آموزش سریع پایتون

پایتون در یک روز به صورت خلاصه !

نویسنده محمد شریعتی مهر

مرداد 94

در این کتاب چه می‌خوانید ؟

پایتون یکی از ساده ترین و رایج ترین زبان های برنامه نویسی است که برنامه نویسان و متخصصان امنیتی از آن به منظور توسعه برنامه های نفوذ و تست و برنامه های شبکه از آن استفاده می کنند . پایتون همچنین به منظور نوشتن برنامه های دسکتاپ و وب بکار می رود . عنوان کتاب آموزش پایتون در یک روز و به روش سریع می باشد . منظور از این عنوان یادگیری سریع و کاربردی پایتون می باشد .

هدف از این کتاب یادگیری اصول اولیه و پایه ای زبان برنامه نویسی پایتون در کمترین زمان ممکن برای مبتدیان به ساده ترین شیوه ممکن می باشد . این کتاب ترجمه یکی از بهترین و پرفروش ترین کتاب های آموزش پایتون می باشد .

Python: Learn Python in One Day and Learn It Well

این کتاب شامل فایل های تمرینی می باشد که برای راحتی کار شما در قالب فایل هایی جداگانه به صورت دسته بندی شده ارایه شده است .

فصل اول درباره پایتون

پایتون چیست ؟
چرا پایتون ؟
چرا ما پایتون را انتخاب کرده ایم ؟

فصل دوم آماده پایتون شوید

نصب مفسر پایتون
استفاده از پایتون شل و IDLE به منظور نوشتن اولین برنامه

فصل سوم دنیای متغیرها و عملگرها

متغیرها چه هستند ؟
نام گذاری یک متغیر در پایتون
علامت واگذاری
عملگرهای اصلی
دیگر عملگرهای واگذاری

فصل چهارم انواع داده ای

انواع داده ای
نوع داده ای اینتجر
نوع داده ای فلوت
نوع داده ای رشته ای
توابع درون ساخت رشته ای
فرمت دهی رشته ها با عملگر %
فرمت دهی رشته ها با تابع Input
تایپ کستینگ
نوع داده ای لیست
نوع داده ای تاپل
نوع داده ای دیکشنری

فصل پنجم برنامه خود را تعاملی کنید

تعاملی کردن برنامه خودتان
تابع input
تابع print
کوتیشن سه تایی
عبور از کاراکترها

فصل ششم ایجاد انتخاب ها و تصمیم ها

ایجاد انتخاب و تصمیم
عبارات شرطی
عبارت شرطی if
عبارت شرطی Inline if
حلقه for
حلقه while
کلمه کلیدی break
کلمه کلیدی continue
عبارت کنترلی try, except

فصل هفتم توابع و ماژول ها

توابع و ماژول ها
توابع چه هستند ؟
تعریف توابع خودتان
حوزه متغیرها
واردکردن ماژول ها
ایجاد ماژول های خودتان

فصل هشتم کارکردن با فایل ها

کارکردن با فایل ها
بازکردن و خواندن فایل های متنی
استفاده از حلقه برای خواندن فایل های متنی
نوشتن در یک فایل متنی
بازکردن و خواندن فایل های متنی با buffer size
بازکردن و خواندن و نوشتن فایل های باینری
حذف و تغییر نام فایل ها

برای بیان نظرات و سوالات خود از راههای
ارتباطی زیر با من در تماس باشید :

Email : ShariatiMehr@yahoo.com
Linkedin : https://ir.linkedin.com/in/ShariatiMehr
Twitter : https://twitter.com/shariatimehr
Google + : https://plus.google.com/+netamooznet
Instagram : https://instagram.com/netamooz

فصل یک

درباره پایتون !

به دنیای جذاب برنامه نویسی خوش آمدید . در این آموزش قصد داریم به صورت خلاصه و مفید ، ولی در عین حال کاربردی شما را با زبان برنامه نویسی پایتون آشنا کنیم . قبل از شروع و کدنویسی به یکسری سؤالات رایج پاسخ می‌دهیم .

پایتون چیست ؟

پایتون یک زبان برنامه نویسی سطح بالا می‌باشد که توسط گودیو ون روزوم در اواخر دهه ۱۹۸۰ ایجاد شد . زبان از نظر کدنویسی خوانایی بالایی دارد و در عین حال سادگی بسیار بالایی دارد و به همین دلیل برنامه نویسان را قادر می‌سازد تا به سرعت برنامه‌های خود را توسعه دهند .

شبیه بسیاری از دیگر زبان‌های برنامه نویسی پایتون به زبان انگلیسی نوشته شده و در نتیجه کامپیوتر قادر به درک آن نمی باشد . در نتیجه برای تفسیر آن ابتدا بایستی توسط مفسر تفسیر گردد . در نتیجه قبل از آغاز کدنویسی بایستی آن را بر روی سیستم خود نصب کنیم . نحوه نصب پایتون بر روی سیستم را در درس‌های بعدی توضیح خواهیم داد .

علاوه بر این برخی ابزارهای سوم شخص وجود دارند مثل Py2exe یا Pyinstaller که شما را قادر می‌سازد تا کد پایتون خود را در قالب یک فایل اجرایی بسته بندی کنیم و بدون نیاز به نصب مفسر بر روی سیستم برنامه را اجرا کنیم .

چرا پایتون ؟

تعداد زیادی از زبان‌های برنامه نویسی موجود هستند مثل C یا C++ یا جاوا . خبر خوب این است که همه زبان‌های برنامه نویسی سطح بالا بسیار شبیه هم هستند و تفاوت آن‌ها فقط در سینتکس بکار برده شده و کتابخانه‌های در دسترس و نحوه دسترسی به این کتابخانه‌ها می‌باشد . یک کتابخانه مجموعه‌ای از منابع و کدهای از قبل نوشته شده است که ما می‌توانیم از آن استفاده کرده تا برنامه‌های خود را بنویسیم . اگر شما بتوانید یک زبان برنامه نویسی را به خوبی یاد بگیرید ، به راحتی می‌توانید زبان‌های دیگر را هم فرا بگیرید . اگر در زبان برنامه نویسی جدید هستید ، پایتون مرجع بسیار خوبی برای شروع می‌باشد . یکی از ویژگی‌ها کلیدی پایتون سادگی آن می‌باشد که موجب می‌شود برای تازه کارها یادگیری آن بسیار آسان شود . بیشتر برنامه‌ها در پایتون نیازمند خطوط کد بسیار کمتری هستند . به علاوه پایتون دارای منابع عظیمی از کتابخانه‌های سوم شخص می‌باشد که قابلیت‌های این زبان را گسترش می‌دهد . همین بکاربردن خطوط کمتر و خلاصه بودن دستورات موجب می‌شود که خطاهای برنامه نویسی و زمان مورد نیاز برای توسعه برنامه به شدت کاهش یابند . همچنین پایتون را می‌توان به منظور انجام وظایف گسترده ای از جمله نوشتن برنامه‌های دسکتاپ ، برنامه‌های پایگاه داده ، برنامه نویسی شبکه ، برنامه نویسی بازی ، برنامه نویسی وب و حتی برنامه نویسی موبایل استفاده کرد . پایتون یک زبان برنامه نویسی کراس پلتفورم می‌باشد . به این معنی که کدنوشته شده در یک سیستم عامل مثل ویندوز در دیگر سیستم عامل ها مثل لینوکس و مک نیز بدون تغییر کارخواهد کرد .

چرا ما زبان پایتون را برای یادگیری انتخاب کرده‌ایم ؟

پایتون مهم‌ترین زبان برنامه نویسی در زمینه امنیت می‌باشد و اکثر برنامه‌های امنیتی از این زبان استفاده می کنند . ساختار پایتون و کتابخانه‌های موجود در آن به نحوی طراحی شده‌اند که به منظور ساخت برنامه‌های تحت شبکه بسیار کاربردی هستند . پس برای یادگیری زبان پایتون برای متخصصان امنیتی یک ضرورت به شمار می‌رود .

فصل دو

آماده پایتون شوید !

نصب مفسر پایتون

قبل شروع به نوشتن برنامه‌ها در پایتون ما بایستی مفسر مناسب را بر روی سیستم‌های خود نصب کنیم . در این کتاب ما زبان برنامه نویسی پایتون نسخه 3 را نصب آموزش می‌دهیم چرا که در سایت پایتون ، پایتون نسخه دو را نسخه Legacy معرفی کرده‌اند . به این معنی که اشخاصی که می‌خواهند یادگیری پایتون را آغاز کنند بایستی پایتون ۳ را یاد بگیرند و آینده پایتون نسخه ۳ می‌باشد .

هرچند بایستی اشاره کرد که پایتون نسخه ۲ هم‌اکنون به صورت گسترده استفاده می‌شود و در کل این دو نسخه ۹۰ درصد مشابه یکدیگر هستند . پس با یادگیری زبان پایتون ۳ شما ۹۰ درصد پایتون ۲ را هم یاد خواهید گرفت . اگر بخواهیم از دید دیگر نگاه کنیم اکنون اکثر برنامه‌های نفوذ و شبکه با پایتون ۲ نوشته می‌شوند چرا که کتابخانه‌های پایتون ۳ هنوز خیلی گسترش نیافته اند ولی بازهم برای شروع کار توصیه بر یادگیری جدیدترین منبع می‌باشد .

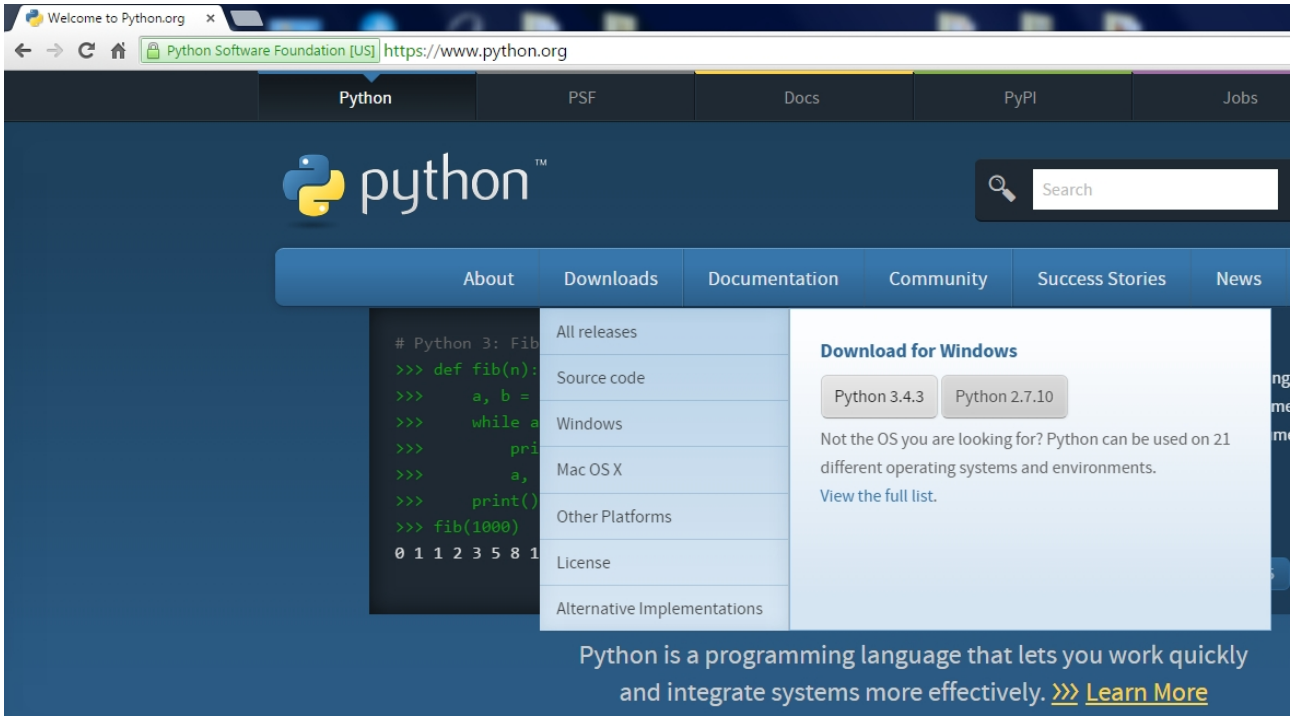
اگر از سیستم عامل لینوکس ابونتو استفاده می‌کنید ، پایتون ۳ و ۲ به صورت پیش‌فرض برای شما نصب شده است . به منظور استفاده کافی است خط فرمان را باز کنید . برای دسترسی به جدیدترین نسخه پایتون ۲ ، کافی است دستور python را مشابه تصویر زیر درون خط فرمان وارد کنید :

```
shariatimehr@Netamooz:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

برای دسترسی به پایتون نسخه ۳ در سیستم عامل اوبنتو کافی است در خط فرمان دستور python3 را وارد کنید :

```
shariatimehr@Netamooz:~$ python3
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

اگر می‌خواهید در سیستم عامل لینوکس کار کنید کافی است به آدرس زیر رفته و بسته مناسب برای کار نصب پایتون ۳ را دریافت و بر روی سیستم عامل خود نصب کنید :



پس از نصب از استارت می‌توانید به رابط های پایتون درستی پیدا کنید که در درس بعدی توضیح می‌دهیم .

استفاده از پایتون شل و IDLE به منظور نوشتن اولین برنامه خود

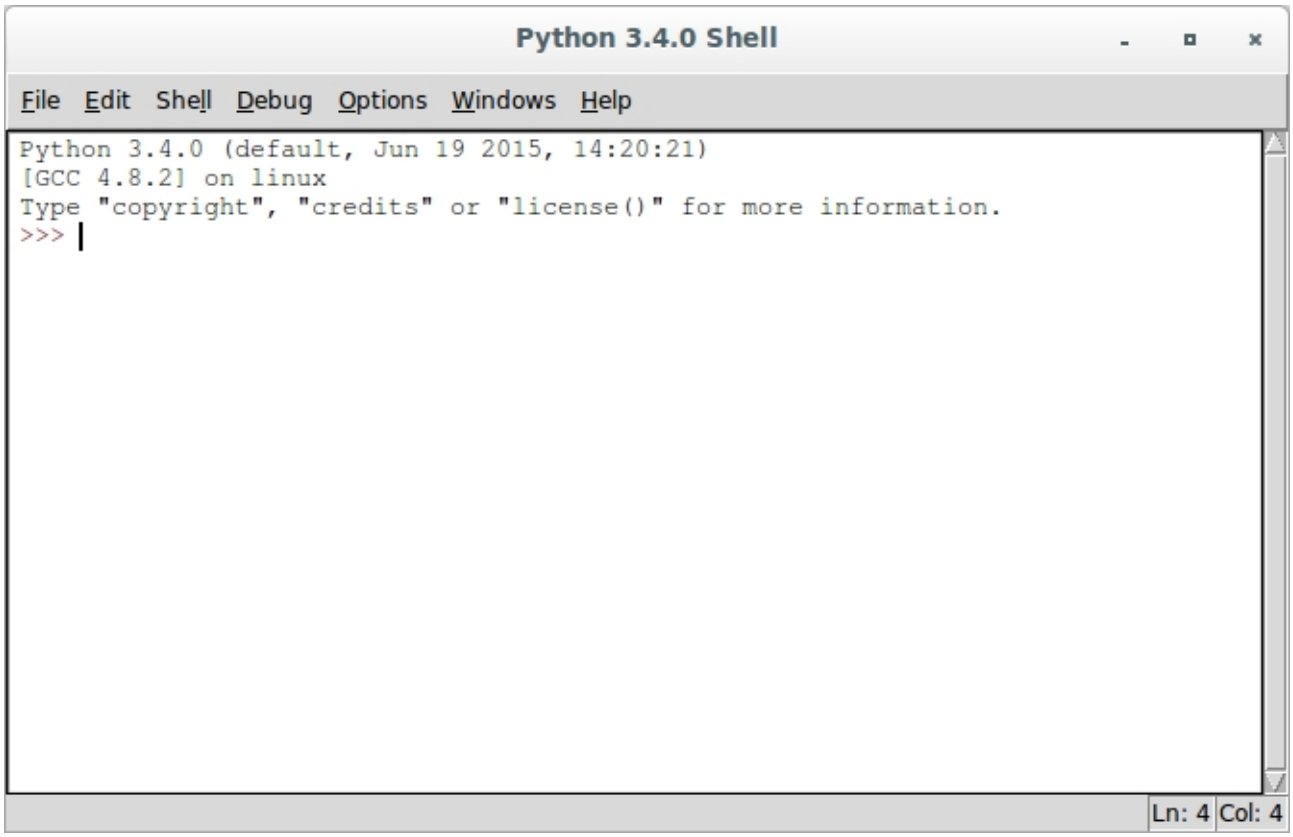
ما اولین برنامه خود را با استفاده از از IDLE خواهیم نوشت . این برنامه پس از نصب بسته نصبی در ویندوز به صورت پیش‌فرض نصب و آماده استفاده می‌شود . به منظور نصب IDLE در اوبونتو کافی است دستور زیر را درون خط فرمان لینوکس وارد کنید :

```
sudo apt-get install idle3
```

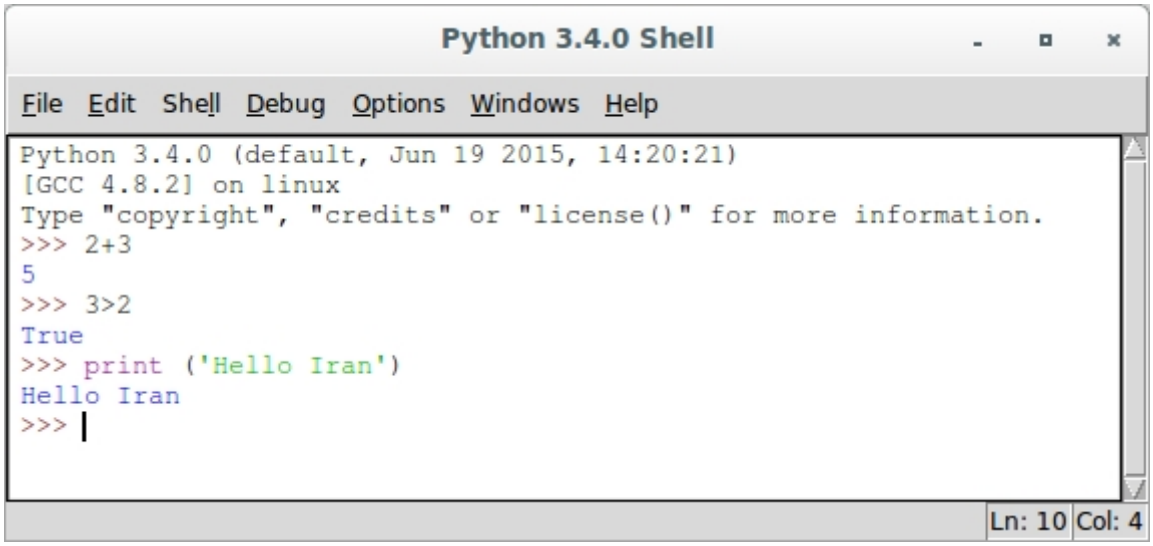
```
shariatimehr@Netamooz:~$ sudo apt-get install idle3
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  idle-python3.4 python3-tk
Suggested packages:
  tix python3-tk-dbg
The following NEW packages will be installed:
  idle-python3.4 idle3 python3-tk
0 upgraded, 3 newly installed, 0 to remove and 16 not upgraded.
Need to get 59.5 kB of archives.
After this operation, 365 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ir.archive.ubuntu.com/ubuntu/ trusty/main python3-tk amd64 3.4.0-0ubuntu1 [22.9 kB]
Get:2 http://ir.archive.ubuntu.com/ubuntu/ trusty-updates/main idle-python3.4 all 3.4.0-2ubuntu1.1 [33.5 kB]
Get:3 http://ir.archive.ubuntu.com/ubuntu/ trusty/main idle3 all 3.4.0-0ubuntu2 [3,080 B]
Fetched 59.5 kB in 13s (4,510 B/s)
Selecting previously unselected package python3-tk.
(Reading database ... 490142 files and directories currently installed.)
Preparing to unpack .../python3-tk_3.4.0-0ubuntu1_amd64.deb ...
Unpacking python3-tk (3.4.0-0ubuntu1) ...
Selecting previously unselected package idle-python3.4.
Preparing to unpack .../idle-python3.4_3.4.0-2ubuntu1.1_all.deb ...
Unpacking idle-python3.4 (3.4.0-2ubuntu1.1) ...
Selecting previously unselected package idle3.
Preparing to unpack .../idle3_3.4.0-0ubuntu2_all.deb ...
Unpacking idle3 (3.4.0-0ubuntu2) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Processing triggers for menu (2.1.46ubuntu1) ...
Processing triggers for mime-support (3.54ubuntu1.1) ...
Processing triggers for gnome-menus (3.10.1-0ubuntu2) ...
Processing triggers for desktop-file-utils (0.22-1ubuntu1) ...
Processing triggers for bamfdaemon (0.5.1+14.04.20140409-0ubuntu1) ...
Rebuilding /usr/share/applications/bamf-2.index...
Setting up python3-tk (3.4.0-0ubuntu1) ...
Setting up idle-python3.4 (3.4.0-2ubuntu1.1) ...
Setting up idle3 (3.4.0-0ubuntu2) ...
Processing triggers for menu (2.1.46ubuntu1) ...
shariatimehr@Netamooz:~$
```

پس از نصب IDLE3 را باز کنید تا دستورات خود را وارد کنیم .

پایتون شل به شما اجازه می‌دهد که از یک محیط تعاملی برای وارد کردن دستورات پایتون استفاده کنید . هرچند می‌توانید با وارد کردن دستور python3 درون خط فرمان نیز به چنین محیط مشابهی در خط فرمان دسترسی پیدا کنید . محیط تعاملی به چه معناست ؟ یعنی اینکه شما دستورات را یک خط یک خط در خط فرمان وارد می‌کنیم . شل منتظر دستور از کاربر می‌ماند و سپس آن را اجرا می‌کند و نتیجه دلخواه را باز می گرداند و منتظر دستور بعدی می‌نشیند .
دستورات زیر را در خط فرمان وارد کنید . خطوطی که با >>> آغاز می‌شوند ، خطوطی هستند که شما بایستی دستورات را درون آن‌ها وارد کنید .



