حلقه for

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

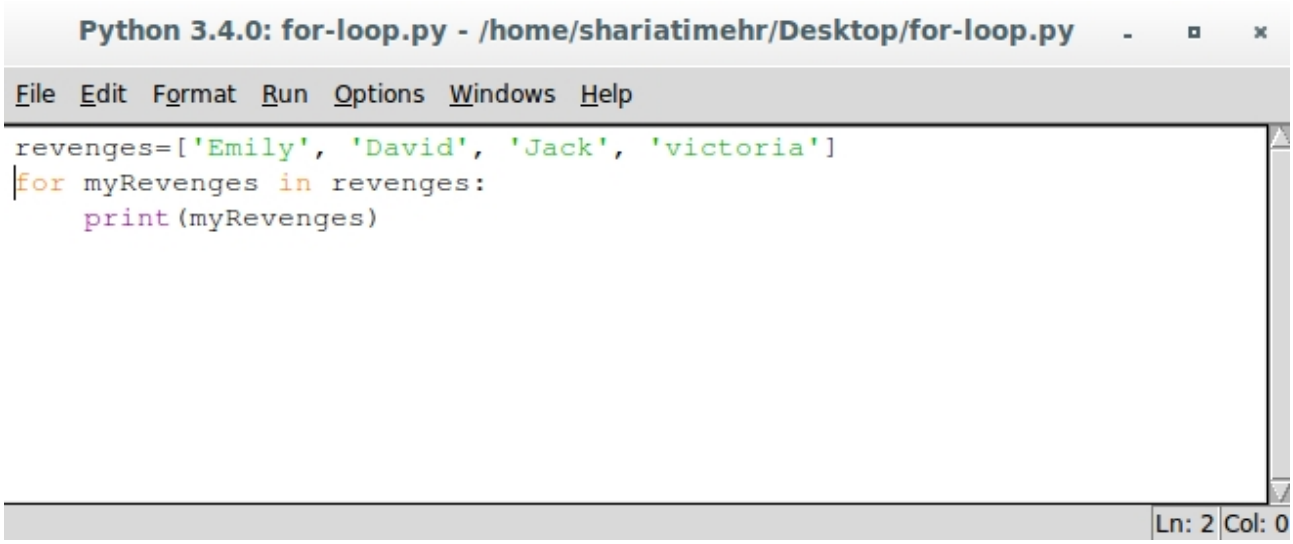
ایجاد حلقه از طریق عنصر یک تکرارشدنی

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 :

