Sets

یکی دیگه از داده ساختارهای آماده توی پایتون، مجموعه ست. خواصی که مجموعه ها دارن، تا حد خوبی شبیه خواص مجموعه ها داخل ریاضیه.

ترتیب:

ما توی لیست، برخلاف داده ساختارهای قبلی ای که داشتیم، ترتیب و حتی اندیس نداریم.

تغییر پذیری:

عناصر مجموعه ها داخل پایتون نمیتونن تغییر کنن؛ البته میتونیم عناصر رو حذف یا اضافه کنیم، اما آپشن تغییر وجود نداره.

عضو تکراری:

مثل مجموعه های ریاضی، داخل پایتون هم یه مجموعه نمی تونه عنصر تکراری داشته باشه. اگه عضو تکراری واردش کنیم تاثیری روی اعضای مجموعمون نمیذاره!

تعریف:

خب کم کم وقتشه ببینیم چطوری میتونیم یک Set رو داخل پایتون تعریف کنیم. نماد مجموعه توی پایتون مثل ریاضی، `{}` هستش؛ اما از طرفی میدونیم که این نماد برای دیکشنری ها هم استفاده میشه. یعنی چی؟ یعنی اگه بیایم و بگیم

```
1 unknown = {}
2 print(type(unknown))
```

به ما میگه که متغیر `unknown` از جنس دیکشنریه.

پس چطوری میتونیم یک مجموعه خالی بسازیم؟

```
s = set()
print(type(s))
```

خروجی کد بالا، به ما میگه که s یک مجموعه ست.

و اگه نخوایم مجموعه ای که داریم از اول خالی باشه چی؟

```
s = {'py', 's', 13, 13, False}
print(s)
```

▼ خروجی چیه؟

```
{False, 'py', 's', 13}
```

پس می بینیم که داخل یه مجموعه میتونیم عناصر از جنس های مختلف هم داشته باشیم.

دسترسی:

گفتیم که داخل مجموعه ها اندیس گذاری نداریم، پس چطوری می تونیم با مجموعه ها کار کنیم؟

حلقه روی مجموعه

```
s = {'py', 's', 13, False}
for item in s:
    print(item)
```

▼ خروجی چیه؟

```
False
py
s
13
```

چطوری چک کنیم یه عنصری داخل مجموعه هست؟

مثل همون چیزی که توی رشته ها دیدیم، با استفاده از دو سرکلید `in` و `not in` بودن یا نبودن یه عضو داخل مجموعه رو چک کنیم. میدونیم که خروجی این سرکلید ها از جنس boolean هستش.

```
s = {'py', 's', 13, False}
print('s' in s)
print('Py' in s)
print(13 not in s)
print(100 not in s)
```

▼ خروجی چیه؟

```
True
False
False
True
```

اضافه کردن: با استفاده از متد `add()` میتونیم عضو جدید به مجموعمون اضافه کنیم. دقت کن که گفتیم عضو تکراری اضافه کردن تفاوتی توی نمایش اعضامون ایجاد نمیکنه.

```
s = {'py', 's', 13, False}
s.add('ABC')
print(s)
```

▼ خروجی چیه؟

```
{False, 'py', 'ABC', 13, 's'}
```

نکته مهم:

`add` دقیقا یک ورودی داره.

اگه بخوایم بیش از یک عنصر رو به مجموعه اضافه کنیم یا حتی کل عنصرهای یک لیست رو، میتونیم از `update` استفاده کنیم.

```python
s = {'py', 's', 13, False}
li = ['AP', 13]
s.update(li)
print(s)
```

▼ خروجی چیه؟

```
{False, 'py', 13, 'AP', 's'}
```

حذف کردن:

واسه حذف یک عنصر از یک مجموعه هم میشه از `remove` استفاده کرد و هم از `discard`

```python
s = {'py', 's', 13, False}
s.remove('s')
print(s)
```

▼ خروجی چیه؟

```
{False, 'py', 13}
```

```python
s = {'py', 's', 13, False}
s.discard('s')
print(s)
```

▼ خروجی چیه؟

```
{False, 'py', 13}
```

پس تفاوت کجاست؟

در صورتی که عنصری که قصد حذفش رو داریم داخل مجموعه *نباشد*، `remove` ارور میده اما `discard` نه.

یک روش دیگه واسه حذف عنصر هست و اون هم اینه که از `pop` استفاده کنیم اما نکته اینجاست که `pop` یک عنصر رو به صورت *تصادفی* حذف میکنه.

اگر قصد داشته باشیم تا یک مجموعه رو به صورت کامل خالی کنیم چی؟ کافیه که از متد `clear()` استفاده کنیم.

```
s = {'py', 's', 13, False}
s.clear()
print(s)
```

▼ خروجی چیه؟

```
{}
```

عملیات های ریاضی روی مجموعه ها:

برای هر عملیاتی که میخوایم بگیم، سه روش معرفی می کنیم. یک متد که مقدار جدید رو برمیگردونه، یه متد که خود مجموعه رو تغییر میده و یک عملگر که مقدار جدید رو برمیگردونه.