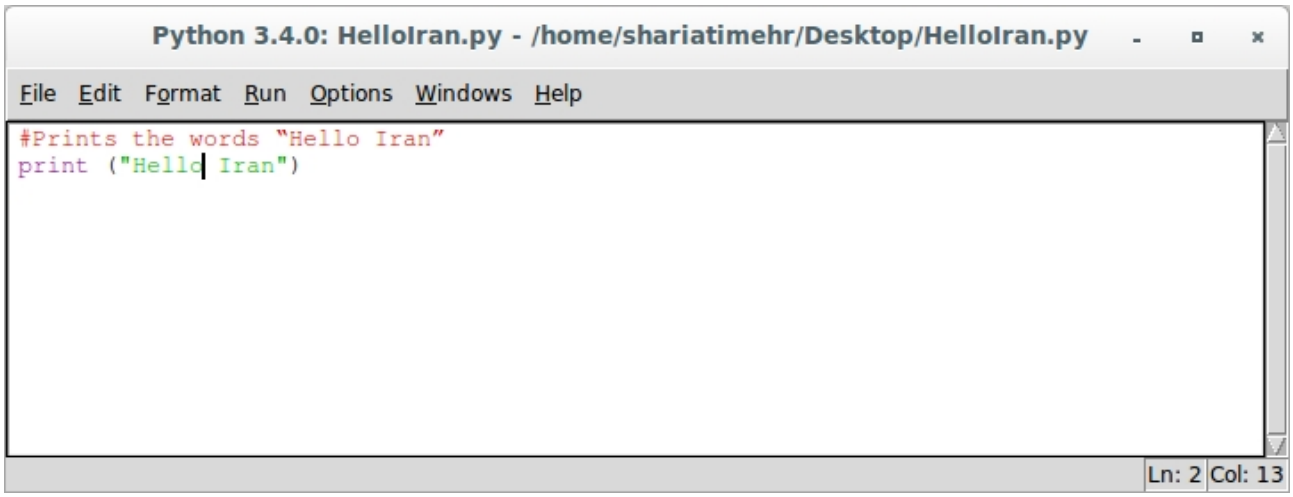
وقتی که شما دستور 3+2 را وارد می‌کنید ، دستوری را به شل می‌دهید تا مقادیر ۲ و ۳ را با هم جمع کند و پاسخ را به کاربر برگرداند . وقتی که دستور 3>2 را وارد می‌کنید از شل می‌پرسید که آیا مقدار ۳ بزرگ‌تر از دو هست یا خیر و پاسخ را به صورت بولین True باز می گرداند . در نهایت دستور print موجب چاپ پیام Hello iran در کنسول می‌شود .



شل پایتون یک ابزار بسیار مناسب برای تست دستورات پایتون می‌باشد ، بویژه برای شروع یادگیری برنامه نویسی بسیار مناسب است . هرچند اگر شما از شل خارج شوید همه دستوراتی که تا اینجای کار وارد کردید پاک می‌شوند . برای نوشتن یک برنامه واقعی بایستی فایل خود را درون یک ادیتور وارد کرده و درون یک فایل متنی با پسوند py. ذخیره کنید . این فایل را یک اسکریپت پایتون می نامند .

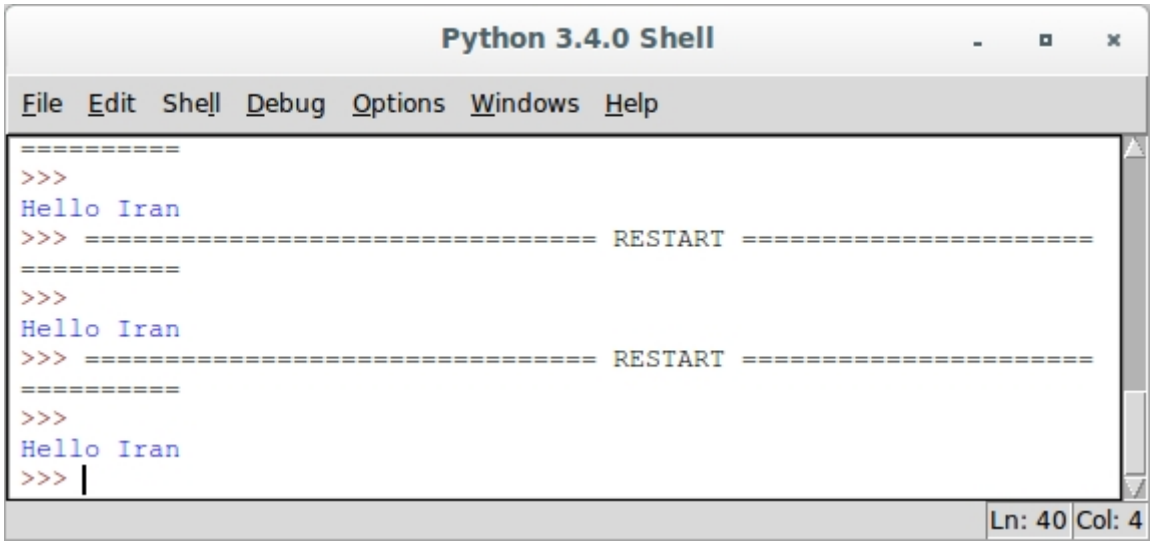
برای ایجاد یک فایل جدید درون پایتون شل از منو File بر روی new file کلیک کنید . این کار موجب می‌شود یک پنجره جدید باز شود . دستور زیر را درون آن وارد کنید و سپس از منو فایل save را انتخاب کنید و فایل را بر روی دیسک با نام دلخواه ذخیره کنید :

```
#Prints the words "Hello Iran"
print ("Hello Iran")
```



در کد بالا خط اول فقط یک کامنت است چرا که با علامت # آغاز می‌شود و خط بعدی جمله Hello iran را در خروجی چاپ می‌کند .

پس از آنکه فایل را با فرمت py. ذخیره کردید به منظور اجرای آن کافی است تا از منو Run گزینه Run Module را انتخاب کنید تا برنامه شما اجرا شود . یا اینکه کلید F5 را برای اجرا فشار دهید . نتیجه این می‌شود که جمله Hello Iran را در خروجی مشاهده می‌کنید . خوب اکنون که یاد گرفتید یک برنامه ساده درون پایتون ایجاد کنید آموزش کدنویسی در پایتون را با هم آغاز می‌کنیم .



فصل سه

دنیای متغیرها

و عملگرها

متغیر ها چه هستند ؟

متغیر ها اسامی هستند که به داده‌های مورد نیاز در برنامه خود برای دستکاری اختصاص می‌دهید . برای مثال فرض کنید که برنامه ما می‌خواهد سن یک کاربر را ذخیره کند . به این منظور ما می‌توانیم این داده را با نام userAge ذخیره کنیم و به صورت زیر متغیر را تعریف کنیم :

```
userAge = 0
```

پس از آنکه متغیر userAge را تعریف نمودید ، برنامه شما بخشی از فضای حافظه کامپیوتر شما را به منظور ذخیره این داده در نظر می‌گیرد . سپس شما می‌توانید به این داده با اشاره به نام آن ، دسترسی پیدا کنید یا آن را تغییر دهید . هر زمان که شما یک متغیر جدید ایجاد می‌کنید ، بایستی به آن یک مقدار اولیه بدهید . در این مثال ما به آن مقدار عدد صفر را دارد . ما همیشه این مقدار را می‌توانیم در برنامه خود تغییر دهیم . همچنین می‌توانیم چندین متغیر را به صورت یکدفعه تعریف کنیم . به این منظور به سادگی از ساختار نوشتاری زیر استفاده می‌کنیم :

```
userAge , userName = 30 , 'Peter'
```

این نوع تعریف و مقدار دهی معادل تعریف و مقدار دهی به صورت زیر می‌باشد .

```
UserAge = 30
userName = 'Peter'
```

در دو مثال بالا یک کار را به دو شیوه متفاوت انجام دادیم و آن تعریف دو متغیر userAge و userName بود و به ترتیب مقدار دهی عدد ۳۰ و مقدار رشته ای Peter به آن دو

نام گذاری یک متغیر در پایتون

نام یک متغیر در پایتون فقط می‌تواند حروف (a-z و A-Z) ، اعداد و آندرلاین باشد .
هرچند حرف اول متغیرها را نمی‌توان یک عدد گذاشت . مثلاً می‌توان متغیرهایی با نام های user_name،userName یا userName2 ایجاد کرد ولی نمی‌توان متغیری با نام 2userName ساخت .
به علاوه برخی کلمات رزور شده وجود دارند که شما نمی‌توانید آن‌ها را به عنوان نام متغیر استفاده کنید . چرا ؟ به این دلیل که آن‌ها قبلاً معنای مشخصی در زبان پایتون دارند . این کلمات رزرو شده شامل print، input ، if، while و ... هستند .
درباره هر کدام از آن‌ها در درس‌های بعدی توضیح خواهیم داد .
نکته آخر و مهم اینکه نام متغیرها در زبان برنامه نویسی پایتون نسبت به حروف و بزرگ و کوچک حساس است . مثلاً username و userName یکی نیستند .
این‌ها دو متغیر جداگانه در زبان پایتون محسوب می‌شوند چرا که دارای حروف بزرگ و کوچک متفاوتی هستند . در نام گذاری متغیرها شما می‌توانید از استایل camelCase و یا آندرلاین نیز استفاده کنیم که اختیاری می‌باشند .

علامت واگذاری

به یاد داشته باشید که علامت = در عبارت userAge = 0 معنی متفاوتی از آنچه که ما در ریاضیات یادگرفتیم دارد . در برنامه نویسی علامت = مساوی با نام Assignment به معنی واگذاری شناخته می‌شود . به این معنی که ما داریم مقداری را در سمت راست علامت = به متغیر اختصاص می‌دهیم . یک راه خوب برای درک این عبارت userAge = 0 این است که آن را مشابه عبارت userAge <- 0 در نظر بگیریم . در برنامه نویسی دو عبارت x = y و y = x معانی کاملاً متفاوتی دارند در حالی که در ریاضیات این دو برابرند . به همین دلیل است که من همیشه به استادان ریاضی عرض می‌کنم که ریاضیات هیچ کاربردی در برنامه نویسی ندارد ;)

گیج شدید ؟ یک مثال راهگشاست .

کد زیر را درون ادیتور وارد کنید و آن را ذخیره کنید :

```
x = 5
y = 10
x = y
print ("x = ", x)
print ("y = ", y)
```

اکنون برنامه را با فرمت py. ذخیره کرده و آن را اجرا کنید . شما بایستی خروجی زیر را دریافت کنید .

خروجی بدست آمده به صورت زیر خواهد بود :

```
x = 10
y = 10
```

هرچند که x دارای یک مقدار اولیه ۵ می‌باشد (که در خط اول تعیین شده است) ، خط سوم x = y مقدار y را به x نسبت می‌دهد . نه اینکه x را برابر y قرار دهد .

بلکه مقدار y را درون متغیر x می‌ریزد . چون مقدار y ۱۰ می‌باشد پس مقدار x هم ۱۰ می‌شود . حالا عبارت زیر چطور ؟

```
x = 5
y = 10
y = x
print ("x = ", x)
print ("y = ", y)
```

تفاوت چیست ؟ فقط در خط سوم جای x و y را عوض کردیم . درواقع این بار به جای اینکه مقدار متغیر y یعنی ۱۰ درون x بریزد و هر دو مقدار 10 را داشته باشند ، مقدار متغیر x یعنی 5 درون متغیر y می‌ریزد و هر دو مقدار ۵ را خواهند داشت .

عملگرهای اصلی

علاوه بر واگذاری یک متغیر به یک مقدار اولیه ما می‌توانیم عملگرهای محاسباتی را بر روی متغیر ها انجام دهیم. عملگرهای اصلی در پایتون شامل + , - , * , / , // , % و ** می‌باشند که به ترتیب معادل جمع ، تفریق ، ضرب ، تقسیم ، تقسیم کف ، باقی‌مانده و توان می‌باشند. به مثال های زیر دقت کنید. فرض کنید که x=5 و y=2 موارد زیر را بررسی کنید :

```
x + y = 7 جمع
x - y = 3 تفریق
x * y = 10 ضرب
x / y = 2.5 تقسیم
x // y = 2 تقسیم کف (پاسخ را به پایین رند می کند)
x % y =1 (باقی مانده تقسیم 5 بر ۲ را یعنی عدد ۱ را به شما می دهد)
x**y = 25 توان
```

دیگر عملگرهای واگذاری

در کنار عملگر = دیگر عملگرهای واگذاری در پایتون (و دیگر زبان‌های برنامه نویسی) وجود دارند . این عملگرها شبیه =+ و =- و =* می‌باشند .

فرض کنید که متغیر x با یک متغیر اولیه ۱۰ را داریم . اگر که بخواهیم به صورت افزایش بالا ببریم مثلاً ۲ تایی می‌توانیم بنویسیم ، x = x + 2

این برنامه ابتدا عبارت را در سمت راست ارزیابی می‌کند و سپس جواب را به مقدار x در سمت چپ واگذار می‌کند . پس در نهایت عبارت بالا مقدار ۱۲ را به متغیر x می‌دهد .

حال به جای عبارت بالا می‌توانیم بنویسیم x += 2 که دقیقاً همان عمل‌کرد و معنی را دارد . درواقع علامت =+ یک عملگر کوتاه نویسی است که از ترکیب نشانه واگذاری با عملگر جمع پدید می‌آید . به صورت مشابه این شیوه خلاصه نویسی برای دیگر عملگرها مثل تفریق و ... نیز وجود دارند .

فصل چهار

انواع داده ای

در پایتون

انواع داده‌ای

خوب متغیرها و انواع داده‌ای در پایتون را یادگرفتیم . در ادامه ابتدا به توضیح انواع داده‌ای در پایتون می‌پردازیم . به ویژه اینتجرها ، فلوت ها و رشته‌ها . سپس مفاهیمType Casting را بررسی کرده و سه نوع داده پیشرفته را در پایتون بررسی می‌کنیم یعنی tuple،list و دیکشنری

اینتجرها Integers

اعدادی بدون بخش اعشاری می‌باشند مثل 7 , 5 , 2 , 0 , -4 , -7 و ...

به منظور اعلان یک عدد اینتجر در پایتون به سادگی نام متغیر را برابر مقدار عددی قرار دهیم مثل :

userAge = 20 , mobileNumber = 09121111111

فلوت Float

فلوت ها اشاره به اعدادی دارند که دارای بخش اعشاری هستند مثل 1.234 و 0.0232 و 2.12

به منظور اعلان یک عدد فلوت در پایتون به صورت مثال زیر عمل می‌کنیم :

userHeight = 1.82 , userWeight = 67.2

رشته String

رشته یا استرینگ به متن اشاره می‌کند . برای اعلان یک مقدار رشته بایستی مقدار را درون تک کوتیشن یا دابل کوتیشن قرار دهیم :

```
variableName = " مقدار اولیه "
variableName = ' مقدار اولیه '
```

مثال

```
userName = 'Emily' , Family = 'Van Camp' , userAge = '28'
```

در مثال بالا سه متغیر را تعریف کردیم . ابتدا به نام کاربری مقدار emily را اختصاص داده و سپس به نام خانوادگی Van Camp و در آخر متغیر سن را ۲۸ تعیین کردیم .
نکته اینکه چون مقدار عددی سن یعنی عدد ۲۸ را درون کوتیشن قرار دادیم اکنون مقدار متغیر userAge یک مقدار رشته ای می‌باشد . در مقابل اگر که نوشته بودید userAge = 30 (بدون کوتیشن) مقدار متغیر userAge اینتجر بود .
ما می‌توانیم چندین زیر رشته را با استفاده از علامت + به هم الحاق کنیم . برای مثال :

```
'' Emily '' + '' VanCamp ''
```

این عبارت معادل مقدار " Emily VanCamp " را به شما می‌دهد .

توابع درون ساخت رشته ای

پایتون شامل تعدادی تابع درون ساخت به منظور دستکاری رشته‌ها می‌باشد . تابع به معنی بلوکی از کدهای قابل استفاده مجدد است که وظایف خاصی را انجام می‌دهند .

جلوتر درباره توابع بیشتر صحبت خواهیم کرد . مثالی از یک تابع موجود در پایتون ، تابع upper() می‌باشد . این تابع به منظور بزرگ کردن همه حروف یک رشته خاص به کار می‌رود . برای مثال اگر رشته Emily را داشته باشید با استفاده از این تابع به صورت زیر :

```
'Emily'.upper()
```

خروجی EMILY را خواهیم داشت .

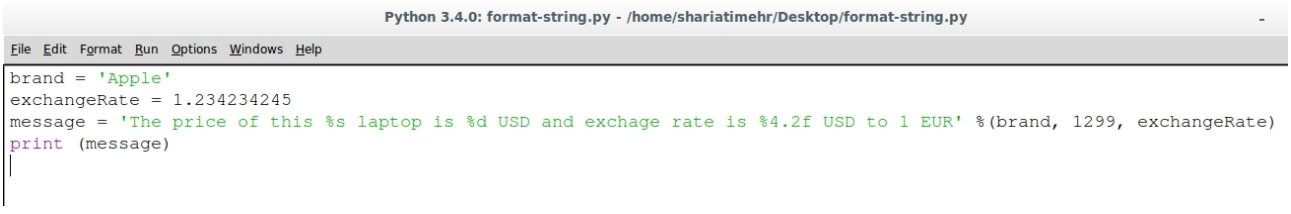
فرمت دهی رشته ها با عملگر %

رشته‌ها را می‌توان با استفاده از عملگر % فرمت دهی کرد . این عملگر به شما کنترل بیشتری بر روی نحوه نمایش و ذخیره سازی رشته می‌دهد . سینتکس استفاده شما برای این عملگر به صورت زیر می‌باشد :

" String to Format " % (مقادیر یا متغیرهایی که درون رشته درج می‌شوند و با کاما از هم جدا می‌شوند)

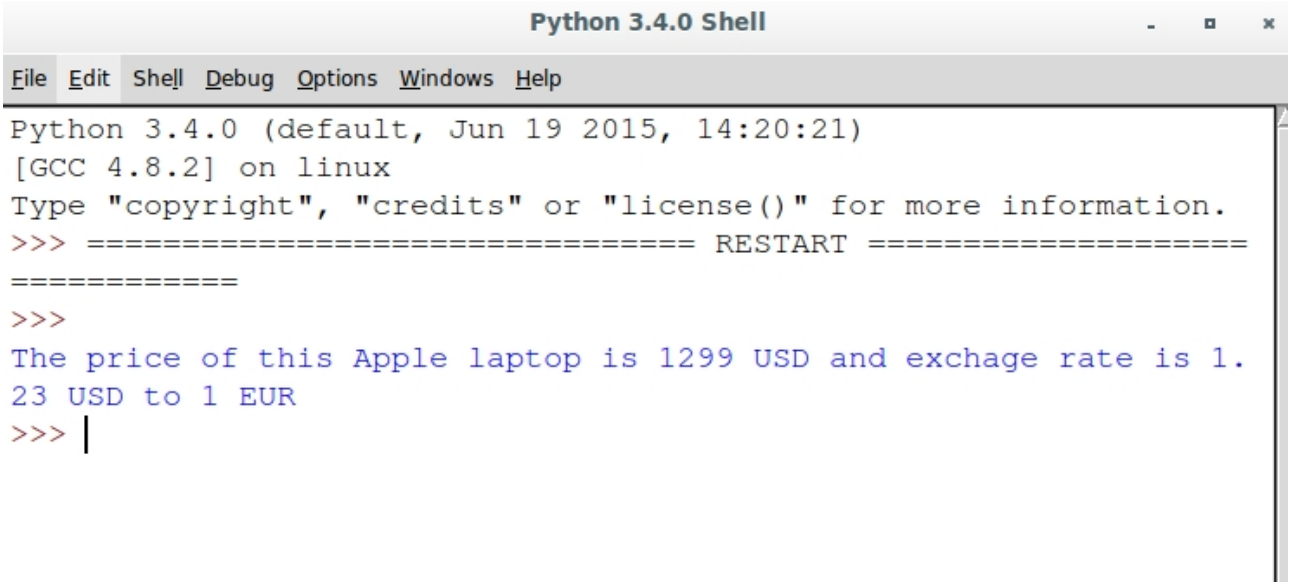
این سینتکس سه بخش دارد . بخش اول رشته ای که می‌خواهیم فرمت دهی کنیم که درون کوتیشن قرار می‌گیرد . سپس علامت درصد % را قرار می‌دهیم . بخش سوم درون پرانتز مقادیر یا متغیرهایی که می‌خواهیم درون رشته درج کنیم را قرار می‌دهیم . کد زیر را درون IDLE وارد کنید و اجرا کنید :

```
brand = 'Apple'
exchangeRate = 1.234234245
message = 'The price of this %s laptop is %d
USD and exchage rate is %4.2f USD to 1 EUR'
%(brand, 1299, exchangeRate)
print (message)
```



در مثال بالا رشته The price of this %s laptop is %d USD and exchage rate is %4.2f USD to 1 EUR که درون کوتیشن قرار دارد ، رشته ای است که می‌خواهیم آن را فرمت دهی کنیم . ما از فرمت دهنده های %s و %d و %4.2f به عنوان جایگاه های خود درون متن استفاده کرده . سپس علامت درصد را آورده و مقادیر موجود درون پرانتز را یعنی متغیرbrand و مقدار 1299 و متغیر exchangeRate را به ترتیب درون جایگاه ها قرار می‌دهیم .

اگر خروجی بگیریم مقدار زیر حاصل می‌شود :



فرمت دهنده %s به منظور ارایه یک رشته استفاده می‌شود در اینجا مثلاً Apple .
فرمت دهنده %d ارایه کننده یک مقدار عددی اینتجر می‌باشد در اینجا 1299 . اگر بخواهیم قبل از مقدار عددی فاصله اضافه کنیم ، می‌توانیم یک شماره بین علامت % و حرف d وارد کنیم که نشان دهنده طول رشته مورد نظر است .

برای نمونه :

```
''%5d'' %(123)
```

به ما مقدار "123 " به همراه دو فاصله در ابتدای آن را خواهد داد . یعنی طول کاراکتر ما یعنی 123 مقدار سه می‌باشد و ما بهش عدد ۵ رو دادیم . پس ۲ فاصله در ابتدا ایجاد می‌کند .

فرمت دهنده %f به منظور فرمت دهی فلوت ها (اعداد اعشاری) به کار می‌رود . در اینجا ما %4.2f را داریم که عدد ۴ اشاره به طول کل رشته و عدد ۲ اشاره به دو جایگاه اعشاری دارد . یعنی طول کل رشته ما ۴ کاراکتر که دو رقم آن اعشاری است . اگر بخواهیم قبل از عدد فاصله اضافه کنیم می‌توانیم به صورت %7.2f استفاده کنیم که در این حالت سه فاصله به کل طول رشته در ابتدای آن اضافه می‌شود .

فرمت دهی رشته ها با تابع Format

علاوه بر استفاده از عملگر ٪ برای فرمت دهی رشته‌ها ، پایتون متد دیگری برای فرمت دهی رشته‌ها را به ارایه می‌دهد که سینتکس آن به صورت زیر می‌باشد :

```
''string to be formatted''.format
```

(مقادیر یا متغیرهایی برای درج در رشته که با کاما جدا می‌شوند)

وقتی که متد format استفاده می‌کنیم دیگر از نگهدارنده های %s یا %f یا %d استفاده نمی‌کنیم . در عوض از آکولاد به صورت زیر استفاده می‌کنیم :

```
message = 'The price of this {0:s} laptop is
{1:d} USD and the exchange rate is {2:4.2f}
USD to 1 EUR'.format('Apple', 1299,
1.235235245)
```

درون آکولاد ابتدا جایگاه پارامتر مورد استفاده را می نویسیم سپس دو نقطه . پس از دو نقطه فرمت دهنده را وارد می‌کنیم . درون آکولاد نبایستی فاصله‌ای قرار داد .
وقتی که عبارت ('Apple', 1299, 1.235235245) را می نویسیم ، درواقع سه پارامتر را به تابع format پاس می‌دهیم .پارامترها داده‌هایی هستند که متد به منظور انجام وظایفش به آن‌ها نیاز دارد . پارامتر Apple دارای دارای موقعیت 0 می‌باشد ، پارامتر ۱۲۹۹ دارای موقعیت 1 و پارامتر 1.235235245 دارای موقعیت 2 می‌باشد . موقعیت ها از ایندکس صفر شروع می‌شوند .

