

حلقه های for

یک حلقه for به عنوان یک تکرار کننده در پایتون عمل می کند؛ این به مواردی که در یک دنباله یا هر مورد قابل تکرار دیگری هستند، می رود. اشیاء ای که ما درباره آنها یاد گرفته ایم که می توانیم روی آنها تکرار کنیم شامل رشته ها، لیست ها، تاپل ها و حتی تکرار کننده های داخلی برای دیکشنری ها مانند کلیدها یا مقادیر است.

ما قبلاً دستور for را در سخنرانی های گذشته کمی دیده ایم اما حالا بیایید درک خود را رسمی کنیم.

در اینجا فرمت عمومی برای یک حلقه for در پایتون است:

```
for item in object:  
    statements to do stuff
```

اسم متغیر استفاده شده برای مورد کاملاً به برنامه نویس بستگی دارد، بنابراین از قضاوت خود برای انتخاب نامی استفاده کنید که منطقی باشد و شما قادر به درک آن باشید وقتی به کد خود باز می گردید. این نام مورد سپس می تواند در داخل حلقه شما ارجاع داده شود، به عنوان مثال اگر می خواستید از دستورات if برای انجام بررسی ها استفاده کنید.

بزارید پیش بروید و چندین نمونه از حلقه های for را با استفاده از انواع مختلف داده های شیء بسط دهید. ما ساده شروع خواهیم کرد و بعداً پیچیدگی را افزایش خواهیم داد.

مثال 1

تکرار در لیست

In [3]:

```
1 # We'll learn how to automate this sort of list in the next lecture  
2 list1 = [1,2,3,4,5,6,7,8,9,10]
```

In [7]:

```
1 for num in list1:  
2     print(num)
```

1
2
3
4
5
6
7
8
9
10

تکرارشونده ها یا iterators

In [8]:

```
1 x = iter([1,2,3,4,5])
```

In [9]:

```
1 x
```

Out[9]:

```
<list_iterator at 0x22c6bf7ae30>
```

In [10]:

```
1 type(x)
```

Out[10]:

```
list_iterator
```

In [11]:

```
1 next(x)
```

Out[11]:

```
1
```

In [12]:

```
1 next(x)
```

Out[12]:

```
2
```

In [13]:

```
1 next(x)
```

Out[13]:

```
3
```

In [14]:

```
1 y = iter('hi')
```

In [15]:

```
1 next(y)
```

Out[15]:

```
'h'
```

In [16]:

```
1 next(y)
```

Out[16]:

'i'

In [17]:

```
1 next(y)
```

-
StopIteration

Traceback (most recent call las

t)

Cell In[17], line 1

----> 1 next(y)

StopIteration:

In [18]:

```
1 z = iter({'k1': 100, 'k2': 200})
```

In [19]:

```
1 next(z)
```

Out[19]:

'k1'

In [20]:

```
1 next(z)
```

Out[20]:

'k2'

برگردیم به حلقه های for !

مثال 2

بیاايد فقط اعداد زوج از آن لیست را چاپ کنیم!

In [21]:

```
1 list1
```

Out[21]:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [22]:

```
1 for num in list1:
2     if num % 2 == 0:
3         print(num)
```

2
4
6
8
10

می توانیم برای اعداد فرد هم از عبارت `else` استفاده کنیم:

In [23]:

```
1 for num in list1:
2     if num % 2 == 0:
3         print(num)
4     else:
5         print('Odd Number')
```

Odd Number
2
Odd Number
4
Odd Number
6
Odd Number
8
Odd Number
10

مثال 3

یک ایده دیگر متداول در حلقه `for` نگهداری یک مقدار در حین تکرارهای یک حلقه است. به عنوان مثال، بیایید یک حلقه `for` ایجاد کنیم که مجموع لیست را بدست آورد:

In [26]:

```
1 # Start sum at zero
2 list_sum = 0
3
4 for num in list1:
5     list_sum = list_sum + num
6
7 print(list_sum)
```

