زبان برنامه نویسی

پایتون

سجاد رضایی
دانشگاه صنعتی همدان
sajjad.rezaee@yahoo.com


بسم الله الرحمن الرحیم

فهرست مطالب

مقدمه

زبانی که ما قصد داریم به شما معرفی کنیم زبانی است بسیار سطح بالا: پایتون؛ پایتون زبان پیشنهادی ما به شماست که به نظر اکثر برنامه نویسان توانسته است به بهترین شکل ممکن سه قابلیت اصلی یک زبان سطح بالا یعنی سهولت بیشتر، فهم راحت تر و توسعه ی سریع تر را پیاده سازی کند. مقاله ای که پیش روی شماست قرار است شما را با این زبان آشنا نماید. امروزه اکثر برنامه نویسان در حال جذب شدن به چنین زبان هایی هستند، چرا که متوجه شده اند دنیای زبان های برنامه نویسی به عصرِ جدیدی پا گذاشته است؛ عصرِ زبان های تفسیری و سطح بالا که سعی دارند برنامه نویسی را برای برنامه نویسان دلچسب تر کنند... عصر زبان های خوش ساخت... عصرِ پایتون!

معرفی پایتون

پایتون یک زبان برنامه نویسی تفسیری ، داینامیک و شی گرا می باشد که می توان از آن در محدوده ی وسیعی از نرم افزار ها و تکنولوژی ها بهره برد. این زبان برنامه نویسی روش های بسیار قدرتمند و حرفه ای را برای کار با زبانها و ابزار های مختلف را با آسانی هر چه تمام تر فراهم می کند. بدین منظور این زبان دارای کتابخانه هایی بسیار گسترده می باشد که یادگیری و استفاده از آنها در عرض چند روز ممکن می باشد ! پایتون همچنین یک زبان بر اساس مجوز های نرم افزار آزاد و اپن سورس می باشد. کد های نوشته شده در این زبان در محدوده ای وسیع از پلتفرم ها چون لینوکس ، ویندوز ، مک ، و حتی گوشی های موبایل و... قابل اجرا می باشد. هم اکنون پایتون در شرکت ها و سازمانهای بزرگی چون ناسا ، گوگل ، یاهو و... بصورت گسترده مورد استفاده قرار می گیرد.

پایتون زبان برنامه‌نویسی تفسیری و سطح بالا ، شی‌گرا و یک زبان برنامه‌نویسی تفسیری سمت سرور قدرتمند است که توسط گیدو ون روسوم در سال ۱۹۹۰ ساخته شد. این زبان در ویژگی‌ها شبیه پرل، روبی (رابی)، اسکیم، اسمال‌تاک و تی‌سی‌ال است و از مدیریت خودکار حافظه استفاده می‌کند.

پایتون به شکل پروژه‌ای متن باز توسعه یافته است و توسط بنیاد نرم‌افزار پایتون مدیریت می‌گردد.

نسخهٔ ۲.۴.۲ این زبان در تاریخ ۲۸ سپتامبر ۲۰۰۵ منتشر شد.

تاریخچه

نوشتن این زبان برای اولین بار در سال ۱۹۹۰ توسط فردی به نام گویدو ون رسوم(Guido van Rossum) کلید خورد. پایتون در یک محیط آموزشی، ایجاد و توسعه یافته است. یعنی در کریسمس سال ۱۹۹۸ میلادی در موسسه ملی تحقیقات ریاضی و رایانه (CWI) شهر آمستردام. در آن زمان گویدو یک محقق در CWI بود و در زمان بیکاری خود بر روی پروژه شخصی خود یعنی پایتون کار می کرد. اولین نسخه عمومی از پایتون در ماه فوریه سال ۱۹۹۱ منتشر شد. برای مدتی نسبتاً طولانی پایتون توسط موسسه ملی تحقیقات و ابتکارات (CNRI) واقع در رستون ایالات متحده امریکا توسعه می‌یافت. تا اینکه در سال ۲۰۰۰ تیم توسعه دهنده پایتون به آزمایشگاه های پایتون منتقل شدند. نام پایتون از برنامه مورد علاقه سازنده آن یعنی مونتی پایتون که یک برنامه کمدی انگلیسی بود گرفته شده است.

ویژگی ها

۱. شی گرایی

پایتون یک زبان برنامه‌نویسی شی‌گرا است و از ویژگی های پیشرفته‌ای چون وراثت، چند شکلی، سر بار گذاری عملگر و... پشتیبانی می کند. یک از ویژگی‌های پایتون که لقب چسب را برای پایتون به ارمغان آورده است، امکان استفاده از کد ها و کلاس‌های نوشته شده در زبان های دیگری چون سی‌پلاس‌پلاس و جاوا است که در حقیقت کار چسباندن قطعات کد جدا و نوشتن بدنه اصلی به عهده پایتون است.

۲. آزاد بودن

پایتون یک زبان برنامه‌نویسی آزاد و بازمتن است. می‌توانید متن آن و خود برنامه را از اینترنت دریافت یا در توسعه آن همکاری کنید.

۳. قابلیت حمل

چون پایتون با زبان قابل حمل سی نوشته شده می تواند به صورت مجازی بر روی هر پردازش‌گری همگردانی و اجرا شود. ماشین مجازی (مفسر پایتون) متن برنامه را خوانده و هم‌زمان تفسیر کرده و اجرا می کند. پس شما می توانید یک برنامه را در ویندوز بنویسید و سپس بدون تغییر روی لینوکس یا مکینتاش یا هر سیستم عامل و سخت‌افزار دیگری که پایتون روی آن نصب باشد اجرا کنید.

٤. قدرتمند بودن

پایتون زبانی چند رگه است که از زبان‌های برنامه‌نویسی تفسیری (برای مثال : تی‌سی‌ال، اسکیم، پرل) و زبان‌های سیستمی (برای مثال: سی‌پلاس‌پلاس، سی و جاوا) مشتق شده. بنابراین تمام سادگی و راحتی کار زبان‌های برنامه‌نویسی تفسیری و ویژگی‌ها و قدرت زبانهای سطح پایین را داراست.

۵. درونی سازی و گسترش

این ویژگی یکی از پرکاربرد ترین و قوی ترین ویژگی های پایتون می باشد. شما می توانید قطعه از کد را در زبانی چون سی‌پلاس‌پلاس، سی و جاوا نوشته سپس از آن در برنامه نوشته شده با پایتون استفاده کنید. و یا می توان از توابع کتابخانه‌ای و کامپوننت هایی چون API COM استفاده کرد. البته نوع این نوع برنامه نویسی (ماژول) با برنامه نویسی معمولی هر زبان متفاوت می باشد. می توان از کد های پایتون در زبانهای دیگر نیز استفاده کرد (درونی سازی).

٦. سهولت یادگیری و استفاده

بی شک و حداقل از نظر بسیاری از برنامه نویسان پایتون این زبان یکی از آسان ترین زبان ها برای یادگیری و استفاده می باشد و از آن به عنوان یک زبان سریع برنامه نویسی یاد می کنند. این زبان نیازی به کامپایلر ندارد و شما مستقیما می توانید پس از نوشتن کد و با یک دستور آن را اجرا کنید. دستورات این زبان بسیار

نزدیک به زبان انسان می باشد. برای مثال برنامه Hello World را که اولین برنامه ساده مـی باشـد را در دو زبان سی و پایتون مقایسه کنید:

Hello World در سی

```
#include <stdio.h>
int main()
{
      printf(" Hello world ");
      return 0;
}
```

Hello World در پایتون

```
print "Hello World !!"
```

برنامه های نوشته شده با پایتون

- بیتتورنت (BitTorrent) : نرم افزاری برای جستجوی فایل های به اشتراک گذاشته شده و...
- بلندر (Blender) : یک نرم افزار ۳ بعدی و اپن سورس بسیار معروف
- چندلر (Chandler) : مدیر اطلاعات شخصی شامل تقویم ، میل ، کار های روزلنه ، یادداشت ها و...
- Civilization IV : یک گیم کامپیوتری بر مبنای **پایتون** کـه از boost.python استفاده می کند
- میلمن (Mailman) : یکی از معروفترین نرم افزار های مرتبط با ایمیل
- Kombilo : مدیر پایگاه داده و مرورگرگیم های go
- موین موین (MoinMoin) : یکی از قدرتمندترین و معروفترین ویکی های موجود
- پلون (Plone) : یک ابزار مدیریتی محتوایی اپن سورس ، قدرتمند و کاربر پسند
- پورتاژ (Portage) : قلب توزیع جنتو. یک مدیر بسط های سیستم عامل لینوکس
- زوپ (zope) : یک پلاتفورم شی گرای مبتنی بر وب. زوپ شامل یک سرور نرم افـزار بـه همـراه پایگاه داده شی گرا و یک رابط مدیریتی درونی مبتنی بر وب می باشد
- اس پی ای (SPE) : یک IDE رایگان ، اپن سورس برای سیستم عامل های ویـنـدوز ، لینـوکس ، مک که از wxGlade(طراحـی رابـط کـاربر),PyChecker (دکتـر کـد) و (Blender3D) پشتیبانی می کند.

نکته: در مفسر پایتون می توان دستورات را خط به خط اجرا نمود. با زدن کلید Enter پـس از هـر دسـتور، آن دستور اجرا می شود. در مورد متغیرها نیز با نوشتن نـام متغیـر و فـشردن کلیـد Enter مقـدار آن تـابع نمایش داده می شود. و علامت <<< را مفسر پایتون در **ابتدای دستور** قرار می دهد.

