کتاب آموزش برنامه نویسی

Python

مولف:مهندس افشین رفوآ
www.tahlildadeh.com

بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

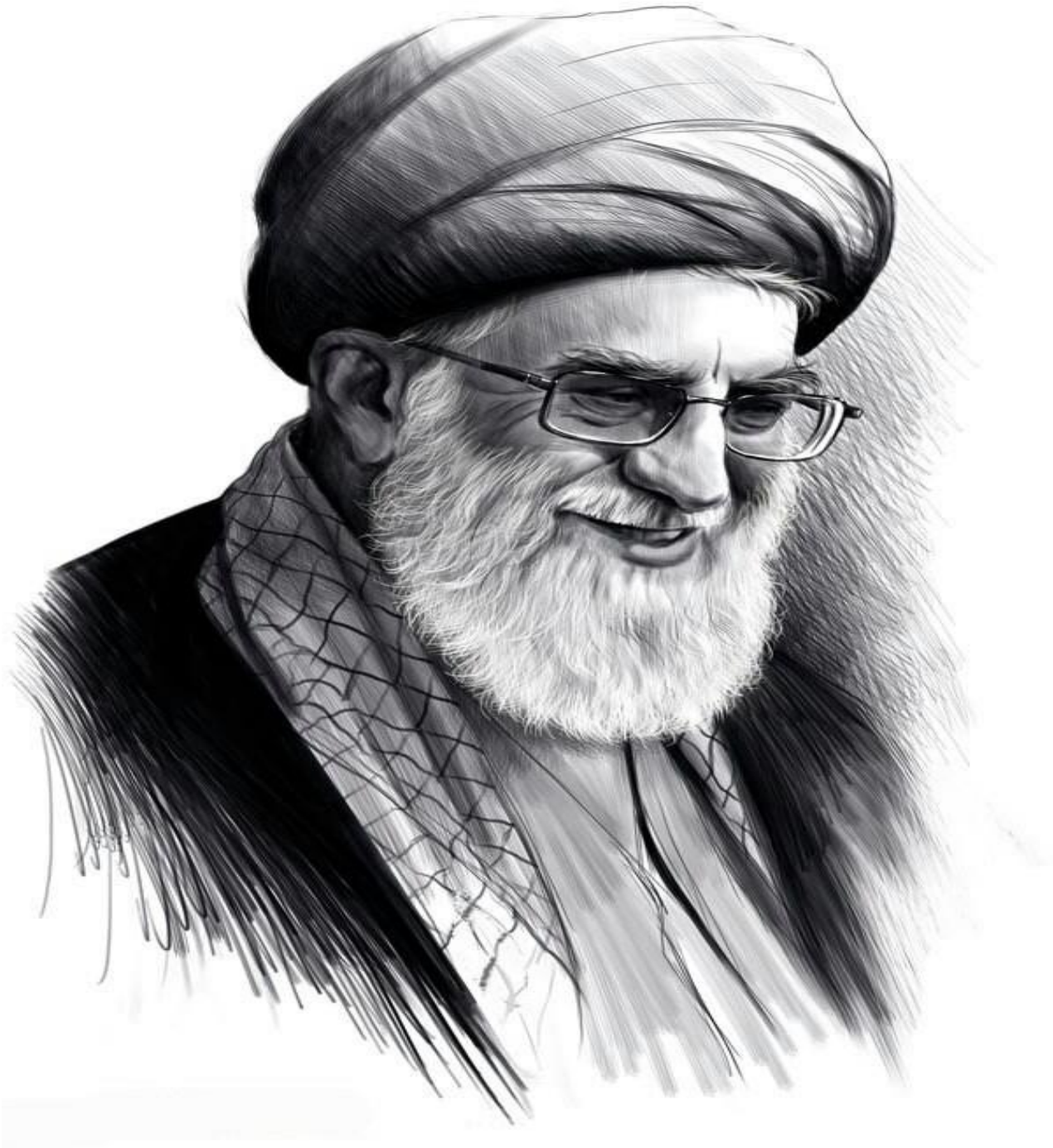
تخصصی ترین مرکز برنامه نویسی و دیتا بیس در ایران

کتاب آموزشی برنامه نویسی اندروید در Android Studio

نویسنده : مهندس افشین رفوآ

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 651 طبقه دوم - واحد7

88146323 - 88446780 - 88146330



تقدیم به نائب امام عصر، حضرت آیت الله خامنه ای که عصا زدنش ضرب آهنگ حیدری دارد

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 651 طبقه دوم - واحد7

88146323 - 88446780 - 88146330

تقدیم به همه ء جویندگان علم که توان و امکان شرکت در کلاس های ما را ندارند

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 651 طبقه دوم - واحد7

88146323 - 88446780 - 88146330

فهرست

زبان برنامه نویسی Python

پایتون به انگلیسی Python یک زبان برنامه‌نویسی همه منظوره، سطح بالا (high-level)، شی گرا (object-oriented) و مفسر (interpreter) است که توسط خودو فان روسوم در سال ۱۹۹۱ در کشور هلند پا به عرصه ی وجود گذاشت.

فلسفه ایجاد آن تاکید بر دو هدف اصلی خوانایی بالای برنامه‌های نوشته شده و کوتاهی و کارایی نسبتا بالای آن است. کلیدواژه های اصلی این زبان به صورت حداقلی تهیه شده‌اند و در مقابل کتابخانه‌هایی که در اختیار کاربر است بسیار وسیع هستند.

بر خلاف برخی از زبان‌های برنامه‌نویسی متادول دیگر که قطعه های کد در {} تعریف می‌شوند (به‌ویژه زبان‌هایی که از دستور نگارشی زبان C پیروی می‌کنند) در زبان پایتون از کاراکتر فاصله و جلوبردن متن برنامه برای مشخص کردن قطعه های کد استفاده می‌شود، بدین معنی که تعدادی یکسان از کاراکتر فاصله در ابتدای سطرهای هر بلاک قرار می‌گیرند، و این تعداد در بلاک‌های کد درونی‌تر افزایش می‌یابد. بدین ترتیب قطعه های کد به صورت پیش فرض ظاهری مرتب خواهند داشت.

پایتون تکنیک های مختلف برنامه نویسی همچون شی گرا و برنامه نویسی دستوری و تابع محور را پشتیبانی می‌کند و برای مشخص کردن نوع متغییرها از یک سیستم داینامیک بهره می گیرد.

این زبان از زبان‌های برنامه نویسی مفسر بوده و به صورت کامل یک زبان شی‌گرا است که در ویژگی‌ها با زبان های تفسیری Perl، Ruby تشابهاتی دارد و از قابلیت مدیریت خودکار حافظه استفاده می‌کند.

پایتون پروژه‌ای آزاد و متن‌باز (open-source) توسعه‌یافته‌است و توسط بنیاد نرم‌افزار پایتون اداره و رهبری می شود.

بررسی اجمالی زبان برنامه نویسی Python

پایتون یک زبان script نویسی سطح بالا، مفسر، تعاملی و شی گرا است. پایتون با هدف خوانایی بالا تعبیه شد. این زبان به طور مکرر از کلمات کلیدی انگلیسی بهره می گیرد (در حالی که زبان های

دیگر اغلب از علائم نگارشی بهره می گیرند) و همچنین ساختار نگارشی که می بایست نوشت نسبت به سایر زبان ها کمتر است (در مقایسه با زبان های دیگر کوتاه است).

1. پایتون تفسیر می شود: این زبان در زمان اجرا توسط مفسر پردازش می شود. بنابراین نیازی نیست شما برنامه را پیش از اجرای آن کامپایل یا ترجمه کنید، مشابه دو زبان PERL و PHP.

2. پایتون تعاملی می باشد: می توانید پای پنجره ی prompt نشسته و مستقیم با مفسر (interpreter) تعامل برقرار کنید و برنامه های خود را بنویسید.

3. پایتون شی گراست: این زبان از مدل برنامه نویسی شی گرا/روش برنامه نوسی که در آن کدها درون اشایی کپسوله سازی می شوند.

4. پایتون نقطه ی شروع مناسبی برای تازه واردان به عرصه ی برنامه نویسی می باشد: پایتون یک زبان بسیار سودمند و کارآمد برای طیف وسعیی از برنامه های کاربردی است که شامل برنامه های پردازش و مدیریت متن و مرورگرها و حتی بازی های رایانه ای نیز می شود.

تاریخچه ی پایتون

همان طور که قبلا ذکر شد، این زبان توسط خودو فان روسوم در اواخر دهه ی 80 و اوایل 1990 در موسسه ی ملی تحقیقات علوم ریاضی و کامپیوتر در هلند توسعه یافت.

پایتون از زبان هایی همچون ABC، Modula-3، C، C++، Algol-68، SmallTalk، Unix shell و دیگر زبان های اسکریپت نویسی مشتق شده است.

کد منبع این زبان مانند زبان Perl تحت لیسانس GNU در اختیار عموم قرار می گیرد.

پایتون هم اکنون توسط تیم برنامه نویسی در موسسه ی مزبور پشتیبانی و مدیریت می شود، اما مخترع آن هنوز نقش اساسی در هدایت پیشرفت آن ایفا می کند.

ویژگی ها و امکانات Python

1. یادگیری آسان آن: محدود بودن تعداد کلیدواژه ها، همچنین ساختار و دستور نگارشی ساده ی آن نقش اساسی در یادگیری سریع این زبان بازی می کند.

2. خوانایی بالا: کد پایتون فوق العاده صریح تعریف شده و خواندن آن سهل می باشد.
3. نگهداشت آن بسیار آسان می باشد: نگهداشت کد منبع این زبان بسیار آسان می باشد.
4. دارای کتابخانه ی بسیار گسترده می باشد: کتابخانه ی پایتون portable (دارای نصب آسان) بوده و قابلیت استفاده (سازگاری) در محیط های مختلف همچون UNIX، Windows و Macintosh را دارد.
5. پشتیبانی از قابلیت تعامل با برنامه نویس (interactive mode): پایتون از interactive mode پشتیبانی می کند: به این معنی که به برنامه نویس اجازه می دهد تکه های کد را به صورت تعاملی مورد آزمایش قرار داده و اشکال زدایی کند.
6. قابلیت نصب آسان (portable): پایتون می تواند بر روی طیف وسیعی از محیط های سخت افزاری (hardware platform) اجرا شود و دارای یک رابط می باشد که برای تمامی محیط ها یکسان است.
7. توسعه پذیری (Extendable): می توان ماژول های سطح پایین به مفسر پایتون اضافه نمود. ماژول های مزبور به برنامه نویس این امکان را می دهند که به ابزارهای در دست افزوده یا آن ها را طبق نیاز خود سفارشی تنظیم کنند تا بازدهی و کارایی بیشتری دریافت کند.
8. پایگاه داده: پایتون رابط هایی (interface) را برای تمامی پایگاه داده های تجاری پرطرفدار ارائه می دهد.
9. برنامه نویسی GUI (رابط گرافیکی کاربری): با زبان پایتون می توان رابط کاربری طراحی کرد (برنامه های GUI نوشت) و به system call ها، کتابخانه ها و windows system های متعددی نظیرWindows MFC ، Macintosh و سیستم پنجره X (یک سامانهٔ نرم‌افزاری و پروتکل تحت شبکه است که اساس قابلیت‌های رابط‌های گرافیکی کاربری (GUI) و دستگاه‌های ورودی پیشرفته را برای رایانه‌های تحت شبکه فراهم می‌کند) انتقال داد.
10. مقیاس پذیر (scalable): پایتون پشتیبانی و ساختار بهتری را برای برنامه های با مقیاس بزرگ در مقایسه با shell script (یک برنامه رایانه ای که برای اجرا با مفسر خط فرمان Unix shell) ارائه می دهد.

علاوه بر ویژگی های نام برده، پایتون قابلیت ها و امکانات بیشتری را ارائه می دهد که در زیر تعدادی از آن ها فهرست شده:

1. جدا از مدل برنامه نویسی شی گرا، از روش های تابع محور و دستوری (functional & structured) برنامه نویسی نیز پشتیبانی می کند.
2. می توان آن را به صورت یک زبان اسکریپت نویسی مورد استفاده قرار داد یا آن را برای برنامه های حجیم به byte-code ترجمه کرد.
3. نوع داده های پویا سطح بالا ارائه کرده و از قابلیت بررسی پویا نوع پشتیبانی می کند.
4. از قابلیت garbage collection خودکار پشتیبانی می کند.
5. می توان آن را به آسانی با C، C++، COM، ActiveX، COBRA و Java ترکیب کرد.

نصب و راه اندازی محیط برنامه نویسی Python

پایتون را می توان بر روی طیف وسیعی از محیط ها همچون Linux، Mac OS X اجرا کرد. ابتدا بایستی به نحوه ی نصب محیط برنامه نویسی این زبان بپردازیم.

نصب محیط محلی برنامه نویسی

برای اینکه پی ببریم آیا محیط پایتون نصب شده و اگر نصب شده کدام ویرایش آن قابل استفاده می باشد، terminal window باز کرده و واژه ی "python" را وارد کنید:

Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, etc.)

Win 9x/NT/2000

Macintosh (Intel, PPC, 68K)

OS/2

DOS (multiple versions)

PalmOS

Nokia mobile phones

Windows CE

Acorn/RISC OS

BeOS

Amiga

VMS/OpenVMS

QNX

VxWorks

Psion

پایتون همچنین به ماشین های مجازی Java و .NET منتقل شده است.

دریافت Python

جهت دسترسی به بروز ترین کد منبع پایتون، کدهای binary ، مستندسازی، اخبار آن می توانید به وب سایت رسمی به آدرس http://www.python.org/ مراجعه نمایید.

می توانید مستندسازی پایتون را از سایت زیر دریافت کنید. مستند سازی آن در تمامی فرمت های موجود، اعم از HTML، PDF و PostScript قابل دسترسی می باشد.

نصب python

پایتون ویژه ی طیف گسترده ای از محیط ها (platform) ارائه و توزیع شده. کافی است کد باینری سازگار با محیط خود را بارگیری کرده و آن را نصب کنید.

در صورت فراهم نبودن کد باینری پایتون قابل اجرا بر روی محیط خود، لازم است با استفاده از کامپایلر C، کد منبع را خود به صورت دستی ترجمه و به زبان ماشین برگردانید. ترجمه ی کد منبع در خصوص انتخاب امکانات مورد نیاز در برنامه ی کاربردی، انعطاف پذیری بیشتری را ارائه می دهد.

در زیر به نحوه ی نصب پایتون بر روی محیط های گوناگون خواهیم پرداخت:

جهت نصب پایتون بر روی ماشینی که سیستم عامل آن Linux/Unix است، می بایست گام های زیر را دنبال کنید:

1. مرورگر دلخواه را راه اندازی کرده و به آدرس http://www.python.org/download/ مراجعه نمایید.
2. لینکی که کد منبع برای Linux/Unix را به صورت zip شده ارائه می دهد، دنبال کنید.
3. فایل های مربوطه را بارگیری کرده، سپس از حالت فشرده استخراج نمایید.
4. برای تنظیم سفارشی برخی از گزینه ها، فایل Modules/Setup را ویرایش کنید.
5. اسکریپت configure/. را اجرا کنید.
6. حال نصب را انجام دهید.

در پایان، پایتون در مسیر پیش فرض /usr/local/bin نصب شده و کتابخانه های آن نیز در صورتی که نسخه ی مورد استفاده ی پایتون XX باشد، در مسیر /usr/local/lib/pythonXX جای می گیرد.

جهت نصب پایتون بر روی سیستم عامل ویندوز، مراحل زیر را دنبال کنید:

1. مرورگر را باز کرده و به آدرس http://www.python.org/download/ پیمایش کنید.
2. لینکی که فایل نصبی پایتون سازگار با ویندوز پایتون را ارائه می دهد (فایل Windows installer python-XYZ.msi) را دنبال کنید. نسخه ای که بایستی نصب کنید XYZ می باشد.
3. برای اینکه ویندوز بتواند از این فایل استفاده کند، می بایست Microsoft Installer 2.0 بر روی آن نصب شده و پشتیبانی شود. کافی است فایل installer را بر روی ماشین نصب کرده و آن را اجرا کنید تا مطمئن شوید رایانه ی شما از MSI پشتیبانی می کند.
4. فایل دانلود شده را اجرا کنید. راهنمای نصب (install wizard) پایتون نمایش داده می شود. کافی است تنظیمات پیش فرض را پذیرفته و صبر کنید تا فرایند نصب به پایان برسد.

تنظیم مسیر

برنامه ها و دیگر فایل اجرایی می توانند در پوشه های مختلفی قرار داشته باشند. سیستم عامل جهت سهولت در دسترسی به این فایل ها یک مسیر جستجو ارائه نموده که فهرست پوشه های حاوی فایل های اجرایی پایتون را نمایش می دهد.

مسیر در یک متغیر محیطی (Environment variable مجموعه‌ای از مقادیر نام‌گذاری شده هستند که می توانند نحوه رفتار کردن فرایند های در حال اجرا را تغییر داده و بر روی آنها اثر بگذارند.) ذخیره می شود. این متغیر محیطی

یک رشته ی نام گذاری شده است که توسط سیستم عامل نگهداری می شود. متغیر نام برده دربردارنده ی اطلاعاتی است که در دسترس command shell (یک برنامه ی مجزا و مستقل که ارتباط مستقیم بین کاربر و سیستم عامل را فراهم می نماید) و سایر برنامه ها می باشد.

متغیر path در سیستم عامل Unix ، PATH و در ویندوز Path نام گذاری می شود(Unix حساس به کوچک و بزرگی حروف است و ویندوز نیست).

در سیستم عامل Mac تمامی جزئیات مربوط به مسیر قرار گیری فایل ها توسط installer مدیریت می شود. به منظور فراخوانی مفسر (interpreter) پایتون از هر پوشه ای، می بایست پوشه ی Python را به مسیر خود اضافه کنید.

تنظیم مسیر در محیط Unix/Linux

به منظور افزودن پوشه ی پایتون به مسیر مورد نظر برای یک session در Unix، گام های زیر را دنبال می کنیم:

1. در csh shell: عبارت setenv PATH "$PATH:/usr/local/bin/python" را وارد کرده و کلید Enter فشار دهید.
2. در bash shell (Linux): عبارت export PATH="$PATH:/usr/local/bin/python" را وارد نموده و Enter را فشار دهید.
3. در sh یا ksh shell : عبارت PATH="$PATH:/usr/local/bin/python" را وارد کرده، سپس Enter را بزنید.

نکته: /usr/local/bin/python در واقع مسیر قرار گیری پوشه ی پایتون می باشد.

تنظیم مسیردر سیستم عامل ویندوز

جهت افزودن پوشه ی پایتون به مسیر مورد نظر برای یک session در محیط ویندوز، مراحل زیر را دنبال کنید:

1. در خط فرمان (command prompt): عبارت path %path%;C:\Python را تایپ کرده و Enter را فشار دهید.

نکته: C:\Python مسیر قرارگیری پوشه ی پایتون می باشد.

متغیرهای محیطی پایتون (environment variable)

متغیرهای محیطی که توسط پایتون شناخته و پشتیبانی می شود، به شرح زیر می باشد:

متغیر	شرح
PYTHONPATH	نقشی مشابه نقش ای که PATH ایفا می کند را بازی می کند. این متغیر به مفسر پایتون اطلاع می دهد کجا می تواند فایل های ماژول وارد شده به یک برنامه را پیدا کرده و به آن ها دسترسی داشته باشد. متغیر ذکر شده بایستی پوشه ی دربردارنده ی کتابخانه ی منبع (source library directory) پایتون و همچنین پوشه های حاوی کد منبع/سورس کد پایتون را شامل شود. PYTHONPATH گاهی توسط installer پایتون از پیش تنظیم شده است.
PYTHONSTARTUP	دربردارنده ی مسیر فایل آغاز سازی (initialization) است که آن فایل حاوی کد منبع پایتون می باشد.این متغیر هر بار که مفسر را راه اندازی می کنید، اجرا می شود. PYTHONSTARTUP درسیستم عامل Unix، بدین نام می باشد: .pythonrc.py و دربرگیرنده ی دستوراتی است که برنامه های کاربردی (utilities) را بارگذاری کرده یا متغیر PYTHONPATH را modify می کنند.
PYTHONCASEOK	با PYTHONCASEOK در محیط ویندوز می توان به پایتون دستور داد که اولین نمونه یا مورد منطبق غیر حساس به کوچک و بزرگی حروف در یک دستور import را پیدا کند. با تنظیم این متغیر بر روی هر مقداری می توان آن را فعال ساخت.
PYTHONHOME	این متغیر یک جایگزین برای module search path (مسیر جستجو ماژول) می باشد. متغیر مزبور معمولا در پوشه های PYTHONSTARTUP یا PYTHONPATH گنجانده می شود تا بدین وسیله پروسه ی عوض کردن کتابخانه های ماژول آسان گردد.

راه اندازی پایتون

پایتون را می توان به سه روش زیر راه اندازی کرد:

1. مفسر تعاملی (Interactive Interpreter): پایتون را می توان از طریق Unix، Dos یا هر سیستم دیگری که مفسر خط فرمان (command-line interprator) یا shell window فراهم می نماید، اجرا و راه اندازی کرد.

واژه ی پایتون را در خط فرمان وارد کنید.

حال می توانید در مفسر تعاملی شروع به کدنویسی کنید.

```
$python          # Unix/Linux
or
python%          # Unix/Linux
or
C:>python        # Windows/DOS
```

در جدول زیر تمامی دستورات و گزینه های قابل استفاده ی خط فرمان را مشاهده می کنید:

دستور	شرح
-d	خروجی را به صورت debug ارائه می دهد.
-O	bytecode بهینه سازی شده ایجاد می کند (خروجی آن فایل هایی است که دارای پسوند pyo. می باشد).
-S	از دستور import site برای جستجوی مسیر پایتون هنگام شروع استفاده نکنید.
-v	خروجی طولانی (گزارشات trace با جزئیات درباره ی دستورات import)
-X	استثناهای مبتنی بر کلاس توکار را غیرفعال می کند(فقط بایستی رشته استفاده کنید)؛ از ویرایش 1.6 به بعد منسوخ شد.
-c cmd	اسکریپت ارسالی را به صورت رشته ی cmd اجرا می کند.

اسکریپت پایتون را از یک فایل مشخص اجرامی کند.	**file**

2. اجرای اسکریپت پایتون از خط فرمان (command-line): یک اسکریپت پایتون را می توان در خط فرمان با فراخوانی مفسر برای برنامه ی خود اجرا کرد:

```
$python  script.py          # Unix/Linux
or
python% script.py           # Unix/Linux
or
C:>python script.py         # Windows/DOS
```

نکته: مطمئن شوید permission mode فایل به شما اجازه ی اجرای فایل را می دهد.

3. با بهره گیری از محیط برنامه نویسی یکپارچه (IDE): می توانید پایتون را در یک محیط رابط گرافیکی کاربری (GUI) نیز اجرا کنید، البته اگر یک برنامه ی GUI بر روی رایانه ی خود نصب دارید که از پایتون پشتیبانی می کند.

Unix: IDLE اولین IDE یا محیط یکپارچه ی برنامه نویسی است که برای پایتون عرض شده است.

Windows: PythonWin نیز اولین interface یا رابط ای است که ویژه ی پایتون برای محیط ویندوز ارائه شده که علاوه بر محیط توسعه، یک رابط گرافیکی کاربری نیز محسوب می شود.

Macintosh: پایتون ارائه شده ویژه ی محیط Mac را می توانید به همراه محیط برنامه نویسی IDLE از وب سایت رسمی آن به صورت فایل های MacBinary یا BinHex دریافت کنید.

اگر موفق نشدید محیط را به درستی تنظیم و راه اندازی کنید، می توانید از system admin کمک بگیرید. بایستی محیط را به درستی راه اندازی کرده و عملکرد صحیح آن اطمینان حاصل نمایید.

ساختار نگارشی پایه پایتون (basic syntax)

زبان پایتون شباهت های زیادی به زبان های Perl، C و Java دارد. با این حال، تفاوت های اساسی هم بین این زبان ها وجود دارد.

اولین برنامه ی پایتون

برنامه نویسی خود را از روش های زیر انجام می دهیم.

برنامه نویسی با فراخوانی مفسر

اگر مفسر را بدون ارسال یک فایل اسکریپت به آن به عنوان پارامتر، صدا بزنید با prompt (کادری حاوی دستورات) زیر مواجه خواهید شد:

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

دستور زیر را وارد کنید و کلید Enter را فشار دهید:

```
print "Hello, Python!"
```

در صورت استفاده از نسخه ی جدید پایتون، می بایست همراه با دستور Print از () استفاده کنید، بدین صورت: ;("Hello, Python!")print. در ویرایش 2.4.3 این زبان، نتیجه ی زیر حاصل می شود:

```
"Hello, Python!"
```

فراخوانی مفسر با یک اسکریپت به عنوان پارامتر

فراخوانی مفسر (interpreter) با یک اسکریپت به عنوان پارامتر، با اجرای آن اسکریپت آغاز شده و تا زمانی که اسکریپت به پایان نرسیده ادامه می یابد. هنگامی که اسکریپت به پایان می رسد، مفسر غیرفعال می شود.

در این بخش یک برنامه ی ساده ی پایتون در یک اسکریپت می نویسیم. فایل های پایتون دارای پسوند .py می باشد.

کد زیر را در یک فایل به نام test.py تایپ کنید:

```
print "Hello, Python!"
```

برای این منظور لازم است مفسر پایتون را در متغیر PATH تنظیم کرده باشید. اکنون برنامه را بدین ترتیب اجرا کنید:

```
$ python test.py
```

نتیجه ی زیر بدست می آید:

```
Hello, Python!
```

حال اسکریپت پایتون را از روش دیگری اجرا می کنیم.

```
#!/usr/bin/python
print "Hello, Python!"
```

پیش از کامپایل، می بایست مفسر پایتون را در پوشه ی /usr/bin آماده داشته باشید. اکنون برنامه را بدین صورت اجرا می کنیم:

```
$ chmod +x test.py    # This is to make file executable
$./test.py
```

نتیجه ی زیر حاصل می گردد:

```
Hello, Python!
```

شناسه ها در پایتون

شناسه یا identifier یک اسم است که به منظور شناسایی متغیر، تابع، کلاس، ماژول یا دیگر اشیا به آن ها تخصیص داده می شود. شناسه با یک حرف از A تا Z (یا a تا z) یا زیرخط (_) آغاز شده و به دنبال آن یک یا چند صفر، حرف، زیرخط و عدد قرار می گیرد.

استفاده از علائم نگارشی همچون @، $ و % در شناسه مجاز نمی باشد. پایتون یک زبان حساس به کوچک و بزرگی حروف است. از این رو، دو واژه ی Manpower و manpower دو شناسه ی کاملا متفاوت از یکدیگر هستند.

قوانین نام گذاری توابع، متغیرها و اشیا در زیر فهرست شده:

1. اسم کلاس ها با حرف بزرگ آغاز می شود. دیگر شناسه ها با حرف کوچک شروع می شود.
2. آغاز کردن یک شناسه با زیرخط (_)، بیانگر این است که آن شناسه private می باشد.
3. آغاز کردن یک شناسه با دو زیرخط نشانگر strongly private بودن آن شناسه است.
4. اگر شناسه ای با دو زیرخط پشت سرهم پایان یابد، در آن صورت شناسه ی مورد نظر اسم خاص language defined می باشد.

کلمات رزرو شده

لیست زیر کلمات رزرو شده ی زبان پایتون را نمایش می دهد. این کلمات را نمی توان به عنوان اسم متغیر، ثابت (constant) یا هر چیز دیگر استفاده کرد. لازم به ذکر است که تمامی کلیدواژه های پایتون تماما با حروف کوچک نوشته می شوند.

And	**exec**	**Not**
Assert	finally	or
Break	for	pass
Class	from	print
Continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

خطوط فاصله و تورفتگی

پایتون از {} برای مشخص کردن قطعه کدهای تعریف کلاس، تابع یا جریان کنترل داده (flow control) در ارتباطات داده‌ای، کنترل جریان یک فرایند مدیریت نرخ انتقال بین دو گره برای جلوگیری از ارسال از طرف فرستنده سریع به دریافت کننده کند است. کنترل جریان مکانیزمی را برای دریافت کننده جهت کنترل سرعت انتقال فراهم می‌کند.) استفاده نمی کند. قطعه کدها با استفاده از تورفتگی مشخص می شوند. از این رو در استفاده از آن بایستی بسیار دقیق بود.

مقدار فضای خالی در تورفتگی ها متغیر است، اما تمامی دستورات درون قطعه کد بایستی به یک اندازه توگذاشته شوند. مثال:

```
if True:
print "True"
else:

print "False"
```

اما قطعه کد زیر خطا می دهد:

```
if True:
print "Answer"
print "True"
else:
print "Answer"

print "False"
```

بنابراین، در زبان پایتون تمامی خطاهای متوالی که به یک اندازه توگذاشته شده اند، در کل یک قطعه کد را تشکیل می دهند. اگر با دقت به مثال زیر دقت کنید، می بینید که چندین قطعه کد در آن وجود دارد:

```
#!/usr/bin/python
import sys
try:
# open file stream
file = open(file_name, "w")
except IOError:
print "There was an error writing to", file_name
sys.exit()
print "Enter '", file_finish,
print "' When finished"
while file_text != file_finish:
file_text = raw_input("Enter text: ")
if file_text == file_finish:
# close the file
file.close
break
file.write(file_text)
file.write("\n")
file.close()
file_name = raw_input("Enter filename: ")
if len(file_name) == 0:
print "Next time please enter something"
sys.exit()
try:
file = open(file_name, "r")
except IOError:
print "There was an error reading file"
```

```
sys.exit()
file_text = file.read()
file.close()
print file_text
```

دستورهای چندخطی

دستورات پایتون معمولا به خط جدید ختم می شوند (یک خط تمام شده و خط دیگری شروع می شود). کاراکتر (\) در انتهای خط نشانگر ادامه ی دستور مورد نظر در خط دیگر است. مثال:

```
total = item_one + \
item_two + \
item_three
```

دستوراتی که داخل {}، [] یا () قرار می گیرند نیازی به کاراکتر (\) برای نشان دادن اینکه ادامه دستور در خط بعدی قرار گرفته، ندارد.

```
days = ['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday']
```

علامت نقل و قول یا کوتیشن در پایتون

می توان در پایتون از تک کوتیشن (')، دابل کوتیشن (") و سه کوتیشن با هم (''' یا """) استفاده کرد. مقداری که درون این علامت ها قرار می گیرد، یک رشته ی نوشتاری (string literal) را تشکیل می دهد. بایستی دقت داشت که یک رشته با یک نوع کوتیشن آغاز شده و با همان نوع نیز پایان می یابد.

از ''' یا """ برای محصور کردن یک رشته که در چندین خط ادامه دارد استفاده می شود. تمامی نمونه های زیر مجاز و صحیح هستند:

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comment (توضیحات) در پایتون

علامت # اگر داخل کوتیشن محصور نباشد، نشانگر شروع comment خواهد بود. تمام کاراکترهایی که پس از # قرار می گیرند تا پایان آن خط بخشی از توضیح محسوب می شوند و مفسر پایتون آن ها را نادیده گرفته و اجرا نمی کند.

```
#!/usr/bin/python
# First comment
print "Hello, Python!" # second comment
```

خروجی کد بالا:

```
Hello, Python!
```

می توانید یک comment را درست بعد از یک دستور یا عبارت شروع کرد، بدین نحو:

```
name = "Madisetti" # This is again comment
```

می توانید چند خط را به صورت comment دربیاورید، بدین شکل:

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

استفاده از خطوط تهی

خطی که چیزی به جز فضای خالی در آن بکار نرفته و احیانا دارای یک comment می باشد، درواقع یک خط تهی (blank space) محسوب می شود و مفسر پایتون آن را کاملا نادیده می گیرد.

در یک session که برنامه نویس مستقیما با مفسر تعامل دارد، می بایست بین این دستور و دستور بعدی یک خط خالی فاصله بیاندازید.

منتظر کاربر بودن

خط زیر یک پنجره ی prompt حاوی دستور "Press the enter key to exit" نمایش می دهد و منتظر کاربر می ماند تا اقدامات لازم را انجام دهد:

```
#!/usr/bin/python
raw_input("\n\nPress the enter key to exit.")
```

در اینجا، دستور "\n\n" دو خط جدید ایجاد کرده، سپس خود خط را نمایش می دهد.

پس از اینکه کاربر کلید Enter را فشار می دهد، برنامه پایان می یابد. با این روش می توان پنجره ی console را تا زمانی که کاربر کارش با برنامه تمام نشده، باز نگه داشت.

چندین دستور در یک خط

با استفاده از نقطه ویرگول (;) می توان چندین دستور را در یک خط واحد جای داد، لازم به ذکر است هیچ یک از دستورات مزبور یک قطعه کد مجزا را تشکیل نمی دهد. در زیر یک تکه کدی را مشاهده می کنید که از نقطه ویرگول در آن استفاده شده است:

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

suite یا مجموعه دستورات در پایتون

یک گروه از دستورات منفرد، که در مجموع یک قطعه کد مجزا را تشکیل می دهند در زبان پایتون به اصطلاح suite اطلاق می گردند. دستورات مرکب نظیر if، while، def و class نیازمند یک خط سرآیند (header line) و یک suite هستند.

خطوط سرآیند (header) با یک دستور (یک کلیدواژه) آغاز می شود و با یک دونقطه (:) به پایان می رسد، همچنین به دنبال آن یک یا چندین خط قرار می گیرد که suite (مجموعه دستور) را تشکیل می دهد. نمونه:

```
if expression :
    suite
elif expression :
    suite
else :
    suite
```

آرگومان های خط فرمان (command-line arguments)

بسیاری از دستورات را می توان اجرا کرد تا اطلاعات ابتدایی دررابطه با نحوه ی اجرای برنامه در اختیار شما قرار دهد. پایتون با فراهم نمودن دستور h–، این امکان را به شما می دهد:

```
$ python -h
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser (also PYTHONDEBUG=x)
-E     : ignore environment variables (such as PYTHONPATH)
-h     : print this help message and exit
```

[etc.]

انواع متغیر (variable type) در پایتون

متغیر صرفا فضاهای رزرو شده در حافظه هستند که مقادیری را در آن ها ذخیره می کنیم، بدین معنا که در زمان ایجاد یک متغیر، بخشی از حافظه اشغال شده و به آن متغیر تخصیص داده می شود.

بسته به نوع داده ای متغیر، مفسر بخشی از حافظه را رزرو کرده و تصمیم می گیرد چه مقداری در حافظه ی تخصیص داده شده، ذخیره گردد. بنابراین، با تخصیص نوع داده های مختلف به متغیرها، می توانید اعداد صحیح (integer)، اعداد اعشاری (decimal) یا character در این متغیرها ذخیره کنید.

تخصیص مقادیر به متغیرها

برای تخصیص حافظه، در پایتون نیازی به اعلان صریح متغیر نیست. زمانی که مقداری را به متغیر انتساب می دهد، اعلان به صورت خودکار رخ می دهد. مانند زبان های برنامه نویسی دیگر، انتساب مقدار به متغیر توسط علامت مساوی " = " صورت می پذیرد.

عملوندی (operand) که در سمت چپ عملگر (operator) " = " قرار می گیرد، اسم متغیر و عملوندی که در سمت راست عملگر " = " قرار می گیرد، مقداری است که در متغیر ذخیره می شود. مثال:

```
#!/usr/bin/python
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
print counter
print miles
print name
```

در اینجا، 100، 1000.0و "John" همگی مقادیر تخصیص داده شده به ترتیب به متغیرهای counter، miles و name هستند. کد بالا نتیجه ی زیر را بدست می دهد:

```
100
1000.0
John
```

چندین تخصیص به صورت یکجا

پایتون به شما امکان می دهد یک مقدار را همزمان به چندین متغیر تخصیص دهید. مثال:

```
a = b = c = 1
```

در اینجا، یک شی integer ایجاد شده سپس مقدار 1 در آن ذخیره گردیده. همان طور که مشاهده می کنید هر سه متغیر به یک مکان واحد در حافظه تخصیص داده شده اند. همچنین می توان چندین شی همزمان به چندین متغیر انتساب داد. مثال:

```
a, b, c = 1, 2, "john"
```

در این مثال، سه مقادیر دو عدد صحیح و یک رشته به ترتیب به متغیرهای a، b و c تخصیص داده شده اند.

نوع داده های رایج

داده های ذخیره شده در حافظه می توانند از هر نوعی باشند. به عنوان مثال، سن یک شخص به صورت یک مقدار عددی ذخیره شده و آدرس وی در قالب حروف الفبا (alpha-numeric) داخل حافظه ذخیره می شود. پایتون دارای نوع داده های متعددی است که عملیات ممکن بر روی آن ها و روش های ذخیره ویژه ی هر یک را تعریف می کند.

در کل زبان پایتون از نوع داده های زیر پشتیبانی می کند:

1. نوع عددی
2. نوع رشته ای
3. لیست
4. tuple(نوع داده ی چندتایی)

5. Dictionary

اعداد یا نوع عددی در پایتون

نوع داده های عددی قادرند مقادیر عددی در خود نگه دارند. اشیا number زمانی ایجاد می شوند که مقداری را به آن تخصیص دهید. برای مثال:

```
var1 = 1
var2 = 10
```

می توانید ارجاع (reference) به یک شی عددی را با استفاده از دستور del پاک کنید. نحوه ی نگارش این دستور به شکل زیر است:

```
del var1[,var2[,var3[....,varN]]]
```

می توانید با استفاده از دستور del یک یا چند شی را حذف کنید. مثال:

```
del var
del var_a, var_b
```

پایتون نوع های عددی زیر را پشتیبانی می کند:

1. int (اعداد صحیح علامت دار)
2. long (اعداد صحیح بسیار بزرگ یا long integers، آن ها را می توان به صورت شانزده شانزدهی و هشت هشتی نمایش داد)
3. float (مقادیر حقیقی میمز شناور)
4. complex (اعداد مختلط)

مثال:

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j

-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

1. پایتون به شما اجازه می دهد برای مشخص کردن نوع عددی Long از l کوچک استفاده کنید، اما برای اینکه آن l با عدد 1 اشتباه گرفته نشود، توصیه می کنیم از L بزرگ استفاده کنید. بنابراین پایتون اعداد صحیح بسیار بزرگ را با L بزرگ نمایش می دهد.
2. یک عدد مختلط متشکل است از دو عدد ممیز شناور حقیقی و یک بخش که یکه ی موهومی نام دارد. برای مثال در yj + x، x و y اعداد حقیقی هستند و j نشانگر واحد یا یکه ی موهومی (imaginary) می باشد.

رشته ها در پایتون

رشته ها در پایتون عبارت است از مجموعه کاراکترهای همجوار که در علامت نقل و قول نمایش داده می شوند. پایتون از هر دو شکل تک کوتیشن و دابل کوتیشن پشتیبانی می کند. می توان با بهره گیری از عملگر برش یا slice operator ([:] [ ]) که اندیس آن در آغاز رشته با اندیس 0 شروع شده و تا 1- در انتها ادامه می یابد، بخش هایی از یک رشته را استخراج کرد.

علامت (+) یک عملگر اتصال است که دو رشته را به هم پیوند می دهد. علامت * درواقع یک repetition operator است که دستوری را تکرار می کند (برای مثال یک رشته را دوبار چاپ می نمایند).

```
#!/usr/bin/python
str = 'Hello World!'
print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

نتیجه ی زیر حاصل می گردد:

```
Hello World!
```

```
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

نوع داده ای List در پایتون

از میان نوع های داده ای پایتون، List ها تطبیق پذیرترین نوع داده ای هستند، بدین معنا که برای منظورهای مختلف می توان از آن ها بهره گرفت. یک لیست شامل مجموعه ای از آیتم ها است که توسط ویرگول از هم جدا شده و داخل [] محصور می شوند. تا حدی می توان گفت که List شبیه به نوع داده ای آرایه در زبان C است. یک تفاوت اساسی بین آرایه و لیست این است که آیتم های موجود در لیست می توانند از نوع داده های مختلف باشند (از نظر نوع با هم متفاوت باشند).

مقادیر ذخیره شده در یک لیست را می توان با استفاده از عملگر برش ([:] [ ]) از طریق اندیس که از در ابتدای لیست ار صفر آغاز شده و تا 1- در انتهای لیست ادامه می یابد، مورد دسترسی قرار داد. علامت + به عنوان یک عملگر اتصال نقش ایفا کرده و عملگر * نیز صرفا یک دستور را تکرار می کند. مثال :

```
#!/usr/bin/python
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd till 3rd
print list[2:]      # Prints elements starting from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```

کد فوق، نتیجه ی زیر را بدست می دهد:

```
['abcd', 786, 2.23, 'john', 70.200000000000003]
abcd
[786, 2.23]
[2.23, 'john', 70.200000000000003]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']
```

نوع داده ای Tuple در پایتون

Tuple نیز یک نوع داده ای متشکل از رشته یا مجموعه ای از آیتم هاست که مشابه نوع داده ای List می باشد. یک Tuple تعدادی مقادیر را در خوش دارد که این مقادیر توسط ویرگول از هم جدا می شوند. اما برخلاف List، نوع داده ای Tuple داخل پرانتز محصور می شود.

بین نوع داده ای مذکور تفاوت هایی وجود دارد: در List مقادیر درون [] جای می گیرند، در حالی که این مقادر در tuple داخل پرانتز محصور می شوند. تفاوت دیگر این است که المان های List و اندازه ی آن را می توان اصلاح نمود ولی این امکان برای tuple وجود ندارد. می توان به tuple به چشم list های فقط خواندنی (read-only) نیز نگریست. مثال :

```
#!/usr/bin/python
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')
print tuple               # Prints complete list
print tuple[0]            # Prints first element of the list
print tuple[1:3]          # Prints elements starting from 2nd till 3rd
print tuple[2:]           # Prints elements starting from 3rd element
print tinytuple * 2       # Prints list two times
print tuple + tinytuple # Prints concatenated lists
```

نتیجه ی زیر حاصل می گردد:

```
('abcd', 786, 2.23, 'john', 70.200000000000003)
abcd
(786, 2.23)
(2.23, 'john', 70.200000000000003)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')
```

کد زیر از آنجایی که سعی می کند المان های tuple را بروز رسانی کند، مجاز و معتبر نیست. اما همین عملیات را می توان بر روی List پیاده کرد:

```
#!/usr/bin/python
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
list = [ 'abcd', 786 , 2.23, 'john', 70.2  ]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000     # Valid syntax with list
```

نوع داده ای Dictionary

Dictionary در پایتون تا حدی شبیه به جداول hash (hash table type) هستند. این نوع داده ای علمکردی مشابه آرایه های شرکت پذیر - associative array - یا hash ها در Perl دارند و از جفت های کلید مقدار (key-value pairs) تشکیل می شوند. کلید می تواند از هر نوعی باشد، با این وجود اغلب از نوع اعداد و رشته ها هستند. اما مقادیر، می توانند از هر شی دلبخواه و اختیاری در پایتون باشند.

آیتم های Dictionary داخل {} محصور می شوند. جهت دسترسی و استخراج مقادیری از dictionary می بایست از [] استفاده کرد. مثال :

```
#!/usr/bin/python
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

نتیجه ی زیر را ارائه می دهد:

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

در Dictionary، المان ها دارای ترتیب یا order مشخصی نیستند.

تبدیل نوع داده ای

گاهی لازم است بین نوع داده های درون ساخته یا توکار پایتون، عملیات تبدیل انجام داد و نوعی را به نوع دیگر تبدیل نمود. کافی است اسم نوع را به عنوان تابع بکار ببرید.

توابع توکار متعددی در پایتون وجود دارد که عملیات تبدیل از نوعی به نوع دیگر را انجام می دهد. این توابع شی جدیدی باز می گرداند که نشانگر مقدار تبدیل شده می باشد.

تابع	شرح
int(x [,base])	X را به عدد صحیح integer تبدیل می کند.
long(x [,base])	X را به یک عدد صحیح بزرگ long integer تبدیل می کند.
float(x)	X را به یک عدد ممیز شناور تبدیل می کند.
complex(real [,imag])	یک عدد مختلط ایجاد می کند.
str(x)	شی x را به صورت یک رشته نمایش می دهد.
repr(x)	شی x را به یک expression string تبدیل می نماید.
eval(str)	یک رشته را ارزیابی کرده و یک شی به عنوان خروجی برمی گرداند.
tuple(s)	به نوع tuple تبدیل می کند.
list(s)	به نوع داده ای list تبدیل می کند.
set(s)	به یک set (مجموعه) تبدیل می کند.
dict(d)	یک dictionary ایجاد می کند. d بایستی یک مجموعه ی پست سرهم یا دنباله ای از tuple ها (به صورت جفت کلید، مقدار) باشد.
frozenset(s)	S را به یک مجموعه ی ثابت یا ایستا (frozen set) تبدیل می کند.
chr(x)	یک integer را به کاراکتر تبدیل می کند.
unichr(x)	یک integer را به کاراکتر یونیکد تبدیل می کند.
ord(x)	یک تک کاراکتر را به معادل یا مقدار عدد صحیح (integer) آن تبدیل می کند.
hex(x)	یک عدد صحیح را به رشته ی شانزده شانزدهی تبدیل می کند.
oct(x)	یک عدد صحیح را به رشته ی هشت هشتی تبدیل می کند.

عملگرهای اصلی پایتون (Python operator)

عملگرها سازه هایی هستند که توسط آن ها می توان مقدار عملوندها را دستکاری کرد.

به این عبارت دقت کنید: 9 = 5 + 4. در این عبارت، اعداد 4 و 5 عملوند (operand) خوانده شده و علامت جمع، عملگر (operator) نامیده می شود.

انواع عملگرها

زبان پایتون از عملگرهای زیر پشتیبانی می کند.

1. عملگرهای محاسباتی (arithmetic operators)
2. عملگرهای مقایسه ای (comparison operators)
3. عملگرهای انتساب (assignment operator)
4. عملگرهای منطقی (logical operator)
5. عملگرهای بیتی (bitwise operators)
6. Membership operators
7. Identity operators

در زیر به شرح تک تک این عملگرها خواهیم پرداخت.

عملگرهای محاسباتی

فرض کنید متغیر به نام a مقدار 10 و متغیر به نام b مقدار 20 را نگه می دارد:

عملگر	شرح	مثال
جمع +	مقدار a را با مقدار b جمع می بندد.	a + b = 30

-تفریق	عملوند سمت راست را از عملوند سمت چپ کسر می کند.	a – b = -10
***ضرب**	مقادیر متغیرهای a و b را درهم ضرب می کند.	a * b = 200
/تقسیم	عملوند سمت چپ را بر عملوند سمت راست تقسیم می کند.	b / a = 2
%باقی مانده تقسیم	عملوند سمت چپ را بر عملوند سمت راست تقسیم کرده و باقی مانده را بازمی گرداند.	b % a = 0
****توان**	عملوندی را به توان عملوند دیگری می برد.	a**bبه توان10 = 20
// تقسیم به کف	دو عدد را بر هم تقسیم کرده و نتیجه ی آن را به پایین گرد می کند.	9//2 = 4و9.0//2.0 = 4.0

عملگرهای مقایسه ای پایتون

این عملگرها مقادیر در دو طرف عملگر را با هم مقایسه کرده و رابطه ی بین آن ها را ارزیابی می کند. این عملگرها تحت عنوان relational operators نیز شناخته می شوند.