اجتماع:

اگه بخوایم اجتماع دو مجموعه رو بگیریم و نتیجه رو داخل یک متغیر دیگه بریزیم، میتونیم از متد `union` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set3 = set1.union(set2)

print(set1)
print(set2)
print(set3)
```

▼ خروجی چیه؟

```
{'py', False, 13, 's'}
{'s', True, 'ABC'}
{'py', False, True, 'ABC', 's', 13}
```

همچنین با استفاده از متد `union` میشه اجتماع چندتا مجموعه رو هم گرفت.

```python
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set3 = {False, 15}
set4 = set1.union(set2, set3)

print(set4)
```

▼ خروجی چیه؟

```
{'py', False, 'ABC', True, 's', 13, 15}
```

به جای استفاده از `union` میتونیم از عملگر `|` استفاده کنیم.

```python
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set3 = set1 | set2

print(set1)
print(set2)
print(set3)
```

▼ خروجی چیه؟

```
{'py', False, 's', 13}
{'ABC', True, 's'}
{False, True, 's', 'py', 'ABC', 13}
```

و اگر هم بخوایم که از عملگر = استفاده نکنیم و نتیجه رو داخل یکی از مجموعه ها ذخیره کنیم، میتونیم از متد `update` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set1.update(set2)

print(set1)
print(set2)
```

▼ خروجی چیه؟

```
{False, True, 's', 'py', 'ABC', 13}
{'ABC', 's', True}
```

اشتراک:

اگه بخوایم اشتراک دو مجموعه رو بگیریم و نتیجه رو داخل یک متغیر دیگه بریزیم، میتونیم از متد `intersection` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set3 = set1.intersection(set2)

print(set1)
print(set2)
print(set3)
```

▼ خروجی چیه؟

```
{'py', False, 13, 's'}
{'s', True, 'ABC'}
{'s'}
```

همچنین با استفاده از متد `intersection` میشه اشتراک چندتا مجموعه رو هم گرفت.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set3 = {False, 15}
set4 = set1.intersection(set2, set3)

print(set4)
```

▼ **خروجی چیه؟**

```
{'s'}
```

به جای استفاده از `intersection` میتونیم از عملگر `&` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set3 = set1 & set2

print(set1)
print(set2)
print(set3)
```

▼ **خروجی چیه؟**

```
{'py', False, 's', 13}
{'ABC', True, 's'}
{'s'}
```

و اگر هم بخوایم که از عملگر `=` استفاده نکنیم و نتیجه رو داخل یکی از مجموعه ها ذخیره کنیم، میتونیم از متد `intersection_update` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set1.intersection_update(set2)
```

```
5
6  print(set1)
   print(set2)
```

▼ خروجی چیه؟

```
{'s'}
{'ABC', 's', True}
```

تفاضل:

اگه بخوایم تفاضل دو مجموعه رو بگیریم و نتیجه رو داخل یک متغیر دیگه بریزیم، میتونیم از متد `difference` استفاده کنیم.

```
1  set1 = {'py', 's', 13, False}
2  set2 = {'ABC', 's', True}
3  set3 = set1.difference(set2)
4
5  print(set1)
6  print(set2)
7  print(set3)
```

▼ خروجی چیه؟

```
{'py', False, 13, 's'}
{'s', True, 'ABC'}
{'py', False, 13}
```

همچنین با استفاده از متد `difference` میشه اختلاف یک مجموعه رو با چندتا مجموعه رو هم گرفت.

```
1  set1 = {'py', 's', 13, False}
2  set2 = {'ABC', 's', True}
3  set3 = {False, 15}
4  set4 = set1.difference(set2, set3)
5
```

```
print(set4)
```

▼ خروجی چیه؟

```
{'py', 13}
```

به جای استفاده از `difference` میتونیم از عملگر `-` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set3 = set1 - set2

print(set1)
print(set2)
print(set3)
```

▼ خروجی چیه؟

```
{'py', False, 's', 13}
{'ABC', True, 's'}
{False, 'py', 13}
```

و اگر هم بخوایم که از عملگر `=` استفاده نکنیم و نتیجه رو داخل یکی از مجموعه ها ذخیره کنیم، میتونیم از متد `difference_update` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set1.difference_update(set2)

print(set1)
print(set2)
```

▼ خروجی چیه؟

```
{False, 'py', 13}
{'ABC', 's', True}
```

تفاضل متقارن:

اگه بخوایم تفاضل متقارن دو مجموعه رو بگیریم و نتیجه رو داخل یک متغیر دیگه بریزیم، میتونیم از متد `symmetric_difference` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set3 = set1.symmetric_difference(set2)

print(set1)
print(set2)
print(set3)
```

▼ خروجی چیه؟

```
{'py', False, 13, 's'}
{'s', True, 'ABC'}
{'py', False, True, 'ABC', 13}
```

به جای استفاده از `symmetric_difference` میتونیم از عملگر `^` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set3 = set1 ^ set2

print(set1)
print(set2)
print(set3)
```

▼ خروجی چیه؟

```
{'py', False, 's', 13}
{'ABC', True, 's'}
{'py', False, True, 'ABC', 13}
```