روش های اجرای کدهای پایتون

قبلا ما از پایتون به عنوان یک زبان برنامه نویسی یاد کردیم. اما پایتون نام یک بسته نرم افزاری به نام مفسر نیز هست که کار اصلی آن اجرای برنامه می باشد. مفسر سطر به سطر کد برنامه (Python) را خوانده و همزمان اجرا می کند. در حالت کلی یک برنامه پایتون به چهار طریق اجرا می گردد :

- به حالت محاوره ای
- به عنوان ماژول پایتون
- به عنوان فایل اسکریپت unix
- متد های خاص یک سیستم عامل

۱. اجرای کد به حالت محاوره ای

حالت محاوره ای راحت ترین روش اجرای کد پایتون می باشد. کافیست عبارت پایتون را در ترمینال وارد کنید :

```
# python
```

و سپس دستورات خود را به صورت محاوره ای وارد کنید :

```
>>> print "Hello world!"
Hello world!
>>>
```

برای خروج Ctrl + D را بفشارید. (در بعضی سیستم ها Ctrl + Z)

۲. اجرا به عنوان ماژول پایتون

در روش دوم کد های پایتون داخل یک فایل نوشته می شد. در حالت قبلی اگر اشتباهی در کد نوشته شده موجود باشد باید همه کد های وارد شده را از اول وارد کنید. و نیز برنامه نوشته شده ذخیره نمی گردد و با تمام شدن اجرای آن در حقیقت از بین می رود. هدف ما از نوشتن نرم افزار امکان استفاده مجدد از آن می باشد. پس کد مورد نظر خود را در داخل یک فایل نوشته و با پسوند py ذخیره می نمائیم. این فایل می تواند توسط هر نرم افزار ویرایشگری ایجاد گردد. از ساده ترین برنامه تحت ترمینال vi تا IDE (محیط های برنامه نویسی) حرفه ای و پیشرفته. برای مثال کد زیر را در داخل یک فایل نوشته و با نام test1.py یا هر نام دیگری ذخیره کنید. نحوه اجرای این فایل نیز می تواند متفاوت باشد.

```
import sys
print sys.argv
```

در حالت معمول و ساده می توان این برنامه نوشته شده را از طریق ترمینال اجرا نمود.

```
# python test1.py -I eggs -o bacon
['test1.py', '-I', 'eggs', '-o', 'bacon']
```

این برنامه بسیار کوچک که با نام test1.py ذخیره شده است آرگومان های خط فرمان را به صورت یک لیست به نمایش می گذارد. اجرا آن در سیستم عامل دیگر نیز مشابه هست. برای مثال برای اجرای این فایل در سیستم عامل ویندوز دستور زیر را وارد کنید :

```
C:\book\tests> python test1.py -i eggs -o bacon
['test1.py', '-i', 'eggs', '-o', 'bacon']
```

۳. به عنوان فایل اسکریپت Unix

حالت سوم اجرا در حال اسکریپت می باشد. این نوع فایل متنی حالت اجرایی دارد و مشخصه آن اولین خط آن می باشد. این روش در سیستم عامل های مبتنی بر لینوکس قابل استفاده است و با مقدار دهی مجوز اجرایی (x) به فایل مورد نظر امکان اجرا شدن مستقیم آن را محیا می سازد. در سیستم عامل های دیگر چون ویندوز هم چنین کاری ممکن می باشد. یک مثال ساده :

```
#!/usr/bin/env python
print 'The Bright Side of Life...' # comment
```

دقت داشته باشید که اولین خط این برنامه کامنت (توضیحات) نمی باشد. این سطر به این معنی است که کل کد ادامه فایل را با برنامه ای که آدرس آن پس از عبارت !# آمده فرستاده و به اجرا در می آید. پس این خط آدرس فایل اجرایی مفسر پایتون به همراه علامت های !# است. سپس با استفاده از دستور chmod یا از طریق گرافیکی این فایل را به حالت اجرایی تبدیل کرده و اجرا کنید :

```
# chmod +x test2.py
#./test2.py
The Bright Side of Life...
```

روش های ذکر شده مهمترین و پرکاربرد ترین روش های اجرای کد پایتون می باشند. ولی روش های پیشرفته تری هم برای اجرای کدهای پایتون وجود دارد که از ذکر آن ها خودداری می شود.

٤. متد های خاص یک سیستم عامل

کدهای پایتون در سیستم عامل های مختلف ممکن است به صورت های متفاوتی اجرا شوند، برای مثال در سیستم عامل ویندوز شما می توانید روی فایل با پسوند Py دوبار کلیک کرده و آن را مانند دیگر فایلهای ویندوز اجرا کنید. (البته باید مفسر پایتون نصب شده باشد.)

انواع داده اولیه

متغیر ها

یکی از مهمترین و اساسی ترین جنبه و مزیت زبانهای برنامه نویسی داشتن توانایی کار با **متغیر ها** می باشد. متغیر نامی است که به یک **مقدار** نسبت داده می شود. در اکثر زبان ها برای استفاده از متغیرها باید ابتدا آنها را تعریف یا ایجاد کنید. و نیز معمولا هنگام تعریف هر متغیر نوع آن را که تا آخر عمر متغیر ثابت خواهد ماند را مشخص کنید. در زبان پایتون برای تعریف یک متغیر کافی است آن را مقداری دهی کنید. یعنی نیازی به تعریف جداگانه نمی باشد. نکته بعدی این که در زبان پایتون ابزار مدیریتی داینامیک و بسیار قوی برای متغیر پیاده سازی شده است که امکانات بسیاری را فراهم می کند. برای مثال نوع متغیر در پایتون می تواند در طول عمر آن متغیر یا در طول اجرای برنامه به تعداد دلخواه تغییر یابد.

(نکته: در مثال ها، از تابع `print` برای چاپ یک مقدار یک متغیر یا ثابت استفاده می شود)

نوع داده منطقی(Boolean)

متغیرهای منطقی `logicalVariable` می توانند حامل یکی از دو مقدار درست(`True`) یا غلط(`False`) باشند:

```
>>> b1 = True
>>> b2 = False
```

حاصل عبارات منطقی نیز به همین صورت می باشد:

```
>>> 2>1
True
```

اعداد صحیح کوتاه (Plain):

این نوع، اعداد صحیح از ۲۱۴۷۴۸۳۶۴۸- تا ۲۱۴۷۴۸۳۶۴۷ را در بر می گیرد:

```
>>> I1 = 33233
>>> I2 = -36
```

اعداد صحیح بلند(Long):

این نوع، اعداد صحیح را تا جایی که حافظه اجازه دهد در خود نگهداری می کنند:

```
>>> L1 = 332233223322
>>> L2 = -332233223322
```

اعداد اعشاری (Float)

این نوع داده می تواند اعداد اعشاری را در خود ذخیره کند.

```
>>> pi = 3.14159
```

اعداد مختلط (Complex)

با استفاده این نوع داده به راحتی می توان با متغیر های مختلط کار کرد:

```
>>> z1 = 33+3j
>>> z2 = 22+2j
>>> z3 = z1+z2
>>> print z3
(66+5j)
```

رشته ها (String)

این نوع داده می تواند یک رشته را در خود ذخیره کند. در این نوع شما نمی توانید به طور مستیقم کاراکترها را تغییر دهید:

```
>>> string1 = "first"
>>> string2 = "second"
>>> string1 + string2
'firstsecond'
```

Unicodes

این نوع داده همانند رشته عمل می کند، با این تفاوت که رشته ها را با فرمت Unicode را در خود ذخیره می کند.

```
>>> uString1 = "یونیکد"
>>> print(uString1)
یونیکد
```

متغیر، نامی برای یک مقدار

در پایتون نوع داده در زمان اجرا مشخص می شود و نیازی به اعلان آن قبل از اجرا نیست. در پایتون تمامی داده ها شی (object) می باشند.برای مثال عدد ۳ در پایتون یک شیء را در حافظه مشخص می کند و متغیرها نیز نام هایی هستند که به اشیاء داخل حافظه اشاره می کنند. شاید این سوال پیش بیاید که چگونه مفسر پایتون می تواند نوع داده ها را بدون اعلان اولیه تشخیص دهد!. پاسخ:

```
name1='Francesco Totti'
name2='Paolo Maldini'
```

وقتی در پایتون انتساب هایی به صورت بالا صورت می گیرند مراحل زیر به ترتیب انجام می شوند:

۱. شیء Francesco Totti که مقداری از نوع رشته است به وجود می آید.

۲. متغیرهای name2 , name1 درست می شود.(اگر قبلا درست نشده باشد)

۳. متغیر name2 , name1 به شیء مورد نظر اشاره می کند.

شیء موردنظر باید نوع خود را بداند، برای همین منظور هر object در پایتون دارای دو فیلد هدر (header field) به نام های طراح نوع (type designator) و شمارنده ی مراجعات (reference counter) می باشد، که طراح نوع، نوع داده را در خود ذخیره می کند و شمارنده ی مراجعات تعداد رجوع های متغیرهای متفاوت به شیء را مشخص می کند.

** برای مثال بالا type designator=str و reference counter=1

با داشتن این فیلد ها در زمان اجرا مفسر می تواند نوع داده را تشخیص دهد.

با استفاده از تابع type شما می توانید نوع یک متغیر را مشخص کنید:

```
>>> s = "string"
>>> i = 32
>>> type(s)
<type 'str'>
>>> type(i)
<type 'int'>
```

مدیریت حافظه ی خودکار

reference counter کار زباله روبی (garbage collection) را بسیار آسان می سازد. به این صورت که وقتی reference counter مساوی صفر میشود زباله روب متوجه می شود که هیچ ارجاعی به شیء مورد نظر وجود ندارد, بنابراین شی ء را از حافظه پاک می کند. هم چنین از به وجود آمدن Dangling Pointer ها نیز جلوگیری می کند.

نام گذاری متغیر ها

معمولا برنامه نویسان برای نام گذاری متغیر ها از اسامی استفاده می کنند که نشان دهنده کاربرد و حتی نوع متغیر می باشد. اسامی متغیر ها می تواند با طول دلخواه باشد. در نام گذاری متغیر ها استفاده از حرف و عدد مجاز است. بیاد داشته باشید که در برخی از زبانها از جمله پایتون حروف برزگ با کوچک یکسان نیستند. بنابراین متغیری با نام Joker با متغیر joker یکسان نیست و این دو ممکن است مقادیر و حتی نوع های مختلفی داشته باشند. در نام گذاری متغیر ها می توان از کاراکتر زیر خط (_) نیز استفاده کرد و معمولا در اسامی که از بیش از یک کلمه تشکیل شده باشند به کار می رود.

همچون تمام زبانهای برنامه نویسی پایتون هم قوانینی برای نام گذاری متغیر ها دارد که باید هنگام نام گذاری به آنها دقت کنید. برخی از این قوانین شامل موارد زیر است :

- اسم متغیر باید با یک حرف شروع شود.
- در نام گذاری مجاز به استفاده از کاراکتر های خاص چون $٪^#/ و... نیستید.
- در نام گذاری متغیر ها نباید از اسامی رزرو شده زبان برنامه نویسی استفاده کرد. برای مثال استفاده از اسامی چون if , for , class و... مجاز نمی باشد.

همانطور که گفتیم استفاده از اسامی رزور شده در پایتون مجاز نمی باشد. در صورت استفاده از این اسامی یا هر یک از موارد بالا برنامه از روند عادی خود خارج شده و متوقف می گردد و در اصطلاح یک خطای نحوی(مربوط به نحوه نوشتن و یا املای دستورات برنامه) (syntax error) رخ می دهد.

بیاد داشته باشید ممکن است اسمی که برای یک متغیر انتخاب می کنید تکراری نیز باشد! در این حالت شما خطایی دریافت نخواهید کرد اما برنامه نتایجی اشتباه محاسبه خواهد کرد یا روند اجرای آن به مشکل بر خواهد خورد.