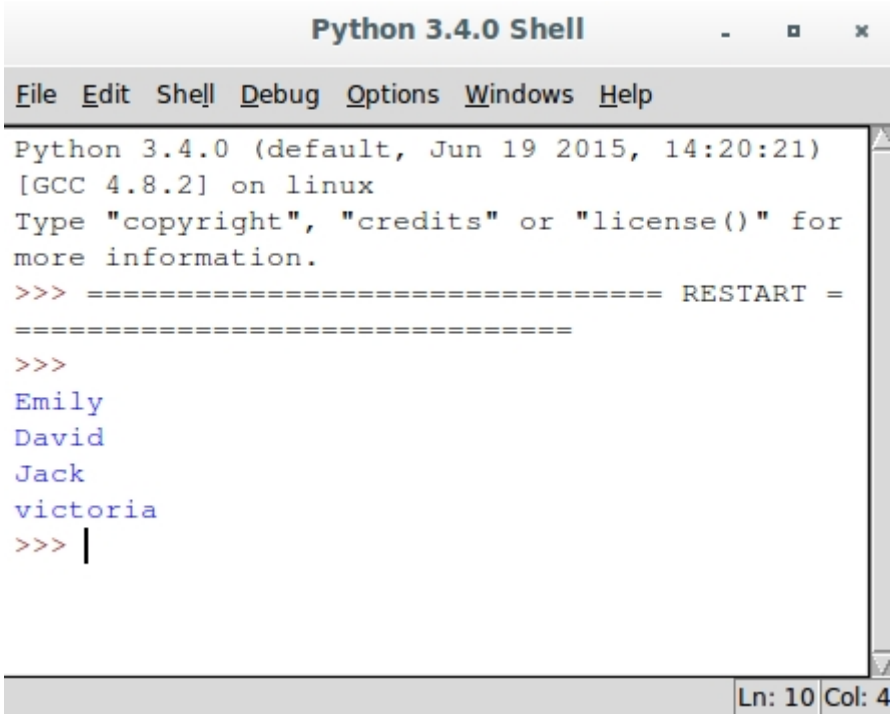
```
for a in iterable:
print (a)
```

برای مثال

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

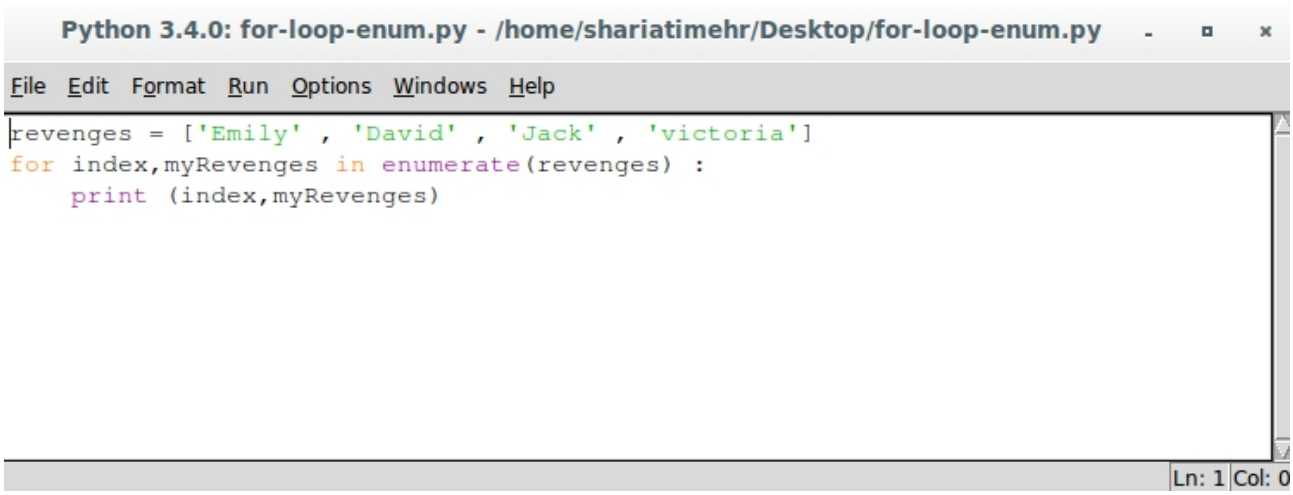


در مثال بالا ابتدا لیستی با نام revenges را تعریف کردیم و چهار عضو را درون آن قرار دادیم . سپس در خط بعد با استفاده از حلقه for هر کدام از اعضا را به ترتیب از لیست revenges بیرون کشیده و درون متغیر myRevenges قرار می‌دهیم . مثلاً اولین عضو یعنی emily را از لیست revenges بیرون کشیده و درون متغیر myRevenges 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 .

