برنامه نویسی پیشرفته

مدرس
دکتر حمید ترکاشوند

Imh.Torkashvand@gmail.com

فصل سوم

مجموعه‌ها (آرایه‌ها) در پایتون

Imh.Torkashvand@gmail.com

مجموعه‌ها (آرایه‌ها) در پایتون

چهار نوع داده جمع‌آوری در زبان برنامه‌نویسی پایتون وجود دارد:

- **لیست** مجموعه‌ای مرتب و قابل تغییر است که اعضای تکراری را می‌پذیرد.
- **تاپل** مجموعه‌ای مرتب و غیرقابل تغییر است. اعضای تکراری را مجاز می‌داند.
- **مجموعه (Set)** مجموعه‌ای است که نامرتب، غیرقابل تغییر* و بدون اندیس است. هیچ عضو تکراری ندارد.
- **دیکشنری** مجموعه‌ای است که مرتب** و قابل تغییر است. هیچ عضو تکراری ندارد.

Imh.Torkashvand@gmail.com

لیست

- از لیست‌ها برای ذخیره چندین آیتم در یک متغیر واحد استفاده می‌شود.
- لیست‌ها یکی از ۴ نوع داده‌ی داخلی در پایتون هستند که برای ذخیره‌ی مجموعه‌ای از داده‌ها استفاده می‌شوند.
- سه نوع دیگر Tuple، Set و Dictionary هستند که هر کدام کیفیت‌ها و کاربردهای متفاوتی دارند.
- لیست‌ها با استفاده از براکت‌های مربعی ایجاد می‌شوند.

آیتم‌های لیست مرتب، قابل تغییر و دارای مقادیر تکراری هستند.
آیتم‌های لیست اندیس‌گذاری می‌شوند، آیتم اول اندیس دارد `[0]` ، آیتم دوم اندیس دارد `[1]` و غیره.

وقتی می‌گوییم لیست‌ها مرتب هستند، به این معنی است که آیتم‌ها ترتیب مشخصی دارند و این ترتیب تغییر نخواهد کرد.
اگر موارد جدیدی را به لیست اضافه کنید، موارد جدید در انتهای لیست قرار می‌گیرند.

نکته: برخی متدهای لیست وجود دارند که ترتیب را تغییر می‌دهند، اما به طور کلی ترتیب آیتم‌ها تغییر نخواهد کرد

متدهای لیست

پایتون مجموعه‌ای از متدهای از پیش تعریف شده دارد که می‌توانید روی لیست‌ها از آنها استفاده کنید.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Imh.Torkashvand@gmail.com

متدهای لیست

متد `append()` یک عنصر را به انتهای لیست اضافه می‌کند.

```
fruits = ['apple', 'banana', 'cherry']
fruits.append("orange")
print(fruits)
```

```
['apple', 'banana', 'cherry', 'orange']
```

```
a = ["apple", "banana", "cherry"]
b = ["Ford", "BMW", "Volvo"]
a.append(b)
print(a)
```

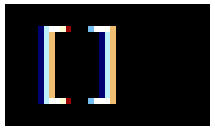
```
['apple', 'banana', 'cherry', ["Ford", "BMW", "Volvo"]]
```

Imh.Torkashvand@gmail.com

متدهای لیست

متد `clear()` تمام عناصر یک لیست را حذف می‌کند.

```
fruits = ['apple', 'banana', 'cherry', 'orange']
fruits.clear()
print(fruits)
```



Imh.Torkashvand@gmail.com

متدهای لیست

متد copy() یک کپی از لیست مشخص شده را برمی‌گرداند.

```
fruits = ['apple', 'banana', 'cherry', 'orange']
x = fruits.copy()
print(x)
```

```
['apple', 'banana', 'cherry', 'orange']
```

متدهای لیست

متد `count()` تعداد شمارش عناصر با مقدار مشخص شده را برمی‌گرداند.