زبان پایتون دارای ۲۸ اسم رزرو شده می باشد. این اسامی یا keyword ها تشکیل دهنده ساختار کلی هر زبان برنامه نویسی می باشند.

اسامی رزرو شده در پایتون

and	continue	else	for	import	not	raise
assert	def	except	from	in	or	return
break	del	exec	global	is	pass	try
class	elif	finally	if	lambda	print	while

ساختمان داده

در اصطلاح کامپیوتری، ساختمان داده به روشهایی از ذخیره اطلاعات گفته می شود که برای استفاده بهینه از اطلاعات ذخیره شده اتخاذ می شود. غالباً انتخاب یک ساختمان داده موجب ایجاد الگوریتم های متناسب با آن خواهد شد که این دو در کنار هم موجب افزایش سرعت انجام یک وظیفه یا کاهش مصرف حافظه برای پردازش داده می شود؛ سنگ بنای ساختمان های داده انواع داده و اشاره گرهای گوناگون است. که با توجه به چگونگی تعریف کاربرد آنها در هر زبان برنامه نویسی پیاده سازی آنها متفاوت خواهد بود. ما اکنون به پیاده سازی ساختمان های داده نمی پردازیم بلکه به توضیح انواع داده موجود در زبان پایتون می پردازیم؛ به دلیل سطح بالای این زبان انواع داده موجود در آن دارای ساختار پیچیده ای هستند که باعث شد ما از این انواع به عنوان ساختمان های داده یاد کنیم.

در زبان های سطح پایین تر که اکثر آنها از پایه های پایتون به حساب می آیند انواع داده پیش فرض انواعی ابتدایی هستند که در زبان اسمبلی نیز قابل تعریف هستند. مثلاً در زبان C از انواع `int , char,` `float, double, long ,short` استفاده می شود که همه آنها دارای خاصیتی مشترک هستند و این خاصیت این است که بر روی پردازنده به طور مستقیم دارای دستور العمل هایی هستند که می توان با آنها کار کرد. همچنین برای ایجاد یک زنجیره(آرایه) از این انواع از علامت "[]" استفاده می شد، ولی از این انواع داده غیر از عملیات ریاضی کاری بر نمی آید ، مگر اینکه از آنها با قرار دادهای خاصی ساختمان داده هایی بسازیم.

انواع ساختمان داده در پایتون

۱. یکی از مهمترین و پرکاربرد ترین این ساختمان های داده رشته های کاراکتری می باشند که در واقع یک زنجیره (`Sequence`) از بایت ها می باشند که در کار با ورودی ها، خروجی ها و ارتباطات گوناگون نقش مهمی ایفا می کنند، زیرا یکی از راههای محدود فهم انسان از دنیای کامپیوتر ارتباط متنی با این جهان می باشد.

۲. دیگر ساختمان داده ای مهم در این زبان لیست ها (آرایه ها) هستند. در واقع این نوع داده یک نوع بسیار پیشرفته از آرایه های زبانهای سطح پایین است که علاوه بر خاصیت اندیس پذیری ، خاصیت تغییر اندازه و نگهداری انواع داده را بطور هم زمان دارا می باشد.

۳. چند تایی های مرتب (`Tuple`)در پایتون نوعی از داده با شباهت هایی به لیست می باشد که در بخش مربوطه به تفاوت ها و شباهت های این دو نوع خواهیم پرداخت.

۴. `Dictionary` یک نوع دیگر از ساختمان داده در پایتون است که شبیه به آرایه عمل می کند، با این تفاوت که اندیس ها در این نوع اجباری نبوده و می توانند هر نوع داده تغییر ناپذیر باشند.

۵. مجموعه (`Set`) نوعی دیگر از ساختمان داده چندتایی در پایتون می باشد که مطابق با تعریف ریاضی مجموعه ها عمل می نماید و اعمال خاص مجموعه ها برای آن تعریف شده است.

ساختمان داده های دیگر

در جملات فوق الذکر مشاهده کردید که ما با تعداد محدودی ساختمان داده روبرو هستیم. اما ما مجبور نیستیم که با این ساختمان داده ها بسوزیم و بسازیم. بلکه این ساختمان های داده سنگ بنای چندین

ساختمان داده دیگر هستند که هر کدام کاربرد و پیچیدگی های خاص خود را دارند از آن جمله می توان موارد زیر را نام برد:

۱. لیست های پیوندی
   ۱. یک طرفه
   ۲. دوطرفه
   ۳. حلقوی
۲. صف ها
   ۱. صف های دو طرفه
   ۲. صف های با اولویت
۳. درخت ها
   ۱. دودویی
   ۲. دودویی جستجو
   ۳. درخت های دو–سه
۴. heap
   ۱. Heap
   ۲. MinMax Heap

و...

لیست ها

لیست یک نوع داده ی چند قسمتی است. لیست داده ای است که می توان یک یا چند داده از هر نوع را در آن قرار داد.

```
>>>L = [ 'mahdy' , 20 , 1.2 ]
```

دسترسی به داده های درون لیست

برخلاف Dictionary (این نوع داده در قسمت های بعد تعریف می شود)، در لیست دیگر اندیس توسط برنامه نویس مشخص نمی شود. اندیس داده ها در لیست از چپ به راست و از صفر به تعداد داده ها در لیست مشخص می شود:

```
[ 'mahdy' , 20 , 1.2 ]
```

در مثال بالا رشته ی 'mahdy' با اندیس ۰ و عدد ۲۰ با اندیس ۱ و عدد اعشاری ۱.۲ با اندیس ۲ مشخص می شوند برای بدست آوردن داده ی یک اندیس کافی است فقط اندیس مورد نظر را درون [] جلوی نام لیست قرار دهیم

```
>>>L=[ 'mahdy' , 20 , 1.2 ]
>>>L[1]
20
```

خوب همیشه لیست ما به این شکل نیست و شاید یک لیست درون لیست قرار داشته باشد و ما برای بدست آوردن داده ی درون لیستی که درون لیست دیگر قرار دارد باید از روش دیگری استفاده کنیم برای روشنتر شدن مطلب لطفا به مثال زیر توجه کنید:

```
>>>L=[ 'mahdy' , 20 , [ 'ali' , 99 ] , 1.2]
>>>L[2][0]
'ali'
```

خوب در مثال بالا دیدید که اندیس L[2] ما را به داده ای که خودش یک لیست هست رساند و L[2][0] ما را به داده ی درون لیست که خودش در لیست هست رساند. حال اگر چندین لیست تو در تو باشند، این روال ادامه می یابد.

نکته۱: اگر درون [] عبارت محاسبه ای قرار بگیرد ابتدا عبارت محاسبه شده و بعد بقیه کارها طبق روال پیش می روند.

نکته۲: اگر اندیس قرار گرفته در [] در محدوده شماره ی اندیس ها نباشد پیغام خطا ظاهر می شود.

نکته۳: از خواص لیست ها انتساب هر یک از داده های درون لیست به متغیرهایی است که همگی تشکیل یک لیست می دهند.

نکته۴: در شماره گزاری اندیس ها درلیست برنامه نویس دخیل نیست بدین ترتیب که می توان از راست به چپ هم به داده ها درون لیست دسترسی داشت از ۱- شروع شده و به صورت نزولی اندیسها شماره گزاری میشوند.البته انتخاب هر روش به دلخواه برنامه نویس است.

نکته۵: فقط اپراتورهای + و * بر روی لیست ها تعریف شده اند که اپراتور + بین یک لیست و یک لیست دیگر تعریف شده (که لیست دوم را به انتهای لیست اول چسبانده و بر می گرداند) و اپراتور * بین یک لیست و عدد تعریف شده که نیازی به توضیح بیشتر نیست.

دسترسی به مجموعه ای از داده های درون لیست

خوب شاید ما بخواهیم به چندتا از داده های درون لیست دسترسی داشته باشیم برای این کار به جای [] از [:] استفاده می کنیم به مثال زیر توجه کنید.

```
>>>L=[ 'mahdy' , 'ali' , 100 , 99 , 'reza' , 20 ]
>>>L[1:5]
['ali', 100, 99, 'reza']
```

می بینیم که L[1:5] از داده ای که اندیسش ۱ است تا ماقبل ۵ را درون یک لیست قرار می دهد.

نکته۱: اگر در سمت راست علامت : عددی بیشتر از محدوده قرار بگیرد یا عددی نوشته نشود پیغام خطایی ظاهر نمی شود بلکه تا آخرین خانه ی لیست در نظر گرفته می شود و برعکس.

نکته۲: اگر شماره طوری باشند که از چپ به راست آخرین اندیس برابر یا قبل از اندیس اولی باشد حاصل یک لیست تهی است:

```
>>>L=[ 'mahdy' , 'ali' , 100 , 99 , 'reza' , 20 ]
>>>L[1:]
[ 'ali' , 100 , 99 , 'reza' , 20 ]
>>>L[1:1]
[]
```

تغییر مقادیر درون لیست

خوب لیست ها از داده های تغییر پذیرند و می توان با استفاده از عمل انتساب (=) مقادیر یک لیست را تغییر داد:

```
>>>L=[ 'mahdy' , 'ali' , 100 , 99 , 'reza' , 20 ]
>>>L[1] = 'boys'
>>>L[1]
```

```
'boys'
>>>L
['mahdy' , 'boys' , 100 , 99 , 'reza' , 20 ]
```

اضافه کردن داده در لیست

بدون استفاده از عملیات محاسباتی می توان داده ای را به لیست اضافه کرد. برای این کار L[n:n] را به مقدار جدید(داخل علامت []) انتساب می دهیم.(n اندیسی است که می خواهیم داده جدید در آن جا اضافه شود)

```
>>>L=[ 1 , 2 , 3 ]
>>>L[3:3] = [4]
>>>L
[ 1 , 2 , 3 , 4 ]
```

حذف داده از لیست

برای حذف داده می توانیم از عمل انتساب استفاده کنیم بدین ترتیب که داده را به [] انتساب بدهیم

```
>>>L=[ 1 , 2 , 3 , 4 ]
>>>L[1:2] = []
>>>L
[ 1 , 3 , 4 ]
```

نکته مهم در این قسمت این است که L[1:2] تنها بخشی از لیست را حذف می کند که اندیس آن ۱ است. حال اگر بنویسم L[0:2] بخشهایی که اندیس آن ۰ و ۱ است حذف می شود. یعنی پارامترهای ما از ابتدا تا قبل از انتها حذف می شوند.

با استفاده از دستور del می توان هر داده یا کل لیست را حذف کرد.