متغیر a مقدار 10 و متغیر b مقدار 20 را در خود ذخیره دارد:

عملگر	شرح	مثال

==	در صورت برابر بودن مقدار دو عملوند، شرط صحیح می باشد.	صحیح نمی باشد.(a == b)
!=	در صورت برابر نبودن مقدار عملوندها، شرط صحیح می شود.	
<>	اگر مقادیر دو عملوند برابر نباشد، در آن صورت شرط صحیح می شود.	شرط (a <> b)صحیح می باشد. درست شبیه عملگر =! عمل می کند.
>	چنانچه مقدار عملوند سمت چپ بزرگتر از مقدار عملوند سمت راست باشد، شرط صحیح می باشد.	(a > b)صحیح نمی باشد.
<	اگر مقدار عملوند سمت چپ کمتر از مقدار عملوند سمت راست باشد، شرط صحیح می باشد.	صحیح می باشد. (a < b)
>=	اگر مقدار عملوند سمت چپ بزرگتر از یا برابر مقدار عملوند سمت راست باشد، در آن صورت شرط صحیح می شود.	صحیح نمی باشد. (a >= b)
<=	اگر مقدار عملوند سمت چپ کوچکتر از یا مساوی مقدار عملوند سمت چپ باشد، شرط صحیح می شود.	صحیح می باشد. (a <= b)

عملگرهای انتساب

عملگر	شرح	مثال
=	یک مقداری را به متغیر تخصیص می دهد.	جمع دو مقدار a و b را داخل متغیر c می ریزد.
+= Add AND	عملوند سمت راست را به عملوند سمت چپ اضافه کرده و نتیجه را در داخل عملوند سمت چپ می ریزد. c و a را جمع زده و نتیجه را داخل c می ریزد.	c += a درواقع همان c = c + a می باشد.
-= Subtract AND	عملوند a را از c کسر می کند و نتیجه را به c تخصیص می دهد.	c -= a درحقیقت همان c = c – a می باشد.

*= Multiply AND	عملوند سمت راست را در عملوند سمت چپ ضرب کرده و نتیجه را در عملوند سمت چپ می ریزد.	a =* c برابر است با c = c * a.
/= Divide AND	متغیرهای c و a را بر هم تقسیم کرده و نتیجه را در عملوند سمت چپ می ریزد.	a =/ c در واقع همان c = c / a می باشد.
%= Modulus AND	باقی مانده تقسیم مقدار دو متغیر را محسابه کرده و آن را به عملگر سمت چپ تخصیص می دهد.	a =% c درواقع همان c = c % a می باشد.
**= Exponent AND	عملیات توان بر روی دو متغیر انجام داده و نتیجه را در متغیر سمت چپ عملگر می ریزد.	a =** c درواقع همان c = c ** a می باشد.
//=تقسیم به کف	c و a را بر هم تقسیم کرده، نتیجه ی آن را به پایین گرد می کند، سپس آن را داخل c می ریزد.	a =// c در واقع همان c = c // a می باشد.

عملگرهای بیتی در پایتون

عملگرهای bitwise با بیت ها سروکار داشته و مقادیر بیت ها را تغییر می دهد. اگر a = 60 باشد و b = 13، هر یک در فرمت دودیی بدین صورت خواهند بود –

a = 0011 1100
b = 0000 1101
---
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011

زبان پایتون از عملگرهای بیتی زیر پشتیبانی می کند:

عملگر	شرح	مثال
& Binary AND	در صورتی که در هر دو 1 باشد، 1 را در نتیجه جای گذاری می کند، در غیر این صورت صفر را در نتیجه کپی می کند.	a & b)(معادل 0000 1100
| Binary OR	اگر در یکی از دو عملوند، 1 وجود داشته باشد، 1 در نتیجه کپی می شود.	(a | b) = 61 معادل 0011 1101
^ Binary XOR	اگر هر دو عملوند یکی باشند صفر و در غیر این صورت 1 را در نتیجه کپی می کند.	(a ^ b) = 49 معادل 0011 0001
~ Binary Ones Complement	یک عملگر یگانی نقیض است که جای بیت ها را با هم عوض می کند. هرجا 1 است 0 و هر 1 است صفر می گذارد.	چون که در فرمت دودویی عدد علامت دار نداریم، مکمل 2 عدد 0011 1100 را نمایش دادیم. 61- = (a~) معادل 0011 1100 است.
<< Binary Left Shift	مقدار عملوند سمت چپ به تعداد بیت های مشخص شده توسط عملوند سمت راست به چپ رانده می شوند.	a << = 240 معادل 0000 1111 می باشد.
>> Binary Right Shift	مقدار عملوندهای سمت چپ به تعداد بیت های مشخص شده توسط عملوند سمت راست، به راست shift می شوند.	a >> = 15 معادل 0000 1111 می باشد.

عملگرهای منطقی پایتون

زبان پایتون از عملگرهای منطقی زیر پشتیبانی می کند. فرض بگیرید متغیر a دارای مقدار 10 و متغیر b دارای مقدار 20 می باشد:

عملگر	شرح	مثال
and Logical AND	اگر هر دو علوند صحیح باشند، شرط برقرار و صحیح می باشد.	(a and b) صحیح می باشد.
or Logical OR	چنانچه یکی از دو عملوند صفر نباشد، شرط برقرار می شود (مقدار true برگردانده می شود).	(a or b)صحیح می باشد.
not Logical NOT	به منظور معکوس کردن وضعیت منطقی عملوند بکار می رود.	Not(a and b) می شود.

Memberships operator در پایتون

عملگرهای membership بررسی می کنند آیا متغیر مورد نظر در یک مجموعه (sequence) همچون رشته، list یا tuple وجود دارد یا خیر. در کل دو عملگر بررسی عضویت وجود دارد که در زیر شرح داده شده:

عملگر	شرح	مثال
in	در صورت یافتن متغیر مورد نظر در مجموعه ی مشخص شده، true برمی گرداند و در غیر این صورت false.	x in y، اگر x یک عضو از مجموعه ی y باشد، 1 برمی گرداند.
not in	در صورت یافت نشدن متغیر مورد نظر در مجموعه ی مشخص شده، true برمی گرداند و در غیر این صورت false می دهد.	x not in y، اگر x عضوی از مجموعه ی y نباشد، 1 برمی گرداند.

Identity operator ها در پایتون

عملگرهای Identity، مکان های قرار گیری دو شی را با هم مقایسه می کند (بررسی می کند آیا دو شی باهم برابر هستند یا خیر.)

عملگر	شرح	نمونه
is	اگر متغیرهای هر دو طرف عملگر به شی یکسان اشاره داشته باشند، true برمی گرداند و در غیر این صورت false.	x is y، اگر id(x) برابر با id(y) باشد، 1 برمی گرداند.
is not	چنانچه متغیر در دو طرف عملگر به شی یکسان اشاره داشته باشد، false برمی گرداند و در غیر این صورت true بازمی گرداند.	x is not y ، اگر id(x) با id(y) برابر نباشد، 1 برمی گرداند.

اولویت عملگرها در پایتون

جدول زیر تمامی عملگرها را به ترتیب اولویت فهرست کرده:

عملگر	شرح
**	توان
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	ضرب، تقسیم، باقی مانده ی تقسیم و تقسیم به کف
+ -	جمع و تفریق

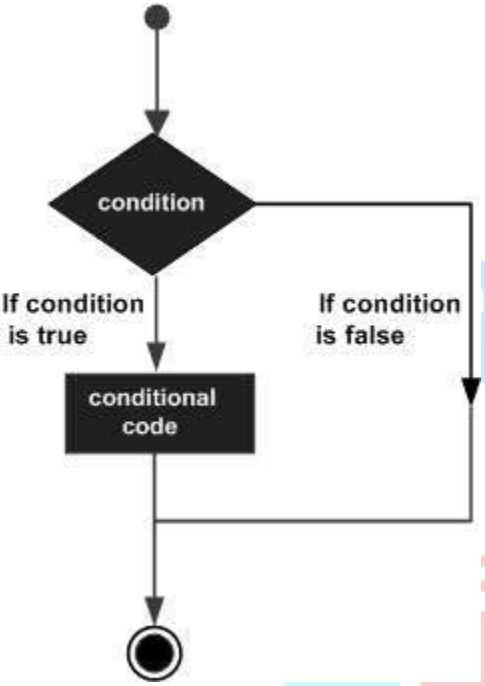
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	عملگرهای مقایسه ای
<> == !=	عملگرهای برابری
= %= /= //= -= += *= **=	عملگرهای تخصیص یا انتساب
is is not	Identity operators عملگرهای بررسی برابری اشیا
in not in	Membership operators عملگرهای بررسی عوضیت متغیر در مجموعه
not or and	عملگرهای منطقی

ساختارهای کنترلی در پایتون (python Decision making)

Decision making در واقع بررسی شرط ها حین اجرای برنامه و انجام عملیات صحیح بر اساس آن شرط است.

ساختارهای تصمیم گیری چندین عبارت را ارزیابی می کنند و به عنوان خروجی مقدار TRUE یا FALSE را برمی گردانند. برنامه نویس می بایست تصمیم بگیرد کدام دستور را در صورت صحیح یا غلط بودن شرط اجرا کند.

در زیر نموداری را شاهد هستیم که ساختار تصمیم گیری را در برنامه نویسی به نمایش می گذارد:



زبان پایتون هر مقداری که صفر و NULL نباشد را به عنوان TRUE درنظر می گیرد و اگر مقداری صفر یا NULL بود، آن را FALSE تلقی می کند.

دستورهای تصمیم گیری در زبان پایتون به شرح زیر می باشند:

دستور	شرح
if statements	از این ساختار در مواقعی که می خواهیم در صورت بر قرار بودن شرط یا شرط هايي يكسری دستورات خاص اجرا شوند، استفاده می شود. این دستور از یک عبارت بولی تشکیل می شود که به دنبال آن یک یا چند دستور دیگر می آید.
if...else statements	از این ساختار در مواقعی استفاده می کنیم که می خواهیم در صورت بر قرار بودن شرط یا شرط هايي ، يكسری دستورات و در صورت عدم بر قراری آن شروط ، گروهي ديگر از دستورات اجرا شوند . به دنبال دستور شرطی if یک دستور اختیاری else می آید که در صورت FALSE بودن عبارت بولی اجرا می شود.
nested if statements	دستورات تودرتوی if – می توان یک if یا else if را درون یک دستور if یا else if دیگر گنجاند.

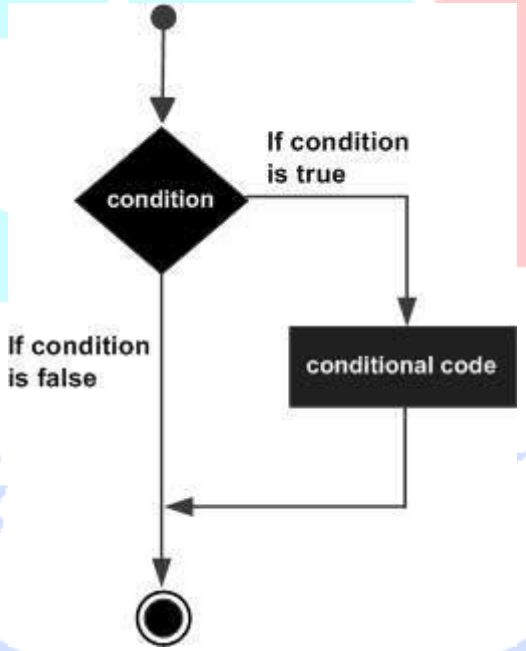
در صورتيكه نياز باشد تا چندين حالت منطقي مورد بررسي قرار گيرد و دستورات مربوط به يكي از آنها اجرا شود، از فرم تصميم‌گيري چندگانه استفاده مي‌نماييم.	

دستور if در پایتون

نحوه ی نگارش:

```
if expression:
statement(s)
```

در صورت TRUE بودن عبارت بولی، قطعه کد داخل ساختمان if اجرا می شود. اگر عبارت بولی False برگرداند، در آن صورت دستوراتی که بلافاصله پس از if درج شده، اجرا می شود.



مثال:

```
#!/usr/bin/python
var1 = 100
if var1:
print "1 - Got a true expression value"
print var1
var2 = 0
if var2:
```

```
print "2 - Got a true expression value"
print var2
print "Good bye!"
```

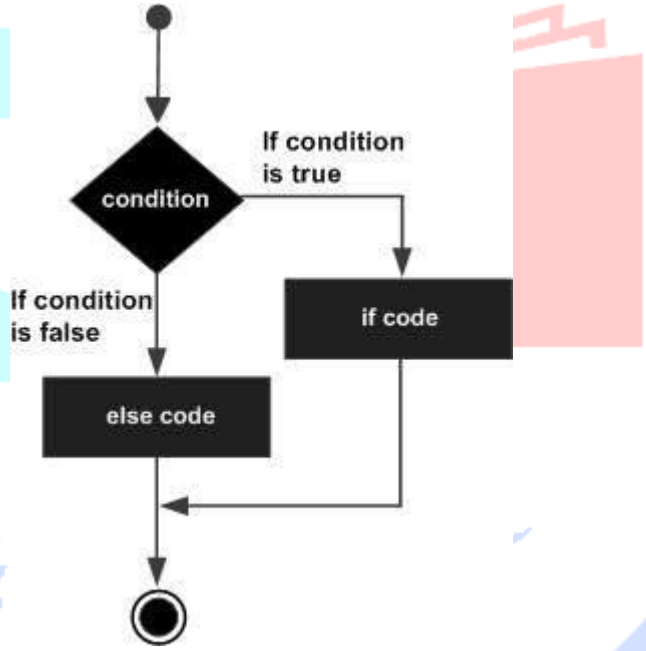
پس از اجرای دستورات فوق، نتیجه ی زیر حاصل می گردد:

```
1 - Got a true expression value
100
Good bye!
```

IF...ELIF...ELSE در پایتون

نحوه ی نگارش دستور if...else بدین صورت می باشد:

```
if expression:
statement(s)
else:
statement(s)
```



مثال:

```
#!/usr/bin/python
var1 = 100
if var1:
print "1 - Got a true expression value"
print var1
else:
print "1 - Got a false expression value"
print var1
var2 = 0
if var2:
print "2 - Got a true expression value"
```

```
print var2
else:
print "2 - Got a false expression value"
print var2

print "Good bye!"
```

پس از اجرا نتیجه ی زیر حاصل می گردد:

```
1 - Got a true expression value
100
2 - Got a false expression value
0
Good bye!
```

دستور elif

دستور elif به شما این امکان را می دهد که چندین عبارت را بررسی کنید و در صورت صحیح بودن (برقرار بودن) یکی از شرط ها (برگردانده شدن مقدار TRUE)، یک قطعه کد معین را اجرا کند.

مانند else، دستور elif کاملا اختیاری می باشد. اما بر خلاف else، می توان پس از دستور if چندین دستور elif داشت.

نحوه ی نگارش:

```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
```

در زبان پایتون، بر خلاف دیگر زبان های برنامه نویسی، ساختار کنترلی switch و دستورات case وجود ندارد. در عوض این زبان به منظور شبیه سازی عملکرد switch از همان دستورات if..elif... کمک می گیرد:

مثال:

```
#!/usr/bin/python
var = 100
```

```
if var == 200:
print "1 - Got a true expression value"
print var
elif var == 150:
print "2 - Got a true expression value"
print var
elif var == 100:
print "3 - Got a true expression value"
print var
else:
print "4 - Got a false expression value"
print var
print "Good bye!"
```

خروجی کد بالا:

```
3 - Got a true expression value
100
Good bye!
```

If های تودرتو

در صورتی که لازم باشد چندین حالت منطقی مورد بررسی قرار گرفته و دستورات مربوط به یکی از آنها اجرا شود، از ساختار تصمیم‌گیری چندگانه بهره می گیریم. این نوع استفاده از دستور if در اصطلاح به if تودرتو (Nested If) معروف است زیرا در آن از چندین دستور if مرتبط به یکدیگر استفاده شده است.

در ساختار تودرتو، می توان یک if...elif...else در دل if...elif...else داشت.

نحوه ی نگارشی:

```
if expression1:
   statement(s)
   if expression2:
      statement(s)
elif expression3:
   statement(s)
   else
      statement(s)
elif expression4:
   statement(s)
   else:
      statement(s)
```

مثال:

```
#!/usr/bin/python

var = 100
if var < 200:
print "Expression value is less than 200"
if var == 150:
print "Which is 150"
elif var == 100:
print "Which is 100"
elif var == 50:
print "Which is 50"
elif var < 50:
print "Expression value is less than 50"
else:
print "Could not find true expression"

print "Good bye!"
```

نتیجه:

```
Expression value is less than 200
Which is 100
Good bye!
```

Statement Suite

چنانچه ساختمان if تنها از یک دستور تشکیل شود، در آن صورت می توان دستور if را در همان خط دستور سرآیند (header statement) قرار داد:

مثال:

```
#!/usr/bin/python
var = 100
if ( var  == 100 ) : print "Value of expression is 100"
print "Good bye!"
```

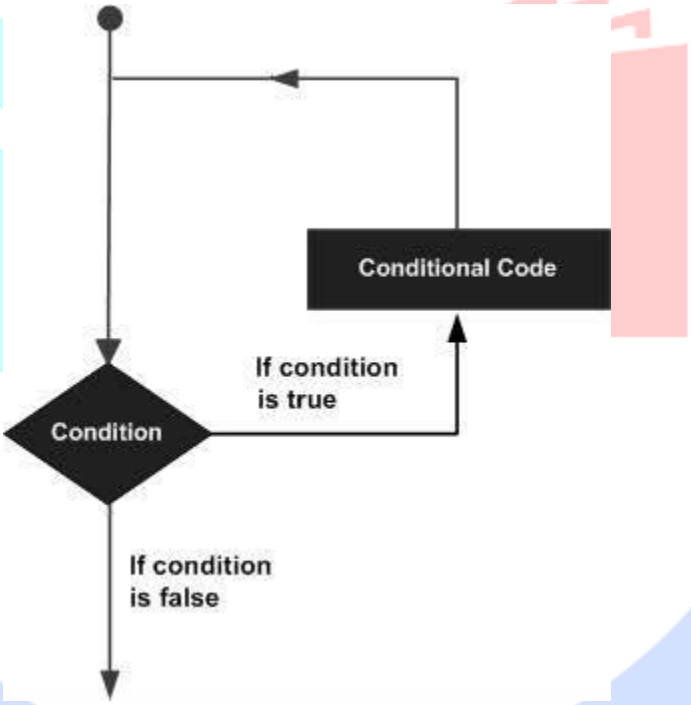
نتیجه:

```
Value of expression is 100
Good bye!
```

ساختارهای تکرار (python loops)

در کل، دستورات به ترتیب اجرا می شوند: در یک تابع ابتدا دستور اول، سپس دستور دوم و به همین ترتیب ادامه می یابد . اما گاهی لازم است مجموعه دستورات داخل یک قطعه کد بارها تکرار شود.

از ساختار کنترلی حلقه ها در برنامه نویسی، برای اجرای مجموعه ای از دستورها به تعداد دفعات لازم یا تا زمانی که یک شرط معین درست و برقرار باشد، استفاده می شود .
در حلقه ، هنگامی که مجموعه دستورات حلقه به طور کامل اجرا می شوند، برنامه بار دیگر به ابتدای مجموعه دستورات حلقه رفته و در صورت برقرار بودن شرط حلقه، یکبار دیگر دستورات آن به طور کامل اجرا می کند.



زبان برنامه نویسی پایتون دو حلقه ی for و while پشتیبانی می کند که هریک در زیر تشریح شده:

حلقه	شرح
while loop	در این نوع حلقه ، مجموعه دستورالعمل ها به تعداد معلوم و مورد نیاز ، تا زمانی که شرط مشخص شده صحیح می باشد، تکرار خواهد شد . این حلقه قبل از اجرای دستورات بدنه ی حلقه، شرط را بررسی می کند.
for loop	در این نوع حلقه ، مجموعه دستورالعمل ها به تعداد معلوم و مورد نیاز تکرار خواهد شد .
nested loops **(حلقه های تودرتو)**	می توان درون یک حلقه ی while، while.. do و for یک یا چند حلقه ی دیگر گنجاند.

دستورات کنترلی حلقه ها

دستورات کنترلی اجرا را از روال عادی خود خارج کرده و تغییراتی در آن بوجود می آورد. پس از اینکه اجرا آن حوزه یا scope را ترک می کند، تمامی اشیایی که به صورت خودکار در آن حوزه بوجود آمده بودند، از بین خواهد رفت.

دستور کنترلی	شرح
break statement	از دستور break برای خروج کامل از ادامه اجرای دستورات یک حلقه در صورت بر قرار بودن شرط تعیین شده برای آن استفاده می شود .
continue statement	از دستور continue ، برای خارج شدن از ادامه اجرای یکبار دستورات حلقه و پرش به گام بعدی حلقه استفاده می شود .
pass statement	دستور pass در زبان پایتون زمانی مورد استفاده قرار می گیرد که یک دستور از لحاظ ساختار برنامه نویسی مورد نیاز باشد ، اما شما هیچ فرمان یا کدی را برای اجرا لازم نداشته باشید .

دستور break

دستور break در زبان پایتون ، حلقه جاری را به پایان رسانده و درست همانند break سنتی موجود در C، اجرا را از دستور بعدی از سر می گیرد .

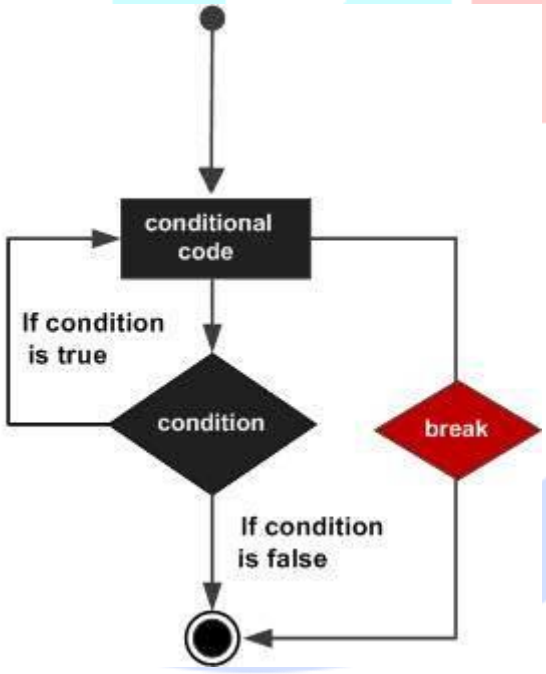
بیشترین کاربرد break زمانی است که اتفاقی در خارج از حلقه رخ داده است که خروج سریع از حلقه را می طلبد .

دستور break را می توان در هر دو حلقه while و for استفاده نمود .

در صورت استفاده از حلقه های تودرتو، دستور break اجرای درونی ترین حلقه را متوقف کرده و به اجرای دستور بعدی پس از قطعه کد می پردازد.

نحوه ی نگارش:

Break



مثال:

```
for letter in 'Python':     # First Example
```

```
        if letter == 'h':
            break
    print 'Current Letter :', letter

var = 10                    # Second Example
while var > 0:
    print 'Current variable value :', var
    var = var -1
    if var == 5:
        break

print "Good bye!"
```

نتیجه ی زیر را بدست می دهد:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

دستور continue در زبان برنامه نویسی پایتون

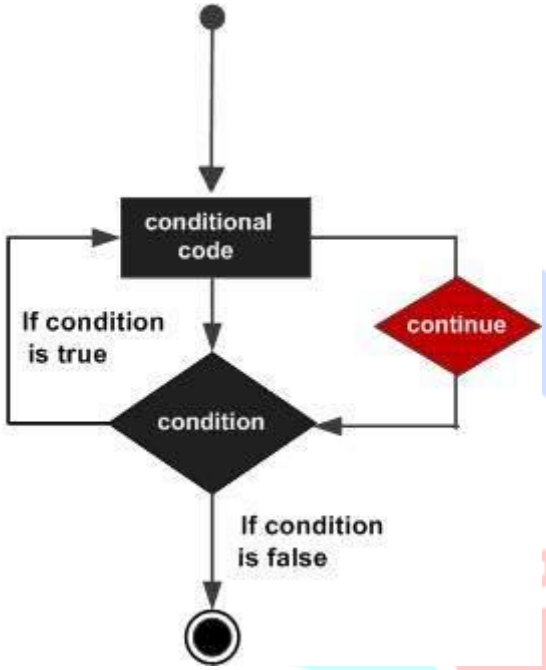
دستور continue از روی تمامی دستورهای باقی مانده در تکرار جاری حلقه پریده و کنترل را به بالای حلقه بازمی گرداند (انتقال می دهد) .

دستور continue را می توان در هر دو حلقه while و for بکار برد .

نحوه ی نگارش:

```
continue
```

نتیجه:




```
for letter in 'Python':     # First Example
   if letter == 'h':
      continue
   print 'Current Letter :', letter

var = 10                    # Second Example
while var > 0:
   print 'Current variable value :', var
   var = var -1
   if var == 5:
      continue

print "Good bye!"
```

نتیجه:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
```

```
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

دستور pass

دستور pass در زبان پایتون زمانی بکار می رود که یک دستور از لحاظ ساختار برنامه نویسی مورد نیاز است، اما شما هیچ فرمان یا کدی را برای اجرا لازم نداشته باشید .

دستور pass ، یک عملیات تهی می باشد؛ بدین معنا که هنگام اجرای آن هیچ اتفاقی نمی افتد.

دستور pass همچنین در جاهایی که کد شما بعدا نوشته خواهد شد ، اما هنوز نوشته نشده است، بسیار مفید می باشد : (به عنوان مثال ، در stub ها).

نحوه ی نگارش:

```
pass
```

```
#!/usr/bin/python

for letter in 'Python':
   if letter == 'h':
      pass
   print 'This is pass block'
   print 'Current Letter :', letter

print "Good bye!"
```

نتیجه:

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

اعداد در پایتون

Number type مقادیر عددی را در خود ذخیره می کند. این نوع داده ای immutable هست، بدین معنا با تغییر مقدار نوع عددی، آن خانه ی حافظه پاک شده و خانه ی جدید برای آن شی در نظر گرفته شود و مقدار آن شی در خانه ی تازه ایجاد شده جای گذاری می شود.

شی number زمانی که مقداری را به آن ها تخصیص می دهید، ایجاد می گردند.

مثال:

```
var1 = 1
var2 = 10
```

با استفاده از دستور del می توان ارجاع (reference) به یک شی را پاک کرد. نحوه ی نگارش این دستور بدین ترتیب است:

```
del var1[,var2[,var3[....,varN]]]
```

با استفاده از این دستور می توان یک یا چندین شی را حذف نمود. مثال:

```
del var
del var_a, var_b
```

پایتون از چهار نوع عددی پشتیبانی می کند:

1. Int (اعداد صحیح علامت دار): اعداد صحیح منفی یا مثبت که بخش اعشاری، نقطه و ممیز اعشار ندارد.
2. long (اعداد صحیح بزرگ): اینتیجرهای طولانی که میتوانند به فرمت اکتال یا هگزادسیمال نیز باشند. این اعداد بسیار بزرگ هستند (به عبارتی بی نهایت هستند) که به صورت همان اینتیجر نوشته شده و به دنبال آن یک L بزرگ یا کوچک نمایش داده می شود.

3. float (اعداد حقیقی ممیز شناور): اعدادی که به صورت اعشاری نمایش داده می شوند که بخش عدد صحیح از بخش اعشاری یا کسر توسط نقطه ی ممیز جدا می شود. اعداد Float با نماد علمی نیز نمایش داده می شوند، برای مثال e یا E که نشانگر توان 10 می باشد (2.5e2 = 2.5 x 10^2 = 250).
4. Complex (اعداد مختلط): هر عدد مختلط از دو بخش تشکیل شده است: بخش Real یا حقیقی و بخش Imaginary یا انتزاعی. برای مثال عبارت a + bJ، دو عدد a و b ممیز شناور می باشند و J بیانگر ریشه ی 1- می باشد که یک عدد موهومی یا انتزاعی است. این نوع اعداد در پایتون کاربرد چندانی ندارند.

مثال ها

مثالی از اعداد را در جدول زیر مشاهده می کنید:

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEL	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

1. پایتون به شما اجازه می دهد برای مشخص کردن نوع عددی Long از l کوچک استفاده کنید، اما برای اینکه آن l با عدد 1 اشتباه گرفته نشود، توصیه می کنیم از L بزرگ استفاده کنید. بنابراین پایتون اعداد صحیح بسیار بزرگ را با L بزرگ نمایش می دهد.

2. یک عدد مختلط متشکل است از دو عدد ممیز شناور حقیقی و یک بخش که یکه ی موهومی نام دارد. برای مثال در x + yj، x و y اعداد حقیقی هستند و j نشانگر واحد یا یکه ی موهومی (imaginary) می باشد.

تبدیل نوع های عددی

پایتون اعداد موجود در یک عبارت را که متشکل از انواع مختلف است به یک نوع متداول برای ارزیابی تبدیل می کند. اما گاهی لازم است یک عدد را به صورت صریح به یک نوع دیگر تبدیل کنید تا شرایط مورد نیاز یک پارامتر (function parameter) یا عملگر برآورده شود.

1. به منظور تبدیل x به یک عدد صحیح ساده بکار می رود.
2. long(x) جهت تبدیل x به یک عدد صحیح بسیار بزرگ (long integer) بکار می رود.
3. float(x)جهت تبدیل x به یک عدد ممیز شناور بکار می رود.
4. برای تبدیل x به یک عدد مختلط که دارای بخش حقیقی و بخش موهومی 0 باشد، complex(x) را تایپ کنید.
5. complex(x, y) را جهت تبدیل x و y به یک عدد مختلط دارای بخش حقیقی x و بخش موهومی y بکار ببرید. X و y عبارت های عددی هستند.

توابع ریاضی

پایتون با استفاده از توابع زیر محاسبات ریاضی انجام می دهد:

توابع تصادفی در پایتون

تابع	خروجی
abs(x)	قدر مطلق x را برمی گرداند: فاصله ی مثبت بین x و 0.
ceil(x)	سقف یک عدد را برمی گرداند: کوچکترین یا نزدیک ترین عدد صحیح بزرگتر از x را برمی گرداند. به عبارتی دیگر این تابع کوچکترین عدد صحیح بزرگتر یا مساوی با يك عدد را که بعنوان آرگومان آن می‌باشد محاسبه می‌کند.
cmp(x, y)	چنانچه x از y کوچکتر باشد، 1- و چنانچه بزرگتر مساوی باشد 0 برمی گرداند.

exp(x)	این تابع برای محاسبه توانی از e (پایه لگاریتم طبیعی) مورد استفاده قرار می گیرد.
fabs(x)	قدر مطلق یک عدد را برمی گرداند.
floor(x)	این تابع بزرگترین مقدار صحیح کوچکتر یا مساوی یک عدد را برمی گرداند؛ کف عدد را برمی گرداند.
log(x)	**این تابع لگاریتم طبیعی یک عدد مثبت را محاسبه می کند.**
log10(x)	**این تابع لگاریتم مبنای 10 اعداد مثبت را محاسبه می‌کند.**
max(x1, x2,...)	این تابع تعدادی عدد رو به عنوان آرگومان دریافت می کند و ماکزیمم آن رو براتون محاسبه می کند؛ نزدیک ترین مقدار به مثبت بی نهایت.
min(x1, x2,...)	این تابع نیز تعدادی عدد به عنوان ورودی گرفته و کوچکترین عدد از میان آن ها را گزینش کرده و نمایش می دهد؛ نزدیکترین مقدار به منفی بی نهایت.
modf(x)	این تابع یک عدد اعشاری را گرفته و بخش اعشاری و خود عدد را از هم جدا می کند. در واقع بخش عدد صحیح و اعشاری عدد x را در یک تاپل دو بخشه نمایش می دهد. هر دو بخش دارای علامتی یکسان با x هستند. بخش عدد صحیح به صورت ممیز شناور (float) برگردانده می شود.
pow(x, y)	این تابع دو عدد را از کاربر گرفته و عدد اول را به توان عدد دوم می برد.
round(x [,n])	دو عدد از کاربر به عنوان آرگومان گرفته و عدد اول را به مقدار عدد دوم گرد می کند. برای مثال round(0.5) می شود 1.0 و نیز round(-0.5) می شود 1.0.
sqrt(x)	جذر یا ریشه ی دوم x را برای x > 0 را برمی گرداند. این تابع جذر یك عدد مثبت را حساب می‌کند.

DFDGD

تابع	شرح
choice(seq)	این تابع یک عدد را به صورت تصادفی از یک list، tuple یا string بر می گرداند.

randrange ([start,] stop [,step])	سه مقدار از کاربر دریافت کرده که مقدار اول و دوم عدد و مقدار سوم گام شمارش هست. به این شکل که این تابع بین عدد اول و دوم با گام شمارش ای که برای اون تعیین کردیم یک عدد تصادفی را انتخاب می کند.
random()	این تابع به خودی خود یک عدد تصادفی بین 0 و 1 را برگزیده و بازیابی می نماید.
seed([x])	این تابع یک مقدار اولیه برای random تعیین می کند و در ادامه برنامه بر اساس همان مقدار اعداد تصادفی را برمی گرداند. خروجی ندارد. این تابع را پیش از فراخوانی هر module function دیگری صدا بزنید.
shuffle(lst)	این تابع تعدادی عدد از لیست خوانده و ترتیب آن ها را به هم می ریزد. خروجی ندارد.
uniform(x, y)	این متد عدد حقیقی بین دو عدد را به شما نشان می دهد.

توابع مثلثاتی

پایتون با استفاده از توابع زیر محاسبات مثلثاتی انجام می دهد:

تابع	شرح
acos(x)	آرک کوسینوس عدد را بر حسب رادیان برمی گرداند.
asin(x)	آرک سینوس عدد x را بر حسب رادیان برمی گرداند.
atan(x)	آرک تانژانت عدد x را بر حسب رادیان برمی گرداند.
atan2(y, x)	این تابع آرک تانژانت y بر روی x را محاسبه می کند.
cos(x)	کوسینوس x را بر حسب رادیان محاسبه کرده و برمی گرداند.
hypot(x, y)	این تابع به نورم تقلیدسی جذر (x*x + y*y) می باشد و این مقدار را بر می گرداند.
sin(x)	این تابع مقدار سینوس فایل را محاسبه و نمایش می دهد.
tan(x)	این تابع مقدار تانژانت عدد را بر حسب رادیان برمی گرداند.
degrees(x)	این تابع مقدار زاویه بر حسب رادیان را گرفته و به درجه برمی گرداند.

مقدار عدد x بر حسب درجه را گرفته و به رادیان برمی گرداند.	**radians(x)**

ثوابت ریاضی

شرح	ثوابت
عدد ثابت pi در ریاضی.	**pi**
عدد ثابت e در ریاضیات.	**e**

رشته ها در پایتون (نوع داده ای string)

string از پرکاربرد ترین انواع داده ای در پایتون می باشد. به منظور ایجاد آن، کافی است تعدادی کاراکتر را در علامت نقل و قول محصور کرد. پایتون تک کوتیشن را با دابل کوتیشن یکسان دانسته و با آن ها به طور مشابه برخورد می کند. پروسه ی ایجاد رشته به آسانی تخصیص مقدار به یک متغیر می باشد. مثال:

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

دسترسی به مقادیر در رشته ها

در پایتون چیزی به نام نوع داده ای character وجود ندارد؛ زبان پایتون با این نوع داده ای به صورت رشته هایی با طول 1 برخورد می کند (رشته هایی که خاصیت length آن برابر 1 می باشد). از این رو آن ها را می توان یک substring نیز قلمداد کرد.

جهت استخراج substring ها، بایستی از [] همراه با اندیس شروع و اندیس پایان بهره گرفت.به عبارتی دیگر برای برش بخشی از یک رشته و و استخراج آن، از [] همراه با دو اندیس شروع و پایان بایستی استفاده کرد. مثال:

```
#!/usr/bin/python
var1 = 'Hello World!'
var2 = "Python Programming"
print "var1[0]: ", var1[0]
```

```
print "var2[1:5]: ", var2[1:5]
```

نتیجه:

```
var1[0]:  H
var2[1:5]:  ytho
```

بروز رسانی رشته ها

می توان با الحاق (مجدد) یک متغیر به رشته ی دیگر، رشته ی جاری را بروز رسانی کرد. مثال:

```
#!/usr/bin/python
var1 = 'Hello World!'
print "Updated String :- ", var1[:6] + 'Python'
```

نتیجه:

```
Updated String :-  Hello Python
```

کاراکترهای Escape

جدول زیر فهرستی از کاراکترهای گریز یا چاپ نشدنی را ارائه می دهد که با علامت \ شروع شده و نمایش داده می شوند.

کاراکتر گریز در هر دو نوع رشته ای که داخل تک کوتیشن و دابل کوتیشن محصور هستند قابل تفسیر می باشد.

Backslash notation	کاراکتر شانزده شانزدهی (hexadecimal)	شرح
\a	0x07	alertیاBell ایجاد صدای هشدار.
\b	0x08	Backspace بازگشت به عقب.

\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed ایجاد صفحه ی جدید.
\M-\C-x		Meta-Control-x
\n	0x0a	Newline خط جدید
\nnn		نشان گذاری هشت هشتی که در آن n در رینج 0.7 قرار دارد.
\r	0x0d	Carriage return بازگشت به سر خط.
\s	0x20	Space خط فاصله.
\t	0x09	Tab چندین خط فاصله.
\v	0x0b	Vertical tab ایجاد چندین خط فاصله به صورت عمودی.

Character x		\x
نشان گذاری شانزده شانزدهی که در آن n در برد 0.9، a.f یا A.F قرار دارد.		\xnn

عملگرهای رشته

با فرض اینکه متغیر رشته ای a مقدار 'Hello' و متغیر b مقدار 'Python' را نگه می دارد:

مثال	شرح	عملگر
a + b رشته ی HelloPython را تولید می کند.	اتصال – یک رشته را به رشته ی دیگر متصل می کند.	+
a*2رشته ی HelloHello را تولید می کند.	تکرار – با پیوند زدن چندین کپی از یک رشته، رشته ی جدیدی ایجاد می نماید.	*
a[1] کاراکتر e را ارائه می دهد.	برش – کاراکتر مورد نظر را از اندیس مشخص شده انتخاب کرده و استخراج می کند.	[]
a[1:4] کاراکترهای ell را استخراج می کند.	برش در رینج مشخص شده (range slice) –دو اندیس می پذیرد که یکی شروع و مشخص کننده ی اولین کاراکتر و دیگری اندیس پایان و نشانگر آخرین کاراکتر استخراج شده می باشد.	[:]
H in a بخاطر اینکه کاراکتر در رشته ی مورد نظر موجود می باشد، 1 برمی گرداند.	عضویت – در صورت موجود بودن کاراکتر مشخص شده در رشته ی مورد نظر، true برمی گرداند.	in
M not in a.مقدار 1 را برمی گرداند	عضویت – در صورت موجود نبودن کاراکتر مورد نظر در رشته، true برمی گرداند.	not in

r/R	Raw string (رشته ی خام) – کارکرد و معنی اصلی کاراکتر گریز را لغو می کند. به استثنای عملگر r/R که پیش از علامت نقل و قول قرار می گیرد، سینتکس رشته های خام درست مانند رشته های معمولی می باشد. "r" می تواند حرف کوچک یا حرف بزرگ (R) باشد، تنها مسئله ای که باید به آن دقت داشت، قرار دادن R درست پیش از علامت نقل و قول است.	دستور print r'\n'، \n را چاپ می کند و دستور print R'\n'، \n را.
%	فرمت دهی – یک رشته را قالب بندی می کند.	

عملگر فرمت دهی رشته

یکی از امکانات جالب پایتون، عملگر % می باشد که برای فرمت دهی رشته ها بکار می رود. این عملگر منحصر به رشته ها می باشد (ورودی آن فقط رشته است) و جایگزین خانواده ی ()printf در زبان C می باشد. مثال:

```
#!/usr/bin/python
print "My name is %s and weight is %d kg!" % ('Zara', 21)
```

نتیجه:

```
My name is Zara and weight is 21 kg!
```

در زیر فهرست علامت هایی که می توان همراه با این عملگر مورد استفاده قرار داد را مشاهده می کنید:

تبدیل	علامت فرمت دهی

کاراکتر.	%c
تبدیل به رشته با استفاده از تابع str()، قبل از فرمت دهی.	%s
عدد صحیح در مبنای ده علامت دار.	%i
عدد صحیح ده دهی علامت دار.	%d
عدد صحیح درمبنای ده بدون علامت.	%u
عدد صحیح هشت هشتی.	%o
عدد صحیح در مبنای شانزده (حروف کوچک).	%x
عدد صحیح در مبنای شانزده (حروف بزرگ).	%X
نماد توانی یا نمایی (با 'e' کوچک).	%e
نماد توانی یا نمایی (با 'E' بزرگ).	%E
عدد حقیقی ممیز شناور.	%f
مخفف یا فرم کوتاه تر %f و %e می باشد.	%g
شکل کوتاه تر %f و %E.	%G

در زیر دیگر علامت ها و قابلیت هایی پشتیبانی شده را مشاهده می کنید:

کارکرد/قابلیت	علامت
آرگومان دقت اعشار یا حداقل تعداد کل را مشخص می کند.	*
تنظیم به سمت چپ.	-
علامت را نمایش می دهد.	+

یک خط فاصله قبل از عدد مثبت ایجاد می کند.	<sp>
صفر آغازین هشت هشتی ('0') یا صفر آغازین شانزده شانزدهی '0x' و یا '0X'، بسته به اینکه 'x' یا 'X' بکار رفته، اضافه می کند.	#
بجای خط فاصله یا فضای خالی، سمت چپ عدد را با صفر پر می کند.	0
استفاده ی همزمان از دو عملگر، '%% '، به یک عملگر '%' منتج می شود.	%
نگاشت متغیر (آرگومان های dictionary)	(var)
m نشانگر حداقل تعداد عدد مجاز و n بیانگر تعداد عدد که بایستی پس از نقطه ی اعشار نمایش داده شود.	m.n.

سه علامت نقل و قول به هم چسبیده (Triple Quotes)

علامت (""" """) این امکان را فراهم می کند که رشته ها را در چندین خط پخش کنید، در این میان می توانید از کاراکترهای ویژه نظیر NEWLINEs، TABs و غیره ... استفاده کنید.

نحوه ی نگارش آن را می توانید در زیر مشاهده کنید:

```
#!/usr/bin/python
para_str = """this is a long string that is made up of
several lines and non-printable characters such as
TAB ( \t ) and they will show up that way when displayed.
NEWLINEs within the string, whether explicitly given like
this within the brackets [ \n ], or just a NEWLINE within
the variable assignment will also show up.
"""
print para_str
```

پس از اجرای کد بالا، نتیجه ی زیر حاصل می گردد. مشاهده می کنید که چگونه هر یک از کاراکترها ویژه اثر خود را در نتیجه به نمایش گذاشته (برای مثال tab/ دو فاصله ایجادکرد یا n/ رشته را در خط بعدی ادامه داده) است.

```
this is a long string that is made up of
several lines and non-printable characters such as
TAB (	) and they will show up that way when displayed.
NEWLINEs within the string, whether explicitly given like
this within the brackets [
], or just a NEWLINE within
the variable assignment will also show up.
```

کاراکتر " \ " در رشته های خام، به عنوان کاراکتر ویژه تفسیر نمی شود. کلیه ی کاراکترهایی که در یک رشته ی خام تایپ می کنید به همان گونه که نوشته اید چاپ می شوند:

```
#!/usr/bin/python
print 'C:\\nowhere'
```

نتیجه:

```
C:\nowhere
```

حال به مثال زیر توجه کنید. عبارت را بدین صورت ' عبارت مربوطه ' r بکار می بریم:

```
#!/usr/bin/python
print r'C:\\nowhere'
```

نتیجه:

```
C:\\nowhere
```

رشته های یونیکد (Unicode string)

رشته های معمولی در پایتون به صورت 8 بیتی اسکی (ASCII) ذخیره می شوند، در حالی که رشته های یونیکد به صورت 16 بیتی Unicode ذخیره می گردند. کاراکترهای یونیکد این اجازه را می دهد که از مجموعه ی کاراکتری متشکل از انواع زبان های زنده ی دنیا بهره بگیرید. مثال:

```
#!/usr/bin/python
print u'Hello, world!'
```

نتیجه:

```
Hello, world!
```

همان طور که در این مثال مشاهده می کنید، رشته های یونیکد پیشوند u را بکار می برند، مشابه رشته های خام که از پیشوند r استفاده می کنند.

