مترجم : پس این متغیر از طریق نام classname._classname__privatevar همچنان قابل دسترس و عمومی ست [.

از اینرو طبق عرف ، هر متغیری که تنها در داخل خود کلاس و یا اشیاء مورد استفاده قرارمی گیرد ، می بایست نامش با یک زیرخط شروع شود ، و تمامی دیگر نام ها عمومی هستند و می توانند توسط دیگر کلاس ها / اشیاءها استفاده شوند . به خاطر داشته باشید که این تنها یک عرف هست و اجباری از طرف پایتون برای آن نیست . (البته به جز برای نام هایی با پیشوند دو زیر خط . (

همچین ، دقت کنید که متد __del__ در قیاس می توان گفت که همان مفهوم مخرب (destructor) را در زبان های بالا دارد .

وراثت

یکی از مزایای اصلی برنامه نویسی شیء گرا ، قابلیت استفاده مجدد از کد هست ، و یکی از روش های رسیدن به آن از طریق مکانیسم وراثت است . ارث بری می تواند بصورت بهتری اینگونه تصور شود که پیاده سازی یک رابطه نوع و زیر نوع ما بین کلاس ها است .

فرض کنید که می خواهید برنامه ایی بنویسید که اطلاعات مربوط به اساتید و دانشجو یک دانشکده را نگه دارد . این موجودیت ها دارای چندین ویژگی رایج همچون نام ، سن و آدرس هستند . همچنین دارای چندین ویژگی مخصوص به خود همچون حقوق ، کلاس درس ، مرخصی برای اساتید و شهریه و نمره برای دانشجویان هستند .

شما می توانید دو کلاس مستقل برای هر کدام از این ها ایجاد کنید و آنها را پردازش کنید ولی اضافه شدن یک ویژگی جدید رایج ، به معنی اضافه کردن این ویژگی به هر دو کلاس مستقل است . که این مطلب به سرعت باعث عدم اداره کردن آسان کلاس ها می شود .

روش بهتری هم هست ، و آن اینکه یک کلاس رایج به نام SchoolMember ایجاد کنیم و سپس دو کلاس به نام دانشجو و استاد داشته باشیم که از این کلاس ارث ببرند . بدین معنی که این دو ، زیر- نوع هایی از این نوع (کلاس (رایج هستند و سپس ما می توانیم ویژگی های مختص به خودشان را به این زیر –نوع ها اضافه کنیم .

این شیوه مزایای بسیاری دارد . اگر ما هرعملیاتی را در کلاس SchoolMemberاضافه کنیم و یا تغییر بدهیم ، این تغییر بطور خودکار در زیر- نوع ها هم به خوبی منعکس می شود ، برای مثال ، شما می توانید برای هر دو موجودیت اساتید و دانشجو یک فیلد شماره کارت (ID) جدید اضافه کنید و این کار را به سادگی ، تنها با اضافه کردن آن به کلاس SchoolMember انجام بدهید . البته ، تغییر در یک زیرنوع تاثیری بروی دیگر زیرنوع ها ندارد . مزیت دیگر آن اینست که اگر شما به یک کلاس استاد و یا دانشجو بعنوان یک شیء SchoolMemberرجوع کنید می تواند در مواردی همچون شمارش تعداد اعضای دانشکده مفید باشد . این **چندریختی** نامیده می شود ، که در آن یک زیر -نوع می تواند جانشین هر مکانی شود که انتظار می رود یک نوع والد درآن مکان قرار گیرد ، بدین معنی که یک شیء می تواند بعنوان یک نمونه از کلاس والد رفتار کند.

همچنین مشخص هست که ما از کد کلاس والد استفاده مجدد می کنیم و نیازی به تکرار آن در کلاس های متفاوت نیست ، و همینطوری که ما در همین جا داشتیم ، و در کلاس هایی مستقل از آن استفاده بردیم.

در این وضعیت کلاس SchoolMember ، بعنوان کلاس پایه (base class) و یا فوق کلاس (superclass)شناخته می شود ، و کلاس های استاد و دانشجو کلاس های مشتق شده(derived classes) و یا زیرکلاس (subclasses) نامیده می شوند .

حالا ما این مثال را بعنوان یک برنامه خواهیم دید .

به کارگیری وراثت

مثال 11.5. استفاده از وراثت

```
#!/usr/bin/python
# Filename: inherit.py

class SchoolMember:
'''Represents any school member.'''
def __init__(self, name, age):
self.name = name
```

```
self.age = age
print '(Initialized SchoolMember: %s)' %
self.name

def tell(self):
'''Tell my details.'''
print 'Name:"%s" Age:"%s"' % (self.name,
self.age),

class Teacher(SchoolMember):
'''Represents a teacher.'''
def __init__(self, name, age, salary):
SchoolMember.__init__(self, name, age)
self.salary = salary
print '(Initialized Teacher: %s)' %
self.name

def tell(self):
SchoolMember.tell(self)
print 'Salary: "%d"' % self.salary

class Student(SchoolMember):
'''Represents a student.'''
def __init__(self, name, age, marks):
SchoolMember.__init__(self, name, age)
self.marks = marks
print '(Initialized Student: %s)' %
self.name

def tell(self):
SchoolMember.tell(self)
print 'Marks: "%d"' % self.marks

t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 22, 75)

print # prints a blank line

members = [t, s]
for member in members:
```

```
member.tell() # works for both Teachers and
Students
```

خروجی

```
$ python inherit.py
(Initialized SchoolMember: Mrs. Shrividya)
(Initialized Teacher: Mrs. Shrividya)
(Initialized SchoolMember: Swaroop)
(Initialized Student: Swaroop)
Name:"Mrs. Shrividya" Age:"40" Salary: "30000"
Name:"Swaroop" Age:"22" Marks: "75"
```

نحوه ی عملکرد این مثال

برای استفاده از وراثت ، ما نام های کلاس پایه را در یک چندتایی به دنبال نام کلاس در تعریف کلاس مشخص می کنیم . سپس ، مشاهده می کنیم که متد __init__از کلاس پایه با استفاده از متغیر self بطور صریح فراخوانی می شود ، بدین ترتیب که مقدمات اولیه کلاس پایه را که بخشی از شیء است انجام می دهیم . این بسیار مهم است که به خاطر داشته باشیم که پایتون بطور خودکار سازنده ی کلاس پایه را فراخوانی نمی کند ، و شما می بایست بطور صریح آن را برای خودتان صدا بزنید .

همچنین می بینیم که فراخوانی متدهای کلاس پایه در زیرکلاس ها با پیشوند نام کلاس پایه همراه با اسم متد آن و سپس فرستادن متغیر self بعنوان اولین آرگومان در کنار دیگرآرگومان ها صورت می گیرد .

توجه کنید که، هنگامیکه متد Tell از کلاس SchoolMember را به کار می بریم با نمونه هایی از Teacherو Studentمی توانیم همان طوری رفتار کنیم که با نمونه هایSchoolMember رفتار می کنیم .