وقتی که می نویسیم {s:0} از مفسر می‌خواهیم که این پارامتر را با موقعیت پارامتری ۰ که یک مقدار رشته ای است جایگزین کنیم (فرمت دهنده ما s می‌باشد چون مقدار ما رشته ای است).

وقتی که ما می نویسیم {d:1} به پارامتر موجود در موقعیت ۱ که یک مقدار عددی است (فرمت دهنده ما هم d می باشد) اشاره می‌کنیم .
وقتی عبارت {2:4.2f} را می نویسیم . به پارامتر موقعیت ۲ که مقداری اعشاری است اشاره می‌کنیم و می‌خواهیم که با دو جایگاه اعشاری و طول کل ۴ رشته فرمت دهی کنیم

اگر که مقدار message را چاپ کنیم خروجی به صورت زیر خواهد بود :

```
The price of this Apple laptop is 1299 USD
and the exchange rate is 1.24 USD to 1 EUR
```

نکته : اگر که نمی‌خواهیم رشته را فرمت دهی کنید و فقط می‌خواهید جایگذاری کنید می‌توانید به صورت زیر بنویسیم :

```
message = 'The price of this {} laptop is {}
USD and the exchange rate is {} USD to 1
EUR'.format('Apple', 1299, 1.234234245)
```

در این حالت نیازی به قرار دادن موقعیت پارامتر ها نداریم و مفسر پایتون به صورت خودکار به ترتیب پارامترها را جایگذاری خواهد کرد .

متد format() برای مبتدی ها ممکن است گیج‌کننده باشد . در حقیقت فرمت دهی رشته‌ها می‌تواند بیش از این پیچیده باشند ولی آنچه در اینجا گفتیم در اکثر موارد و بیشتر اهداف کارساز خواهد بود . برای درک بهتر از متد format() برنامه زیر را امتحان کنید :

```
message1 = '{0} is easier than
{1}'.format('Python', 'Java')
message2 = '{1} is easier than
{0}'.format('Python', 'Java')
message3 = '{:10.2f} and
{:d}'.format(1.234234234, 12)
message4 = '{}'.format(1.234234234)

print (message1)
#You will get 'Python is easier tha Java'

print (message2)
#You will get 'Java is easier Python'

print (message3)
#You will get '       1.23 and 12'
#You dont need to indicate the positions of
the parameters.

print (message4)
#You will get '1.234234234. no formatting is
done.'
```

Python 3.4.0: format-example.py - /home/shariatimehr/Desktop/format-example.py

File Edit Format Run Options Windows Help

```
message1 = '{0} is easier than {1}'.format('Python', 'Java')
message2 = '{1} is easier than {0}'.format('Python', 'Java')
message3 = '{:10.2f} and {:d}'.format(1.234234234, 12)
message4 = '{}'.format(1.234234234)

print (message1)
#You will get 'Python is easier tha Java'

print (message2)
#You will get 'Java is easier Python'

print (message3)
#You will get '      1.23 and 12'
#You dont need to indicate the positions of the parameters.

print (message4)
#You will get '1.234234234. no formatting is done.'
```

Ln: 3 Col: 5

خروجی به صورت زیر خواهد بود :

Python 3.4.0 Shell

File Edit Shell Debug Options Windows Help

```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
============
>>>
Python is easier than Java
Java is easier than Python
      1.23 and 12
1.234234234
>>>
```

Ln: 10 Col: 4

تایپ کستینگ

برخی مواقع در برنامه‌های خود ، نیاز داریم تا یک نوع داده‌ای را به نوعی دیگر تبدیل کنیم . مثلاً تبدیل یک عدد اینتجر به یک رشته . این کار را تایپ کستینگ Type Casting یا تبدیل نوع داده‌ای می نامند .

سه تابع درون ساخت برای پایتون وجود دارد که این کار را برای ما انجام می‌دهند . این توابع به شرح زیر می‌باشند :

```
int ()
float ()
str ()
```

تابع int در پایتون یک مقدار float یا یک مقدار رشته ای مناسب را گرفته و آن را به مقدار اینتجر تبدیل می‌کند . مثلاً برای تبدیل یک مقدار فلوت به اینتجر به صورت زیر عمل می‌کنیم :

```
int (5.712987)
```

که نتیجه آن می‌شود عدد ۵

برای تبدیل یک مقدار رشته ای به یک مقدار اینتجر می‌توانیم به صورت زیر عمل کنیم :

```
int ("4")
```

و نتیجه می‌شود عدد ۴ به صورت عدد (نه رشته ای)

مثال بالا رو دیدید ولی ما نمی‌توانیم بنویسیم int ("Hello") ویا int ("4.22321") چرا که بایستی مقدار رشته مناسب به ورودی بدهیم . در دو حالت خطا دریافت می‌کنیم .

تابع float در عوض یک مقدار اینتجر یا مقدار رشته ای مناسب را از ورودی گرفته و آن را به مقدار فلوت تبدیل می‌کند . برای نمونه اگر بنویسیم float (2) و یا float ("2") که نتیجه دریافت خروجی 2.0 می‌باشد . اگر که بنویسیم float ("2.09109") خروجی می‌شود 2.09109 که یک مقدار اعشاری ‹فلوت› می‌باشد و دیگر رشته ای نیست .

تابع str در دو سمت مقدار اینتجر یا فلوت را گرفته و خروجی رشته ای به ما می‌دهد .
برای مثال اگر بنویسیم str (2.1) خروجی می‌شود مقدار "2.1"
اکنون که شما سه نوع داده‌ای ابتدایی در پایتون را آموختید به انواع پیشرفته‌تر آن خواهیم پرداخت . در درس‌های بعدی با ما همراه شوید .

نوع داده‌ای لیست

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 . برای نمونه فرض کنید برنامه ما نیاز دارد که سن ۵ کاربر را ذخیره کند . به جای ذخیره سازی آن‌ها در پنج متغیر با نام های userAge1 تا userAge5 می‌توانیم آن‌ها را در قالب یک لیست ذخیره کنیم .

به منظور اعلان یک لیست از ساختار دستوری زیر استفاده می‌کنیم :

```
listName = [initial values]
```

به یاد داشته باشید که به منظور اعلان یک لیست در پایتون از براکت [] استفاده می‌کنیم . همچنین مقادیر لیست را با علامت کاما , از هم جدا می‌کنیم .

مثال :

```
userAge = [21, 22, 23, 24, 25]
```

ما همچنین می‌توانیم یک لیست را بدون تعیین هیچ مقدار اولیه‌ای اعلان کنیم . به این منظور به سادگی می نویسیم :

```
listName = []
```

اکنون یک لیست خالی بدون هیچ مقدار تعیین شده‌ای داریم . به منظور اضافه کردن مقادیر به لیست بایستی از تابع append استفاده کنیم که در زیر توضیح خواهیم داد . مقادیر تکی درون لیست توسط مقادیر ایندکس قابل دسترسی هستند . دقت کنید که ایندکس ها از مقدار صفر آغاز می‌شود نه از مقدار ۱ که این شیوه در اکثر زبان‌های برنامه نویسی رایج است مثل زبان C و یا Java

از آنجایی که اولین مقدار ایندکس صفر است مقدار دوم ۱ می‌باشد و الی آخر . مثلاً :

```
userAge [0] = 21
userAge [1] = 22
```

به صورت جایگزین شما می‌توانید مقادیر یک لیست را از آخر فراخوانی کنید . آخرین آیتم موجود در لیست ما ایندکس 1- و دومین آیتم از آخر ایندکس 2- را دارد . چگونه ؟ به صورت زیر:

```
userAge [-1] = 25
userAge [-2] = 24
```

شما می‌توانید یک لیست یا بخشی از آن را به یک متغیر اختصاص دهید . اگر که بنویسید

```
userAge2 = userAge
```

متغیر userAge2 می‌شود :

```
userAge = [21, 22, 23, 24, 25]
```

اگر که شما بنویسید :

```
userAge3 = userAge [2:4]
```

شما آیتم های دارای ایندکس ۲ را تا ایندکس ۱-۴ از لیست userAge برداشته و به userAge3 اختصاص می‌دهیم . یعنی از ایندکس ۲ تا ۳ که می‌شود :

```
userAge3 = [23, 24]
```

این نوع علامت گذاری 2:4 را slice یا برش می نامند . اگر از شیوه علامت گذاری برش در پایتون استفاده کنیم ، آیتم در ایندکس آغازین همیشه در لیست ما شامل می‌شود ولی آیتم آخرین همیشه از لیست حذف می‌شود . به همین دلیل است که نتیجه 2:4 به صورت زیر می‌شود :

```
userAge3 = [23, 24]
```

نه به صورت زیر :

```
user Age3 = [23, 24, 25]
```

نشانه گذاری slice دارای عدد سومی با نام Stepper می‌باشد . اگر که بنویسیم :

```
userAge4 = userAge[1:5:2]
```

در اینجا کاری که استپر انجام می‌دهد این است که هر زیر لیست شامل هر شماره دومی را از ایندکس ۱ تا ۴ به ما می‌دهد . در اینجا برای ما چونکه استپر عدد ۲ می‌باشد نتیجه می‌شود :

```
userAge4 = [22, 24]
```

علاوه براین نشانه گذاری اسلایس دارای مقادیر پیش‌فرض ‹Defaults› می باشد . مقدار پیش‌فرض برای اولین عدد صفر می‌باشد و مقدار پیش‌فرض برای عدد دوم اندازه برش داده شده لیست می‌باشد ! به مثال زیر توجه کنید واضح بیان شده است :

```
userAge [   :4 ]
```

در مثال بالا عدد اول آورده نشده و به جای آن مقدار پیش‌فرض یعنی صفر در نظر گرفته می‌شود یعنی اینکه از ایندکس صفر تا ایندکس ۱-۴ را به شما می‌دهد .

مثال دوم :

```
userAge[1:  ]
```

در مثال بالا عدم دوم آورده نشده است در نتیجه اندازه برش داده شده لیست در نظر گرفته می‌شود . اندازه اسلایس لیست ۱-۵ می‌باشد . (چرا که اندازه userAge۵ می‌باشد و دارای پنج آیتم می باشد) . پس مثال بالا از ایندکس ۱ تا ایندکس ۱-۵ یعنی ۱ تا ۴ را به ما می‌دهد .

برای ویرایش و تغییر ایتم ها در یک لیست می نویسیم :

```
listName [ ایندکس آیتم هایی که می‌خواهیم تغییر دهیم] = مقدار جدید
```

برای نمونه اگر بخواهیم که آیتم دوم را ویرایش کنیم به سادگی می نویسیم :

```
userAge [1] = 5
```

که در نتیجه عبارت بالا لیست شما به صورت زیر تغییر پیدا می‌کند :

```
userAge = [21, 5, 23, 24, 25]
```

برای اضافه کردن آیتم ها به لیست از تابع append استفاده می‌کنیم . برای مثال اگر بنویسیم :

```
userAge.append(99)
```

شما در حقیقت مقدار ۹۹ را به پایان لیست اضافه کرده‌اید . اکنون لیست شما به صورت زیر تغییر یافته است :

```
userAge = [21, 5, 23, 24, 25, 99]
```

برای حذف یک آیتم از لیست از ساختار دستوری زیر استفاده می‌کنیم :

```
del listName[ ایندکس آیتم هایی که می‌خواهیم از لیست حذف کنیم]
```

برای مثال اگر بنویسیم :

```
del userAge[2]
```

لیست به صورت زیر تغییر خواهد کرد (مقدار سومی با ایندکس ۲ حذف می شود)

```
userAge = [21, 5, 24, 25, 99]
```

برای اینکه در مبحث لیست ها خبره شوید برنامه زیر را اجرا کنید و به خروجی ها و نتایج بدست آمده دقت کنید :

```
#declaring the list, list elements can be of
different data types
myList = [1, 2, 3, 4, 5, "Hello"]
#print the entire list.
print(myList)
#You'll get [1, 2, 3, 4, 5, "Hello"]
#print the third item (recall: Index starts
from zero).
print(myList[2])

#You'll get 3
#print the last item.
print(myList[-1])
#You'll get "Hello"
```

```
#assign myList (from index 1 to 4) to
myList2 and print myList2
myList2 = myList[1:5]
print (myList2)
#You'll get [2, 3, 4, 5]
#modify the second item in myList and print
the updated list
myList[1] = 20
print(myList)
#You'll get [1, 20, 3, 4, 5, 'Hello']


#append a new item to myList and print the
updated list
myList.append("How are you")
print(myList)
#You'll get [1, 20, 3, 4, 5, 'Hello', 'How
are you']
#remove the sixth item from myList and print
the updated list
del myList[5]
print(myList)
#You'll get [1, 20, 3, 4, 5, 'How are you']
```

نوع داده‌ای تاپل

Tuples درست شبیه لیست ها می‌باشند با این تفاوت که شما نمی‌توانید مقادیر آن‌ها را ویرایش کنید . مقادیر اولیه که برای تاپل ها تعیین می کنید ، تا آخر برنامه ثابت باقی می‌مانند و قابل تغییر نیستند . کجا مفید هستند ؟ برای مثال برای ذخیره سازی اسامی ماه های سال درون برنامه مفید هستند .

برای اعلان یک تاپل ، از ساختار دستوری زیر استفاده کنید :

```
tupleName = (initial values)
```

دقت کنید که این بار به جای براکت از پرانتز برای اعلان تاپل استفاده کردیم . چندین مقدار را هم مثل قبل با کاما ، از هم جدا می کنیم . به عنوان مثال :

```
Example:
monthsOfYear = (“Jan”, “Feb”, “Mar”, “Apr”,
“May”, “Jun”, “Jul”, “Aug”,“Sep”, “Oct”,
“Nov”, “Dec”)
```

درست شبیه لیست به مقادیر تکی را از طریق ایندکس ها دسترسی پیدا می کنیم . مثلاً :

```
monthsOfYear[-1]    و    monthsOfYear[0]
```

مثال بالا به ترتیب از چپ مقادیر "Jan" و "Dec" را به ما می‌دهد .

نوع داده‌ای دیکشنری

دیکشنری مجموعه‌ای از جفت داده‌های به هم مرتبط می‌باشد . برای نمونه اگر که ما بخواهیم نام کاربری و سن ۵ کاربر را ذخیره سازی کنیم ، می‌توانیم از نوع داده‌ای دیکشنری استفاده کنیم.

به منظور اعلان یک دیکشنری از ساختار دستوری زیر استفاده می‌کنیم :

```
dictionaryName = {dictionary key : data}
```

در ساختار بالا نام دیکشنری را در سمت چپ تعیین کرده و در سمت راست کلید دیکشنری و داده مورد نظر را درون آکولاد تعریف می‌کنیم . توجه داشته باشید که کلید ها بایستی ‹درون یک دیکشنری› یگانه باشند . مثلاً شما نمی‌توانید یک دیکشنری به صورت زیر تعریف کنید :

```
myDictionary = {''Peter'' : 38, ''Jhon'' :
51, ''Peter'' : 13}
```

چرا ؟ به این دلیل که Peter دو بار به عنوان کلید درون یک دیکشنری استفاده شده است . دقت داشته باشید که برای اعلان دیکشنری از علامت آکولاد استفاده می‌کنیم .

همچنین چندین جفت را با استفاده از کاما از هم جدا می‌کنیم . برای مثال :

```
userNameAndAge = {''Emily'' : 28, ''Amanda''
: 26, Jack = 34, David = ''Not Available''}
```

همچنین شما می‌توانید یک دیکشنری را با استفاده از متد dict اعلان کنید . برای اعلان دیکشنری بالا با استفاده از متد dict به صورت زیر عمل می‌کنیم :

```
userNameAndAge = dict( Emily : 28, Amanda :
26, Jack = 34, David = ''Not Available''}
```

زمانی که از متد dict به منظور اعلان دیکشنری استفاده می‌کنیم ، به جای آکولاد از علامت پرانتز استفاده کرده و نیازی به قراردادن کلیدهای دیکشنری درون کوتیشن نمی باشد .

برای دسترسی یک آیتم درون دیکشنری از کلید دیکشنری استفاده می‌کنیم . برای مثال برای دسترسی به سن امیلی می نویسیم :

```
userNameAndAge [''Emily'']
```

که نتیجه آن می‌شود ۲۸ سال

برای ویرایش آیتم ها درون یک دیکشنری از ساختار دستوری زیر استفاده می‌کنیم :

```
dictionaryName [dictionary key of item to be
modified]
```

برای مثال به منظور ویرایش سن امیلی به صورت زیر عمل می‌کنیم :

```
userNameAndAge[''Emiy''] = 25
```

اکنون دیکشنری ما به صورت زیر تغییر می‌کند :

```
userNameAndAge = {''Emily'' : 25, ''Amanda''
: 26, Jack = 34, David = ''Not Available''}
```

همچنین ما می‌توانیم یک دیکشنری را بدون تعیین هیچ مقدار اولیه‌ای تعریف کنیم . به این منظور به سادگی به صورت زیر عمل کنیم :

```
dictionaryName = { }
```

در این حالت یک دیکشنری بدون هیچ آیتمی داریم .
به منظور اضافه کردن آیتم ها به یک دیکشنری از ساختار دستوری زیر استفاده می‌کنیم :

```
dictionaryName [dictionary key] = data
```

برای مثال اگر بخواهیم مشخصات ویکتوریا را به دیکشنری خود اضافه کنیم به صورت زیر عمل کنیم :

```
userNameAndAge [''Victoria''] = 45
```

در این وضعیت دیکشنری ما به صورت زیر تغییر می‌کند :

```
userNameAndAge = {''Emily'' : 25, ''Amanda''
: 26, Jack = 34, David = ''Not Available'',
''Victoria'' = 45}
```

به منظور حذف آیتم ها از یک دیکشنری از ساختار دستوری زیر استفاده می‌کنیم :

```
del dictionaryName [ dictionary key ]
```

برای نمونه به منظور حذف Amanda از دیکشنری به صورت زیر عمل می‌کنیم :

```
del userNameAndAge [''Amanda'']
```

اکنون دیکشنری ما به صورت زیر تغییر می‌کند و آماندا حذف می‌گردد :

```
userNameAndAge = {''Emily'' : 25, Jack = 34,
David = ''Not Available'', ''Victoria'' =
45}
```

به منظور تمرین بیشتر در این زمینه برنامه زیر را اجرا و خروجی را بررسی کنید :

```
#declaring the dictionary, dictionary keys
and data can be of different data
types
myDict = {“One”:1.35, 2.5:”Two Point Five”,
3:”+”, 7.9:2}
#print the entire dictionary
print(myDict)
#You’ll get {2.5: 'Two Point Five', 3: '+',
'One': 1.35, 7.9: 2}
#Note that items in a dictionary are not
stored in the same order as the way
you declare them.
#print the item with key = “One”.
print(myDict[“One”])
#You’ll get 1.35
#print the item with key = 7.9.
print(myDict[7.9])
#You’ll get 2
#modify the item with key = 2.5 and print
the updated dictionary
myDict[2.5] = “Two and a Half”

print(myDict)
#You’ll get {2.5: 'Two and a Half', 3: '+',
```

```
'One': 1.35, 7.9: 2}

#add a new item and print the updated
dictionary
myDict[“New item”] = “I’m new”
print(myDict)
#You’ll get {'New item': 'I’m new', 2.5:
'Two and a Half', 3: '+', 'One':
1.35, 7.9: 2}
#remove the item with key = “One” and print
the updated dictionary
del myDict[“One”]
print(myDict)
#You’ll get {'New item': 'I’m new', 2.5:
'Two and a Half', 3: '+', 7.9: 2}
```

فصل پنج

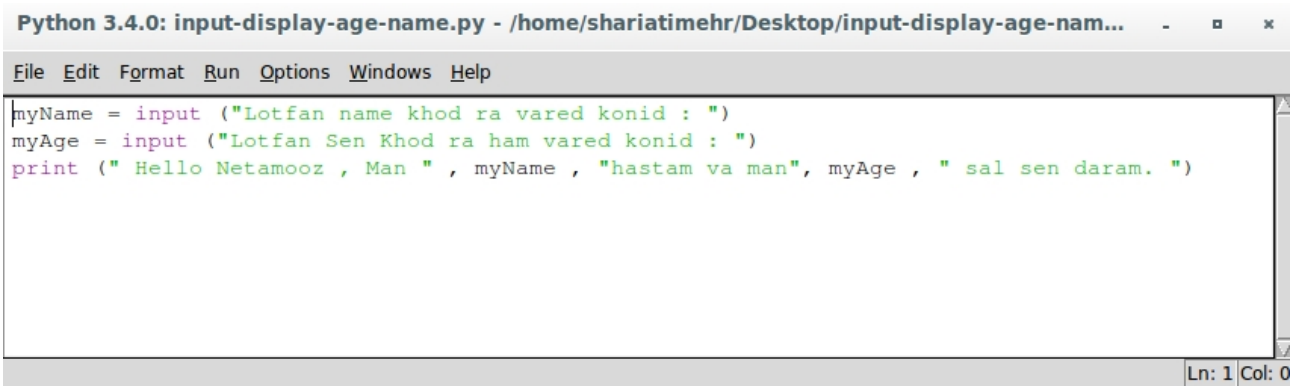
تعاملی کردن

برنامه خود در پایتون

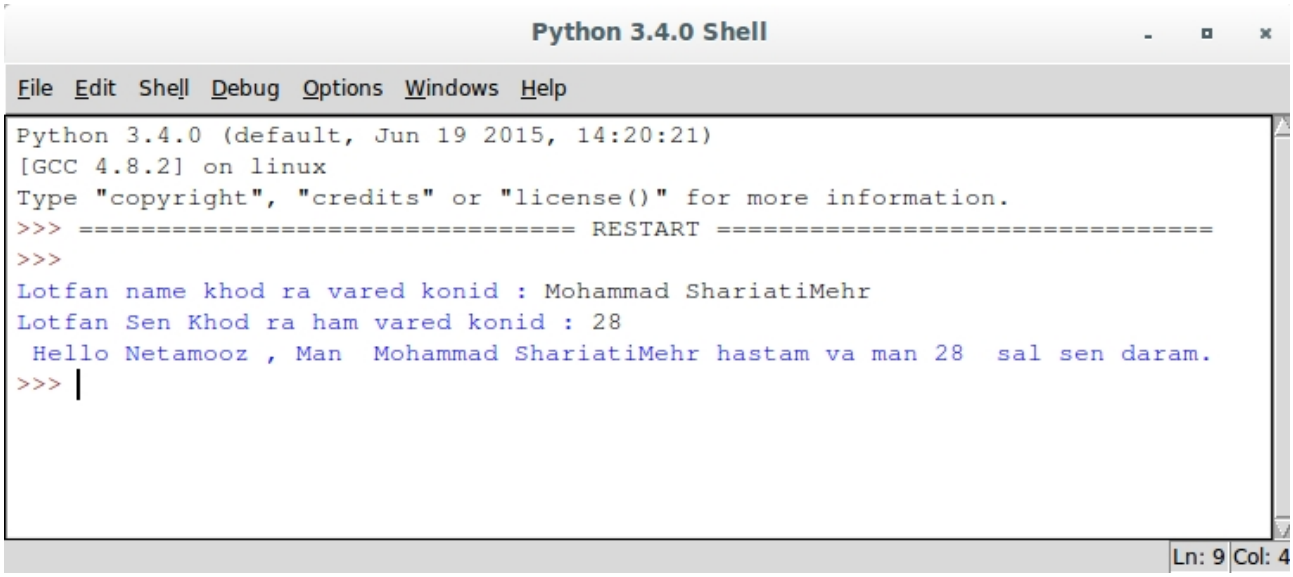
تعاملی کردن برنامه خودتان

حال که مقدمات متغیرها را یاد گرفتیم زمان آن رسیده که برنامه‌های کاربردی‌تر ایجاد کنیم . در این بخش همان برنامه hello world که قبلاً ساختیم را استفاده کرده ولی این بار به شکلی تعاملی . به جای اینکه فقط پیام Hello World را چاپ کنیم ، این بار کاری می کنیم که برنامه ما اسامی و سن افراد را نیز بداند . به این منظور برنامه ما بایستی قادر به دریافت اطلاعات کاربر باشد و آن‌ها را در صفحه نمایش نشان دهد . دو تابع درون ساخت وجود دارند که این کار را برای ما انجام می‌دهند . توابع input و print . خوب به این منظور به سادگی برنامه را درون IDLE نوشته و اجرا کنیم .

```
myName = input ("Lotfan name khod ra vared
konid : ")
myAge = input ("Lotfan Sen Khod ra ham vared
konid : ")
print (" Hello Netamooz , Man " , myName ,
"hastam va man", myAge , " sal sen daram. ")
```



همانطور که ملاحظه می‌کنید ، پس از اجرای برنامه ، ابتدا نام شما پرسیده می‌شود و سپس سن شما و در نهایت هر دو را در قالب یک جمله چاپ می‌کند .