متدهایی درون ساخته ای که عملیاتی را روی رشته انجام دهند (Built-in String Methods)

پایتون با استفاده از توابع زیر روی رشته ها عملیاتی را انجام می دهد:

شماره ی رشته ی مورد نظر	متد مربوطه به همراه شرح کارکرد آن
1	capitalize() حرف اول یک رشته را به حرف بزرگ تبدیل می کند.
2	center(width, fillchar) این تابع دو ورودی می پذیرد؛ اولی طول رشته و دومی کاراکتری که در هر دو طرف رشته پر می کند، را مشخص می کند. سپس رشته را در وسط یا مرکز قرار می دهد.
3	count(str, beg= 0,end=len(string)) این متد سه ورودی می پذیرد. اولی زیررشته ی ای که تعداد تکرار آن شمرده و برگردانده می شود را مشخص می کند، دومین پارامتر اندیس شروع که شمارش از آن آغاز می شود را تعیین می نماید و سومین آرگومان اندیس پایان که شمارش در آنجا خاتمه می یابد را تعریف می کند.
4	decode(encoding='UTF-8',errors='strict') رشته ی مورد نظر را با استفاده ی الگورتیم رمزگذاری یا کدک مشخص/ثبت شده، decode و رمز گشایی می کند. پارامتر encoding به صورت پیش فرض بر روی رمزگذاری default تنظیم می شود. به عبارتی دیگر این تابع رشته ی مورد نظر را بر اساس کدک مشخص شده رمزگشایی کرده و رشته ی decode

	شده را به عنوان خروجی برمی گرداند. پارامتر دومی نحوه ی مدیریت خطا را مشخص می شود که پیش فرض آن strict می باشد.
5	encode(encoding='UTF-8',errors='strict') نسخه ی کدگذاری شده ی رشته ی مورد نظر را بازمی گرداند. پارامتر اولی الگوریتم کد گذاری مشخص شده و پارامتر دوم نحوه ی مدیریت خطا را مشخص می کند. پیش فرض پارامتر دوم آن strict می باشد.
6	endswith(suffix, beg=0, end=len(string)) مشخص می کند آیا یک رشته یا زیرشته ای از آن به پارامتر suffix ختم می شود یا خیر، در صورتی که به مقدار مشخص شده ختم شد true و در غیر این صورت false برمی گرداند. پارامتر beg اندیس شروع و پارمتر end اندیس پایان را مشخص می کند.
7	expandtabs(tabsize=8) این متد تعداد فاصله ی تب (مربوط به کاراکتر ویژه ی تب /t) یک رشته را به مقدار مشخص شده در پارامتر ورودی (تعیین کننده ی تعداد تب) بسط می دهد، در صورت مشخص نکردن پارامتر ورودی، به صورت پیش فرض بر روی 8 تنظیم می شود.
8	find(str, beg=0 end=len(string)) کاراکترهای مورد نظر را در رشته می یابد (مشخص می کند آیا پارامتر ورودی اول متد در رشته وجود دارد یا خیر)، پارامتر دوم مشخص کننده ی اندیس شروع و پارامتر سوم تعیین کننده ی اندیس پایان می باشد. در صورت یافتن رشته اندیس آغازی آن و در صورت نیافتن 1- را برمی گرداند.
9	index(str, beg=0, end=len(string)) عملکردی مشابه متد ()find دارد، با این تفاوت که در صورت نیافتن رشته ی مورد نظر باعث رخداد یک خطا یا exception می شود.
10	isalnum()

	چنانچه حداقل 1 کاراکتر یا تمامی کاراکترهای رشته الفبایی عددی (حرف عدد) باشد، مقدار true و در غیر این صورت false برمی گرداند.
11	isalpha() اگر حداقل یکی یا تمامی کاراکترهای رشته از نوع الفبا باشد، true و در غیر این صورت false را برمی گرداند.
12	isdigit() در صورتی که رشته مورد نظر حاوی فقط عدد باشد، مقدار true و در غیر این صورت false برمی گرداند.
13	islower() اگر حداقل یکی یا همه ی کاراکترهای داخل رشته lowercase باشد، true و در غیر این صورت false را بازگردانی می نماید.
14	isnumeric() چنانچه رشته ی یونیکد دربردارنده ی کاراکترهای عددی باشد، مقدار true و در غیر این صورت false را بازیابی می نماید.
15	isspace() چنانچه رشته ی مورد نظر تهی باشد (در آن هیچ کاراکتری به جز خط فاصله وجود نداشته باشد)، true و در غیر این صورت false را بازیابی می کند.
16	istitle() اگر حروف اول تمامی واژگان رشته uppercase باشد، true بازمی گرداند.
17	isupper() چنانچه تمامی کاراکترهای رشته ی مورد نظر با حروف بزرگ نوشته شده باشد، true و در غیر این صورت false را برمی گرداند.
18	join(seq)

	المان های رشته ی seq را توسط تفکیک گر مشخص شده (برای مثال "-") به هم متصل کرده و آن را به عنوان خروجی برمی گرداند.
19	len(string) طول رشته (تعداد کاراکترهای) رشته را برمی گرداند.
20	ljust(width[, fillchar]) یک رشته را با کاراکتر تعریف شده در پارامتر دوم از سمت راست پر می کند تا به تعداد کاراکتر تعریف شده در پارامتر اول برسد.
21	lower() کلیه ی حروفی که در رشته ی مورد نظر uppercase هستند را به حروف کوچک تبدیل می کند.
22	lstrip() تمامی کاراکترهای مشخص شده (به صورت پیش فرض کاراکتر space) در پارامتر ورودی را از اول یک رشته حذف می کند.
23	maketrans() برای استفاده از این تابع ابتدا باید آن را تعریف کرد که در ابتدای اسکریپت این کار انجام شده است. این متد دو آرگومان می پذیرد که کار ترجمه را انجام می دهند، بدین معنی که به ازای مقادیری که در intab قرار داده می شود معادل آن ها در outtab قرار می گیرند. مثلا اگر مقدار intab برابر a بود و مقدار outtab برابر 1 بود آنگاه در رشته هرگاه کاراکتر a دیده شد برنامه مقدار آن را با 1 جایگزین می کند.
24	max(str) این متد بزرگترین کاراکتر را به ترتیب حروف الفبا بر میگرداند.
25	min(str) کوچکترین کاراکتر را به ترتیب حروف الفبا بر میگرداند.
26	replace(old, new [, max])

	این تابع سه آرگومان می پذیرد که دوتای آنها الزامی و یکی از آنها بسته به نوع استفاده لازم می شود. در پارامتر اول مقداری را ارائه می دهیم و در پارامتر دوم مشخص می کنیم که این مقدار را با مقدار پارامتر اول جایگزین کن. پارامتر سوم یک عدد را می گیرد که نشانگر این است که تا چند مرتبه این فعل در رشته ی مورد نظر انجام شود.
27	rfind(str, beg=0,end=len(string)) عملکردی مشابه متد find() دارد با این تفاوت که جستجو را از سمت راست شروع می کند.
28	rindex(str, beg=0, end=len(string)) کارکرد مشابه متد index() دارد با این تفاوت که جستجو را بجای چپ از سمت راست شروع می کند.
29	rjust(width,[, fillchar]) در این متد آرگومان width نمایشگر تعداد کاراکتری که مایلید justify (هم تراز) کنید و fillchar نیز نشانگر مقداری می باشد که می خواهید از آن برای justify کردن استفاده نمایید. این تابع یک نسخه ی از رشته بازمی گرداند که در آن سمت چپ رشته با کاراکتر مشخص شده در پارامتر ورودی دوم پر شده تا طول کلی رشته با مقدار مشخص شده در پارامتر اول برابر شود.
30	rstrip() تمامی کاراکترهای مشخص شده (در صورت مشخص نکردن، پیش فرض کاراکتر space می باشد) در پارامتر ورودی را از پایان یک رشته حذف می کند.
31	split(str="", num=string.count(str)) این متد رشته ی مورد نظر را در قالب لیستی برمی گرداند و کلمات موجود در آن را توسط تفکیک گر (پیش فرض "") و به تعداد (زیر رشته های مشخص شده در ورودی num) مشخص شده تقسیم می کند.
33	startswith(str, beg=0,end=len(string))

	این متد بررسی می کند که رشته ی مورد نظر با زیررشته ی مشخص شده در پارامتر ورودی str آغاز شده یا خیر، اگر چنین بود مقدار true و در غیر این صورت مقدار false را برمی گرداند. Beg بیانگر اندیس شروع و end نشانگر اندیس پایان می باشد.
34	strip([chars]) عملیات هر دو متد ()lstrip و ()rstrip را بر روی رشته ی مورد نظر اجرا می کند.
35	swapcase() حروف کوچک را بزرگ و حروف بزرگ را کوچک می کند.
36	title() یک کپی از رشته بازمی گرداند که اولین حرف تمامی کلمات آن با حرف بزرگ و باقی آن ها با حروف کوچک چاپ شده است.
37	translate(table, deletechars="") متد translate یک کپی از رشته برمی گرداند که درآن تمامی کاراکترها با استفاده از table (ساخته شده از تابع ()maketrans در string module)، ترجمه شده است و در صورت نیاز تمامی کاراکترهای موجود در رشته ی deletechars را حذف می کند.
38	upper() حروف کوچک یک رشته را به حروف بزرگ تبدیل می کند.
39	zfill (width) کپی از رشته ی مورد نظر برمی گرداند که سمت چپ آن با صفر پر شده تا طول نهایی رشته برابر با طول مشخص شده در پارامتر width شود.
40	isdecimal() این متد در صورتی که رشته ی یونیکد دربردارنده ی فقط کاراکترهای ده دهی باشد، true و در غیر این صورت false برمی گرداند.

نوع داده ای لیست در پایتون (tuple/lists)

اساسی ترین ساختار داده ای در پایتون sequence (زنجیره ای از بایت ها) می باشد. به هر یک از المان های یک sequence یک عدد اختصاص داده می شود که همان شماره ی مکان قرار گیری یا اندیس می باشد. اندیس در زبان پایتون از صفر آغاز می شود.

پایتون در کل 6 نوع داده ای که ساختار آن sequence می باشد، ارائه می دهد که پرکاربردترین آن ها عبارتند از list ها و tuple ها.

عملیات خاصی وجود دارد که می توان بر روی انواع داده ای که دارای ساختاری sequence و دنباله دار هستند انجام داد. این عملیات شامل فهرست کردن با اندیس، برش، اضافه کردن، ضرب و بررسی عضویت می باشد. بعلاوه، پایتون دارای توابع درون ساخته (توکار) می باشد که طول یک دنباله را بدست آورده و بزرگترین یا کوچکترین المان های آن را پیدا/مشخص می کند.

نوع داده ای list

list تطبیق پذیرترین نوع داده ای در پایتون می باشد که به صورت یک لیست نوشته می شود و آیتم های آن توسط ویرگول محصور در [] از یکدیگر جدا می شوند. مهم ترین نکته ای که بایستی درباره ی نوع داده ای لیست به خاطر داشت، این است که آیتم های محصور در آن باید از یک نوع باشد.

لیست به مجموعه‌ای داده ای اشاره دارد که به صورت عادی به هم مرتبط اند. به جای ذخیره این داده‌ها به عنوان متغیرهای جداگانه ما می‌توانیم آن‌ها در یک لیست ذخیره کنیم.

ایجاد و اعلان یک لیست به سادگی قرار دادن مقادیری بین [] و تفکیک آن ها به وسیله ی ویرگول می باشد. مثال:

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
```

درست مشابه اندیس رشته، اندیس لیست ها از صفر آغاز می شود. بر روی لیست عملیاتی همچون برش (slice) و اتصال (concatenation) را می توان اجرا کرد.

دسترسی به مقادیر یک لیست

جهت دسترسی به مقادیر یک لیست، بایستی اندیس شروع و پایان را داخل [] فراهم نمود، با این کار مقادیری که در آن شماره ی مکان قرارگیری یا اندیس ذخیره شده اند، قابل دسترسی می شوند.

```
#!/usr/bin/python
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];
print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

پس از اجرا کد، نتیجه ی زیر بدست می آید:

```
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

بروز رسانی لیست ها

می توان مقدار یک لیست را برش داده و آن را با مقدار دیگری جایگزین نمود و از این طریق آن لیست را بروز رسانی کرد. برای این منظور در سمت راست عملگر تخصیص علامت [] و اندیس مقدار مربوطه را تایپ کنید. همچنین با استفاده از متد append() می توان عناصری را به لیست جاری الحاق کرد.

```
#!/usr/bin/python
list = ['physics', 'chemistry', 1997, 2000];
print "Value available at index 2 : "
print list[2]
list[2] = 2001;
print "New value available at index 2 : "
print list[2]
```

پس از اجرای کد، نتیجه ی زیر حاصل می گردد:

```
Value available at index 2 :
1997
New value available at index 2 :
2001
```

حذف المان های لیست

برای حذف المان مورد نظر از لیست، اگر المان خاصی مورد نظرتان است دستور del و اگر آیتم مشخصی برای حذف مد نظر ندارید، متد remove() را بکار ببرید.

```
#!/usr/bin/python
list1 = ['physics', 'chemistry', 1997, 2000];
print list1
del list1[2];
print "After deleting value at index 2 : "
print list1
```

نتیجه ی بدست می آید:

```
'physics', 'chemistry', 1997, 2000]
After deleting value at index 2 :
['physics', 'chemistry', 2000]
```

عملیات ابتدایی که روی لیست اجرا می شود

عملگرهای * و + در لیست همان عملیاتی را انجام می دهند که در رشته اجرا می کنند. همان طور که می دانید هر یک به ترتیب عملیات تکرار و اتصال را صورت می دهند، با این تفاوت که خروجی یک لیست جدید می باشد و نه یک رشته.

بر روی لیست می توان همان عملیات رایجی که بر روی ساختارهای دنباله ای (sequence) همچون رشته قابل اجرا می باشد، انجام داد.

عبارت پایتون	نتیجه	شرح
len([1, 2, 3])	3	Length (طول)
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation

(اتصال)		
Repetition (تکرار)	['Hi!', 'Hi!', 'Hi!', 'Hi!']	**['Hi!'] * 4**
Membership (عضویت)	True	**3 in [1, 2, 3]**
Iteration (حلقه تکرار)	1 2 3	**for x in [1, 2, 3]: print x,**

اندیس گذاری، برش و ماتریس

از آن جایی که list ها زیرمجموعه ی ساختار داده ای sequence هستند، اندیس گذاری و اتصال در هر دو نوع داده ای لیست و رشته یکسان می باشد.

ورودی زیر را درنظر داشته باشید:

```
L = ['spam', 'Spam', 'SPAM!']
```

شرح	نتیجه	عبارت
اندیس از صفر آغاز می شود، می شمارد و نتیجه را برمی گرداند.	'SPAM!'	**L[2]**
اگر منفی بود از سمت راست می شمارد و برمی گرداند.	'Spam'	**L[-2]**
از یک اندیسی به بعد را بازمی گرداند.	['Spam', 'SPAM!']	**L[1:]**

توابع و متدهای توکار لیست در پایتون

شماره ی متد	تابع مورد نظر با شرح عملکرد آن
1	cmp(list1, list2) المان های موجود در دو لیست را با هم مقایسه می کند.
2	len(list) طول کلی لیست را بازمی گرداند.
3	max(list) آیتمی که بیشترین مقدار را دارد از میان آیتم های موجود در لیست بازمی گرداند.
4	min(list) آیتمی با کوچکترین مقدار را از میان آیتم های موجود در لیست بازمی گرداند.
5	list(seq) نوع داده ای tuple را به list تبدیل می کند.

متدهای مربوط به لیست:

شماره ی متد	متد مورد نظر با شرح آن
1	list.append(obj)

	این متد شی پاس داده شده به عنوان ورودی را به لیست اضافه می کند.
2	list.count(obj) تعداد دفعاتی که شی مورد نظر در لیست تکرار شده را برمی گرداند.
3	list.extend(seq) محتویات پارامتر ورودی (seq) را به لیست الحاق می کند.
4	list.index(obj) اندیس یا شماره ی مکان قرار گیری شی را در لیست برمی گرداند.
5	list.insert(index, obj) شی پاس داده شده به متد را در اندیس مشخص شده درج می کند.این متد دو پارامتر می گیرد که اولی اندیس را مشخص می کند و دومی شی ای که باید در آن شماره ی مکان قرار گیری وارد شود.
6	list.pop(obj=list[-1]) آخرین شی موجود در لیست را برش داده و برمی گرداند. پارامتر دومی اختیاری است که نشانگر اندیس شی ای که باید برش داده شود می باشد.
7	list.remove(obj) شی مورد نظر (مشخص شده داخل پرانتز) را از لیست حذف می کند.
8	list.reverse() مکان قرارگیری اشیا در لیست را معکوس (جابجا) می کند.
9	list.sort([func])

اشیا یک لیست را مرتب می سازد.	

نوع داده ای tuple یا چندتایی در پایتون

Tuples درست شبیه لیست ها می‌باشند با این تفاوت که شما نمی‌توانید مقادیر آن‌ها را ویرایش کنید. مقادیر اولیه که برای تاپل ها تعیین می‌کنید ، تا آخر برنامه ثابت باقی می‌مانند و قابل تغییر نیستند.

tuple ها در پایتون چیزی شبیه به نوع داده ای list می باشند. تاپل ها تعدادی از مقادیر هستند که با ویرگول از یکدیگر تفکیک می شوند. tuple ها داخل () تعریف می شوند.

تفاوت بنیادین میان list ها و tuple ها این است که لیست ها داخل آکولاد [] مشخص می شوند و المان ها و نیز اندازه آن ها تغییر پذیر است در حالی که tuple ها درون پرانتز تعریف می شوند و قابلیت بروز رسانی را ندارند. tuple را می توان لیست های فقط خواندنی نیز نام گذاشت.

فرایند اعلان tuple در ایجاد لیستی از مقادیر تفکیک شده توسط ویرگول خلاصه می شود. در صورت تمایل می توان این مقادیر را داخل پرانتز محصور کرد. مثال:

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

مثالی از یک tuple تهی را در زیر مشاهده می کنید:

```
tup1 = ();
```

در صورت وجود حتی یک مقدار در tuple مورد نظر، بایستی از ویرگول استفاده شود:

```
tup1 = (50,);
```

اندیس در نوع داده ای tuple نیز از صفر آغاز می شود. عملیاتی همچون برش و استخراج مقدار، اتصال و غیره ... را می توان بر روی tuple اجرا نمود:

دسترسی به مقادیر یک tuple

جهت دسترسی به مقادیر موجود در یک tuple، می بایست از [] و اندیس مقدار مورد نظر استفاده کرد.

```
#!/usr/bin/python
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );
print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

نتیجه:

```
tup1[0]:  physics
tup2[1:5]:  [2, 3, 4, 5]
```

بروز رسانی tuple

همان طور که در بالا تشریح شد، tuple غیر قابل تغییر هستند؛ بدین معنا که امکان ویرایش و بروز رسانی آن ها وجود ندارد. با این حال می توان دو tuple را به هم متصل کرده و tuple جدید خلق کرد، همانند مثال زیر:

```
#!/usr/bin/python
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');
# Following action is not valid for tuples
# tup1[0] = 100;
# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3
```

نتیجه:

```
(12, 34.56, 'abc', 'xyz')
```

حذف المان های یک tuple

حذف المان های یک tuple به صورت تکی امکان پذیر نیست. با این وجود، می توان دو tuple مورد نظر را که المان های ناخواسته در آن لحاظ نشده، به هم متصل کرد و یک tuple جدید ایجاد نمود.

به منظور حذف کلی یک tuple کافی است دستور del را بکار ببرید:

```
#!/usr/bin/python
tup = ('physics', 'chemistry', 1997, 2000);
print tup
del tup;
print "After deleting tup : "
print tup
```

نتیجه ی زیر حاصل می گردد. همان طور که مشاهده می کنید، یک استثنا رخ داده است، زیرا با اجرا شدن دستور del، دیگر tuple وجود ندارد:

```
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last):
File "test.py", line 9, in <module>
print tup;
NameError: name 'tup' is not defined
```

عملیات رایج که بر روی tuple قابل اجرا می باشد

از آنجایی که tuple ها از نوع ساختمان داده ای sequence (زنجیره ای از بایت ها) هستند، می توان عملیاتی نظیر برش و اندیس گذاری را بر روی آن ها پیاده کرد. با درنظر داشتن ورودی زیر:

```
L = ('spam', 'Spam', 'SPAM!')
```

عبارت	خروجی	شرح
L[2]	'SPAM!'	اندیس از صفر آغاز می شود.
L[-2]	'Spam'	در صورت منفی بودن از راست شمرده و المان مد نظر را استخراج می کند.
L[1:]	['Spam', 'SPAM!']	از یک اندیس به بعد المان ها را برمی گرداند.

نکته ی قابل توجه: هر مجموعه ای که از اشیای تشکیل و سپس توسط تفکیک گری مانند ویرگول از هم جدا شده باشد، اما داخل delimiter هایی همچون پرانتز برای tuple و [] برای list محصور نشده باشد، در آن صورت پایتون با این مجموعه به مثابه ی tuple برخورد می کند.

```
#!/usr/bin/python
print 'abc', -4.24e93, 18+6.6j, 'xyz'
x, y = 1, 2;
print "Value of x , y : ", x,y
```

نتیجه:

```
abc -4.24e+93 (18+6.6j) xyz
Value of x , y : 1 2
```

توابع توکار tuple

شماره ی تابع	تابع و شرح کارکرد آن
1	cmp(tuple1, tuple2) المان های دو tuple را با هم مقایسه می کند.
2	len(tuple) طول (تعداد کل المان های موجود در) tuple را برمی گرداند.
3	max(tuple) آیتمی که دارای بیشترین مقدار است را از میان المان های tuple استخراج کرده و بازمی گرداند.
4	min(tuple) آیتمی که دارای کم ترین مقدار می باشد را برمی گرداند.
5	tuple(seq) یک لیست را به عنوان پارامتر پذیرفته و آن را به تاپل تبدیل می کند.

نوع داده ای Dictionary در پایتون

نوع داده ای dictionary در زبان همه منظوره ی پایتون شبیه به نوع جدول hash شده می باشد. آنها مانند associative arrays (آرایه های انجمنی) یا hash ها در perl هستند. دیکشنری ها می توانند هر نوعی از داده باشند، اما غالبا از نوع داده ی عددی یا رشته ای هستند. دیشکنری ها با کروشه یا کاراکتر {} تعریف می شوند و جهت دسترسی به مقادیر آن بایستی از آکولاد یا عملگر [] کمک گرفت.

در واقع دیکشنری مجموعه‌ای از جفت داده‌های به هم مرتبط می‌باشد.

در dictionary هر اسم یا کلید توسط تفکیک گر نقطه ویرگول از مقدار خود جدا می شود، آیتم ها نیز با ویرگول از هم تفکیک شده، سپس کل dictionary داخل کروشه محصور می گردد.

در پایتون، یک dictionary می تواند کاملا تهی باشد و هیچ مقداری داخل کروشه ی آن قرار نگیرد، بدین صورت:{} .

در نوع داده ای dictionary، کلید ها اسم های منحصر بفرد هستند (در حالی که مقادیر منحصر بفرد نیستند). مقادیر dictionary می تواند از هر نوعی باشد، اما کلیدها بایستی از نوع داده های غیرقابل تغییر مانند tuple، list و عدد باشد.

دسترسی به مقادیر در dictionary

به منظور دسترسی به المان های یک dictionary، لازم است از عملگر [] و اسم (کلید) المان مورد نظر استفاده کرد:

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

خروجی:

```
dict['Name']:  Zara
dict['Age']:  7
```

اگر سعی کنید به یک آیتم داده ای دیکشنری با کلیدی که عضو dictionary نیست دسترسی پیدا کنید، با خطا مواجه خواهید شد:

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
print "dict['Alice']: ", dict['Alice']
```

نتیجه ی زیر را بدست می دهد:

```
dict['Zara']:
Traceback (most recent call last):
File "test.py", line 4, in <module>
print "dict['Alice']: ", dict['Alice'];
KeyError: 'Alice'
```

بروز رسانی dictionary

برای بروز رسانی دیکشنری بایستی اسم المان را داخل آکولاد، در سمت چپ عملگر تخصیص تایپ کنید، سپس مقدار جدید را در سمت راست عملگر مزبور درج نمایید و یا جهت افزودن المان جدید کلید المان را در سمت چپ عملگر و مقدار را در سمت راست تایپ کنید:

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

خروجی:

```
dict['Age']:  8
dict['School']:  DPS School
```

حذف المان های dictionary

می توانید المان های یک dictionary را به صورت تکی پاک کنید یا تمامی محتویات آن را به صورت یکجا حذف نمایید. همچنین می توانید خود متغیر dictionary را به طور کلی و صریح حذف نمایید.

جهت حذف یک متغیر dictionary به صورت صریح، از دستور del استفاده می کنیم.

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
del dict['Name']; # remove entry with key 'Name'
dict.clear();     # remove all entries in dict
del dict ;        # delete entire dictionary
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

نتیجه ی زیر را بدست می دهد. مشاهده می کنید که یک خطا رخ داده و آن مربوط به عدم وجود dictionary پس از اجرای دستور del dict می باشد.

```
dict['Age']:
Traceback (most recent call last):
File "test.py", line 8, in <module>
print "dict['Age']: ", dict['Age'];
TypeError: 'type' object is unsubscriptable
```

خصیصه ی کلیدهای dictionary

در خصوص مقادیر، در زبان پایتون هیچ محدودیتی وجود ندارد. این مقادیر می توانند هر نوع شی باشند، توسط کاربر تعریف شده یا هر شی معمولی دیگری که به طور متداول در پایتون مورد استفاده قرار می گیرد، باشند.

دو نکته ی بسیار مهم در رابطه با کلیدهای dictionary وجود دارد، که در زیر شرح داده شده:

1. به ازای هر کلید نمی توان بیش از یک entry یا مقدار داشت، بدین معنی که یک کلید نمی تواند به هیچ وجه تکراری باشد. در صورت تخصیص دو کلید به یک مقدار، معمولا دومی به عنوان کلید انتخاب می شود. مثال:

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'};
```

```
print "dict['Name']: ", dict['Name']
```

نتیجه:

```
dict['Name']:  Manni
```

2. کلیدها بایستی تغییر ناپذیر باشند (پس از ایجاد تغییر نکنند)، بدین معنا که می توانید از رشته ها، اعداد یا tuple به عنوان کلید استفاده کنید، اما چیزی مانند [' key '] مجاز نیست.

مثال:

```
#!/usr/bin/python
dict = {['Name']: 'Zara', 'Age': 7};
print "dict['Name']: ", dict['Name']
```

نتیجه ی زیر بدست می آید:

```
Traceback (most recent call last):
File "test.py", line 3, in <module>
dict = {['Name']: 'Zara', 'Age': 7};
TypeError: list objects are unhashable
```

توابع و متدهای توکار Dictionary

شماره ی متد	تابع مورد نظر به همراه شرح عملکرد آن
1	cmp(dict1, dict2) المان های دو متغیر dictionary را با هم مقایسه می کند.
2	len(dict) طول یا تعداد کل آیتم های موجود در متغیر dictionary را برمی گرداند.
3	str(dict) معادل رشته ای یک متغیر dictionary را برمی گرداند. در واقع یک دیکشنری را به رشته تبدیل می کند.
4	type(variable)

نوع متغیر ارسالی را بازمی گرداند. اگر متغیر پاس داده شده، dictionary بود در آن صورت، نوع را dictionary مشخص می کند.	

متدهای پایتون همراه با شرح آن ها را در جدول زیر مشاهده می کنید:

شماره ی متد	شرح عملکرد متد
1	dict.clear() کلیه ی المان های متغیر dictionary را حذف می کند.
2	dict.copy() نسخه عینی (shallow copy) از متغیر dictionary بازمی گرداند. در فرایند shallow copying، B تمامی مقادیر فیلد (field value) A را کپی می کند. اگر مقدار فیلد یک آدرس حافظه باشد، تنها آدرس حافظه را دانلود می کند و اگر مقدار فیلد یک نوع اولیه (primitive type) باشد در آن صورت مقدار نوع اولیه را کپی می کند.
3	dict.fromkeys() یک dictionary جدید بازمی گرداند که کلیدهای المان توسط پارامتر اول و مقادیر آن توسط پارامتر دوم تعیین می شود.
4	dict.get(key, default=None) این متد کلید مورد نظر که توسط پارامتر اول مشخص می شود را در dictionary می یابد، در صوت یافت نشدن مقدار مورد نظر، یک مقدار پیش فرض (که در پارامتر دوم تعریف می شود) تعیین می کنیم که آن را برمی گرداند.
5	dict.has_key(key) در صورت یافتن کلید مورد نظر در dictionary،که توسط پارامتر key مشخص می شود، مقدار true برمی گرداند و در غیر این صورت false را بازیابی می نماید.

6	dict.items() جفت های کلید و مقدار المان های یک متغیر دیکشنری را برمی گرداند.
7	dict.keys() لیستی از کلیدهای متغیر dictionary را برمی گرداند.
8	dict.setdefault(key, default=None) عملکردی مشابه متد get() دارد، با این تفاوت که در صورت نیافتن مقدار مورد نظر، مقدار پیش فرض که در پارامتر دوم تعریف می شود را برمی گرداند.
9	dict.update(dict2) جفت های کلید و مقدار دیکشنری dict2 را به متغیر dict اضافه می کند.
10	dict.values() لیستی از مجموعه ی مقادیر موجود در dictionary را بازیابی می کند.

تاریخ و زمان در پایتون

برنامه ای که توسط زبان پایتون نوشته شده قادر است زمان و تاریخ را با فرمت های مختلف نمایش دهد. تبدیل تاریخ و نمایش آن در قالب های (فرمت های) یک روتین در برنامه های کامپیوتری می باشد. ماژول های تقویم و زمانی که در زبان پایتون طراحی شده این امکان را می دهد که تاریخ و زمان را رد گیری کرده و در قالب های مختلف نمایش داد.

تاریخ در پایتون اعداد ممیز شناور هستند که در واحد ثانیه محاسبه می شوند. برخی تاریخ ها در واحد ثانیه و از 1970(epoch)، January 1،12:00am محاسبه می شوند (تعداد ثانیه های سپری شده از تاریخ مزبور تاکنون).

در پایتون یک ماژول time وجود دارد که بسیار پرکاربرد بوده و توابعی برای کار با زمان و تبدیل فرمت نمایش آن فراهم می نماید. از جمله این توابع می توان به متد time.time() اشاره کرد. متد مذکور زمان جاری سیستم را برحسب ثانیه های سپری شده از تاریخ January 1،12:00am ، 1970(epoch) تا زمان کنونی محاسبه کرده و برمی گرداند.مثال:

```
#!/usr/bin/python
import time;  # This is required to include time module.

ticks = time.time()
print "Number of ticks since 12:00am, January 1, 1970:", ticks
```

نتیجه ی زیر را بدست می دهد:

```
Number of ticks since 12:00am, January 1, 1970: 7186862.73399
```

بسیاری از توابع مربوط به time در پایتون، زمان را به صورت tuple نه تایی نمایش می دهند:

اندیس	فیلد	مقدار
0	4-digit year (سال به صورت چهار رقمی)	2008
1	Month	1 to 12
2	Day	1 to 31
3	Hour	0 to 23
4	Minute	0 to 59
5	Second	0 to 61 (60 or 61 are leap-seconds)
6	Day of Week	0 to 6 (0 is Monday)
7	Day of year	1 to 366 (Julian day)
8	Daylight savings	-1, 0, 1, -1 means library determines DST

	(ساعات تابستانی)	

Tuple فوق معادل ساختار struct_time می باشد. ساختار نام برده دارای attribute های زیر می باشد:

اندیس	Attribute	مقدار
0	tm_year	2008
1	tm_mon	1 to 12
2	tm_mday	1 to 31
3	tm_hour	0 to 23
4	tm_min	0 to 59
5	tm_sec	0 to 61 (60 or 61 are leap-seconds ثانیه ی 0 کبیسه/)
6	tm_wday	0 to 6 (0 is Monday)
7	tm_yday	1 to 366 (Julian day)
8	tm_isdst	-1, 0, 1, -1 means library determines DST

بازیابی زمان جاری

برای تبدیل زمان از مقدار عددی ممیز شناور که در قالب تعداد ثانیه های سپری شده از تاریخ January 1،12:00am 1970 نمایش داده می شود، به فرمت tuple نه تایی، بایستی آن مقدار ممیز

شناور را به یک تابع (برای مثال localtime) پاس دهید. این تابع در خروجی تاریخ را به صورت یک tuple نه تایی نمایش می دهد.

```
#!/usr/bin/python
import time;

localtime = time.localtime(time.time())
print "Local current time :", localtime
```

نتیجه ی زیر حاصل می گردد:

```
Local current time : time.struct_time(tm_year=2013, tm_mon=7,
tm_mday=17, tm_hour=21, tm_min=26, tm_sec=3, tm_wday=2, tm_yday=198, tm_isdst=0)
```

بازیابی تاریخ فرمت شده

می توان تاریخ را با توجه به نیاز خود فرمت کرد. روش ساده برای بازیابی تاریخ مورد نظر در فرمت خوانا، متد asctime() می باشد:

```
#!/usr/bin/python
import time;

localtime = time.asctime( time.localtime(time.time()) )
print "Local current time :", localtime
```

نتیجه:

```
Local current time : Tue Jan 13 10:17:09 2009
```

بازیابی و نمایش تقویم مربوط به ماه

ماژول calendar طیف وسیعی از متدها را برای کار با تقویم سالانه و ماهانه ارائه می دهد. در مثال زیر تقویم مربوط به ژانویه ی سال 2008 نمایش داده شده است:

```
#!/usr/bin/python
import calendar
```

```
cal = calendar.month(2008, 1)
print "Here is the calendar:"
print cal
```

خروجی:

```
Here is the calendar:
    January 2008
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

ماژول time

در پایتون ماژولی به نام time وجود دارد که چندین تابع برای کار با زمان و تبدیل فرمت نمایش تاریخ ارائه می دهد. در جدول زیر لیستی از این توابع را مشاهده می کنید:

شماره ی متد	تابع و شرح عملکرد آن
1	time.altzone The offset of the local DST timezone, in seconds west of UTC, if one is defined. This is negative if the local DST timezone is east of UTC (as in Western Europe, including the UK). Only use this if daylight is nonzero. اختلاف زمان هماهنگ جهانی تا ساعت محلی را به ثانیه محاسبه می کند.
2	time.asctime([tupletime]) این متد توابع gmtime یا localtime را به عنوان ورودی پذیرفته، سپس خروجی آن را به صورت رشته ی 24 کاراکتری برمی گرداند (برای مثال 'Tue Dec 11 18:07:14 2008').
3	time.clock( ) زمان فعلی پردازنده را در قالب یک عدد ممیز شناور بر حسب ثانیه برمی گرداند.

4	time.ctime([secs]) در صورت داشتن پارامتر مانند تابع asctime(localtime(secs)) عمل می کند و در صورت دم تعریف پاس دادن پارامتر به آن مانند متد asctime( ) عمل می کند.
5	time.gmtime([secs]) یک تاریخ را بر حسب ثانیه های سپری شده از تاریخ 1970 به عنوان ورودی دریافت می کند و در خروجی به صورت tuple نه تایی بر اساس زمان جهانی یا UTC برمی گرداند.
6	time.localtime([secs]) یک تاریخ را برحسب ثانیه های سپری شده از 1970به عنوان ورودی پذیرفته و در خروجی به صورت tuple های نه تایی بر اساس زمان محلی برمی گرداند.
7	time.mktime(tupletime) یک تاریخ را در قالب tuple های نه تایی که بر اساس زمان محلی می باشد پذیرفته و در خروجی به صورت عدد ممیز شناور، بر حسب ثانیه های سپری شده از تاریخ 1970 برمی گرداند.
8	time.sleep(secs) تعداد ثانیه هایی که اجرا به صورت موقت متوقف می شود را به عنوان ورودی می پذیرد. اجرا را به مدت مشخص شده در پارامتر ورودی، متوقف می سازد.
9	time.strftime(fmt[,tupletime]) یک تاریخ را (بر اساس زمان محلی) در قالب tuple های نه تایی به عنوان ورودی پذیرفته و آن را به صورت رشته (که فرمت آن توسط پارامتر ورودی اول مشخص می شود) برمی گرداند.
10	time.strptime(str,fmt='%a %b %d %H:%M:%S %Y') تاریخ را در قالب رشته به عنوان ورودی پذیرفته سپس آن را بر اساس فرمتی که پارامتر fmt مشخص می کند، parse می کند و به صورت tuple نه تایی برمی گراند.

11	time.time( ) زمان جاری را به صورت عدد ممیز شناور برحسب ثانیه های سپری شده از تاریخ 1970 برمی گرداند.

Attribute هایی که با ماژول time بکار می روند، به شرح زیر می باشند:

SN	Attribute و شرح آن
1	time.timezone Attribute time.timezone is the offset in seconds of the local time zone (without DST) from UTC (>0 in the Americas; in most of Europe, Asia, Africa). اختلاف زمانی UTC از زمان منطقه ای را (بدون درنظر گرفتن ساعت تابستانی یا DTS) محاسبه کرده و بر حسب ثانیه برمی گرداند (در آمریکا 0< و 0=> در اروپا، آسیا و آفریقا).
2	time.tzname Attribute time.tzname is a pair of locale-dependent strings, which are the names of the local time zone without and with DST, respectively.

ماژول calendar

ماژول calendar توابعی برای کار با تقویم ارائه می دهد که به وسیله ی آن می توان، به عنوان نمونه، تقویم یک ماه یا سال مشخص را چاپ کرد.

به صورت پیش فرض، ماژول ذکر شده، روز دوشنبه را به عنوان اولین روز هفته و یکشنبه را آخرین روز آن درنظر می گیرد. برای تغییر این روال پیش فرض، بایستی تابع calendar.setfirstweekday() را صدا بزنید.

در زیر لیستی از توابع کار با تقویم را مشاهده می کنید:

شماره ی متد	تابع و شرح آن
1	calendar.calendar(year,w=2,l=1,c=6) تقویم سال را به صورت یک رشته ی چند خطی برمی گرداند که در آن سال با سه ستون مشخص شده است و ستون ها با c (تعداد فاصله ها) از هم جدا شده اند، w (تعداد کاراکترهای هر تاریخ) طول هر خط 21*w+18+2*c و L تعداد خطوط درنظر گرفته شده برای هر هفته است.
2	calendar.firstweekday( ) تنظیمات جاری برای روز اول هفته را برمی گرداند. به صورت پیش فرض، هنگامی که ماژول تقویم را import می کنید، اولین روز هفته بر روی 0 (دوشنبه) تنظیم می شود.
3	calendar.isleap(year) در صورتی که سال جاری، سال کبیسه باشد، مقدار true و در غیر این صورت مقدار false را برمی گرداند.
4	calendar.leapdays(y1,y2) تعداد کل روزهای کبیسه را بین دو سال مشخص شده در پارامتر ورودی، برمی گرداند (y1,y2).
5	calendar.month(year,month,w=2,l=1) این تابع یک رشته چند خطی را برمی گرداند که در آن month، شماره ی ماه سال و ، شماره ی ماه سال و year سال است. سپس برای هر هفته یک خط به اضافه ی دو خط برای ستون درنظرمی گیرد. W تعداد کاراکترهای هر تاریخ و l تعداد خطوط هر هفته. طول هر سطر برابر است با 7*w+6.
6	calendar.monthcalendar(year,month) لیستی از اعداد صحیح برمی گرداند. هر زیرلیست بیانگر یک هفته می باشد. روزهایی که خارج از ماه موردنظر هستند، بر روی صفر تنظیم می شوند؛ در صورت وجود روز در آن ماه، بر روی روز مربوطه در آن ماه تنظیم می شود، 1 به بالا.
7	calendar.monthrange(year,month)

	دو عدد صحیح برمی گرداند. اولین عدد صحیح کد روز هفته برای اولین روز را برمی گرداند و دومی تعداد روزهای یک ماه را بازیابی می کند. کد روز دوشنبه (اولین روز هفته در حالت پیش فرض) 0 می باشد و یکشنبه 6 می باشد. کد ماه اول سال، ژانویه، 1 و ماه آخر سال، دسامبر، 12 می باشد.
8	calendar.prcal(year,w=2,l=1,c=6) مشابه تابع print calendar.calendar(year,w,l,c) عمل می کند.
9	calendar.prmonth(year,month,w=2,l=1) عملکردی مشابه تابع print calendar.month(year,month,w,l) دارد.
10	calendar.setfirstweekday(weekday) روز اول هفته را مشخص می کند. کد اولین روز 0 (به صورت پیش فرض دوشنبه) و کد روز آخر هفته (به صورت پیش فرض، روز یکشنبه) 6 می باشد.
11	calendar.timegm(tupletime) این تابع درست برعکس تابع time.gttime می باشد: یک تاریخ را در قالب tuple های نه تایی به عنوان ورودی پذیرفته و همان تاریخ را به صورت عدد ممیز شناور (بر حسب ثانیه سپری شده از تاریخ 1970) برمی گرداند.
12	calendar.weekday(year,month,day) کد روز هفته برای روز مشخص شده در پارامتر سوم را برمی گرداند. کد روز دوشنبه، 0 و یکشنبه 6 می باشد.

توابع در پایتون (Function)

تابع یک قطعه کد سازمان دهی شده است که می توان آن را بارها فراخوانی کرده و مورد استفاده قرار داد. تابع به منظور اجرای یک عملیات خاص بکار می رود. توابع modularity (قابلیت تفکیک مولفه های سیستم و ادغام مجدد آن ها؛ در واقع modularity معماری نرم افزار را به کامپوننت هایی تقسیم می کند که پیاده سازی و نگهداشت آن را آسان می سازد) برنامه و قابلیت استفاده ی مجدد آن را بالا می برد.

همان طور که می دانید، پایتون توابع درون ساخته ی متعددی همچون print() ارائه می دهد، با این حال کاربر می تواند توابع خود را تعریف کند که به آن توابع user-defined یا توابع کاربر می گویند.

تعریف تابع

می توانید توابعی تعریف کنید که عملیات دلخواه را انجام دهد. برای تعریف توابع کاربر، بایستی از قوانین زیر پیروی کرد:

1. قطعه کد تابع باید با کلیدواژه ی def آغاز شود. به دنبال آن اسم تابع و پرانتز درج می شود (()).

2. پارامترهای ورودی یا آرگومان ها باید داخل پرانتز قرار داده شوند.

3. اولین دستور تابع می تواند یک دستور اختیاری باشد - function_docstring.

4. قطعه کد داخل ساختمان یا بدنه ی تابع با دو نقطه آغاز می شود، سپس دستوراتی که زیر آن قرار می گیرند، همگی توگذاشته می شوند.

5. دستور return اجرای تابع را متوقف کرده نتیجه را برمی گرداند (جمع بندی یک سری عملیات و یا کارهایی رو نمایش می دهد) و در صورت نیاز یک عبارت را به فراخواننده پاس می دهد. دستور return None نیز یعنی هیچ مقداری را به عنوان خروجی برنگرداند.

نحوه ی نگارش (syntax):

```
def functionname( parameters ):
"function_docstring"
function_suite
return [expression]
```

پارامترها به ترتیبی که تعریف شده اند، عمل می کنند و بایستی آن ها را به همان ترتیبی که تعریف شده اند، مقداردهی کرد.

مثال

تابع زیر یک رشته به عنوان ورودی پذیرفته و آن را چاپ می کند.

```
def printme( str ):
"This prints a passed string into this function"
print str
```

```
return
```

فراخوانی تابع

با تعریف تابع فقط یک اسم به آن تخصیص می یابد، سپس پارامترهای آن مشخص شده و ساختمان کد ایجاد می شود.

پس از اینکه ساختمان تابع ایجاد می شود، می توانید آن را از تابع دیگر صدا بزنید یا آن را مسقیم از پنجره ی prompt پایتون فراخوانی کنید. مثال زیر تابع printme() را صدا می زند:

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
"This prints a passed string into this function"
print str
return;
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

نتیجه ی زیر حاصل می گردد:

```
I'm first call to user defined function!
Again second call to the same function
```

ارسال پارامتر با reference در برابر ارسال با مقدار

تمامی پارامترها (آرگومان ها) در زبان پایتون با reference پاس داده می شوند، بدین معنی که اگر آنچه یک پارامتر به آن اشاره دارد را در تابع تغییر دهید، تغییر در تابع فراخواننده نیز منعکس می شود.

```
#!/usr/bin/python
# Function definition is here
def changeme( mylist ):
"This changes a passed list into this function"
mylist.append([1,2,3,4]);
print "Values inside the function: ", mylist
return
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

در اینجا reference به شی ارسالی حفظ شده و مقادیر جدید را به همان شی الصاق می کنیم.

نتیجه:

```
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

یک مثال دیگر را در زیر مشاهده می کنید که آرگومان با reference ارسال شده و reference مورد نظر در تابع فراخوانده شده، بازنویسی (overwrite) شده است.

```
#!/usr/bin/python
# Function definition is here
def changeme( mylist ):
"This changes a passed list into this function"
mylist = [1,2,3,4]; # This would assig new reference in mylist
print "Values inside the function: ", mylist
return
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

پارامتر mylist، نسبت به تابع changeme محلی (local) می باشد. ویرایش پارامتر مزبور در تابع موردنظر هیچ تاثیری بر روی mylist نمی گذارد. درواقع تابع هیچ کار خاصی انجام نمی دهد، نتیجه ای که از آن حاصل می گردد به شرح زیر می باشد:

```
Values inside the function:  [1, 2, 3, 4]
Values outside the function:  [10, 20, 30]
```

آرگومان های تابع

می توان یک تابع را به وسیله ی نوع آرگومان های لیست شده در زیر، فراخوانی کنید:

1. آرگومان های الزامی
2. آرگومان های Keyword
3. آرگومان های پیش فرض
4. آرگومان های با طول متغیر (Variable-length)

آرگومان های الزامی

آرگومان های الزامی، آرگومان هایی هستند که به ترتیب (تعریف شده) به تابع مورد نظر پاس داده می شوند. در اینجا، تعداد آرگومان هایی که در فراخوانی تابع مشخص می شود باید با تعریف تابع منطبق باشد.

برای فراخوانی تابع printme()، می بایست یک آرگومان به آن ارسال کنید، در غیر این صورت خطای نحوی (syntax error) می دهد:

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
"This prints a passed string into this function"
print str
return;
# Now you can call printme function
printme()
```

آرگومان های keyword

آرگومان های keyword در فراخوانی توابع مورد استفاده قرار می گیرد. هنگامی که از آرگومان های keyword در فراخوانی تابع استفاده می کنید، فراخواننده آرگومان ها را به وسیله ی اسم آن (پارامتر) شناسایی می کند.

این کار به شما اجازه می دهد ترتیب آرگومان ها را تغییر دهید، زیرا که مفسر پایتون قادر است با استفاده از کلیدواژه ای ارائه شده، مقادیر را به پارامترها match (وصل) کند. می توانید تابع printme() را به ترتیب زیر فراخوانی کنید:

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
"This prints a passed string into this function"
print str
return;
# Now you can call printme function
```

```
printme( str = "My string")
```

کد بالا پس از اجرا، نتیجه ی زیر را بدست می دهد:

```
My string
```

مثال زیر تصویر روشن تری از آن ارائه می دهد. توجه داشته باشید که ترتیب پارامترها اهمیتی ندارد.

```
#!/usr/bin/python
# Function definition is here
def printinfo( name, age ):
"This prints a passed info into this function"
print "Name: ", name
print "Age ", age
return;
# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

خروجی:

```
Name:  miki
Age  50
```

آرگومان پیش فرض آرگومانی است که در صورت مشخص نکردن مقداری در فراخوانی تابع برای آن، به صورت خودکار مقدار پیش فرض می پذیرد. نمونه ی زیر نشان می دهد که مقداری برای آرگومان age (در فراخوانی تابع) تعریف نشده، با این وجود تابع دوم مقدار 50 را برای آن چاپ می کند:

```
#!/usr/bin/python
# Function definition is here
def printinfo( name, age = 35 ):
"This prints a passed info into this function"
print "Name: ", name
print "Age ", age
return;
# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

نتیجه:

```
Name:  miki
Age  50
Name:  miki
```

```
Age  35
```

آرگومان های با طول متغیر (Variable-length arguments)

گاهی لازم است یک تابع را با آرگومان های بیشتری نسبت به آنچه در زمان تعریف تابع مشخص کردید، پردازش و فراخوانی کنید. این دست از آرگومان ها در اصطلاح آرگومان های با طول متغیر (variable length) خوانده می شوند و برخلاف آرگومان های الزامی و پیش فرض، در تعریف تابع ذکر نمی شوند.