متدهای مهم لیست (با فرض اینکه L لیست باشد)

۱	L.append(x)	داده ی x را به انتهای لیست L اضافه می کند
۲	L.remove(x)	داده ی x را از لیست L حذف می کند
۳	L.index(x)	اندیس x در لیست L را برمی گرداند
۴	L.count(x)	تعداد دفعات تکرار x را در لیست L را برمی گرداند
۵	L.extend(x)	لیست x را با لیست L الحاق می کند
۶	L.insert(I,x)	داده ی x را درشماره اندیس I به لیست اضافه می کند

تاپل (Tuple)

تاپل ها نیز همانند لیست ها جز داده های چند قسمتی هستند. با این تفاوت که داده های درون یک تاپل به طور مستقیم قابل تغییر نیستند. به عنوان مثال شما نمی توانید عمل انتساب را برای داده های درون لیست انجام دهید.

بعد از لیست ها، تاپل ها بیشتر در میان داده های چند قسمتی مورد استفاده قرار می گیرند و این شاید دلیلش نزدیک بودن خصوصیات تاپل ها به لیست ها و راحتی کار با تاپل ها باشد.

تاپل ها با کاما(,) از هم جدا می شوند.شکل ظاهری تاپل ها همانند این است که چند داده را بـه یـک متغیـر انتساب بدهید اما باید بدانید که نوع داده ی نگهداری شده از نوع تاپل است. شما می توانیـد بـرای تـشخیص راحت و زیبایی کدتان از پرانتز( ) استفاده کنید و داده ها را در داخل پرانتز قرار دهید اما بـه ایـن نکتـه توجـه داشته باشید که علامت ( ) نشان دهنده تاپل بودن داده نیست. مثالی برای درک بهتر :
در این مثال دو متغیر از نظر نوع داده ها و مقدار داده ها با هم تفاوتی ندارند. (اما از نظر نـوع ذخیـره سـازی داده ها با هم متفاوت هستند)

```
a = 1, 2, 3, 4
b = (1, 2, 3, 4)
```

- نکته۱: به این نکته توجه داشته باشید که داده های چند قسمتی می توانند داده های چند قسمتی دیگـر را در خود نگه دارند.
- نکته۲: برای تاپل های تک عضوی باید به صورت زیر عمل کنید.

```
a = 33  # این یک متغیر ساده است
b = 33, # این یک داده از نوع تاپل است
```

شاید فکر کنید که پایتون زبان خیلی سختی است اما فقط کافی است کمی دقت داشته باشید تا متوجه آسانی و انعطاف پذیری باشید.

اپراتورها در تاپل

اپراتورهایی که برای تاپل ها تعریف شده اند + (بین دو تاپل) و * (بین یک عدد و تاپل)هستند.
سعی کنید برای این کار تاپل ها را درمیان پرانتز قرار دهید یا به یک متغیر انتساب دهید به مثال توجه کنید:

```
>>>1, 2, 3 + 4, 5, 6
(1, 2, 7, 5, 6)
>>>(1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)
```

دسترسی به داده های درون تاپل

این قسمت همانند لیست هاست.
فقط یک مثال ساده...

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

نکته: تاپل ها از نوع داده های تغییرناپذیرند یعنی نمی توان آنها را به طور مستقیم تغییر داد. ولـی بطـور غیـر مستقیم امکان دارد. به مثال زیر توجه کنید.

```
>>> T = (11, 22, 33, 44)
>>> T [0] = 55

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in -toplevel-
    t[0] = 11
TypeError: object does not support item assignment

>>> T = (55,) + T[0:3]
```

```
>>> T
(55, 22, 33, 44)
```

نکته ی بعدی اینکه بعضی از اعمال برروی لیست ها روی تاپل نیز جواب می دهند.
نکات زیادی در تاپل ها (و همه قسمت های پایتون) وجود دارند که در این مقوله نمی گنجند.

دیکشنری (Dictionary)

Dictionary ها نیز از جمله داده های چند قسمتی هستند آغاز Dictionary ها با { و پایان آنها با } مشخص میشود و داده های آن با علامت کاما(,) از هم جدا می شوند. در هر خانه و هر قسمت Dictionary علاوه بر داده، اندیس مربوط به آن نیز آمده است. بر خلاف رشته ها، لیست ها و Tuple ها اندیس داده ها در Dictionary به دلخواه برنامه نویس مشخص می شود و اندیس داده ها علاوه بر عدد، رشته، tuple و هر داده غیر قابل تغییر نیز می تواند باشد. به کمک علامت : اندیس هر داده را مشخص می کنیم. بدین ترتیب که داده در سمت راست علامت : و اندیس آن در سمت چپ علامت : نوشته می شود. قابل ذکر است که اندیس ها در Dictionary نمی تواند یک داده ی تغییر پذیر، لیست و یا Dictionary باشد. داده های درون Dictionary از هر نوع می تواند باشد:

```
>>> D = { 'ali':20 , 'mahdy':'m' , 22:11 }
```

دسترسی به داده های درون Dictionary

جهت دسترسی به داده های درون Dictionary با قرار دادن اندیس داده ی مورد نظر درون [ ] در مقابل Dictionary این امر امکان پذیر است:

```
>>>D={ (1,2):11 , 'mahdy':'m' , 78:88 }
>>>d['mahdy']
'm'
>>>d[(1,2)]
11
```

توجه داشته باشید که ممکن است در اندیس داده های Dictionary اندیس های مثل هم وجود داشته باشد که در این صورت تنها مقدار اولین داده از سمت راست با همان اندیس نشان داده می شود و در حقیقت دیگر داده های موجود با آن اندیس از Dictionary حذف می شوند.

```
>>>d={ 'mahdy':'m' , 'python':'c' , 'mahdy':55 }
>>>d['mahdy']
55
>>>d
{'python': 'c', 'mahdy': 55}
```

اگر در مقابل Dictionary اندیسی قرار دهیم که در Dictionary وجود نداشته باشد پیغام خطا ظاهر می شود البته در این مورد استثنا نیز وجود دارد که به آن اشاره خواهیم کرد.

تغییر و افزودن داده به Dictionary

Dictionary ها از داده های تغییرپذیرند و می توان با علامت انتساب داده های هر اندیس را تغییر داد همچنین می توان داده ی جدیدی به همراه اندیس آن به Dictionary اضافه نمود:

```
>>>d={'mahdy':'m' , 2:'ali' }
>>>d['reza']= 66
>>>d
{ 'mahdy':'m' , 2:'ali' , 'reza':66 }
```

دستور del

با استفاده از این دستور می توان داده های درون Dictionary و یا حتی کل متغیر را حذف کرد:

```
>>>d={'mahdy':'m' , 2:'ali' }
>>>del d[2]
>>>d
{ 'mahdy':'m' }
```

تابع len()

این تابع یک داده ی چند قسمتی به عنوان پارامتر گرفته و تعداد خانه های این داده ی چند قسمتی را بر می گرداند. یعنی حاصل تابع عدد صحیح است.

دستور for برای Dictionary

اگر در مقابل in در دستور for داده ی چند قسمتی Dictionary قرار گیرد، for در هر بار چرخش اندیس های تعریف شده در Dictionary را در متغیر قبل از in قرار داده و وارد حلقه می شود. توجه داشته باشید که مسأله ترتیب داده ها در Dictionary مطرح نیست چرا که اندیس هر داده از سوی ما تعیین می شود. نکته مهم اینجاست که اگر داده ی چند قسمتی قرار گرفته در دستور for , Dictionary باشد نباید تعداد داده های درون آن در حلقه for تغییر کند در غیر این صورت پیغام خطا ظاهر می شود.

به عنوان مثال:

```
>>>a={1:1,2:'c',3:'b'}
>>> for a in d:
      print a
1
c
b
>>>
```

متدهای مربوط به Dictionary

اگر d را یک Dictionary در نظر بگیریم داریم:

d.clear()	تمام داده های درون d حذف می شود
d.copy()	یک کپی از d برمی گرداند
d.has_key(m)	اگر اندیس m در d باشد ۱ در غیر این صورت صفر برمی گرداند
d.keys()	لیستی از تمام اندیسها برگردانده می شود
d.values()	لیستی از تمام داده ها برمی گرداند

مجموعه ها Sets

ماژول جدید sets شامل یک پیاده ساز از طرف یک مجموعه Datatype است. مجموعه کلاسی است برای مجموعه های تغییر پذیر؛ مجموعه هایی که می تواند عضو اضافه شده یا کم شده داشته باشد.
به مثال زیر توجه کنید:

```
>>> S = set([1,2,3])
>>> S
set([1, 2, 3])
>>> 1 in S
True
>>> 0 in S
False
>>> S.add(5)
>>> S.remove(3)
>>> S
set([1, 2, 5])
>>>
```

اجتماع و اشتراک را در این ماژول می توان با استفاده از متد union() و intersection() بدست آورد. و راه ساده تر استفاده از نمادهای & و | است.

```
>>> S1 = set([1,2,3])
>>> S2 = set([4,5,6])
>>> S1.union(S2)
set([1, 2, 3, 4, 5, 6])
>>> S1 | S2                    # نماد دیگر
set([1, 2, 3, 4, 5, 6])
>>> S1.intersection(S2)
set([])
>>> S1 & S2                    # نماد دیگر
set([])
>>> S1. update(S2)    #  S1=S1|S2
>>> S1
set([1, 2, 3, 4, 5, 6])
```

همچنین ممکن است تفاوت و شباهت میان دو مجموعه را نشان داد. این مجموعه همه ی عناصر در اجتماع و اشتراکها نیستند و ممکن است با هم تفاوت داشته باشند.برای این کار ما می توانیم از متد symmetric_difference() و یا نماد ^ استفاده کنیم.

```
>>> S1 = set([1,2,3,4])
>>> S2 = set([3,4,5,6])
>>> S1.symmetric_difference(S2)
set([1, 2, 5, 6])
>>> S1 ^ S2
set([1, 2, 5, 6])
>>>
```

همچنین متدهای issubset() و issuperset() برای چک کردن اینکه آیا یک مجموعه زیر مجموعه یا مجموعه مرجع دیگری است یا نه.

```
>>> S1 = set([1,2,3])
>>> S2 = set([2,3])
>>> S2.issubset(S1)
True
>>> S1.issubset(S2)
False
>>> S1.issuperset(S2)
True
```

عبارات و کنترل ترتیب اجرا

یک عبارت در حقیقت یک دستور برنامه نویسی می باشد که مفسر پایتون قدرت اجرای آن را دارد. تا کنون و در این درس شما با دو نوع عبارت آشنا شدید. یکی دستور چاپ و دیگری مقدار دهی. معمولا در زبان پایتون هر عبارت در یک سطر مجزا نوشته می شود و مفسر پس از اجرای هر سطر یا عبارت سراغ سطر بعدی می رود.

برنامه یا یک اسکریپت شامل تعدادی از عبارت ها یا دستورات اجرایی پشت سر هم می باشند.