تابع input در پایتون

در مثال قبلی ما از تابع input دو بار به منظور دریافت نام و سن کاربر برای نمایش در برنامه استفاده کردیم .

```
MyName = input ('' Lotfan Name khod ra vared
konid: '')
```

رشته " Lotfan Name khod ra vared konid " پیامی است که در صفحه نمایش نشان داده می‌شود و به کاربر توضیح می‌دهد که چه چیزی را بایستی وارد کند . پس از آنکه کاربر ورودی های خود را درون input وارد کرد ، این اطلاعات به صورت رشته درون متغیر myName ذخیره می‌شوند . فیلد input بعدی که از کاربر خواسته می‌شود سن وی را از او می‌خواهد و آن را پس از دریافت درون متغیر myAge ذخیره می‌کند .

کمی در دو نسخه ۲ و ۳ پایتون تفاوت دارد . در پایتون ۲ اگر که بخواهید input تابع استفاده raw_input ورودی کاربر را به صورت رشته ای قبول کنید ، بایستی از تابع کنید .

تابع print در پایتون

تابع print به منظور نشان دادن اطلاعات کاربران در صفحه نمایش استفاده می‌شود . این تابع هیچ یا چند عبارت را به عنوان پارامتر قبول می‌کند و آن‌ها را با کاما از هم جدا می‌کند .

در عبارت زیر ما ۵ پارامتر را به تابع print می‌دهیم . سعی کنید آن‌ها را تشخیص دهید :

```
print (''Hello World , Esme man hast'' ,
myName, '' va man '' , myAge, ''sal sen
daram.'')
```

این پارامترها به صورت زیر می‌باشند :

```
'Hello World , Esme man hast''
myName
'' va man ''
myAge
''sal sen daram.''
```

این پارامترها شامل دو متغیر و سه رشته می‌باشند .

زمانی که رشته‌ها را وارد می‌کنیم از دابل کوتیشن استفاده کرده و زمان وارد کردن متغیرها نیاز به قرار دادن کوتیشن نیست . نتیجه آن می‌شود که مقادیر متغیرها جایگذاری شده و در خروجی چاپ می‌شود . شیوه ای دیگر برای چاپ کردن متغیرها استفاده از فرمت دهنده % می‌باشد که در درس‌های قبلی یادگرفتیم . برای نوشتن مثال بالا با فرمت دهنده % به صورت زیر عمل می‌کنیم :

```
print (''Hello World , Esme man hast %s va
man %s sal sen daram'' % (myName, myAge) )
```

همچنین برای نوشتن همان عبارت با استفاده از متد format به صورت زیر عمل می‌کنیم :

```
print (''Hello World , Esme man hast {} va
man {} sal sen daram''.format(myName, myAge)
)
```

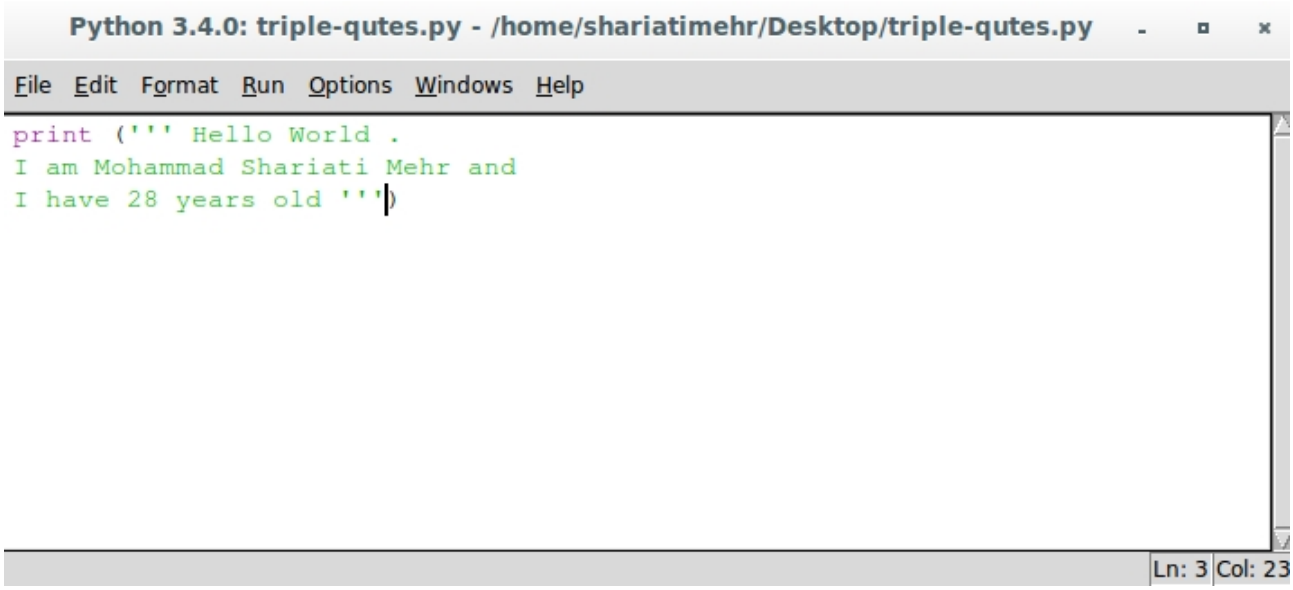
تابع print هم از دیگر توابعی است که در نسخه های ۳ و ۲ پایتون تفاوت دارد . در پایتون نسخه ۲ تابع print را بایستی بدون پرانتز و به صورت زیر نوشت ‹به جای کاما از بعلاوه استفاده می‌کنیم .›

```
print ''Hello World , Esme man hast'' +
myName + '' va man '' + myAge + ''sal sen
daram''
```

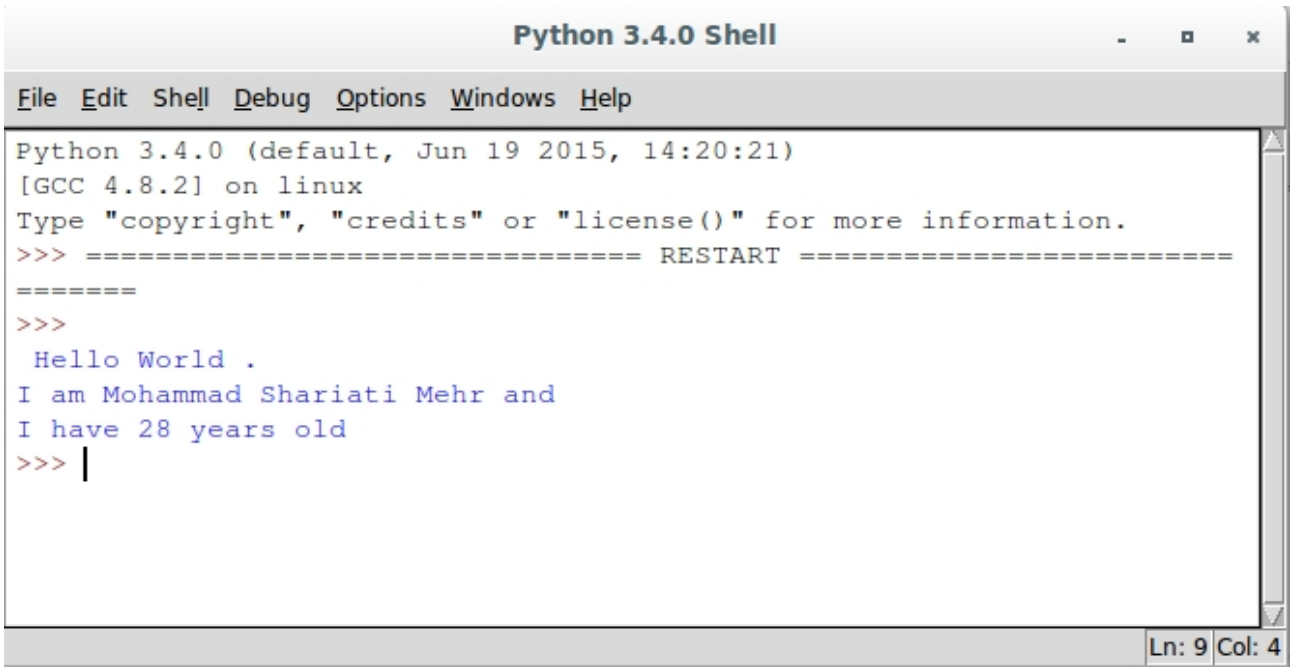
کوتیشن سه تایی

اگر که می‌خواهید یک پیام طولانی را نمایش دهید می‌توانید از کاراکترهای سه کوتیشن استفاده کنید تا پیام را در چندین خط نمایش دهید . برای نمونه :

```
print (''' Hello World .
 I am Mohammad Shariati Mehr and
I have 28 years old ''')
```



که خروجی آن به صورت زیر در چند خط جداگانه چاپ می‌شود . این کار موجب می‌شود که خوانایی متن خود را افزایش دهید .


Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
=======
>>>
Hello World .
I am Mohammad Shariati Mehr and
I have 28 years old
>>>
Ln: 9 Col: 4

عبور از کاراکترها

برخی اوقات شاید لازم باشد که برخی کاراکترهای چاپ نشدنی را مثل کاراکتر تب یا خط جدید را چاپ کنیم . در این شرایط نیاز دارید که از کاراکترهای عبور مثل بک اسلش (\) استفاده کنیم .

برای نمونه برای چاپ یک tab به این صورت می نویسیم \t در صورتی که قبل از t بک اسلش نگذاریم خود کاراکتر t چاپ می‌شود . برای چاپ تب به صورت زیر عمل می‌کنیم :

```
print('Hello \t World')
```

دیگر موارد استفاده از کاراکتر بک اسلش به صورت زیر می‌باشد :

\n که یک خط جدید را چاپ می‌کند

```
print ('Hello \n World')
```

خروجی آن

```
Hello
World
```

\\ خود کاراکتر بک اسلش را چاپ می‌کند

```
print ('\\')
```

خروجی آن

```
\
```

دابل کوتیشن را در شرایطی چاپ می‌کند که با کوتیشن های دیگر تداخلی ایجاد نکند "\

```
print (''I am 5'9\" tall'')
```

خروجی آن

```
I am 5'9'' tall
```

اگر که نمی‌خواهیم کاراکتر \ به عنوان یک کاراکتر ویژه تفسیر شود ، می‌توانید قبل از آن r را اضافه کرده که به صورت raw نمایش داده شود . برای نمونه اگر نمی‌خواهید t\ به عنوان یک کاراکتر تب تفسیر شود ، بایستی به صورت زیر بنویسید :

```
print (r 'Hello \t World')
```

در نتیجه آن خروجی زیر بدست می‌آید یعنی کاراکتر t\ به صورت raw خام چاپ می‌شود :

```
Hello \t World
```

فصل شش

ایجاد انتخاب

و تصمیم

ایجاد انتخاب و تصمیم

در این فصل تلاش می کنیم تا برنامه خود را هوشمندانه تر کنیم و قابلیت تصمیم گیری و انتخاب را به آن اضافه کنیم . این قابلیت‌ها را با استفاده از عبارات for،if و while ایجاد می کنیم . این‌ها را ابزارهای کنترل جریان برنامه می نامند . این ابزارها جریان برنامه را کنترل می کنند .

بعلاوه عبارات try , except را بررسی می کنیم . این عبارات تشخیص می‌دهند که هنگام بروز خطا در برنامه چه اتفاقی بایستی رخ دهد .

عبارات شرطی

همه ابزارهای کنترل جریان شامل یک عبارت شرطی برای سنجش هستند . در این حالت برنامه شما در شرایط مختلف عمل‌کرد متفاوتی خواهد داشت .

رایج ترین عبارت شرطی ، عبارت مقایسه می‌باشد . اگر که بخواهیم برابر بودن دو متغیر را مقایسه کنیم ، از علامت == برای مقایسه استفاده می‌کنیم . مثلاً اگر بنویسیم x == y در حقیقت دارید از برنامه می‌پرسید که آیا x با y برابر است یا خیر . اگر که برابر باشند شرط برقرار می‌شود و عبارت خروجی True را بر می گرداند و اگر برابر نباشند شرط برقرار نشده و خروجی False خواهد بود .

دیگر علامت های مقایسه عبارت اند از =! (مقایسه عدم برابری) ، < (مقایسه کوچکتر بودن) ، > (مقایسه بزرگ‌تر بودن) ، =< (مقایسه کوچکتر یا برابری) ، >= (مقایسه بزرگ‌تر یا برابری) .

در زیر لیستی از نحوه استفاده این علامت ها مثال‌هایی را برای درک بهتر نشان می‌دهیم :

عدم برابری

```
5  != 2
```

بزرگ‌تر بودن

```
5 > 2
```

کوچکتر بودن

```
2 < 5
```

بزرگ‌تر یا مساوی بودن

```
5>=2
5>=5
```

کوچکتر یا مساوی بودن

```
2<=5
2<=2
```

علاوه بر علامت های ذکر شده ، عملگرهای (و and ، یا or ، نفی not) در شرط های ترکیبی مفید هستند .

عملگر and فقط در صورتی که همه شرط ها برقرار باشند خروجی True باز می گرداند در غیر اینصورت خروجی False خواهد بود . برای مثال :

```
5==5 and 2>1
```

خروجی true را باز می گرداند چرا که هر دو شرط برقرار است .

عملگر or تنها در صورتی که یکی از شرط ها برقرار باشد true باز می گرداند .در غیر اینصورت خروجی false باز می گرداند . برای مثال :

```
5>2 or 7>10 or 3==2
```

خروجی true باز می گرداند چرا که حداقل عبارت 5>2 برقرار است .

عملگر not در صورتی که شرط پس از not برقرار نباشد ، خروجی true را باز می گرداند . در غیراینصورت خروجی false باز می گرداند . برای مثال :

```
not 2>5
```

خروجی true را باز می گرداند چرا که شرط 2>5 برقرار نیست و ۲ بزرگ‌تر از ۵ نیست .

عبارت شرطی if

عبارت شرطی if یکی از رایج ترین عبارات شرطی برای کنترل جریان برنامه می‌باشد . این عبارت برنامه را قادر می‌سازد تا وقتی برخی شرط ها بوجود آمد ، بر اساس شرط مورد نظر و نتیجه بدست آمده عکس‌العمل مناسب را انجام دهد .ساختار دستوری عبارت if به صورت زیر می‌باشد :

if شرط شماره ۱ برقرار شد :
```
do A
```
elif شرط شماره ۲ برقرار شد :
```
do B
```
elif شرط شماره ۳ برقرار شد :
```
do C
```
elif شرط شماره ۴ برقرار شد :
```
do D
else:
do E
```

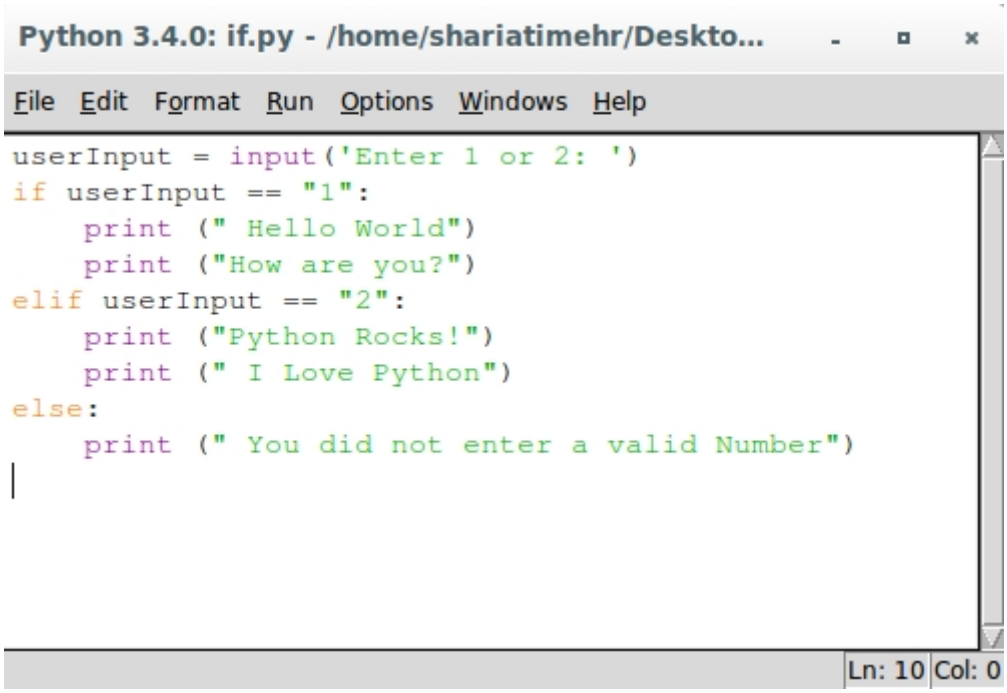
عبارت elif همان else if می‌باشد به معنی در غیر اینصورت می‌باشد . در زبان‌های برنامه نویسی دیگر مثل C یا جاوا ساختار آن به شکل دیگری است و پرانتز وجود دارد یا براکت . در حالی که در پایتون نیازی به پرانتز یا براکت بعد از if یا elif یا else نیست . پایتون از براکت استفاده نمی کند . به جای آن برای شروع یا پایان عبارت شرطی از تورفتگی استفاده می کند .

هر چیزی که دارای تورفتگی (Indentation) باشد ، به عنوان یک بلاک کد در نظر گرفته می‌شود که در صورت برقرار بودن شرط بلاک کد اجرا می‌شود .

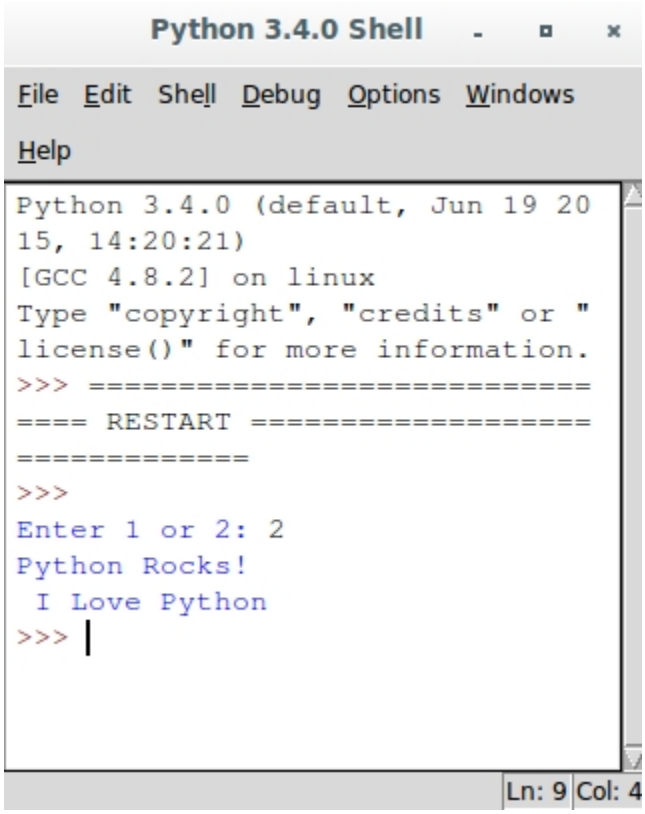
به منظور ایجاد تورفتگی های صحیح و عدم بروز خطا در کدنویسی توصیه می‌شود از یک ادیتور مناسب پایتون مثل IDLE استفاده کنید .

برای اینکه درک درستی از نحوه اجرای شرط if پیدا کنید ، IDLE را باز کنید و کد زیر را درون آن اجرا کنید :

```
userInput = input('Enter 1 or 2: ')
if userInput == "1":
    print (" Hello World")
    print ("How are you?")
elif userInput == "2":
    print ("Python Rocks!")
    print (" I Love Python")
else:
    print (" You did not enter a valid
Number")
```



برنامه ابتدا از کاربر می‌خواهد که یکی از اعداد ۱ یا ۲ را وارد کند . در صورتی که عدد ۱ را وارد کند برنامه یک خروجی را نشان می‌دهد و در صورتی که عدد را وارد کند خروجی متفاوت و در صورتی که عدد اشتباهی را وارد کند پیام خطایی را نشان می‌دهد .



عبارت شرطی Inline If

عبارت شرطی inline if شکلی ساده‌تر از عبارت شرطی if می‌باشد و برای انجام وظایف ساده‌تر بسیار مناسب می‌باشد . ساختار دستور inline If به صورت زیر می‌باشد :

وظیفه شماره ۱ را انجام بده else شرط برقرار بود if وظیفه شماره ۱ را انجام بده

برای مثال :

```
num1 = 12 if myInt==10 else 13
```

این عبارت عدد ۱۲ را به متغیر num1 اختصاص داده اگر متغیر myInt برابر ۱۰ باشد در غیر اینصورت عدد ۱۳ را به num1 اختصاص می‌دهد . مثالی دیگر :

```
print ('' This is task A'' if myInt == 10
else ''This is task B'')
```

این عبارت در صورتی که متغیر myInt برابر ۱۰ باشد عبارت This is task A را چاپ می‌کند و در غیراینصورت عبارت This is task B را چاپ می‌کند .

حلقه for

حلقه for یک بلاک کد را به صورت تکراری اجرا کرده تا اینکه شرط دیگر معتبر نباشد .