و اگر هم بخوایم که از عملگر = استفاده نکنیم و نتیجه رو داخل یکی از مجموعه ها ذخیره کنیم، میتونیم از متد `symmetric_difference_update` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'ABC', 's', True}
set1.symmetric_difference_update(set2)

print(set1)
print(set2)
```

▼ **خروجی چیه؟**

```
{'py', 13, False, True, 'ABC'}
{'ABC', 's', True}
```

زیرمجموعه و ابرمجموعه:

اگر بخوایم چک کنیم یک مجموعه زیر مجموعه یک مجوعه دیگر هست یا نه، میتونیم از متد `issubset` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'py', 's', 13, False}
set3 = {'ABC', 's', True}
set4 = {'py', 's'}
set5 = {'PY'}



print(set2.issubset(set1))
print(set3.issubset(set1))

```

```
print(set4.issubset(set1))
print(set5.issubset(set1))
```

▼ خروجی چیه؟

```
True
False
True
False
```

به جای استفاده از `issubset` میتونیم از عملگر `<=` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'py', 's', 13, False}
set3 = {'ABC', 's', True}
set4 = {'py', 's'}
set5 = {'PY'}

print(set2 <= set1)
print(set3 <= set1)
print(set4 <= set1)
print(set5 <= set1)
```

▼ خروجی چیه؟

```
True
False
True
False
```

اگر هم بخوایم چک کنیم که یک مجموعه، زیرمجموعه محض یا سره یک مجموعه دیگه هست یا نه، میتونیم از عملگر `>` استفاده کنیم.

```
set1 = {'py', 's', 13, False}
set2 = {'py', 's', 13, False}
set3 = {'ABC', 's', True}
```

```
set4 = {'py', 's'}
set5 = {'PY'}

print(set2 < set1)
print(set3 < set1)
print(set4 < set1)
print(set5 < set1)
```

▼ خروجی چیه؟

```
False
False
True
False
```

همچنین میتونیم چک کنیم ایا یک مجموعه حاوی یک مجموعه دیگه هست یا نه.

```
set1 = {'py', 's', 13, False}
set2 = {'py', 's', 13, False}
set3 = {'ABC', 's', True}
set4 = {'py', 's'}
set5 = {'PY'}

print(set1.issuperset(set2))
print(set1.issuperset(set3))
print(set1.issuperset(set4))
print(set1.issuperset(set5))
```

▼ خروجی چیه؟

```
True
False
True
False
```

به جای استفاده از `issuperset` میتونیم از عملگر `>=` استفاده کنیم.

```python
set1 = {'py', 's', 13, False}
set2 = {'py', 's', 13, False}
set3 = {'ABC', 's', True}
set4 = {'py', 's'}
set5 = {'PY'}

print(set1 >= set2)
print(set1 >= set3)
print(set1 >= set4)
print(set1 >= set5)
```

▼ خروجی چیه؟

```
True
False
True
False
```

اگر هم بخوایم چک کنیم که یک مجموعه شامل یک مجموعه دیگه میشه و باهاش هم برابر نباشه، میتونیم از عملگر `>` استفاده کنیم.

```python
set1 = {'py', 's', 13, False}
set2 = {'py', 's', 13, False}
set3 = {'ABC', 's', True}
set4 = {'py', 's'}
set5 = {'PY'}

print(set1 > set2)
print(set1 > set3)
print(set1 > set4)
print(set1 > set5)
```

▼ خروجی چیه؟

```
False
False
```

```
True
False
```

مزایا:

- غیرتکراری: داخل مجموعه، عنصر تکراری نمی تونیم داشته باشیم.
- سرعت چک کردن: مجموعه ها داخل پایتون بهینه شدن برای اینکه بتونیم چک کنیم ببینیم عنصری داخل مجموعه هست یا نه. (عملگر `in` که بالاتر اشاره شد.)
- قابل تغییر: با استفاده از `add` و `pop` می تونیم یک عنصر رو به مجموعه اضافه کنیم یا از از مجموعه حذف کنیم.

معایب:

- ترتیب نداشتن: اینکه مجموعه ها ترتیب ندارن باعث میشه که نتونیم به ترتیبی که می بینیم (ترتیبی که پرینت شده) اعتماد کنیم و بعضی جاها ممکنه برامون دردسر بشه.
- محدود بودن متدها: مجموعه ها متدهای خیلی محدودتری نسبت به بقیه داده ساختارهای آماده دارن.
- حافظه: مجموعه ها از حافظه بیشتری نسبت به داده ساختاری مثل لیست استفاده میکنن به این دلیل که یه حافظه اضافه ای برای ذخیره کردن مقدار هش نیاز دارن و این داخل مجموعه های کوچیک، خیلی بده.
- محبوبیت پایین: مجموعه ها نسبت به داده ساختارهای دیگه کمتر استفاده میشن و همین باعث شده که کتابخونه های کمتری برای کار باهاشون وجود داشته باشه.

مطالعه بیشتر: یکی از داده ساختارهایی که میتونه جالب باشه، frozenset هستش که می تونین از این لینک درموردش مطالعه کنین.