عبارات محاسباتی

هر عبارت ترکیبی از مقادیر ، متغیر ها و عملگر ها می باشد. اگر شما یک عبارت محاسباتی را به عنوان یک دستور وارد کنید، مفسر مقدار آن را محاسبه و چاپ خواهد کرد.

```
>>> 1 + 1
2
>>> 2*2
4
```

عملگر در حقیقت یک سمبل یا نماد قراردادی برای نمایش اتفاق افتادن عملی خاص چون جمع یا تفریق می باشد. بدین ترتیب هر عملگر بر روی تعدادی عملوند عمل کرده و نتیجه ای را حاصل می دهد.

اولویت عملگر

زمانی که یک عبارت محاسباتی شامل چندین عملگر باشد، تقدم و ترتیب اجرا عملگر ها مهم بوده و در نتیجه کلی عبارت تاثیر گذار می باشد. بنابراین برای بدست آوردن نتیجه ای یکسان و استاندارد از قوانینی برای ترتیب اجرای عملگر ها استفاده می شود. قوانین اولویت عملگر در پایتون دقیقا با قوانین علم ریاضیات یکسان می باشد.

- پرانتز دارای بیشترین اولویت می باشد. و می توان از آن برای تغییر اجرای عملگر ها استفاده کرد. مفسر پایتون ابتدا مقدایر داخل پرانتز را با شروع از درونی ترین پرانتز اجرا کرده و نتیجه محاسبه شده را در محاسبات بعدی بکار می گیرد.
- به جز توان، عملگرهایی که اولویت یکسانی دارند به ترتیب و از سمت چپ به راست اجرا می‌گردند.

```
>>> 3 ** 2 ** 2
81
>>> 5*6+7-1/2
37
```

در قسمت اول مثال فوق ابتدا عدد ۲ به توان ۲ رسیده و نتیجه یعنی ۴ محاسبه می شود. سپس عدد پایه و اصلی ۳ به توان ۴ می رسد. دقت کنید که این قانون نیز برگرفته از ریاضیات می باشد.

درقسمت دوم ابتدا باید جمع انجام شود. برای این کار دوطرف عملگر باید آماده شود(یعنی 5*6 و 1/2-7) پس از محاسبه دوطرف عمل جمع بین 30 و 7 انجام شده و حاصل برابر 37 می گردد.

تقدم عملگر در زبان پایتون

تقدم عملگر ها در زبان پایتون به صورت جدول صفحه بعد می باشد.

۱	() [] .	گروهبندی و دسترسی
۲	+ - !	عملگر های یکتایی
۳	**	توان
۴	* / %	ضرب ، تقسیم ، باقیمانده
۵	+ -	جمع و تفریق
۶	<< >>	عملگر های شیفت به چپ و راست باینری
۷	< <= > >=	عملگرهای مقایسه ای : بزرگتر ، کوچکتر و...
۸	== !=	عملگرهای مقایسه : مساوی ، نا برابر
۹	&	و منطقی باینری
۱۰	^	یا انحصاری (XOR) باینری
۱۱	|	یا منطقی باینری
۱۲	and	و منطقی
۱۳	or	یا منطقی
۱۴	= ، += ، -= ، *= ، /= ، %=	عملگر های مقدار دهی

عملگر های رشته

اتصال دو رشته با استفاده از عملگر جمع

عملگر ترکیب دو رشته که با سمبل مربوط به عمل جمع ریاضی یا "+" نشان داده می شود دو عملوند از نوع رشته را دریافت و رشته سمت راست خود را به انتهای رشته سمت چپی خود می چسباند.

```
>>> fname = "Alex"
>>> lname = "Del Piero"
>>> fname + lname
'Alex Del Piero'
```

تکرار یک رشته با استفاده از عملگر ضرب

عملگر ضرب با سنبل * نیز معنی خاصی برای رشته دارد. این عملگر دو عملوند یکی از نوع رشته و دیگری از نوع عدد صحیح را دریافت و رشته را به تعداد عدد دریافت شده تکرار کرده و در قالب یک رشته واحد بر می گرداند.

```
>>> "Python ! " * 3
'Python ! Python ! Python ! '
```

عملگر ضرب در مورد رشته ها منطقی شبیه به نوع ریاضی آن دارد. ضرب دو عدد در ریاضیات را می توان بصورت جمع نشان داد. برای مثال ۳×۴ را می توان بصورت ۴+۴+۴ نمایش داد. در مورد رشته ها نیز در حقیقت برای محاسبه ضرب یک عدد در رشته ، همان رشته با تعداد عدد وارد شده باهم جمع می شود.

کار با بیت ها

شاید وجود این مبحث در پایتون (پیتونی که یک از آسانترین زبانهای برنامه نویسی است) کمی به نظرات عجیب باشد، ولی گاهی لازم می شود که با بیت ها سروکار داشته باشیم. از این رو در اکثر زبانهای برنامه نویسی عملگرهایی برای کار با بیت تعبیه شده اند.

همان طور که می دانید بیت (Bit) کوچکترین واحد برای ذخیره سازی داده هاست. کار با بیت ها می تواند دلایل مختلفی ازجمله انجام عملیات محاسباتی ، کار کردن و ارتباط داشتن با سخت افزار ها به صورت مستقیم ، رمزنگاری و... داشته باشد. یکی دیگر از موارد مهم استفاده, صرفه جویی در حافظه می باشد (در برنامه های بزرگ). البته برای کار با بیت ها باید با محاسبات باینری(جبر بول) آشنا باشید.

عملگرهای بیتی (bit operator):

عملگرهای بیتی را می توان به چند دسته تقسیم کرد:

۱. عملگر انتقال(shift)
۲. عملگر منطقی (همانند درست و غلط)
۳. عملگر متمم (یا معکوس)

۱. عملگرهای انتقال:

این نوع عملگر برای انتقال بیت ها به سمت راست یا چپ استفاده می شود. به عنوان مثال عدد ۵۰ به صورت باینری به این صورت نمایش داده می شود(۰۰۱۱۰۰۱۰) اگر بخواهیم این عدد را دو بیت به سمت راست منتقل کنیم(تقسیم بر ۴) به عدد ۱۲ تبدیل می شود(۰۰۰۰۱۱۰۰).

```
>>> 50 >> 2
12
>>>
```

همان طور که مشاهده می کنید چون قرار است دو بیت به سمت راست انتقال دهیم دو بیت آخر حذف می شود و طبق قاعده باید دو بیت (با ارزش صفر) به سمت راست اضافه کنیم تا ۸ بیت کامل باشد.

۲. عملگرهای منطقی:

عملگرهای منطقی برای مقایسه دو بیت از یک عدد صحیح مورد استفاده قرار می گیرد:

۱. & (and): بیت نتیجه ۱ است اگر و فقط اگر دو بیت ۱ باشند در غیر این صورت بیت نتیجه ۰ است.
۲. | (or): اگر حداقل یکی از بیت ها ۱ باشد بیت نتیجه ۱ است در غیر این صورت بیت نتیجه ۰ است.
۳. ^ (xor): اگر دو بیت ارزش مخالف داشته باشند (مثلا یکی ۰ و دیگری ۱ باشد) بیت نتیجه ۱ خواهد بود در غیر این صورت بیت نتیجه ۰ است.

این عملگرها برای دو عدد صحیح بکار می روند و دو بیت متناظر هر یک را با هم مقایسه می کنند.

به مثال زیر توجه کنید:

```
>>> 50 & 20
16
```

```
>>> 50 | 20
54
>>> 50 ^ 20
38
>>>
```

۳. عملگر متمم (معکوس):

این عملگر (~) روی ارزش تمام بیت ها تاثیر می گذارد و تمام آنها را معکوس می کند یعنی هر ۰ به ۱ و هر ۱ به ۰ تبدیل می شود مثلا عدد ۱۵ (۰۰۰۰۱۱۱۱) را معکوس کنیم بـه عـدد ۲۴۰ (۱۱۱۱۰۰۰۰) تبـدیل مـی شود.

در پایان باز هم به این نکته اشاره می کنم که عملگرهـای بیتـی بـر روی داده هـایی از نـوع (long , int) کار می کنند.

ساختارهای کنترلی

همان طور که دانستیم برنامه ی کامپیوتری متشکل از دنباله ای از دستورالعمل ها است. که توسط مـا بـرای انجام عملیات خاصی نوشته می شود. در هر برنامه با توجه به هدف یـا سـاختمان داده ای کـه مـی خـواهیم بدست آوریم و یا اهدافی که باید به آنها دست یابیم، باید روند اجرای دستورات را کنترل کنیم. یعنی برخـی از آنها را در شرایطی و مواقعی خاص اجرا کرده یا نکنیم و یا حتی برخی از آنها را تکرار نماییم.

به همین دلیل برنامه ی ما به قطعه کد هایی تقسیم می شود، که اجرای آنها را توسط **ساختارهای کنترلی** تنظیم می کنیم.

دو ساختار کنترلی عمده در برنامه نویسی موجود می باشد :

۱. شرط یا تصمیم گیری
۲. حلقه یا اصطلاحاً تکرار

در این قسمت ابتدا به بررسی شروط و منطق حاکم بر تصمیم گیری صحبت کرده و سپس به حلقـه هـا مـی پردازیم.

قطعه کد چیست؟

قطعه کد قسمتی از برنامه می باشد که روند اجرای آن بصورت طبیعی و پشت سر هم نباشد. استفاده از حلقـه و تابع در زبان سطح پایین موجب به وجود آمدن قطعه کد گردید کـه تقریبـا هـم قـدمت بـا برنامـه نویـسی کامپیوتر می باشد. برای مثال یک قطعه کد فقط در زمانی که شرط خاصی برقرار باشد اجرا می گردد یا قطعه ای از کد به دفعات زیاد و پشت سر هم و به تعداد از پیش تعیین شده ای تکرار می گـردد. بـرای نـشان دادن قطعه کد از نشانه هایی استفاده می گردد که محل شروع و اتمام کد را نشان می دهند. برای مثال اسـتفاده از عباراتی چون begin و end در زبانهایی چون پاسکال یا دلفی نشان دهنده محل شروع و خاتمه قطعه کد می باشد. یا در زبان قوی و محبوب C از اکولاد باز ({) به عنوان شروع کننده قطعه کد و از اکولاد بـسته (}) به عنوان خاتمه دهنده یک قطعه کد استفاده می گـردد و کـدی کـه مـابین ایـن دو باشـد درون قطعـه کـد محسوب می گردد و جدا از روند اصلی برنامه.

