لیست

لیست ساختار داده‌ای در پایتون است که برای نگه داشتن اشیا در تعداد به کار می‌رود.

- لزومی ندارد که اشیا داخل لیست نوع یکسانی داشته باشند.
- لیست نوع داده‌ای تغییرپذیر (*mutable*) است؛ به این معنی که پس از تعریف می‌توان آن را تغییر داد. به طور مثال رشته‌ها و تاپل‌ها اینگونه نیستند.
- عناصر لیست از شماره‌ی صفر شروع می‌شوند.

Copy Python

```
>>> a = [1 , 2 , 3 , 4 , 5 , 6.0 , "HELLO"]
>>> a
[1, 2, 3, 4, 5, 6.0, 'HELLO']
>>> a[0]
1
>>> a[0] = 12
>>> a
[12, 2, 3, 4, 5, 6.0, 'HELLO']
```

در مثال بالا تغییرپذیر بودن اعضای لیست و یکسان نبودن نوع آنها نشان داده شده است.

بریدن لیست

فرض کنید لیستی با نام `a` داریم.

- `a[i]` عنصر $i+1$ ام لیست است.(با توجه به اینکه لیست از اندیس ۰ شروع می‌شود)
- `a[-i]` به عنصر i ام از انتهای لیست اشاره می‌کند یعنی `a[-1]` به آخرین عنصر لیست اشاره می‌کند.
- `a[l:r]` به معنی اعضا $l+1$ تا r ام لیست است.
- `a[l:]` وقتی پارامتر انتها خالی باشد، از عنصر $l+1$ ام تا انتهای لیست در نظر گرفته می‌شود.
- `a[:r]` وقتی پارامتر ابتدا خالی باشد از ابتدا تا عنصر r ام در نظر گرفته می‌شود.
- `a[l:r:s]` اعضا لیست از $l+1$ تا قبل از $r+1$ با قدم‌هایی به اندازه‌ی s (s می‌تواند منفی باشد).

```
>>> a = [1 , 2 , 3 , 4 , 5 , 6.0 , "HELLO"]
>>> a[3:5]
[4, 5]
>>> a[:5]
[1, 2, 3, 4, 5]
>>> a[5:]
[6.0, 'HELLO']
>>> a[::2]
[1, 3, 5, 'HELLO']
>>> a[::-1]
['HELLO', 6.0, 5, 4, 3, 2, 1]
```

- `a[:]` لیست را کپی می‌کند و معادل متد *copy* است.

متدهای پرکاربرد لیست

```
>>> a = [1,2,3,4]
>>> a.append(10)
>>> a
[1, 2, 3, 4, 10]
>>> a.append(20)
>>> a.pop()
20
>>> a
[1, 2, 3, 4, 10]
>>> a.pop(0)
1
>>> a
[2, 3, 4, 10]
>>> a.insert(0,5)
>>> a
[5, 2, 3, 4, 10]
```

append `list.append(element)`

عنصری را به انتهای لیست اضافه می‌کند.

pop `list.pop(index)`

عنصر در `index` مشخص شده را حذف می‌کند. اگر آرگومانی به آن داده نشده باشد عنصر آخر را حذف می‌کند و معادل با کد زیر است.

```
del list[index]
```

insert `list.insert(index,element)`

عنصر داده شده را در اندیس داده شده می‌گذارد.

- با ۳ تابع بالا می‌توان پشته (stack) و صف (queue) را پیاده‌سازی کرد.

remove `list.remove(element)`

اولین کاربرد این عنصر در لیست را حذف می‌کند.

```
>>> b = [1,2,3,4,5,4,3,3]
>>> b.remove(3)
>>> b
[1, 2, 4, 5, 4, 3, 3]
```

index `list.index(element)`

اندیس اولین کاربرد این عنصر در لیست را برمی‌گرداند و اگر این عنصر در لیست وجود نداشته باشد، `ValueError` پرتاب می‌کند.

```
>>> b
[1, 2, 4, 5, 4, 3, 3]
>>> b.index(4)
2
>>> b.index(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 10 is not in list
```

extend `list.extend(iterable)`

لیست (یا هر شی قابل پیمایش دیگری) را به انتهای لیست کنونی اضافه می‌کند. جمع بین دو لیست هم به همین شکل عمل می‌کند با این تفاوت که لیست دوم را به لیست اول اضافه می‌کند و خروجی می‌دهد و هیچکدام تغییر نمی‌کنند. ولی در متد `extend` لیست تغییر می‌کند. به زبان دیگر متد `extend` *in place* است.

```
>>> l = [1,2,3]
>>> r = [4,5,6]
>>> l.extend(r)
>>> l
[1, 2, 3, 4, 5, 6]
```

کد زیر همین عمل را با استفاده است عملگر جمع انجام می‌دهد.

```
>>> l = [1,2,3]
>>> r = [4,5,6]
>>> l + r
[1, 2, 3, 4, 5, 6]
```

- کلمه‌ی `in` وجود یک `element` را در لیست بررسی می‌کند.

```
>>> a = [1,2,3,4,5,6]
>>> 3 in a
True
>>> 10 in a
False
```

- تابع `len` طول لیست را برمی‌گرداند.

```
len(list)
```

معرفی لیست (*list comprehension*)

به این وسیله می‌توانیم به جای درست کردن یک لیست خالی، و سپس پر کردن یکی یکی عنصرها در آن، یک لیست را به طور کامل مقداردهی اولیه کنیم.

```
new_list = [expression for element in old_list if condition]
```

- دقت کنید که در اینجا نمی‌توان از `else` استفاده کرد.

```
>>> l = [1,2,3,4,5,6]
>>> second_l = [2*item for item in l]
>>> second_l
[2, 4, 6, 8, 10, 12]
>>> second_l = [2*item for item in l if item%3 !=0]
>>> second_l
[2, 4, 8, 10]
```

در کد بالا مثال مقداردهی اولیه‌ی لیست `second_l` بر مبنای لیست `l` را می‌بینید.