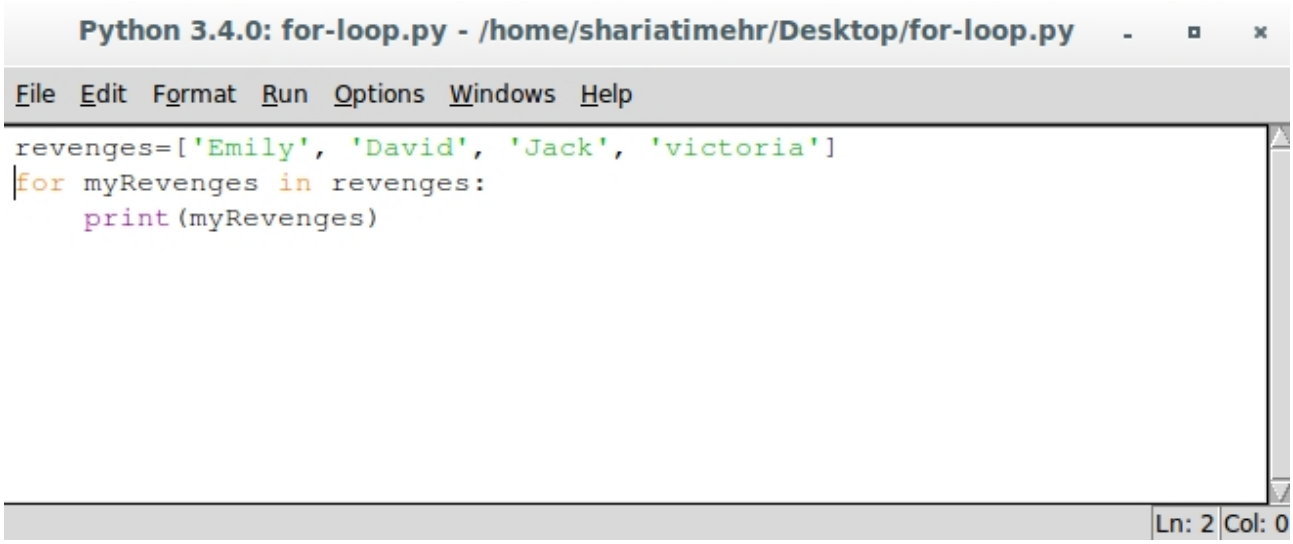
ایجاد حلقه از طریق عنصر یک تکرارشدنی

در پایتون هرچیزی که بتواند تکرار شود را تکرارشدنی می‌گویند . مثل یک رشته یا یک لیست یا یک تاپل . سینتکس ایجاد یک حلقه به صورت زیر می‌باشد :

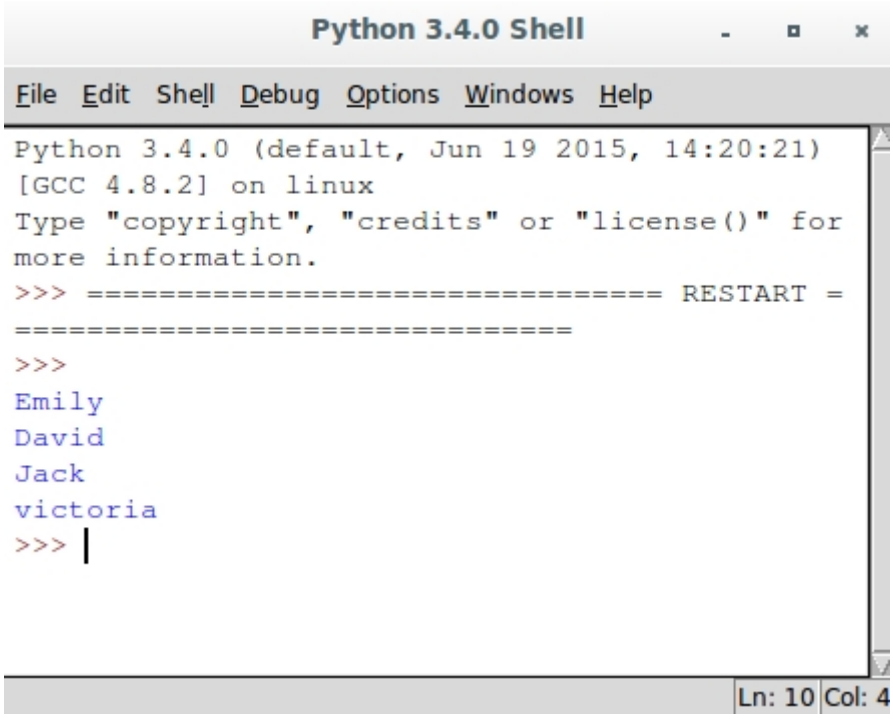
```
for a in iterable:
print (a)
```

برای مثال

```
revenges = ['Emily' , 'David' , 'Jack' ,
'victoria']
for myRevenges in revenges :
print (  myRevenges )
```

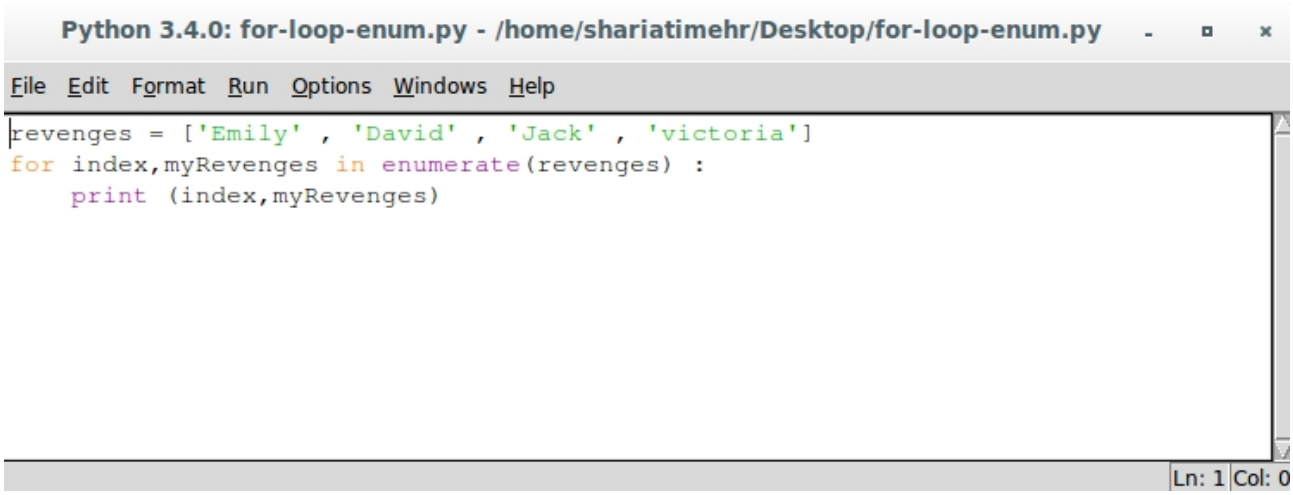


در مثال بالا ابتدا لیستی با نام revenges را تعریف کردیم و چهار عضو را درون آن قرار دادیم . سپس در خط بعد با استفاده از حلقه for هر کدام از اعضا را به ترتیب از لیست revenges بیرون کشیده و درون متغیر myRevenges قرار می‌دهیم . مثلاً اولین عضو یعنی emily را از لیست revenges بیرون کشیده و درون متغیر myRevenges ریخته در خط بعدی با استفاده تابع پرینت این عضو را چاپ می‌کنیم . چون لیست ما هنوز ۳ عضو دیگر دارد پس شرط باقی است و همین فرایند را برای سایر اعضای لیست انجام می‌دهیم و آن‌ها هم چاپ می‌شوند . وقتی که هیچ عضوی از لیست باقی نماند ، حلقه ما هم پایان می‌پذیرد .

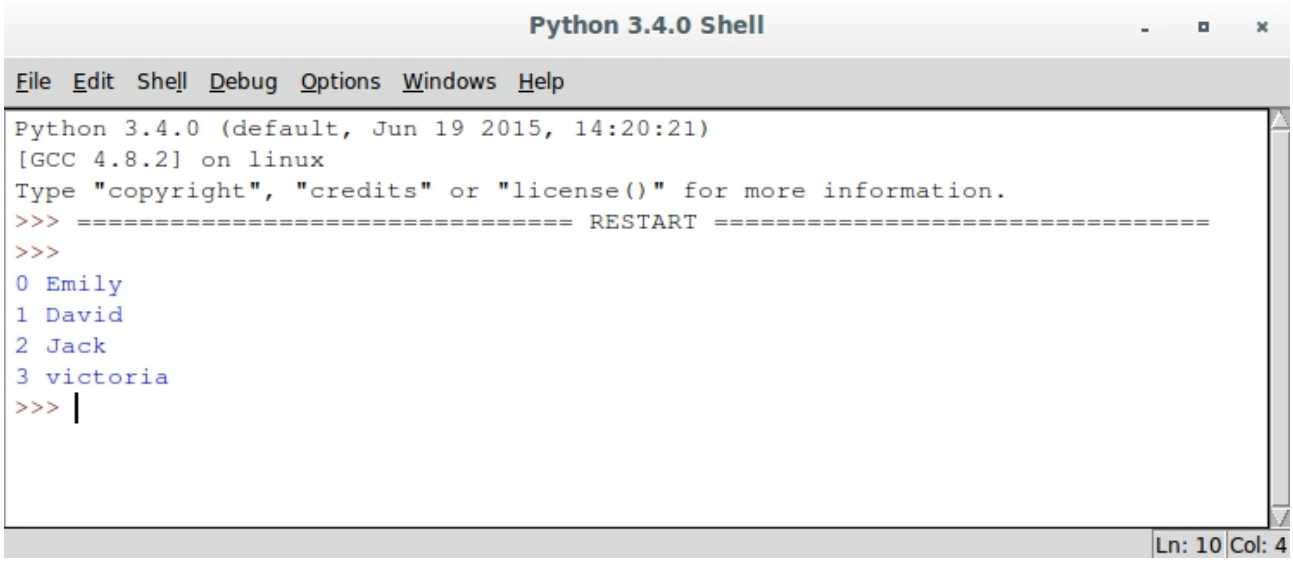


به منظور نمایش ایندکس اعضا می‌توانیم از تابع enumerate استفاده کنیم :

```
revenges = ['Emily' , 'David' , 'Jack' ,
'victoria']
for index,myRevenges in enumerate(revenges)
:
    print (index,myRevenges)
```



در این حالت علاوه بر متغیر myRevenges ، متغیر دیگری با نام index نیز اضافه می‌شود . همچنین لیست revenges را با استفاده از تابع enumerate سرشماری می‌کنیم . اتفاقی که رخ می‌دهد این است که علاوه برداشتن اعضا از لیست و قرار دادن آن‌ها در متغیر myRevenges برای چاپ ، ایندکس آن‌ها توسط تابع enumerate به متغیر index برای چاپ داده می‌شود .



در مثال بعدی مشاهده می‌کنید که چگونه درون یک حلقه وارد می‌شویم :

```
message = 'Hello'
for i in message:
print (i)
```

خروجی مثال بالا می‌شود :

```
H
e
l
l
o
```

در مثال بالا ابتدا متغیری با نام message ایجاد می‌کنیم . سپس مقدار Hello را به آن اختصاص می‌دهیم . اکنون این بار به جای اینکه درون اعضای یک لیست برای تکرار حلقه بگردیم . کاراکتر های متغیر را انتخاب می‌کنیم . در مثال بالا هر بار یکی از کاراکترهای عبارت Hello از متغیر message درون متغیر i ریخته می‌شود و به ترتیب چاپ می‌گردد .

ایجاد حلقه از طریق تکرار بین اعداد

برای ایجاد حلقه از طریق توالی بین اعداد ، تابع درون ساخت range بسیار کارآمد است . تابع range لیستی از اعداد را ایجاد می کند و ساختار دستوری آن به شکل زیر می‌باشد

```
range (start, end, step)
```

اگر مقدار start داده نشود ، شروع اعداد از صفر خواهد بود . نکته قابل توجه در پایتون این است که در بیشتر موارد ما از مقدار صفر شروع خواهیم کرد .
برای مثال ایندکس یک لیست و یک تاپل از صفر شروع می‌شود . در هنگام استفاده از متد format برای رشته موقعیت پارامترها هم از صفر آغاز می‌شود . در هنگام استفاده از تابع range اگر که مقدار start تعیین نشود ، اعداد از ایندکس صفر ایجاد می‌شوند . step چطور ؟ اگر مقدار step تعیین نگردد یک لیست متوالی و پشت سرهم از اعداد ایجاد خواهند شد . ‹برای مثالstep مقدار ۱ در نظر گرفته می‌شود › . مقدار end بایستی حتماً تعیین گردد . هرچند که یک نکته عجیب در باره تابع range وجود دارد و اینکه مقدار تعیین شده end هیچ وقت بخشی از لیست ایجاد شده اعداد نخواهد بود .
برای نمونه :

```
range (5)
```

لیستی از اعداد را به صورت زیر ایجاد می کند

```
[0, 1, 2, 3, 4]
```

```
range(3, 10)
```

لیستی از اعداد را به صورت زیر ایجاد می کند

```
[3, 4, 5, 6, 7, 8, 9]
```

```
range(4, 10, 2)
```

لیستی از اعداد به صورت زیر ایجاد می کند

```
[4, 6, 8]
```

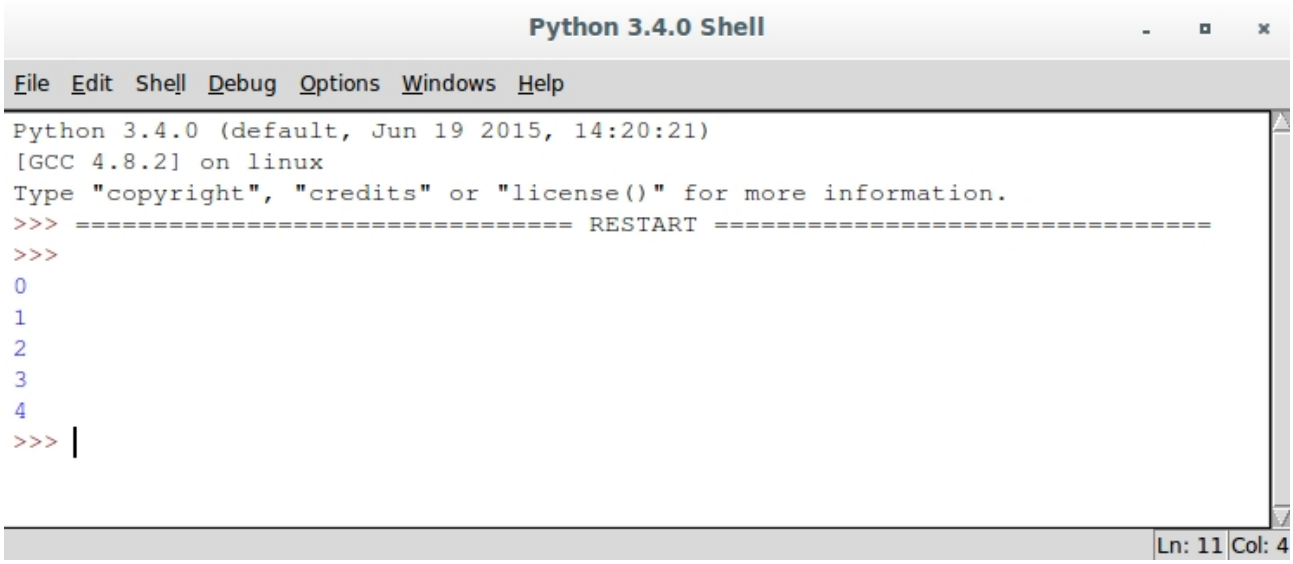
برای مشاهده نحوه عمل‌کرد تابع range در یک عبارت for مثال زیر را اجرا کنید :

```
for i in range (5) :
print (i)
```

در مثال بالا ابتدا با استفاده از تابع range لیستی از اعداد به صورت زیر ایجاد می‌شود :

```
[0,1,2,3,4]
```

سپس چون حلقه for را بکارگرفته ایم ، تک تک این اعداد به ترتیب درون متغیر i ریخته می‌شوند و با استفاده از تابع print چاپ می‌شوند .



while حلقه

عبارت کنترل جریان دیگر که استفاده خواهیم کرد حلقه while می‌باشد . همانطور که از نام while بر می‌آید یک حلقه به صورت تکراری دستورالعمل های درون حلقه را اجرا می‌کند

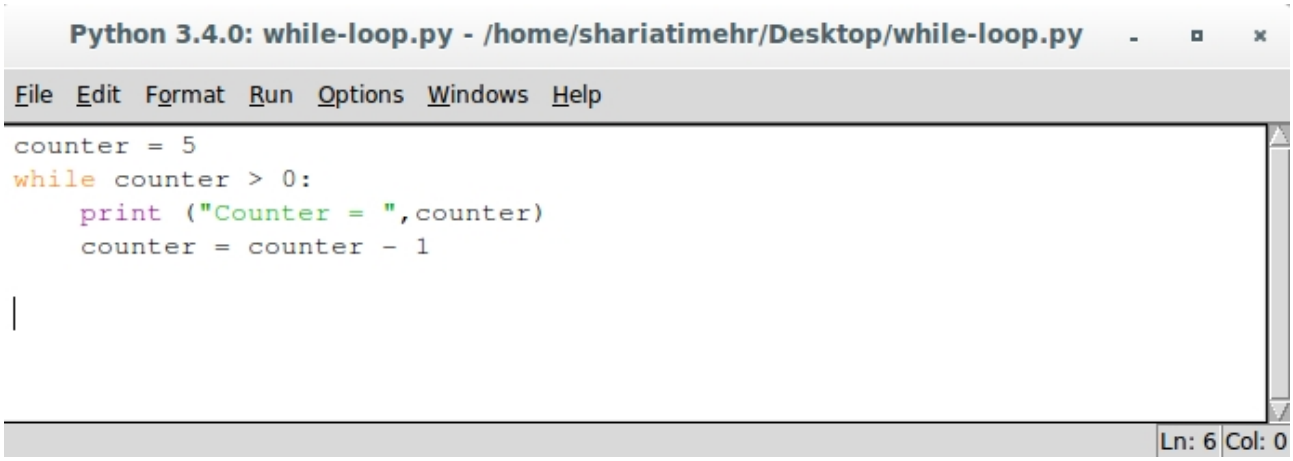
!!!!!!! تا زمانیکه !!!!!! برخی شرایط معتبر باقی بماند . ساختار دستوری حلقه while به صورت زیر می‌باشد :

while شرط برقرار باشد :

این کار را انجام بده

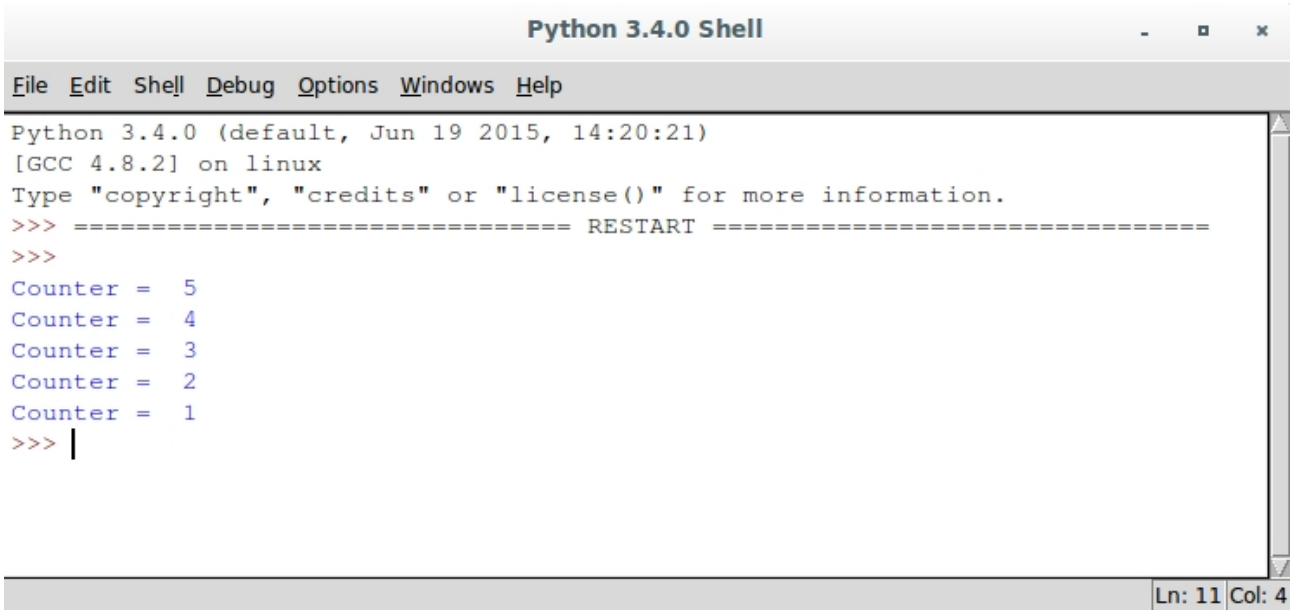
بیشتر اوقات وقتی که از یک حلقه while استفاده می‌کنیم نیاز به یک کانتر یا شمارنده برای عمل‌کرد درست حلقه خواهیم داشت . این شمارنده را counter می‌نامیم . شرط حلقه مقدار counter را ارزیابی کرده تا تشخیص دهد که آیا از مقداری بخصوص بزرگ‌تر یا کوچکتر است . اگر اینگونه بود حلقه اجرا می‌شود در غیر اینصورت حلقه پایان می‌پذیرد . یک مثال همیشه گویاتر خواهد بود . به مثال زیر توجه کنید :

```
counter = 5
while counter > 0:
print (''Counter ='',counter)
counter = counter - 1
```



در مثال بالا ابتدا یک مقدار پیش‌فرض در اینجا 5 به counter می‌دهیم . سپس در حلقه شرط می‌کنیم که تا وقتی که که مقدار counter از عدد صفر بزرگ‌تر است حلقه ادامه پیدا کند .

در ادامه بلوک اصلی حلقه که خروجی آن می‌باشد را چاپ می‌کنیم . در اینجا با استفاده از تابع print مقدار کنونی counter را در خروجی چاپ می‌کنیم . در این شرایط برای اینکه حلقه ما بی پایان نباشد در خط آخر هر دفعه یک مقدار از counter کم می‌کنیم تا پس از چند بار اجرا مقدار آن صفر شود و دیگر حلقه ادامه نیابد .



کلمه کلیدی break

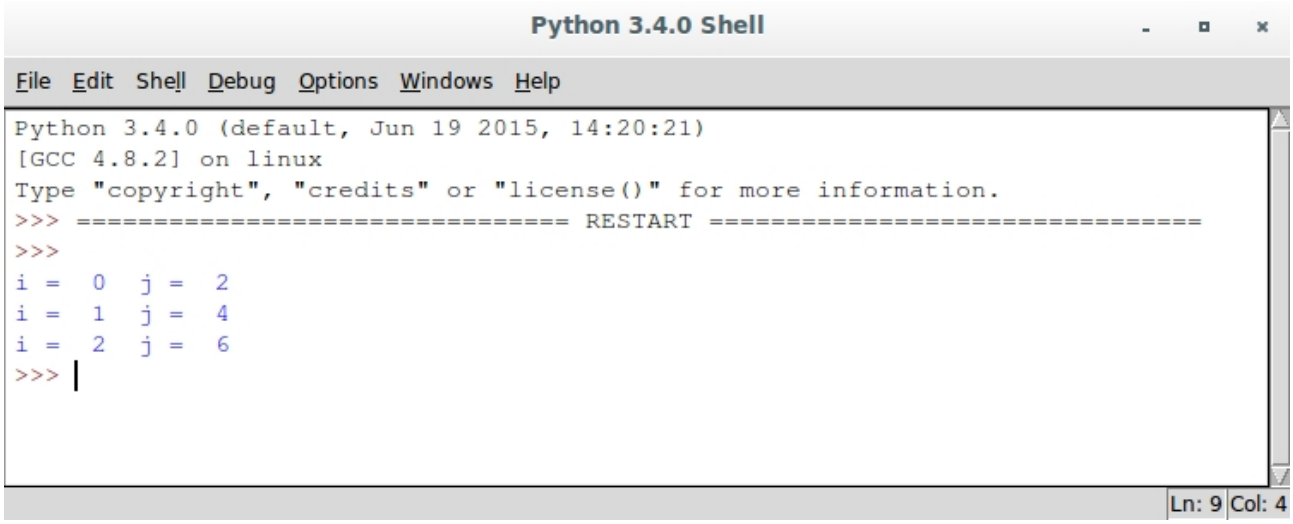
زمانی که با حلقه ها کار می‌کنیم ، گاهی اوقات نیاز است تا در صورت بوجود آمدن یک شرط خاص از کل حلقه خارج شویم . به این منظور از کلمه کلید break استفاده می‌کنیم . برنامه زیر را اجرا کنید تا با نحوه عمل‌کرد break آشنا شوید .

```
j=0
for i in range(5):
    j=j+2
    print ('i = ', i, ' j = ', j)
    if j == 6:
        break
```

در مثال بالا ابتدا به متغیر j مقدار پیش‌فرض صفر را می‌دهیم . سپس یک حلقه for ایجاد می‌کنیم . در این حلقه با استفاده از تابع range که قبلاً توضیح دادیم لیست از اعداد را ایجاد می‌کنیم و برای چاپ ترتیبی حلقه درون متغیر i می‌ریزیم .