همچنین ، روشن هست که متد Tell از زیرنوع صدا زده می شود و نه متد Tellاز کلاس SchoolMember .یک راه برای درک این مطلب ، این است که پایتون همیشه شروع جستجو برای متدها را در نوعی (کلاسی) که در آن هست آغاز می کند ، سپس اگر نتواند متد مورد نظر را در آن نوع (کلاس) پیدا کند ، ادامه جستجو را در متدهای متعلق به کلاس های پایه اش شروع می کند ، بصورت یکی یکی و به ترتیبی که در چندتایی در تعریف کلاس مشخص شده اند .

نکته ایی در مورد یک اصطلاح – اگر در زمان تعریف ارث بری ، بیش از یک کلاس در چندتایی ارث بری فهرست شده باشد ، آن را وراثت چندگانه (multiple inheritance)می نامند .

خلاصه

ما اکنون مفاهیم متنوعی از کلاس ها و اشیاء و همین طور اصطلاحات گوناگون مربوط به آنها را مورد بررسی قرار دادیم . همچنین مزایا و معایب برنامه نویسی شیء گرا را دیدیم . پایتون قویاً شیء گرا و قابل فهم است . این مفاهیم بطور قابل ملاحظه ایی در جای خودش به شما کمک بیشتری خواهد کرد .

در فصل بعدی ، درباره مواجهه با ورودی / خروجی و چگونگی دسترسی به فایل ها در پایتون خواهیم آموخت .

کتاب یک بایت از پایتون.فصل دوازدهم.ورودی/خروجی

فهرست مندرجات

ورودی / خروجی

بسیاری از اوقات هست که می خواهید برنامه تان با کاربر تعامل داشته باشد)که می تواند با خودتان باشد) . می خواهید که ورودی ایی از کاربر بگیرد وسپس نتایجی را برگردانده وچاپ کند . ما می توانیم به این خواسته ها به ترتیب با دستورات raw_input و print برسیم . همچنین برای خروجی ، می توانیم از متدهای گوناگون کلاس str(string) استفاده ببریم . برای مثال ، شما می توانید از متد rjust برای تنظیم راست چین شدن یک رشته با عرض مشخص ، بهره ببرید . برای جزئیات بیشتر help(str) را ملاحظه کنید .

دیگر روش رایج در ورودی / خروجی ، مواجهه با فایل هاست . توانایی خواندن و نوشتن فایل ها ، برای اکثر برنامه ها ضروری ست و ما می خواهیم در این فصل آن را شرح دهیم.

فایل ها

برای بازکردن و استفاده کردن از فایل ها برای خواندن و یا نوشتن می توان ابتدا یک شیء از کلاس file را ایجاد کرد و سپس از متدهای read ، readlineو یا write که مختص خواندن و نوشتن در فایل ها هستند ، کمک گرفت . توانایی خواندن و یا نوشتن در فایل ها منوط به مُدی ست که شما در حین بازکردن فایل مشخص کرده اید . و در نهایت ، هنگامیکه کارتان با فایل تمام می شود، متد close از کلاس file را صدا می زنید تا به پایتون بگوئید که کارتان با فایل به پایان رسیده است .

به کارگیری فایل

مثال 12.1. استفاده از فایل ها

```
#!/usr/bin/python
# Filename: using_file.py

poem = '''\
Programming is fun
When the work is done
if you wanna make your work also fun:
use Python!
'''

f = file('poem.txt', 'w') # open for 'w'riting
f.write(poem) # write text to file
f.close() # close the file

f = file('poem.txt') # if no mode is specified, 'r'ead
mode is assumed by default
while True:
line = f.readline()
if len(line) == 0: # Zero length indicates EOF
break
print line, # Notice comma to avoid automatic
newline added by Python
f.close() # close the file
```

خروجی

```
$ python using_file.py
Programming is fun
When the work is done
if you wanna make your work also fun:
use Python!
```

نحوه ی عملکرد این مثال

در ابتدا ، ما یک نمونه از کلاس file را با مشخص کردن نام فایل و مُدی که می خواهیم فایل باز شود ، ایجاد نمودیم . مُد می تواند مقادیر ('r') برای خواندن ، ('w')برای نوشتن و یا ('a') برای اضافه کردن به فایل را داشته باشد . باید گفت که در واقع مُدهای بیشتری نیز در دسترس هستند ، دستور help(file) ، جزئیات بیشتری دراین باره به شما خواهد داد .

ما در ابتدا ، فایل را در مُد نوشتن بازکردیم ، و سپس از متد write از کلاس file برای نوشتن در فایل استفاده کردیم و پس از آن در نهایت فایل را بستیم .

سپس ، بار دیگری همان فایل را برای خواندن باز کردیم . اگر که ما مُد فایل را مشخص نکنیم ، بطور پیش فرض مُد فایل بروی حالت خواندن قرار می گیرد . هر خط از فایل را در یک حلقه با استفاده از متد readline خواندیم ، این متد یک خط کامل ، شامل کارکتر خط جدید که در انتهای هر خط هست را برمی گرداند . خب وقتی که یک رشته خالی برگشت داده می شود ، مشخص می کند که به پایان فایل رسیده ایم و درنتیجه چرخش در حلقه را متوقف می کنیم .

دقت کنید که ما از یک ویرگول در دستورprintاستفاده کردیم تا از اضافه کردن خط جدیدی که دستورprint بصورت خودکار چاپ می کند جلوگیری کند ، چرا که خطی که از فایل خوانده می شود خود دارای کاراکتر خط جدید در انتهای آن هست ، سپس در نهایت فایل را می بندیم .

اکنون ، محتوای فایل poem.txt را ببینید تا موجه بشوید که برنامه واقعا بدرستی کار می کند .

Pickle

پایتون ماژول استانداری به نام pickle ارائه می دهد که با استفاده از آن می توانید هر شیء ایی از پایتون را در یک فایل ذخیره نموده و سپس بعدها بدون کوچک ترین کم و کاستی به آن دسترسی پیدا کنید . که به این عمل *ذخیره سازی پایدار شیء* نامیده می شود .

ماژول دیگری به نام cPickle هم هست که دقیقا همان عمل ماژول Pickle را انجام می دهد ، به استثنای اینکه با زبان C نوشته شده است و درنتیجه 1000)برابر) سریعترست . شما می توانید از هر دوی این ماژول ها استفاده کنید ، هرچند که ما در اینجا از ماژول cPickle استفاده خواهیم کرد . به خاطر داشته باشید که ما به هردوی این ماژول ها برای سادگی به اسم ماژول pickleاشاره می کنیم .

Pickling وUnpickling

مثال 12.2 Pickling . و Unpickling

```
#!/usr/bin/python
# Filename: pickling.py

import cPickle as p
#import pickle as p

shoplistfile = 'shoplist.data' # the name of the file
where we will store the object

shoplist = ['apple', 'mango', 'carrot']

# Write to the file
f = file(shoplistfile, 'w')
p.dump(shoplist, f) # dump the object to a file
f.close()

del shoplist # remove the shoplist

# Read back from the storage
f = file(shoplistfile)
storedlist = p.load(f)
print storedlist
```

خروجی :

```
$ python pickling.py
['apple', 'mango', 'carrot']
```

نحوه ی عملکرد این مثال

در ابتدا ، توجه کنید که ما از رسم الخط import .. as استفاده کردیم . چرا که استفاده از یک نام کوتاه تر برای یک ماژول ، باعث آسان شدن تایپ های بعدی در برنامه می شود ، .و در این مورد ، حتی باعث می شود که ما بتوانیم بین ماژول های متفاوت (Pickle و یا (cPickle تنها با تغییر دادن یک خط سوئیچ کنیم ! ، در مابقی برنامه ما به سادگی با حرف pبه این ماژول رجوع می کنیم .