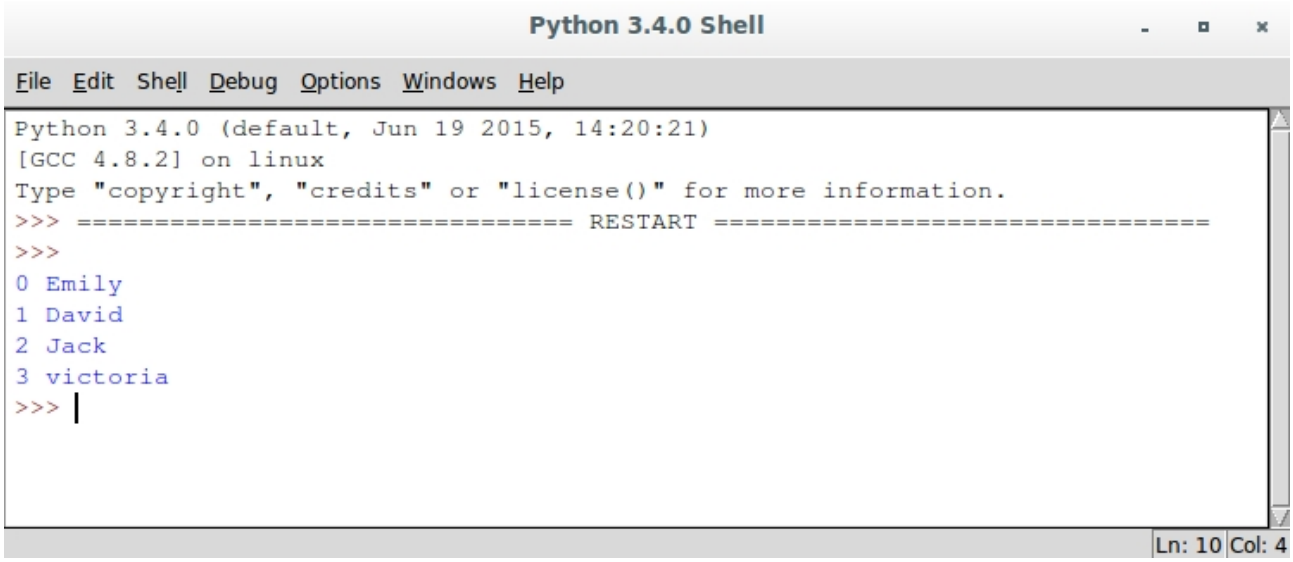


به منظور نمایش ایندکس اعضا می‌توانیم از تابع enumerate استفاده کنیم :

```
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 می‌کند 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 می‌کند

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

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

لیستی از اعداد را به صورت زیر ایجاد می‌کند

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

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

لیستی از اعداد به صورت زیر ایجاد می‌کند

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

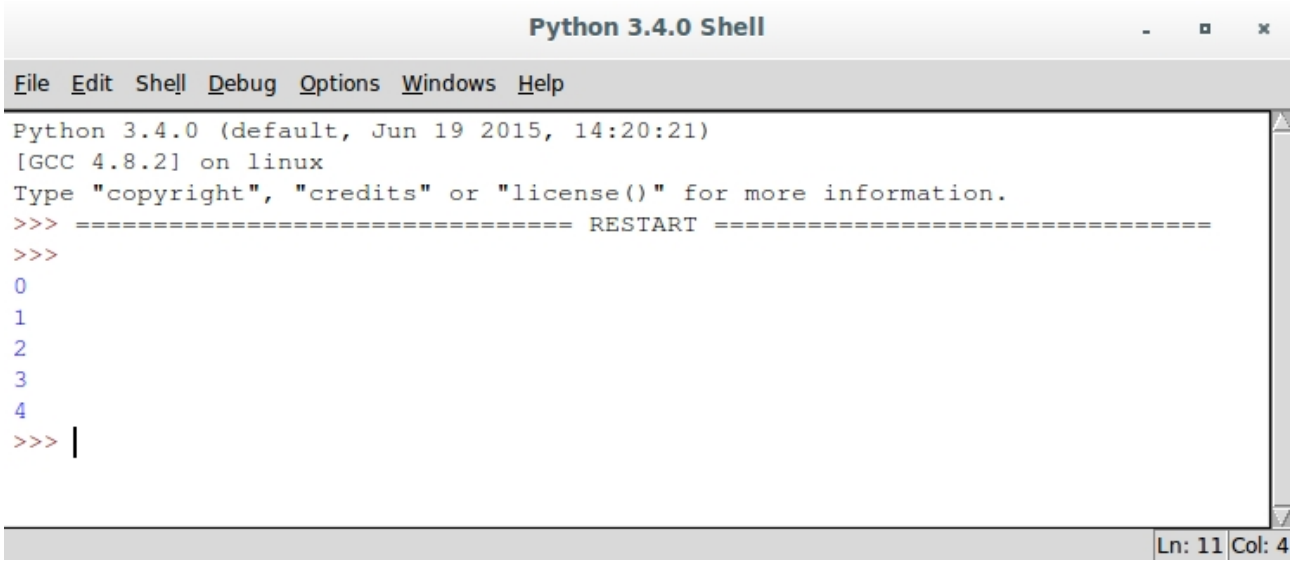
برای مشاهده نحوه عمل‌کرد تابع range در یک عبارت for مثال زیر را اجرا کنید :