55

عالی! بالای سلول را بخوانید و مطمئن شوید که کاملاً متوجه هستید که چه اتفاقی می افتد. همچنین ما می توانستیم از عملگر `+=` را برای انجام جمع به سمت مجموع پیاده سازی کنیم. به عنوان مثال:

In [27]:

```
1 # Start sum at zero
2 list_sum = 0
3
4 for num in list1:
5     list_sum += num
6
7 print(list_sum)
```

55

مثال 4

ما از حلقه for با لیست‌ها استفاده کردیم، چطور با رشته‌ها؟ به یاد داشته باشید که رشته‌ها یک دنباله هستند، بنابراین وقتی از آن‌ها تکرار می‌کنیم، به هر مورد در آن رشته دسترسی خواهیم داشت.

In [28]:

```
1 for letter in 'This is a string':
2     print(letter)
```

T
h
i
s

i
s

a

s
t
r
i
n
g

مثال 5

حالا به این نگاه کنیم که چگونه یک حلقه for می‌تواند با یک tuple استفاده شود:

In [29]:

```
1 tup = (1,2,3,4,5)
2
3 for t in tup:
4     print(t**2)
```

1
4
9
16
25

مثال 6

تاپل‌ها یک کیفیت خاص در مورد حلقه‌های `for` دارند. اگر در حال تکرار از طریق یک دنباله که شامل تاپل‌ها باشید، هر عضو می‌تواند خود تاپل باشد، این یک مثال از *Tuple Packing* است. در طول حلقه `for` ما در حال باز کردن تاپل درون یک دنباله هستیم و می‌توانیم به موارد فردی درون آن تاپل دسترسی پیدا کنیم!

In [36]:

```
1 # Now with packing!
2 x = 1, 2, 3
```

In [31]:

```
1 x
```

Out[31]:

(1, 2, 3)

In [32]:

```
1 type(x)
```

Out[32]:

tuple

In [33]:

```
1 x = (1,)
```

In [34]:

```
1 type(x)
```

Out[34]:

tuple

In [37]:

```
1 x = 1, 2, 3
2 x1, x2, x3 = x
```

In [38]:

```
1 x1
```

Out[38]:

1

In [39]:

```
1 x3
```

Out[39]:

3

In [41]:

```
1 list2 = [(2,4), (6,8), (10,12)]
```

In [42]:

```
1 for tup in list2:
2     print(tup)
```

(2, 4)
(6, 8)
(10, 12)

In [43]:

```
1 # Now with unpacking in for Loop!
2 for t1, t2 in list2:
3     print(t1, t2)
```

2 4
6 8
10 12

عالی! با تاپل‌ها در یک دنباله می‌توانیم به موارد درون آن‌ها از طریق باز کردن دسترسی پیدا کنیم! دلیل اینکه این مهم است این است که بسیاری از اشیاء قابل تکرار خود را از طریق تاپل‌ها تحویل می‌دهند. بیا یاد شروع به کاوش در تکرار از طریق دیکشنری‌ها برای بررسی بیشتر این موضوع کنیم!

مثال 7

همانگونه که در ساخت تکرارشونده‌ها مشاهده کردیم، در زمان ساخت تکرار شونده از یک دیکشنری تنها به کلیدها دسترسی خواهیم داشت، حالا با هم مثال‌هایی از حالات مختلف برای دیکشنری‌ها حل می‌کنیم.

In [44]:

```
1 d = {'k1': 100, 'k2': 200, 'k3': 300}
```

In [45]:

```
1 for item in d:
2     print(item)
```

k1
k2
k3

In [46]:

```
1 for item in d:  
2     print(item, d[item])
```

```
k1 100  
k2 200  
k3 300
```

توجه کنید که این فقط کلیدها را تولید می‌کند. پس چگونه می‌توانیم مقادیر را بدست آوریم؟ یا هر دو کلید و مقدار؟

ما در مورد متدهای: `.keys()` و `.values()` و `.items()` قبلا یاد گرفتیم و در این بخش در حلقه های `for` از این متدها استفاده خواهیم کرد.