برای ذخیره کردن یک شیء در یک فایل ، در ابتدا یک شیء فایل را در حالت نوشتن باز می کنیم و سپس شیء مورد نظررا با فراخوانی تابع dump از ماژول pickle ، در داخل فایل باز شده می نویسیم ، این فرآیند *pickling*نامیده می شود .

سپس ، شیء مورد نظر را با استفاده از تابع load از ماژول pickle که شیء را برمی گرداند بازیابی می کنیم ، این فرآیند *unpickling*نامیده می شود .

خلاصه

ما درباره روش های گوناگونی از ورودی / خروجی و همچنین اداره کردن فایل و استفاده از ماژول pickle بحث کردیم .

در فصل بعدی ، مفاهیم مربوط به استثناء ها را شرح خواهیم داد .

کتاب یک بایت از پایتون.فصل سیزدهم.استثناءها

فهرست مندرجات

استثناء ها

استثناء ها زمانیکه یک وضعیت غیرعادی و استثنایی در برنامه رخ می دهد، اتفاق می افتند . برای مثال ، اگر شما بخواهید فایلی را بخوانید که آن فایل وجود ندارد چه می شود ؟ و یا وقتی که بطور تصادفی آن فایل را در زمانی که برنامه در حال اجرا بوده است حذف کرده اید ؟ چنین شرایطی بوسیله ی **استثناء ها** (**exceptions**)اداره می شوند .

اگر برنامه ی شما دارای برخی دستورات نامعتبرباشد چطور ؟ دراین حالت بوسیله ی پایتون اداره می شود که دستانش را بلند (raise) می کند و به شما می گوید که یک خطا (error)وجود دارد .

خطاها

یک دستور ساده ی print را در نظر بگیرید . اگر ما به جای print اشتباها بنویسیم Print چه می شود ؟ به حرف بزرگ P در دستور printدقت کنید . در این مورد ، پایتون یک خطای نحوی (syntax error) را برپا ((raiseمی کند .

```
>>> Print 'Hello World'
    File "<stdin>", line 1
      Print 'Hello World'
                         ^
SyntaxError: invalid syntax

>>> print 'Hello World'
Hello World
```

در این مثال واضح است که یک خطای نحوی (SyntaxError) برخاسته (raise) و همچنین مکانی که خطا تشخیص داده شده است چاپ شده است . همچین کاری توسط *اداره کننده ی خطا* (*error handler*)انجام می شود .

Try..Except

در برنامه زیر ما می خواهیم **سعی کنیم** (**try**)که ورودی را از کاربر بخوانیم . کلیدهای Ctrl-z را فشار بدهید و ببینید که چه اتفاقی می افتد .

```
>>> s = raw_input('Enter something --> ')
Enter something --> Traceback (most recent call last):
  File "<stdin>", line 1, in ?
EOFError
```

همان طور که مشاهده کردید ، پایتون خطایی را که به نام خطای انتهای فایل (EOFError)شناخته می شود را برپا (raise) می کند . که بطور اساسی به این معنی ست که پایتون با انتهای فایل مواجهه شده است در حالی که انتظار آن را ندارد . (که با Ctrl-z نشان داده شد (.

در ادامه ، خواهیم دید که به چه شکل خودمان همچین خطاهایی را اداره می کنیم .

اداره کردن استثناء ها

می توانیم با استفاده از عبارت try..except استثناء ها را اداره کنیم . بطور کلی برای انجام این کار دستورات معمولی را در داخل بلاک try قرار می دهیم و دستورات مربوط به اداره کردن خطاها را در داخل بلاک exceptمی گذاریم .

مثال 13.1 . اداره کردن استثناء ها

```
#!/usr/bin/python
# Filename: try_except.py

import sys

try:
s = raw_input('Enter something --> ')
except EOFError:
print '\nWhy did you do an EOF on me?'
sys.exit() # exit the program
except:
print '\nSome error/exception occurred.'
# here, we are not exiting the program

print 'Done'
```

خروجی

```
$ python try_except.py
Enter something -->
Why did you do an EOF on me?

$ python try_except.py
Enter something --> Python is exceptional!
Done
```

نحوه ی عملکرد این مثال

در این مثال ، ما تمامی دستوراتی را که امکان برخاستن خطا در آن ها هست را در داخل بلاک try قرار دادیم و سپس تمامی خطاها و استثناء ها را در داخل بلوک /عبارتِ except اداره کردیم . عبارت except می تواند یک خطا و یا استثناء مشخص شده مجرد را و یا یک لیستی پرانتزی از استثناء ها و خطاها را اداره کند . اگر نامی از خطا ها و یا استثناء ها در جلوی عبارت except آورده نشده باشد ،آن گاه عبارت except تمام خطاها و استثناء ها را اداره خواهد کرد . حداقل به ازی هر عبارت try یک عبارت except می بایست مشخص شده باشد .

اگر که هر خطا و یا استثنایی اداره نشده باشد ، آن گاه اداره کننده پیش فرض پایتون صدا زده می شود و باعث می شود که استثناء ی برنامه متوقف شود و پیغامی چاپ شود ، همان طوری که این مطلب را در عمل مشاهده کرده بودیم .

همچنین می توانیم یک عبارت else نیز با بلاک try..ecxept مشخص کنیم . عبارت else زمانیکه استثنایی روی ندهد ، اجرا می شود .

ما همچنین می توانیم یک شیء از نوع استثناء را ایجاد کنیم ، از اینرو که اطلاعات اضافی تری در خصوص استثنایی که روی داده است بدست آوریم . این مطلب در مثال بعدی نشان داده می شود .

برانگیختن استثناء ها

شما می توانید با استفاده از دستور raise یک استثناء را برانگیزید . (raise)همچنین می بایست بعد از آن نام خطا و یا استثناء و یا شیء استثنایی که پرتاب شده است را در جلوی استثناء مشخص کنید . این خطا و یا

استثنایی که شما می توانید برپا (raise) کنید ، می بایست کلاسی باشد که بطور مستقیم و یا بطور غیرمستقیم به ترتیب از کلاس Error و یا کلاس Exceptionمشتق شده باشد .