نحوه ی نگارش:

```
def functionname([formal_args,] *var_args_tuple ):
"function_docstring"
function_suite
return [expression]
```

علامت (*) پیش از اسم متغیر (vartuple) که دارنده ی آرگومان های متغیر nonkeyword است، درج می شود. لازم به ذکر است که این tuple، چنانچه به هنگام فراخوانی تابع (function call) هیچ آرگومان اضافی مشخص نشود، تهی باقی می ماند. مثال:

```
#!/usr/bin/python
# Function definition is here
def printinfo( arg1, *vartuple ):
"This prints a variable passed arguments"
print "Output is: "
print arg1
for var in vartuple:
print var
return;
# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

کد فوق نتیجه ی زیر را بدست می دهد:

```
Output is:
10
Output is:
70
60
50
```

توابع بی نام (Anonymous functions)

توابعی که به شیوه ی معمول و با درج کلیدواژه ی def تعریف نشده اند، توابع anonymous نام دارند. برای ایجاد توابع anonymous، بایستی از کلیدواژه ی lambda استفاده نمود.

1. توابعی که به شکل lambda تعریف شده اند، قادراند چندین آرگومان به عنوان ورودی بپذیرند، اما فقط یک مقدار را در قالب عبارت به عنوان خروجی برمی گرداند. همچنین نمی توانند چندین دستور یا عبارت درخود داشته باشند.

2. یک تابع anonymous نمی تواند به صورت مستقیم برای چاپ (print) فراخوانی شود، زیرا lambda به یک عبارت نیاز دارد.

3. توابع lambda دارای فضای نامی محلی (local namespace) خود هستند و نمی توانند به متغیرهایی که در لیست پارامتر خود آورده نشده و نیز متغیرهایی که در فضای نامی سراسری هستند، دسترسی داشته باشند.

4. اگرچه بنظر می رسد که lambda ها، نسخه ی تک خطی از یک تابع هستند، با این وجود معادل دستورات درون برنامه ای (in-line statement) در زبان های C و C++ محسوب نمی شوند که هدف از آن افزایش کارایی تابع به وسیله ی ارسال پشته ی تخصیص تابع هنگام فراخوانی است.

ساختار نگارشی

سینتکس توابع lambda همان طور که در نمونه ی زیر مشاهده می کنید، شامل تنها یک خط می باشد:

```
lambda [arg1 [,arg2,.....argn]]:expression
```

در زیر نحوه ی عملکرد تابعی که به صورت lambda تعریف شده باشد، را مشاهده می کنید:

```
#!/usr/bin/python
# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2;
# Now you can call sum as a function
print "Value of total : ", sum( 10, 20 )
```

```
print "Value of total : ", sum( 20, 20 )
```

نتیجه:

```
Value of total :  30
Value of total :  40
```

دستور return

دستور return [expression] عملیات تابع را به پایان می رساند و خروجی آن را برمی گرداند و در صورت لزوم یک عبارت را به فراخواننده ارسال می نماید. دستور return ای که جلوی آن هیچ آرگومانی درج نشده باشد برابر با return none می باشد.

مثال های بالا هیچ مقداری را برنمی گردانند. مثال زیر یک مقدار را از تابع به صورت زیر برمی گرداند:

```
#!/usr/bin/python
# Function definition is here
def sum( arg1, arg2 ):
# Add both the parameters and return them."
total = arg1 + arg2
print "Inside the function : ", total
return total;
# Now you can call sum function
total = sum( 10, 20 );
print "Outside the function : ", total
```

نتیجه ی آن را در زیر مشاهده می کنید:

```
Inside the function :  30
Outside the function :  30
```

حوزه ی دسترسی متغیر (variable scope)

امکان دسترسی به تمامی متغیرهایی که در مکان های مختلف یک برنامه قرار دارند، وجود ندارد.

قابلیت دسترسی به یک متغیر درواقع به مکان تعریف متغیر بستگی دارد.

حوزه ی دسترسی یا scope تعیین می کند که در چه قسمت هایی از برنامه می توانید به شناسه ی مورد نظر دسترسی داشته باشید. در کل دو نوع حوزه ی دسترسی در پایتون وجود دارد:

1. متغیرهای سراسری (global)
2. متغیرهای محلی (local)

مقایسه ی متغیر سراسری با محلی

متغیرهایی که داخل بدنه ی تابع تعریف می شوند، حوزه ی دسترسی آن ها محلی محسوب می شود. متغیرهایی که بیرون بدنه یا ساختمان تابع تعریف می شوند، متغیرهای سراسری نامیده می شوند.

متغیرهای محلی را فقط می توان درون تابعی که در آن (متغیر) تعریف شده، مورد دسترسی قرار داد، در حالی که متغیرهای سراسری از تمام بخش های برنامه (توسط تابع) قابل دستیابی می باشد. به هنگام فراخوانی تابع، متغیرهای تعریف شده داخل آن همگی قابل دسترسی می باشند (در حوزه ی دسترسی قرار می گیرند). مثال:

```
#!/usr/bin/python
total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
# Add both the parameters and return them."
total = arg1 + arg2; # Here total is local variable.
print "Inside the function local total : ", total
return total;
# Now you can call sum function
sum( 10, 20 );
print "Outside the function global total : ", total
```

نتیجه:

```
Inside the function local total :  30
Outside the function global total :  0
```

ماژول ها در پایتون (module)

ماژول به شما این امکان را می دهد که کدهای خود را در پایتون سازمان دهی کنید. گروه بندی کدهای مرتبط با هم در یک ماژول، خوانایی کد و قابلیت استفاده از آن را بهبود می بخشد. ماژول یک شی است که دارای متغیرهای عضو (attribute) می باشد. این متغیرها را می توان bind(متصل) کرده و مورد ارجاع (reference) قرار داد.

ماژول درواقع یک فایل است که حاوی کد پایتون می باشد. ماژول توابع، کلاس ها و متغیرهایی را در اختیار شما قرار می دهد. ماژول همچنین می تواند دربردارنده ی کد اجرایی باشد.

مثال:

کد پایتون ماژول aname، داخل فایل aname.py جای گذاری می شود. در زیر مثالی از یک ماژول ساده (support.py) را مشاهده می کنید:

```
def print_func( par ):
print "Hello : ", par
return
```

دستور import

می توان با استفاده از دستور import، یک source file پایتون را در source file دیگری مورد استفاده قرار داد. نحوه ی استفاده از دستور import به ترتیب زیر می باشد:

```
import module1[, module2[,... moduleN]
```

هنگامی که مفسر با دستور import مواجه می شود، اگر آن ماژول در search path (مسیر جستجو) موجود باشد، ماژول مربوطه را وارد برنامه ی جاری می کند. search path، لیستی از پوشه ها (directory) است که مفسر در آن ها جستجو کرده و در صورت یافتن ماژول مورد نظر آن را وارد می کند. برای مثال، به منظور وارد کردن ماژول hello.py، می بایست دستور زیر را بالای اسکریپت درج نمایید:

```
#!/usr/bin/python
# Import module support
import support
# Now you can call defined function that module as follows
support.print_func("Zara")
```

خروجی:

```
Hello : Zara
```

یک ماژول، صرف نظر اینکه چندبار وارد (import) می شود، فقط یکبار بارگذاری می گردد. در صورت وجود چندین نمونه از دستور import، این امر مانع از این می شود که ماژول بارها و بارها اجرا شود.

The from...import Statement

دستور from، به شما اجازه می دهد متغیرهای عضو (attribute) را از یک ماژول وارد فضای نامی جاری کنید. طریقه ی بکار بردن دستور from...import در زیر نمایش داده شده است:

```
from modname import name1[, name2[, ... nameN]]
```

برای مثال، جهت وارد کردن تابع fibonacci از ماژول fib، دستور زیر را استفاده نمایید:

```
from fib import fibonacci
```

این دستور کل ماژول fib را در فضای نام جاری وارد نمی کند، بلکه صرفا آیتم fibonacci را از ماژول fib داخل جدول سراسری symbol ماژول import شده وارد می نماید.

دستور from...import *

همچنین می توان تمامی اسم ها را از یک ماژول، وارد فضای نامی جاری کرد. این کار با استفاده از دستور زیر امکان پذیر می باشد:

```
from modname import *
```

دستور یاد شده، روشی آسان برای وارد کردن تمامی آیتم های مورد نظر از یک ماژول در فضای نام جاری می باشد. با این حال توصیه می شود از این دستور فقط مواقع ضروری استفاده کنید.

مکان یابی ماژول

به هنگام وارد کردن یک ماژول، مفسر زبان پایتون به ترتیب شرح داده شده در زیر به دنبال ماژول مورد نظر می گردد.

1. پوشه ی جاری.
2. در صورت نیافتن ماژول، پایتون هر پوشه (directory) را در shell variable که PYTHONPATH نام دارد جستجو می کند.
3. در صورت موفق نبودن دو روش ذکر شده، پایتون مسیر پیش فرض را سرچ می کند. در محیط UNIX، این مسیر پیش فرض /usr/local/lib/python/ می باشد.

مسیری که ماژول در آن جستجو می شود (module search path)، داخل ماژول system module در قالب متغیر sys.path ذخیره می شود. متغیر sys.path حاوی پوشه ی جاری، متغیر PYTHONPATH است و مقدار پیش فرض آن به مسیر نصب بستگی دارد.

متغیر PYTHONPATH

PYTHONPATH، همان طور که پیش تر در این سری آموزشی تشریح شد، یک environment variable (متغیرهای محیطی مجموعه‌ای از مقادیر نام‌گذاری‌شده هستند که قادراند چگونگی رفتار کردن پروسه های در حال اجرا را تغییر داده و بر روی آنها تاثیر بگذارند. متغیرهای محیطی، از فرایند parent به فرایندهای child به ارث می‌رسند. این متغیرها بخشی از محیط عملیاتی هستند که فرایند در آن اجرا می‌شود.) می باشد که از لیستی از پوشه ها (directory) تشکیل شده است. سینتکس متغیر نام برده مشابه shell variable، PATH می باشد.

در زیر مثالی از متغیر PYTHONPATH در سیستم عامل ویندوز را می بینید:

```
set PYTHONPATH=c:\python20\lib;
```

نمونه ای از متغیر محیطی PYTHONPATH از سیستم UNIX:

```
set PYTHONPATH=/usr/local/lib/python
```

فضای نامی و تعیین حوزه ی دسترسی

متغیرها اسم ها یا شناسه هایی هستند که به اشیا نگاشت (map) می شوند. فضای نام یک dictionary از اسم متغیر (کلید) و اشیای مرتبط با آن (مقادیر) هستند.

دستور پایتون می تواند به متغیرهایی که در فضای نام محلی و همچنین در فضای نام سراسری قرار دارد، دسترسی داشته باشد.چنانچه متغیر سراسری و محلی هر دو دارای اسمی یکسان باشند، متغیر محلی بر متغیر سراسری اولویت دارد.

هر تابع دارای فضای نام محلی و مختص به خود است. متدهای کلاس نیز از همان قوانین تعیین حوزه ی دسترسی که توابع معمولی دنبال می کنند، پیروی می کنند.

زبان پایتون برآورد می کند متغیرها سراسری هستند یا محلی. بدین معنی که فرض می گیرد هر متغیری که در یک تابع مقداردهی می شود، نسبت به آن تابع محلی می باشد.

از این رو، جهت تخصیص یک مقدار به متغیر سراسری در حوزه ی یک تابع، ابتدا بایستی از دستور سراسری استفاده کنید.

دستور global VarName به زبان پایتون اطلاع می دهد که VarName یک متغیر سراسری است، در پی آن پایتون جستجو برای متغیر مورد نظر در فضای نام محلی را متوقف می سازد.

فرض بگیرید، یک متغیر به نام Money در فضای نام سراسری تعریف کرده ایم. سپس داخل تابع، متغیر ذکر شده را مقداردهی می کنیم. به دنبال آن پایتون متغیر Money را یک متغیر محلی در نظر می گیرد. با این حال، پیش از اینکه متغیر محلی Money را تنظیم (set) کنیم، سعی کردیم به مقدار آن دسترسی پیدا کنیم. در نتیجه با خطای UnboundLocalError مواجه می شویم. برای رفع آن، دستور سراسری global Money را از حالت comment خارج می کنیم:

```
#!/usr/bin/python
Money = 2000
def AddMoney():
# Uncomment the following line to fix the code:
# global Money
Money = Money + 1
print Money
AddMoney()
print Money
```

تابع dir()

تابع توکار dir() لیست مرتب سازی شده ای برمی گرداند که حاوی رشته های متعدد می باشد. این رشته ها دربردارنده ی اسم ماژول ها می باشد.

لیستی که این تابع بازمی گرداند، دربردارنده ی ماژول ها، متغیرها و توابع می باشد که در ماژول تعریف شده اند. مثال:

```
#!/usr/bin/python
# Import built-in module math
import math
content = dir(math)
print content
```

نتیجه:

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

در اینجا، متغیر رشته ای – name – اسم ماژول می باشد و – file – اسم فایل می باشد که ماژول از آن بارگذاری می شود.

The globals() and locals() Functions –

توابع globals() و locals() اسم های فضای نام محلی و سراسری را بسته به مکانی که از آن فراخوانی می شود، برمی گرداند.

اگر تابع locals() از داخل یک تابع فراخوانی شود، در آن صورت تمامی اسم هایی که به صورت محلی قابل دسترسی می باشد، با صدا خوردن تابع نام برده برگردانده می شوند.

اگر تابع globals() ار درون تابع صدا زده شود، در آن صورت کلیه ی اسم هایی که به صورت سراسری قابل دسترسی هستند، با فراخوانی تابع ذکر شده بازیابی می شوند.

خروجی این دو تابع، متغیری از نوع داده ای dictionary خواهد بود. برای استخراج اسم ها، کافی است تابع keys() را مورد استفاده قرار دهید.

تابع reload()

هنگامی که ماژول در یک اسکریپت import می شود، کدی که در بالاترین بخش ماژول قرار می گیرد، تنها یکبار اجرا می شود.

بنابراین، اگر می خواهید کدی که در بالاترین قسمت یک ماژول قرار دارد را مجددا اجرا کنید، بایستی تابع reload() را مورد استفاده قرار دهید.

تابع reload() یک ماژول که قبلا وارد شده بود را مجددا import می کند. نحوه ی استفاده از آن به ترتیب زیر می باشد:

```
reload(module_name)
```

در این نمونه، module_name در واقع اسم ماژولی است که می خواهید مجددا بارگذاری یا reloadشود، نه رشته ای که حاوی اسم ماژول است. به عنوان مثال، برای بارگذاری مجدد ماژول hello، می بایست دستور زیر را وارد نمایید:

```
reload(hello)
```

پکیج ها در پایتون

پکیج یک پوشه ی فایل یا file directory است که ساختار سلسله مراتبی دارد و محیط برنامه ی پایتون را که از ماژول ها، subpackage ها و sub-subpackage ها تشکیل شده است را مشخص می کند.

فرض کنید، فایل Pots.py در پوشه ی Phone جای گرفته است. source code این فایل به ترتیب زیر می باشد:

```
#!/usr/bin/python
def Pots():
print "I'm Pots Phone"
```

به طور مشابه، دو فایل دیگر داریم که با همان اسم دربردارنده ی توابع متفاوتی هستند.

1. فایل Phone/Isdn.py حاوی تابع Isdn()

2. فایل Phone/G3.py دربردارنده ی تابع G3()

حال یک فایل __init__.py دیگر در پوشه ی Phone ایجاد کنید:

Phone/__init__.py

برای اینکه بتوانید پس از import کردن Phone، تمامی توابع خود را در آماده ی استفاده داشته باشید، بایستی دستور صریح import را در __init__.py به صورت زیر قرار دهید:

```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

بعد از اینکه خطوط فوق را به فایل __init__.py، اضافه کردید، با وارد کردن پکیج Phone، تمامی این کلاس ها در دسترس خواهند بود.

```
#!/usr/bin/python
# Now import your Phone Package.
import Phone
Phone.Pots()
Phone.Isdn()
Phone.G3()
```

نتیجه:

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

آموزش توابع مربوط به ورودی و خروجی در پایتون و آبجکت File - Python Files I/O

این فصل تمامی توابع پایه و پرکاربرد ورودی/خروجی قابل فراخوانی در زبان Python را شرح می دهد. جهت مطالعه و آشنایی با توابع بیشتر در زبان پایتون می توانید به مستندات آموزشی Python در وب سایت رسمی خود این زبان مراجعه نمایید.

چاپ خروجی در نمایشگر (Print)

آسان ترین روش برای تولید و چاپ خروجی استفاده از دستور print می باشد. جهت نمایش خروجی برای کاربر کافی است عبارات دلخواه را در حالی که توسط ویرگول از هم جدا شده اند، به این

دستور ارسال نمایید. این تابع عبارت هایی که به عنوان پارامتر به آن پاس می دهید را به مقادیر رشته ای تبدیل نموده و سپس در خروجی چاپ می کند.

```
#!/usr/bin/python
print "Python is really a great language,", "isn't it?"
```

دستور بالا خروجی زیر را در نمایشگر چاپ می کند.

```
Python is really a great language, isn't it?
```

خواندن و دریافت ورودی از صفحه کلید

Python دو تابع درون ساخته جهت خواندن متن و مقدار ورودی که به طور پیش فرض حاصل فشردن دکمه های صفحه کلید می باشد، در اختیار توسعه دهنده قرار می دهد. این توابع عبارت اند از:

- raw_input
- input

تابع raw_input

تابع raw_input([prompt]) یک خط متن یا نوشته را از ورودی خوانده (از کاربر دریافت می کند) و آن را به عنوان رشته (String) در خروجی بازمی گرداند.

```
#!/usr/bin/python
str = raw_input("Enter your input: ");
print "Received input is : ", str
```

این دستور از شما درخواست می کند یک مقدار رشته ای وارد نمایید و سپس همان رشته را در نمایشگر برای کاربر چاپ می کند. به عنوان مثال، زمانی که کاربر مقدار "Hello Python!" را تایپ می کند، خروجی آن به صورت زیر خواهد بود:

```
Enter your input: Hello Python
Received input is :  Hello Python
```

تابع input

input([prompt]) از لحاظ کاربرد تقریبا معادل raw_input می باشد، با این تفاوت تابع مزبور فرض را بر این می گذارد که مقدار ورودی یک عبارت پایتون مجاز بوده و نتیجه ی محاسبه شده و حاصل را در خروجی به شما بازمی گرداند.

```
#!/usr/bin/python
str = input("Enter your input: ");
print "Received input is : ", str
```

در مقابل ورودی کاربر، خروجی زیر را برمی گرداند.

```
Enter your input: [x*5 for x in range(2,10,2)]
Recieved input is :  [10, 20, 30, 40]
```

باز کردن و بستن فایل ها (اعمال تغییرات و مدیریت فایل ها)

تاکنون، صرفا مقداری ساده را از کاربر به عنوان ورودی دریافت کرده و در خروجی نمایش می دادیم. در این قسمت از مقاله ی آموزشی به بسط نحوه ی استفاده از فایل های واقعی و مدیریت محتوای آن ها می پردازیم.

زبان Python به طور پیش فرض تعدادی تابع درون ساخته جهت مدیریت و ویرایش فایل ها در اختیار برنامه نویس قرار می دهد. جهت انجام اغلب این عملیات ویرایش یا مدیریت فایل ها، می بایست از آبجکت file استفاده نمایید.

تابع Open

پیش از اینکه بتوان داخل یک فایل اطلاعات درج کرده یا محتوای آن را خواند، می بایست آن را با استفاده از تابع درون ساخته ی open() فراخوانی نمایید. این تابع یک آبجکت فایل ایجاد می کند که با استفاده از آن سایر متدهای تکمیلی مربوط به آبجکت مزبور را می توان فراخوانی کرد.

دستور نگارشی یا گرامر استفاده از file

```
file object = open(file_name [, access_mode][, buffering])
```

در زیر جزئیات و اطلاعات مربوط به هر یک از پارامترها را مشاهده می کنید:

- file_name: آرگومان حاضر در واقع یک مقدار رشته ای است که اسم فایلی که می خواهید به آن دسترسی داشته باشید را شامل می شود.
- access_mode: این پارامتر وضعیت و سطح عملیاتی که بر روی فایل قابل اجرا خواهد بود را مشخص می کند. به طور مثال فایل بایستی فقط خواندنی باشد یا اینکه اجازه ی درج اطلاعات و ضمیمه ی آن به فایل دیگر را بدهد. در زیر لیست کاملی از تمامی مقادیر قابل استفاده را مشاهده می کنید. این پارامتر اختیاری بوده و در حالت پیش فرض بر روی read تنظیم می باشد (r).
- file_name: آرگومان جاری یک مقدار رشته ای است که اسم فایلی که قصد دسترسی به آن را دارید، دربرمی گیرد.
- buffering: چنانچه مقدار buffering بر روی 0 تنظیم شده باشد، هیچگونه داده یا فایلی در بخش مربوط به buffer در حافظه به طور موقت ذخیره نمی شود (فایلی buffer نمی شود). چنانچه مقدار buffering برابر 1 باشد، line buffering به هنگام دسترسی به فایل مورد نظر انجام می شود. اگر مقدار buffering را یک مقدار integer یا عدد صحیح بزرگ تر از 1 قرار دهید، در آن صورت عملیات buffer و ذخیره ی موقتی داده ها در حافظه ی میانی با توجه به buffer size اعلان شده، انجام می گیرد. در صورت منفی بودن، این مقدار پیش فرض سیستم اعمال می شود (رفتار و عملکرد پیش فرض سیستم).

در زیر حالات مختلف باز کردن فایل با سطوح دسترسی مختلف تشریح شده است:

وضعیت های مختلف	شرح
r	یک فایل را تنها به منظور خواندن محتوای آن باز می کند. file pointer در ابتدای فایل قرار داده می شود. این وضعیت در واقع حالت پیش فرض باز کردن فایل می باشد.

rb	یک فایل را برای خواندن در فرمت باینری باز می کند. file pointer در ابتدای فایل جای می گیرد. این حالت باز کردن فایل، وضعیت پیش فرض می باشد.
r+	یک فایل را برای خواندن محتوا و نیز درج محتوای جدید در آن باز می کند. file pointer در ابتدای فایل قرار داده می شود.
rb+	یک فایل را برای خواندن محتوا و درج محتوای جدید در آن با فرمت باینری باز می کند. file pointer (اشاره گر) در ابتدای فایل درج می شود.
w	یک فایل را منحصرا به منظور درج محتوای جدید در آن باز می کند. اگر فایل از قبل موجود باشد، آن را بازنویسی می کند (overwrite) می کند. اگر فایل وجود نداشته باشد، یک فایل جهت درج اطلاعات جدید ایجاد می کند.
wb	یک فایل را منحصرا در قالب باینری و جهت درج اطلاعات جدید در آن باز می کند. محتوای فایل را در صورتی که از قبل چنین فایلی وجود داشته باشد، بازنویسی می نماید. اگر فایلی وجود نداشته باشد، خود یک فایل جدید ایجاد می کند.
w+	فایل مورد نظر را هم جهت خواندن و هم جهت درج اطلاعات در آن باز می کند. در صورتی که فایل از قبل موجود باشد، داده های آن را بازنویسی می کند. اگر فایلی وجود نداشته باشد، یک فایل جدید جهت خواندن اطلاعات و درج داده در آن ایجاد می نماید.
wb+	یک فایل را جهت خواندن و نوشتن اطلاعات جدید در آن در قالب باینری باز می کند. چنانچه فایلی از قبل موجود باشد، محتوای آن را بازنویسی می کند و در غیر این صورت یک فایل جدید جهت عملیات خواندن و نوشتن می سازد.
a	یک فایل را جهت ضمیمه کردن محتوای جدید در انتهای محتوای فعلی آن باز می نماید. چنانچه فایلی از قبل موجود باشد، آنگاه اشاره گر یا file pointer در انتهای فایل خواهد بود. به عبارت دیگر فایل در وضعیت append قرار داشته و با سطح دسترسی append (قابلیت افزودن متن جدید در انتهای فایل جاری) باز می شود. اگر فایلی وجود نداشت، در آن صورت فایلی جدید ایجاد خواهد شد.

ab	یک فایل را جهت افزودن محتوای جدید به انتهای آن در قالب باینری باز می نماید. اشاره گر (file pointer) در انتهای فایل جای می گیرد. بدین معنی که فایل در وضعیت append قرار داشته و اجازه ی ضمیمه کردن اطلاعات جدید به انتهای خود را به توسعه دهنده می دهد. اگر چنین فایلی وجود نداشت، یک فایل جدید برای نوشتن محتوای جدید در آن باز می شود.
a+	یک فایل برای افزودن محتوای جدید به انتهای فایل جاری و خواندن محتوای آن باز می کند. اشاره گر در انتهای فایل جاری قرار دارد. فایل مورد نظر در حالت append باز شده و اجازه ی درج محتوای جدید به انتهای آن را می دهد. اگر چنین فایلی وجود نداشت، فایل جدیدی برای خواندن و نوشتن محتوا ایجاد می نماید.
ab+	یک فایل را جهت افزودن محتوای جدید به انتهای آن و نیز خواندن اطلاعات جاری باز می کند. file pointer در انتهای فایل جاری قرار دارد. فایل در حالت append باز شده و اجازه ی اضافه شدن محتوا در انتهای خود را می دهد. اگر فایل موجود نبود، یک فایل جدید برای خواندن محتوا و درج اطلاعات در آن ایجاد می کند.

attribute های آبجکت file

پس از باز کردن فایل و ایجاد آبجکت file، می توانید اطلاعات مختلف مربوط به آن فایل را بازیابی نمایید.

در زیر لیستی از تمامی attribute های مربوط به آبجکت file را مشاهده می کنید:

Attribute	شرح
file.closed	در صورتی که فایل بسته شده باشد، مقدار true و در غیر این صورت false را برمی گرداند.

file.mode	وضعیت و سطح دسترسی که فایل با آن باز شده را در خروجی بازمی گرداند.
file.name	اسم فایل را در خروجی برمی گرداند.
file.softspace	یک مقدار بولی برمی گرداند که مشخص می کند آیا بایستی به هنگام استفاده از دستور print یک خط فاصله (کاراکتر space) قبل از چاپ مقدار مورد نظر، درج شود یا خیر.

مثال

```
#!/usr/bin/python
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

نتیجه ی زیر را در پی دارد:

```
Name of the file:  foo.txt
Closed or not :  False
Opening mode :  wb
Softspace flag :  0
```

متد close()

متد close() از آبجکت file تمامی اطلاعاتی که به طور دائمی درج نشده را پاک (flush) کرده و سپس آبجکت فایل را می بندد. پس از بسته شدن این آبجکت امکان درج اطلاعات جدید در فایل وجود نخواهد داشت. به عبارت دیگر با فراخوانی close() بر روی آبجکت file، فایل بسته شده و منابع اشغال شده توسط این فایل آزاد می شود. زمانی که reference object (آبجکت اشاره گر و ارجاع) یک فایل به فایل دیگری تخصیص می یابد، پایتون به صورت خودکار فایل مورد نظر (قبلی) را می بندد.

توصیه می شود همیشه با فراخوانی تابع مزبور، فایل را بسته و منابع مورد استفاده ی آن را آزاد نمایید.

ساختار نگارشی و نحوه ی استفاده از Close()

```
fileObject.close();
```

مثال

```
#!/usr/bin/python
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
# Close opend file
fo.close()
```

خروجی زیر را به دنبال دارد:

```
Name of the file:  foo.txt
```

خواندن و درج اطلاعات در فایل

آبجکت file تعدادی متد جهت دسترسی آسان به محتوای فایل در اختیار توسعه دهنده قرار می دهد. در زیر به شرح نحوه ی استفاده از read() و write() جهت نوشتن در فایل و خواندن اطلاعات آن می پردازیم.

متد write()

متد write() تمامی مقادیر رشته ای از هر نوعی را در فایل باز و قابل دسترس درج می کند. لازم به ذکر است که مقادیر رشته ای Python می توانند علاوه بر متن ساده، داده های باینری را نیز دربرداشته باشند.

متد write() کاراکتر ('\n') را به انتهای رشته اضافه نمی کند.

نحوه ی استفاده از متد

```
fileObject.write(string);
```

در قطعه کد بالا، پارامتر ارسالی دربردارنده ی محتوایی است که داخل فایل باز نوشته می شود.

مثال

```
#!/usr/bin/python
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n");
# Close opend file
fo.close()
```

متد بالا یک فایل foo.txt ایجاد کرده و محتوای ارسال شده به عنوان پارامتر را در فایل مورد نظر درج نموده و سرانجام آن فایل را می بندد. اگر شما این فایل را باز کنید، با محتوای زیر مواجه خواهید شد.

```
Python is a great language.
Yeah its great!!
```

متد read()

متد read() یک مقدار رشته ای را از فایل باز می خواند. لازم است دقت داشته باشید که رشته های Python می توانند علاوه بر متن داده های باینری نیز به همراه داشته باشد.

نحوه ی استفاده از متد

```
fileObject.read([count]);
```

در قطعه کد فوق، پارامتر ارسالی تعداد بایت هایی است که قرار است از فایل باز و بارگذاری شده در حافظه خوانده شود را مشخص می کند. این متد محتوای فایل را از ابتدا شروع به خواندن کرده و در صورت عدم اعلان پارامتر count، سعی می کند کل محتوای فایل را تا انتهای آن بخواند.

مثال

حال این متد را بر روی فایل foo.txt که در ابتدای مقاله ی حاضر ایجاد کردید، فراخوانی نمایید.

```
#!/usr/bin/python
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close opend file
fo.close()
```

نتیجه ی زیر را به دنبال دارد:

```
Read String is :  Python is
```

بازیابی موقعیت جاری در فایل (File Position)

متد ()tell موقعیت جاری داخل فایل را برمی گرداند. به عبارت دیگر، عملیات خواندن داده و درج اطلاعات جدید داخل فایل، از موقعیت جاری که بر حسب تعداد بایت های خوانده شده از اول فایل محاسبه شده، انجام می شود.

در واقع ()f.tell یک عدد صحیح (integer) برمی گرداند که موقعیت جاری file object را داخل فایل بازمی گرداند. موقعیت جاری بر حسب بایت از ابتدای فایل محاسبه می شود.

متد ([seek(offset[, from موقعیت جاری آبجکت file را داخل فایل تغییر می دهد. آرگومان offset تعداد بایت هایی که بایستی به جلو حرکت کند را مشخص می نماید. آرگومان دوم، from، آدرسی از خانه ی حافظه (reference position) یا بایت که حرکت از آنجا بایستی آغاز شود را مشخص می کند.

اگر from بر روی 0 تنظیم شده باشد، بدین معنی است که حرکت از ابتدای فایل آغاز می شود (ابتدای فایل به عنوان reference position انتخاب می شود). چنانچه مقدار این پارامتر برابر 1 قرار داده شده باشد، آنگاه حرکت از موقعیت جاری داخل فایل سر گرفته می شود. در صورتی که مقدار آرگومان مورد نظر بر روی 2 تنظیم شده باشد، انتهای فایل به عنوان موقعیتی که حرکت از آنجا باید آغاز شود، تنظیم می شود.

مثال

این توابع را بر روی فایل foo.txt آزمایش و فراخوانی می کنیم.

```
#!/usr/bin/python
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Check current position
position = fo.tell();
```

```
print "Current file position : ", position
# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str
# Close opend file
fo.close()
```

نتیجه ی زیر را در پی دارد:

```
Read String is :  Python is
Current file position :  10
Again read String is :  Python is
```

ویرایش اسم و حذف فایل ها

ماژول OS در پایتون، تعدادی متد کارا جهت انجام عملیات مربوط به پردازش فایل از جمله ویرایش اسم فایل و حذف آن ها از حافظه در اختیار توسعه دهنده قرار می دهد. به منظور استفاده از این ماژول، کافی است آن را با استفاده از دستور OS در بستر پروژه وارد کرده (import)، سپس توابع آن را به راحتی فراخوانی نمایید.

متد ()rename

متد ()rename دو آرگومان ورودی دریافت می کند: اسم فایل جاری و اسم فایل جدید.

نحوه ی استفاده از متد

```
os.rename(current_file_name, new_file_name)
```

مثال

مثال زیر اسم فایل جاری به نام test1.txt را تغییر می دهد:

```
#!/usr/bin/python
import os
# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

متد ()remove

می توانید با استفاده از متد ()remove فایل های مورد نظر را حذف نمایید. برای این منظور کافی است اسم فایل مد نظر را به عنوان آرگومان به تابع مذکور ارسال کنید.

نحوه ی استفاده از متد

```
os.remove(file_name)
```

مثال

در زیر مثالی را مشاهده می کنید که فایل test2.txt را به طور دائمی حذف می کند.

```
#!/usr/bin/python
import os
# Delete file test2.txt
os.remove("text2.txt")
```

پوشه های اصلی در پایتون و توابع مربوط به مدیریت دایرکتوری ها

فایل های اپلیکیشن طبیعتا در پوشه های مختلف جای گرفته اند. پایتون به واسطه ی ماژول OS توابع کارامد متعددی در اختیار برنامه نویس قرار می دهد که ایجاد، حذف و ویرایش پوشه های اصلی و دایرکتوری را تسهیل می بخشد.

متد mkdir()

می توانید با استفاده از متد mkdir() قابل فراخوانی از ماژول OS، پوشه هایی (directory) در دایرکتوری فعلی ایجاد نمایید. لازم است اسم دایرکتوری دلخواه را به عنوان پارامتر به این متد ارسال نمایید تا آن را ایجاد کند.

نحوه ی استفاده از متد

```
os.mkdir("newdir")
```

مثال

در زیر مثالی را مشاهده می کنید که در آن یک دایرکتوری به نام test در پوشه (دایرکتوری) جاری ایجاد می شود.

```
#!/usr/bin/python
import os
# Create a directory "test"
os.mkdir("test")
```

متد chdir()

جهت تغییر دایرکتوری یا پوشه ی جاری می توانید متد chdir() را فراخوانی نمایید. این متد اسم پوشه یا دایرکتوری که می خواهید پوشه ی جاری را به آن تغییر دهید، به عنوان آرگومان دریافت می کند.

نحوه ی استفاده از متد

```
os.chdir("newdir")
```

مثال

مثال زیر پوشه (دایرکتوری) جاری را به دایرکتوری "/home/newdir" تغییر می دهد.

```
#!/usr/bin/python
import os
# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```

متد getcwd()

متد getcwd() همان طور که از اسم آن پیدا است، موقعیت پوشه ی فعال و جاری را در خروجی نمایش می دهد.

دستور استفاده از متد

```
os.getcwd()
```

مثال

قطعه کد زیر موقعیت پوشه ی جاری را در خروجی بازگردانی می کند.

```
#!/usr/bin/python
import os
# This would give location of the current directory
os.getcwd()
```

متد rmdir()

متد rmdir() پوشه ی مورد نظر که به عنوان آرگومان به آن ارسال شده را به طور دائمی حذف می کند.

پیش از حذف پوشه (دایرکتوری)، لازم است محتوای آن به طور کامل پاک شده باشد.

دستور استفاده از متد

```
os.rmdir('dirname')
```

مثال

در زیر مثالی را می بینید که در آن پوشه ی "/tmp/test" به طور دائمی حذف می شود. لازم است اسم و آدرس پوشه که به صورت کامل قید شده را به عنوان پارامتر به متد نام برده ارسال نمایید چرا که در غیر این صورت متد به دنبال آن دایرکتوری در پوشه ی جاری می گردد.

```
#!/usr/bin/python
import os
# This would  remove "/tmp/test"  directory.
os.rmdir( "/tmp/test"  )
```

توابع مربوط به مدیریت فایل و دایرکتوری

سه منبع بسیار مهم وجود دارد که طیف گسترده ای از توابع کمکی و کارا جهت مدیریت و دستکاری فایل ها و پوشه ها در محیط سیستم عامل های Windows و Unix را در دسترس توسعه دهندگان قرار می دهد. این توابع به شرح زیر می باشند:

- متدهای آبجکت file (File Object Methods): آبجکت file توابع متعددی جهت مدیریت و ویرایش فایل ها در اختیار برنامه نویس می گذارد.
- متدهای آبجکت OS (OS Object Methods): توابعی که در دسترس توسعه دهنده قرار می دهد قادر هستند که علاوه بر فایل، پوشه را نیز (دایرکتوری) پردازش کنند.

مدیریت خطا در پایتون / Exception Handling

پایتون دو امکان ویژه برای مدیریت خطاهای پیشبینی نشده در اپلیکیشن ها فراهم کرده و قابلیت های اشکال زدایی (debugging) پرکاربردی به آن ها اضافه می کند.

- Exception Handling: در آموزش حاضر به تشریح مدیریت خطاها و استثناها خواهیم پرداخت. در زیر لیستی از خطاهای استاندارد از در پایتون را مشاهده می کنید.
- Assertion ها: این مبحث در آموزش مربوطه ی خود مورد پوشش قرار می گیرد.

لیستی از خطاهای متعارف در پایتون:

اسم خطا	شرح
Exception	کلاس پایه برای تمامی خطاها و exception ها در پایتون
StopIteration	زمانی صادر (raise) می شود که متد next() از کد تکرارکننده (iterator) به هیچ آبجکتی اشاره نکند. در واقع این خطا زمانی رخ می دهد که متد next() از iterator می خواهد اشاره کند که دیگر هیچ مقداری برای تکرار باقی نمانده است.
SystemExit	در نتیجه و به دلیل فراخوانی تابع sys.exit() صادر می شود.
StandardError	کلاس پایه برای تمامی خطاهای درون ساخته به جز StopIteration و SystemExit.
ArithmeticError	کلاس پایه برای تمامی خطاهایی که در نتیجه ی محاسبات عددی رخ می دهد.
OverflowError	زمانی رخ می دهد که خروجی محاسبات از حداکثر یا سقف نوع عددی مورد نظر فراتر رود. در واقع چنانچه نتیجه ی عملیات ریاضی و محاسبات بیش از حد قابل ارائه و نمایش باشد (بزرگ تر از ظرفیت نوع عددی مورد نظر و غیر قابل نمایش توسط این type باشد)، این خطا فراخوانی می شود.
FloatingPointError	

	این خطا زمانی بروز می دهد که عملیات ممیز شناور با شکست مواجه شود.
	Raised when division or modulo by zero takes place for all numeric types.
ZeroDivisionError	زمانی صادر می شود که عملیات تقسیم بر صفر برای انواع عددی رخ دهد. به عبارت دیگر این خطا زمانی رخ می دهد که آرگومان دوم یک عملیات تقسیم یا modulo عدد صفر باشد.
AssertionError	زمانی صدا زده می شود که دستور Assert (انتظار نتیجه ی مد نظر را در تست داشتن) با خطا مواجه شود.
	Raised in case of failure of attribute reference or assignment.
AttributeError	زمانی صدا زده می شود که attribute reference/assignment با خطا مواجه شود.
EOFError	این خطا هنگامی رخ می دهد که ورودی از تابع raw_input() یا input() دریافت نشده و نیز پایتون به انتهای فایل رسیده در حالی که قطعه مربوطه به طور صحیح بسته نشده باشد.
ImportError	زمانی رخ می دهد که دستور import با موفقیت اجرا نشود.
KeyboardInterrupt	هنگامی رخ می دهد که کاربر روند اجرای برنامه را، اغلب با فشردن همزمان کلیدهای Ctrl+c، مختل نماید.
	Base class for all lookup errors.
LookupError	کلاس پایه برای تمامی خطاهای مربوط به lookup. در واقع این خطا هنگامی رخ می دهد که یک کلید یا اندیس مورد استفاده در mapping یا sequence مجاز و معتبر نباشد.
IndexError	زمانی رخ می دهد که اندیس در sequence (رشته ها، لیست ها یا تاپل ها) مورد نظر وجود نداشته و یافت نشود.

KeyError	هنگامی صدا زده می شود که کلید مشخص شده در نوع داده ای dictionary موجود نباشد.
NameError	زمانی رخ می دهد که یک شناسه (identifier) در فضای نامی یا namespace محلی یا سراسری یافت نشود.
UnboundLocalError	زمانی صدا زده می شود که پایتون سعی داشته باشد به یک متغیر محلی در بدنه ی تابع یا متد دسترسی داشته باشد اما این متغیر مقداردهی نشده باشد.
EnvironmentError	کلاس پایه تمامی خطاهایی که خارج از محیط پایتون رخ می دهد.
IOError	زمانی صادر می شود که عملیات ورودی/خروجی با شکست مواجه شود مانند زمانی که دستور print اجرا شده یا تابع open() برای باز کردن فایل فراخوانی شود که اصلا وجود ندارد.
IOError	به هنگام بروز خطاهای مربوط به سیستم عامل رخ می دهد.
SyntaxError	زمانی که خطای دستور نحوی و syntax در کد وجود داشته باشد، رخ می دهد.
IndentationError	کلاس پایه برای تمامی خطاهای ساختار نحوی (سینتکس) مربوطه توگذاری و indentation.
SystemError	زمانی رخ می دهد که مفسر با مشکل داخلی مواجه شود. لازم به ذکر است که به هنگام رخداد این خطا، مفسر پایتون از کد خارج نمی شود.
SystemExit	این خطا به سبب فراخوانده شدن متد sys.exit() صادر می شود. اگر این خطا در کد / با کدنویسی مدیریت نشود، باعث می شود که مفسر از کد برنامه خارج شود.
TypeError	زمانی صادر می شود که یک عملیات یا تابع بر روی آبجکتی از نوع نامربوط اعمال شود.
ValueError	چنانچه عملیات یا تابع درون ساخته ای آرگومانی از نوع صحیح اما با مقدار غلط دریافت کند، خطای جاری رخ می دهد.

RuntimeError	هنگامی صادر می شود که خطا در زمان اجرا رخ داده و به هیچ یک از گروه های تعریف شده از خطا تعلق نداشته باشد. مقدار مربوطه یک رشته ی متنی است که اشاره می کند دقیقا چه مشکلی رخ داده است.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented. خطای جاری زمانی رخ می دهد که متد abstract که بایستی داخل کلاس به ارث برده شده پیاده سازی شده باشد، بدنه ی آن طبق انتظار تعریف و پیاده سازی نشده باشد. به عبارت دیگر، این خطا را متدهای abstract که بدنه آن ها داخل کلاس های مشتق شده پیاده سازی نشده باشد، صدا زده می شود.

Assertion ها (دستورات assert و بررسی صحت شرط) در پایتون

assertion یک تست sanity-check یا تست ساده جهت بررسی معقولانه بودن ادامه ی فرایند تست می باشد. دستورات assert انتظار دارند که نتیجه ی عبارت صحیح باشد و مقدار true را برگرداند و چنانچه شرط برقرار نبود، یک خطا صادر می شود.

جهت درک آسان مفهوم assertion در پایتون، می توان آن را به دستور raise-if (یا به عبارت دقیق تر raise-if-not) تشبیه کرد. در این سناریو یک عبارت تست شده و صحت آن بررسی می شود. اگر نتیجه غلط بوده و خروجی مورد انتظار تولید نشد، یک خطا صادر می شود.

برای اجرای تست های assertion در اپلیکیشن، کافی است از دستورات assert استفاده نمایید. این کلیدواژه از ویرایش 1.5 به بعد زبان پایتون برای برنامه نویس قابل استفاده می باشد.

برنامه نویسان معمولا دستورات assert را در ابتدای یک تابع قرار داده تا معتبر یا مجاز بودن ورودی را بررسی کند و نیز یک دستور assert دیگر جهت بررسی صحت خروجی تابع فراخوانی شده (در انتها و پس از صدا زدن تابع)، درج می کنند.

دستور assert

زمانی که پایتون در حین خواندن و تفسیر کد با دستور assert مواجه می شود، مفسر آن عبارت همراه را بررسی کرده که طبق انتظار باید صحیح باشد. اگر عبارت false بود، پایتون خطای AssertionError را صادر می کند.

نحوه ی استفاده از assert در زیر نمایش داده شده است:

```
assert Expression[, Arguments]
```

اگر نتیجه ی expression یا عبارتی که پس از دستور assert درج می شود false بود (assertion ناموفق بود)، پایتون ArgumentExpression را به عنوان پارامتر برای AssertionError مورد استفاده قرار می دهد. خطاهای AssertionError را می توان به راحتی با استفاده از دستور try-except مدیریت کرد. اما اگر آن را مدیریت نکرده باشید، سبب می شوند برنامه خاتمه یافته و یک traceback (بازتاب مجموعه خطاهای تولید شده که به صورت پشته بر روی هم قرار گرفته اند) تولید شود.

مثال

در زیر تابعی را مشاهده می کنید که دما را از مقیاس کلوین به فارنهایت تبدیل می کند. از آنجایی که 0 درجه کلوین سردترین نقطه است، تابع اگر با مقدار منفی مواجه شود، خطا می دهد.

```
#!/usr/bin/python
def KelvinToFahrenheit(Temperature):
assert (Temperature >= 0),"Colder than absolute zero!"
return ((Temperature-273)*1.8)+32
print KelvinToFahrenheit(273)
print int(KelvinToFahrenheit(505.78))
print KelvinToFahrenheit(-5)
```

کد بالا پس از اجرا، نتیجه ی زیر را بدست می دهد:

```
32.0
451
Traceback (most recent call last):
File "test.py", line 9, in
print KelvinToFahrenheit(-5)
File "test.py", line 4, in KelvinToFahrenheit
```

```
assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError: Colder than absolute zero!
```

Exception (خطای زمان اجرا و پیشبینی نشده) چیست؟

Exception یک رخداد (event) است که در طول اجرا برنامه رخ می دهد و روند اجرای برنامه را مختل می نماید. در کل، زمانی که اسکریپت پایتون با شرایطی که قابلیت مدیریت آن را ندارد، مواجه می شود، یک خطا صادر می کند. exception در پایتون یک آبجکت پایتون است که نماینده ی خطا می باشد.

زمانی که اسکریپت پایتون یک خطا (exception) صادر می کند، یا باید خطا را مدیریت کند و یا برنامه را به طور کلی ببندد.

مدیریت Exception (خطا)

در صورت مواجه با کد مشکوک که ممکن است سبب رخداد خطا شود، می توانید آن را داخل بدنه ی try: قرار دهید. پس از بدنه ی try:، یک دستور except: در متن برنامه درج کرده و به دنبال آن قطعه کدی که مشکل برنامه را به صورت بهینه اداره می کند، تایپ نمایید.

نحوه ی استفاده از دستور

در زیر نحوه ی بکار بردن دستور try ...except ... else را مشاهده می کنید:

```
try:
You do your operations here;
.....................
except ExceptionI:
If there is ExceptionI, then execute this block.
except ExceptionII:
If there is ExceptionII, then execute this block.
.....................
else:
If there is no exception then execute this block.
```

در زیر نکات آموزشی مهمی در خصوص ساختار فوق عنوان شده است:

- یک دستور try واحد می تواند چندین دستور متناظر except داشته باشد. این ویژگی به خصوص زمانی مفید واقع می شود که بدنه ی try حامل دستورات متعددی باشد که ممکن است هر یک خطای متفاوتی را سبب شود.
- می توانید یک عبارت except سراسری (generic) تنظیم کنید که هر خطایی را به راحتی مدیریت نماید.
- می توانید پس از دستور یا دستورات except، عبارت else را مورد استفاده قرار دهید. چنانچه کد موجود در بدنه ی :try خطا نداد، آنگاه قطعه کد else اجرا می شود.
- قطعه کد else جای مناسبی برای تعریف کدهایی می باشد که احتمالا مشکلی برای کل برنامه ایجاد نکرده و اجرای بدون خطای آن حتمی است.

مثال

کد زیر سعی دارد ابتدا یک فایل را باز کند و سپس در آن اطلاعاتی را درج نماید. از آنجایی مجوز در سطح نوشتن و درج داده های جدید داخل فایل اعطا نشده، خطای مربوطه توسط قطعه کد except صادر می شود.