تعداد دفعاتی که مقدار ۹ در لیست ظاهر می‌شود را برمی‌گرداند:

```
points = [1, 4, 2, 9, 7, 8, 9, 3, 1]
x = points.count(9)
print(x)
```

2

```
fruits = ['apple', 'banana', 'cherry']
x = fruits.count("cherry")
print(x)
```

1

Imh.Torkashvand@gmail.com

متدهای لیست

متد extend() عناصر لیست مشخص شده (یا هر عنصر تکرارشونده) را به انتهای لیست فعلی اضافه می‌کند.

```
fruits = ['apple', 'banana', 'cherry']
cars = ['Ford', 'BMW', 'Volvo']
fruits.extend(cars)
print(fruits)
```

```
['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']
```

متدهای لیست

متد index() موقعیت اولین وقوع مقدار مشخص شده را برمی‌گرداند.

مثال:
موقعیت «گیلاس» را پیدا کن، اما جستجو را از موقعیت ۴ شروع کن:

```
fruits =
['apple', 'banana', 'cherry', 'kiwi', 'mango',
'orange', 'cherry']
x = fruits.index("cherry", 4)
print(x)
```

```
6
```

```
fruits = ['apple', 'banana', 'cherry']
x = fruits.index("cherry")
print(x)
```

```
2
```

Imh.Torkashvand@gmail.com

متدهای لیست

متد `insert()` مقدار مشخص شده را در موقعیت مشخص شده درج می‌کند.

```
fruits = ['apple', 'banana', 'cherry']
fruits.insert(1, "orange")
print(fruits)
```

```
['apple', 'orange', 'banana', 'cherry']
```

متدهای لیست

متد pop() عنصر را در موقعیت مشخص شده حذف می‌کند.

این متد هم یه آیتم را از لیست حذف می‌کند، ولی مقدارش را برمی‌گرداند. اگر اندیسی را به pop ندیم، آخرین آیتم رو حذف می‌کند و برمی‌گرداند. اگر اندیس مشخص کنیم، آیتم همون اندیس رو حذف می‌کند.

```
fruits = ['apple', 'banana', 'cherry']
fruits.pop(1)
print(fruits)
```

```
['apple', 'cherry']
```

```
my_list = [10, 20, 30]
x = my_list.pop()  # عنصر آخر (۳۰) حذف می‌شود و برمی‌گرداند
print(my_list)
print(x)
```

```
[10, 20]
30
```

```
fruits = ['apple','banana','cherry']
fruits=fruits.pop(1)
print(fruits)
```

???

Imh.Torkashvand@gmail.com

متدهای لیست

متد `remove()` اولین مورد از عنصری که مقدار مشخص شده را دارد، حذف می‌کند.

- این متد یه مقدار خاص رو از لیست حذف می‌کند. فقط مقدار را حذف می کند هیچ مقداری برگردانده نمی شود که در متغیری ذخیره شود.

اما اگر آن مقدار چند بار توی لیست باشد، فقط اولین مورد رو حذف می‌کند. اگر آن مقدار توی لیست نباشد، Error می‌دهد.

```
fruits = ['apple','banana','cherry','banana']
fruits.remove("banana")
print(fruits)
```

```
['apple', 'cherry', 'banana']
```

Imh.Torkashvand@gmail.com

متدهای لیست

متد del ( ) این دستور می‌تونه یه آیتم خاص رو از لیست حذف کنه یا حتی کل لیست رو از حافظه پاک کند.

```
my_list = [1, 2, 3]
del my_list[1]
#del my_list
print (my_list)
```

```
[1, 3]
```

Imh.Torkashvand@gmail.com

متدهای لیست

متد `reverse()` ترتیب مرتب سازی عناصر را معکوس می‌کند.

```
fruits = ['apple', 'banana', 'cherry']
fruits.reverse()
print(fruits)
```

```
['cherry', 'banana', 'apple']
```

متدهای لیست

متد `sort()` به طور پیش‌فرض لیست را به صورت صعودی مرتب می‌کند.
همچنین می‌توانید تابعی برای تعیین معیارهای مرتب‌سازی ایجاد کنید.

