سِت

ست، ساختار داده‌ای در پایتون معادل مفهوم مجموعه در ریاضی است. به این معنی که اشیا تکراری نمی‌توانند داخل آن وجود داشته باشند. اعضای آن ترتیب ندارند و اندیس گذاری نشده‌اند. ولی قابل پیمایش هستند یعنی می‌توان با حلقه به همه‌ی اعضای آنها دسترسی داشت. در نوشتار پایتون مثل ریاضی اعضای آن بین `{}` قرار می‌گیرند. دقت کنید که آن‌ها را با دیکشنری اشتباه نکنید.

ساختن ست

با استفاده از سازنده *set* می توان از یک ساختار داده دیگر مثل تاپل یا لیست، یک مجموعه ساخت به این صورت که ساختار داده جدید(set) همه‌ی عناصر تکراری را حذف می‌کند و از عناصر به دست آمده یک مجموعه می‌سازد. اگر این تابع بدون هیچ آرگومانی صدا شود یک ست خالی برمیگرداند. با نوشتار `{}` نیز می‌توان ست‌ها را مقداردهی اولیه کرد.

```
>>> filled_set = {"a" , "a" , "b" , "c"}
>>> filled_set
{'b', 'c', 'a'}
>>> empty_set = set()
>>> set_with_constructor = set(['a','a','b','c'])
>>> set_with_constructor
{'b', 'c', 'a'}
```

- مثل کلید دیکشنری‌ها، اعضای ست هم باید غیر قابل تغییر (*immutable*) باشند.

```
>>> invalid_set = {[1], 1}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> valid_set = {(1,), 1}
```

  - چون لیست، تغییرپذیر است، نمی‌تواند عضو ست باشد.

متدهای کاربردی

`set_name.add(element)`

یک عنصر را به ست اضافه می‌کند. بدیهی است اگر این عنصر از قبل وجود داشته باشد، هیچ تغییری ایجاد نمی‌شود.

```
>>> filled_set
{'b', 'c', 'a'}
>>> filled_set.add('d')
>>> filled_set
{'b', 'c', 'a', 'd'}
```

`set_name.remove(element)`

در ست به دنبال این عنصر می‌گردد و آن را حذف می‌کند. اگر عنصر داده شده وجود نداشته باشد `KeyError` می‌دهد.

```
>>> filled_set
{'b', 'c', 'a'}
>>> filled_set.remove('b')
>>> filled_set
{'c', 'a'}
>>> filled_set.remove('d')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'd'
```

`set_name.update(collection)`

سعی می‌کند تک تک عناصر شی `iterable` داده شده را به ست اضافه کند.

```
>>> filled_set
{'b', 'c', 'a'}
>>> filled_set.update(['b','d','h'])
>>> filled_set
{'d', 'c', 'b', 'a', 'h'}
```

`in`

با کلمه‌ی `in` میتوان وجود یک شی در یک ست را چک کرد.

```
>>> filled_set = {"a" , "a" , "b" , "c"}
>>> "d" in filled_set
False
>>> "c" in filled_set
True
```

عملگرهای بین ست‌ها

دو ست به صورت زیر تعریف شده‌اند.

```
>>> first_set = {1,2,3,4,5}
>>> second_set = {2,4,6,8,10}
```

اجتماع

```
>>> first_set | second_set
{1, 2, 3, 4, 5, 6, 8, 10}
```

اشتراک

```
>>> first_set & second_set
{2, 4}
```

تفاضل

```
>>> first_set - second_set
{1, 3, 5}
```

تفاضل متقارن

```
>>> first_set ^ second_set
{1, 3, 5, 6, 8, 10}
```

زیرمجموعه بودن

به جای علامت ⊂ از `>` و به جای علامت ⊆ از `=>` استفاده می‌شود.

```
>>> {1, 2} >= {1, 2, 3}
False
>>> {1, 2} <= {1, 2, 3}
True
```

`Frozenset`

نوعی *ست* است با این تفاوت که تغییرناپذیر (*immutable*) است. پس فقط از عملگرها و متدهایی که تغییری ایجاد نمی‌کنند، پشتیبانی می‌کند.

```
>>> a = frozenset([1,1,2,3,4])
>>> a.add(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
>>> len(a)
4
>>> b = frozenset([1,2,3])
>>> a - b
frozenset([4])
```

می‌بینید که این نوع *ست* ، از تابع *len* که طول را برمی‌گرداند و ست را تغییر نمی‌دهد و همچنین عملگر `-` پشتیبانی می‌کند.