قطعه کد در پایتون

برخلاف زبانهای دیگری چون C و Pascal که از علایم و عباراتی چون {} و begin,end و... برای نمایش شروع و پایان یک قطعه کد استفاده می کنند پایتون دارای علامت یا عبارت خاصی برای این منظور نمی باشد! و برای مشخص کردن محدوده یک قطعه کد (مثلا تابع ، دستورات شرطی ، حلقه ، کلاس و...) از فاصله گزاری اول دستورات استفاده می گردد. به این صورت که پس از خطی که بقیه دستورات زیر مجموعه آن می باشند به اندازه دلخواه فاصله یا تورفتگی داده می شود این فاصله های یکسان تا زمانی ادامه می یابد که محدوده دستور پایان یابد. که این روش باعث کاهش مقدار برنامه و خوانایی برنامه می شود.
از این روش در حالت معمول و برای خوانایی در زبانهای دیگر نیز استفاده می گردد و اکثر برنامه نویسان از تورفتگی کد برای افزایش خوانایی برنامه استفاده می کنند. پایتون با استفاده از این قابلیت هم مشکل خود را رفع کرده و هم الزامی برای رعایت این نظم و روند خاص برنامه نویسی بهره برده است. به این ترتیب کد ها و برنامه های پایتون تقریبا استاندارد و ظاهر یکسانی را دارا می باشند. برای مثال به کد زیر دقت کنید. این کد شکل و منطق کلی دستور شرطی If در پایتون می باشد.

```
>>> name = 'python'
>>> lang = 'python'
>>> if lang == 'python':
        print 'You Are in right way !'

You Are in right way !
```

این یک مثال درست از قطعه کد می باشد. دستور چاپ دارای تورفتگی نسبت به دستور شرط می باشد. پس این دستور زیر مجموعه ای از شرط می باشد و در صورت درست بودن آن اجرا می گردد. اگر عبارت شرط حاوی دستورات بیشتری بود همه آن درست زیر دستور چاپ نوشته می شدند. در غیر این صورت برنامه اجرا نشده و پیغام خطایی ظاهر می گردد. پس از تمام شدن قطعه کد باید دقیقا به زیر محل شروع شدن قطعه کد (در اینجا if) برگردید و ادامه برنامه را دقیقا از زیر دستور اصلی قطعه کد ادامه بدهید.
در مثال بعدی یکی از دستورات داخل قطعه شرط (خط سوم) با دستور قبلی خود به اندازه یک فضای خالی فاصله دارد. برنامه اجرا نشده و با نمایش پیغامی محل وقوع خطا و نوع آن نمایش داده می شود.

```
>>> if lang == 'python':
        print 'You Are in right way !'
          print 'go on'
  File "<stdin>", line 3
    print 'go on'
            ^
IndentationError: unindent does not match any outer indentation
level
```

پس اگر حتی دستور ساده ای چون دستور چاپ (print) را با فاصله (حتی یک فضای خالی) از سمت چپ بنویسید برنامه اجرا نخواهد شد. شاید این نوع خاص و منحصر به فرد در دفعات اول کمی سخت و تا حدودی مضحک بنظر آید اما یکی از مهمترین جنبه های پایتون می باشد که زیبایی خاصی به کد های نوشته شده می دهد و برنامه نویسی را لذت بخش و راحت می کند.

شرط

شرط در واقع ساختاری است که اجازه ی اجرای قطعه کدی را به ازای کنترل یک عبارت منطقی مـی دهـد. در پایتون شرط ساده با عبارت if مشخص شده و بصورت زیر اعمال می شود:

```
if logicalVariable:
        Code Block
```

متغیر منطقی چیست؟

متغیرهای منطقی (logical Variables) می توانند حامل یکی از دو مقدار درست یا غلط باشند. تصمیم گیری نیز بر همین اساس صورت می گیرد. یعنی اگر این متغیر حامل مقدار درست باشد، شرط قطعـه کد مربوطه را به اجرا در می آورد. از دید دیگر این متغیر ساده ترین نوع یک عبـارت منطقـی اسـت. عبـارت های منطقی را بطور مختصر می توان به دو دسته ساده و پیچیده تقسیم کرد. همه عبارت هـای منطقـی بـه یکی از مقادیر درست یا غلط ختم می شوند. این عبارات با مقادیر و عملگر های منطقی ساخته می شوند کـه همگی دودویی هستند یعنی برای انجام عمل به دو عملوند احتیاج دارند. مقدماتی ترین این عملگرهـا از ایـن قرارند:

عملگر های منطقی

عملگر	توضیح
==	برابری دو عملوند
!=	عدم برابر دو عملوند
>	بزرگتر
<	کوچکتر
>=	بزرگتر یا مساوی
<=	کوچکتر یا مساوی
and	و منطقی
or	یا منطقی
not	متمم (not)

ساخت عبارات منطقی

در بالا در مورد چگونگی شکل گیری عبارات مختصر گفته شد. در اصل مقادیر منطقی اولیـه بـا عملگرهـای منطقی که در متن فوق بصورت کم رنگ نوشته شده اند بوجود آمده و با سه عملگر پررنگ تر یعنـی and و or و not ترکیب خواهند شد.

عبارات شرطی (تصمیم گیری)

if ساده

پایتون با استفاده از روش بسیار ساده ای امکان تصمیم گیری در برنامه را فراهم می کند. برای این کار ابتدا از یک کلمه رزرو شده بنام if استفاده می گردد. پس از این عبارت شرطی ظاهر می گردد که این عبارت هنگام اجرا تست شده و در صورت صحت نتیجه کلی آن قطعه کد مربوط به شرط به اجرا در می آید. این خط شروع کننده قطعه شرطی با کاراکتر دو نقطه (:) به اتمام می رسد.

برای شروع برنامه ای می نویسیم که مقداری را از ورودی گرفته سپس علامت آنرا اعلام می کند؛

```
Val = raw_input("enter a number: ")
Val = int(Val)

if Val > 0 :
    print "Entered value is positive!"
if Val < 0 :
    print "Entered value is negative!"
```

خروجی:

```
enter a number: -32
Entered value is negative!
enter a number: 24
Entered value is positive!
>>>
```

این برنامه چگونه عمل می کند؟ در این برنامه مفسر هر شرط را چک می کند. و در صورت درست بودن هر یک قطعه کد مربوطه را اجرا می کند. در صورت درست بودن شرط اول دیگر نیاز به تست کردن شروط دیگر نداریم و سایر تست ها کاری بیهوده به حساب می آیند. برای جلوگیری از این مشکل در ادامه با ساختار جدیدی آشنا خواهیم شد.

ترکیب if و elif

اگر ما if را معادل **اگر** بگیریم elif هم معنی واژه ای مثل **و اگر نه** خواهد بود. این ترکیب با if شروع شده و با elif ها ادامه می یابد. در این حالت ابتدا شرط if اصلی تست می گردد. در صورت درست بودن شرط قطعه کد مربوطه اجرا شده و برنامه بدون تست شرط سایر elif ها به روند عادی اجرا ادامه می دهد و در صورتی که شرط نادرست باشد elif بعدی مورد پردازش قرار خواهد گرفت. تا زمانی که یکی از مقادیر شرطی elif معادل True باشد. ناگفته نماند که این دستور (elif) یک عضو از دستور if می باشد و خود بطور جداگانه کاربردی ندارد.

```
if <شرط>:
        Code Block 1
elif <شرط دوم>:
        Code Block 2
elif <شرط سوم>:
        Code Block 3
```

....
تفاوت اینگونه شرط ها این است که مفسر به محض درست شدن یکی از سلسله شرط هـا از مـابقی سلـسله صرف نظر می کند. این خود یک عامل سرعت دهنده می باشد. البته این روش تنها در مواردی خاص کـاربرد دارد که باید توسط نویسنده برنامه تشخیص داده شود و آن حالتی است که حد اکثر یکی از این شـرط هـا در یک زمان برقرار باشد.
اکنون مثال قبلی را که مثالی ساده بود توسط این روش باز نویسی می کنیم:

```
Val = raw_input("enter a number: ")
Val = int(Val)

if Val > 0 :
    print "Entered value is positive!"
elif Val < 0 :
    print "Entered value is negative!"
```

البته این امر در این برنامه تقریباً تاثیری ندارد اما در مثال های بزرگتر و پیچیده تر آخر فصل با این امر بهتـر مواجه خواهیم شد.

عبارت else

این عبارت نیز نوعی شرط است اما به تنهایی کاربرد ندارد و باید بعد از if یـا elif آورده شـود. در واقـع کاربرد تنهای آن یک خطا محسوب می شود. else ایـن امکـان را بـه مـا مـی دهـد کـه اگـر در زنجیـره شرطهای ما هیچ شرطی درست نبود قطعه کدی را که در خود دارد اجرا کنـد. همـانطور در مثالهـا و مطالـب قبلی مشخص می باشد ، وجود این عبارت در شرطها الزامی نیست و وجود آن باز هم به برنامـه و کـاری کـه برنامه نویس قصد انجام آنرا دارد وابسته می باشد. حال با این عبارت برنامه خـود را گـسترش داده و قابلیـت تشخیص صفر را به آن می دهیم:

```
Val = raw_input("enter a number: ")
Val = int(Val)

if Val > 0 :
    print "Entered value is positive!"
elif Val < 0 :
    print "Entered value is negative!"
else :
    print "Entered value is zero"
```

خروجی:

```
enter a number: 0
Entered value is zero
```

مثال – تبدیل شماره روز به تاریخ

برنامه ای که شماره یک روز از سال را بگیرد و اگر آن روز در سال گنجید. تاریخ معـادل شمـسی آن را چـاپ کند؟

```
1   # The program taking day number and returns which day it is in
jalali calendar!
2
3  day = input("Enter day number: ")
4
5  yLen = 365
6
```

```
7  mDay = 0
8  month = 0
9  #if the day is in first half of the year
10 if 0< day <= 186:
11    month = day / 31 + (day % 31 > 0)
12    mDay = day % 31
13
14 #if the day in the second half of the year
15 elif 186 < day <=365:
16    day -= 186
17    month = day / 30 + (day % 30 >0) + 6
18    mDay = day % 30
19  #day out of year
20 else:
21    print "Bad day number entered"
22
23 print "in jalali calendar ---->" ,month, ".", mDay
```

در خطوط ۱۱ و ۱۷ چون از تقسیم صحیح استفاده شده است باقیمانده و اعشار در نظر گرفته نمی شود؛ پایتون خودبخود برای این عمل مثل جزء صحیح عمل کرده و حاصل را به طرف پایین گرد می کند. با جمع بستن عبارت منطقی کاری شبیه به تابع Ceiling ریاضی انجام و تقسیم را به طرف بالا گرد شد. عبارات داخل پرانتز مقدار منطقی است که در صورت وجود باقیمانده در تقسیم مقدار ۱ و در غیر این صورت صفر خواهد بود.

خروجی برنامه:

```
>>>
Enter day number: 366
Bad day number entered
in jalali calendar ----> 0. 0

>>>
Enter day number: 56
in jalali calendar ----> 2. 25

>>>
Enter day number: 323
in jalali calendar ----> 11. 17
>>>
```

حلقه و تکرار

برای ایجاد حلقه و تکرار در پایتون از دو ساختار : ... while و ... in ... for استفاده می شود. در ادامه چگونگی استفاده از این دو دستور را شرح خواهیم داد.