در خط بعد متغیری با نام j تعریف می‌کنیم و هر بار مقدار آن را بعلاوه دو می‌کنیم. چون مقدار پیش‌فرض آن صفر است بار اول مقدار ۲ بار دوم مقدار ۴ بار سوم مقدار ۶ و ...

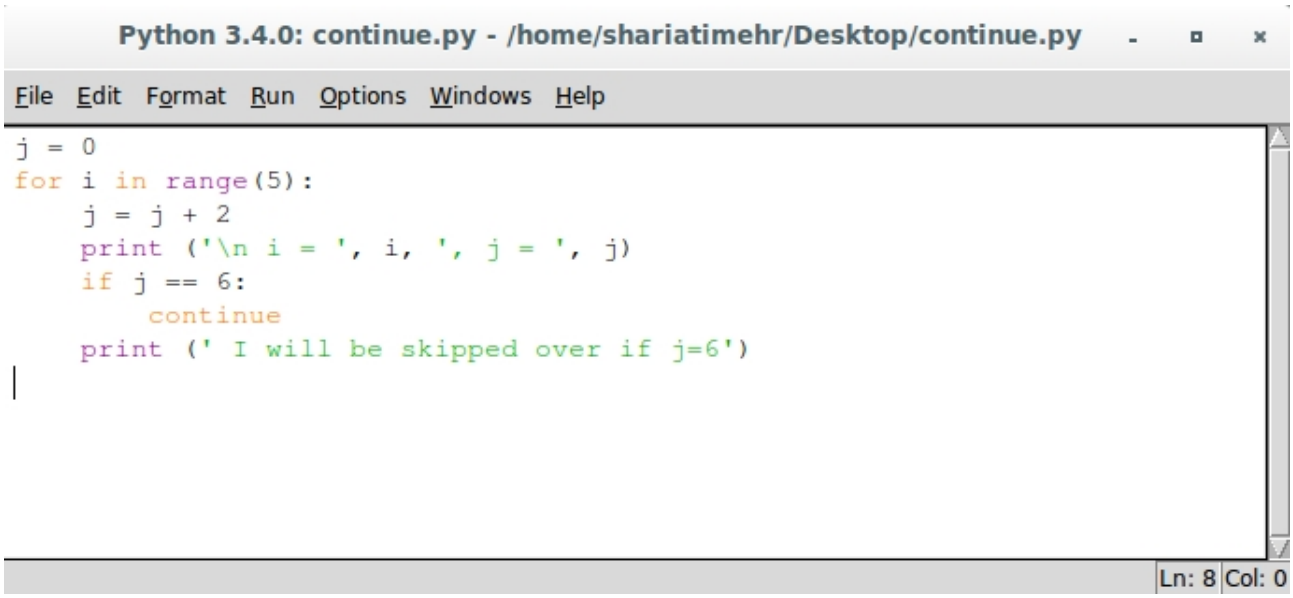
در خط بعدی با استفاده از تابع پرینت مقدار کنونی دو متغیر i و j را چاپ می‌کنیم. نکته مهم این برنامه این است که اگر مقدار متغیر جی برابر ۶ شد (if j == 6) ، با استفاده از کلمه کلید break حلقه پایان پذیرد. پس خروجی به صورت زیر حاصل می‌شود :



کلمه کلیدی continue

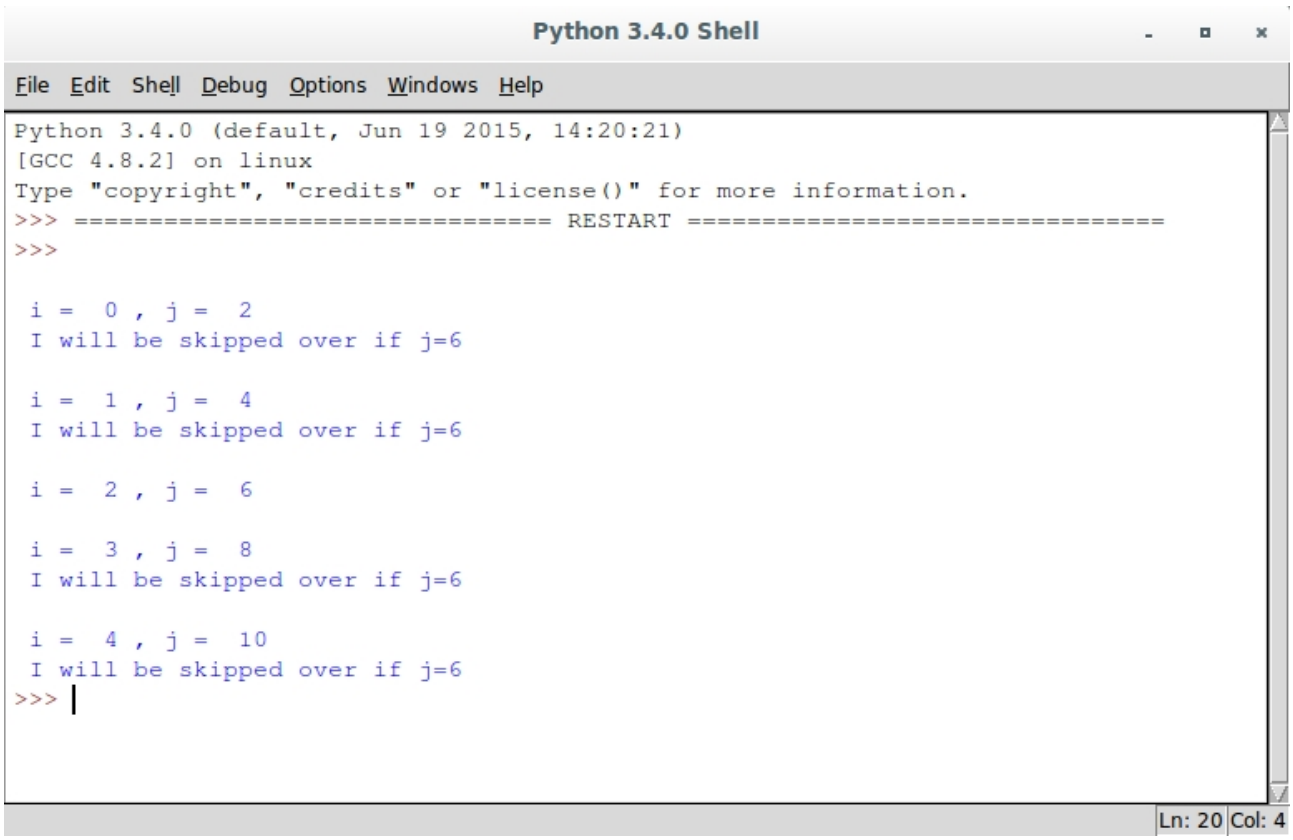
یک کلمه کلیدی دیگر برای ایجاد حلقه ها continue می‌باشد . وقتی که ما از continue استفاده می کنیم ، بقیه حلقه که پس از continue آمده فقط برای همان تکرار نادیده گرفته می‌شود . مثال زیر را بررسی و تحلیل کنید :

```
j = 0
for i in range(5):
    j = j + 2
    print ('\n i = ', i, ', j = ', j)
    if j == 6:
        continue
    print (' I will be skipped over if j=6')
```



در مثال بالا وقتی که مقدار j برابر ۶ می‌شود عبارت آخر چاپ نمی‌شود ولی در بقیه حالات عبارت زیر چاپ می‌شود :

```
I will be skipped over if j=6
```



عبارت کنترلی try, except

آخرین عبارت کنترلی که به آن خواهیم پرداخت try, except می‌باشد . این عبارت کنترلی ، نحوه پردازش برنامه را در زمان مواجهه با خطاها کنترل می‌کند . ساختار دستوری آن به صورت زیر می‌باشد :

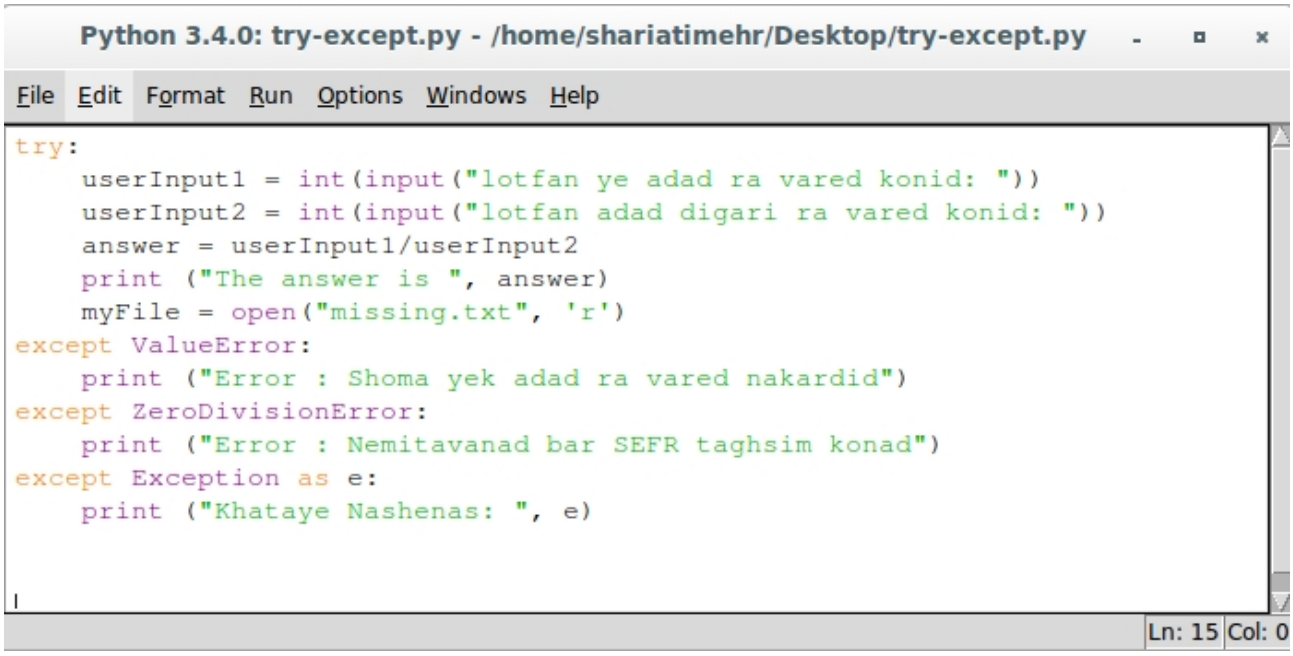
```
try:
یک کاری را انجام بده
except:
در مواجهه با خطا کار دیگری را انجام بده
```

برای مثال :

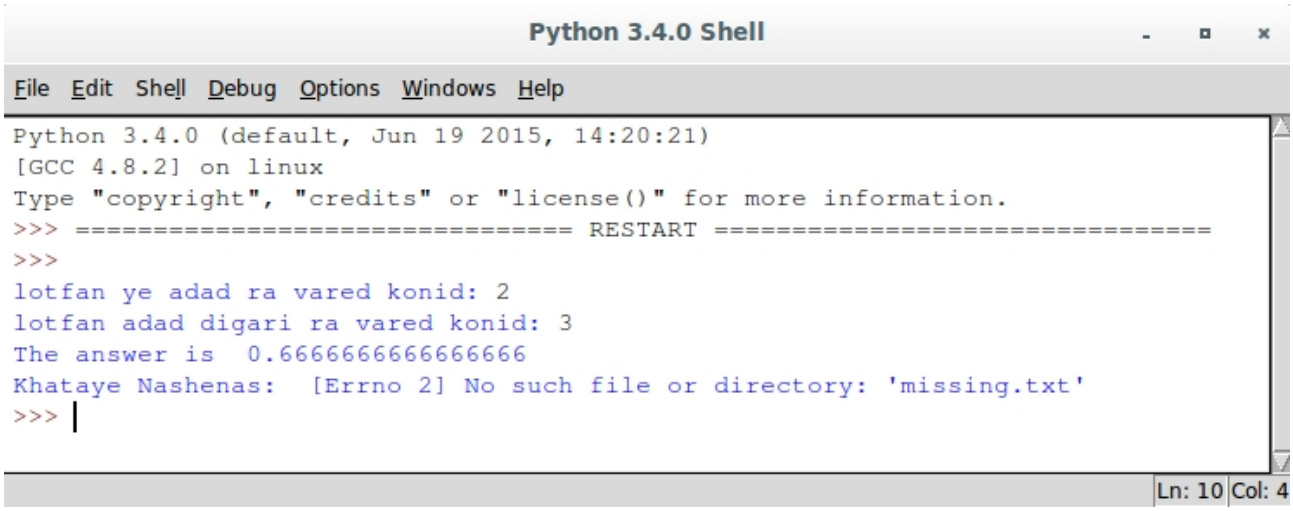
```
try:
answer = 12/0
print (answer)
except:
print (''An error occurred'')
```

در حین اجرای برنامه بالا پیام خطای An error occurred چاپ می‌شود . چرا ؟ به این دلیل که زمانیکه برنامه سعی می‌کند که عبارت answer = 12/0 را اجرا کند نمی‌تواند چرا که اعداد بر صفر تقسیم نمی‌شوند . در نتیجه به جای نمایش باقی بلاک try نادیده گرفته می‌شود و بلاک except چاپ می‌شود . اگر که می‌خواهید پیام‌های خطای اختصاصی تری را به کاربران نمایش دهید می‌توانید نوع خطا را پس از کلمه کلیدی except اختصاص دهید . مثال زیر را امتحان کنید

```
try:
    userInput1 = int(input("lotfan ye adad
ra vared konid: "))
    userInput2 = int(input("lotfan adad
digari ra vared konid: "))
    answer = userInput1/userInput2
    print ("The answer is ", answer)
    myFile = open("missing.txt", 'r')
except ValueError:
    print ("Error : Shoma yek adad ra vared
nakardid")
except ZeroDivisionError:
    print ("Error : Nemitavanad bar SEFR
taghsim konad")
except Exception as e:
    print ("Khataye Nashenas: ", e)
```



در عبارت بالا سه ساختار متفاوت خطا را بررسی کردیم.



برای مشاهده خطاهای مختلف به این آدرس مراجعه کنید

فصل هفت

توابع و ماژول ها

توابع و ماژول ها

در فصل قبل به صورت مختصر درباره توابع و ماژول ها صحبت کردیم . در این فصل به صورت جزیی تر درباره ماژول ها و توابع صحبت خواهیم کرد . تمام زبان‌های برنامه نویسی دارای یکسری کدهای درون ساخت و آماده هستند که می‌توانیم از آن‌ها به منظور ایجاد برنامه‌ها استفاده کنیم و این موجب می‌شود که کار برنامه نویسان آسان‌تر شود . این کدها شامل کلاس‌های از پیش نوشته شده ، متغیرها و توابع به منظور انجام برخی وظایف هستند که درون فایل‌هایی با نام ماژول ها ذخیره می‌شوند . خوب بیایید ابتدا نگاهی به توابع بیندازیم و در ادامه به ماژول ها خواهیم پرداخت .

توابع چه هستند ؟

توابع کدهای از پیش نوشته شده‌ای هستند که یک وظیفه بخصوص را انجام می‌دهند . برای یک مقایسه توابع ریاضی موجود در مایکروسافت اکسل را در نظر بگیرید . برای انجام عمل جمع می‌توانیم از تابع sum استفاده کنیم و بنویسیم (sum(A1:A5 ،به جای اینکه بنویسیم A1+A2+A3+A4+A5+A6،

بسته به این موضوع که تابع چگونه نوشته شده است ، ممکن است بخشی از یک کلاس باشد (کلاس موضوعی است در برنامه نویسی شی گرا که در این کتاب به آن نمی پردازیم) و اینکه چگونه تابع وارد می‌کنیم ، ما می‌توانیم تابع را نوشته نام آن و به همراه نقطه قبل از آن فراخوانی کنیم . برخی توابع به منظور انجام وظایفشان نیاز به پاس دادن برخی داده‌ها دارند . این داده‌ها را با نام پارامتر می‌شناسیم . برای پاس دادن پارامترها به توابع مقادیر آن را درون پرانتز قرار داده و با کاما از هم جدا می‌کنیم . برای مثال برای استفاده از تابع print آن را به صورت زیر می نویسیم :

```
print ('' Hello World'')
```

که در این مثال print نام تابع است و " Hello World " پارامتری است که درون آن قرار می‌دهیم .

از طرف دیگر برای استفاده از تابع replace به منظور دستکاری رشته‌ها ، بایستی بنویسیم :

```
''Hello World''.replace(''World'' ,
''Universe'')
```

که در این تابع replace نام تابع است و "World" و "Universe" پارامترهای آن هستند . رشته قبل از علامت نقطه رشته ای است که تحت تأثیر تابع قرار خواهد گرفت . در نتیجه رشته ما با صورت زیر تغییر می‌کند :

```
''Hello Universe''
```

تعریف توابع خودتان

ما می‌توانیم در پایتون توابع خود را تعریف کنیم و در برنامه‌ها از آن‌ها استفاده کنیم .

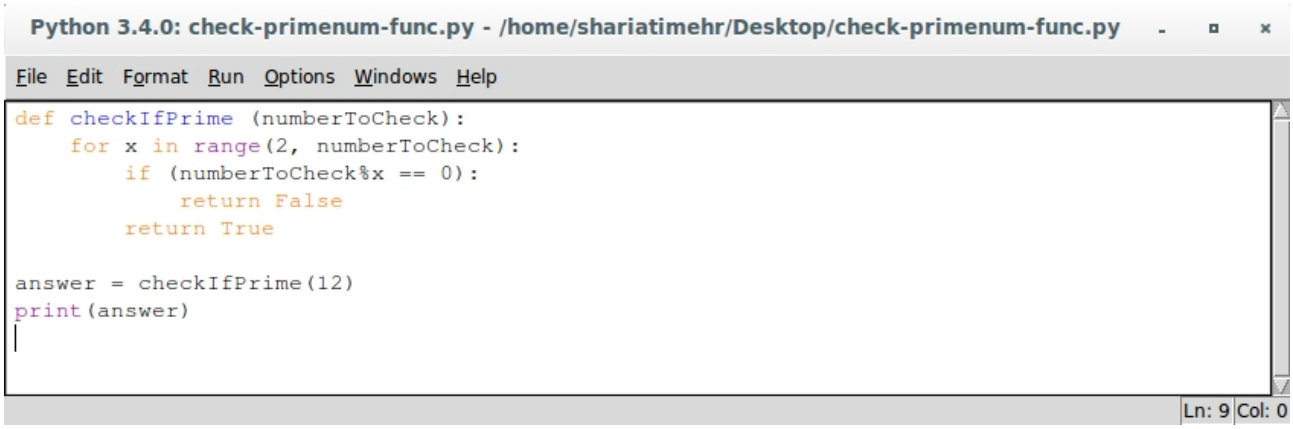
ساختار دستوری برای تعریف یک تابع به صورت زیر می‌باشد :

```
def نام تابع (parameters) :
کد تابع
return [expression]
```

در ساختار دستوری بالا def به برنامه می‌گوید که کدی که در خط بعد می‌آید و دارای تورفتگی می‌باشد ، بخشی از یک تابع است . Return کلمه کلیدی است که به منظور خروجی دادن یک پاسخ از تابع استفاده می‌شود . در یک تابع می‌توانیم بیش از یک return داشته باشیم . هرچند که زمانی که تابع یک عبارت return را اجرا می‌کند ، تابع پایان می‌پذیرد و خارج می‌شود . اگر که تابع شما نیاز به خروجی دادن return هیچ مقداری ندارد ، می‌توانید عبارت return را حذف کنید .

خوب اکنون می‌خواهیم اولین تابع خود را ایجاد کنیم . فرض کنید می‌خواهیم تشخیص دهیم که عدد ما عدد اول است یا خیر . در اینجا با استفاده از عملگر % که یادگرفتیم و حلقه for و عبارت شرطی if یک تابع ایجاد می‌کنیم :

```
def checkIfPrime (numberToCheck):
    for x in range(2, numberToCheck):
        if (numberToCheck%x == 0):
            return False
        return True
```

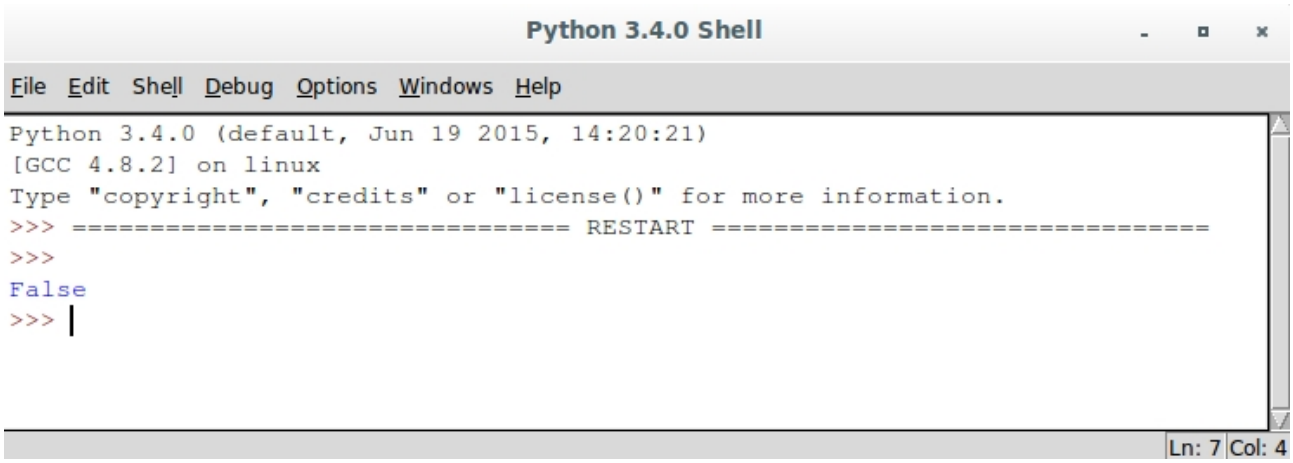


در تابع بالا در خط اول تابعی با نام checkIfPrime را تعریف کردیم و پارامتری با نام numberToCheck را به آن اختصاص دادیم . در خط دوم یک حلقه for ایجاد می‌کنیم و با استفاده از تابع درون ساخت range یک لیست از اعداد ایجاد می‌کنیم . این لیست با عدد ۲ شروع شده و به عدد numberToCheck (منهای یک چونکه عدد پایانی خودش شامل نمی شود) ختم می‌شود . این مقدار را درون متغیر x برای حلقه می‌ریزیم . سپس با استفاده از if بررسی می‌کنیم که عدد ما تقسیم بر مقدار x دارای باقی‌مانده صفر می‌باشد یا خیر . در صورتی که مقدار آن صفر شد خروجی False باز می‌گردد و از برنامه خارج می‌شود . در صورتی که با عبور از حلقه ها هیچ کدام از آن‌ها خروجی صفر را باز نگرداند ، مقدار true را بر می گرداند .