چگونه یک استثناء را برانگیزیم

مثال 13.2. چگونگی برانگیختن یک استثناء

```
#!/usr/bin/python
# Filename: raising.py

class ShortInputException(Exception):
'''A user-defined exception class.'''
def __init__(self, length, atleast):
Exception.__init__(self)
self.length = length
self.atleast = atleast

try:
s = raw_input('Enter something --> ')
if len(s) < 3:
raise ShortInputException(len(s), 3)
# Other work can continue as usual here
except EOFError:
print '\nWhy did you do an EOF on me?'
except ShortInputException, x:
print 'ShortInputException: The input was of
length %d, \
was expecting at least %d' % (x.length,
x.atleast)
else:
print 'No exception was raised.'
```

خروجی

```
$ python raising.py
Enter something -->
```

```
Why did you do an EOF on me?

$ python raising.py
Enter something --> ab
ShortInputException: The input was of length 2, was
expecting at least 3

$ python raising.py
Enter something --> abc
No exception was raised.
```

نحوه ی عملکرد این مثال

در این مثال ، ما هر چند که می توانستیم از هریک از خطا ها و یا استثناء های پیش فرض پایتون استفاده کنیم ، اما برای نشان دادن هدف مان ، نوع استثناء خودمان را ایجاد کردیم . این نوع استثناء جدید ، کلاس ShortInputExceptionنام دارد ، که دارای دو فیلد هست : فیلد طول (length) که طول داده شده به ورودی است ، و فیلد حداقل (atleast) که حداقل طولی است که برنامه انتظار دریافت آن را دارد .

همچنین ما در بلاک except ، به کلاس خطا به صورت یک متغیر اشاره می کنیم تا به خوبی شیء مربوط به خطا و یا استثناء را در خود نگه دارد . این مطلب را می توان با پارامترها و آرگومان ها در فراخوانی یک تابع مقایسه نمود . در داخل این بلاک except بخصوص ، ما از فیلدهای length و atleast از شیء استثناء استفاده کردیم تا پیغام مناسبی را برای کاربر چاپ کنیم .

Try..Finaly

اگر شما بخواهید فایلی را که می خوانید ، چه استثناء رخ بدهد و چه ندهد ، درهر حال در پایان کار ، فایل بسته شود چه می کنید؟ این کار با استفاده از بلاک finaly انجام می شود . توجه کنید که شما می توانید یک عبارت except را همراه با یک بلاک finally برای همان بلاک try متناظرش ، استفاده کنید . اگر که می خواهید از هر دو استفاده کنید می بایست یکی را در داخل دیگری قرار بدهید .

استفاده از Finally

مثال 13.3 . بکارگیری Finally

```
#!/usr/bin/python
# Filename: finally.py

import time

try:
f = file('poem.txt')
while True: # our usual file-reading idiom
line = f.readline()
if len(line) == 0:
break
time.sleep(2)
print line,
finally:
f.close()
print 'Cleaning up...closed the file'
```

خروجی

```
$ python finally.py
Programming is fun
When the work is done
Cleaning up...closed the file
Traceback (most recent call last):
  File "finally.py", line 12, in ?
    time.sleep(2)
KeyboardInterrupt
```

نحوه ی عملکرد این مثال

ما از قضیه معمول خواندن – فایل استفاده کردیم . اما من به طور قراردادی روشی را برای صبر کردن در حد 2 ثانیه پیش از چاپ کردن هر خط با استفاده از دستور time.sleep معرفی کردم .تنها دلیل آن هم اینست که برنامه به آرامی اجرا شود (پایتون بر طبق ذاتش خیلی سریع هست) . هنگامی که برنامه در حال اجرا ست ، کلید Ctrl-c را برای قطع کردن/ لغو کردن برنامه فشار بدهید .

روشن هست که استثناء KeyboardInterrupt پرتاپ می شود و برنامه خارج می شود ، اما قبل از اینکه برنامه خارج شود ، عبارات درون بلاک finnaly اجرا می شود و فایل بسته می شود .

خلاصه

ما در مورد نحوه ی استفاده از دستورات try..except وtry..finally بحث داشتیم . ملاحظه نمودیم که چطور نوع استثناء خودمان را ایجاد کنیم و چگونه یک استثناء را برانگیزیم .

در فصل بعدی ، کتابخانه استاندارد پایتون را بررسی خواهیم کرد .

کتاب یک بایت از پایتون.فصل چهاردم .کتابخانه استاندارد پایتون

فهرست مندرجات

کتابخانه استاندارد پایتون

معرفی

کتابخانه استاندارد پایتون با نصب پایتون در دسترس قرار می گیرد . این کتابخانه شامل خیل عظیمی از ماژول های مفید هست . بسیار مهم هست که شما با کتابخانه استاندارد پایتون آشنا بشوید چرا که اگر با ماژول های این کتابخانه آشنا باشید ، بسیاری از مشکلات شما می تواند آسان تر و سریع تر حل شود .

می خواهیم برخی از این ماژول های کتابخانه را که استفاده رایج تری دارند توضیح بدهیم . شما می توانید جزئیات کاملتری برای تمام این ماژول های کتابخانه استاندارد پایتون را در قسمت "Library Refrence"از مستندات پایتون به دست بیاورید . این راهنما نیز با نصب پایتون در اختیار شما قرار می گیرد .

ماژول sys

ماژول sys شامل عملیات مخصوص سیستم است . برای مثال در زیر خواهیم دید که sys.argv ، لیستی شامل آرگومان های خط فرمان هست .

آرگومان های خط فرمان

مثال 14.1. استفاده از sys.argv

```
#!/usr/bin/python
# Filename: cat.py

import sys

def readfile(filename):
'''Print a file to the standard output.'''
f = file(filename)
while True:
line = f.readline()
if len(line) == 0:
break
print line, # notice comma
f.close()
```

```
# Script starts from here
if len(sys.argv) < 2:
print 'No action specified.'
sys.exit()

if sys.argv[1].startswith('--'):
option = sys.argv[1][2:]
# fetch sys.argv[1] but without the first two
characters
if option == 'version':
print 'Version 1.2'
elif option == 'help':
print '''\
This program prints files to the standard output.
Any number of files can be specified.
Options include:
  --version : Prints the version number
  --help    : Display this help'''
else:
print 'Unknown option.'
sys.exit()
else:
for filename in sys.argv[1:]:
readfile(filename)
```

خروجی

```
$ python cat.py
No action specified.

$ python cat.py --help
This program prints files to the standard output.
Any number of files can be specified.
Options include:
  --version : Prints the version number
  --help    : Display this help
```

```
$ python cat.py --version
Version 1.2

$ python cat.py --nonsense
Unknown option.

$ python cat.py poem.txt
Programming is fun
When the work is done
if you wanna make your work also fun:
        use Python!
```

نحوه ی عملکرد این مثال

این برنامه سعی می کند که کار دستور cat در لینوکس/یونیکس را برای کاربران تقلید کند . شما کافیست نام های برخی از فایلهای متنی را در جلوی آن مشخص کنید تا محتوای آنها را بروی خروجی چاپ کند .

وقتی که یک برنامه پایتون اجرا می شود ، یعنی نه در مُد تعاملی ، حداقل یک آیتم در لیست sys.argv قرار می گیرد که نام برنامه ی فعلی ست که در حال اجراست و در خانه sys.argv[0] قرار می گیرد ، چرا که لیست ها در پایتون از 0 شروع می شوند . ما بقی آرگومان ها ی خط فرمان به دنبال این آیتم می آیند .