ساختار : ... while

ساختار کلی این while به صورت زیر می باشد:

```
>>> while <condition>:
        <statement1>
        <statement2>
        …

>>>
```

در استفاده از while هم باید به رعایت فاصله بندی ها توجه نمود.

مثال

```
>>> StrList = [ 'str1', 'str2', 'str3' ]
>>> count = 0
>>> while count < len(StrList):
      print StrList[count]
      count = count+1

str1
str2
str3
>>>
```

در این مثال StrList یک لیست سه تایی از رشته هاست که می خواهیم توسط while محتویات آن را چاپ کنیم. برای این کار یک متغیر count تعریف می کنیم که بیانگر اندیسی است که محتوای آن چاپ می شود. countدر هر مرحله یک واحد اضافه می شود. این حلقه تا زمانی ادامه می یابد که شرط حلقه (یعنی کوچکتر بودن مقدار count از طول لیست) برقرار باشد.

ساختار : for ... in ...

ساختار کلی :for ... in ... به صورت زیر می باشد:

```
>>> for <variable> in <iterator>:
          <statement1>
          <statement2>
          ...
```

در این ساختار <variable> در هر مرحله به یکی از محتویات چندتایی <iterator> اشاره می کند و می توان در دستورات داخل حلقه از آن استفاده نمود.

لازم به ذکر است که<iterator>حتما باید یک ساختار چندتایی مانند لیست ها، تاپل ها و سایر ساختمان های داده چندتایی باشد.

استفاده از for برای نوع داده دیکشنری کمی متفاوت است که در توضیحات این نوع داده به آن اشاره شد.

به عنوان مثال:

```
>>> StrList = ['str1', 'str2', 'str3']
>>> for s in StrList:
          print s

str1
str2
str3
>>>
```

در این مثال با استفاده از ساختارfor محتویات لیست StrList را چاپ نمودیم.

توابع و کنترل زیربرنامه

برای تعریف و استفاده از توابع در پایتون از عبارت های ویـژه و از پـیش تعیـین شـده def، return و global استفاده می گردد که در ادامه به توضیح هر یک پرداخته می شود.

اصول توابع در پایتون

در قسمت های قبل شما با توابع کار کردید! برای مثال تابع len که طول یک متغیر را بـر مـی گردانـد. در این فصل شما یاد خواهید گرفت که توابع جدید ایجاد کنید.

تعریف تابع

با استفاده از عبارت def یک تابع ایجاد شده و نامی به آن تعلق می گیرد. در زبان پایتون برای تعریف یـک تابع جدید از دستور def استفاده می گردد. با استفاده از def یک تابع جدید ایجاد شده و اسمی به آن تعلق می گیرد. نام تابع درست بعد از عبارت def ظاهر می گردد در مقابل نام تـابع و در داخـل پرانتـز تنهـا نـام پارامتر های قابل دریافت توسط تابع نوشته می شود و در نهایت همانند سایر قطعه کد های موجود با کـاراکتر دو نقطه (:) ختم می گردد. در ادامه و در سطر های بعدی و با رعایت کردن تـورفتگی کـد هـای داخـل تـابع نوشته می شود.

بازگشت مقدار از تابع

زمانی که یک تابع فراخوانی می شود ، فراخواننده تابع تا زمان اتمام کار تابع و برگشت نتیجه تابع منتظر مـی ماند. در حقیقت روند اجرا برنامه به از سطر فراخوانده شده به داخل تابع منتقل و پـس از اتمـام کـار از سـطر بعدی ادامه می یابد. در صورتی که تابعی بخواهد مقداری را بـه عنـوان نتیجـه کـار برگـشت دهـد از دسـتور return برای انجام این کار استفاده می کند.

استفاده از متغیر های سراسری

در حالت معمول تمام متغیر های تعریف شده در داخل تابع محلی می باشند. یعنی طول عمر آنها تا زمان اجرا تابع بوده و فقط از داخل تابع امکان دسترسی را دارند. در صورتی که بخواهیم از نام یـا متغیـری در خـارج از تابع نیز استفاده کنیم از دستور global استفاده می کنیم. این دستور متغیر های محلی را بـه متغیـر هـای سراسری تبدیل می نماید.

ارسال پارامتر به تابع

در پایتون ارسال متغیر به توابع از طریق فراخوانی با ارجاع(call by reference) صورت می گیرد. یعنی همان متغیر و نه یک کپی از آن به تابع ارسال می گردد. پس هر گونه دستکاری آن در داخل تابع باعث تغییر متغیر اصلی خواهد شد. حتی با تغییر نام و مقدار دهی دوباره به متغیری جدید باعث ایجاد متغیر جدیدی در حافظه نخواهد شد! و فقط داده موجود یک نام (اشاره گر) جدید خواهد داشت.

مقدار دهی پیش فرض پارامتر ها

در هنگام فراخوانی تابع باید به هر پارامتر آن یک مقدار نسبت داد (فرستاد). و در صورتی که مقداری به یک پارامتر فرستاده نشود باعث بروز خطای برنامه نویسی خواهد شد. برای جلوگیری از این حالت پایتون شما را قادر می سازد تا برای پارامترها تابع خود مقدار پیش فرض نسبت دهید. این کار را می توانید در هنگام تعریف تابع و با نسبت دادن مقدار پیش فرض بوسیله علامت تساوی به نام پارامتر انجام دهید.

```
def make_omelet2(omelet_type = "cheese"):
```

حال شما می توانید این تابع را با ورودی یا بدون ورودی فراخوانی کنید.

تابع در داخل تابع

شما می توانید تابعی را درون تابع تعریف کنید. زمانی که می خواهید تابعی بزگتر و پیچیده را به بخش هایی کوچکتر تقسیم کنید می توانید هر یک از این بخش ها را به عنوان یک تابع درون تابع اصلی تعریف کنید. که در این حالت نحوه تعریف همانند تعریف تابع معمولی می باشد. با این تفاوت که از بیرون تابع اصلی نمی توان به توابع عضو دسترسی داشت.

تست پارامتر ها

چون متغیر ها در پایتون دارای نوع ثابت نیستند و معمولا می توان در بازه های زمانی مختلف انواع متفاوتی از یک متغیر را داشت پس امکان ارسال انواع داده های مختلف به توابع هم هست که ممکن است باعث بروز خطا گردند یا نتایج اشتباهی را تولید کنند. برای جلوگیری از اینگونه موارد باید نوع متغیر خود را قبل از استفاده در داخل تابع تست کنیم.

```
def make_omelet(omelet_type):
    if type(omelet_type) == type({}):
        print "omelet_type is a dictionary with ingredients"
        return make_food(omelet_type, "omelet")
    elif type(omelet_type) == type(""):
        omelet_ingredients = get_omelet_ingredients(omelet_type)
        return make_food(omelet_ingredients, omelet_type)
    else:
        print "I don't think I can make this : %s" % omelet_type
```

در این مثال ابتدا نوع متغیر `omelet_type` بررسی می شود. اگر نوع آن دیکشنری باشد نشان دهنده مواد لازم برای تهیه غذا می باشد پس تابع `make_food` با این دیکشنری و غذایی با نام `omelet` که به همراه آن ارسال شده فراخوانی و سپس نتیجه از طریق دستور `return` برگشت داده می شود. در قسمت بعدی (`elif` معادل `else if` می باشد) اگر متغیر ارسالی از نوع رشته باشد فرض بر این خواهد بود که نام نوع خاصی از املت می باشد. پس ابتدا از طریق تابع `get_omelet_ingredients` مواد لازم برای پختن آن را بدست آورده و این مواد را همراه با نام آن به تابعی که کار درست کردن غذا را انجام می دهد، ارسال می کنیم. در بقیه موارد پیامی را چاپ می کنیم مبنی بر اینکه نمی توانیم غذای خواسته شده را درست کنیم!!

شی گرایی

مقدمه

پایتون یکی از آن زبان هایی است که بر خلاف C ، امکانات شی گرایی را در بطن وجودش پیاده سازی کرده است. برنامه نویسان می توانند با استفاده از پایتون به صورت قدرتمندی اقدام به تولید برنامه های شی گرا نمایند. اما به خاطر داشته باشید وجود تفکر شی گرایی به هنگام برنامه نویسی مهم تر از وجود امکانات شی گرایی در آن زبان خاص است. در این مقاله سعی شده است تا جدید ترین روش های شی گرایی در پایتون نوشته شود. به همین خاطر ما از کلاس های سبک جدید پایتون استفاده کرده ایم که به مراتب قابلیت های بالاتری نسبت به کلاس های کلاسیک پایتون دارند.

مفهوم کلاس

مقدمه

شی گرایی در برنامه نویسی دقیقا همانند کار با اشیا در دنیای واقعی است. در دنیای واقعی اشیا قابل لمس و حقیقی هستند پس می توانیم خیلی راحت با آن ها تعامل برقرار کنیم، اما در دنیای مجازی اوضاع به همین صورت نیست. برای مثال شما نمی توانید یک دوچرخه یا یک ماشین را به صورت فیزیکی به برنامه وارد کنید! اما می توانید توسط ساختارهای خاص برنامه نویسی آن ها را با کدهایتان پیاده سازی کنید. ساختاری که وظیفه ی پیاده سازی اشیا در دنیای برنامه نویسی را به عهده دارد، کلاس نامیده می شود. به زبان راحت تر اشیا در دنیای برنامه نویسی به صورت کلاس تعریف می شوند.

پیاده سازی کلاس ها

در این قسمت قصد داریم ساختار سی دی پلیر را توسط یک کلاس پیاده سازی کنیم. بهتر است ابتدا کدهایمان را بنویسیم و سپس راجع به تک تک خط هایمان توضیح دهیم. این فقط یک مثال ساده است پس ما قرار نیست زیاد به جزییات توجه کنیم. در حال حاضر شما باید سعی کنید بیشتر از طرز کار کدها، به ساختار بندی کدها توجه کنید:

```
class CdPlayer(object):

    def __init__(self):
        self.volume = 20
        self.currentSong = 0

    def setVolume(self, newValue):
        self.volume = newValue

    def play(self, trackNumber):
        #operation for palying songs

    def nextSong(self):
        self.currentSong += 1
        self.play(currentSong)

    def previousSong(self):
        self.currentSong -= 1
        self.play(currentSong)
```

توضیح مثال

در خط اول از مثالمان ما توسط کلید واژه ی class شیءای با نام CdPlayer را تعریف کردیم. و چون این شی از object ارث برده است پس جزو کلاس های جدید محسوب میگردد و امکان استفاده از مزایای آن را خواهد داشت.