تا اینجای کار تنها تابع را تعریف کردیم . به منظور فراخوانی تابع به صورت زیر عمل می‌کنیم :

```
answer = checkIfPrime(12)
print(answer)
```

در بالا تابع را فراخوانی کرده و مقدار ۱۲ را به عنوان پارامتر به آن می‌دهیم . عدد ۱۲ جای numberToCheck می‌نشیند و سپس خروجی تابع درون متغیر answer ذخیره می‌شود و با استفاده از تابع print برای نمایش چاپ می‌شود .



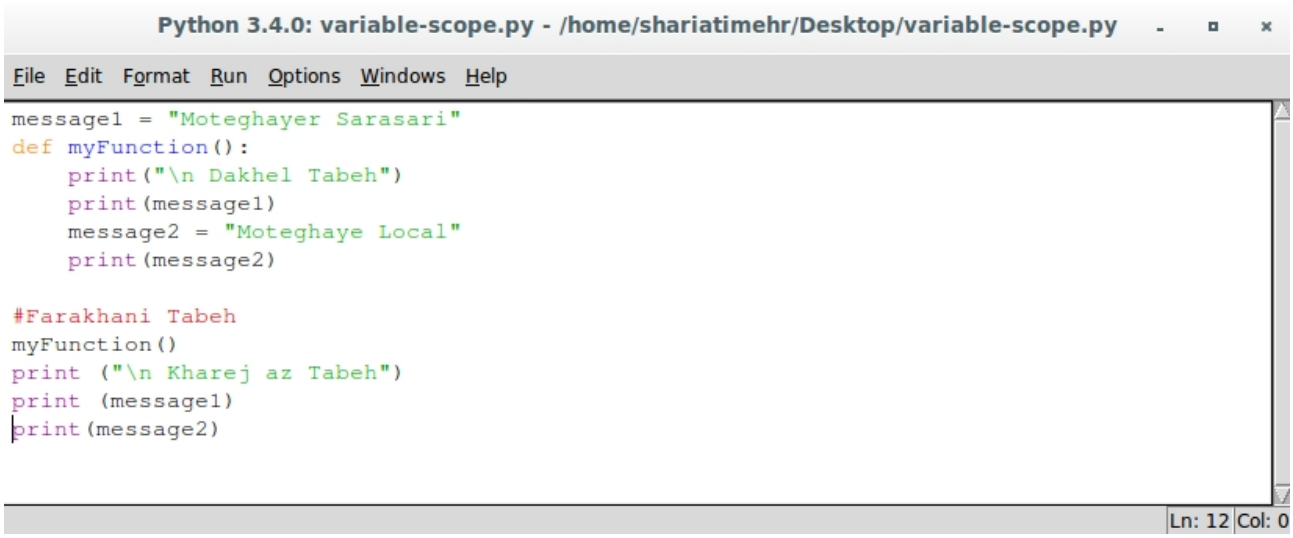
حوزه متغیرها

یک مفهوم بسیار مهم که بایستی در حین تعریف یک تابع یادبگیرید ، تعیین حوزه متغیرهاست . متغیرهایی که درون یک تابع تعریف می‌شوند به نحو متفاوتی با متغیرهای خارج از تابع تلقی می‌شوند . دو تفاوت بسیار مهم وجود دارد .

ابتدا هر متغیری که درون یک تابع اعلان می‌شود تنها درون تابع قابل دسترسی می‌باشد . این نوع متغیرها را متغیرهای لوکال یا محلی می نامند . هر متغیری که خارج از تابع اعلان شود به عنوان متغیر سراسری شناخته می‌شود و در هر کجای برنامه قابل دسترسی می‌باشد .

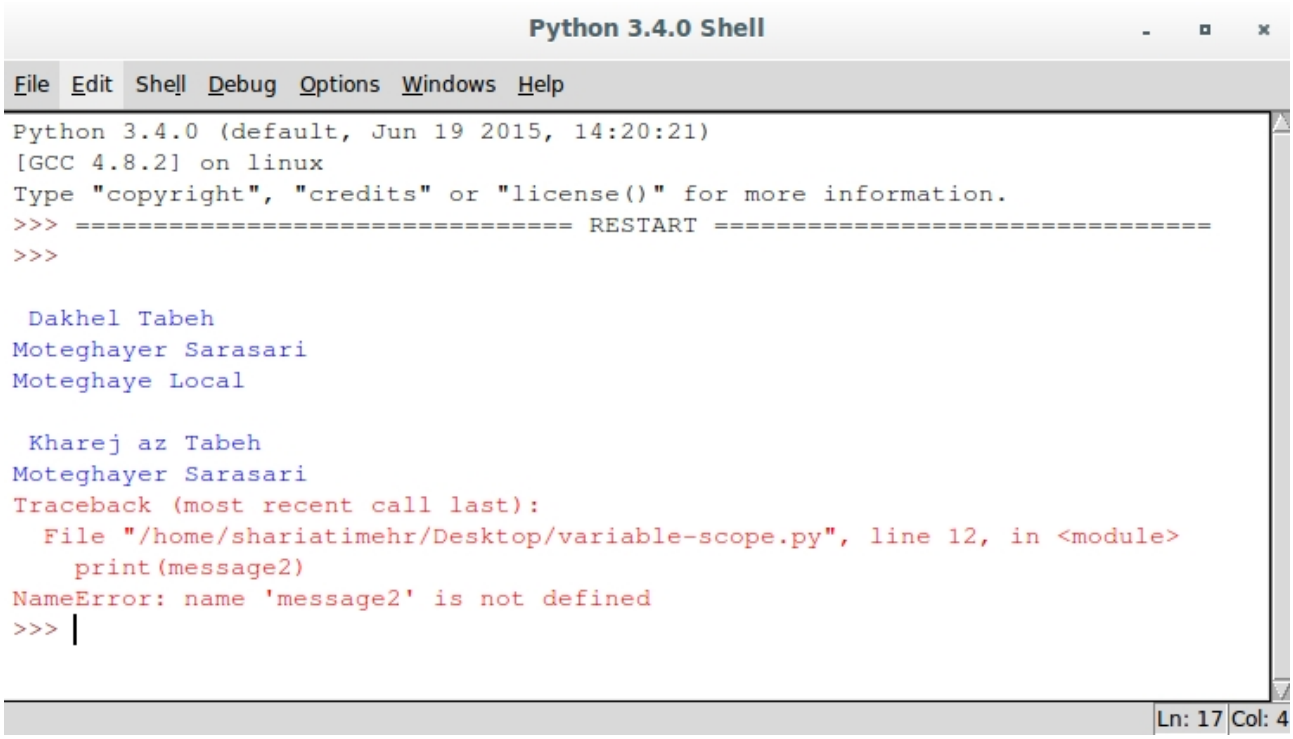
به منظور درک بهتر برنامه زیر را اجرا کنید :

```
message1 = "Moteghayer Sarasari"
def myFunction():
    print("\n Dakhel Tabeh")
    print(message1)
    message2 = "Moteghaye Local"
    print(message2)

#Farakhani Tabeh
myFunction()
print ("\n Kharej az Tabeh")
print (message1)
print(message2)
```



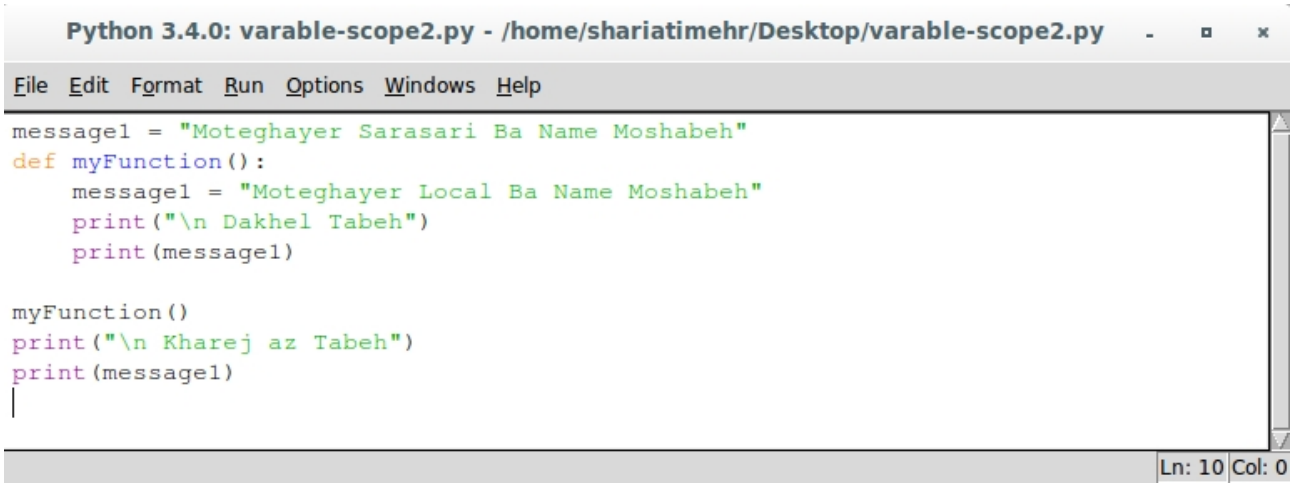
در صورتی که تابع بالا را اجرا کنیم . خروجی زیر را دریافت خواهید کرد . همانطور که می‌بینید متغیر message2 درون تابع چاپ می‌شود و قابل دسترسی است ولی خارج از تابع قابل دسترسی نمی باشد و پیام خطا چاپ می‌شود .

ولی متغیر message1 به صورت سراسری و خارج از تابع تعریف شده و در همه جای برنامه قابل دسترسی می‌باشد .



دومین مفهوم درباره حوزه متغیرها این است که اگر یک متغیر لوکال نام یکسانی را به صورت سراسری به اشتراک بگذارد ،در هر دو حوزه خارج و داخل تابع تعریف و مقداردهی شود، ، هر کدی که داخل تابع باشد به متغیر لوکال دسترسی پیدا می‌کند و هر کدی که خارج از تابع باشد به متغیر سراسری دسترسی خواهد داشت. کد زیر را امتحان کنید :

```
message1 = "Moteghayer Sarasari Ba Name
Moshabeh"
def myFunction():
    message1 = "Moteghayer Local Ba Name
Moshabeh"
    print("\n Dakhel Tabeh")
    print(message1)

myFunction()
print("\n Kharej az Tabeh")
print(message1)
```

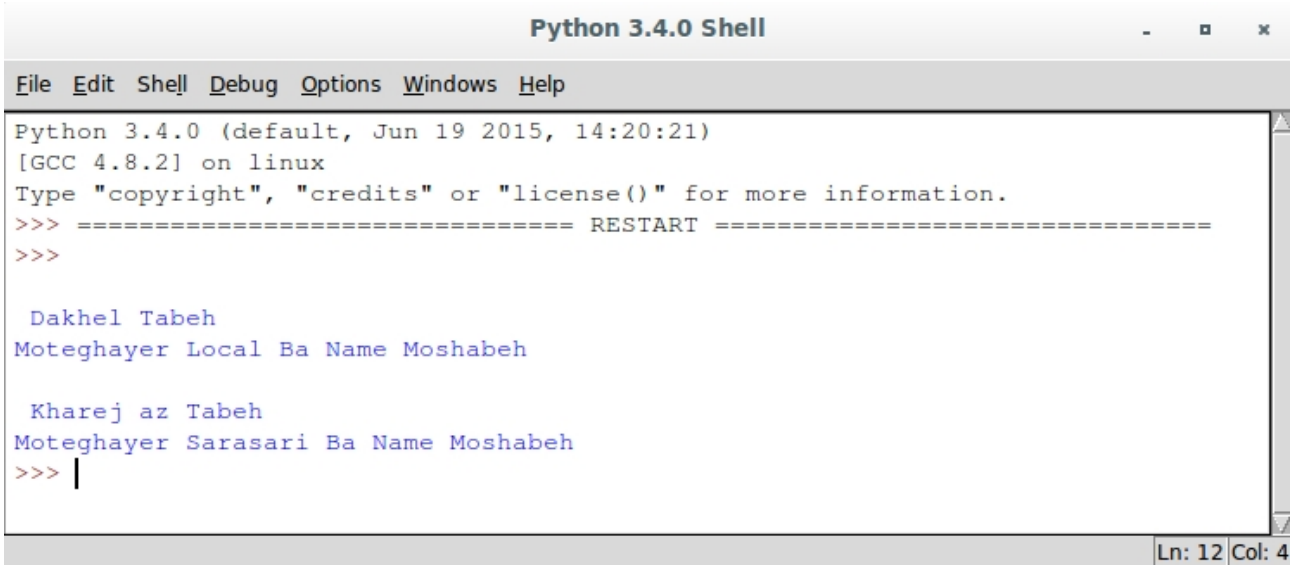


همانطور که مشاهده می‌کنید زمانی که message1 را درون تابع چاپ می‌کنیم خروجی زیر چاپ می‌شود :

```
Moteghayer Local Ba Name Moshabeh
```

و زمانی که message1 خارج از تابع چاپ می‌شود مقدار سراسری آن با نام مشابه به صورت زیر چاپ می‌شود :

```
Moteghayer Sarasari Ba Name Moshabeh
```



وارد کردن ماژول ها

پایتون دارای تعداد عظیمی از توابع درون ساخت و آماده می‌باشد . این توابع در فایل‌هایی با نام ماژول ها ذخیره می‌شوند . به منظور استفاده از کدهای درون ساخت پایتون یا همان ماژول ها ، بایستی ابتدا آن‌ها را به درون برنامه‌های خود وارد کنیم . این کار را با استفاده از کلمه کلیدی import انجام می‌دهیم . سه راه برای وارد کردن یک ماژول به برنامه می‌باشد .

اولین راه این است که کل ماژول را با نوشتن دستور زیر وارد کنیم :

```
import moduleName
```

برای مثال برای وارد کردن ماژول random می نویسیم :

```
import random
```

در این شرایط مثلاً برای استفاده از تابع randrange از ماژول random می نویسیم :

```
random.randrange(1,10)
```

اگر فکر می‌کنید که نوشتن random در هر بار نوشتن تابع خسته‌کننده است ، بهتر است شیوه دوم وارد کردن یک ماژول به برنامه را یاد بگیرید . در روش دوم شما می‌توانید ماژول را به شیوه زیر وارد کنید :

```
import random as r
```

در شیوه بالا ما ماژول random را وارد می‌کنیم ولی این بار یک نام دلخواه و کوتاه برای آن تعیین می‌کنیم . که در اینجا r هست و شما هر نام دیگری را می‌توانید انتخاب کنید . اکنون به منظور استفاده از تابع randrange کافی است از کلمه مخفف r به صورت زیر استفاده کنید :

```
r.randrange(1, 10)
```

شیوه سوم وارد کردن ماژول ها این است که به صورت اختصاصی توابع بخصوصی از ماژول را به درون برنامه خود وارد کنیم . برای مثال برای وارد کردن تابع randrange از ماژول random کافی است بنویسیم :

```
from random import randrange
```

اگر که می‌خواهید بیش از یک تابع را وارد کنید نام آن‌ها را با کاما از هم جدا می‌کنیم :

```
from random import randrange,randit
```

در این شیوه وارد کردن برای استفاده از تابع دیگر لازم نیست از علامت نقطه استفاده کنیم چرا که قبلاً در برنامه تابع به عنوان بخشی از ماژول تعریف شده است و فقط کافی است اسم تابع را به صورت زیر بنویسیم :

```
randrange(1, 10)
```

ایجاد ماژول های خودتان

جدای از وارد کردن ماژول های از پیش ساخته شده به برنامه خود در پایتون ، می‌توانید ماژول های خود را ایجاد و در برنامه‌های خود از آن‌ها استفاده کنید . این کار در شرایطی که توابعی مفید و قابل استفاده مجدد ایجاد کرده‌اید بسیار کارساز خواهد بود . ایجاد یک ماژول در پایتون کار بسیار ساده‌ای است . تنها کافی است فایل ایجاد شده خود را با پسوند py. ذخیره کنید و آن را درون همان پوشه ای که فایل برنامه وجود دارد قرار دهید . سپس به شیوه‌های قبلی می‌توانید ماژول خود را import کنید .

فرض کنید که می‌خواهید از تابع checkIfPrime که قبلاً کد آن را نوشتیم استفاده کنیم . شیوه کار به این صورت است . کد را درون فایلی با نام prime.py ذخیره می‌کنیم . این فایل دارای کد زیر می‌باشد :

```
def checkIfPrime (numberToCheck):
    for x in range(2, numberToCheck):
        if (numberToCheck%x == 0):
            return False
        return True
```

سپس یک فایل دیگر پایتون ایجاد کرده و آن را useCheckIfPrime.py می‌نامیم . این فایل را نیز در کنار فایل قبلی در پوشه ای ذخیره می‌کنیم و کد زیر را درون آن قرار می‌دهیم :

```
import prime
answer = prime.checkIfPrime(13)
print (answer)
```

اکنون کافی است تا فایل useCheckIfPrime.py را اجرا کنید تا خروجی True را دریافت کنید . به همین سادگی .

هرچند فرض کنید که می‌خواهید فایل‌های prime.py و useCheckIfPrime.py را در پوشه های متفاوتی ذخیره کنید ! برای اینکار بایستی یکسری کدها به فایل useCheckIfPrime.py اضافه کنید تا به مفسر پایتون بگوید که ماژول را از کجا پیدا کند .

فرض کنید که پوشه ای با نام MyPythonModules به مسیر زیر ایجاد کرده‌اید (من در لینوکس کار می‌کنم . اگر شما از ویندوز استفاده می‌کنید مسیر شما مثلاً درایو C خواهد بود) :

```
/home/shariatimehr/Desktop/MyPythonModules
```

برای اینکه برنامه ما بتواند مسیر فایل را پیدا کند ، بایستی قبل از import prime کد زیر را وارد کنیم :

```
import sys
if
'/home/shariatimehr/Desktop/MyPythonModules'
not in sys.path:

sys.path.append('/home/shariatimehr/Desktop/
MyPythonModules')
```

```
import prime
answer = prime.checkIfPrime(13)
print(answer)
```

این عبارت اضافه شده در حقیقت بررسی می‌کند که آیا مسیر ما در sys.path موجود هست یا خیر اگر نباشد آن را به sys.path با استفاده از تابع append اضافه می‌کند .
sys.path اشاره به مسیر سیستمی در پایتون دارد .

فصل هشت

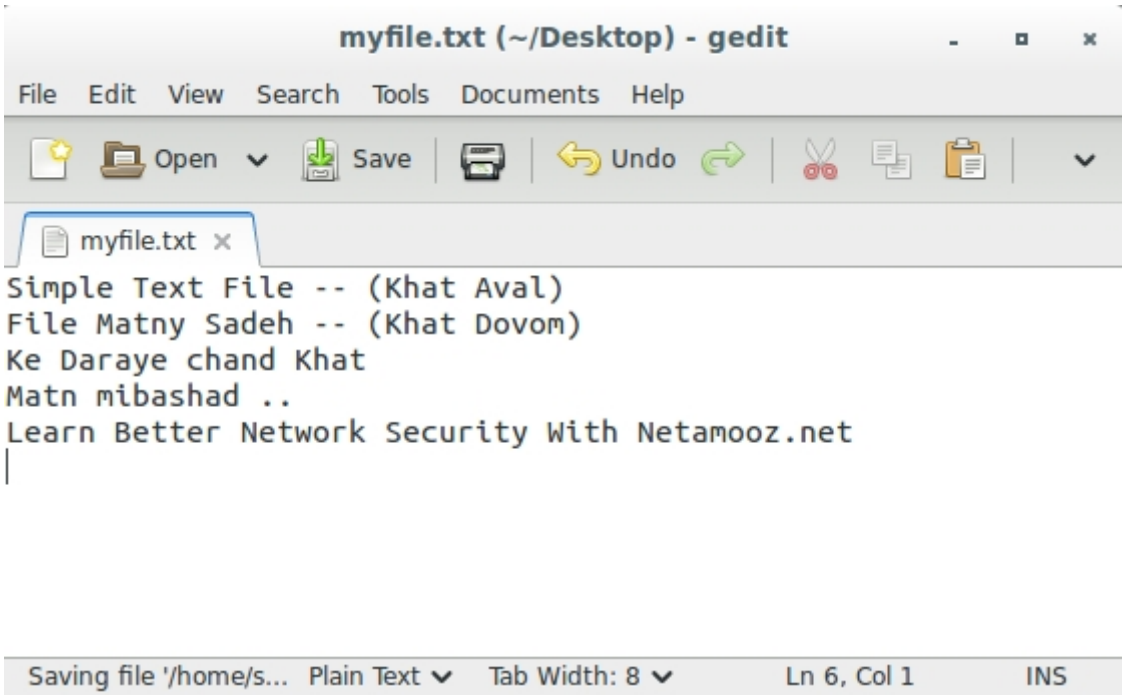
کارکردن با فایل ها

کارکردن با فایل ها

در این فصل نگاهی به کار کردن با فایل‌های خارجی خواهیم انداخت . قبلاً یاد گرفتید که چگونه با استفاده از تابع input اطلاعات را از کاربران دریافت کنیم . هرچند که در برخی شرایط وارد کردن داده‌های عظیم به برنامه کاربردی نیست بویژه وقتی که با حجم بالایی از اطلاعات کار می‌کنیم . در این موارد بهتر است که اطلاعات مورد نیاز را به عنوان فایل‌های خارجی وارد کند . در این فصل نحوه کار کردن با فایل‌ها را در پایتون آموزش می‌دهیم .

