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 شئ بسط دهید. ما ساده شروع خواهیم کرد و بعداً پیچیدگی را افزایش خواهیم داد.

مثال 1

تکرار در لیست

In [1]:

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

In [2]:

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

```
1
2
3
4
5
6
7
8
9
10
```

تکرارشونده ها یا itrators

In [3]:

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

In [4]:

```
1 x
```

Out[4]:

```
<list_iterator at 0x1bcd3537850>
```

In [5]:

```
1 next(x)
```

Out[5]:

```
1
```

In [6]:

```
1 next(x)
```

Out[6]:

```
2
```

In [7]:

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

Out[7]:

```
<str_iterator at 0x1bcd3623e20>
```

In [8]:

```
1 next(y)
```

Out[8]:

```
'h'
```

In [9]:

```
1 next(y)
```

Out[9]:

```
'i'
```

In [10]:

```
1 next(y)
```

```
---------------------------------------------------------------------------
-
StopIteration                             Traceback (most recent call las
t)
Cell In[10], line 1
----> 1 next(y)

StopIteration:
```

In [11]:

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

In [12]:

```
1 next(z)
```

Out[12]:

```
'k1'
```

In [13]:

```
1 next(z)
```

Out[13]:

```
'k2'
```

برگردیم به حلقه های for !

مثال 2

بیایید فقط اعداد زوج از آن لیست را چاپ کنیم!

In [14]:

```
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 [15]:

```
for num in list1:
    if num % 2 == 0:
        print(num)
    else:
        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 [16]:

```
# Start sum at zero
list_sum = 0

for num in list1:
    list_sum = list_sum + num

print(list_sum)
```

```
55
```

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

In [17]:

```
# Start sum at zero
list_sum = 0

for num in list1:
    list_sum += num

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

In [18]:

```
for letter in 'This is a string.':
    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 [19]:

```
tup = (1,2,3,4,5)

for t in tup:
    print(t)
```

```
1
2
3
4
5
```

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 [20]:

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

In [21]:

```
1 x
```

Out[21]:

```
(1, 2, 3)
```

In [22]:

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

In [23]:

```
1 x
```

Out[23]:

```
1
```

In [24]:

```
1 type(x)
```

Out[24]:

```
int
```

In [25]:

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

In [26]:

```
1 x
```

Out[26]:

```
(1,)
```

In [27]:

```
1 type(x)
```

Out[27]:

```
tuple
```

In [28]:

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

In [29]:

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

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

In [30]:

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

In [31]:

```
x1, x2, x3 = x
```

In [32]:

```
print(x1, x2, x3)
```

```
1 2 3
```

In [33]:

```
# Now with unpacking in for loop!
for t1,t2 in list2:
    print(t1)
```

```
2
6
10
```

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. بیایید شروع به کاوش در تکرار از طریق دیکشنری‌ها برای بررسی بیشتر این موضوع کنیم!

مثال 7

همانگونه که در ساخت تکرارشونده ها مشاهده کردیم، در زمان ساخت تکرار شونده از یک دیکیشنری تنها به کلیدها دسترسی خواهیم داشت، حالا با هم مثال هایی از حالات مختلف برای دیکشینری ها حل می کنیم.

In [34]:

```
d = {'k1':1,'k2':2,'k3':3}
```

In [35]:

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

```
k1
k2
k3
```

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: `values()`، `keys.` و `items()` قبلا یادگرفتیم 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 [36]:

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

Out[36]:

```
dict_items([('k1', 1), ('k2', 2), ('k3', 3)])
```

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 [37]:

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

```
k1
1
k2
2
k3
3
```

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

In [38]:

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

Out[38]:

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

به یاد داشته باشید که دیکشنری‌ها بدون ترتیب هستند و کلیدها و مقادیر به ترتیب دلخواه برمی‌گردند. می‌توانید یک

In [39]:

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

Out[39]:

```
[1, 2, 3]
```

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 [40]:

```
tup = (1,2,3,4,5)

for t in tup:
    print(t)

else:
    print('the for loop ended')

print('outside')
```

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

In [41]:

```
tup = (1,2,3,4,5)

for t in tup:
    print(t)
    if(t == 3):
        break
else:
    print('the for loop ended')

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 اوریم ، در این حالت استفاده از یک لیست یا دنباله که بصورت دستی نوشته شده است به دو دلیل تقریبا 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 [42]:

```
range(10)
```

Out[42]:

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

In [43]:

```
# cast to list
list(range(10))
```

Out[43]:

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

In [45]:

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

```
20
40
60
80
```

نتیجه‌گیری

ما یاد گرفتیم که چگونه از حلقه‌های for برای تکرار از طریق تاپل‌ها، لیست‌ها، رشته‌ها و دیکشنری‌ها استفاده کنیم. این یک ابزار مهم برای ما خواهد بود، بنابراین مطمئن شوید که آن را خوب می‌دانید و مثال‌های فوق را درک کرده‌اید.