```
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

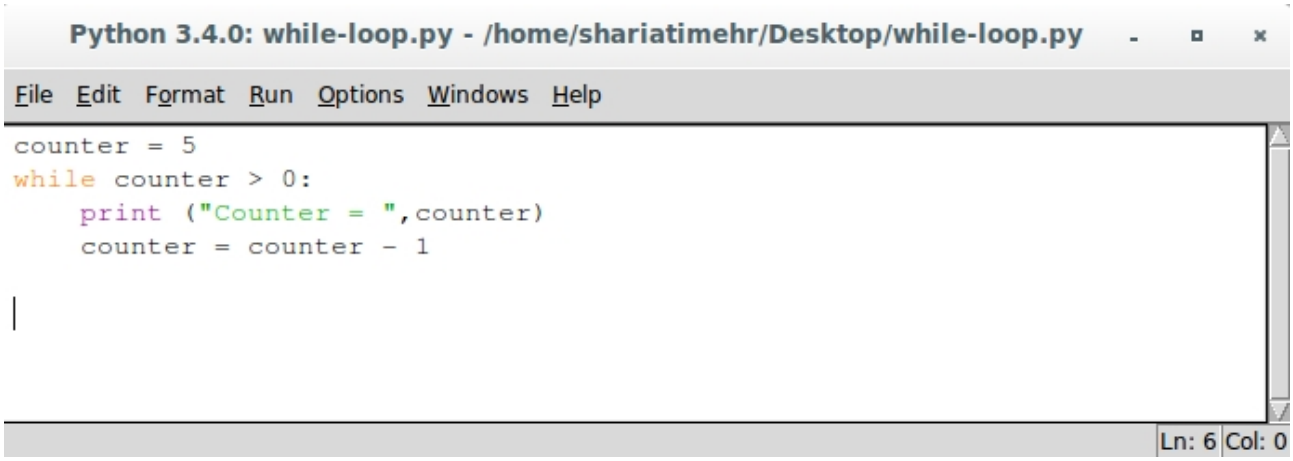
!!!!!!! تا زمانیکه !!!!!! برخی شرایط معتبر باقی بماند . ساختار دستوری حلقه while به صورت زیر می‌باشد :

while شرط برقرار باشد :

این کار را انجام بده

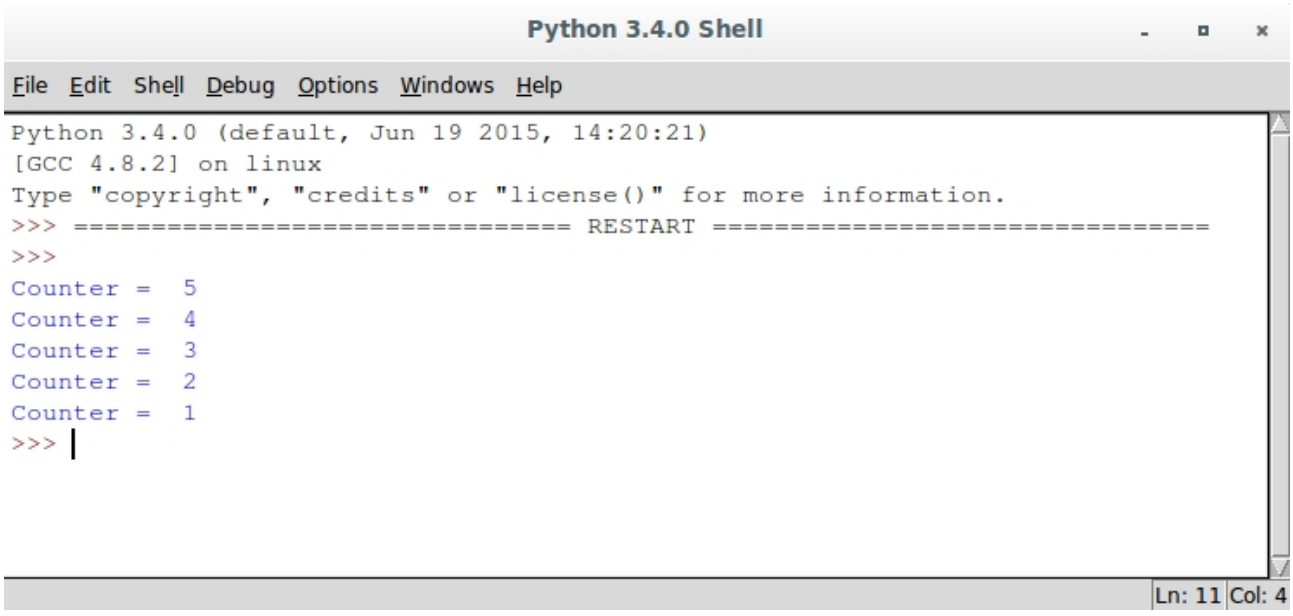
بیشتر اوقات وقتی که از یک حلقه while استفاده می‌کنیم نیاز به یک کانتر یا شمارنده برای عمل‌کرد درست حلقه خواهیم داشت . این شمارنده را counter می‌نامیم . شرط حلقه مقدار counter را ارزیابی کرده تا تشخیص دهد که آیا از مقداری بخصوص بزرگ‌تر یا کوچکتر است . اگر اینگونه بود حلقه اجرا می‌شود در غیر اینصورت حلقه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