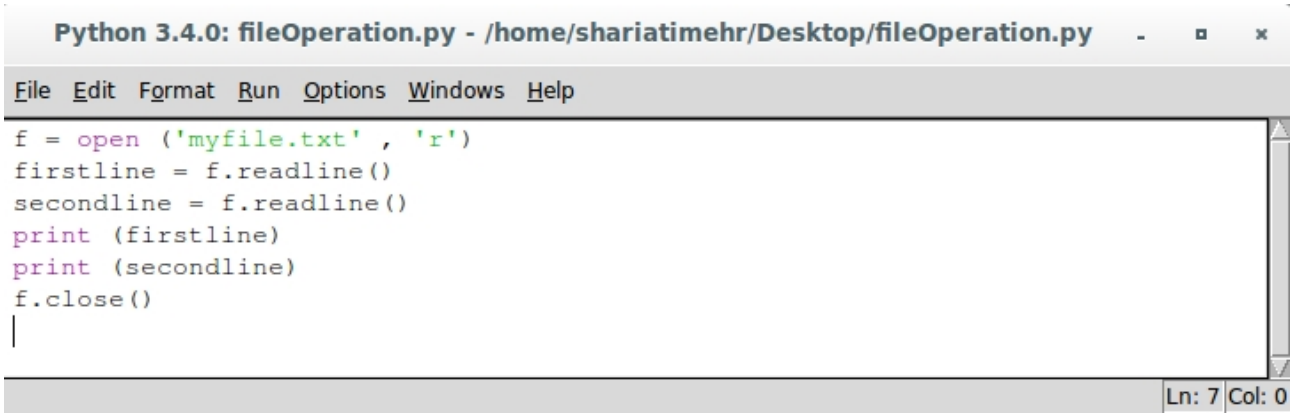
بازکردن و خواندن فایل‌های متنی

اولین نوع از فایل که می‌خواهیم با پایتون بازکنیم و بخوانیم فایل‌های متنی ساده چند خطی می‌باشد . به این منظور ابتدا یک فایل متنی چند خطی ایجاد می‌کنیم . هر متنی را به دلخواه می‌توانید درون این فایل قرار دهید . فایل را با نام myfile.txt بر روی دسکتاپ ذخیره کنید :



سپس IDLE را باز کنید و کد زیر را درون آن قرار داده و با نام fileOperation.py بر روی دسکتاپ ذخیره کنید :

```
f = open ('myfile.txt' , 'r')
firstline = f.readline()
secondline = f.readline()
print (firstline)
print (secondline)
f.close()
```



در مثال بالا خط اول فایل مورد نظر ما را باز می‌کند . قبل از اینکه بتوانیم چیزی را از فایل بخوانیم بایستی آن را باز کنیم . به این منظور از تابع open استفاده می‌کنیم . این تابع نیازمند دو پارامتر می‌باشد . پارامتر اول مسیر قرارگیری فایل را تعیین می‌کند . ما چونکه فایل ورودی myfile.txt را در کنار فایل fileOperation.py ذخیره کرده‌ایم نیاز به مسیردهی دقیق نیست در غیر این صورت بایستی مسیر حقیقی فایل را به آن بدهیم .

مثلاً در ویندوز C:\\myfile.txt

یا در لینوکس /home/shariatimehr/Desktop/myfile.txt

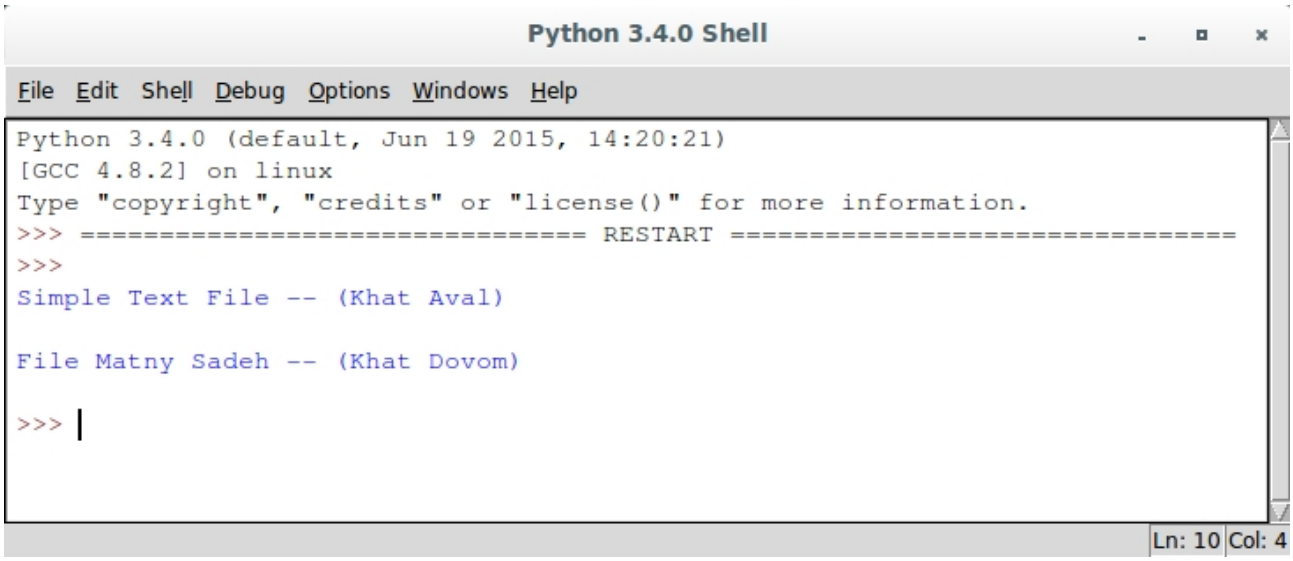
پارامتر دوم مود mode می‌باشد . این پارامتر تعیین می‌کند که فایل چگونه استفاده خواهد شد . مودهای رایج برای استفاده به شرح زیر می‌باشند .

مود 'r' تنها برای خواندن .

مود 'w' تنها برای نوشتن . در این مود اگر فایل وجود نداشته باشد ایجاد خواهد شد و اگر فایل موجود باشد همه اطلاعات موجود حذف شده و اطلاعات جدید جایگزین می‌شود .

مود 'a' برای اضافه کردن اطلاعات . اگر فایل وجود نداشته باشد ایجاد خواهد شد و اگر فایل موجود باشد اطلاعات جدید به انتهای اطلاعات قبلی اضافه خواهد شد .
مود 'r+' برای خواندن و نوشتن .

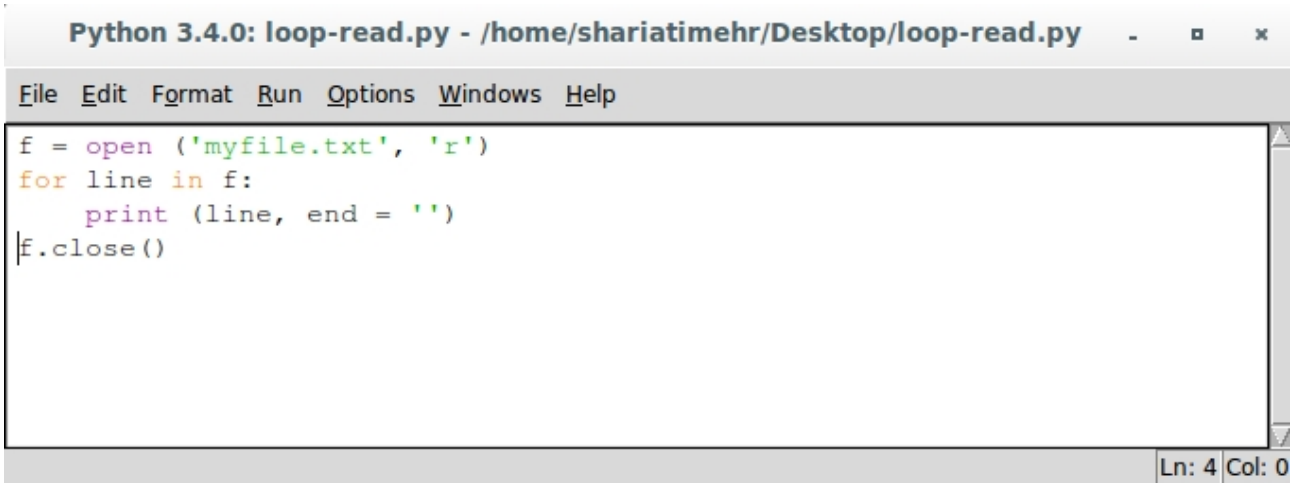
خوب مودها را شناختید ما در این مثال از مود r برای خواندن اطلاعات استفاده می‌کنیم . در خط بعدی پس از بازکردن فایل خط اول را خوانده و آن را درون متغیر firstline ذخیره می‌کند .
هر زمان تابع readline فراخوانی می‌شود یک خط جدید را از فایل که درون متغیر r ذخیره شده می‌خواند .پس از ذخیره درون دو متغیر firstline و secondline آن‌ها را چاپ می‌کنیم .
پس از اجرای فایل مشاهده می‌کنید که به صورت خودکار یک جدید ایجاد می‌شود . به این دلیل که تابع readline یک کاراکتر \n پس از هر خط اضافه می‌کند .
در آخر با استفاده از تابع close فایل را می‌بندیم . شما همیشه بایستی پس از پایان خواندن فایل را ببندید تا منابع سیستم را آزاد کنید .



استفاده از حلقه برای خواندن فایل‌های متنی

علاوه بر استفاده از تابع readline برای خواندن از یک فایل متنی ما می‌توانیم از یک حلقه for نیز استفاده کنیم . در حقیقت حلقه شیوه بهتری برای خواندن فایل‌های متنی می‌باشد . برنامه زیر نحوه استفاده از حلقه برای خواند فایل‌های متنی را نشان می‌دهد :

```
f = open ('myfile.txt', 'r')
for line in f:
    print (line, end = '')
f.close()
```



در مثال بالا ابتدا مثل قبل فایل را باز می‌کنیم ولی این بار برای خواندن از حلقه for استفاده می‌کنیم . در اینجا حلقه خط به خط اطلاعات را درون متغیر f ذخیره می‌کند و سپس با استفاده از تابع print چاپ می‌کند .

Python 3.4.0 Shell

File Edit Shell Debug Options Windows Help

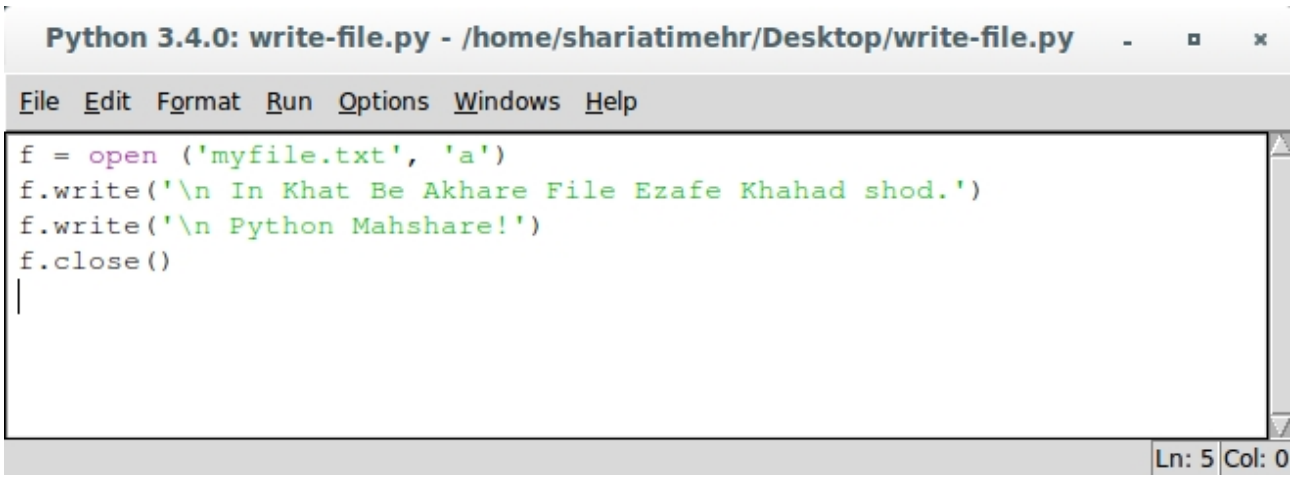
```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Simple Text File -- (Khat Aval)
File Matny Sadeh -- (Khat Dovom)
Ke Daraye chand Khat
Matn mibashad ..
Learn Better Network Security With Netamooz.net

>>>
```

Ln: 12 Col: 4

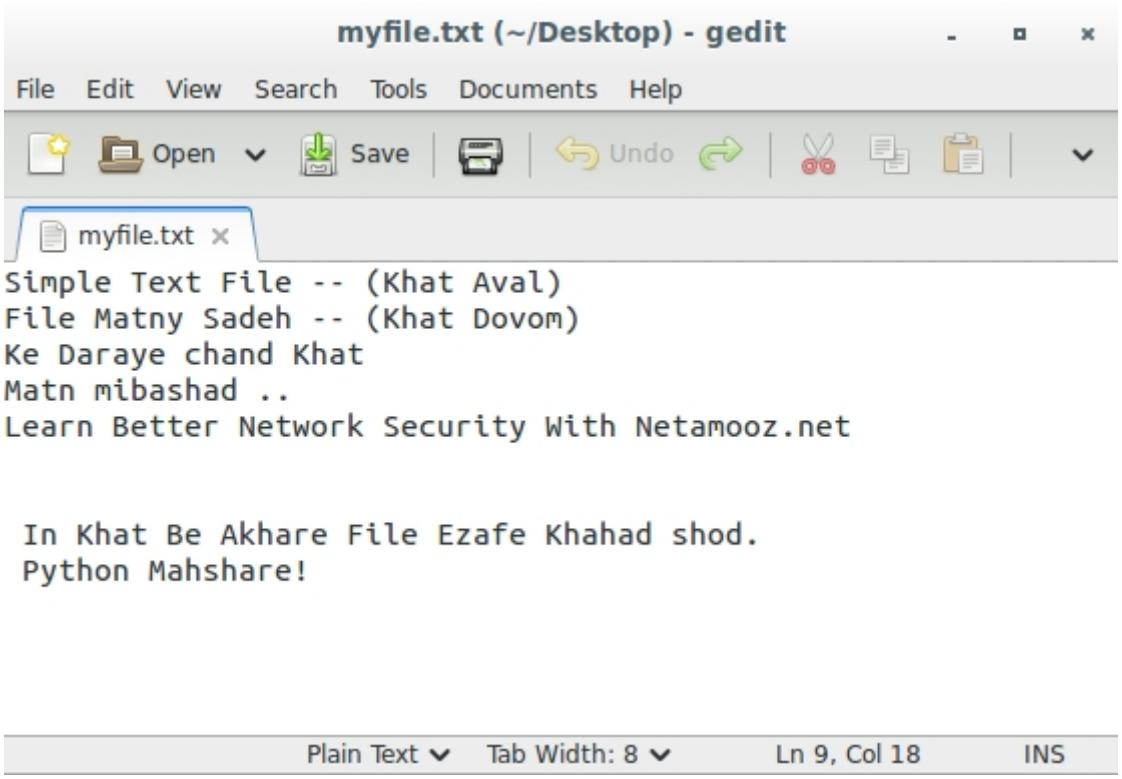
نوشتن در یک فایل متنی

اکنون که نحوه بازکردن یک فایل متنی و خواندن از آن را آموختیم ، زمان نوشتن در فایل متنی فرارسیده است . به این منظور از مود 'a' استفاده می‌کنیم تا به محتوای قبلی اضافه کنیم . می‌توانید از مود 'w' نیز استفاده کنید ولی این مود محتوای قبلی شما را پاک خواهد کرد . برنامه اجرایی به صورت زیر می‌باشد :

```
f = open ('myfile.txt', 'a')
f.write('\n In Khat Be Akhare File Ezafe
Khahad shod.')
f.write('\n Python Mahshare!')
f.close()
```



مثل قبل ابتدا با تابع open فایل مورد نظر را باز می‌کنیم . سپس با استفاده از تابع write دو خط به پایان فایل خود اضافه می‌کنیم . در آخر هم با تابع close فایل خود را می‌بندیم . برای اضافه شدن هر خط در خطی جدید از کاراکتر \n استفاده می‌کنیم .


myfile.txt (~/Desktop) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
myfile.txt
Simple Text File -- (Khat Aval)
File Matny Sadeh -- (Khat Dovom)
Ke Daraye chand Khat
Matn mibashad ..
Learn Better Network Security With Netamooz.net
In Khat Be Akhare File Ezafe Khahad shod.
Python Mahshare!
Plain Text
Tab Width: 8
Ln 9, Col 18
INS

بازکردن و خواندن فایل‌های متنی با buffer size

برخی اوقات ممکن است بخواهیم یک فایل را بوسیله buffer size بخوانیم تا برنامه ما منابع زیادی از حافظه را آشغال نکند . به این منظور به جای استفاده از تابع readline از تابع read استفاده می‌کنیم چرا که این تابع به ما اجازه می‌دهد تا اندازه بافر را تعیین کنیم . برنامه زیر را امتحان کنید :

```
inputFile = open ('myfile.txt' , 'r')
outputFile = open ('myoutputfile.txt' , 'w')
msg = inputFile.read(10)
while len(msg):
    outputFile.write(msg)
    msg = inputFile.read(10)
inputFile.close()
outputFile.close()
```

در اینجا ابتدا دو فایل inputFile.txt و outputFile.txt را برای خواندن و نوشتن باز می‌کنیم . سپس از عبارت msg = inputFile.read(10) و یک حلقه while به منظور ۱۰ بایت ۱۰ بایت فایل را می‌خواند . مقدار ۱۰ که درون پرانتز قرار دارد به تابع read می‌گوید که تنها ۱۰ بایت را در هر مرتبه حلقه بخواند . شرط موجود در حلقه while یعنی while len(msg) طول متغیر msg را بررسی می‌کند . تا زمانی که طول آن صفر نباشد حلقه اجرا خواهد شد .

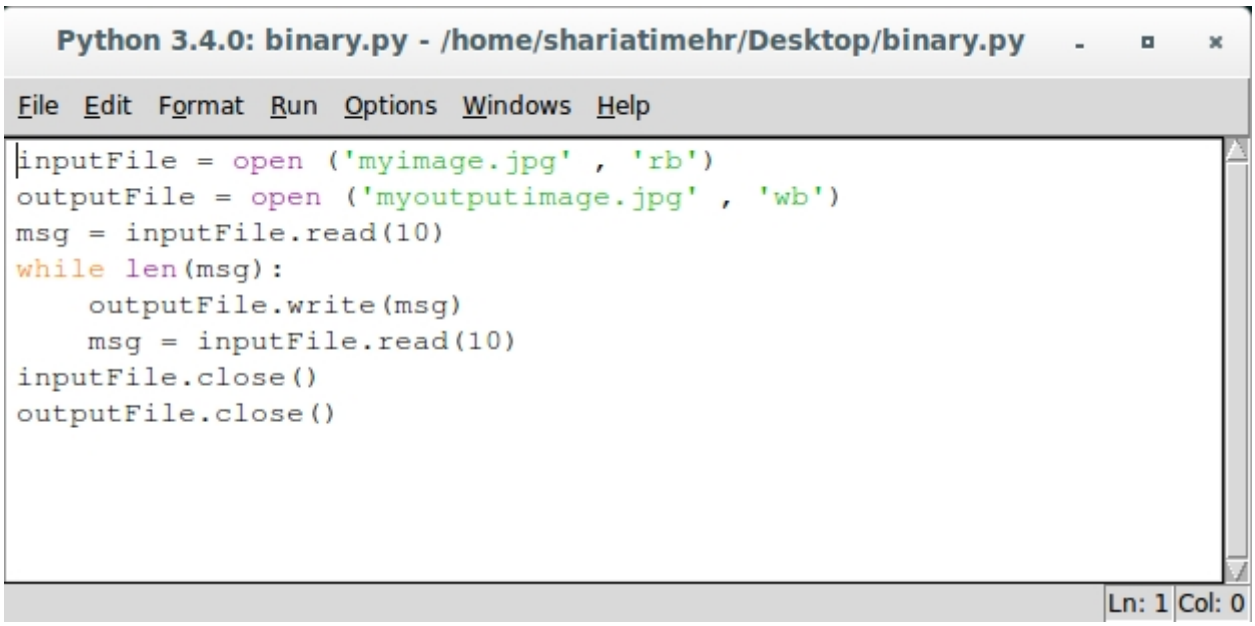
درون حلقه while عبارت outputFile.write(msg) پیام را درون فایل خروجی می‌نویسد . پس از نوشتن پیام ، عبارت msg = inputFile.read(10) ده بایت بعدی را می‌خواند و این کار را ادامه می‌دهد تا کل فایل خوانده شود . سپس برنامه هر دو فایل را می‌بندد .

پس از اجرای برنامه یک فایل جدید با نام myoutputfile.txt ایجاد خواهد شد . پس از باز کردن فایل متوجه خواهید شد که همان محتوایی که درون فایل myfile.txt وجود دارد ایجاد خواهد شد .

بازکردن و خواندن و نوشتن فایل‌های باینری

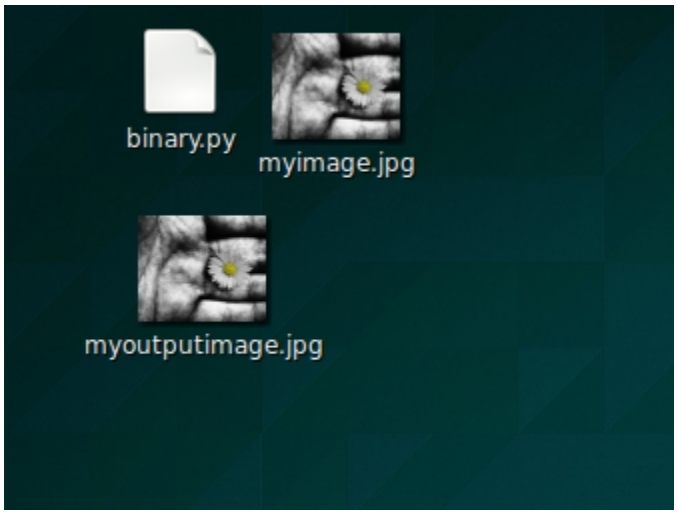
فایل‌های باینری یعنی هر فایل غیرمتنی مثل تصاویر ، ویدیو و .. برای کارکردن با فایل‌های باینری از مودهای 'rb' یا 'wb' استفاده می‌کنیم . یک فایل تصویری بر روی دسکتاپ خود کپی کنید و آن را myimage.jpg بنامید . سپس برنامه قبلی را به صورت زیر تغییر دهید :

```
inputFile = open ('myimage.jpg' , 'rb')
outputFile = open ('myoutputimage.jpg' ,
'wb')
msg = inputFile.read(10)
while len(msg):
    outputFile.write(msg)
    msg = inputFile.read(10)
inputFile.close()
outputFile.close()
```



برنامه را اجرا کنید . همانطور که ملاحظه می‌کنید یک فایل تصویر اضافی با نام

myoutputimage.jpg بر روی دسکتاپ ایجاد می‌شود . این فایل دقیقاً یک کپی از فایل تصویر قبلی می‌باشد .



حذف و تغییر نام فایل ها

دو تابع مفید دیگر که در حین کارکردن با فایل ها بایستی یاد بگیریم توابع remove و rename هستند . این توابع درون ماژول os موجود هستند و قبل از استفاده بایستی وارد شوند .

تابع remove یک فایل را حذف می‌کند . ساختار دستوری آن به صورت زیر می‌باشد :

```
remove (filename)
```

برای نمونه برای حذف فایلی با نام myfile.txt به صورت زیر عمل می‌کنیم :

```
remove('myfile.txt')
```

تابع rename نام یک فایل را تغییر می‌دهد . ساختار دستوری برای استفاده از تابع rename به صورت زیر می‌باشد :

```
rename (نام جدید فایل ، نام قدیمی فایل)
```

برای مثال برای تغییر نام فایلی با نام oldfile.txt به فایلی با نام newfile.txt به صورت زیر عمل می‌کنیم :

```
rename ('oldfile.txt' , 'newfile.txt')
```