در خط بعد شما یک متد مخصوص به نام __init__ را مشاهده میکنید. به حالت عادی شما نباید از خود کلاس اصلی استفاده کنید و برای استفاده از آن کلاس باید نمونه ای از همان کلاس را ایجاد کنید. در مقالات بعدی با نمونه سازی کلاس هم آشنا می شوید. اما فعلا همین قدر بدانید که هر کلاسی هنگام نمونه سازی به طور اتوماتیک عکس العملی از خود نشان میدهد. این عکس العمل فراخانی متد مخصوص __init__ است که برای مقدار دهی و اعمال تنظیمات اولیه کلاس به کار میرود. تنظیماتی که ما هنگام ساخت این شی به آن ها احتیاج داشتیم در این متد اعمال شده اند. حتما توجه کرده اید که لسیت پارامترهای هر کدام از متدهای این کلاس با پارامتر self شروع شده است. self ارجاعی به خود کلاس است و وقتی از self استفاده میکنیم یعنی داریم به خود کلاس اشاره میکنیم. تمام متدهای تعریف شده در کلاس باید با پارامتر self شروع شوند تا نشان دهند که به کلاس ما وابسته اند.

در متد __init__ ما با استفاده از self که به خود کلاس اشاره میکند، دو متغیر برای کلاسمان تعریف کردیم. یکی به نام volume که با مقدار ۲۰ ارزش دهی شده است و دیگری currentSong که نشان دهنده ی شماره ی آهنگ فعلی است.

متد بعدی متد setVolume است که باز هم با استفاده از self به متغیر volume اشاره میکند و مقدار آن را تغییر میدهد. هر وقت بخواهیم از درون یکی از متد های کلاسمان متغیری را فراخانی کنیم باید از self برای فراخوانی آن استفاده کنیم. این قانون در مورد فراخانی متدها هم صادق است.

حالا که متوجه اصل ماجرا شده اید حتما خودتان می توانید طرز کار دیگر متدهای کلاس را حدس بزنید. یا بیایید حدس زدن ساختار بقیه متدها را به عنوان تمرین شما به حساب بیاوریم که باید توسط خود شما حل شود. مطمئنا کار ساده ایست.

مفهوم اشیا

استفاده از اشیا

در مقاله ی قبل یاد گرفتیم که چگونه باید توسط ساختار کلاس، اشیا خود را بسازیم. در این مقاله قصد داریم نحوه ی استفاده از اشیایی که ساخته ایم را به شما آموزش دهیم. برای این کار باید اشیای خود را نمونه سازی کنیم یعنی به زبان ساده تر از شی اصلی خود یک کپی بسازیم. مثلا شما هنگامی که نیاز به استفاده از شناسنامه ی خود را داشته باشید اصولا سعی میکنید از کپی آن استفاده کنید و اصل شناسنامه را با خود حمل نکنید. در مورد اشیا هم این قانون حاکم است. شما می توانید تا هر چقدر که دلتان بخواهد از روی شی اصلی خود نمونه سازی کنید.

این نمونه ها به صورت کاملا مستقل عمل می کنند. مثلا اگر از روی شی CdPlayer که در مقاله ی قبل ایجاد کردیم دو نمونه به نام های myPlayer1 و myPlayer2 بسازیم، هیچ کدام از این اشیا به

هنگام استفاده در کار یکدیگر دخالتی نمی کنند. میزان صدا میتواند برای myPlayer1 روی ۴۰ تنظیم شود در حالی که میزان بلندی صدای myPlayer2 روی ۶۰ تنظیم شده است.
برای نمونه سازی از یک کلاس باید به صورت زیر عمل کرد:

```
myPlayer1 = CdPlayer()
myPlayer2 = CdPlayer()
```

با این روش دو نمونه ی کاملا مستقل از روی CdPlayer ایجاد می شود. اگر دقت کنید می بینید که ما شی CdPlayer را به همراه پرانتز احضار کردیم. این یعنی اینکه ما میتوانیم هنگام نمونه سازی از CdPlayer پارامترهایی را نیز را مشخص کنیم. این پارامترها مستقیما به متد مخصوص __init__ که در بدنه ی کلاس تعریف شده است فرستاده می شوند. همانطور که قبلا گفته بودیم هر کلاس به هنگام نمونه سازی خود یک بار این متد را صدا میزند. در مورد شی CdPlayer چون هیچ آرگومان خاصی در متد __init__ مشخص نشده بود پس لازم نیست به هنگام نمونه سازی از روی این شی پارامتری را به آن بفرستیم.

فراخوانی خصوصیات اشیا

حالا ما دو نمونه از شی CdPlayer در اختیار داریم. برای ارتباط برقرار کردن با اشیا باید از خصوصیات آن ها استفاده کرد. خصوصیات همان متغیر ها و متد های تعریف شده برای اشیا هستند. مثلا اگر لازم باشد myPlayer1 آهنگ سوم از سی دی فرضی ما را با بلندی صدای ۵۰ درجه اجرا کند باید اعمال زیر را انجام دهیم:

```
myPlayer1.volume = 50
myPlayer1.play(3)
```

در مثال بالا ما ابتدا توسط عملگر نقطه (.) خصوصیت volume را با ۵۰ تنظیم کردیم. سپس به همین طریق با فراخوانی متد play ، آهنگ سوم از سی دی فرضی خود را اجرا کردیم. البته برای تنظیم صدا می توانستیم از متد setVolume هم استفاده کنیم که کاری مشابه با همین عمل ما را انجام می داد. حالا اگر بخواهیم به آهنگ بعدی پرش کنیم فقط کافی است متد nextSong را صدا بزنیم:

```
myPlayer1.nextSong()
```

همانطور که میبینید ساختار کلاس طوری شی سی دی پلیر را پیاده سازی کرده است که انگار ما در حال استفاده از یک سی دی پلیر واقعی هستیم و متد هایی مثل play هم همانند دکمه های این سی دی پلیر عمل می کنند.

وراثت

وقتی می گوییم کلاس B از کلاس A ارث می برد، یعنی به طور اتوماتیک یه سری از خصوصیات A در کلاس B گنجانده میشود. مثل بچه ای که امکان دارد رنگ چشم یا موهایش را از خانواده اش به ارث ببرد. به مثال زیر توجه کنید:

```
class A(object):

        def sayHello(self):
                print "Hello A!"

class B(A):
        pass
```

```
b = B()
b.sayHello()
```

در مثال بالا A کلاسی است که یک متد تعریف شده به نام sayHello را در خود جای داده است. هنگام تعریف کلاس B ما مشخص کردیم که این کلاس باید از A ارث ببرد، پس به طور اتوماتیک متد sayHello که اصلا در B وجود خارجی ندارد، به کلاس B اضافه میشود.

اگر متدی با همین نام در کلاس B هم تعریف شده بود، دیگر متد sayHello که مربوط به کلاس A است اجرا نمی شد و از متد تعریف شده در کلاس B استفاده میشد. در یک همچین وضعیتی می گوییم متد sayHello ی موجود در کلاس B ، متد sayHello ی موجود در کلاس A را لغو کرده است. باید به این نکته توجه کنیم که چون خود A از شی اصلی object ارث برده است، پس تمام کلاس هایی که از آن ارث می برند – مانند B – هم خود به خود در دسته ی کلاس های سبک جدید پایتون جای میگیرند.

بارگذاری عملگرها

بارگذاری عملگرها در پایتون این توان را به کاربر می دهد که تعیین کند انواع جدید با عملگرهای موجود چگونه استفاده شوند. عملگرهای قابل بارگذاری در پایتون از قرار زیرهستند:

+	-	*	**	/	//	%	<<
>>	&	|	^	~	<	>	<=
>=	==	!=	+=	-=	*=	**=	/=
//=	%=	<<=	>>=	&=	^=	|=	[]
()	.	``	in				

البته باید توجه داشت که در عمل بارگذاری نمی توان خواص عملگرها مانند اولویت ها ، دودویی یا یکانی، شرکت پذیری و... را تغییر داد، یا به خلق عملگر جدیدی پرداخت ؛ برخی عملگرها دو نوع دودویی و یکانی دارند که هر کدام بطور جداگانه تعریف خواهند شد. به ازای هر عملگر در کلاس شئ یک یا چند متد ویژه وجود دارد، زیرا بارگذاری عملگرها از جمله خواصی است که زندگی در پایتون داشتن آنها را ایجاب می کند. که در زیر آنها را می بینید: برخی عملگرهای دودویی(binary) :

```
+         __add__, __radd__
-        __sub__, __rsub__
*        __mul__, __rmul__
/        __div__, __rdiv__, __truediv__, __rtruediv__
//       __floordiv__, __rfloordiv__
%        __mod__, __rmod__
**       __pow__, __rpow__
```

برای بارگذاری اپراتورهای دودویی متد ویژه آنرا با امضای زیر در کلاس شئ مورد نظر تعریف خواهیم کرد:

```
def __methodName__(self, other):
```

عملگرهای یکانی موجود که قابل بار گذاری می باشند به شرح زیر هستند:

```
~,  __invert__
-,   __neg__
+,  __pos__
```

عملگرهای یکانی این گونه تعریف می شوند:

```
def __methodName__(self):
```

مثال

نقطه هندسی:

```
class point:
    """ A geometric 2Dimensional point """

    def __init__(self, Xvalue = 0, Yvalue = 0):

        self.x = float(Xvalue)
        self.y = float(Yvalue)
    #adding conversion to type string
    def __str__(self):
        print "converting point to string"
        return '(%s, %s)'% \
               (str(self.x), str(self.y))
    #adding binary '+' operator
    def __add__(self, other):
        print "adding two points"
        return point(self.x + other.x , self.y + other.y )
    #adding unary '+' operator
    def __pos__(self):
        print "making point positive"
        return point (abs(self.x), abs(self.y))
    #adding unary invert operator
    def __invert__(self):
        print "inverting a point"
        return point(self.y, self.x)

  #adding in-place addition ability
  def __iadd__(self, other):
        return point(self.x + other.x, self.y + other.y)
```

و مشاهده چگونگی عملکرد آنها:

```
>>> p1 = point(-1,-1)
>>> p2 = point(3,4)
>>> print (+p1) + ~p2
making point positive
inverting a point
adding two points
converting point to string
(5.0, 4.0)
>>> p1 += p2
>>> print p1
converting point to string
(2.0, 3.0)
```

منابع:

1. http://www.python.org
2. http://www.pylearn.com
3. Beginning Python by Peter Norton, Alex Samuel, David Aitel, Eric Foster-Johnson, Leonard Richardson, Jason Diamond, Aleatha Parker and Michael Roberts, WROX Press.
4. Python Developer's Handbook by André Dos Santos Lessa, Sams Publishing.