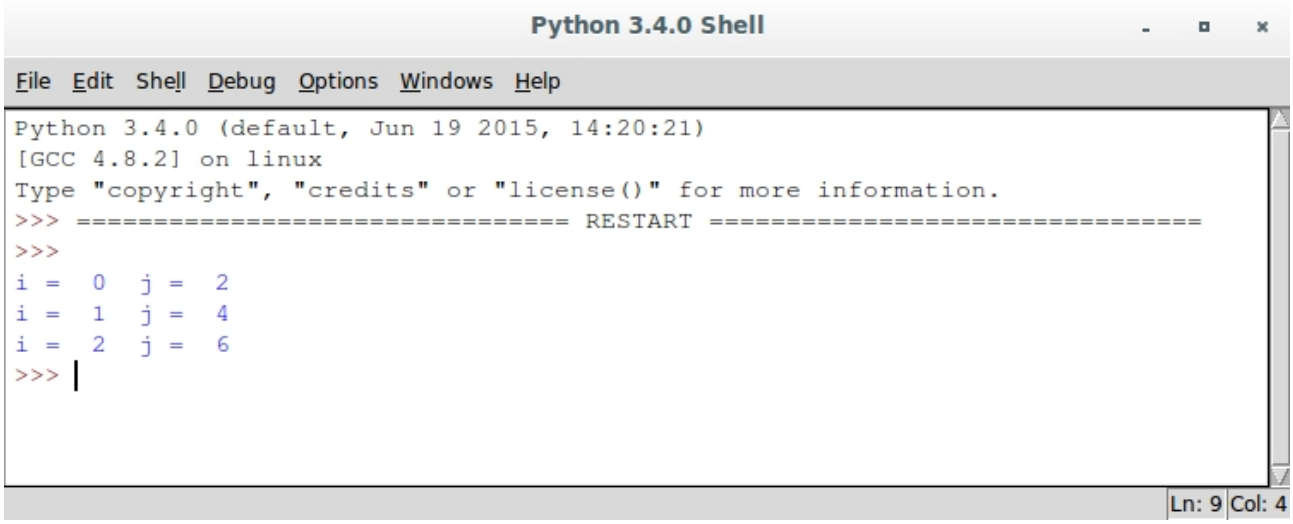
زمانی که با حلقه ها کار می‌کنیم ، گاهی اوقات نیاز است تا در صورت بوجود آمدن یک شرط خاص از کل حلقه خارج شویم . به این منظور از کلمه کلید break استفاده می‌کنیم . برنامه زیر را اجرا کنید تا با نحوه عمل‌کرد break آشنا شوید .

```
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 ...

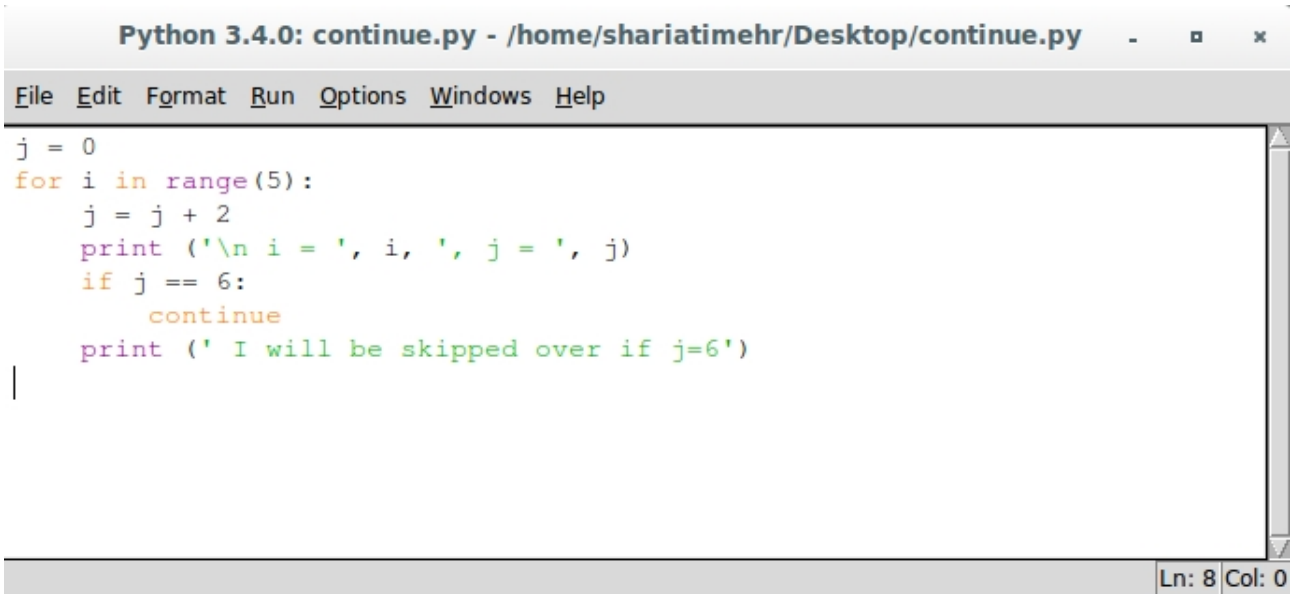
در خط بعدی با استفاده از تابع پرینت مقدار کنونی دو متغیر i و j را چاپ می‌کنیم. نکته مهم این برنامه این است که اگر مقدار متغیر جی برابر ۶ شد (if j == 6) ، با استفاده از کلمه کلید break حلقه پایان پذیرد. پس خروجی به صورت زیر حاصل می‌شود :



کلمه کلیدی continue

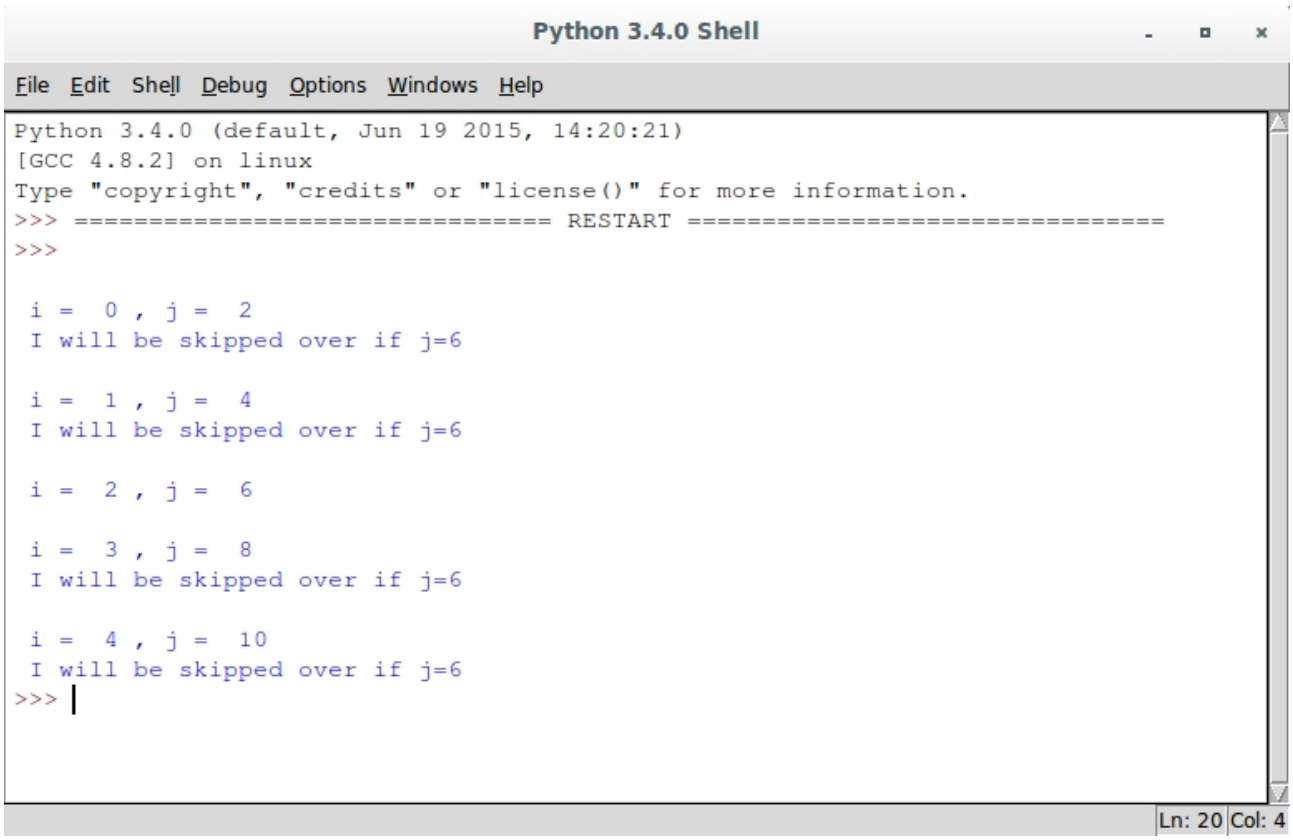
یک کلمه کلیدی دیگر برای ایجاد حلقه ها continue می‌باشد . وقتی که ما از continue استفاده می کنیم ، بقیه حلقه که پس از continue آمده فقط برای همان تکرار نادیده گرفته می‌شود . مثال زیر را بررسی و تحلیل کنید :