```
#!/usr/bin/python
try:
fh = open("testfile", "r")
fh.write("This is my test file for exception handling!!")
except IOError:
print "Error: can\'t find file or read data"
else:
print "Written content in the file successfully"
```

کد بالا خروجی زیر را ارائه می دهد:

```
Error: can't find file or read data
```

عبارت except بدون مشخص کردن نوع خطا

می توانید از دستور except که هیچ خطا یا exception ای در آن تعریف نشده نیز به صورت زیر استفاده نمایید و در واقع چنانچه هر گونه خطایی وجود داشت، دستور موجود در قطعه کد except را انجام دهد:

```
try:
You do your operations here;
.....
سعی کن یک عملیاتی را در اینجا انجام دهی
except:
If there is any exception, then execute this block.
در صورت برخورد با خطا،  عملیات دیگری را انجام بده
.....
else:
If there is no exception then execute this block.
در غیر این صورت این کار را انجام بده
```

این نوع دستور try-except تمامی خطاهایی که در برنامه رخ می دهد را گرفته و مدیریت می نماید. با این حال استفاده از این تکنیک در برنامه نویسی به هیچ وجه توصیه نمی شود چرا که تمامی خطاها را اداره می کند اما در ریشه یابی خطا و شناسایی اصل مشکل هیچ کمکی به توسعه دهنده نمی کند.

عبارت except با چندین Exception

می توانید با یک دستور except همزمان چندین exception را به صورت زیر مدیریت نمایید:

```
try:
You do your operations here;
عملیاتی را در این قطعه کد امتحان نمایید
.....
except(Exception1[, Exception2[,...ExceptionN]]]):
If there is any exception from the given exception list,
then execute this block.
در صورت مواجه شدن با خطا، کارهای تعریف شده در این بخش را اجرا نمایید
.....
else:
If there is no exception then execute this block.
اگر خطایی نبود، دستورات این بخش را اجرا کن
```

استفاده از ساختمان try-finally جهت مدیریت خطا

می توانید در انتهای قطعه کد try: از finally: نیز استفاده نمایید. دستوری که در بدنه ی finally: قرار می گیرد، صرف نظر اینکه خطایی در قطعه ی try رخ داده یا خیر، به طور قطع اجرا می شود. نحوه ی استفاده از این ساختار در زیر نمایش داده شده است:

```
try:
```

```
You do your operations here;
عملیات را در اینجا انجام دهید
.....
Due to any exception, this may be skipped.
ممکن است به دلیل وجود هر گونه خطایی این کد اجرا نشود
finally:
This would always be executed.
دستوری که حتما اجرا خواهد شد
.....
```

نمی توانید عبارت else و finally را یکجا بکار ببرید.

مثال

```
#!/usr/bin/python
try:
fh = open("testfile", "w")
fh.write("This is my test file for exception handling!!")
finally:
print "Error: can\'t find file or read data"
```

اگر مجوز باز کردن فایل در سطح درج اطلاعات در آن را نداشته باشید، در آن صورت نتیجه ی زیر حاصل می گردد:

```
Error: can't find file or read data
```

مثال فوق را می توان به صورت کارا و مختصرتر نیز، همچون نمونه کد زیر، نوشت:

```
#!/usr/bin/python
try:
fh = open("testfile", "w")
try:
fh.write("This is my test file for exception handling!!")
finally:
print "Going to close the file"
fh.close()
except IOError:
print "Error: can\'t find file or read data"
```

زمانی که سیستم در بدنه ی try با مشکل مواجه شده و خطای مربوطه را صادر کرد، اجرا سریعا به قطعه کد finally انتقال می یابد. پس از اینکه کلیه ی دستورات موجود در بدنه ی finally اجرا شدند، خطای مزبور بار دیگر رخ داده، ولی این بار در لایه ی بالاتر دستور try-except بعدی، به واسطه ی دستورات except مدیریت می شود.

آرگومان ارسال شده به Exception

یک exception می تواند آرگومان (argument) نیز داشته باشد. آرگومان پاس داده شده به exception، اطلاعات بیشتری را درباره ی خطای رخ داده ارائه می دهد. محتوای آرگومان می تواند با توجه به نوع خطا متفاوت باشد. می توانید با لحاظ کردن یک متغیر ساده در عبارت except، مانند زیر، آرگومانی را به exception ارسال نمایید:

```
try:
You do your operations here;
.....................
except ExceptionType, Argument:
You can print value of Argument here...
```

اگر کدی بنویسید که تنها یک خطا (exception) را مدیریت کند، در آن صورت کافی است یک متغیر پس از اسم خطا، در دستور except، لحاظ نمایید. چنانچه لازم است چندین خطا را گیر انداخته و مدیریت نمایید، در آن صورت می توانید متغیری پس از چندتایی exception خطا (متغیر exception از نوع چندتایی یا tuple) لحاظ نمایید.

این متغیر مقدار خطا که یک رشته باشد را گرفته و در واقع دلیل بروز خطا را در خود ذخیره می کند. این متغیر می تواند یک مقدار را دریافت کند یا همزمان چندین مقدار را در قالب یک چندتایی (tuple) در خود نگه دارد. این متغیر چندتایی (از نوع tuple) اغلب حامل رشته ی متنی خطا، شماره ی خطا و محل رخداد آن می باشد.

مثال

مثال زیر تنها یک خطا را مدیریت می کند.

```
#!/usr/bin/python
# Define a function here.
def temp_convert(var):
try:
return int(var)
except ValueError, Argument:
print "The argument does not contain numbers\n", Argument
# Call above function here.
temp_convert("xyz");
```

نتیجه ی زیر را بدست می دهد:

```
The argument does not contain numbers
```

```
invalid literal for int() with base 10: 'xyz'
```

تعریف و مدیریت خطا با استفاده از دستور raise

می توانید با استفاده از دستور raise خطاهای متعددی را مدیریت کنید. سینتکس و ساختار نحوی کلی برای مدیریت خطا با استفاده از دستور raise در زیر نمایش داده شده است.

ساختارکلی و نحوه ی استفاده از raise

```
raise [Exception [, args [, traceback]]]
```

در اینجا، پارامتر Exception در واقع نوع خطا (برای مثال NameError) و پارامتر argument یک مقدار است که به آرگومان exception یا خطا ارسال می شود. پارامتر argument کاملا اختیاری است و در صورت عدم ارسال مقداری به آن، آرگومان مزبور برابر None خواهد بود.

آخرین آرگومان، traceback، نیز اختیاری است و در واقع به ندرت مورد استفاده قرار می گیرد. این آرگومان پشته ای از خطاها و یک آبجکت traceback هست که مربوط به خطای مورد نظر می باشد.

مثال

exception می تواند یک رشته، کلاس و یا آبجکت باشد. اغلب خطاهایی که Python core صادر می کند از جنس کلاس هستند و یک آرگومان نیز دارند که نمونه ای از آن کلاس می باشد. تعریف خطاهای جدید امری بسیار آسان است و به راحتی زیر قابل پیاده سازی می باشد:

```
def functionName( level ):
   if level < 1:
      raise "Invalid level!", level
      # The code below to this would not be executed
      # if we raise the exception
```

نکته: جهت گرفتن و مدیریت خطا، عبارت "except" بایستی به همان خطایی که آبجکت، کلاس یا رشته ی ساده صادر کرده، اشاره کند. برای مثال، جهت گرفتن و ضبط خطای فوق، لازم است دستور except را به صورت زیر تنظیم نمایید:

```
try:
   Business Logic here...
except "Invalid level!":
   Exception handling here...
else:
```

```
Rest of the code here...
```

خطاهای اختصاصی و user-defined

پایتون به توسعه دهنده این امکان را می دهد تا خطاهای دلخواه خود را تعریف کند. برای این منظور می بایست از کلاس های درون ساخته ی exception ارث بری نماید.

در زیر مثالی را مشاهده می کنید که در آن یک خطای زمان اجرا و مربوط به RuntimeError رخ داده است. در مثال حاضر، یک کلاس تعریف شده که این کلاس خود از کلاس پایه ی RuntimeError مشتق (subclass) شده است. چنانچه لازم است اطلاعات بیشتری در خصوص خطای رخ داده برای کاربر نمایش دهید، توصیه می شود از این روش استفاده نمایید.

در بدنه ی دستور try: خطاهای اختصاصی (user-defined exception) تعریف (raise) شده و سپس این خطا داخل ساختمان except ضبط و مدیریت (catch) می شود. متغیر e در واقع یک آبجکت یا نمونه ای ساخته شده از روی کلاس Networkerror می باشد.

```
class Networkerror(RuntimeError):
def __init__(self, arg):
self.args = arg
```

زمانی که کلاس فوق را تعریف کردید، آنگاه می توانید خطا را به صورت زیر اعلان و پیاده سازی کنید:

```
try:
raise Networkerror("Bad hostname")
except Networkerror,e:
print e.args
```

شی گرایی در زبان پایتون / Python Object-Oriented

Python از زمان طراحی و انتشار همگانی یک زبان شی گرا بوده است. از این رو ایجاد و استفاده از کلاس ها و آبجکت ها بسیار آسان می باشد. فصل حاضر اصول شی گرا و استفاده از این قابلیت زبان در توسعه ی آسان اپلیکیشن را به شما آموزش می دهد.

چنانچه از قبل با تکنیک برنامه نویسی شی گرا آشنایی ندارید، توصیه می شود ابتدا یک دوره ی آموزشی مقدماتی در خصوص اصول آن گذرانده و یا حداقل یک آموزش ساده جهت آشنایی با تکنیک های آن مطالعه کنید تا در درک مفاهیم این درس با مشکل مواجه نشوید.

با این وجود، در زیر مقدمه ای مختصر درباره ی مفاهیم و اصول OOP (اصول برنامه نویسی و تکنیک شی گرا) در اختیار شما قرار می گیرد.

مروری بر واژگان و اصطلاحات تخصصی OOP

- Class (کلاس): مجموعه ای از آبجکت ها که ویژگی های یکسان دارند یک کلاس را می سازند. به عبارت دیگر یک نمونه ی اختصاصی یا user-defined که اجازه ی ساخت آبجکت یا نمونه های متعدد از یک الگو مشترک را می دهد. هر کلاس واحدی مجزا از اپلیکیشن است که داده هایی را در قالب متغیرهای عضو که attribute نام داشته و عملیات که در قالب method تعریف می شود را دربرمی گیرد. همان طور که گفته شد، attribute ها اعضایی از کلاس هستند که داده های را در خود نگه می دارند (class variable و instance variable) و متدها نیز رفتارهای کلاس بوده و عملیات خاصی را انجام می دهند. جهت دسترسی به متدها و متغیرهای عضو کلاس کافی است از عملگر نقطه استفاده نمایید.
- class variable (متغیرهای عضو کلاس): متغیری که بین تمامی نمونه های ایجاد شده از کلاس مشترک می باشد، متغیر عضو کلاس یا class variable خوانده می شود. متغیرهای عضو داخل بدنه ی کلاس اما خارج از حوزه ی اختصاصی متدهای آن کلاس (scope) تعریف می شوند. Class variable یا متغیرهای عضو کلاس نسبت به متغیرهای نمونه ی ایجاد شده از کلاس یا همان instance variable بسیار کم تر مورد استفاده قرار می گیرد.
- Data member: یک class variable یا instance variable که داده های مربوط به یک کلاس و آبجکت های ساخته شده از روی آن را دربرمی گیرد.
- function overloading: تخصیص و تعریف چندین رفتار برای تابعی یکسان. به عبارت دیگر، ایجاد چندین نسخه ی مختلف از تابعی با نام یکسان. حال این عملیاتی که تابع انجام می دهد به نوع آبجکت ها یا آرگومان هایی که شرکت دارند، وابسته است.
- instance variable: یک متغیر که داخل حوزه ی اختصاصی یا بدنه ی متد تعریف شده و تنها به نمونه ی جاری از کلاس تعلق دارد.
- Inheritance (وراثت): انتقال ویژگی های یک کلاس به کلاس هایی که از آن مشتق می شوند را در اصطلاح Inheritance یا وراثت می گویند.

- instance: به یک آبجکت واحد از کلاس در اصطلاح Instance گفته می شود. برای مثال، آبجکتی به نام obj که متعلق به کلاسی به نام Circle می باشد، در واقع یک نمونه (ساخته شده از) کلاس Circle (که الگویی برای ساخت آبجکت ها می باشد) است.
- Instantiation (نمونه سازی از کلاس): به ساخت آبجکت یا نمونه ای از کلاس مورد نظر Instantiation گویند.
- method: همان تابع که داخل بدنه یا حوزه ی اختصاصی کلاس اعلان می شود.
- Object: عبارت است از یک نمونه ی یکتا که از روی کلاس مورد نظر ساخته می شود. یک آبجکت می تواند علاوه بر data member ها (متغیرهای عضو کلاس و متغیرهای عضو نمونه ی ایجاد شده از کلاس)، متد نیز دربرداشته باشد.
- Operator overloading: تخصیص چندین رفتار به عملگری یکسان. به عبارت دقیق تر، یک نمونه ی خاص از چندریختی است که عملگرها می توانند بر اساس آرگومان ها و پارامترهای ورودی پیاده سازی متفاوتی داشته باشد.

تعریف کلاس

دستور class همان طور که از اسم آن مشخص است، یک کلاس جدید تعریف می کند. اسم کلاس بلافاصله پس از کلیدواژه ی class و عملگر دو نقطه درج می شود:

```
class ClassName:
'Optional class documentation string'
class_suite
```

کلاس، داخل ساختمان خود یک docstring (یک رشته ی ثابت که در کد برنامه لحاظ شده و عملکردی مشابه comment دارد و برای توضیح هدف اصلی کد مورد استفاده قرار می گیرد ، اما بر خلاف comment معمولی، در زمان اجرای برنامه همچنان باقی مانده و برای کاربر نمایش داده می شود.) دارد که با درج دستور ClassName.__doc__ در دسترس توسعه دهنده قرار می گیرد.

class_suite دربردارنده ی تمامی دستورات تشکیل دهنده ی کلاس از جمله دستورات تعریف اعضای کلاس، attribute ها و توابع می باشد.

مثال

در زیر نمونه ای از کلاس ساده ی پایتون را مشاهده می کنید:

```
class Employee:
'Common base class for all employees'
empCount = 0
def __init__(self, name, salary):
self.name = name
self.salary = salary
Employee.empCount += 1
def displayCount(self):
print "Total Employee %d" % Employee.empCount
def displayEmployee(self):
print "Name : ", self.name,  ", Salary: ", self.salary
```

- متغیر empCount یک class variable است که مقدار آن بین تمامی نمونه های ایجاد شده از کلاس جاری مشترک می باشد. جهت دسترسی به این متغیر، چه از داخل کلاس یا خارج از آن، کافی است از دستور Employee.empCount استفاده نمایید.
- اولین متد __init__()، تابع سازنده یا متد مقداردهنده ی اولیه (constructor یا initialization method) است که پایتون آن را به هنگام ساخته شدن نمونه ی جدید از کلاس، فراخوانی می کند.
- سایر متدهای کلاس مانند توابع عادی تعریف می شوند با این تفاوت که اولین آرگومان ارسالی به متد پارامتر self می باشد. Python به صورت خودکار آرگومان مزبور را به لیست اضافه می کند و نیازی نیست که توسعه دهنده به هنگام فراخوانی متد آن را به عنوان پارامتر به طور صریح لحاظ کند.

ایجاد آبجکت های نمونه (ساخت آّبجکت یا نمونه از روی کلاس)

به منظور ایجاد نمونه هایی از یک کلاس، توسعه دهنده آن را با استفاده از اسم کلاس فراخوانی کرده و سپس پارامترهای مورد نظر را به تابع سازنده یا constructor ارسال نمایید.

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

دسترسی به attribute ها () کلاس

جهت دسترسی به attribute های یک آبجکت، کافی است از اسم آبجکت و عملگر نقطه استفاده نمایید. class variable (متغیر عضو کلاس) به صورت زیر، با استفاده از اسم کلاس و عملگر نقطه قابل دسترسی می باشد:

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

حال تمامی مفاهیم را با هم در یک مثال به صورت کاربردی مورد استفاده قرار می دهیم:

```
#!/usr/bin/python

class Employee:
'Common base class for all employees'
empCount = 0
def __init__(self, name, salary):
self.name = name
self.salary = salary
Employee.empCount += 1
def displayCount(self):
print "Total Employee %d" % Employee.empCount
def displayEmployee(self):
print "Name : ", self.name,  ", Salary: ", self.salary
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

کد فوق پس از اجرا، نتیجه ی زیر را بدست می دهد:

```
Name :  Zara ,Salary:  2000
Name :  Manni ,Salary:  5000
Total Employee 2
```

توسعه دهنده می تواند attribute های کلاس ها و آبجکت ها را هر زمان که لازم دانست ویرایش (اضافه، حذف و غیره ...) کند:

```
emp1.age = 7  # Add an 'age' attribute.
emp1.age = 8  # Modify 'age' attribute.
del emp1.age  # Delete 'age' attribute.
```

می توان علاوه بر روش معمول دسترسی به attribute ها، از توابع زیر برای فراخوانی و دستیابی به attribute مورد نظر استفاده کرد:

- متد getattr(obj, name[, default]): جهت دسترسی به متغیر عضو یا attribute از آبجکت مورد نظر.
- hasattr(obj,name): جهت بررسی صحت وجود یک attribute در آبجکت مد نظر.
- متد setattr(obj,name,value): به منظور مقداردهی و تنظیم یک attribute مورد استفاده قرار می گیرد. اگر attribute مورد نظر وجود نداشت، متد نام برده آن را ایجاد کرده و مقداردهی می کند.
- متد delattr(obj, name): به منظور حذف attribute فراخوانی می شود.

```
hasattr(emp1, 'age')    # Returns true if 'age' attribute exists
getattr(emp1, 'age')    # Returns value of 'age' attribute
setattr(emp1, 'age', 8) # Set attribute 'age' at 8
delattr(empl, 'age')    # Delete attribute 'age'
```

Attribute های درون ساخته ی کلاس

کلاس های پایتون attribute های درون ساخته و پیش فرضی دارند که به وسیله ی عملگر نقطه به راحتی برای توسعه دهنده قابل دسترسی می باشند.

این attribute ها عبارتند از:

- __dict__: یک attribute از نوع داده ای dictionary که دربردارنده ی namespace کلاس می باشد.
- __doc__: رشته ی مستندسازی و درج توضیحات در کلاس / documentation string یا در صورتی که تعریف نشده باشد، None.
- __name__: اسم کلاس.

- __module__: اسم ماژولی که در آن کلاس تعریف شده است. در حالت تعاملی یا interactive mode، این attribute "__main__" است. (interactive mode: یک امکان مبتنی بر خط دستور است که قابلیت اجرا و پردازش کدهای پایتون را فراهم می آورد. زمانی که اسکریپت می نویسید با فشردن کلید enter برنامه ی مفسر آن را به صورت خودکار اجرا می کند.)
- __bases__: یک متغیر چندتایی (از نوع tuple) تهی که دربردارنده ی کلاس های پایه، به ترتیبی که در لیست کلاس های پایه قید شده است، می باشد.

با کد زیر سعی می کنیم به تمامی attribute های نام برده در کلاس حاضر دسترسی پیدا کنیم:

```
#!/usr/bin/python
class Employee:
'Common base class for all employees'
empCount = 0
def __init__(self, name, salary):
self.name = name
self.salary = salary
Employee.empCount += 1
def displayCount(self):
print "Total Employee %d" % Employee.empCount
def displayEmployee(self):
print "Name : ", self.name,  ", Salary: ", self.salary
print "Employee.__doc__:", Employee.__doc__
print "Employee.__name__:", Employee.__name__
print "Employee.__module__:", Employee.__module__
print "Employee.__bases__:", Employee.__bases__
print "Employee.__dict__:", Employee.__dict__
```

کد فوق پس از اجرا خروجی زیر را بدست می دهد:

```
Employee.__doc__: Common base class for all employees
Employee.__name__: Employee
Employee.__module__: __main__
Employee.__bases__: ()
Employee.__dict__: {'__module__': '__main__', 'displayCount':
<function displayCount at 0xb7c84994>, 'empCount': 2,
'displayEmployee': <function displayEmployee at 0xb7c8441c>,
'__doc__': 'Common base class for all employees',
'__init__': <function __init__ at 0xb7c846bc>}
```

حذف آبجکت های غیرضروری از حافظه (مدیریت حافظه یا Garbage collection)

پایتون تمامی آبجکت های بلااستفاده را به صورت خودکار، جهت آزاد سازی منابع، از حافظه پاک می کند. فرایندی که پایتون به واسطه ی آن در فواصل زمانی معین قطعاتی از حافظه را که دیگر مورد استفاده نیستند بازپس گرفته و آبجکت های ذخیره شده در آن را پاک می کند، در اصطلاح garbage collection خوانده می شود.

Garbage collector پایتون به هنگام اجرای برنامه فعال می گردد و زمانی که reference count (تعداد دفعات ارجاع به آبجکت و استفاده از آن) به صفر می رسد، اجرا شده و آبجکت های بلااستفاده را از حافظه حذف می کند. لازم به ذکر است زمانی که نام های مستعاری (aliases) که به یک آبجکت اشاره دارند، تغییر می کنند، reference count یک آبجکت نیز به تبع تغییر می نماید.

Reference count یک آبجکت هنگامی افزایش می یابد که اسم جدیدی به آن آبجکت تخصیص یافته یا داخل یک ظرف (container) همچون list، tuple یا dictionary قرار داده شود. زمانی که آبجکت مورد نظر با کلیدواژه ی del حذف می شود یا reference اشاره کننده به آن به آبجکت دیگری تخصیص می یابد و یا از حوزه (scope) خارج می شود، reference count آبجکت کاهش می یابد.

همچنین زمانی که reference count یک آبجکت به صفر می رسد، پایتون آن را به صورت خودکار collect کرده و از حافظه حذف می نماید.

```
a = 40      # Create object <40>
b = a       # Increase ref. count  of <40>
c = [b]     # Increase ref. count  of <40>
del a       # Decrease ref. count  of <40>
b = 100     # Decrease ref. count  of <40>
c[0] = -1   # Decrease ref. count  of <40>
```

اغلب زمانی که garbage collector یک نمونه ی بلااستفاده (orphaned instance و سرگردان) را حذف کرده و منابع اختصاص یافته به آن را آزاد می سازد، شما متوجه این عملیات پس زمینه ای نمی شوید. یک کلاس همچنین می تواند متد __del__() را پیاده سازی کند که در اصطلاح destructor

(حذف کننده ی نمونه ی کلاس از حافظه) خوانده می شود. این متد به هنگام حذف نمونه ی مورد نظر از حافظه صدا زده شده و آن نمونه ی بلااستفاده یا سرگردان را از حافظه پاک می نماید.

مثال

تابع __del__() اسم کلاسی که نمونه ی مورد نظر از روی آن ساخته شده را به هنگام حذف آبجکت از حافظه، در نمایشگر چاپ می کند.

```
#!/usr/bin/python
class Point:
def __init( self, x=0, y=0):
self.x = x
self.y = y
def __del__(self):
class_name = self.__class__.__name__
print class_name, "destroyed"
pt1 = Point()
pt2 = pt1
pt3 = pt1
print id(pt1), id(pt2), id(pt3) # prints the ids of the obejcts
del pt1
del pt2
del pt3
```

کد فوق پس از اجرا خروجی زیر را تولید می کند:

```
3083401324 3083401324 3083401324
Point destroyed
```

نکته: بهتر است که کلاس های خود را داخل فایل مجزا تعریف نموده، سپس آن ها را با استفاده از دستور import وارد متن برنامه ی اصلی (main program) نمایید.

مبحث وراثت و class inheritance

می توانید بجای اینکه کلاسی را از از پایه ایجاد کنید، آن را از یک کلاس آماده و از پیش موجود به راحتی مشتق نمایید. برای این منظور کافی است اسم کلاس پدر (parent) را داخل پرانتز بعد از اسم کلاس مورد نظر درج کنید.

کلاس مشتق (child) attribute ها و ویژگی های کلاس پدر/پایه (parent) را به ارث برده و شما می توانید به آن ها به سادگی دسترسی داشته باشید گویا این attribute ها داخل خود کلاس فرزنده تعریف شده اند. کلاس مشتق/فرزند سپس این قابلیت را دارد تا اعضا (متغیرهای عضو کلاس و متدها) را به طور دلخواه بازنویسی (override) نماید.

ساختار دستوری و سینتکس

کلاس های مشتق شبیه به کلاس های پایه یا پدر خود تعریف می شوند، با این تفاوت که لیستی از اسم کلاس های پایه (که از آن ارث بری صورت گرفته)، پس از اسم کلاس جدید درج می شود:

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
'Optional class documentation string'
class_suite
```

مثال

```
#!/usr/bin/python
class Parent:        # define parent class
parentAttr = 100
def __init__(self):
print "Calling parent constructor"
def parentMethod(self):
print 'Calling parent method'
def setAttr(self, attr):
Parent.parentAttr = attr
def getAttr(self):
print "Parent attribute :", Parent.parentAttr
class Child(Parent): # define child class
def __init__(self):
print "Calling child constructor"
def childMethod(self):
print 'Calling child method'
c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method
```

کد فوق پس از اجرا نتیجه ی زیر را برمی گرداند:

```
Calling child constructor
Calling child method
Calling parent method
Parent attribute : 200
```

به همین روال می توان یک کلاس جدید را همزمان از چندین کلاس پدر مشتق نمود:

```
class A:        # define your class A
.....
class B:        # define your class B
.....
class C(A, B):  # subclass of A and B
.....
```

می توان با فراخوانی توابع issubclass() یا isinstance() رابطه ی بین دو کلاس و نمونه را بررسی کرد.

- چنانچه پارامتر sub واقعا کلاسی مشتق شده از پارامتر sup باشد، تابع بولی issubclass(sub, sup) مقدار true را برمی گرداند.
- چنانچه پارامتر obj واقعا نمونه ای از پارامتر Class باشد، آنگاه تابع بولی isinstance(obj, Class) مقدار true را در خروجی بازگردانی می نماید.

بازنویسی متدها (overriding)

توسعه دهنده این امکان را دارد که توابع ارث برده شده از کلاس پدر را بازنویسی نمایند. یکی از دلایل بازنویسی متد کلاس پدر می تواند نیاز به افزودن قابلیت جدید به تابع مورد نظر در بدنه ی کلاس فرزند و مشتق شده باشد.

مثال

```
#!/usr/bin/python
class Parent:        # define parent class
    def myMethod(self):
        print 'Calling parent method'
class Child(Parent): # define child class
    def myMethod(self):
        print 'Calling child method'
c = Child()          # instance of child
```

```
c.myMethod()        # child calls overridden method
```

کد فوق پس از اجرا نتیجه ی زیر را در خروجی تولید می کند:

```
Calling child method
```

معرفی متدهایی جهت بازنویسی

جدول زیر تعدادی توابع با قابلیت های کلی در اختیار شما قرار می دهد که می توانید آن ها را داخل بدنه ی کلاس های خود بر اساس نیاز بازنویسی نمایید:

متد، شرح کاربرد و مثالی کاربردی از فراخوانی آن	شماره ی مثال
__init__ (self [,args...]) Constructor (آرگومان های اختیاری) Sample Call : *obj = className(args)*	1
__del__(self) Destructor, یک آبجکت را از حافظه حذف می کند Sample Call : *del obj*	2
__repr__(self) Evaluatable string representation Sample Call : *repr(obj)*	3
__str__(self) Printable string representation Sample Call : *str(obj)*	4
__cmp__ (self, x) Object comparison Sample Call : *cmp(obj, x)*	5

Operator overloading

فرض کنید کلاس جدیدی به نام Vector ایجاد کرده اید که نشانگر vector های دو بعدی می باشد. اگر بخواهیم آن ها را با عملگر + با یکدیگر جمع کنیم، چه رخ می دهد؟ پایتون قطعا واکنش نشان داده و خطا می گیرد.

برای رفع این مشکل می توانید یک متد __add__ داخل کلاس مورد نظر تعریف کنید که عمل جمع دو vector را انجام دهد. با این کار عملگر مزبور طبق انتظار عمل خواهد کرد:

مثال

```
#!/usr/bin/python
class Vector:
def __init__(self, a, b):
self.a = a
self.b = b
def __str__(self):
return 'Vector (%d, %d)' % (self.a, self.b)
def __add__(self,other):
return Vector(self.a + other.a, self.b + other.b)
v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

کد فوق پس از اجرا خروجی زیر را بدست می دهد:

```
Vector(7,8)
```

از دسترس خارج ساختن attribute های کلاس (Data hiding)

attribute های یک آبجکت ممکن است برای المان ها یا موجودیت های خارج از کلاس میزبان قابل دسترسی باشند. برای اینکه این attribute ها تنها از داخل قابل دسترسی بوده و منحصرا برای اعضای کلاس جاری visible باشند، کافی است پیشوند دو خط تیره پشت سرهم "__" را قبل از attribute بکار ببرید.

مثال

```
#!/usr/bin/python
class JustCounter:
__secretCount = 0
def count(self):
self.__secretCount += 1
print self.__secretCount
counter = JustCounter()
counter.count()
counter.count()
print counter.__secretCount
```

کد فوق پس از اجرا خروجی زیر را برمی گرداند:

```
1
2
Traceback (most recent call last):
File "test.py", line 12, in <module>
print counter.__secretCount
AttributeError: JustCounter instance has no attribute '__secretCount'
```

پایتون این اعضا را خود با افزودن اسم کلاس به اسم متغیر عضو محافظت می نماید. حال به منظور دسترسی به attribute مورد نظر از کلاس، لازم است از فرمول object._className__attrName استفاده نمایید. با ویرایش کد به صورت زیر، می توانید این قابلیت را خود به راحتی امتحان کنید:

```
.....
print counter._JustCounter__secretCount
```

کد فوق، خروجی زیر را برمی گرداند:

```
1
2
2
```

عبارات باقاعده/Regular expression در پایتون (قابلیت تطبیق و یافتن رشته در متن)

Regular expression در واقع دنباله ای از کاراکترها است که به توسعه دهنده کمک می کند تا رشته ی مورد نظر را بر اساس گرامر و سینتکس اختصاصی که در قالب یک الگو تعریف می شود،

پیدا کند. به عبارت دیگر regular expression ها تکنیک ها و قوانینی هستند که جهت استخراج و بررسی صحت وجود دنباله ی خاصی از مقادیر (عدد یا متن) بکار می روند. این قوانین بر اساس تنظیمات مشخص شده، بخش به خصوصی از یک متن یا عدد را بیرون می کشند. Regular expression ها در دنیای UNIX کاربرد فراوانی دارد.

ماژول re به توسعه دهنده این امکان را می دهد تا از عبارات باقاعده مشابه آنچه در زبان Perl پیاده سازی می شود، در پروژه ی خود استفاده کند. در صورت رخداد خطا به هنگام استفاده یا کامپایل عبارت باقاعده در اپلیکیشن، ماژول نام برده خطای re.error را صادر می کند.

پیش از تشریح دو تابع بسیار کاربردی که با استفاده از آن ها می توان عبارات باقاعده را مدیریت کرد، لازم است نکته ی کوچکی را شرح دهیم. کاراکترهای متعددی وجود دارد که به هنگام استفاده در عبارات باقاعده معنا و کاربرد خاصی دارند. جهت اجتناب از سردرگمی در استفاده از عبارات باقاعده، توصیه می شود از Raw String استفاده نمایید: r'expression'.

'r' قبل از عبارت مورد نظر، به پایتون اعلان می کند که رشته ی مورد نظر یک raw string یا رشته ی خام است. در یک رشته ی خام، کاراکترهای گریز (escape sequence) پردازش و تفسیر نمی شوند. به طور مثال، '\n' منحصرا یک کاراکتر newline است و مفهوم دیگری ندارد. اما r'\n' در پایتون دو کاراکتر در نظر گرفته می شود: یک backslash و نیز یک 'n' هر کدام به صورت مجزا.

تابع match

این تابع سعی می کند پارامتر pattern و string را با یکدیگر تطبیق دهد (پارامتر flags اختیاری می باشد).

در زیر دستور استفاده از این تابع را مشاهده می کنید:

```
re.match(pattern, string, flags=0)
```

جدول زیر پارامترهای ورودی این تابع را شرح می دهد:

پارامتر	شرح مورد کاربرد

پارامتر حاضر همان عبارت باقاعده یا الگویی است که باید تطبیق داده شود. در واقع توسعه دهنده سعی دارد مورد منطبق با این پارامتر را در متن پیدا کند.	pattern
این پارامتر رشته ای است که پارامتر فوق با بخش آغازین (اول) آن تطبیق داده می شود. در واقع تابع سعی دارد تا پارامتر اول را در پارامتر دوم که آن هم رشته است پیدا کند.	string
می توانید با استفاده از عملگر بیتی (|) OR، flag های مختلف تعریف نمایید. این flag ها، modifier هایی هستند که در جدول زیر شرح داده می شوند.	flags

تابع re.match در صورت یافتن مورد منطبق، در خروجی آبجکت match برمی گردانده و چنانچه مورد منطبقی یافت نشد، None مقدار خروجی خواهد بود. جهت یافتن و بازیابی عبارت منطبق در متن، توابع group(num) یا groups() از آبجکت match را فراخوانی می کنیم.

متدهای بررسی و یافتن مورد منطبق	شرح کاربرد
group(num=0)	این متد کل عبارت منطبق (match) را در خروجی برمی گرداند (یا یک زیرگروه معین که با توجه به پارامتر num مشخص می شود).
groups()	تمامی زیرگروه های منطبق موجود در یک چندتایی یا tuple را در خروجی برمی گرداند (در صورت عدم وجود مورد منطبق empty برمی گرداند).

مثال

```
#!/usr/bin/python
```

```
import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
print "matchObj.group() : ", matchObj.group()
print "matchObj.group(1) : ", matchObj.group(1)
print "matchObj.group(2) : ", matchObj.group(2)
else:
print "No match!!"
```

پس از اجرا گرفتن از کد فوق، خروجی زیر در نمایشگر درج می شود:

```
matchObj.group() :  Cats are smarter than dogs
matchObj.group(1) :  Cats
matchObj.group(2) :  smarter
```

تابع search

این تابع اولین نمونه از مقدار پارامتر pattern که مورد منطبق با آن را داخل پارامتر string می یابد، به عنوان خروجی بازگردانی می نماید.

دستور نحوی استفاده از این تابع در زیر شرح داده شده است:

```
re.search(pattern, string, flags=0)
```

جدول زیر پارامترهای این متد را همراه با شرح کاربرد هریک شرح می دهد:

پارامتر	شرح
pattern	پارامتر جاری همان عبارت باقاعده ای است که با پارامتر دوم تطبیق داده می شود. در واقع این پارامتر با تمامی بخش های پارامتر دوم تطبیق داده می شود تا مورد منطبق یافت شود.
string	این پارامتر رشته ای است که تمامی بخش های آن با پارامتر اول تطبیق داده شده تا مورد منطبق یافت شود. در واقع متد مورد نظر در تمامی بخش های رشته ی دوم جستجو کرده و مورد منطبق را در خروجی بازیابی می کند.
flags	می توانید به وسیله ی (|) OR، عملگر بیتی، flag های متفاوت تعریف کنید.

در صورت یافتن مورد منطبق، تابع re.search یک آبجکت match در خروجی برمی گرداند و در غیر این صورت None را بازگردانی می نماید. با استفاده از توابع group(num) یا groups() می توان عبارات منطبق در رشته ی مورد نظر را استخراج و بازیابی نمود.

متدهای بررسی و یافتن مورد منطبق	شرح کاربرد
group(num=0)	این متد کل مورد منطبق (match) را در خروجی برمی گرداند (یا یک زیرگروه معین که بر اساس num مشخص می شود).
groups()	این متد تمامی زیرگروه های منطبق موجود در یک چندتایی یا tuple را در خروجی بازگردانی می نماید (در صورت نیافتن مورد منطبق empty را در خروجی بازگردانی می کند).

مثال

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs";
searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)
if searchObj:
print "searchObj.group() : ", searchObj.group()
print "searchObj.group(1) : ", searchObj.group(1)
print "searchObj.group(2) : ", searchObj.group(2)
else:
print "Nothing found!!"
```

نتیجه ی زیر را بدست می دهد:

```
matchObj.group() :  Cats are smarter than dogs
matchObj.group(1) :  Cats
matchObj.group(2) :  smarter
```

مقایسه ی دو متد Match و Search

پایتون جهت یافتن و استخراج مورد منطبق (الگو) در متن مورد نظر، دو عملیات پایه مبتنی بر عبارات باقاعده ارائه می دهد: 1. تطبیق الگو یا عبارت باقاعده با بخش اول پارامتر دوم (string) که توسط تابع match قابل پیاده سازی می باشد 2. تطبیق عبارات باقاعده و سعی بر یافتن مورد منطبق در کل پارامتر دوم (string) که توسط تابع search انجام می شود (زبان Perl در حالت پیش فرض عبارت باقاعده را با تمامی بخش های رشته تطبیق می دهد و در تمامی بخش های پارامتر دوم به دنبال مورد منطبق می گردد).

مثال

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs";
matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
print "match --> matchObj.group() : ", matchObj.group()
else:
print "No match!!"
searchObj = re.search( r'dogs', line, re.M|re.I)
if searchObj:
print "search --> searchObj.group() : ", searchObj.group()
else:
print "Nothing found!!"
```

کد فوق پس از اجرا خروجی زیر را به دست می دهد:

```
No match!!
search --> matchObj.group() :  dogs
```

یافتن و جایگزینی مقدار در متن (search&replace)

یکی از مهم ترین متدهای ماژول re که عبارات باقاعده را به عنوان ورودی می گیرد، تابع sub می باشد.

دستور استفاده از متد

```
re.sub(pattern, repl, string, max=0)
```

این متد تمامی موارد منطبق با پارامتر pattern را با مقدار پارامتر repl جایگزین می کند. لازم به ذکر است که این متد تمامی موارد منطبق را جایگزین می کند مگر اینکه با مقداردهی پارامتر max بر آن محدودیت اعمال نمایید. خروجی تابع حاضر رشته ی ویرایش شده می باشد.

مثال

```
#!/usr/bin/python
import re
phone = "2004-959-559 # This is Phone Number"
# Delete Python-style comments
num = re.sub(r'#.*$', "", phone)
print "Phone Num : ", num
# Remove anything other than digits
num = re.sub(r'\D', "", phone)
print "Phone Num : ", num
```

کد حاضر خروجی زیر را برمی گرداند:

```
Phone Num :  2004-959-559
Phone Num :  2004959559
```

تنظیم و ویرایش عبارات باقاعده با استفاده از flag های اختیاری (Regular expression modifier)

literal ها یا مقادیر رشته ای محصور در تک کوتیشن که regular expression ها هستند را می توان به واسطه ی یک پارامتر اختیاری (optional flag/modifier) مطابق نیاز تنظیم کرده و عملیات انطباق و نیز استخراج مقدار مورد نظر از متن را به صورت اختصاصی انجام داد. modifier ها که به منظور تنظیم اختصاصی عملیات تطبیق و استخراج مقدار از آن استفاده می کنیم، در قالب پارامترهای اختیاری (optional flag) به متد مربوطه ارسال می شوند.

می توانید با استفاده از عملگر بیتی OR (|) چندین Modifier جهت تنظیم اختصاصی عملیات تطبیق تعریف نمایید.

تنظیم کننده ی عملیات انطباق / Modifier	شرح کاربرد
re.I	عملیات تطبیق و استخراج مقدار مورد نظر از متن را بدون حساسیت نشان دادن به کوچک و بزرگی حروف انجام می دهد(case insensitive-matching).

re.L	کلمات و واژگان را بر اساس زبان محلی (locale) تفسیر می کند.
re.M	سبب می شود $ با انتهای هر خطی (نه منحصرا یک رشته) تطبیق داده شده و نیز سبب می شود ^ با ابتدای هر خطی (نه صرفا رشته) تطبیق داده شود.
re.S	سبب می شود یک نقطه با هر کاراکتری، حتی کاراکتر newline قابل انطباق باشد.
re.U	حروف را بر اساس مجموعه کاراکترهای Unicode تفسیر می کند. این پارامتر رفتار \w، \W، \b و \B را تحت تاثیر قرار می دهد.
re.X	امکان تنظیم عبارات با قاعده ی خواناتری را فراهم می آورد. کاراکتر space را نادیده گرفته (مگر اینکه داخل [] بوده یا قبل از آن کاراکتر گریز \ درج شود)، همچنین کاراکتر # که قبل از آن هیچ کارکتر گریزی قرار نمی گیرد را به عنوان نشانگر comment و توضحیات در نظر می گیرد.

الگوها و مجموعه کاراکترهایی که جهت تطبیق در عبارات باقاعده بکار می روند (Regular Expression Patterns)

به استثنای کاراکترهای کنترلی، (\ | { } [ ] ( ) $ ^ * . ? +)، تمامی کاراکترها در فرایند تطبیق با خودشان منطبق اعلام می شوند. شما می توانید با درج کاراکتر backslash قبل از کاراکترهای کنترلی آن ها را نیز escape کرده و امکان تطبیق با خودشان را فراهم آورید.

جدول زیر تمامی دستورات و الگوهایی که در پایتون جهت انطباق و استخراج موارد منطبق استفاده می شود را همراه با شرح کاربرد هریک لیست می نماید.

الگوی بررسی تطبیق/عبارت باقاعده	شرح عملکرد
^	تنها با ابتدای خط قابل تطبیق می باشد.
$	فقط با انتهای خط، فعل تطبیق را انجام می دهد. در صورتی که کلمه ی منطبق در انتهای رشته، وجود داشته آن را یافته و برمی گرداند.
.	با تمامی کاراکترها به استثنای newline قابل تطبیق می باشد. با استفاده از پارامتر m به راحتی می توان امکان تطبیق آن با newline را نیز فراهم آورد.
[...]	با تمامی کاراکترهای موجود در [] تطبیق داده می شود.
[^...]	با تمامی کاراکترهای خارج از [] انطباق انجام می دهد.
re*	با 0 یا چندین نمونه از عبارت قبلی منطبق می شود.
re+	با 1 یا چندین نمونه از عبارت قبلی منطبق می شود.
re?	با 0 یا 1 نمونه از عبارت قبلی منطبق می شود.
re{ n}	چنانچه در عبارت قبلی n تا نمونه از عبارت قبلی وجود داشته باشد، با آن منطبق می شود.

re{ n,}	با n یا بیشتر نمونه از عبارت قبلی مچ می شود.
re{ n, m}	با حداقل n و حداکثر m نمونه از عبارت پیشین منطبق می شود.
a| b	در صورت وجود هر یک از دو کاراکتر a یا b انطباق رخ می دهد.
(re)	عبارات باقاعده را گروه بندی کرده و متن منطبق را به خاطر می آورد.
(?imx)	به طور موقت بین پارامترهای i، m یا x داخل عبارت باقاعده یا الگوی مورد نظر سویچ می کند (toggle). چنانچه در پرانتز قرار داشت، در آن صورت تنها آن ناحیه تحت تاثیر قرار می گیرد.
(?-imx)	به طور موقت بین پارامترهای i، m یا x داخل عبارت باقاعده سویچ می کند. داخل پرانتز، تنها ناحیه ی مربوطه تحت تاثیر قرار می گیرد.
(?: re)	عبارات باقاعده را بدون اینکه متن منطبق را به خاطر بیاورد، گروه بندی می کند.
(?imx: re)	به طور موقت بین پارامترهای i، m یا x داخل پرانتز سویچ (toggle) می کند.
(?-imx: re)	به طور موقت بین پارامترهای i، m یا x داخل پرانتز سویچ می کند.
(?#...)	Comment
(?= re)	موقعیت تطبیق متن را بر اساس الگوی اعلان شده، مشخص می نماید. بازه ی خاصی را مشخص نمی کند.

(?! re)	موقعیت تطبیق متن را باتوجه به نقیض الگو (pattern negation) مشخص می کند. بازه ی خاصی را مشخص نمی کند. (pattern negation زمانی که با استفاده از کاراکتر !? انتظار می رود، الگو منطبق نباشد و نتیجه ی صحیح حاصل نشود)
(?> re)	Matches independent pattern without backtracking.
\w	چنانچه پارامترهای LOCALE و UNICODE استفاده نشده باشند، در آن صورت با تمامی کاراکترهای الفبایی-عددی منطبق می شود.
\W	در صورتی که پارامترهای LOCALE و UNICODE قید نشده باشند، با تمامی کاراکترهایی که الفبایی-عددی هستند منطبق می شود.
\s	با کاراکتر خط فاصله (whitespace) منطبق می شود. این الگو کارایی مشابه [\t\n\r\f] دارد.
\S	با مواردی که خط فاصله در آن وجود ندارد منطبق می شود.
\d	Matches digits. Equivalent to [0-9]. منحصرا با کاراکترهای عددی منطبق شده و معادل [0-9] می باشد.
\D	منحصرا با کاراکترهای غیرعددی تطبیق انجام داده و در صورت یافتن مورد منطبق آن را استخراج می کند.
\A	تنها با ابتدای یک رشته یا متن تطبیق انجام می دهد.
\Z	تنها با انتهای رشته ی مورد نظر مچ شده و مورد منطبق در انتهای رشته را استخراج می کند.

\z	با انتهای رشته تطبیق انجام داده و مورد منطبق در انتهای رشته را بازگردانی می کند.
\G	Matches point where last match finished.
\b	Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets. زمانی که متن خارج از [] باشد، با اول یا انتهای کلمه منطبق می شود. اگر متن داخل [] باشد، با کاراکتر backspace منطبق می شود.
\B	Matches nonword boundaries. اگر انتها یا ابتدای کلمه کاراکتری nonword باشد، با آن منطبق می شود.
\n, \t, etc.	با کاراکترهای newline، carriage return، tab و غیره ... منطبق می شود.
\1...\9	Matches nth grouped subexpression.
\10	Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

نمونه هایی از عبارات با قاعده

کاراکترهای ثابت (literal)

شرح	مثال
با واژه ی "python" منطبق می شود.	python

Character class (مجموعه کاراکترها)

به واسطه ی این امکان توسعه دهنده می تواند به موتور regex اعلان کند که از میان چندین کاراکتر، تنها یکی را استخراج کند.

شرح	مثال
با هر یک از دو واژگان "Python" یا "python" منطبق می شود.	[Pp]ython
با هر یک از دو واژه ی "ruby" یا "rube" انطباق می یابد.	rub[ye]
با هر یک از پنج مصوت کوچک (lowercase vowel) منطبق می شود. به عبارت دیگر، یکی از حروف بین بازه ی فوق را انتخاب و استخراج می نماید.	[aeiou]
با هر یک از اعداد (از 9 الی 0) منطبق می باشد.	[0-9]
با هر یک از حروف ASCII از a الی z منطبق می شود.	[a-z]

[A-Z]	هر یک از حروف ASCII بزرگ در متن مورد نظر موجود بود، انطباق انجام می گیرد.
[a-zA-Z0-9]	در صورت وجود a تا z یا A تا Z و نیز 0 تا 9 انطباق صورت می گیرد.
[^aeiou]	با هر یک از مصوت هایی که با حروف بزرگ نوشته می شوند، منطبق محسوب می شود.
[^0-9]	با تمامی کاراکترها، به استثنای اعداد، منطبق محسوب می شود.

Character class ها (مجموعه کاراکترها)

مثال	شرح
.	با هر کاراکتری به غیر از newline منطبق شده و آن را استخراج می کند.
\d	[0-9] : با کاراکترهای عددی منطبق شده و آن را استخراج می کند.
\D	[^0-9] : با کاراکترهای غیرعددی منطبق شده و آن ها را استخراج می کند.
\s	[\t\r\n\f] :با کاراکتر فاصله (space) منطبق می شود.
\S	[^ \t\r\n\f] :با تمامی کاراکترهای غیر خط سفید (space) منطبق می شود. به عبارت دیگر هر چیزی به جز فاصله را استخراج می کند.

\w	از میان [A-Za-z0-9_] یکی را انتخاب کرده و استخراج می کند (با تنها یک کاراکتر word منطبق می شود).
\W	با تمامی کاراکترها به غیر از [^A-Za-z0-9_] منطبق می شود (با تنها یک کاراکتر nonword منطبق می شود).

Repetition Cases (مواردی که در آن چندبار انطباق رخ می دهد)

مثال	شرح
ruby?	با "rub" یا "ruby" منطبق شده و آن را استخراج می نماید: y اختیاری می باشد.
ruby*	کلمه ی "rub" را به همراه 0 یا بیشتر ys انتخاب کرده و استخراج می کند.
ruby+	با واژه ی "rub" به همراه 1 ys یا بیشتر منطبق شده و آن را استخراج می کند.
\d{3}	با دقیقا 3 عدد منطبق شده و آن ها را استخراج می نماید.
\d{3,}	با 3 یا بیشتر عدد منطبق شده و آن را استخراج می کند.
\d{3,5}	با 3، 4 یا 5 کاراکتر عددی منطبق شده و آن ها را استخراج می کند.

انطباق با کمترین تعداد مورد تکراری در رشته (nongreedy repetition)

با کمترین تعداد مورد تکراری (بخش قبلا یافته و استخراج شده) در رشته ی مورد نظر منطبق می شود:

مثال	شرح
<.*>	اگر رشته ی مورد نظر شما "<python>" باشد (فقط این رشته خروجی دلخواه شما باشد) ، آنگاه از متن "<python>perl>"، عبارت باقاعده ی جاری کل متن "<python>perl>" را بازمی گرداند که در اصطلاح به آن greedy repetition می گویند.
<.*?>	فرض بگیرید رشته ی تستی شما به این شکل باشد: "<python>perl>" . عبارت باقاعده ی جاری تنها با "<python>" منطبق شده و آن را استخراج می کند.

مشخص کردن انتها و ابتدای موقعیت استخراج با پرانتز (Grouping)

مثال	شرح
\D\d+	No group: + repeats \d
(\D\d)+	Grouped: + repeats \D\d pair
([Pp]ython(,)?)+	با هر یک از رشته های "Python"، "Python, python, python" منطبق می باشد.

Backreferences (تطبیق مجدد و استفاده از بخش های یافته شده ی قبلی)

Backreferences (تطبیق مجدد و استخراج مورد منطبق قبلی) تطبیق و استخراج گروهی که قبلا انطباق با آن رخ داده را فراهم می آورد.

نمونه	شرح
([Pp])ython&\1ails	با هر یک از دو رشته ی python&pails یا Python&Pails منطبق شده و آن را استخراج می کند.
(['"])[^\1]*\1	با رشته ی تک کوتیشن یا دابل کوتیشن منطبق شده و آن را استخراج می کند. 1\ با هر موردی که گروه اول با آن منطبق شد، مچ می شود. 2\ نیز به همین ترتیب، با هر موردی که گروه اول با آن منطبق شده، مچ و در نهایت آن را استخراج می کند.

نمونه های دیگر از عبارات باقاعده

مثال	شرح
python|perl	با یکی از دو رشته ی "python" یا "perl" منطبق شده و آن را استخراج می کند.
rub(y|le))	"ruby" یا "ruble" را استخراج می کند.
Python(!+|\?)	با "Python" که در دنباله ی آن یک یا بیشتر ! و یا ? باشد، منطبق می شود.

Anchor ها در عبارات باقاعده

Anchor ها به کاراکتر خاصی اشاره نمی کنند، بلکه محل استخراج (match position) را مشخص کرده و به موقعیت انطباق اشاره می کنند.

مثال	شرح
^Python	با واژه ی "Python"، چنانچه در ابتدای رشته ی متنی یا خط قرار داشته باشد، منطبق شده و آن را استخراج می کند.
Python$	چنانچه کلمه ی "Python" در انتهای خط یا رشته بود، آن را استخراج کن.
\APython	در صورتی که واژه ی "Python" در ابتدای رشته باشد، با آن منطبق شده و واژه ی مذکور را استخراج می کند.
Python\Z	با واژه ی "Python"، زمانی که در انتهای رشته جای گرفته باشد، منطبق شده و آن را استخراج می کند.
\bPython\b	Match "Python" at a word boundary چنانچه "Python" در ابتدا یا انتهای متن مورد نظر بود، آن را استخراج می کند.
\brub\B	\B is nonword boundary: match "rub" in "rube" and "ruby" but not alone

Python(?=!)	با واژه ی"Python"، در صورتی که در دنباله ی آن علامت تعجب یا کاراکتر ! ذکر شده باشد، منطبق شده و آن را استخراج می کند.
Python(?!!)	با واژه ی "Python" منطبق شده و آن را استخراج می کند، به شرطی که در دنباله ی آن علامت تعجب وجود نداشته باشد.

ساختار نحوی ویژه با پرانتز

مثال	شرح
R(?#comment)	با "R" منطبق شده و آن را استخراج می کند. باقی آنچه مشاهده می کنید صرفا comment است.
R(?i)uby	عدم حساس بودن به کوچک یا بزرگی حروف به هنگام انطباق با "uby" و استخراج آن
R(?i:uby)	مشابه ی نمونه ی فوق
rub(?:y|le))	گروه بندی بدون پیاده سازی \1 انطباق با مورد قبلی و استفاده ی مجدد از آن یا به عبارت دیگر بدون ایجاد backreference \1.

برنامه نویسی CGI در پایتون (نوشتن برنامه های تولید محتوای پویا در سرویس دهنده بر اساس استاندارد های CGI با پایتون)

Common Gateway Interface (رابط درگاه مشترک) یا به اختصار CGI یک سری استاندارد است که نحوه ی تبادل اطلاعات بین سرویس دهنده (web server) و اسکریپت اختصاصی (برنامه ی CGI) را مشخص می کند.

CGI چیست؟

Common Gateway Interface یک قسمتی از سرویس دهنده (web server) است که امکانی را فراهم می کند تا برنامه ای در سمت سرویس دهنده اجرا شود و خروجی آن از طریق صفحه ی اپلیکیشن تحت وب برای کاربری که به سرویس دهنده متصل شده به نمایش در آید. CGI در زمره ی اولین روش هایی است که برای تولید و ارائه ی محتوای پویا در صفحات وب بکار گرفته شد.

CGI یک متد استاندارد است که برای ایجاد محتوای پویا در صفحات وب و برنامه‌های کاربردی تحت وب استفاده می‌شود. CGI هنگامی که روی سرور یک وب‌سرور اجرا می‌شود، یک واسطه میان وب‌سرور و برنامه‌هایی که محتوای وب را ایجاد می‌کنند به وجود می‌آورد. این برنامه‌ها را CGI Script یا به‌طور خلاصه CGI می‌نامند که اغلب با زبان‌های اسکریپت نویسی نوشته می‌شوند، اما امکان نوشتن آن‌ها با زبان‌های برنامه‌نویسی نیز وجود دارد.

وبگردی

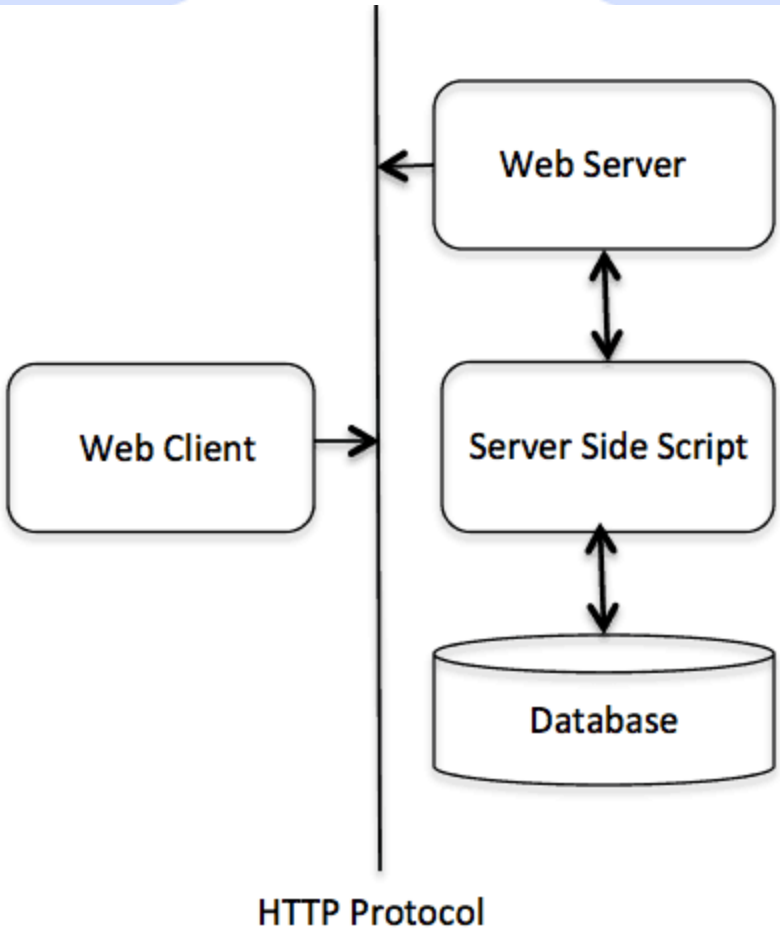
به منظور درک مفهوم CGI، یک لینک را درنظر بگیرید که کاربر با کلیک آن می خواهد صفحه ی وب یا آدرس اینترنتی را مشاهده کند.

- مرورگر به سرور HTTP متصل شده و درخواست URL، به طور مثال، filename را می دهد. سرویس دهنده ی وب URL را parse (تحلیل) کرده و به دنبال filename می گردد. پس از یافتن فایل، آن را به مرورگر ارسال می کند. در صورت عدم وجود فایل، یک پیغام خطا به کاربر ارسال و اعلان می کند که چنین فایلی وجود ندارد.
- مرورگر پاسخ را از سرویس دهنده گرفته و سپس یا فایل مدنظر را تحویل می دهد و یا پیغام خطا را برای کاربر به نمایش می گذارد.

حال این سناریو را درنظر بگیرید. سرویس دهنده ی HTTP را طوری تنظیم کنید که هرگاه یک فایل معین در پوشه ی خاصی فراخوانی شد، سرویس دهنده فایل را به مرورگر ارسال نکند بلکه آن را به

صورت یک برنامه اجرا نماید. سپس هر آنچه خروجی برنامه بود به مرورگر جهت نمایش برای کاربر ارسال کند. این تابع Common Gateway Interface یا CGI خوانده شده و برنامه هایی که تابع جزئی از آن می باشد، اسکریپت های CGI می گویند. برنامه های CGI می توانند یک اسکریپت Python، PERL، Shell، برنامه ی نوشته شده با C یا C++ باشد.

نمودار معماری CGI



تنظیمات سرویس دهنده و پشتیبانی آن از CGI (Web Server (Configuration

پیش از اقدام به برنامه نویسی CGI، لازم است اطمینان حاصل کنید که سرویس دهنده ی وب قابلیت پشتیبانی از CGI را داشته و طوری تنظیم شده که توانایی مدیریت برنامه های CGI را داشته باشد. تمامی برنامه های CGI که قرار است بر روی سرور HTTP اجرا شود، داخل پوشه ای از پیش تعریف شده (pre-configured directory) نگهداری

می شود. این پوشه به CGI Directory معروف بوده و طبق قرارداد var/www/cgi-bin نامیده می شود. به طور پیش فرض فایل های CGI دارای پسوند cgi. می باشند، اما شما می توانید پسوند py. فایل های پایتون را برای آن ها استفاده کنید.

به طور پیش فرض، سرویس دهنده ی لینوکس طوری تنظیم شده که فقط اسکریپت های داخل پوشه ی cgi-bin، تحت آدرس var/www/ را اجرا کند. اگر می خواهید پوشه ی دیگری را جهت میزبانی و اجرای اسکریپت های CGI مشخص نمایید، خط های زیر را در داخل فایل httpd.conf به Comment تبدیل نمایید:

```
<Directory "/var/www/cgi-bin">
AllowOverride None
Options ExecCGI
Order allow,deny
Allow from all
</Directory>
<Directory "/var/www/cgi-bin">
Options All
</Directory>
```

در آموزش حاضر فرض را بر این می گذاریم که شما سرویس دهنده (web server) را تنظیم و آن را برای اجرای تمامی برنامه های CGI که با اسکریپت های Perl یا Shell نوشته شده، آماده کرده اید.

اولین برنامه ی CGI

در زیر لینک ساده ای مشاهده می کنید که شما را به یک برنامه ی CGI به نام hello.py هدایت می کند. این فایل تحت پوشه ی var/www/cgi-bin/ نگهداری شده و دربردارنده ی محتوای زیر می باشد. پیش از اجرای برنامه ی CGI، لازم است با فراخوانی دستور chmod 755 hello.py یونیکس وضعیت یا مد فایل را به executable و قابل اجرا، تغییر داده باشید.

```
#!/usr/bin/python
print "Content-type:text/html\r\n\r\n"
print '<html>'
print '<head>'
print '<title>Hello Word - First CGI Program</title>'
print '</head>'
print '<body>'
print '<h2>Hello Word! This is my first CGI program</h2>'
print '</body>'
```

```
print '</html>'
```

پس از کلیک بر روی hello.py، خروجی زیر به نمایش در می آید:

```
Hello Word! This is my first CGI program
```

hello.py یک اسکریپت ساده ی Python است که خروجی خود را در فایل STDOUT یا همان نمایشگر چاپ می کند. در اینجا لازم است به نکته ی دیگری اشاره کنیم و آن اولین خطی از کد است (Content-type:text/html\r\n\r\n) که با دستور print چاپ می شود. این خط کد به مرورگر ارسال شده و نوع محتوایی که نهایتا در نمایشگر (پنجره ی مرورگر) چاپ می شود را اعلان می کند.

تا به اینجای آموزش قطعا با مفاهیم ساده ی CGI آشنا شده و می توانید با استفاده از زبان پایتون برنامه های قدرتمند CGI بنویسید. اسکریپتی که با پایتون نوشته می شود قادر است با سایر سیستم های خارجی نظیر سیستم های مدیریت دیتابیس رابطه ای یا RDBMS تعامل برقرار کرده و اطلاعات لازم را رد و بدل نماید.

HTTP Header (اطلاعاتی درباره ی بسته ی ارسال شده به مرورگر)

دستور Content-type:text/html\r\n\r\n که در بالا به آن اشاره شد، در واقع بخشی از http header است که به مرورگر ارسال شده و آن را از محتوای فایل آگاه می سازد. کل http header به صورت زیر خواهد بود:

```
HTTP Field Name: Field Content
For Example
Content-type: text/html\r\n\r\n
```

در زیر تعدادی HTTP header پرکاربرد که در برنامه نویسی CGI به طور گسترده مورد استفاده قرار می گیرد را همراه با شرح کاربرد مشاهده می کنید:

Header	شرح

Content-type:	نوع رشته ای (MIME string) که فرمت فایل بازگشتی را مشخص می کند. به طور مثال می توان به Content-type:text/html اشاره کرد.
Expires: Date	تاریخی که پس از آن اطلاعات مربوطه اعتبار خود را از دست می دهد. مرورگر بر اساس مقدار این header صفحات وب را بروز رسانی می کند. یک رشته ی مجاز و معتبر با فرمت 01 Jan 1998 12:00:00 GMT نگاشته می شود.
Location: URL	URL ای که بجای URL درخواستی بازگردانده می شود. می توانید با استفاده از این فیلد درخواست را به هر فایلی بازگشت یا هدایت (redirect) نمایید.
Last-modified: Date	تاریخ آخرین بروز رسانی منبع و محتوای مورد نظر.
Content-length: N	طول داده و حجم اطلاعات بازگشتی بر حسب بایت. مرورگر با استفاده از این مقدار مدت زمانی که طول می کشد تا فایل کاملا بارگیری شود را تخمین می زند.
Set-Cookie: String	کوکی را بر اساس رشته ی ارسال شده مقداردهی و تنظیم می کند.

متغیرهای CGI

تمامی برنامه های CGI به مقادیر متغیرهای زیر دسترسی دارند. این متغیرها در برنامه نویسی CGI بسیار نقش مهمی را ایفا کرده و کاربرد زیادی دارند.

اسم متغیر	شرح

CONTENT_TYPE	نوع داده ای در این متغیر نگه داری می شود. زمانی مورد استفاده قرار می گیرد که سرویس گیرنده یا کلاینت محتوای الصاق شده را به سرویس دهنده یا سرور ارسال می کند. برای مثال می توان به بارگذاری فایل در سرور و file upload اشاره کرد.
CONTENT_LENGTH	طول و حجم اطلاعات کوئری یا درخواست شده در این متغیر ذخیره می شود. متغیر جاری تنها برای درخواست هایی که با POST ارسال می شوند، قابل بهره برداری می باشد.
HTTP_COOKIE	کوکی های تنظیم شده (مقداردهی شده) را در قالب جفت های کلید و مقدار بازگردانی می کند.
HTTP_USER_AGENT	اسم مرورگری که درخواست را به سرویس دهنده ارسال می کند. این متغیر اطلاعات مربوط به user agent که همان فرستنده ی درخواست محتوا از سرور می باشد را در خود ذخیره می کند.
PATH_INFO	اطلاعات مربوط به path و محل قرارگیری اسکریپت (برنامه ی CGI) در قالب این متغیر نگهداری می شود.
QUERY_STRING	اطلاعات کدگذاری شده در قالب URL که همراه با متد GET به سرویس دهنده ارسال می شود، داخل این متغیر جای می گیرد.
REMOTE_ADDR	آدرس IP میزبان یا سرور راه دور (remote server) که درخواست اطلاعات را می کند داخل متغیر جاری نگه داری می شود. مورد کاربرد این متغیر غالبا در گزارش گیری (logging) یا احرازهویت (authentication) خلاصه می شود.

REMOTE_HOST	اسم کامل و دقیق میزبان (host) که درخواست را ارسال می کند. چنانچه این اطلاعات دردسترس نبود، می توان با استفاده از REMOTE_ADDR آدرس IR را بازیابی کرد.
REQUEST_METHOD	متد مورد استفاده برای ایجاد و ارسال درخواست. پرکاربردترین این متدها عبارتند از GET و POST.
SCRIPT_FILENAME	آدرس کامل و دقیق که اسکریپت یا برنامه ی CGI در آن مستقر می باشد.
SCRIPT_NAME	اسم اسکریپت یا برنامه ی CGI.
SERVER_NAME	اسم سرور یا آدرس IP
SERVER_SOFTWARE	اسم و ورژن نرم افزار که بر روی سرویس دهنده در حال اجرا می باشد.

در زیر یک برنامه ی ساده ی CGI را می بینید که تمامی متغیرهای CGI را لیست می کند.

```
#!/usr/bin/python
import os
print "Content-type: text/html\r\n\r\n";
print "<font size=+1>Environment</font><\br>";
for param in os.environ.keys():
print "<b>%20s</b>: %s<\br>" % (param, os.environ[param])
```

در زیر خروجی کد را مشاهده می کنید:

Environment<r> HTTP_COOKIE:
__utma=55973678.1477211342.1492087219.1492087219.1492087219.1; __utmb=55973678;
__utmc=55973678;

__utmz=55973678.1492087219.1.1.utmccn=(referral)|utmcsr=google.de|utmcct=/|utmcmd=referral
HTTP_COOKIE: __utma=55973678.1477211342.1492087219.1492087219.1492087219.1;
__utmb=55973678; __utmc=55973678;
__utmz=55973678.1492087219.1.1.utmccn=(referral)|utmcsr=google.de|utmcct=/|utmcmd=referral
CONTEXT_DOCUMENT_ROOT: /var/www/cgi-bin/
CONTEXT_DOCUMENT_ROOT: /var/www/cgi-bin/
SERVER_SOFTWARE: Apache/2.4.6 (CentOS)
SERVER_SOFTWARE: Apache/2.4.6 (CentOS)
CONTEXT_PREFIX: /cgi-bin/
CONTEXT_PREFIX: /cgi-bin/
REQUEST_SCHEME: http
REQUEST_SCHEME: http
SERVER_SIGNATURE:
SERVER_SIGNATURE:
GEOIP_COUNTRY_CODE: EU
GEOIP_COUNTRY_CODE: EU
SERVER_PROTOCOL: HTTP/1.1
SERVER_PROTOCOL: HTTP/1.1
QUERY_STRING:
QUERY_STRING:
PATH: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
PATH: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
HTTP_USER_AGENT: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:52.0) Gecko/20100101 Firefox/52.0
HTTP_USER_AGENT: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:52.0) Gecko/20100101 Firefox/52.0
SERVER_NAME: www.tutorialspoint.com
SERVER_NAME: www.tutorialspoint.com
REMOTE_ADDR: 5.104.65.183
REMOTE_ADDR: 5.104.65.183
GEOIP_COUNTRY_NAME: Europe
GEOIP_COUNTRY_NAME: Europe
HTTP_X_BLUECOAT_VIA: eaefec6a124f3f39
HTTP_X_BLUECOAT_VIA: eaefec6a124f3f39
HTTP_VIA: HTTP/1.1 ECS (fcn/41B7)
HTTP_VIA: HTTP/1.1 ECS (fcn/41B7)
SERVER_PORT: 80
SERVER_PORT: 80
SERVER_ADDR: 10.34.18.35
SERVER_ADDR: 10.34.18.35
DOCUMENT_ROOT: /var/www/tutorialspoint
DOCUMENT_ROOT: /var/www/tutorialspoint
HTTP_X_FORWARDED_PROTO: http
HTTP_X_FORWARDED_PROTO: http

SCRIPT_FILENAME: /var/www/cgi-bin/get_env.py
SCRIPT_FILENAME: /var/www/cgi-bin/get_env.py
SERVER_ADMIN: contact@tutorialspoint.com
SERVER_ADMIN: contact@tutorialspoint.com
SCRIPT_URI: http://www.tutorialspoint.com/cgi-bin/get_env.py
SCRIPT_URI: http://www.tutorialspoint.com/cgi-bin/get_env.py
GEOIP_CONTINENT_CODE: EU
GEOIP_CONTINENT_CODE: EU
HTTP_HOST: www.tutorialspoint.com
HTTP_HOST: www.tutorialspoint.com
SCRIPT_URL: /cgi-bin/get_env.py
SCRIPT_URL: /cgi-bin/get_env.py
HTTP_UPGRADE_INSECURE_REQUESTS: 1
HTTP_UPGRADE_INSECURE_REQUESTS: 1
HTTP_CACHE_CONTROL: max-stale=0
HTTP_CACHE_CONTROL: max-stale=0
REQUEST_URI: /cgi-bin/get_env.py
REQUEST_URI: /cgi-bin/get_env.py
HTTP_ACCEPT: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
GATEWAY_INTERFACE: CGI/1.1
GATEWAY_INTERFACE: CGI/1.1
HTTP_X_FORWARDED_FOR: 91.212.206.55
HTTP_X_FORWARDED_FOR: 91.212.206.55
SCRIPT_NAME: /cgi-bin/get_env.py
SCRIPT_NAME: /cgi-bin/get_env.py
REMOTE_PORT: 33398
REMOTE_PORT: 33398
HTTP_ACCEPT_LANGUAGE: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
HTTP_ACCEPT_LANGUAGE: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
HTTP_X_HOST: www.tutorialspoint.com
HTTP_X_HOST: www.tutorialspoint.com
GEOIP_ADDR: 5.104.65.183
GEOIP_ADDR: 5.104.65.183
REQUEST_METHOD: GET
REQUEST_METHOD: GET
HTTP_ACCEPT_ENCODING: gzip, deflate
HTTP_ACCEPT_ENCODING: gzip, deflate
UNIQUE_ID: WO9x6LegxBEa9DE2Vqkc2AAAAHs
UNIQUE_ID: WO9x6LegxBEa9DE2Vqkc2AAAAHs

متدهای ارسال اطلاعات به سرور (GET و POST)

گاهی لازم می شود که اطلاعاتی را از مرورگر به سرویس دهنده ی وب و در نهایت به برنامه ی CGI ارسال کنید. مرورگرها برای ارسال این اطلاعات به سرویس دهنده از دو متد GET و POST استفاده می کنند.

ارسال اطلاعات با استفاده از متد GET

متد GET اطلاعات کاربری کدگذاری شده که به درخواست صفحه ضمیمه شده را به سرور ارسال می کند. صفحه و اطلاعات کدگذاری شده به وسیله ی کاراکتر ? از یکدیگر جدا می شوند:

http://www.test.com/cgi-bin/hello.py?key1=value1&key2=value2

متد پیش فرض برای ارسال اطلاعات از مرورگر به سرویس دهنده، متد GET می باشد. این متد یک رشته ی طولانی تولید می کند که داخل Location:box مرورگر نمایان می شود. لازم به توضیح است که اگر اطلاعات ارسالی شما حساس هستند (به عنوان مثال گذرواژه)، توصیه می شود از متد GET برای فرستادن اطلاعات استفاده نکنید. متد GET در خصوص حجم اطلاعات قابل ارسال، محدودیتی اعمال می کند. بدین معنی که کاراکترهای ارسالی در رشته ی درخواست اطلاعات (request string) نباید از مرز 1024 کاراکتر تجاوز کند. متد GET اطلاعات مورد نظر را به وسیله ی هدر QUERY_STRING به سرویس دهنده ارسال می کند. این اطلاعات سپس از طریق متغیر QUERY_STRING در برنامه ی CGI قابل دسترسی می باشد.

جهت ارسال اطلاعات با متد GET می توانید به دو طریق زیر اقدام نمایید:

1. می توانید جفت های کلید و مقدار را به انتهای URL متصل کنید.
2. می توانید اطلاعات درخواستی را با استفاده از تگ های <FORM> ارسال کنید (برای این منظور کافی است مقدار ویژگی method را داخل این تگ، برابر "get" قرار دهید).

ارسال اطلاعات از طریق URL و Query String

در زیر یک URL ساده مشاهده می کنید که دو مقدار را با استفاده از متد GET به hello_get.py ارسال می کند.

/cgi-bin/hello_get.py?first_name=ZARA&last_name=ALI

برنامه ی hello_get.py مقادیر دریافتی از مرورگر (ورودی) را مدیریت می نماید. حال با بهره گیری از ماژول cgi به داده های ارسالی دسترسی پیدا می کنیم:

```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
# Create instance of FieldStorage
form = cgi.FieldStorage()
# Get data from fields
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')
print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

نتیجه ی زیر حاصل می گردد:

Hello ZARA ALI

مثالی کاربردی از ارسال اطلاعات FORM با استفاده از متد GET

در زیر یک HTML FORM مشاهده می کنید که مقادیری را پس از فشردن دکمه ی submit به سرویس دهنده ارسال می کند. سپس داده های ارسالی به برنامه ی hello_get.py جهت مدیریت تحویل داده می شود.

```
<form action="/cgi-bin/hello_get.py" method="get">
First Name: <input type="text" name="first_name">  <br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
```

در زیر خروجی form بالا را مشاهده می کنید. پس از وارد کردن مقادیر در دو کادر First Name و Last Name، بر روی دکمه ی Submit کلیک کرده و نتیجه را مشاهده نمایید.

First Name: zara

Last Name: ali Submit

خروجی زیر را در صفحه ی مجزا مشاهده خواهید نمود:

Hello zara ali

ارسال اطلاعات از طریق متد POST

متد امن تری که برای ارسال اطلاعات به برنامه ی CGI می توان از آن بهره گرفت، متد POST می باشد. متد مذکور اطلاعات را درست مشابه GET پکیج بندی می کند، اما بجای اینکه داده های مورد نظر را به صورت یک رشته ی متنی پس از ? در URL همراه با فرم ارسال کند، آن را در قالب پیغامی مجزا به سرویس دهنده تحویل می دهد.

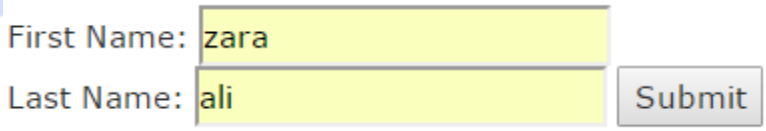
در زیر همان اسکریپت ساده ی hello_get.py را مشاهده می کنید که اطلاعات ارسالی از هر دو روش (متد GET و POST) را مدیریت می نماید.

```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
# Create instance of FieldStorage
form = cgi.FieldStorage()
# Get data from fields
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')
print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

بار دیگر همان مثال را مشاهده می کنید که دو مقدار را به واسطه ی HTML FORM و دکمه ی Submit به سرویس دهنده ارسال می کند. برای مدیریت مقادیر ارسالی به سرویس دهنده نیز از همان برنامه hello_get.py استفاده می شود.

```
<form action="/cgi-bin/hello_get.py" method="post">
First Name: <input type="text" name="first_name"><br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
```

در زیر خروجی واقعی کد فوق را مشاهده می کنید. مقادیر لازم را در کادرهای مربوطه وارد کرده و پس از فشردن دکمه ی Submit، خروجی را مشاهده نمایید:



خروجی زیر را دریافت می کنید:

Hello zara ali

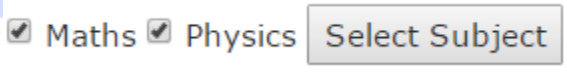
ارسال داده های Checkbox به سرویس دهنده و مدیریت آن با برنامه ی CGI

Checkbox ها زمانی کاربرد دارند که کاربر احتیاج داشته باشد تا چندین گزینه را همزمان انتخاب نماید.

در زیر نمونه ای از کد HTML که یک فرم ساده با دو CHECKBOX را تشکیل داده و در پنجره ی مرورگر نمایش می دهد، مشاهده می نمایید:

```
<form action="/cgi-bin/checkbox.cgi" method="POST" target="_blank">
<input type="checkbox" name="maths" value="on" /> Maths
<input type="checkbox" name="physics" value="on" /> Physics
<input type="submit" value="Select Subject" />
</form>
```

خروجی کد فوق، فرم زیر می باشد:



خروجی فرم حاضر را در زیر مشاهده می کنید:

CheckBox Maths is : ON

CheckBox Physics is : ON

در زیر کد اسکریپت یا برنامه ی checkbox.cgi را که ورودی کاربر (داده های ارسالی از مرورگر به سرویس دهنده) را مدیریت می کند، مشاهده می نمایید:

```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
# Create instance of FieldStorage
form = cgi.FieldStorage()
# Get data from fields
if form.getvalue('maths'):
   math_flag = "ON"
else:
   math_flag = "OFF"
if form.getvalue('physics'):
   physics_flag = "ON"
else:
   physics_flag = "OFF"
print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Checkbox - Third CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> CheckBox Maths is : %s</h2>" % math_flag
print "<h2> CheckBox Physics is : %s</h2>" % physics_flag
print "</body>"
print "</html>"
```

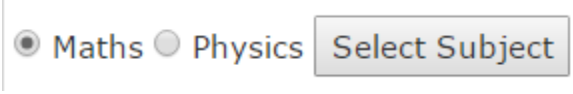
ارسال داده های Radio Button به برنامه ی CGI در سرویس دهنده

Radio Button ها، برخلاف المان Checkbox، به کاربر اجازه ی انتخاب تنها یک گزینه را می دهند.

در زیر کد HTML که یک فرم به همراه دو Radio button را تشکیل می دهد، مشاهده می نمایید:

```
<form action="/cgi-bin/radiobutton.py" method="post" target="_blank">
<input type="radio" name="subject" value="maths" /> Maths
<input type="radio" name="subject" value="physics" /> Physics
<input type="submit" value="Select Subject" />
</form>
```

خروجی کد را در زیر مشاهده می کنید:



در زیر اسکریپت radiobutton.py را که داده های ورودی از کاربر (مرورگر) را دریافت کرده و مدیریت می کند، مشاهده می نمایید:

```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
# Create instance of FieldStorage
form = cgi.FieldStorage()
# Get data from fields
if form.getvalue('subject'):
subject = form.getvalue('subject')
else:
subject = "Not set"
print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Radio - Fourth CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> Selected Subject is %s</h2>" % subject
print "</body>"
print "</html>"
```

ارسال داده های Text Area از مرورگر به برنامه ی CGI در سرویس دهنده

المان TEXTAREA زمانی استفاده می شود که لازم باشد چندین خط متن همراه با فرم به برنامه ی CGI در سرویس گیرنده فرستاده شود.

در زیر یک نمونه کد HTML که یک فرم با کادر TEXTAREA را ساخته و در صفحه نمایش می دهد، مشاهده می نمایید:

```
<form action="/cgi-bin/textarea.py" method="post" target="_blank">
<textarea name="textcontent" cols="40" rows="4">
Type your text here...
</textarea>
<input type="submit" value="Submit" />
</form>
```

خروجی کد فوق، فرم زیر می باشد:

Hello Ali Zara

Submit

خروجی اسکریپت به صورت زیر می باشد:

Entered Text Content is Hello Ali Zara

در زیر کد برنامه ی textarea.cgi که ورودی کاربر و مقدار ارسالی از مرورگر را در سمت سرویس گیرنده دریافت کرده و مدیریت می نماید، مشاهده می کنید:

```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
# Create instance of FieldStorage
form = cgi.FieldStorage()
# Get data from fields
if form.getvalue('textcontent'):
text_content = form.getvalue('textcontent')
else:
text_content = "Not entered"
print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>";
print "<title>Text Area - Fifth CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> Entered Text Content is %s</h2>" % text_content
print "</body>"
```

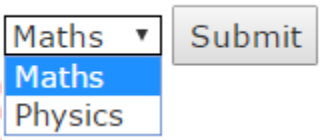
ارسال مقادیر کادر کشویی یا Drop down به برنامه ی CGI در سرویس دهنده

از المان کادر کشویی یا drop-down زمانی استفاده می کنیم که لازم باشد گزینه های متعددی برای کاربر نمایش داده و در عین حال اجازه ی انتخاب تنها یک یا دو گزینه در آن واحد را به وی بدهیم:

در زیر نمونه کد HTML که یک فرم به همراه کادر کشویی (drop-down) را در پنجره ی مرورگر برای کاربر نمایش می دهد، مشاهده می کنید:

```
<form action="/cgi-bin/dropdown.py" method="post" target="_blank">
<select name="dropdown">
<option value="Maths" selected>Maths</option>
<option value="Physics">Physics</option>
</select>
<input type="submit" value="Submit"/>
</form>
```

خروجی کد فوق به این صورت می باشد:



در زیر کد اسکریپت dropdown.py که داده های ارسالی از مرورگر را در سمت سرویس گیرنده دریافت و مدیریت می کند، مشاهده می نمایید:

```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
# Create instance of FieldStorage
form = cgi.FieldStorage()
# Get data from fields
if form.getvalue('dropdown'):
subject = form.getvalue('dropdown')
else:
subject = "Not entered"
print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Dropdown Box - Sixth CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> Selected Subject is %s</h2>" % subject
print "</body>"
print "</html>"
```

استفاده از کوکی ها در CGI

پروتکل HTTP داده های مربوط به وضعیت جاری را ذخیره نمی کند و به عبارتی stateless می باشد. اما همان طور که می دانید یک وب سایت تجاری می بایست اطلاعات جلسه ی کاری یا session جاری که اطلاعات مربوط به کاربر را دربردارد، بین صفحات مختلف وب نگه دارد. به طور مثال، شرایطی را در نظر بگیرید که در آن ثبت نام کاربر پس از تکمیل چندین صفحه ی وب به پایان می رسد. در چنین سناریویی چگونه می بایست اطلاعات کاربر مقیم در session را در تمامی صفحات وب ذخیره نگه داشت؟

در اغلب شرایط استفاده از کوکی بهترین روش برای ذخیره و یادآوری تنظیمات و اطلاعات کاربر است که منجر به کیفیت بالاتر و تجربه ی کاربری بهتر می شود.

کوکی چگونه مورد استفاده قرار می گیرد؟

سرویس دهنده ای که اپلیکیشن بر روی آن مستقر می باشد، اطلاعاتی را در قالب کوکی به مرورگر ارسال می کند. مرورگر بر اساس تنظیمات کاربر می تواند این داده ها را پذیرفته و متعاقبا بر روی هارد دیسک کامپیوتر کاربر ذخیره نماید. حال، هنگامی که کاربر به صفحه ی دیگری از وب سایت مراجعه می کند، اطلاعات ذخیره شده در کوکی به راحتی از کامپیوتر کاربر بازیابی می شود. پس از واکشی اطلاعات از کوکی، سرویس دهنده به خاطر می آورد که کاربر جاری از چه صفحاتی بازدید کرده و اطلاعات لازم را می خواند.

کوکی ها رکوردی از داده های متنی ساده متشکل از 5 فیلد با طول متغیر هستند:

- Expires: تاریخی که کوکی پس از آن اعتبار خود را از دست داده و غیرقابل استفاده می شود. چناچه فیلد جاری مقداری نداشته باشد، در آن صورت به مجرد خروج کاربر از مرورگر، داده های ذخیره شده در آن نامعتبر و غیرقابل استفاده می شوند.
- Domain: اسم دامنه یا آدرس سایت شما در این فیلد ذخیره می شود.

- Path: آدرس پوشه یا محل قرارگیری صفحه ی وب که کوکی را مقداردهی می کند. در شرایطی که ممکن است کوکی از هر صفحه یا پوشه ای بازیابی شود، این فیلد می تواند مقداری نداشته باشد.

- Secure: چنانچه فیلد جاری با رشته ی "secure" مقداردهی شده باشد، در آن صورت کوکی مورد نظر منحصرا از یک سرویس دهنده ی ایمن و secure قابل واکشی می باشد. اگر این فیلد مقداری نداشته باشد، چنین محدودیتی هم اعمال نخواهد شد.

- Name=Value: کوکی ها به صورت جفت های کلید و مقدار مقداردهی و بازیابی می شوند.

تنظیم و استفاده از کوکی

ارسال کوکی به مرورگر بسیار آسان می باشد. کوکی ها در واقع همراه با HTTP Header، در حالی که قبل از فیلد Content-type درج شده، به مرورگر فرستاده می شوند. حال به فرض اینکه می خواهید UserID و Password را به عنوان کوکی انتخاب کنید، می بایست کد آن را به صورت زیر تنظیم نمایید:

```
#!/usr/bin/python
print "Set-Cookie:UserID=XYZ;\r\n"
print "Set-Cookie:Password=XYZ123;\r\n"
print "Set-Cookie:Expires=Tuesday, 31-Dec-2007 23:12:40 GMT";\r\n"
print "Set-Cookie:Domain=www.tutorialspoint.com;\r\n"
print "Set-Cookie:Path=/perl;\n"
print "Content-type:text/html\r\n\r\n"
.....Rest of the HTML Content....
```

با بررسی مثال جاری قطعا با نحوه ی مقداردهی و تنظیم کوکی آشنا شده اید. در واقع کار مقداردهی کوکی را به وسیله ی Set-Cookie از HTTP Header انجام می دهیم.

تنظیم و مقداردهی ویژگی ها (attribute) و فیلدهای کوکی نظیر Expires، Domain و Path اختیاری می باشد. گفتنی است که کوکی ها پیش از ارسال خط "Content-type:text/html\r\n\r\n به مرورگر، تنظیم و مقداردهی می شوند.

بازیابی کوکی ها

به راحتی می توان تمامی کوکی های مقداردهی شده را بازیابی کرد. کوکی ها در متغیر HTTP_COOKIE به صورت زیر ذخیره می شوند:

```
key1=value1;key2=value2;key3=value3....
```

مثال زیر نحوه ی بازیابی کوکی ها را به صورت کاربردی به نمایش می گذارد:

```
#!/usr/bin/python
# Import modules for CGI handling
from os import environ
import cgi, cgitb
if environ.has_key('HTTP_COOKIE'):
for cookie in map(strip, split(environ['HTTP_COOKIE'], ';')):
(key, value ) = split(cookie, '=');
if key == "UserID":
user_id = value
if key == "Password":
password = value
print "User ID  = %s" % user_id
print "Password = %s" % password
```

کد بالا خروجی زیر را برمی گرداند:

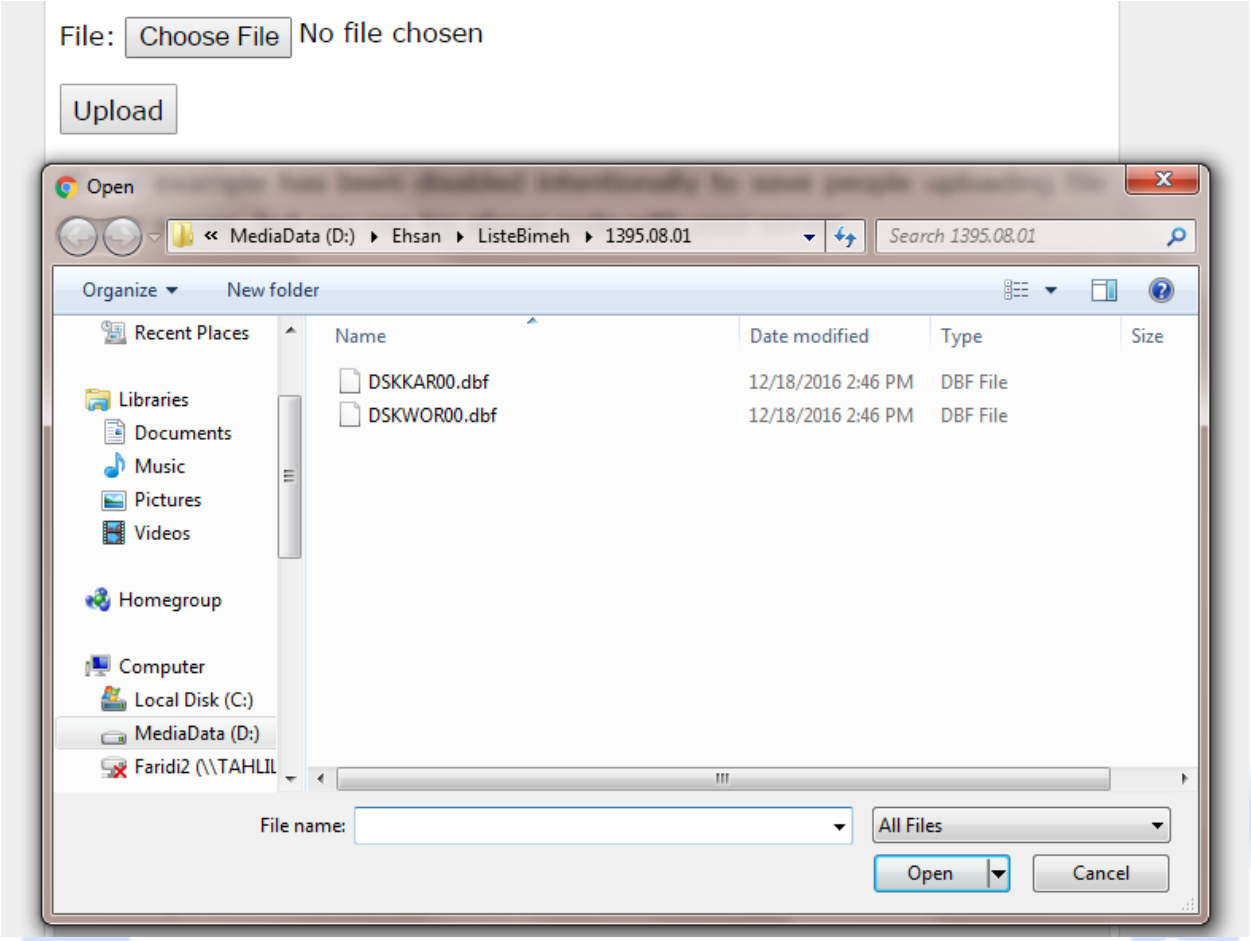
```
User ID = XYZ
Password = XYZ123
```

مثالی از آپلود فایل

به منظور آپلود یک فایل، لازم است ویژگی enctype از فرم HTML را بر روی multipart/form-data تنظیم نمایید. تگ input با ویژگی type آن بر روی file تنظیم شده، یک دکمه ی "Browse" ایجاد می کند.

```
<html>
<body>
<form enctype="multipart/form-data"
action="save_file.py" method="post">
<p>File: <input type="file" name="filename" /></p>
<p><input type="submit" value="Upload" /></p>
</form>
</body>
</html>
```

خروجی کد فوق به صورت زیر می باشد:



در زیر اسکریپت save_file.py که آپلود فایل را مدیریت می کند، مشاهده می کنید:

```
#!/usr/bin/python
import cgi, os
import cgitb; cgitb.enable()
form = cgi.FieldStorage()
# Get filename here.
fileitem = form['filename']
# Test if the file was uploaded
if fileitem.filename:
# strip leading path from file name to avoid
# directory traversal attacks
fn = os.path.basename(fileitem.filename)
open('/tmp/' + fn, 'wb').write(fileitem.file.read())
message = 'The file "' + fn + '" was uploaded successfully'
else:
message = 'No file was uploaded'
print """\
Content-Type: text/html\n
```

```
<html>
<body>
<p>%s</p>
</body>
</html>
""" % (message,)
```

چنانچه اسکریپت فوق را بر روی Unix/Linux اجرا کنید، در آن صورت می بایست خود کار جایگزینی file separator (کاراکتر تفکیک گر) را به صورت انجام دهید. بر روی ویندوز نیازی به این کار نیست و صرف استفاده از دستور open() در کد بالا، عملیات لازم را انجام می دهد.

```
fn = os.path.basename(fileitem.filename.replace("\\", "/" ))
```

نحوه ی نمایش و پیاده سازی کادر محاوره ای "File Download" جهت بارگیری محتوا از اینترنت

گاهی توسعه دهنده لازم می داند، زمانی که کاربر بر روی لینکی از صفحه کلیک کرد، بجای نمایش محتوای آن لینک در صفحه، یک کادر محاوره ای "File Download" جهت دانلود اطلاعات مورد نظر به صورت فایل، به کاربر نشان داده شود. این کار به راحتی از طریق HTTP header قابل اجرا می باشد. این HTTP header می بایست با header های نام برده در بخش های قبلی متفاوت باشد.