در این مثال برای اینکه برنامه ی ما کاربر پسند باشد ، یکسری گزینه هایی (options)را برای برنامه فراهم کردیم تا کاربر با استفاده از آن ها اطلاعات بیشتری در مورد برنامه ما کسب کند . از اولین آرگومان برای بررسی اینکه آیا گزینه ایی برای برنامه مشخص شده است یا خیراستفاده می کنیم . اگر گزینه ی version-- به کار رفته بود ، شماره ی نسخه برنامه چاپ می شود . و بطور مشابه ، هنگامیکه گزینه help-- مشخص می شود ، اطلاعاتی کلی از برنامه داده می شود . از تابع sys.exit نیز برای خروج از برنامه در حال اجرا استفاده می شود . برای جزئیات بیشتر help(sys.exit) را ملاحظه نمائید .

هنگامی که گزینه ایی در جلوی اسم برنامه مشخص نشده باشد ، و نام فایل ها به برنامه پاس داده شود ، برنامه به راحتی به همان ترتیبی که در خط فرمان آورده شده است ، هر خط از فایل ها را در خروجی چاپ می کند .

درحاشیه باید گفت که نام cat مختصری برای) concatenate هم زنجیرکردن) هست که در اصل همان کاری ست که برنامه انجام می دهد . که می تواند یک فایل یا دو فایل و یا شماری از فایل ها را به هم بچسباند/ زنجیر کند و در خروجی چاپ کند .

کمی بیشتر درباره sys

دستور sys.version رشته ایی شامل اطلاعاتی در خصوص نسحه پایتونی که نصب کرده اید را به شما نشان می دهد . دستور sys.version_info شامل یک چندتایی است که کار با مقادیر نسخه ی پایتون در برنامه را آسان ترمی سازد .

```
[swaroop@localhost code]$ python
>>> import sys
>>> sys.version
'2.3.4 (#1, Oct 26 2004, 16:42:40) \n[GCC 3.4.2
20041017 (Red Hat 3.4.2-6.fc3)]'
>>> sys.version_info
(2, 3, 4, 'final', 0)
```

ماژول os

این ماژول عملیات مخصوصِ سیستم عامل را ارائه می دهد . این ماژول مخصوصا هنگامی که می خواهید برنامه شما مستقل از platformباشد ، بسیار مهم است . بدین معنی که ، شما را قادر می سازد برنامه هایی بنویسید که هم بروی Linuxو هم بروی Windows بدون کوچک ترین مشکلی و بدون نیاز به تغییراتی کارکند . بعنوان مثالی در این مورد استفاده از متغیر os.sep به جای جداکننده مسیر خاص هر سیستم عامل هست] . مترجم : جداکننده مسیر ، در لینوکس کارکتر / و در ویندوز کاراکتر \ هست [.

برخی از بخش هایی از ماژول os که مفیدتر هستند در زیر فهرست شده است ، اکثر خود توصیف هستند .

- os.name : •شامل رشته ایی ست که platform ایی که در حال استفاده از آن هستید را مشخص می کند . مثل 'nt' برای ویندوز و 'posix' برای کاربران . Linux/unix
- •تابع os.getcwd() ، دایرکتوری جاری را بر می گرداند ، بدین معنی که مسیر دایرکتوری ایی را که برنامه پایتون فعلی در حال کار بروی آن است را برمی گرداند .
- •توابع os.getenv() و os.putenv() به ترتیب برای گرفتن و تنظیم کردن متغیرهای محیطی به کار می روند .
- •تابع os.listdir() نام تمامی فایل ها و دایرکتوری ها را در دایرکتوری مشخص شده ، برمی گرداند .
- •تابع os.remove() برای حذف یک فایل به کار می رود .

- •تابع os.system() برای اجرای یک دستور تحت cmd/shellبه کار می رود .
- os.linesep : •رشته ایی را می دهد که حاوی پایانگر خط در platformفعلی ست . برای مثال ، ویندوز از '\r\n'، لینوکس از '\n'و مکینتاش از '\r'استفاده می کند .
- •تابع os.path.split() نام دایرکتوری و نام فایل از مسیر مشخص شده را بصورت جداگانه در یک چندتایی بر می گرداند .

```
>>> os.path.split('/home/swaroop/byte/code/poem.txt')
('/home/swaroop/byte/code', 'poem.txt')
```

- •توابع os.path.isfile()وos.path.isdir() به ترتیب بررسی می کنند که آیا مسیر مشخص شده به یک فایل اشاره می کند و یا به یک دایرکتوری . مشابه تابع os.path.exist()است که برای بررسی اینکه مسیر مشخص شده واقعا وجود دارد یا خیر به کار می رود .

شما می توانید مستندات استاندارد پایتون را برای جزئیات بیشتر این توابع و متغیرها وارسی کنید . می توانید از دستور help(sys) و یا غیره استفاده کنید .

خلاصه

برخی از عملیات ماژول های sys و os از کتابخانه ی استاندارد پایتون را مشاهده نمودیم . شما می بایست مستندات استاندارد پایتون را برای این ماژول ها و دیگر ماژول ها وارسی کنید .

در فصل بعدی ، جنبه های مختلفی از پایتون را که گشت ما در دنیای پایتون را کامل تر می کند را پوشش خواهیم داد .

کتاب یک بایت از پایتون.فصل پانزدهم.باز هم از پایتون

فهرست مندرجات

باز هم از پایتون

تا بدین جا ، ما جنبه های مختلفی از پایتون را که شما به کار خواهید گرفت پوشش دادیم . دراین فصل ، ما برخی مفاهیم دیگری را که ، دانش ما را از پایتون کامل تر خواهد کرد ، را پوشش می دهیم .

متدهای خاص

متدهای بخصوصی هستند که معنای خاصی در کلاس ها دارند ، مثل متدهای __init__ و __del__ که ما قبلا مفاهیم آن ها را ملاحظه نمودیم .

بطورکلی ، متدهای خاص برای تقلید رفتار بخصوصی به کار گرفته می شوند .برای مثال ، اگر شما بخواهید از عملگر ایندکس x[key] برای کلاس تان استفاده کنید (همان طوری که برای لیست ها و چندتایی ها استفاده می کنید) ، تنها کافی ست که متدی به نام __getitem__() را به کلاس تان اضافه کنید تا این کار انجام شود . اگر به این موضوع فکر کنید متوجه می شوید که این همان کاری ست که که پایتون خودش برای کلاس listانجام داده است !

برخی از متدهای خاص در جدول زیر لیست شده اند . اگر می خواهید درباره ی تمام این متدهای بخصوص اطلاعات کسب کنید ، در راهنمای مرجع پایتون لیست زیادی از آن ها در دسترس است .

جدول 15.1. برخی از متدهای خاص

نام - توضیحات

- __init__(self , ...)این متد پیش از آن که یک شیء بطور جدید ایجاد و برگردانده شود ، فراخوانی می شود . (نقش سازنده کلاس را دارد (
- __del__(self)پیش از تخریب یک شیء فراخوانی می شود. (نقش مخرب کلاس را دارد (

- __str__(self)مواقعی که از دستور print و یا str() برای یک شیء استفاده می کنیم ، فراخوانی می شود .
- __lt__(self,other)مواقعی که عملگر کوچکتر از (<)به کار برده می شود ، فراخوانی می شود . به طور مشابه متدهای خاصی برای مابقی عملگرها هم وجود دارد (مثل+ ، <و غیره . (
- __getitem__(self,key)وقتی که عملگر ایندکس x[key] به کارگرفته می شود ، فراخوانی می شود .
- __len__(self)هنگامیکه تابع توکار len()برای اشیاء دنباله ایی به کار می رود ، فراخوانی می شود.