```
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 :

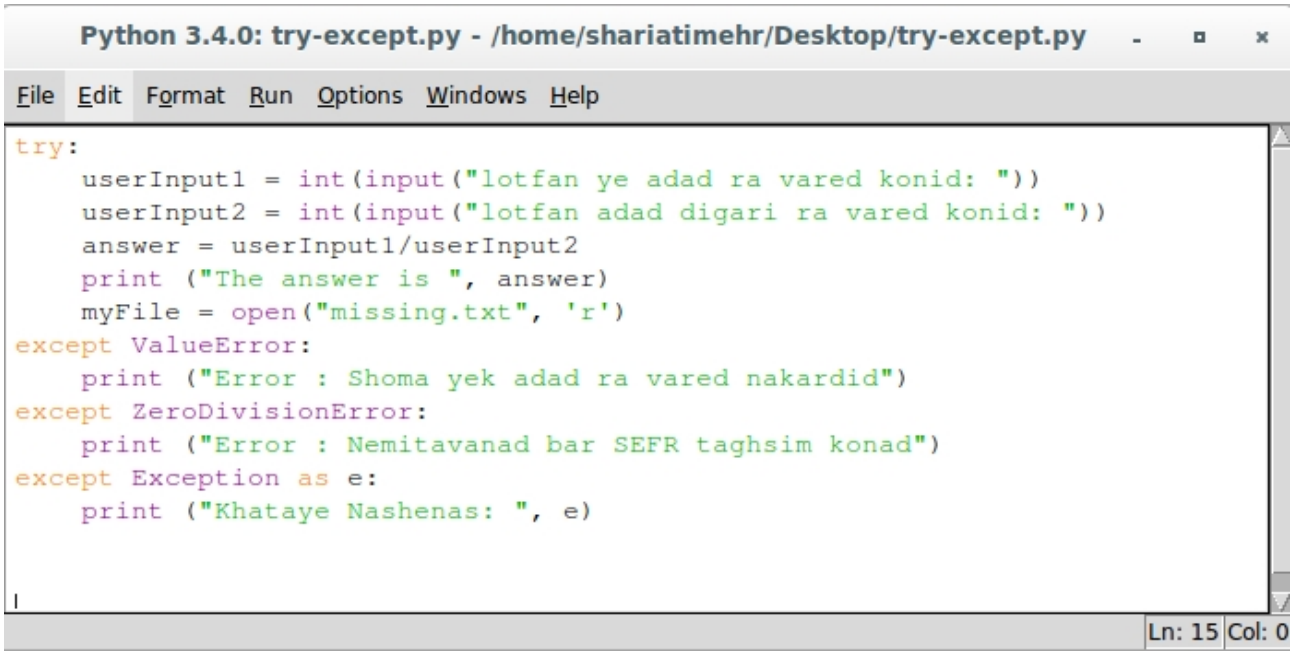
```
try:
یک کاری را انجام بده
except:
در مواجهه با خطا کار دیگری را انجام بده
```

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.