در مثال حاضر، زمانی که کاربر بر روی لینک کلیک می کند، یک کادر محاوره ای نمایان شده که به وی امکان دانلود فایل FileName را می دهد:

```
#!/usr/bin/python
# HTTP Header
print "Content-Type:application/octet-stream; name=\"FileName\"\r\n";
print "Content-Disposition: attachment; filename=\"FileName\"\r\n\n";
# Actual File Content will go here.
fo = open("foo.txt", "rb")
str = fo.read();
print str
# Close opend file
fo.close()
```

Python و دسترسی به دیتابیس MySQL

پایتون جهت دسترسی به دیتابیس از توابع کتابخانه ای DB-API استفاده کرده و interface هایی که برای اتصال به پایگاه داده و مدیریت داده های اپلیکیشن بایستی پیاده سازی شود، بر اساس همین استاندارد می باشد. در واقع بیشتر رابط های (interface) اتصال به دیتابیس از این استاندارد پیروی می کنند.

توسعه دهنده می تواند بر اساس نیاز اپلیکیشن خود، دیتابیس مناسب را انتخاب کند. توابع کتابخانه ای اتصال و استفاده از دیتابیس زبان پایتون (API) از database server های زیر پشتیبانی می کند:

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

برای مشاهده لیست interface های اتصال به دیتابیس می توانید به این لینک مراجعه کنید: http://wiki.python.org/moin/DatabaseInterfaces. لازم به ذکر است که برای اتصال به هر دیتابیس مجزا و جهت دسترسی یا مدیریت داده های اپلیکیشن می بایست یک ماژول DB API جداگانه دانلود و نصب نمایید. به طور مثال، چنانچه توسعه دهنده می بایست علاوه بر MySQL به دیتابیس Oracle دسترسی پیدا کند، بدیهی است که باید ماژول های مجزا هریک را جداگانه از اینترنت بارگیری کرده و نصب نماید (ماژول های دیتابیس MySQL و Oracle).

DB API یک حداقل استاندارد برای مدیریت دیتابیس با استفاده از ساختار و دستور نحوی زبان برنامه نویسی پایتون در اختیار توسعه دهنده قرار می دهد. استفاده از این مجموعه توابع کتابخانه ای یا API مراحل زیر را شامل می شود:

- وارد کردن ماژول این مجموعه توابع کتابخانه ای با استفاده از دستور import
- برقراری اتصال به دیتابیس

- صدور و فراخوانی دستورات و توابع (Store procedure) مورد نیاز SQL
- بستن و قطع اتصال به دیتابیس

در آموزش حاضر تمامی این مباحث را با دیتابیس رابطه ای MySQL مدیریت می کنیم. به همین جهت ماژول MySQLdb را بارگیری نموده و نصب می کنیم.

MySQLdb

MySQLdb یک رابط یا interface برای اتصال به سرویس دهنده دیتابیس MySQL (MySQL Database server) با زبان برنامه نویسی پایتون است که توسعه دهنده می بایست برای دسترسی و مدیریت داده های اپلیکیشن آن را پیاده سازی کند. این اینترفیس ویرایش Database API 2.0 پایتون را پیاده سازی کرده و بر پایه ی MySQL C API ساخته شده است.

نصب MySQLdb

جهت استفاده از توابع MySQLdb لازم است ماژول آن را بر روی دستگاه خود نصب نمایید. کافی است دستورات زیر را در اسکریپت پایتون لحاظ کرده و آن ها را اجرا نمایید:

```
#!/usr/bin/python
import MySQLdb
```

کد فوق یک پیغام خطا مبنی بر اینکه ماژول MySQLdb نصب نشده است تولید می کند:

```
Traceback (most recent call last):
File "test.py", line 3, in <module>
import MySQLdb
ImportError: No module named MySQLdb
```

به منظور نصب ماژول MySQLdb، کافی است دستورات زیر را تایپ نمایید:

```
For Ubuntu, use the following command -
$ sudo apt-get install python-pip python-dev libmysqlclient-dev
For Fedora, use the following command -
$ sudo dnf install python python-devel mysql-devel redhat-rpm-config gcc
For Python command prompt, use the following command -
```

```
pip install MySQL-python
```

نکته: لازم است جهت نصب ماژول فوق، root privilege (مجوز در سطح دسترسی به فایل های ریشه) داشته باشید.

پیاده سازی اتصال به دیتابیس (Database connection)

پیش از اتصال به دیتابیس MySQL، لازم است اقدامات زیر را کامل انجام داده باشید:

- یک دیتابیس به نام TESTDB ایجاد نموده اید.
- یک جدول به نام EMPLOYEE در دیتابیس مزبور تعریف کرده اید.
- جدول مورد نظر فیلدهایی به نام FIRST_NAME، LAST_NAME، AGE، SEX و INCOME را دربرمی گیرد.
- جهت دسترسی به دیتابیس User ID (شناسه ی کاربری) را بر روی "testuser" و گذرواژه را بر روی "test123" تنظیم کرده اید.
- ماژول MySQLdb به طور کامل بر روی دستگاه مورد نظر نصب شده است.
- با مفاهیم پایه و ابتدایی دیتابیس MySQL آشنایی کافی داشته باشید.

ماژول

ذیل مثالی را مشاهده می کنید که در آن توسعه دهنده با زبان پایتون به دیتابیس رابطه ای MySQL به نام "TESTDB" متصل می شود.

```
#!/usr/bin/python
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# execute SQL query using execute() method.
cursor.execute("SELECT VERSION()")
# Fetch a single row using fetchone() method.
data = cursor.fetchone()
print "Database version : %s " % data
# disconnect from server
db.close()
```

خروجی اسکریپت فوق در دستگاه مبتنی بر Linux به صورت زیر می باشد.

Database version : 5.0.45

زمانی که اتصال به دیتابیس یا منبع داده ای مورد نظر با موفقیت انجام می شود، یک آبجکت Connection در خروجی بازگردانی شده و متعاقبا داخل آبجکت db جهت استفاده در آینده ذخیره می گردد. در غیر این صورت مقدار db برابر None قرار داده خواهد شد. آبجکت db سپس جهت اعلان و آماده سازی آبجکت cursor استفاده می شود. حال به منظور اجرای دستورهای درخواست داده و پرس و جو از دیتابیس، متد execute() بر روی آبجکت cursor فراخوانی می شود. در پایان، پیش از خروج از دیتابیس، اتصال به دیتابیس قطع شده و منابع مورد استفاده آزاد می شوند.

ایجاد جدول دیتابیس

پس از اینکه اتصال به دیتابیس برقرار شد، توسعه دهنده می تواند اقدام به ساخت جدول و درج سطر در جداول دیتابیس نماید. برای این منظور لازم است متد execute را بر روی آبجکت cursor صدا بزند.

مثال

در زیر یک جدول به نام EMPLOYEE ایجاد می کنیم:

```
#!/usr/bin/python
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
FIRST_NAME  CHAR(20) NOT NULL,
LAST_NAME  CHAR(20),
AGE INT,
SEX CHAR(1),
INCOME FLOAT )"""
cursor.execute(sql)
# disconnect from server
db.close()
```

عملیات INSERT

این عملیات زمانی اجرا می شود که توسعه دهنده بخواهد سطر و رکورد جدید در جدول دیتابیس جاری درج نماید.

مثال

مثال زیر دستور INSERT زبان SQL را برای ایجاد رکورد جدید در جدول EMPLOYEE اجرا می کند:

```
#!/usr/bin/python
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
LAST_NAME, AGE, SEX, INCOME)
VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
# Execute the SQL command
cursor.execute(sql)
# Commit your changes in the database
db.commit()
except:
# Rollback in case there is any error
db.rollback()
# disconnect from server
db.close()
```

مثال فوق را می توان جهت تولید Query های SQL به صورت dynamic (در زمان اجرا) به صورت زیر نوشت:

```
#!/usr/bin/python
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = "INSERT INTO EMPLOYEE(FIRST_NAME, \
LAST_NAME, AGE, SEX, INCOME) \
VALUES ('%s', '%s', '%d', '%c', '%d' )" % \
('Mac', 'Mohan', 20, 'M', 2000)
try:
# Execute the SQL command
cursor.execute(sql)
# Commit your changes in the database
db.commit()
except:
# Rollback in case there is any error
db.rollback()
# disconnect from server
db.close()
```

مثال

تکه کد زیر روش دیگری از درج داده در سطر است که در آن شما می توانید پارامترها را به صورت مستقیم به متد execute ارسال کنید:

```
.....
user_id = "test123"
password = "password"
con.execute('insert into Login values("%s", "%s")' % \
(user_id, password))
.....
```

عملیات خواندن داده ها (READ)

عملیات READ منحصرا اطلاعات مفیدی را از دیتابیس واکشی می کند.

پس از برقرار اتصال به دیتابیس، می توان از آن جهت درخواست داده های مورد نظر Query گرفت.

دو متد ()fetchone و ()fetchall نیز برای همین منظور تعبیه شده اند.

- ()fetchone: این متد همان طور که از اسم آن پیدا است، تنها یک رکورد یا سطر را در خروجی برمی گرداند. در واقع متد حاضر سطر بعدی از میان مجموعه سطرهای داده (result set خروجی کوئری) را بازگردانی می نماید. زمانی که توسعه دهنده با استفاده از cursor از دیتابیس کوئری می گیرد، خروجی یک آبجکت result set (مجموعه سطرهای داده) می باشد.
- ()fetchall: متد جاری قادر است همزمان چندین مقدار را ازدیتابیس واکشی کند. این متد تمامی سطرهای موجود در مجموعه سطرهای داده یا آبجکت result set را بازیابی می کند. اگر برخی از سطرها قبلا از دیتابیس استخراج شده باشد، در آن صورت باقی سطرها از آبجکت result set واکشی می شود.
- rowcount: این المان یک attribute فقط خواندنی (read-only) است و تعدد سطرهایی که تحت تاثیر متد ()execute قرار گرفتند را بازمی گرداند.

مثال

procedure زیر تمامی سطرهای موجود در دیتابیس را از جدول EMPLOYEE که مقدار فیلد income آن بیشتر از 1000 می باشد را به عنوان خروجی کوئری بازمی گرداند:

```
#!/usr/bin/python
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = "SELECT * FROM EMPLOYEE \
WHERE INCOME > '%d'" % (1000)
try:
# Execute the SQL command
cursor.execute(sql)
# Fetch all the rows in a list of lists.
results = cursor.fetchall()
for row in results:
fname = row[0]
lname = row[1]
age = row[2]
sex = row[3]
income = row[4]
# Now print fetched result
print "fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \
(fname, lname, age, sex, income )
except:
print "Error: unable to fecth data"
# disconnect from server
db.close()
```

خروجی زیر را برمی گرداند:

```
fname=Mac, lname=Mohan, age=20, sex=M, income=2000
```

عملیات UPDATE و بروز رسانی داده ها

زمانی که عملیات UPDATE بر روی دیتابیس اجرا می شود، یک یا چندین سطر موجود در این دیتابیس با داده های جدید بروز رسانی می شوند.

procedure و قطعه کدی که در زیر مشاهده می کنید، تمامی رکوردهایی که مقدار فیلد SEX آن ها 'M' می باشد را بروز رسانی می کند.در مثال جاری، مقدار فیلد AGE تمامی مردها را به میزان یک سال افزایش می دهیم.

مثال

```
#!/usr/bin/python
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1
WHERE SEX = '%c'" % ('M')
try:
# Execute the SQL command
cursor.execute(sql)
# Commit your changes in the database
db.commit()
except:
# Rollback in case there is any error
db.rollback()
# disconnect from server
db.close()
```

عملیات DELETE و حذف رکورد از دیتابیس

عملیات DELETE زمانی استفاده می شود که لازم باشد یک یا چند رکورد از دیتابیس مورد نظر پاک شوند. کد حاضر تمامی رکوردهای جدول EMPLOYEE که مقدار فیلد AGE آن ها بیش از 20 می باشد را حذف می نماید:

مثال

```
#!/usr/bin/python
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
# Execute the SQL command
cursor.execute(sql)
# Commit your changes in the database
db.commit()
except:
# Rollback in case there is any error
db.rollback()
```

```
# disconnect from server
db.close()
```

اجرای تراکنش بر روی دیتابیس (Transactions)

تراکنش یک مکانیزم است که دیتابیس را از یک وضعیت پایدار به وضعیت سالم و پایدار دیگر انتقال می دهد. تراکنش مجموعه ای از دستورها است که یا همه ی آن ها با موفقیت اجرا می شوند یا هیچکدام انجام نمی شوند.

تراکنش دارای چهار ویژگی معروف می باشد:

- Atomicity (اصل یا همه یا هیچ): تراکنش یا کاملا و به صورت یک پکیج اجرا می شود یا هیچ اتفاقی نمی افتد.
- Consistency (اصل یکپارچگی و پایداری): یک تراکنش باید پایگاه داده را از وضعیت پایدار و مشخص به وضعیت سالم، مشخص و پایدار دیگری انتقال دهد.
- Isolation(اصل انزوا): اطمینان حاصل می کند که تراکنش هایی که به طور همزمان اجرا می شوند، بر روی یکدیگر و سلامت دیتابیس اثری نمی گذارد، گویا هر یک در انزوا و به طور جداگانه اجرا می شوند.
- Durability (اصل پایایی و ماندگاری): زمانی که یک تراکنش به صورت نهایی ثبت و به اجرا رسید (commit)، اثرشان ماندگار و پایا خواهد بود، حتی اگر سیستم دچار خرابی ناگهانی شود.

DB API 2.0 و توابع دسترسی و مدیریت دیتابیس دو متد commit یا rollback را به ترتیب جهت ثبت نهایی تراکنش و بازگردانی آن به وضعیت قبل از تراکنش ارائه می دهد.

مثال

از قبل حتما با نحوه ی پیاده سازی تراکنش آشنایی دارید. در زیر مثالی مشابه را می بینید:

```
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
# Execute the SQL command
```

```
cursor.execute(sql)
# Commit your changes in the database
db.commit()
except:
# Rollback in case there is any error
db.rollback()
```

عملیات Commit و ثبت نهایی

Commit عملیاتی است که به دیتابیس اعلان می کند که باید تمامی تغییرات خود را به صورت نهایی ثبت کرده و پس از انجام آن دیگر امکان بازگشت به وضعیت قبلی وجود ندارد.

در زیر تکه کد ساده ای را مشاهده می کنید که متد commit() را بر روی آبجکت db صدا می زند.

```
db.commit()
```

عملیات ROLLBACK و بازگشت به وضعیت قبلی

اگر از تغییرات ثبت شده رضایت کامل ندارید، می توانید دیتابیس را به وضعیت قبل از انجام تراکنش بازگردانید. برای این منظور کافی است متد rollback() را بر روی آبجکت db فراخوانی نمایید.

```
db.rollback()
```

قطع اتصال به دیتابیس (متد close())

جهت بستن اتصال به دیتابیس کافی است متد close() را به صورت زیر فراخوانی نمایید:

```
db.close()
```

اگر اتصال به دیتابیس با فراخوانی متد close() بسته شود، در آن صورت تمامی تراکنش های انجام نشده به صورت نهایی، توسط DB به وضعیت قبلی بازگردانی می شوند.

مدیریت خطاها

خطاها بر اثر عوامل مختلفی رخ می دهند. برخی از خطاها بر اثر خطا گرامری و اشکال در سینتکس دستور SQL اجرا شده، رخ می دهند. برخی دیگر بر اثر عدم موفقیت در اتصال (connection failure) و برخی هم به دلیل فراخوانی متد fetch بر روی دستوری که قبلا یا کاملا انجام شده و یا لغو گردیده، اتفاق می افتند.

DB API تعدادی خطا اعلان می کند که باید در هر ماژولی تعریف شده باشد. این خطاها (exceptions) در جدول زیر لیست شده اند:

خطا	شرح
Warning	برای خطاهای جزئی صادر می شود. لازم است از StandardError ارث بری نمایید.
Error	کلاس پایه برای صدور و تعریف خطا. باید از کلاس StandardError ارث بری شود.
InterfaceError	برای خطاهایی که در ماژول دیتابیس و نه خود دیتابیس رخ می دهد، بکار می رود. بایستی از کلاس Error ارث بری شود.
DatabaseError	برای خطاهایی که در دیتابیس رخ می دهد صادر می شود. بایستی از کلاس Error ارث بری شود.
DataError	کلاس ارث بری شده از DatabaseError که به خطاهای موجود در داده ها اشاره می کند.
OperationalError	کلاس ارث بری شده از DatabaseError که به خطاهایی نظیر قطع اتصال به دیتابیس اشاره می کند. این خطاها معمولا از کنترل Python scripter خارج است.

IntegrityError	کلاس ارث بری شده از DatabaseError که برای سناریوهایی تعبیه شده که در آن به اصل جامعیت ارجاعی (referential integrity) دیتابیس همچون قید های یگانگی (unique) یا کلید خارجی (foreign key constraint) لطمه وارد شده باشد.
InternalError	کلاس ارث بری شده از DatabaseError که به خطاهای داخلی ماژول دیتابیس همچون عدم وجود و فعال بودن cursor اشاره دارد.
ProgrammingError	کلاس ارث بری شده از DatabaseError که به خطاهایی نظیر انتخاب اسم غیرمجاز برای جدول اشاره داشته و مربوط به برنامه نویس می باشد.
NotSupportedError	کلاس مشتق از DatabaseError که زمانی صدا زده می شود که قابلیت فراخوانی شده، پشتیبانی نمی شود.

اسکریپت های پایتون اپلیکیشن شما می بایست تمامی این خطاها را مدیریت کند. اما لازم است قبل از بکار بردن هر کدام از این exception ها اطمینان حاصل نمایید که MySQLdb امکان پشتیبانی از آن ها را دارد.

برنامه نویسی تحت شبکه با ماژول socket پایتون (Python network programming)

پایتون دو سطح دسترسی به سرویس های تحت شبکه (network service) ارائه می دهد. در سطح پایین می توانید به قابلیت های ساده ی پشتیبانی socket در سیستم عامل دسترسی داشته باشید که به شما اجازه می دهد تا سرویس گیرنده و سرویس دهنده را برای پروتکل های اتصال گرا (connection-oriented) و غیرقابل اتصال (connectionless) پیاده سازی کنید.

علاوه بر دسترسی سطح پایین، پایتون کتابخانه هایی دارد که دسترسی سطح بالا به پروتکل های شبکه ای سطح اپلیکیشن نظیر FTP، HTTP را فراهم می آورد.

مبحث حاضر به شرح معروف ترین مفهوم در برنامه نویسی تحت شبکه، Socket Programming، می پردازد.

شرح مفهوم Socket

Socket ها در واقع endpoint های موجود در یک کانال ارتباطی دو طرفه هستند. سکوت ها می توانند در بستر یک فرایند، بین دو فرایند در دستگاه واحد و یا چندین فرایند در دستگاه های مستقر در قاره و نقاط جغرافیایی مختلف با یکدیگر تبادل داده داشته باشند. از دیدگاه kernel و هسته سیستم عامل، socket صرفا نقطه ی نهایی تبادل داده و ارتباط می باشد. از دیدگاه اپلیکیشن و برنامه ی تحت شبکه، socket یک توصیف گر و شناسه ی فایل که به آن امکان و مجوز درج و خواندن داده در / از شبکه را می دهد، قلمداد می شود. Socket ترکیبی از آدرس دستگاه (IP) و آدرس درگاه (port number) می باشد.

سوکت ها بر روی انواع کانال های ارتباطی قابل پیاده سازی می باشند که از جمله ی آن ها می توان به Unix domain socket، TCP، UDP و غیره ... اشاره کرد. کتابخانه ی socket کلاس های اختصاصی ارائه می دهد که علاوه بر انتقال داده های معمولی، Interface های از نوع generic که دیگر انواع عملیات انتقال و غیره را تحت پوشش قرار می دهد، مدیریت می نماید.

برای درک مفهوم سوکت و کار با آن، لازم است با واژگان زیر آشنا شوید:

واژه	شرح
domain	خانواده ی پروتکل هایی که به عنوان مکانیزم انتقال مورد استفاده قرار می گیرد(انتقال داده در بستر شبکه بر اساس آن ها صورت می گیرد) . این مقادیر ثوابتی همچون AF_INET، PF_INET، PF_UNIX، PF_X25 و غیره .. هستند.

type	عبارت است از نوع ارتباطاتی که بین دو endpoint برقرار می شود. این معمولا SOCK_STREAM را برای پروتکل های connection-oriented (امن و تضمین دهنده ی تحویل اطلاعات) و SOCK_DGRAM را ویژه ی پروتکل های connectionless (غیر امن با سرعت بالا که تحویل داده ها را تضمین نمی کند) شامل می شود.
protocol	به طور پیش فرض بر روی 0 تنظیم می شود، این مفهوم غالبا جهت معرفی نوع دیگر از پروتکل داخل یک domain و type بکار می رود.
hostname	شناسه و اسم اینترفیس شبکه است: • یک رشته که می تواند اسم سرویس دهنده (hostname)، آدرس IP نسخه ی 4، آدرس IPV6 (آدرس IP ورژن 6) با ساختار نگارشی دو نقطه باشد. • یک رشته "<broadcast>" که آدرس INADDR_BROADCAST را تعریف می کند. • یک رشته با طول صفر که INADDR_ANY را تعریف می کند یا • یک عدد صحیح اختصاص داده شده به hostname که معرف یک سیستم در آن شبکه است.
port	هر سرویس دهنده به کلاینت هایی که یک یا چند پورت را صدا می زنند، گوش می دهد. پورت می تواند شماره ی پورت Fixnum باشد، یک رشته دربردارنده ی شماره ی پورت یا اسم سرویس باشد.

ماژول Socket

به منظور ایجاد یک Socket، لازم است تابع socket.socket() در ماژول socket را فراخوانی نمایید که سینتکس و دستور کلی آن به صورت زیر می باشد:

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

در زیر شرح هر یک از این پارامترها را مشاهده می کنید:

- socket_family: این پارامتر، همان طور که در بالا توضیح داده شد، می تواند AF_UNIX یا AF_INET باشد.
- socket_type: این پارامتر می تواند یا SOCK_STREAM و یا SOCK_DGRAM باشد.
- protocol: این پارامتر اختیاری بوده و به صورت پیش فرض بر روی 0 تنظیم می شود.

پس از تعریف آبجکت socket، می توانید با استفاده از توابع لازم، برنامه های سمت سرویس دهنده و سمت سرویس گیرنده ی خود را تعریف نمایید. جداول زیر لیست توابع لازم برای این منظور را معرفی می کند:

Server Socket Methods (متدهای به مربوط به سمت سرویس دهنده ی از ماژول socket)

متد	شرح
s.bind()	این متد آدرس (hostname یا اسم سرویس دهنده، جفت آدرس پورت یاport number pair) را به socket به صورت دو طرفه وصل می کند.
s.listen()	این متد یک گوش فرادهنده (Listener) به TCP تنظیم و راه اندازی می کند. در واقع این متد به
s.accept()	این متد درخواست اتصال به سرویس دهنده را می پذیرد و به عبارتی ارتباط معلق را به سرور معرفی می کند.

متدهای ماژول socket مربوط به سمت سرویس گیرنده

متد	شرح
s.connect()	این متد اتصال به سرویس دهنده ی را بر اساس TCP راه اندازی می کند.

متدهای کلی ماژول socket

متد	شرح
s.recv()	این متد پیغام TCP را دریافت می کند.
s.send()	متد حاضر پیغام TCP را ارسال می کند.
s.recvfrom()	متد جاری پیغام UDP را دریافت می کند.
s.sendto()	این متد پیغام UDP را ارسال می کند.
s.close()	این متد socket را می بندد.
socket.gethostname()	اسم سرویس دهنده (hostname) را در خروجی برمی گرداند.

نوشتن فایل مربوط به بخش سرویس دهنده / پیاده سازی بخش مربوط به سرویس دهنده (writing server)

جهت پیاده سازی بخش مربوط به سمت سرور، تابع socket کپسوله سازی شده در ماژول socket را فراخوانی کرده و ابتدا یک آبجکت socket می سازیم. سپس به واسطه ی این آبجکت، دیگر توابع لازم را جهت تنظیم و راه اندازی سرویس دهنده ی socket صدا می زنیم.

حال متد bind(hostname, port) را جهت مشخص کردن یک port برای سرویس خود در دستگاه میزبان یا سرویس دهنده فراخوانی نمایید.

سپس، متد accept() را بر روی آبجکت s (آبجکت ساخته شده ی socket) جهت معرفی ارتباط معلق به ماشین سرور فراخوانی می کنید. این متد صبر می کند که سرویس گیرنده به port یا آدرس درگاه تعیین شده، متصل شود و متعاقبا آبجکت connection را که نشانگر اتصال آن سرویس گیرنده (کلاینت) است در خروجی برمی گرداند.

```
#!/usr/bin/python          # This is server.py file
import socket              # Import socket module
s = socket.socket()        # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345               # Reserve a port for your service.
s.bind((host, port))       # Bind to the port
s.listen(5)                # Now wait for client connection.
while True:
c, addr = s.accept()     # Establish connection with client.
print 'Got connection from', addr
c.send('Thank you for connecting')
c.close()                # Close the connection
```

پیاده سازی بخش مربوط به سرویس گیرنده / نوشتن فایل سرویس گیرنده (Client)

در این بخش از آموزش، یک اپلیکیشن ساده سمت سرویس گیرنده می نویسیم که اتصال به پورت معین 12345و دستگاه سرویس (سرویس دهنده) را فراهم می آورد.

به راحتی می توانید یک کلاینت یا سرویس گیرنده ی socket به واسطه ی تابع مربوطه از ماژول socket ایجاد کرد.

متد (socket.connect(hosname, port یک اتصال بر اساس پروتکل TCP به hostname (دستگاه سرویس دهنده ی مربوطه بر اساس شماره ی port) باز می کند. این متد بر اساس اسن ماشین سرور و آدرس پورت اتصال را جهت تبادل داده برقرار می نماید.

پس از باز کردن socket، می توانید اطلاعات آن را مانند سایر آبجکت های IO بخوانید. لازم است در پایان، سوکت را بسته و اتصال را خاتمه می دهید.

کد زیر همان طور که مشاهده می کنید، بخش مربوط به سرویس گیرنده و در واقع یک کلاینت ساده است که به دستگاه سرویس دهنده و شماره درگاه مربوطه وصل شده، تمامی داده های مورد نیاز را از socket می خواند و در نهایت با فراخوانی تابع close() بر روی آبجکت s، سوکت را بسته و اتصال را خاتمه می دهد.

```
#!/usr/bin/python          # This is client.py file
import socket              # Import socket module
s = socket.socket()        # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345               # Reserve a port for your service.
s.connect((host, port))
print s.recv(1024)
s.close                    # Close the socket when done
```

حال جهت مشاهده ی خروجی، ابتدا فایل server.py زیر را در پس زمینه اجرا و سپس فایل client.py نام برده را اجرا کنید.

```
# Following would start a server in background.
$ python server.py &
# Once server is started run client as follows:
$ python client.py
```

خروجی زیر را برمی گرداند:

```
Got connection from ('127.0.0.1', 48437)
Thank you for connecting
```

ماژول های برنامه نویسی تحت شبکه برای Python / Python Internet modules

در زیر لیستی از ماژول های مهم و پرکاربرد پایتون در زمینه ی برنامه نویسی تحت شبکه را مشاهده می کنید.

ماژول پایتون	شماره ی درگاه یا پورت	کاربرد و مورد استفاده	پروتکل
httplib, urllib, xmlrpclib	80	Web pages (برای اپلیکیشن های تحت وب/پروتکل ارسال اطلاعات بین سرور، کلاینت)	HTTP
nntplib	119	(پروتکل دسترسی به گروه های خبری)	NNTP
ftplib, urllib	20	File transfers (انتقال فایل)	FTP
smtplib	25	Sending email (ارسال ایمیل)	SMTP
poplib	110	Fetching email (واکشی ایمیل)	POP3
imaplib	143	Fetching email (واکشی ایمیل)	IMAP4

Telnet	Command lines (پنجره یا خط فرمان جهت اتصال به سرور راه دور)	23	telnetlib
Gopher	Document transfers ارسال فایل و سند	70	gopherlib, urllib

ارسال ایمیل از طریق پروتکل SMTP با پایتون

Simple Mail Transfer Protocol یا به اختصار SMPT (پروتکل ارسال و انتقال ایمیل) یک پروتکل است که ارسال ایمیل و آدرس دهی (routing) آن بین سرویس دهنده های ایمیل را مدیریت می کند.

پایتون ماژولی به نام smtplib در اختیار توسعه دهنده قرار می دهد که یک آبجکت حاوی اطلاعات session (اطلاعات جلسه ی کاری کاربر یا client session object) را در خود به صورت کپسوله داشته و می توان از آن برای ارسال ایمیل به هر دستگاه آنلاینی که listener daemon (برنامه ی ای که به رخدادها گوش داده و در پس زمینه فعالیت می کند) SMTP یا ESMTP بر روی آن فعال است، استفاده نمود.

در زیر نحوه ی ساخت یک آبجکت SMTP ساده که بعده ها جهت ارسال ایمیل مورد استفاده قرار می گیرد، را مشاهده می کنید:

```
import smtplib
smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

در زیر شرح کاربرد هر یک از پارامترهای عنوان شده در قطعه کد بالا را مشاهده می کنید:

- host : پارامتر جاری همان میزبان یا هاستی است که به عنوان سرویس دهنده ی SMTP شما ایفای نقش می کند (SMTP server شما بر روی آن اجرا می شود). شما می توانید مقدار این پارامتر را آدرس IP میزبان یا اسم دامنه همچون tahlildadeh.com تنظیم نمایید. استفاده از این آرگومان اختیاری است.

- port: در صورت مقداردهی آرگومان اول، لازم است یک پورت یا شماره ی درگاه نیز مشخص نمایید که SMPT Server به آن گوش می دهد. شماره ی این پورت معمولا 25 می باشد.

- local_hostname: چنانچه SMPT Server شما بر روی دستگاه محلی (کامپیوتر شخصی) مستقر و فعال باشد، در آن صورت کافی است مقدار این پارامتر را localhost قرار دهید.

در آبجکت SMPT متدی تعبیه شده به نام sendmail که اغلب، توسعه دهنده با استفاده از آن کار عملیات ارسال پیغام مورد نظر را به انجام می رساند. متد نام برده در کل سه پارامتر ورودی دریافت می کند که به شرح زیر می باشند:

- sender: یک مقدار رشته ای دربردارنده ی آدرس ارسال کننده ی پیغام.
- receivers: لیستی از رشته ها که هریک مختص به دریافت کننده ی مجزا می باشد.
- message: یک پیغام به صورت رشته و فرمت دهی شده بر اساس مشخصات و قواعد RFC ها.

مثال

در زیر یک اسکریپت ساده ی پایتون را مشاهده می کنید که ایمیلی را ارسال می کند.

```
#!/usr/bin/python
import smtplib
sender = 'from@fromdomain.com'
receivers = ['to@todomain.com']
message = """From: From Person <from@fromdomain.com>
To: To Person <to@todomain.com>
Subject: SMTP e-mail test
This is a test e-mail message.
"""
try:
smtpObj = smtplib.SMTP('localhost')
```

```
smtpObj.sendmail(sender, receivers, message)
print "Successfully sent email"
except SMTPException:
print "Error: unable to send email"
```

در تمرین جاری، یک ایمیل ساده داخل متغیر message و کوتیشن سه تایی درج گردید و همان طور که می بینید، هدرها به روش صحیح فرمت دهی شده اند. هر ایمیل، سه هدر به ترتیب From، To و Subject که به وسیله ی ویرگول از هم و به وسیله ی خط سفید از بدنه ی پیغام جدا شده اند را شامل می شود.

به منظور ارسال ایمیل، ابتدا با استفاده از smtpObj به SMPT Server (سرویس دهنده ی سرور) مستقر بر روی دستگاه محلی (local) وصل شوید، سپس متد sendmail را فراخوانی کرده و پیغام، آدرس فرم و آدرس مقصد را به عنوان پارامتر به این متد ارسال نمایید (اگرچه فرم و آدرس داخل خود ایمیل گنجانده شده، با این حال از این مقدارها همیشه برای آدرس دهی یا route ایمیل استفاده نمی شود).

چنانچه شما برای ارسال ایمیل از SMPT Server که بر روی دستگاه شما (local) شما نصب و اجرا شده، استفاده نمی کنید، در آن صورت می توانید با استفاده از smtplib client به یک سرور SMPT راه دور متصل شوید. برای این منظور لازم است ارائه دهنده ی ایمیل جزئیات و اطلاعات mail server خروجی دهنده را در اختیار شما قرار داده باشد و شما نیز آن ها را به صورت زیر بکار ببرید، مگر اینکه برای ارسال ایمیل از یک سرویس آماده همچون Hotmail و Yahoo استفاده نمایید که در آن صورت نیازی به این اطلاعات نیست.

```
smtplib.SMTP('mail.your-domain.com', 25)
```

ارسال فایل ایمیل به صورت HTML با استفاده از Python

زمانی که توسعه دهنده یک پیغام متنی را با استفاده از Python ارسال می کند، تمامی محتوای فایل به عنوان متن ساده در نظر گرفته می شود. به عبارت دیگر حتی اگر تگ های HTML را در پیغام متنی بگنجانید، باز هم محتوای فایل به صورت متن ساده نمایش داده شده و تگ های HTML بر اساس گرامر زبان نشانه گذاری HTML فرمت دهی نمی شوند. پایتون امکانی را در اختیار توسعه

دهنده قرار می دهد که به واسطه ی آن می توان یک پیغام HTML را به صورت یک فایل واقعی HTML ارسال کرد.

به هنگام ارسال یک ایمیل، می توان نوع فایل (Mime version)، نوع محتوا و مجموعه کاراکتری که باید به صورت یک ایمیل HTML ارسال شود را مشخص نمایید.

مثال

در زیر کدی را مشاهده می کنید که محتوایی با فرمت HTML را به صورت ایمیل ارسال می کند:

```
#!/usr/bin/python
import smtplib
message = """From: From Person <from@fromdomain.com>
To: To Person <to@todomain.com>
MIME-Version: 1.0
Content-type: text/html
Subject: SMTP HTML e-mail test
This is an e-mail message to be sent in HTML format
<b>This is HTML message.</b>
<h1>This is headline.</h1>
"""
try:
smtpObj = smtplib.SMTP('localhost')
smtpObj.sendmail(sender, receivers, message)
print "Successfully sent email"
except SMTPException:
print "Error: unable to send email"
```

ارسال محتوا همراه با ایمیل (پیاده سازی قابلیت الصاق محتوا و (Attachment

جهت ارسال ایمیل با محتوای مختلط، لازم است مقدار هدر Content-type را برابر multipart/mixed قرار دهید. پس از آن، متن و محتوای الصاقی (attachment) را در boundaries دقیقا اعلان کنید.

برای تعریف boundary، دو خط تیره (هایفن) و یک عدد منحصربفرد درج کنید که این بخش نباید در بدنه ی ایمیل یا بخش پیغام ظاهر شود. سپس یک boundary نهایی درج می کنید که نشانگر بخش پایانی ایمیل بوده و باید به دو خط تیره ختم شود.

فایل های الصاق شده باید قبل از ارسال، به وسیله ی تابع ("m")pack بر مبنای الگوریتم و روش کدگذاری base64 رمزنگاری شوند.

مثال

در مثال زیر، فایل /tmp/test.txt به عنوان محتوای الصاقی همراه با ایمیل ارسال می شود:

```
#!/usr/bin/python
import smtplib
import base64
filename = "/tmp/test.txt"
# Read a file and encode it into base64 format
fo = open(filename, "rb")
filecontent = fo.read()
encodedcontent = base64.b64encode(filecontent)  # base64
sender = 'webmaster@tutorialpoint.com'
reciever = 'amrood.admin@gmail.com'
marker = "AUNIQUEMARKER"
body ="""
This is a test email to send an attachement.
"""
# Define the main headers.
part1 = """From: From Person <me@fromdomain.net>
To: To Person <amrood.admin@gmail.com>
Subject: Sending Attachement
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=%s
--%s
""" % (marker, marker)
# Define the message action
part2 = """Content-Type: text/plain
Content-Transfer-Encoding:8bit
%s
--%s
""" % (body,marker)
# Define the attachment section
part3 = """Content-Type: multipart/mixed; name=\"%s\"
Content-Transfer-Encoding:base64
Content-Disposition: attachment; filename=%s
%s
--%s--
""" %(filename, filename, encodedcontent, marker)
message = part1 + part2 + part3
try:
smtpObj = smtplib.SMTP('localhost')
smtpObj.sendmail(sender, reciever, message)
print "Successfully sent email"
```

```
except Exception:
print "Error: unable to send email"
```

برنامه نویسی موازی و پردازش همزمان با زبان پایتون (multithreaded programming)

اجرای همزمان چندین thread به منزله ی اجرای همزمان چندین برنامه در آن واحد است که مزایای زیر را به دنبال دارد:

- چندین thread که در بستر یک پردازه یا فرایند (process) اجرا شده و data space یکسانی را دارند، می توانند داده ها را بهتر با یکدیگر به اشتراک گذاشته و بایکدیگر تعامل بهتری داشته باشند، نسبت به زمانی که این thread ها در فرایندهای مجزا قرار دارند.
- thread ها که گاهی پردازه یا فرایندهای سبک نیز نامیده می شوند سربار و memory overhead کمتری نسبت به فرایندهای واقعی داشته و کم هزینه تر می باشند.

هر thread یک نقطه ی آغاز،یک ترتیب یا توالی اجرا و یک نقطه ی پایان دارد. علاوه بر آن، یک instruction pointer دارد که دقیقا مشخص می کند برنامه در کجای بستر (context) جاری در حال اجرا بود و به کدام مرحله و نقطه اجرای دستور رسیده است.

- می توان thread را مختل یا متوقف (pre-empt) کرد.
- می توان thread را در حالی که دیگر thread ها فعال هستند، به طور موقت به حالت تعلیق درآورد. از این کار تحت عنوان yielding نیز یاد می شود.

راه اندازی و اجرای thread جدید

جهت آغاز یک thread جدید، بایستی متد زیر که داخل ماژول thread کپسوله شده را فراخوانی نمایید:

```
thread.start_new_thread ( function, args[, kwargs] )
```

به وسیله ی این متد می توانید به روش سریع و کارا در هر دو محیط ویندوز و لینوکس thread های جدید ایجاد نمایید.

متد مورد نظر بلافاصله بازگشته و thread فرزند آغاز می شود که متعاقبا function را با پارامتر args صدا می زند. زمانی که function به return می رسد، thread خاتمه می یابد.

در این تابع، پارامتر args مجموعه ی چندتایی از آرگومان ها (tuple) است. اگر می خواهید تابع را بدون آرگومان صدا بزنید، بایستی یک tuple خالی به عنوان پارامتر ارسال کنید. kwargs یک آرگومان از نوع dictionary تشکیل شده از کلیدواژه ها بوده و استفاده از آن اختیاری است.

Example

```
#!/usr/bin/python
import thread
import time
# Define a function for the thread
def print_time( threadName, delay):
count = 0
while count < 5:
time.sleep(delay)
count += 1
print "%s: %s" % ( threadName, time.ctime(time.time()) )
# Create two threads as follows
try:
thread.start_new_thread( print_time, ("Thread-1", 2, ) )
thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
print "Error: unable to start thread"
while 1:
pass
```

خروجی:

```
Thread-1: Thu Jan 22 15:42:17 2009
Thread-1: Thu Jan 22 15:42:19 2009
Thread-2: Thu Jan 22 15:42:19 2009
Thread-1: Thu Jan 22 15:42:21 2009
Thread-2: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:25 2009
Thread-2: Thu Jan 22 15:42:27 2009
Thread-2: Thu Jan 22 15:42:31 2009
Thread-2: Thu Jan 22 15:42:35 2009
```

اگرچه توصیه می شود که برای پردازش موازی سطح پایین (low-level threading) از ماژول thread استفاده نمایید، با این وجود ماژول مزبور در مقایسه با ماژول جدید برنامه نویسی موازی (threading module) از قابلیت های بسیار کمتری برخوردار است.

ماژول Threading

ماژول جدیدتری که برای برنامه نویسی موازی همراه با ویرایش 2.4 زبان Python ارائه شد، قابلیت های بسیار بیشتر و سطح بالاتری جهت بهره گیری از پردازش موازی و thread ها در اختیار برنامه نویس قرار داد.

ماژول threading علاوه بر تمامی توابع ماژول قبلی (thread)، تعدادی متد نوین و کارای دیگر جهت پیاده سازی مفهوم برنامه نویسی موازی ارائه می دهد.

- threading.activeCount(): تعداد آبجکت های thread که فعال و در حال اجرا هستند را بازگردانی می نماید.
- threading.currentThread(): thread جاری را به عنوان خروجی برمی گرداند.
- لیستی از تمامی آبجکت های thread که هم اکنون فعال هستند را بازمی گرداند.

علاوه بر متدها، ماژول نام برده کلاس Thread را شامل می شود که threading و پردازش موازی را پیاده سازی می کند. متدهایی که که کلاس Thread در اختیار توسعه دهنده قرار می دهد، به شرح زیر می باشد:

- run(): متد جاری در واقع entry point یا نقطه ی شروع اجرای thread می باشد.
- start(): thread را با فراخوانی تابع run() راه اندازی و اجرا می کند.
- join([time]): متد join() منتظر می ماند تا thread ها خاتمه یابند.
- isAlive(): متد حاضر بررسی می کند آیا یک thread هنوز در حل اجرا است یا خیر.
- getName(): متد جاری اسم آبجکت thread را بازیابی می کند.
- setName(): این متد اسم thread را مقداردهی می کند.

ایجاد آبجکت جدید Thread از ماژول Threading

به منظور پیاده سازی یک thread جدید از ماژول threading، کافی است مراحل زیر را دنبال نمایید:

- یک کلاس مشتق از کلاس Thread ایجاد نمایید (از آن ارث بری کنید).
- متد __init__(self [,args]) را جهت تزریق آرگومان های بیشتر به کلاس، بازنویسی (override) نمایید.
- در پایان، متد run(self [,args]) را پیاده سازی و در بدنه ی آن مشخص کنید که thread به هنگام اجرا چه عملیاتی را انجام دهد.
- پس از ایجاد کلاس مشتق (ارث بری) از Thread، می توانید یک نمونه از آن ایجاد کرده و سپس با فراخوانی متد start() یک thread یا ریزپردازه ی دیگر آغاز نمایید که متعاقبا متد run() را صدا می زند.

مثال

```
#!/usr/bin/python
import threading
import time
exitFlag = 0
class myThread (threading.Thread):
def __init__(self, threadID, name, counter):
threading.Thread.__init__(self)
self.threadID = threadID
self.name = name
self.counter = counter
def run(self):
print "Starting " + self.name
print_time(self.name, self.counter, 5)
print "Exiting " + self.name
def print_time(threadName, delay, counter):
while counter:
if exitFlag:
threadName.exit()
time.sleep(delay)
print "%s: %s" % (threadName, time.ctime(time.time()))
counter -= 1
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)
# Start new Threads
thread1.start()
thread2.start()
print "Exiting Main Thread"
```

زمانی که کد فوق به اجرا گذاشته می شود، خروجی به صورت زیر خواهد بود:

```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1: Thu Mar 21 09:10:03 2013
Thread-1: Thu Mar 21 09:10:04 2013
Thread-2: Thu Mar 21 09:10:04 2013
Thread-1: Thu Mar 21 09:10:05 2013
Thread-1: Thu Mar 21 09:10:06 2013
Thread-2: Thu Mar 21 09:10:06 2013
Thread-1: Thu Mar 21 09:10:07 2013
Exiting Thread-1
Thread-2: Thu Mar 21 09:10:08 2013
Thread-2: Thu Mar 21 09:10:10 2013
Thread-2: Thu Mar 21 09:10:12 2013
Exiting Thread-2
```

همزمان سازی thread ها

ماژول threading که در بالا به آن اشاره کردیم، یک مکانیزم اعمال قفل با قابلیت پیاده سازی آسان در اختیار توسعه دهنده قرار می دهد که به واسطه ی آن می توان به راحتی thread ها و اجرای آن ها را هماهنگ ساخت. جهت ساخت و اعمال قفل جدید کافی است متد ()Lock فراخوانی شود که در خروجی نمونه ی جدید از آبجکت lock را بازگردانی می نماید.

متد acquire(blocking) از نمونه ی (آبجکت) جدید lock، این قابلیت را دارد که با مدیریت thread ها آن ها را به طور همزمان (موازی) اجرا کند. به واسطه ی پارامتر اختیاری blocking توسعه دهنده قادر خواهد بود کنترل اینکه آیا قفل بر روی thread اعمال شود یا خیر را بدست گیرد.

زمانی که مقدار پارامتر blocking برابر 0 باشد، اگر قفل یا lock بر روی thread اعمال شد،متد با 1 و اگر نشد با 0 برمی گردد. هنگامی که blocking روی 1 تنظیم شده باشد، thread مسدود (block) شده و منتظر می ماند تا lock آزاد شود.

زمانی که متد ()release از نمونه ی جدید lock صدا خورده می شود، lock یا قفل اعمال شده بر روی thread که دیگر کاربردی ندارد، آزاد می شود.

مثال

```
#!/usr/bin/python
```

```
import threading
import time
class myThread (threading.Thread):
def __init__(self, threadID, name, counter):
threading.Thread.__init__(self)
self.threadID = threadID
self.name = name
self.counter = counter
def run(self):
print "Starting " + self.name
# Get lock to synchronize threads
threadLock.acquire()
print_time(self.name, self.counter, 3)
# Free lock to release next thread
threadLock.release()
def print_time(threadName, delay, counter):
while counter:
time.sleep(delay)
print "%s: %s" % (threadName, time.ctime(time.time()))
counter -= 1
threadLock = threading.Lock()
threads = []
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)
# Start new Threads
thread1.start()
thread2.start()
# Add threads to thread list
threads.append(thread1)
threads.append(thread2)
# Wait for all threads to complete
for t in threads:
t.join()
print "Exiting Main Thread"
```

خروجی:

```
Starting Thread-1
Starting Thread-2
Thread-1: Thu Mar 21 09:11:28 2013
Thread-1: Thu Mar 21 09:11:29 2013
Thread-1: Thu Mar 21 09:11:30 2013
Thread-2: Thu Mar 21 09:11:32 2013
Thread-2: Thu Mar 21 09:11:34 2013
Thread-2: Thu Mar 21 09:11:36 2013
Exiting Main Thread
```

پیاده سازی queue و قرار دادن آیتم ها در صف بر اساس اولویت در پردازش موازی (Multithreaded Priority Queue)

ماژول Queue به توسعه دهنده این امکان را می دهد تا یک آبجکت queue جدید ایجاد کند که همزمان چندین آیتم را به طور صف بندی شده در خود کپسوله می نماید. برای مدیریت Queue می توانید از توابع زیر استفاده نمایید:

- get(): متد جاری یک آیتم را حذف و از queue بازیابی می کند.
- put(): این متد یک آیتم جدید را به queue اضافه می کند.
- qsize(): تعداد آیتم هایی که در زمان حاضر داخل صف قرار دارند را به عنوان خروجی بازمی گرداند.
- empty(): چنانچه صف یا آبجکت queue خالی بود، مقدار بولی True و در غیر این صورت False را بازمی گرداند.
- full(): چنانچه آبجکت queue ظرفیت خالی نداشت، مقدار بولی True و در غیر این صورت False را بازگردانی می نماید.

مثال

```
#!/usr/bin/python
import Queue
import threading
import time
exitFlag = 0
class myThread (threading.Thread):
def __init__(self, threadID, name, q):
threading.Thread.__init__(self)
self.threadID = threadID
self.name = name
self.q = q
def run(self):
print "Starting " + self.name
process_data(self.name, self.q)
print "Exiting " + self.name
def process_data(threadName, q):
while not exitFlag:
queueLock.acquire()
if not workQueue.empty():
data = q.get()
queueLock.release()
```

```
            print "%s processing %s" % (threadName, data)
        else:
            queueLock.release()
        time.sleep(1)
threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workQueue = Queue.Queue(10)
threads = []
threadID = 1
# Create new threads
for tName in threadList:
    thread = myThread(threadID, tName, workQueue)
    thread.start()
    threads.append(thread)
    threadID += 1
# Fill the queue
queueLock.acquire()
for word in nameList:
    workQueue.put(word)
queueLock.release()
# Wait for queue to empty
while not workQueue.empty():
    pass
# Notify threads it's time to exit
exitFlag = 1
# Wait for all threads to complete
for t in threads:
    t.join()
print "Exiting Main Thread"
```

نتیجه:

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-1 processing One
Thread-2 processing Two
Thread-3 processing Three
Thread-1 processing Four
Thread-2 processing Five
Exiting Thread-3
Exiting Thread-1
Exiting Thread-2
Exiting Main Thread
```

پردازش و تفسیر فایل های XML با پایتون

XML یک زبان متن باز (open source) و قابل اجرا (portable) بر روی چندین سیستم عامل بوده که برنامه نویس به واسطه ی آن می تواند اپلیکیشن هایی طراحی نماید که توسط سایر اپلیکیشن ها، صرف نظر از زبانی که با آن نوشته شده و محیطی که بر روی آن اجرا می شوند، قابل استفاده باشد.