در پایتون هر یک از این روش‌ها یک *dictionary view object* برمی‌گرداند. این عملیات مانند آزمایش عضویت و تکرار را پشتیبانی می‌کند، اما محتوای آن مستقل از دیکشنری اصلی نیست - فقط یک نمایش است. ببینید آن را در عمل:

In [47]:

```
1 # Create a dictionary view object  
2 d.items()
```

Out[47]:

```
dict_items([('k1', 100), ('k2', 200), ('k3', 300)])
```

از آنجا که روش `.items()` تکرار را پشتیبانی می‌کند، ما می‌توانیم بسته‌بندی دیکشنری را انجام دهیم تا کلیدها و مقادیر را جدا کنیم، همانطور که در مثال‌های قبلی انجام دادیم.

In [48]:

```
1 # Dictionary unpacking  
2 for k, v in d.items():  
3     print(k , v)
```

```
k1 100  
k2 200  
k3 300
```

اگر می‌خواهید یک لیست واقعی از کلیدها، مقادیر یا تاپل‌های کلید/مقدار بدست آورید، می‌توانید نمایش را به عنوان یک لیست *cast* کنید:

In [49]:

```
1 list(d.keys())
```

Out[49]:

```
['k1', 'k2', 'k3']
```

به یاد داشته باشید که دیکشنری‌ها بدون ترتیب هستند و کلیدها و مقادیر به ترتیب دلخواه برمی‌گردند. می‌توانید یک لیست مرتب شده با استفاده از `sorted()` بدست آورید:

In [50]:

```
1 sorted(d.values())
```

Out[50]:

```
[100, 200, 300]
```

عبارت else در حلقه for

همانگونه که در حلقه while مشاهده نمودیم پایتون به امکان میدهد تا در حلقه ها از عبارت else پس از حلقه استفاده نماییم. همچنین دقت داشته باشید که دستورات این عبارت تنها زمانی اجرا خواهند شد که حلقه بصورت طبیعی به اتمام برسد!

In [51]:

```
1 tup = (1,2,3,4,5)
2
3 for t in tup:
4     print(t)
5 else:
6     print('the for loop ended')
7
8 print('outside')
```

```
1
2
3
4
5
the for loop ended
outside
```

In [52]:

```
1 tup = (1,2,3,4,5)
2
3 for t in tup:
4     print(t)
5     if(t == 3):
6         break
7 else:
8     print('the for loop ended')
9
10 print('outside')
```

```
1
2
3
outside
```

تابع range

در اکثر موارد در حلقه for امکان تعریف یک دنباله بصورت دستی فراهم نیست. برای مثال اگر نیاز باشد تا اعداد زوج در بازه 1 تا 10000000 را بدست آوریم ، در این حالت استفاده از یک لیست یا دنباله که بصورت دستی نوشته شده است به دو دلیل تقریباً غیر ممکن است:

1. نوشتن دستی اعداد این بازه واقعا خسته کننده و عملا غیر ممکن است.
2. حتی اگر بصورت دستی هم چنین کاری انجام دهید، این تعداد داده به فضای بسیار زیادی در ram رایانه نیاز خواهند داشت.

از این رو از تابع `range` استفاده می نماییم. این تابع بصورت خودکار یک تکرار شونده ایجاد می کند و در هر مرحله تنها یک عضو را برای ما تولید می کند و همچنین در تکرار بعدی عضو بعدی را تولید خواهد کرد. در نتیجه در میزان مصرف رم بشدت صرفه جویی خواهد نمود.

```
range(start, stop, step)
```

In [53]:

```
1 range(10)
```

Out[53]:

```
range(0, 10)
```

In [54]:

```
1 # cast to List
2 list(range(10))
```

Out[54]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [58]:

```
1 for i in range(1, 100):
2     if i % 20 == 0:
3         print(i)
```

```
20
40
60
80
```

نتیجه گیری

ما یاد گرفتیم که چگونه از حلقه های `for` برای تکرار از طریق تاپل ها، لیست ها، رشته ها و دیکشنری ها استفاده کنیم. این یک ابزار مهم برای ما خواهد بود، بنابراین مطمئن شوید که آن را خوب می دانید و مثال های فوق را درک کرده اید.