Loops

حلقه ها

- در پایتون برای ایجاد تکرار دو دستور وجود دارد
  - **حلقه While**
    - در این حلقه تا زمانیکه شرط حلقه صحیح باشد، عبارات داخل حلقه تکرار می شوند.
  - **حلقه For**
    - این حلقه عموما تا زمانیکه عنصری در لیست، تاپل و ... وجود داشته باشد، تکرار می شود.

حلقه while

- در حلقه while تا زمانیکه شرط حلقه صحیح باشد، دستورات داخل بدنه حلقه تکرار می شوند.
- قالب کلی دستور while

```
while (condition):
    statements
```

- در استفاده از حلقه while باید مواظب باشیم تا گرفتار **حلقه بی نهایت** نشویم (یعنی شرط حلقه در هیچ صورت false نشود.)

دستورات break و continue

- اگر بخواهیم قبل از رسیدن به پایان حلقه (false شدن شرط حلقه) به اجرای حلقه پایان بدهیم از دستور **break** استفاده می کنیم.
- اگر بخواهیم تکراری که الان در آن هستیم نادیده گرفته شود و وارد تکرار جدید شویم از دستور **continue** استفاده می کنیم.

```
i = 1
while i < 6:
  print(i)
  if  i == 3:
    break
  i += 1
```

Output: 1 2 3

```
i = 0
while i < 6:
  i += 1
  if  i == 3:
    continue
  print(i)
```

Output: 1 2 4 5 6

دستور else

- در پایتون می توانیم دستوراتی را به محض اینکه شرط حلقه false شد، اجرا کنیم. این دستورات می بایست داخل قسمت else حلقه نوشته بشوند.

```
i = 1
while i < 4:
    print(i)
    i += 1
else:
    print('done!')
```

Output:

1

2

done!

حلقه for

- حلقه for برای ایجاد تکرار در یک دنباله مورد استفاده قرار می گیرد.
- دنباله می تواند لیست (list)، دیکشنری (dictionary)، تاپل (tuple)، مجموعه (set) و یا رشته (string) باشد.
- قالب کلی دستور for

```
for item in items:
    statements
[else:
    statements]
```

- عملکرد دستور for در زبان پایتون با سایر زبان ها تفاوت دارد و تنها قابلیت اجرا روی دنباله های قابل تکرار (iterable) را دارد.
- دستورات break و continue در حلقه for هم عملکردی همانند آنچه در حلقه while داشتند، دارند.

حلقه for - مثال

- مثال

```
numbers= [2,4,6,8,9,6]
for x in numbers:
  print(x)
```

→ Output: 2 4 5 8 9 6

- مثال

```
cars = [“BMW", “Volvo", “Benz"]
for car in cars:
  print(car)
  if car == “Volvo":
     break
```

→ Output: BMW

کاربرد تابع range

- با کمک تابع range می توانیم عملکرد شمارنده را در حلقه for ایجاد کنیم.
- تابع range(a, b) دنباله ای از اعداد صحیح در بازه a تا b-1 ایجاد می کند.

```
for x in range(4, 9):
  print(x)
```

⟶

Output: 4 5 6 7 8

حلقه های تودرتو

- چند حلقه می توانند داخل یکدیگر قرار بگیرند.

```
colors= ["red", “green", “yellow"]
fruits = ["apple", “pepper"]
for x in colors:
  for y in fruits:
    print(x, y)
```

Output:

```
red apple
red pepper
green apple
green pepper
yellow apple
yellow pepper
```

دستور pass

- حلقه for نمی تواند خالی باشد، ولی اگر به دلایلی دستوری برای اینکه درون حلقه بنویسیم، نداشته باشیم از دستور pass استفاده می کنیم.

```
for x in range(1, 100):
  pass
```

Iterator

- یک iterator شی ایست که می تواند شامل تعداد قابل شمارشی از عناصر باشد.
- هر iterator شامل دو متد زیر است:
  - __iter__()
  - __next__()

```
fruits = ["apple", "banana", "cherry"]
x = iter(fruits)
print(next(x))
print(next(x))
print(next(x))
```

Output:
apple
banana
cherry

چند تابع کاربردی در استفاده از حلقه

- تابع zip() از آرگومان های ورودی یک زوج استخراج می کند.

```
question = ['first name', 'last name', 'telephone number']
answer = ['Ali', 'Ahmadi', '09356678754']
for q, a in zip(questions, answer):
    print('What is your {0}?  {1}.').format(q, a))
```

Output:

```
What is your first name? Ali
What is your last name? Ahmadi
What is your telephone number? 09356678754
```

چند تابع کاربردی در استفاده از حلقه

- تابع reversed یک دنباله را برعکس می کند.

```
for i in reversed(range(1,10,2)):
    print(i, end=‘, ‘)
```

Output: 9, 7, 5, 3, 1,

- تابع ()sorted یک لیست را مرتب شده برای نمایش آماده می کند.
- لازم بذکر است که ترتیب عناصر موجود در لیست را تغییر نمی دهد.

```
ch = [f, a, d, c, b]
for i in sorted(ch):
    print(I, end=‘, ‘)
```

Output: a, b, c, d, f,