بلاک های تک دستوری

تا کنون، حتما متوجه شده اید که هر بلوکی از دستورات ، فاصله اش را بوسیله ی سطح تورفته خودش تنظیم می کند . خب ، این مطلب برای بیشتر مواقع درست هست ، اما نه دقیقا صد درصد . اگر بلوکی ازدستورات تنها شامل یک دستور باشد ، می توانید آن را در همان خط مشخص کنید . مثل یک دستور شرطی یا دستور حلقه ایی . مثال زیر می بایست این مطلب را واضح تر نماید .

```
>>> flag = True
>>> if flag: print 'Yes'
...
Yes
```

همان طوری که ملاحظه می کنید ، در اینجا یک تک دستور و نه یک بلوک مجزا استفاده شده است . همچنین می توانید برای کوتاه ترشدن برنامه تان از این حالت استفاده کنید . البته من به شدت به شما پیشنهاد می کنم که نباید از این روش میان بر استفاده کنید ، مگر برای بررسی خطا و غیره . یکی از دلایل اصلی آن هم این هست که ، اگراز تورفتگی معمول استفاده کنید ، خیلی آسان ترمی توان یک دستور دیگر را به آن اضافه نمود.

توجه داشته باشید که هنگامی که مفسر پایتون در مُد تعاملی به کار گرفته می شود ، با تغییر مناسب اعلان خط فرمان ، به شما در وارد کردن دستورات کمک می کند . برای مثالی در این مورد ، بعد از اینکه شما کلمه کلیدی if را وارد کردید ، مفسر پایتون برای نشان دادن اینکه جمله هنوز کامل نشده است اعلان خط نمایش را به ... تغییر می دهد . در این حالت ، وقتی که ما مابقی دستورات را کامل کردیم ، کلید enter را برای تایید این که دستورات ما کامل شده است فشار می دهیم . سپس ، پایتون تمامی دستورات وارد شده را اجرا می کند و اعلان خط فرمان قبلی را بر می گرداند و برای دریافت ورودی بعدی منتظر می ماند.

List Comprehension

List Comprehensionها برای مشتق کردن یک لیست جدید از لیست موجود استفاده می شوند . برای مثال ، شما یک لیستی از اعداد دارید و می خواهید یک لیست مشابه با آن به دست بیاورید ، به طوری که به ازای همه ی اعضای آن لیست اگر هر عضو آن از 2 بزرگتر هستند در 2 ضرب شده باشند ، برای چنین مواردی List Comprehensionمناسب هست .

استفاده از List Comprehension

مثال 15.1. به کارگیری List Comprehension

```
#!/usr/bin/python
# Filename: list_comprehension.py

listone = [2, 3, 4]
listtwo = [2*i for i in listone if i > 2]
print listtwo
```

خروجی

```
$ python list_comprehension.py
[6, 8]
```

نحوه ی عملکرد این مثال

در اینجا ، ما یک لیست جدید را با مشخص کردن محاسبه ایی (2 i*)که باید بسته به شرط تعیین شده if i) (2 <انجام بدهد مشتق می کنیم . توجه کنید که لیست اصلی دست نخورده باقی می ماند . بسیاری از اوقات ، ما از حلقه برای پردازش هر عنصر از لیست استفاده می کنیم ، همچین مواردی می تواند به طرز صریح و فشرده و با صراحت بیشتری بوسیله ی List Comprehensionبه بدست آید .

دریافت چندتایی ها و لیست ها در توابع

روش های بخصوصی برای دریافت پارمترها در یک تابع بعنوان لیست و یا دیکشنری که به ترتیب با استفاده از پیشوندهای * و یا ** مشخص می شوند ، وجود دارد . این مسئله برای زمانی که تعداد متغیرهای آرگومان ها در تابع بدست می آید ، مفید می باشد .

```
>>> def powersum(power, *args):
...     '''Return the sum of each argument raised to
specified power.'''
...     total = 0
...     for i in args:
...             total += pow(i, power)
...     return total
...
>>> powersum(2, 3, 4)
25

>>> powersum(2, 10)
100
```

به دلیل استفاده از پیشوند * در نام متغیر args ، تمامی آرگومان های اضافی ارسالی به تایع در این متغیر args، بعنوان یک tuple ذخیره می شوند . اگر به جای آن از پیشوند ** در نام متغیر استفاده می شد ، پارامترهای اضافی می بایست بصورت جفتِ کلید/مقدار از یک دیکشنری مورد بررسی قرار می گرفتند .

فرم لامدا (lambda)

استفاده از فرم لامدا

مثال 15.2. بکارگیری فرم لامدا

```
#!/usr/bin/python
# Filename: lambda.py

def make_repeater(n):
return lambda s: s * n

twice = make_repeater(2)
```

```
print twice('word')
print twice(5)
```

خروجی

```
$ python lambda.py
wordword
10
```

نحوه ی عملکرد این مثال

در اینجا ، ما از تابع make_repeater برای ایجاد یک شیء تابعی جدید و برگرداندن آن در زمان اجرا استفاده کردیم . که در آن دستور lambda برای ایجاد شیء تابعی به کار گرفته شده است . لازمست که بدنبال پارامتر lambdaیک تک عبارت آورده شود ، که به تنهایی نقش بدنه ی تابع را دارد و مقدار این عبارت با یک تابع جدید برگردانده می شود . توجه کنید که حتی یک دستور print هم نمی تواند در داخل فرم lambda قرار بگیرد ، تنها مجاز به استفاده ازعبارت هستیم .

دستورات exec و eval

دستور exec برای اجرای دستورات پایتون که در یک رشته و یا یک فایل ذخیره شده اند ، استفاده می شود . برای مثال ، ما می توانیم یک رشته شامل کدهای پایتون را در زمان اجرا ایجاد کنیم و سپس این دستورات را با استفاده از دستور exec اجرا کنید . یک مثال ساده در زیر آمده است :

```
>>>exec 'print "Hello World"' Hello World
```

دستور eval برای ارزیابی (evaluate) اعتبار عبارات پایتون که در یک رشته ذخیره شده اند به کار می رود . یک مثال ساده در زیر نشان داده شده است .

```
>>> eval('2*3') 6
```

دستور assert

عبارت assertبرای اذعان به درستی یک چیز به کار می رود . برای مثال ، اگر شما کاملا مطمئن هستید که می بایست حداقل یک عنصر در لیستی که شما در حال استفاده از آن هستید وجود داشته باشد و می خواهید این مسئله را چک کنید بطوری که اگر آن درست نباشد خطایی برخاسته (raise) شود ، برای یک چنین مواردی دستور assert مناسب است . هنگامیکه دستور assert نقض شود ، یک AssertionErrorبرپا (raise)می شود .

```
>>> mylist = ['item']
>>> assert len(mylist) >= 1
>>> mylist.pop()
'item'
>>> assert len(mylist) >= 1
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AssertionError
```

تابع repr

تابع repr برای بدست آوردن نمایش متعارف یک رشته استفاده می شود) . همچنین علامت `` (Backtick) ، که تبدیل (conversion) و یا نقل قول معکوس (reverse quotes)نامیده می شود نیز همین کار را انجام می دهد (.