XML چیست؟

Extensible Markup Language یا به اختصار XML (زبان نشانه گذاری گسترش پذیر) یک زبان نشانه گذاری مشابه HTML یا SGML است. کنسرسیوم وب جهان گستر این زبان را به عنوان یک استاندارد سراسری توصیه می کند.

چنانچه اپلیکیشنی که قصد طراحی آن را دارید، داده های حجیم و سنگینی برای نگهداری ندارد، در آن صورت می توانید بدون زحمت طراحی دیتابیس و استفاده از پشتوانه ی SQL، اطلاعات برنامه ی خود را در قالب XMLذخیره نمایید.

معماری ها و توابع کتابخانه ای تحلیل گر نحوی و مفسر XML Parser (XML Parser & API)

کتابخانه ی استاندارد python تعدادی interface (الگوی پیاده سازی) معدود اما کارا جهت کار با XML در اختیار برنامه نویس قرار می دهد که در زیر به شرح آن ها می پردازیم.

دو API و توابع کتابخانه ای پرکاربرد و ساده ای که توصیه می شود عبارتند از:

- SAX (Simple API for XML): در این API، توابع Callback ای برای رخدادهای مورد نظر معرفی (register) می کنید و سپس به parser اجازه می دهید به تحلیل باقی فایل بپردازد. ابزار جاری برای شرایطی مفید می باشد که فایل های اپلیکیشن حجیم بوده و شما با محدودیت حافظه مواجه هستید. در واقع API حاضر فایل را از روی دیسک تحلیل و تفسیر می کند و به همین جهت هیچگاه کل فایل در حافظه بارگذاری نشده و آن را اشغال نمی کند.
- DOM (Document Object Model): API جاری را کنسرسیوم وب جهان گستر توصیه می کند. در این API کل محتوای فایل داخل حافظه بارگذاری و به صورت درختی یا سلسله مراتبی سازماندهی می شود و کلیه ی ویژگی های یک فایل XML را به نمایش می گذارد.

SAX قادر نیست هنگام کار با فایل های حجیم، اطلاعات را به سرعت DOM پردازش کند. از طرف دیگر، اگر منحصرا از DOM استفاده کنید، به ویژه برای پردازش فایل های کوچک و کم حجم، قطعا میزان قابل توجهی از منابع شما هدر می رود.

SAX فایل ها را با مجوز در سطح فقط خواندن باز می کند، در حالی که DOM امکان اعمال تغییرات در فایل XML را فراهم می آورد. از آنجایی که دو API مزبور مکمل یکدیگر هستند، دلیلی وجود ندارد که از هر دو در پروژه های بزرگ استفاده نکنید.

نمونه کدهای XML که در مثال های زیر استفاده کرده و به عنوان ورودی توابع پردازش و parse مورد استفاده قرار می دهیم، فایل movies.xml با محتوای زیر خواهد بود:

```
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
<type>War, Thriller</type>
<format>DVD</format>
<year>2003</year>
<rating>PG</rating>
<stars>10</stars>
<description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
<type>Anime, Science Fiction</type>
<format>DVD</format>
<year>1989</year>
<rating>R</rating>
<stars>8</stars>
<description>A schientific fiction</description>
</movie>
<movie title="Trigun">
<type>Anime, Action</type>
<format>DVD</format>
<episodes>4</episodes>
<rating>PG</rating>
<stars>10</stars>
<description>Vash the Stampede!</description>
</movie>
<movie title="Ishtar">
<type>Comedy</type>
<format>VHS</format>
<rating>PG</rating>
<stars>2</stars>
<description>Viewable boredom</description>
</movie>
```

```
</collection>
```

پردازش و تفسیر XML به وسیله ی توابع SAX

SAX یک interface حاوی مجموعه توابع استاندارد برای تحلیل و پردازش XML به روش رخداد محور (Event-driven) است. جهت استفاده از interface یاد شده برای پردازش و فایل های XML، لازم است با ارث بری (تعریف کلاس مشتق) از xml.sax.ContentHandler، کلاس ContentHandler اختصاصی خود را ایجاد نمایید.

کلاس ContentHandler اختصاصی شما در واقع می تواند تگ ها و attribute های نسخه یا نسخه های مختلف XML را مدیریت نماید. آبجکت ContentHandler تعدادی متد برای مدیریت رخدادهای مختلف parse (پردازش و تحلیل) ارائه می دهد که parser این متدها را به هنگام بارگذاری محتوای فایل XML در حافظه و پردازش آن ها، از ContentHandler فراخوانی می کند.

متدهای startDocument و endDocument هریک به ترتیب در ابتدا و انتهای فایل XML فراخوانی می شوند. اطلاعات و داده های مربوط به کاراکترهای فایل XML از طریق پارامتر text در اختیار متد character(text) قرار می گیرد.

متد ContentHandler در ابتدا و انتهای هر المان فراخوانی می شود. اگر parser در وضعیت namespace mode قرار داشته باشد، در آن صورت متدهای startElement(tag, attributes) و endElement(tag) صدا زده می شوند و در غیر این صورت متدهای مربوطه یعنی startElementNS و endElementNS فراخوانده می شوند. در اینجا منظور از tag در واقع المان tag و attributes، آبجکت Attributes است.

در زیر سایر متدهای مهم که فهم آن ها برای کار با SAX مهم می باشد را مشاهده می کنید:

متد make_parser

متد جاری یک آبجکت parser جدید ایجاد کرده و آن را در خروجی برمی گرداند. آبجکت parser که در خروجی بازگردانی می شود، اولین نوع parser است که سیستم پیدا می کند.

```
xml.sax.make_parser( [parser_list] )
```

در زیر جزئیات پارامترها به تفصیل شرح داده است:

- parser_list: آرگومان اختیاری متشکل از یک لیست از parser ها برای استفاده که تمامی آن ها بایستی متد make_parser را پیاده سازی می کند.

متد parse

متد زیر یک مفسر و تحلیل گر نحوی SAX تعریف کرده و با استفاده از آن محتوای فایل مورد نظر را parse (تفسیر و تبدیل) می کند.

```
xml.sax.parse( xmlfile, contenthandler[, errorhandler])
```

در زیر اطلاعات پارامترهای متد فوق به تفصیل شرح داده شده است:

- xmlfile: این اسم فایل XML است که محتوای آن قرار است خوانده و تفسیر شود.
- contenthandler: آبجکت ساخته شده از کلاس ContentHandler.
- این پارامتر اختیاری بوده و در صورت استفاده از آن بایستی یک آبجکت ErrorHandler از SAX باشد.

متد parseString

متد دیگری که یک تحلیل گر و مفسر SAX ایجاد کرده و رشته ی XML ارسالی را parse می کند، parseString می باشد:

```
xml.sax.parseString(xmlstring, contenthandler[, errorhandler])
```

در زیر هریک از پارامتر ها به تفصیل شرح داده شده است:

- xmlstring: اسم رشته ی XML که محتوا از آن خوانده می شود.
- contenthandler: بایستی یک آبجکت ContentHandler باشد.
- errorhandler: پارامتر اختیاری که یک آبجکت ErrorHandler از SAX می باشد.

مثال

```
#!/usr/bin/python
import xml.sax
class MovieHandler( xml.sax.ContentHandler ):
def __init__(self):
self.CurrentData = ""
self.type = ""
self.format = ""
self.year = ""
self.rating = ""
self.stars = ""
self.description = ""
# Call when an element starts
def startElement(self, tag, attributes):
self.CurrentData = tag
if tag == "movie":
print "*****Movie*****"
title = attributes["title"]
print "Title:", title
# Call when an elements ends
def endElement(self, tag):
if self.CurrentData == "type":
print "Type:", self.type
elif self.CurrentData == "format":
print "Format:", self.format
elif self.CurrentData == "year":
print "Year:", self.year
elif self.CurrentData == "rating":
print "Rating:", self.rating
elif self.CurrentData == "stars":
print "Stars:", self.stars
elif self.CurrentData == "description":
print "Description:", self.description
self.CurrentData = ""
# Call when a character is read
def characters(self, content):
if self.CurrentData == "type":
self.type = content
elif self.CurrentData == "format":
self.format = content
elif self.CurrentData == "year":
self.year = content
elif self.CurrentData == "rating":
self.rating = content
elif self.CurrentData == "stars":
self.stars = content
elif self.CurrentData == "description":
self.description = content
if ( __name__ == "__main__"):
# create an XMLReader
parser = xml.sax.make_parser()
# turn off namepsaces
```

```
parser.setFeature(xml.sax.handler.feature_namespaces, 0)
# override the default ContextHandler
Handler = MovieHandler()
parser.setContentHandler( Handler )
parser.parse("movies.xml")
```

خروجی:

```
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Year: 2003
Rating: PG
Stars: 10
Description: Talk about a US-Japan war
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Year: 1989
Rating: R
Stars: 8
Description: A schientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Stars: 10
Description: Vash the Stampede!
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Stars: 2
Description: Viewable boredom
```

پردازش و تفسیر فایل های XML با استفاده از توابع DOM

مدل شی گرای فایل، Document Object Model یا به اختصار DOM یک API و مجموع توابع کتابخانه ای است که کنسرسیوم وب جهان گستر برای دسترسی و ویرایش محتوای فایل های XML، به توسعه دهندگان توصیه می کند.

DOMبه ویژه برای اپلیکیشن هایی که لازم است به محتوای فایل XML آن به صورت رندوم دسترسی صورت گیرد، مفید می باشد. SAX به توسعه دهنده امکان دسترسی فقط به یک قسمت از فایل XML را در آن واحد می دهد. به طور مثال، هنگام دسترسی به یک المان از فایل XML، امکان دسترسی به سایر المان های فایل برای شما وجود ندارد.

ماژول xml.dom، به شما این امکان را می دهد تا یک آبجکت minidom ایجاد کرده و محتوای فایل XML را به سرعت در حافظه بارگذاری نمایید. آبجکت minidom متد کارا و ساده تری جهت ساخت درخت DOM از فایل XML در اختیار توسعه دهنده قرار می دهد.

نمونه کد زیر متد parse(file [,parser]) از آبجکت minidom را صدا زده و محتوای فایل XML را تجزیه و به آبجکت درخت DOM تبدیل می کند.

```
#!/usr/bin/python
from xml.dom.minidom import parse
import xml.dom.minidom
# Open XML document using minidom parser
DOMTree = xml.dom.minidom.parse("movies.xml")
collection = DOMTree.documentElement
if collection.hasAttribute("shelf"):
print "Root element : %s" % collection.getAttribute("shelf")
# Get all the movies in the collection
movies = collection.getElementsByTagName("movie")
# Print detail of each movie.
for movie in movies:
print "*****Movie*****"
if movie.hasAttribute("title"):
print "Title: %s" % movie.getAttribute("title")
type = movie.getElementsByTagName('type')[0]
print "Type: %s" % type.childNodes[0].data
format = movie.getElementsByTagName('format')[0]
print "Format: %s" % format.childNodes[0].data
rating = movie.getElementsByTagName('rating')[0]
print "Rating: %s" % rating.childNodes[0].data
description = movie.getElementsByTagName('description')[0]
print "Description: %s" % description.childNodes[0].data
```

خروجی:

```
Root element : New Arrivals
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
```

Rating: PG
Description: Talk about a US-Japan war
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Rating: R
Description: A schientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Description: Vash the Stampede!
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Description: Viewable boredom

ساخت و توسعه ی لایه ی UI اپلیکیشن با زبان پایتون – برنامه نویسی GUI با Python

پایتون امکانات متعددی برای ساخت و توسعه ی GUI یا لایه رابط کاربری اپلیکیشن ارائه می دهد که در زیر به پرکاربردترین و کاراترین آن ها اشاره می کنیم:

- Tkinter: به طور پیش فرض به همراه ویرایش های متعارف زبان پایتون در اختیار برنامه نویس قرار می گیرد که رابط یا interface شی گرا جهت استفاده از ابزار tk در بستر محیط پایتون فراهم می آورد. در مبحث حاضر نیز از این ابزار برای توسعه ی لایه UI اپلیکیشن بهره خواهیم گرفت.
- wxPython: عبارت است از یک سری ابزار طراحی و توسعه لایه ی UI اپلیکیشن که برای پایتون عرضه می شود. این ابزار به برنامه نویس اجازه می دهد تا برنامه های قدرتمند، کارا با رابط کاربری بسیار کارامد را به راحتی طراحی کرده و توسعه دهد. این ابزار به صورت یک

ماژول یا افزونه پیاده سازی شده که کتابخانه ی پرطرفدار wxWidgets را در خود کپسوله سازی می کند (wrap). کتابخانه ی مزبور بسیار پرکاربرد بوده، با محیط های مختلف سازگار می باشد(cross-platform) و با زبان چند منظوره ی C++ نوشته شده است.

- JPython: یک درگاه پایتون برای جاوا است که برنامه ها یا اسکریپت های پایتون می توانند به واسطه ی آن، به راحتی به کتابخانه ها و کلاس های (class library) جاوا در دستگاه محلی و میزبان خود دسترسی داشته باشند.

کتابخانه ی Tkinter

Tkinter کتابخانه ای است که برنامه نویس از آن جهت توسعه و برنامه نویسی لایه ی UI اپلیکیشن استفاده می کند. در واقع زمانی که پایتون با Tkinter ترکیب می شود، کار برنامه نویس در طراحی و برنامه نویسی رابط گرافیکی کاربر بسیار آسان شده و به سرعت قابل پیاده سازی می باشد.

Tkinter کتابخانه ی GUI یک رابط شی گرای کارا برای مجموعه ابزار برنامه نویسی Tk GUI فراهم می سازد.

طراحی یک اپلیکیشن GUI با استفاده از Tkinter امر بسیار آسانی می باشد. کافی است مراحل زیر را طی نمایید:

- ابتدا ماژول Tkinter را با استفاده از دستور import وارد نمایید.
- پنجره ی اصلی اپلیکیشن GUI را ایجاد نمایید.
- یک یا چند ابزارک رابط کاربری (widget) به آن اضافه نمایید.
- event loop (حلقه ی اصلی و کنترل کننده ی روند کلی اجرای برنامه) را پیاده سازی نمایید. این حلقه به event هایی که کاربر آن ها را فعال می کند، پاسخ داده و آن ها را مدیریت می کند.

مثال

```
#!/usr/bin/python
import tkinter
top = tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

خروجی کد را در زیر مشاهده می کنید:

ویدجت ها و ابزارک های رابط کاربری Tkinter

Tkinter چندین کنترل رابط کاربری همچون button، label و text box ارائه می دهد که برنامه نویس به واسطه ی آن لایه ی UI اپلیکیشن را طراحی و توسعه می دهد. به این کنترل ها در اصطلاح Widget یا ابزارک رابط کاربری گفته می شود.

در حال حاضر حدود 15 widget در Tkinter کپسوله شده است. در جدول ذیل تمامی این widget ها را به همراه شرح کاربرد مشاهده می کنید:

ابزارک رابط کاربری یا widget مربوطه	شرح
Button	توسعه دهنده برای ساخت و نمایش المان دکمه در UI اپلیکیشن از این widget استفاده می کند.
Canvas	کنترل Canvas به برنامه نویس این امکان را می دهد تا در نمایشگر اشکال هندسی همچون خط ساده، بیضی، چندضلعی و مستطیل ترسیم نماید.

Checkbutton	کنترل جاری برای طراحی و نمایش چندین گزینه (آیتم قابل گزینش) در قالب checkbox بکار می رود. با پیاده سازی این widget کاربر قادر خواهد بود همزمان چندین گزینه را انتخاب کند.
Entry	ابزارک رابط کاربری حاضر یک text field تک خطی در UI نمایش می دهد که مقادیر متنی از کاربر دریافت می کند.
Frame	کنترل رابط کاربری Frame به عنوان یک ظرف ایفای نقش کرده که دیگر widget ها در آن سازمان دهی شده و جای می گیرند.
Label	یک کنترل رابط کاربری که برای درج نوشته یا توضیحی مختصر برای سایر ویدجت ها ایفای نقش می کند. می تواند عکس را نیز شامل شود.
Listbox	ابزارک رابط کاربری Listbox برای ارائه و نمایش لیستی از آیتم های قابل گزینش برای کاربر مورد استفاده قرار می گیرد.
Menubutton	ابزارک رابط کاربری Menubutton جهت پیاده سازی عنصر منو در UI مورد استفاده قرار می گیرد.
Menu	ابزارک رابط کاربری جاری به منظور ارائه ی تعدادی دستور معین به کاربر در قالب یک لیست مورد استفاده قرار می گیرد. دستورات مزبور داخل Menubutton قرار می گیرند.
Message	المان رابط کاربری Message جهت پیاده سازی و نمایش text field های چند خطی در UI مورد استفاده قرار می گیرد.
Radiobutton	المان رابط کاربری حاضر جهت نمایش تعدادی گزینه در قالب Radiobutton که تنها امکان انتخاب یکی از آن ها در آن واحد وجود دارد، مورد استفاده قرار می گیرد.
Scale	المان رابط کاربری جاری جهت پیاده سازی ابزارک اسلایدر در UI اپلیکیشن بکار می رود.
Scrollbar	ابزارک Scrollbar جهت پیاده سازی یک نوارپیمایش و افزودن قابلیت راهبری به المان های رابط کاربری متعدد نظیر list box ها استفاده می شود.

Text	ابزارک رابط کاربری جاری جهت پیاده سازی قابلیت نمایش چند خط متن در UI مورد استفاده قرار می گیرد.
Toplevel	المان رابط کاربری Toplevel برای ساخت یک پنجره ی مستقل که المان والدی نداشته و فرزند المان دیگری نمی باشد، مورد استفاده قرار می گیرد.
Spinbox	ابزارک رابط کاربری Spinbox یک نوع دیگر از کنترل Entry ماژول Tkinter بوده و ظاهری مشابه کنترل مزبور دارد با این تفاوت که به کاربر اجازه می دهد تا لیستی ثابت از مقادیر را از کادر حاضر انتخاب کند.
PanedWindow	المان رابط کاربری PanedWindow ابزارکی است که نقش ظرف را ایفا کرده و قادر است تا همزمان چندین کادر را که به صورت عمودی یا افقی سازمان دهی (چیده) شده اند در خود جای دهد.
LabelFrame	یک نوع دیگر از Frame widget در ماژول Tkinter است که می تواند میزبان سایر المان های رابط کاربری باشد. این کنترل در واقع یک خط حاشیه پیرامون المان های دیگر ترسیم می کند و قادر است علاوه بر خط ساده متن نیز به عنوان حاشیه ی المان ها لحاظ نماید.
tkMessageBox	با بهره گیری از این ابزارک می توانید المان message box را جهت نمایش پیغام برای کاربر در UI پیاده سازی کنید.

Attribute ها و ویژگی های متعارف کنترل های رابط کاربری

در زیر برخی از attribute های مهم کنترل های رابط کاربری که با مقداردهی آن ها می توانید ویژگی هایی نظیر اندازه، رنگ و فونت المان ها را تنظیم نمایید، مشاهده می کنید:

- Dimensions
- Colors
- Fonts
- Anchors

- Relief styles
- Bitmaps
- Cursors

مدیریت هندسه و چیدمان المان ها

تمامی widget های Tkinter به متدهای geometry (مدیریت چیدمان المان ها) دسترسی دارند. این متدها قادر هستند که ابزارک های رابط کاربری را داخل ناحیه ی میزبان سازمان دهی کرده و چیدمان آن ها را مدیریت کند. Tkinter کلاس های زیر را جهت مدیریت چیدمان المان ها در دسترس قرار می دهد. این کلاس ها به شرح زیر می باشند: pack، grid و place.

متد pack() – تابع حاضر المان های رابط کاربری را به صورت خانه هایی داخل یک بلوک در کنار هم جای می دهد و سپس آن ها را داخل کنترل رابط کاربری پدر (parent widget) سازمان دهی می کند.

متد grid() – تابع جاری ابزارک های رابط کاربری را در ساختاری جدول مانند و به طور سطر و ستون چیده و سپس آن ها را داخل المان والد سازمان دهی می کند.

متد place() – تابع جاری ابزارک ها را با جایگذاری آن ها در موقعیت خاص داخل المان پدر، سازمان دهی می کند.

افزونه نویسی (extension programming) با زبان C برای Python

هر کدی که با زبان های کامپایل شده (compiled) همچون C، C++ یا Java نوشته شده باشد به راحتی قابل ادغام، معرفی و استفاده در اسکریپت های Python می باشد. کدی که با استفاده از دستور import به اسکریپت های پایتون اضافه شود، تحت عنوان extension یا افزونه شناخته می شود.

ماژول extension پایتون در واقع چیزی به جز یک کتابخانه ی متعارف که با زبان C نوشته شده باشد نیست. در دستگاه های مبتنی بر Unix، این کتابخانه ها دارای پسوند so. (Shared object) می باشند. در سیستم عامل ویندوز، همین فایل ها را با پسوند dll. مشاهده می کنید.

ابزار لازم برای نوشتن افزونه ها

به منظور نوشتن افزونه های اختصاصی جهت استفاده در اسکریپت های پایتون و اپلیکیشن های خود، لازم است به فایل های header پایتون دسترسی داشته باشید.

- در دستگاه هایی که سیستم عامل Unix بر روی آن نصب است، می بایست یک پکیج مختص توسعه دهنده (developer-specific) نظیر python2.5-dev را نصب نمایید.
- کاربران ویندوز این فایل های header را به هنگام استفاده از binary Python installer به صورت یک پکیج دریافت می کنند.

علاوه بر آن، برای درک مفاهیم این مبحث و نوشتن افزونه های اختصاصی خود جهت استفاده در اسکریپت های پایتون، لازم است آشنایی در سطح پیشرفته با زبان های C یا C++ داشته باشید.

اولین نمونه از افزونه ی اختصاصی Python

کد ماژول و افزونه های پایتون، بایستی مانند زیر به چهار بخش سازمان دهی شود:

- فایل header با اسم و پسوند Python.h.
- توابع C که می خواهید به عنوان interface و الگوی پیاده سازی ماژول اختصاصی خود در اختیار توسعه دهنده قرار دهید.
- یک جدول که اسم توابع اختصاصی شما را به توابع C داخل افزونه (کتابخانه یا ماژول) نگاشت می کند (method mapping table).
- یک تابع سازنده جهت مقداردهی اولیه و نمونه سازی از کلاس (initialization function).

فایل Python.h

لازم است فایل Python.h را داخل فایلی که کدهای C شما را دربرمی گیرد (source file) قید نمایید. بدین وسیله شما به توابع کتابخانه ای درون ساخته ی پایتون (internal Python API) که

برای ادغام و معرفی ماژول مورد نظر در interpreter (hook کردن کد ماژول شما در مفسر) بکار می رود، دسترسی خواهید داشت.

لازم است Python.h را قبل از هر فایل header مورد نیاز دیگری لحاظ نمایید.

توابع C

اسم متد، نوع و تعداد پارامترهای ورودی (Signature) توابع اختصاصی شما و پیاده سازی آن، بایستی بر اساس یکی از الگوهای زیر انجام شود:

```
static PyObject *MyFunction( PyObject *self, PyObject *args );
static PyObject *MyFunctionWithKeywords(PyObject *self,
                                        PyObject *args,
                                        PyObject *kw);
static PyObject *MyFunctionWithNoArgs( PyObject *self );
```

هر یک از متدهای اعلان شده ی فوق، در خروجی خود یک آبجکت Python برمی گرداند. در پایتون مفهومی به نام تابع void (تابعی که خروجی ندارد یا مقداری را برنمی گرداند) وجود ندارد. اگر شما نمی خواهید که توابع مقدار خروجی داشته باشند، لازم است مقدار None را بازگردانی نمایید. header های پایتون یک macro (خط دستور) به نام Py_RETURN_NONE در خود به صورت از پیش تعریف شده دارند که این کار را انجام می دهند.

از آنجایی که اسم توابع C هیچگاه خارج از ماژول/افزونه قابل مشاهده و دسترسی نیستند، شما می توانید هر اسمی برای متدهای اختصاصی خود انتخاب کنید. لازم به ذکر است که این توابع با کلیدواژه ی static تعریف می شوند.

اسم توابع C معمولا از ترکیبی از اسم ماژول و متد مورد نظر تشکیل می شود. در زیر نمونه ای را مشاهده می کنید:

```
static PyObject *module_func(PyObject *self, PyObject *args) {
    /* Do your stuff here. */
    Py_RETURN_NONE;
}
```

کد حاضر یک تابع Python به نام func را تعریف می کند که داخل افزونه ی module کپسوله سازی شده است. حال شما به این توابع C داخل جدول نگاشت متد (method table) Pointer و اشاره گر تعریف می کنید که در بخش بعدی کد برنامه ی شما انجام می شود.

جدول نگاشت توابع PyMethodDef (Method Mapping Table)

این جدول نگاشت متد (method table) یک آرایه ی ساده از structure های PyMethodDef است (PyMethodDef یک مدل برای تعریف متد است). این structure ساختاری مشابه زیر دارد:

```
struct PyMethodDef {
    char *ml_name;
PyCFunction ml_meth;
    int ml_flags;
    char *ml_doc;
};
```

در زیر هر یک از اعضای این ساختار شرح داده اند:

- فیلد ml_name: اسم تابع پایتون.
- ml_meth: آدرس تابعی که هر یک از signature های نام برده در بخش قبلی را دارا می باشد.
- ml_flags: این فیلد به مفسر پایتون اعلان می کند که فیلد دوم (ml_meth) کدام یک از signature های نام برده را اتخاذ می کند.
  - این flag معمولا مقداری از METH_VARARGS دارد.
  - اگر می خواهید آرگومان های کلیدواژه ای را در تابع تزریق نمایید، این flag می تواند بیتی با METH_KEYWORDS را شامل شود.
  - این flag همچنین می تواند مقدار METH_NOARGS را داشته باشد، بدین معنی که هیچ پارامتری به تابع فرستاده نمی شود.
- ml_doc: این docstring (رشته یا comment ای که توضیحی درباره ی کارایی تابع می دهد) تابع است. اگر برنامه نویس comment ای برای تابع تنظیم نکند، در آن صورت مقدار آن NULL خواهد بود.

این جدول بایستی با یک sentinel که از NULL و 0 برای اعضای مرتبط تشکیل شده، خاتمه یابد.

مثال

برای متد اعلان شده در بالا، از جدول نگاشت تابع (method mapping table) زیر استفاده می کنیم:

```
static PyMethodDef module_methods[] = {
{ "func", (PyCFunction)module_func, METH_NOARGS, NULL },
{ NULL, NULL, 0, NULL }
};
```

تابع مقداردهی اولیه (initModule)

آخرین بخش ماژول یا افزونه ی اختصاصی شما بایستی تابع مقداردهنده ی اولیه (initialization function) را شامل شود. این تابع را مفسر پایتون زمانی که ماژول در حافظه بارگذاری می شود، فرامی خواند. لازم است اسم این تابع initModule انتخاب شود (Module اسم ماژول و init اسم خود تابع می باشد).

تابع مقداردهنده ی اولیه بایستی از کتابخانه که می نویسید export و خروجی گرفته شده باشد. header های Python با اعلان دستور PyMODINIT_FUNC امکان انجام این کار را در محیطی که اسکریپت ها در آن کامپایل می شوند را فراهم می آورد. کافی است به هنگام تعریف تابع مورد نظر از آن استفاده نمایید.

تابع مقداردهنده ی اولیه ی زبان C شما دارای ساختار کلی زیر می باشد:

```
PyMODINIT_FUNC initModule() {
Py_InitModule3(func, module_methods, "docstring...");
}
```

در زیر شرح هر یک از پارامترهای تابع Py_InitModule3 را به تفصیل مشاهده می کنید:

- func: تابعی است که قرار است export و به اصطلاح خروجی گرفته شود.
- module_methods: اسم جدول نگاشت تابع (mapping table) که در بالا به آن اشاره شد.

- docstring: این پارامتر همان رشته ی متنی و comment ای است که جهت ارائه ی توضیح درباره ی قابلیت تابع در افزونه ی اختصاصی درج می شود.

در زیر تمامی بخش های تشکیل دهنده ی یک افزونه ی استاندارد را یکجا مشاهده می کنید:

```
#include <Python.h>
static PyObject *module_func(PyObject *self, PyObject *args) {
/* Do your stuff here. */
Py_RETURN_NONE;
}
static PyMethodDef module_methods[] = {
{ "func", (PyCFunction)module_func, METH_NOARGS, NULL },
{ NULL, NULL, 0, NULL }
};
PyMODINIT_FUNC initModule() {
Py_InitModule3(func, module_methods, "docstring...");
}
```

مثال

نمونه ی کاربردی که کلیه ی مفاهیم فوق را به صورت عملی بکار می برد را در زیر مشاهده می کنید:

```
#include <Python.h>
static PyObject* helloworld(PyObject* self)
{
return Py_BuildValue("s", "Hello, Python extensions!!");
}
static char helloworld_docs[] =
"helloworld( ): Any message you want to put here!!\n";
static PyMethodDef helloworld_funcs[] = {
{"helloworld", (PyCFunction)helloworld,
METH_NOARGS, helloworld_docs},
{NULL}
};
void inithelloworld(void)
{
Py_InitModule3("helloworld", helloworld_funcs,
"Extension module example!");
}
```

دستور Py_BuildValue در مثال بالا، یک مقدار Python را build یا کامپایل می کند. کد مورد نظر را داخل فایل hello.c ذخیره نمایید. در زیر با نحوه ی کامپایل و نصب ماژول که از اسکریپت پایتون فراخوانی می شود، را خواهید آموخت.

کامپایل و نصب افزونه ها (build)

پکیج distutils توزیع و نصب ماژول های پایتون، خواه ماژول های اصلی و خالص خود پایوتن باشد خواه ماژول های اختصاصی و تنظیم شده توسط توسعه دهنده، را با روشی استاندارد بسیار آسان می سازد. ماژول ها در همان قالب اولیه (source form) توزیع شده و در اختیار برنامه نویس قرار می گیرد. برنامه نویس سپس ماژول مورد نظر را با فراخوانی اسکریپت نصب (setup script) به نام setup.py، نصب می نماید.

جهت نصب ماژول ذکر شده در بالا، بایستی اسکریپت setup.py را آماده نموده و به روش زیر اجرا نمایید:

```
from distutils.core import setup, Extension
setup(name='helloworld', version='1.0',  \
ext_modules=[Extension('helloworld', ['hello.c'])])
```

اکنون با فراخوانی دستور زیر، تمامی مراحل لازم نظیر کامپایل و آماده سازی (compilation & linking) کد را انجام دهید. کد زیر کلیه ی مراحل مورد نیاز کامپایل و لینک ماژول با کامپایلر، دستورات linker و flag های مناسب را انجام داده، متعاقبا خروجی (dll.) را در پوشه ی مربوطه جایگذاری (کپی) می کند.

```
$ python setup.py install
```

در سیستم های مبتنی بر Unix، لازم است این دستور را با حساب کاربری root اجرا نمایید تا امکان یا مجوز درج داده در پوشه ی site-packages را داشته باشید. در سیستم عامل ویندوز لازم به انجام این کار نیست.

وارد کردن و استفاده از افزونه ها در پروژه

پس از نصب افزونه ی دلخواه خود، می توانید آن را در اسکریپت پایتون خود با دستور import وارد کرده و فراخوانی نمایید:

```
#!/usr/bin/python
import helloworld
print helloworld.helloworld()
```

خروجی زیر را تولید می کند:

```
Hello, Python extensions!!
```

ارسال پارامتر به تابع

در طول توسعه پروژه، گاه می بایست توابعی را اعلان و فراخوانی نمایید که پارامترهایی را به عنوان ورودی می پذیرد. از اینرو بایستی signature (اسم تابع + نوع، تعداد پارامتر ورودی) مربوطه را برای توابع C ماژول اختصاصی خود انتخاب نمایید. به طور مثال، تابع ذیل را در نظر بگیرید که تعدادی پارامتر به عنوان ورودی پذیرفته و بدین صورت اعلان می شود:

```
static PyObject *module_func(PyObject *self, PyObject *args) {
/* Parse args and do something interesting here. */
Py_RETURN_NONE;
}
```

method table ای که تابع جدید را در خود کپسوله می کند، به صورت زیر خواهد بود:

```
static PyMethodDef module_methods[] = {
{ "func", (PyCFunction)module_func, METH_NOARGS, NULL },
{ "func", module_func, METH_VARARGS, NULL },
{ NULL, NULL, 0, NULL }
};
```

می توانید با استفاده از تابع کتابخانه ای PyArg_ParseTuple آرگومان های مورد نیاز را از متغیر اشاره گر (pointer) به PyObject که به عنوان آرگومان به تابع C ارسال شده، استخراج نمایید.

اولین آرگومان ارسالی به PyArg_ParseTuple، آرگومان args می باشد. این آرگومان همان آبجکتی است که باید parse یا تحلیل نحوی شود. پارامتر دوم یک رشته ی فرمت دهی (format string) است که آرگومان ها را به آن شکلی که مورد انتظار شما است، به نمایش می گذارد. به تعداد آرگومان ها، یک یا چند کاراکتر در رشته ی فرمت دهی وجود دارد که نشانگر آرگومان های مزبور می باشند.

```
static PyObject *module_func(PyObject *self, PyObject *args) {
int i;
double d;
char *s;
if (!PyArg_ParseTuple(args, "ids", &i, &d, &s)) {
return NULL;
}
/* Do something interesting here. */

Py_RETURN_NONE;

}
```

با کامپایل نمودن ورژن جدید از ماژول خود و وارد کردن آن در متن پروژه، قادر خواهید بود تابع مورد نظر را با تعداد دلخواه و نوع مختلف از آرگومان ها فراخوانی نمایید:

```
module.func(1, s="three", d=2.0)
module.func(i=1, d=2.0, s="three")
module.func(s="three", d=2.0, i=1)
```

تابع PyArg_ParseTuple

در زیر تعداد و نوع ورودی های تابع را به شکل استاندارد (signature) PyArg_ParseTuple مشاهده می کنید:

```
int PyArg_ParseTuple(PyObject* tuple,char* format,...)
```

در صورتی که عملیات با موفقیت انجام شود، مقداری غیر صفر و چنانچه عملیات ناموفق بوده و خطا رخ داد، مقدار 0 در خروجی بازگردانی می شود. tuple، آبجکت *PyObject بوده که همان آرگومان دوم ارسال شده به تابع C می باشد. آرگومان سوم، format، یک رشته ی C می باشد که نشانگر آرگومان های الزامی و اختیاری می باشد.

در زیر لیستی از کدهای فرمت دهی که به تابع PyArg_ParseTuple ارسال می شود همراه با شرح هر یک مشاهده می کنید:

کد مربوطه	نوع معادل در C	معنی و کاربرد
c	char	یک رشته ی پایتون با طول 1 (رشته ی حاوی یک کاراکتر) معادل char در C می شود.
d	double	یک مقدار عددی float (ممیز و اعشاری) که معادل double (اعشاری با دقت بیشتر) در C محسوب می شود.

f	float	یک float (مقدار عددی اعشاری از نوع float) در پایتون معادل float در C محسوب می شود.
i	int	یک int (نوع عدد صحیح) معادل long در زبان C درنظر گرفته می شود.
l	long	یک int در زبان پایتون معادل نوع داده ای long در زبان C در نظر گرفته می شود.
L	long long	یک int یا نوع داده ای عدد صحیح در زبان پایتون، معادل long long در زبان C محسوب می شود.
O	PyObject*	یک اشاره گر غیر NULL به آرگومان Python بازگردانی می کند.
s	char*	رشته ی پایتون بدون مقادیر null جاسازی شده (embedded) به char* در زبان C فرمت دهی / تبدیل می شود.
s#	char*+int	رشته ی Python را به آدرس و طول سازگار در C تبدیل می کند.
t#	char*+int	کاربردی مشابه s# دارد با این تفاوت که هر آبجکتی که اینترفیس read-only را پیاده سازی کند، می پذیرد.
u	Py_UNICODE*	کاراکترهای Unicode (null-terminated buffer) مستقر در بافر که انتهای آن به null ختم می شود را به آبجکت Unicode پایتون تبدیل می کند.

u#	Py_UNICODE*+int	نوع دیگر از u که در دو متغیر C ذخیره می شود، اولی یک اشاره گر به آدرس Unicode مستقر در بافر و دومی طول آن.
w#	char*+int	مشابه #s، اما هر اُبجکتی که اینترفیس read/write بافر را پیاده سازی می کند، پذیرفته و با آن سازگاری دارد.
z	char*	کاربری مشابه s دارد با این تفاوت که None نیز می پذیرد (*char زبان C را بر روی NULL تنظیم می کند).
z#	char*+int	کاربردی مشابه #s دارد اما None نیز می پذیرد (*char زبان C را روی NULL تنظیم می نماید).
(...)	as per ...	یک دنباله (sequence) پایتون که هر آیتم در آن یک آرگومان در نظر گرفته می شود.
|		آرگومان های زیر اختیاری می باشد.
:		قبل از اسم تابع در پیغام های خطا قرار می گیرد.
;		قبل از درج کل متن پیغام خطا قرار می گیرد.

بازگردانی مقادیر در خروجی

تابع Py_BuildValue، درست مانند PyArg_ParseTuple، یک رشته ی فرمت دهی (string format) به عنوان ورودی دریافت می کند. بجای ارسال آدرس مقادیری که کامپایل می کنید، بایستی خود مقادیر را به عنوان آرگومان به تابع مورد نظر بفرستید. در زیر مثالی از نحوه ی پیاده سازی یک تابع که عملیات جمع را انجام می دهد، تابع add، مشاهده می کنید:

```
static PyObject *foo_add(PyObject *self, PyObject *args) {
    int a;
    int b;
    if (!PyArg_ParseTuple(args, "ii", &a, &b)) {
        return NULL;
    }
    return Py_BuildValue("i", a + b);
}
```

معادل پیاده سازی آن در زبان پایتون به صورت زیر می باشد:

```
def add(a, b):
    return (a + b)
```

می توانید دو خروجی از این تابع بازگردانی نمایید. این عملیات در پایتون با یک لیست قابل پیاده سازی خواهد بود:

```
static PyObject *foo_add_subtract(PyObject *self, PyObject *args) {
    int a;
    int b;
    if (!PyArg_ParseTuple(args, "ii", &a, &b)) {
        return NULL;
    }
    return Py_BuildValue("ii", a + b, a - b);
}
```

معادل پیاده سازی آن در زبان پایتون به صورت خواهد بود:

```
def add_subtract(a, b):
    return (a + b, a - b)
```

تابع Py_BuildValue

در زیر روش استاندارد تنظیم اسم تابع، نوع و تعداد پارامترهای ورودی آن که signature خوانده می شود را ویژه ی تابع Py_BuildValue مشاهده می کنید:

```
PyObject* Py_BuildValue(char* format,...)
```

پارامتر format، یک رشته ی C بوده و نشانگر آبجکت Python است که پارامتر حاضر باید نهایتا به آن کامپایل شود. آرگومان های زیر مقادیر C هستند که خروجی از آن ها ساخته و کامپایل می شود. نتیجه ی *PyObject یک اشاره گر (reference) جدید می باشد.

جدول زیر code string های پرکاربرد را با ذکر کارایی هر یک در اختیار شما قرار می دهد:

کد	معادل در C	شرح
c	char	یک char زبان C، به رشته ای با طول یک کاراکتر تبدیل می شود.
d	double	یک نوع عددی double زبان C، به float در پایتون تبدیل می شود.
f	float	یک float یا نوع عددی اعشاری زبان C، به همان float در پایتون تبدیل می شود.
i	int	Int زبان C به همان int (نوع عددی صحیح) در پایتون تبدیل می شود.
l	long	یک long در زبان C به int در پایتون تبدیل می شود.
N	PyObject*	یک آبجکت پایتون ارسال کرده ولی reference count (تعداد دفعاتی که آبجکت مورد نظر به آن دسترسی صورت می گیرد) آن را افزایش نمی دهد.
O	PyObject*	یک آبجکت پایتون ارسال کرده و reference count آن را طبق انتظار یک واحد افزایش می دهد.
O&	convert+void*	یک آبجکت Python را به واسطه ی تابع تبدلیگر (Convertor) به متغیر ساده در زبان C تبدیل می کند. دو آرگومان می پذیرد، اولین پارامتر یک تابع است و دومین

		پارامتر آدرس متغیر C (از هر نوعی) که به * void تبدیل می شود.
s	char*	*char که در انتهای خود 0 داشته را به رشته ی Python تبدیل می نماید یا NULL را به None.
s#	char*+int	یک رشته ی C و طول (length) آن را به یک آبجکت Python تبدیل می کند. اگر اشاره گر از نوع string (string pointer) برابر NULL باشد، طول یا length نادیده گرفته شده و None در خروجی بازگردانی می کند.
u	Py_UNICODE*	یک رشته که در سطح زبان C تعریف شده و انتهای آن مقدار NULL وجود دارد را به یونیکد پایتون تبدیل کرده و اگر NULL بود آن را به None تبدیل می کند. Buffer ای از داده های Unicode که انتهای آن null وجود دارد را به آبجکت پایتون تبدیل می کند. اگر Unicode buffer برابر NULL بود، در خروجی None بازیابی می شود.
u#	Py_UNICODE*+int	یک رشته ی تعریف شده در سطح C و طول آن را به آبجکت Unicode پایتون تبدیل می کند یا NULL را به None تبدیل می کند. به عبارت دیگر، یک Unicode (که استانداردهای USC-2 یا UCS-4) مستقر در buffer یا حافظه میانی و طول (length) آن را به آبجکت Unicode پایتون تبدیل می کند. اگر اشاره گر به Unicode موجود در buffer برابر NULL بود، طول آن نادیده گرفته شده و None را در خروجی برمی گرداند.
w#	char*+int	مشابه #s، با این تفاوت که هر آبجکتی که اینترفیس read-write را پیاده سازی می کند، می پذیرد. متغیر * char طوری

		تنظیم شده که به اولین بایت از buffer اشاره کند و Py_ssize_t را بر روی طول buffer تنظیم می کند.
z	char*	مشابه s، با این تفاوت که None نیز می پذیرد (*char در C را روی NULL تنظیم می کند).
z#	char*+int	کاربردی مشابه #s دارد (*char در C را بر روی NULL تنظیم می کند).
(...)	as per ...	از دنباله ی از مقادیر C، یک متغیر tuple در پایتون می سازد.
[...]	as per ...	از مقادیر C، یک لیست (list) در پایتون تولید می کند.
{...}	as per ...	از دنباله ای از مقادیر C، یک dictionary که المان های آن به صورت متناوب، کلید و مقدار، سازمان دهی شده، ایجاد می کند.

به طور مثال تابع Py_BuildValue("{issi}",23,"zig","zag",42) یک dictionary پایتون به صورت {23:'zig','zag':42} در خروجی تولید می کند.