توجه کنید که برای اکثر اوقات داریم eval(repr(object)) == object :

```
>>> i = []
>>> i.append('item')
>>> `i`
"['item']"
>>> repr(i)
"['item']"
```

بطور کلی ، تابع repr یا جفت علامت backtick برای بدست آوردن یک نمایش قابل چاپ از شیء بدست می آید . شما می توانید آنچه که اشیاء شما برای تابع reprبر می گردانند را با تعریف متد __repr__ در کلاس تان کنترل کنید .

خلاصه

ما برخی دیگر از ویژگی های پایتون را در این فصل و مطمئناً نه تمام آن را پوشش دادیم . به هر حال ، در این مرحله ، بیشتر آنچه که اکثر اوقات با آن در برنامه های تان رو به رو می شوید را پوشش دادیم . که برای شروع برنامه هایی که قصد نوشتن آن را دارید ، مناسب است .

در فصل بعدی ، ما درباره نحوه ی اکتشاف بیشتردر پایتون بحث خواهیم کرد .

کتاب یک بایت از پایتون.فصل شانزدهم.کار بعدی چیست ؟

فهرست مندرجات

کار بعدی چیست ؟

اگر کتاب را تا بدین جا خوانده باشید و نوشتن برنامه های زیادی را تمرین کرده باشید ، می بایست پایتون برای شما آشنا و راحت شده باشد . احتمالا برنامه هایی به زبان پایتون ایجاد کرده اید و سعی کرده اید مهارت برنامه نویسی تان را به خوبی به کار بگیرید واگرهم تا کنون این کار را انجام نداده اید لازمست که این کار را انجام بدهید . اما اکنون حتما می پرسید که "بعد از آن چی ؟ "

خب من فرض می کنم که شما با یک چنین مسئله ایی روبه رو هستید : ساخت یک برنامه مثل "کتابچه ی - آدرس" تحت خط فرمان که بتوان اطلاعاتی را اضافه ، حذف و یا ویرایش نمود و یا اطلاعاتی را برای تماس با

دوستان ، خانواده و یا همکاران ، جستجو نمود و یا همچنین اطلاعاتی از آنها همچون آدرس ایمیل و شماره تلفن را جستجو کرد . این چنین جزئیاتی می بایست ذخیره شوند تا بتوان برای استفاده های بعدی آنها را بازیابی نمود .

اگر درباره ی همچین مسئله ایی فکر کنید ، مشخص هست که با توجه به مباحث گوناگونی که تا بدین جا آموختیم ، حل آن آسان است ، اما اگر هنوز می خواهید درباره ی نحوه ی حل این مسئله بدانید ، در زیر یک راهنمایی برای شما آورده شده است .

راهنمایی . (قرار هست که این را نخوانید .)یک کلاس برای نمایش اطلاعات اشخاص ایجاد می کنید . و از یک دیکشنری برای ذخیره اشیاء یی از نوع اشخاص استفاده می کنید ، به طوری که نام شان بعنوان کلید دیکشنری باشد . سپس از ماژول cPickle برای ذخیره ی این اشیاء بصورت پایدار بروی هارد دیسک استفاده کنید . و درنهایت از متدهای خودکار دیکشنری برای اضافه کردن ، حذف و یا تغییر اطلاعات اشخاص بهره ببرید .

یکبار که همچین کاری را انجام بدهید ، می توانید ادعا کنید که یک برنامه نویس پایتون شده اید . پس اکنون فورا بخاطر این کتاب ارزشمند برای من یک ایمیل بفرستید . ;-) این گزینه اختیاری ست ولی پیشنهاد می شه که حتما انجام بدهید !

درادامه برخی از راه ها برای ادامه سفرتان با پایتون آورده شده است .

نرم افزارهای گرافیکی

کتابخانه های GUI برای پایتون – شما می بایست برای ایجاد برنامه های گرافیکی با استفاده از پایتون از یکی از این کتابخانه ها استفاده کنید . شما می توانید نرم افزارهای گرافیکی همچون IrfanView و یا Kuickshowمخصوص به خودتان را و یا هرچیزی شبیه به آن را بر پایه ی کتابخانه های GUI ایی که به پایتون bind شده اند ایجاد کنید . ، منظور از bindشدن آنست که به شما اجازه می دهد برنامه ها ی تان را در پایتون بنویسید و سپس از کتابخانه هایی که خودشان به زبان C و یا C++ یا دیگر زبان ها نوشته شده اند ، استفاده کنید .

انتخاب های زیادی برای استفاده از یک GUI برای پایتون وجود دارد :

- PyQt : •این کتابخانه ، bindشده ایی از جعبه ابزار Qt برای پایتون است ، کتابخانه ایی که پایه و اساس KDE بروی آن ساخته شده است Qt . بینهایت راحت است و برای استفاده بسیار قدرتمندست

بخصوص با استفاده از Qt Designer و مستندات متحیر کننده ی . Qtشما می توانید بصورت آزاد از آن برای لینوکس استفاده کنید ولی برای استفاده ی آن در ویندوز می بایست پول پرداخت کنید و همچنین PyQt در صورتیکه بخواهید نرم افزارهای آزاد در لینوکس /یونیکس ایجاد کنید آزاد هست (GPL'ed)ولی در صورتیکه بخواهید نرم افزارهای تجاری ایجاد کنید می بایست پول پرداخت کنید. یک منبع خوب برای PyQtکتاب "برنامه نویسی GUI با پایتون : نسخه "QT ، GUI") Programming with Python: Qt Edition")می باشد . آدرس [۱] . [برای جزئیات بیشتردرباره ی این کتابخانه به آدرس وب سایت اصلی آن به نشانی [۲] [مراجعه نمائید .

- . PyGTK •این کتابخانه ، bindشده ایی از جعبه ابزار GTk+ برای پایتون می باشد . کتابخانه ایی که پایه و اساس Gnome بر اساس آن ساخته شده است GTK+ . خصلت های بخصوص زیادی دارد اما وقتی یکبار با آن راحت شوید ، می توانید برنامه های کاربردیِ GUIسریعی را ایجاد کنید . استفاده از طراح واسط گرافیکی Glade اجتناب ناپذیرست . مستندات این کتابخانه هنوز هم در حال پیشرفت و اصلاح شدن است Gtk+ . به خوبی در لینوکس کار می کند ، اما انتقال آن به ویندوزهنوز کامل نشده است . شما می توانید هم نرم افزارهای آزاد و هم نرم افزارهای تجاری را با استفاده از GTK+ایجاد کنید . وب سایت رسمی آن را [۳] [برای جزئیات بیشترملاحظه نمائید .

- . WxPython •این کتابخانه ، bindشده ایی از جعبه ابزار wxWidgetsبرای پایتون می باشد . wxPythonزمان زیادی را برای یادگیری به خود اختصاص می دهد . به هر حال قبلیت انتقال بالایی دارد و در ویندوز ، لینوکس ، مکینتاش و حتی platform های جاسازی شده (embedded platforms)هم قابل اجراست IDE . های بسیاری برای wxPython آماده هست که هم برای طراحی GUIمناسب هست ، همچون [SPE (Stani's Python Editor) ۴] [وهم برای ساختن GUI، همچون] wxGlade ۵ . [شما می توانید هم نرم افزارهای آزاد و هم نرم افزارهای تجاری را با استفاده از این کتابخانه ایجاد کنید . برای جزئیات بیشتر به وب سایت رسمی آن به آدرس [۶]] مراجعه نمائید .

- .TkInter •این کتابخانه ،یکی از قدیمی ترین جعبه ابزارهای GUI در حال حاضر هست . اگر شما با IDLEکار کرده باشید ، در واقع یک برنامه که با TkInterساخته شده است را ملاحظه نموده اید . مستندات برای TkInter در وب سایت PythonWare.orgبه آدرس [۷] [بصورت جامع موجود می باشد TkInter . دارای قابلیت انتقال است و بروی لینوکس/یونیکس و همین طور ویندوز کار می کند . نکته با اهمیت اینست که ، TkInterبعنوان بخشی از توزیع استاندارد پایتون است .

- •برای انتخاب های بیشتر در این زمینه ، به صفحه ی ویکی برنامه نویسی واسط گرافیکی در python.orgبه آدرس زیر مرجع کنید .

خلاصه ایی از ابزارهای GUI

متاسفانه ، یک ابزار GUI همه پذیر برای پاتیون وجود ندارد . من حدس می زنم که شما بسته به شرایط تان یکی از ابزارهای بالا را انتخاب می کنید . اولین فاکتور این هست که آیا مایلید برای استفاده از ابزارهای GUI پول پرداخت کنید . دومین فاکتور آنست که آیا می خواهید برنامه تان بروی لینوکس و یا ویندوز و یا بروی هر دو اجرا شود . سومین فاکتور این هست که شما یک کاربر KDE و یا GNOME لینوکس هستید .

فصل های آتی

من قصد دارم که یکی دو فصل برای این کتاب در مورد برنامه نویسی GUI بنویسم . من احتمالا wxPython را بعنوان جعبه ابزار گرافیکی انتخاب می کنم .اگر شما می خواهید نظرتون را درباره این موضوع اعلام کنید ، لطفا به میل لیست byte-of-python به آدرس [۹] ملحق شوید ، در اینجا خواننده ها با من درباره ی اصلاح و پیشرفت این کتاب صحبت می کنند .

کنکاشی بیشتر

- •کتابخانه ی استاندارد پایتون ، کتابخانه ایی غنی و بسیار گسترده است . بیشتر اوقات ، این کتابخانه در درون خودش آنچه که شما به دنبال آن هستید را دارد . این مطلب در فلسفه ی پایتون بعنوان " batteries include"اشاره می شود . من شدیدا پشنهاد می کنم که پیش از آنکه برنامه های بزرگی را آغاز کنید بروی مستندات استاندارد پایتون به آدرس [۱۰] ، یک مرور کلی داشته باشید .

- •Python.org [۱۱] -، وب سایت رسمی زبان برنامه نویسی پایتون . شما می توانید آخرین نسخه زبان پایتون و مفسر آن را در اینجا پیدا کنید .همچنین میل لیست های مختلفی که به بحث درباره ی جنبه های گوناگونی از پایتون می پردازد ، در آنجا قرار دارد .

- •comp.lang.python یک شبکه ی گروه خبری ست که درباره ی این زبان بحث می کنند . شما می توانید پرسش ها و پاسخ های خودتان را در این گروه مطرح کنید . می توانید از طریق گروهای گوگل به بصورت آنلاین به آن دسترسی پیدا کنید [۱۲] . یا به میل لیست آن به آدرس [۱۳] ملحق شوید که mirror ایی برای گروه خبری هست .

• Python Cookbook به آدرس [۱۴] مجموعه ایی بسیار ارزشمند از رهنما ها و نکته هایی برای حل انواعی از مسائل بخصوص بوسیله ی پایتون هست ، که می بایست توسط هر کاربر پایتون کار خوانده شود .

• Charming Python به آدرس [۱۵] که شامل سری های عالی از مقالات پایتون توسط آقای David Mertzمی باشد .

• Dive Into Python به آدرس [۱۶] کتاب بسیار خوبی برای آموختن زبان برنامه نویسی پایتون است . اگر در حال به پایان رساندن کتاب حاضر هستید ، پیشنهاد جدی من به شما این است که بعد از این کتاب به سراغ کتاب 'Dive Into Python' بروید . این کتاب دامنه وسیعی از موضوعات را همچون پردازش XML ، تست Unitو برنامه نویسی عملی را پوشش می دهد .

• Jython به آدرس [۱۷] ، که یک پیاده سازی از مفسر پایتون به زبان جاوا می باشد . این بدان معنی ست که شما می توانید برنامه هایی را در زبان پایتون بنویسید و از کتابخانه های جاوا نیز به خوبی بهره ببریدJython . یک نرم افزار کامل و پایدار است . اگر که شما یک برنامه نویس جاوا هستید ، من به شدت به شما توصیه می کنم که Jyhonرا امتحان کنید .

- IronPython به آدرس [۱۸] ، که یک پیاده سازی از مفسر پایتون به زبان C# است و می تواند بروی platformهای .NET/Mono/DotGNU اجرا بشود . بدین معنی که شما می توانید برنامه هایی در پایتون بنویسید که از کتابخانه های .NET و دیگر کتابخانه های ارائه شده با این سه platformبه درستی استفاده کند IronPython ! هنوز یک نرم افزار pre-alpha هست و اکنون تنها برای آزمایش مناسب است Jim Hugunin . کسی که IronPython را نوشته است در حال ملحق شدن به MicroSoft هست و قرار هست بروی ایجاد یک نسخه ی کامل و استوار آن در آینده کار کند .

- Lython به آدرس [۱۹] که یک Lispاولیه برای زبان پایتون است . که شبیه به Lisp معمولی ست و بطور مستقیم بایت کدهای پایتون را کامپایل می کند ، بدین معنی که قابلیت تبادل (interoperability)با کدهای معمول پایتون را دارد .

- •منابع اصلی بسیاری برای پایتون هست . یکی از آنها لینک های روزانه پایتون هست ! به آدرس [۲۰] که می تواند اطلاعات شما را با آخرین وقایع پایتون به روز کند . همچنین Vaults of Parnassus به آدرس [۲۱] ، و ONLamp.com Python DevCenter به ادرس [۲۲] ، و dirtSimple.orgبه آدرس [۲۳] ، و نکات پایتون به آدرس [۲۴] و بسیار بسیار منابعی از این قبیل موجودست .

خلاصه

اکنون به پایان این کتاب رسیدیم ، اما همان طوری که گفته شد ، این آغاز یک پایان است . ! اکنون شما یک کاربرمشتاق پایتون هستید و شک ندارید که برای حل بسیاری از مسائل با پایتون آماده هستید . شما می توانید شروع کنید به اتومات کردن کامپیوترتان تا هر گونه از کارهای غیرقابل تصور را انجام بدهد ، یا بازی های دلخواه خودتان را بنویسید و خیلی خیلی بیشتر، خب پس شروع کنید !