فقط روی **لیست‌ها** قابل استفاده است .
به‌صورت پیش‌فرض، **صعودی (از کوچک به بزرگ)** مرتب می‌کند .

```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort()
print(cars)
```

```
['Volvo', 'Ford', 'BMW']
```

Imh.Torkashvand@gmail.com

تاپل‌های پایتون

از تاپل‌ها برای ذخیره چندین آیتم در یک متغیر واحد استفاده می‌شود.
تاپل یکی از ۴ نوع داده‌ی داخلی در پایتون است که برای ذخیره‌ی مجموعه‌ای از داده‌ها استفاده می‌شود.
یک تاپل مجموعه‌ای است که مرتب و **غیرقابل تغییر** است .
تاپل‌ها (Tuples) با براکت‌های گرد نوشته می‌شوند.

- آیتم‌های تاپل اندیس‌گذاری می‌شوند، آیتم اول اندیس دارد `[0]` ، آیتم دوم اندیس دارد `[1]` و غیره...
- وقتی می‌گوییم تاپل‌ها مرتب هستند، به این معنی است که آیتم‌ها ترتیب مشخصی دارند و این ترتیب تغییر نخواهد کرد.
- تاپل‌ها غیرقابل تغییر هستند، به این معنی که نمی‌توانیم پس از ایجاد تاپل، آیتم‌ها را تغییر دهیم، اضافه یا حذف کنیم.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

Imh.Torkashvand@gmail.com

تاپل‌های پایتون

از آنجایی که تاپل‌ها اندیس‌گذاری شده‌اند، می‌توانند آیتم‌هایی با مقادیر یکسان داشته باشند:

```
thistuple =
("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'cherry')
```

Imh.Torkashvand@gmail.com

The tuple() Constructor

در پایتون، منظور از سازنده (constructor) برایtuple ، تابعی است که یک تاپل جدید می‌سازد. این سازنده همان تابع tuple() است

همچنین می‌توان از سازنده‌ی tuple() برای ایجاد یک تاپل استفاده کرد.

```
thistuple =tuple(("apple", "banana", "cherry"))
 # note the double round-brackets
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

```
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(my_tuple)
```

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

Imh.Torkashvand@gmail.com

دسترسی به آیتم‌های تاپل

شما می‌توانید با ارجاع به شماره اندیس، داخل کروشه، به آیتم‌های تاپل دسترسی پیدا کنید:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

نکته: اولین آیتم دارای اندیس ۰ است.

نمایه‌سازی منفی تاپل

اندیس‌گذاری منفی به معنای شروع از انتها است.
-1به آخرین مورد اشاره می‌کند،-2 به دومین مورد آخر اشاره دارد و غیره.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

آخرین عنصر تاپل را چاپ کن:

```
cherry
```

Imh.Torkashvand@gmail.com

محدوده شاخص‌ها تاپل

- می‌توان با مشخص کردن نقطه شروع و پایان محدوده، طیفی از شاخص‌ها را مشخص کرد.
- هنگام مشخص کردن یک محدوده، مقدار برگشتی یک تاپل جدید با آیتم‌های مشخص شده خواهد بود.

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
('cherry', 'orange', 'kiwi')
```

توجه: جستجو از اندیس ۲ (شامل) شروع و در اندیس ۵ (شامل نمی‌شود) پایان می‌یابد.

این مثال آیتم‌ها را از ابتدا تا " kiwi " برمی‌گرداند، اما شامل آنها نمی‌شود:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

```
('apple', 'banana', 'cherry', 'orange')
```

Imh.Torkashvand@gmail.com

محدوده شاخص‌های منفی

اگر می‌خواهید جستجو را از انتهای تاپل شروع کنید، اندیس‌های منفی را مشخص کنید:

این مثال آیتم‌ها را از اندیس -۴ (شامل) تا اندیس -۱ (حذف) برمی‌گرداند.

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

```
('orange', 'kiwi', 'melon')
```

بررسی تاپل

برای تعیین اینکه آیا یک آیتم مشخص شده در یک تاپل وجود دارد یا خیر، از in کلمه کلیدی زیر استفاده کنید :

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

```
Yes, 'apple' is in the fruits tuple
```

به‌روز رسانی تاپل‌ها

تاپل‌ها غیرقابل تغییر هستند، به این معنی که پس از ایجاد تاپل، نمی‌توانید آیتم‌ها را تغییر دهید، اضافه یا حذف کنید.
اما برخی راه‌حل‌ها وجود دارد.

وقتی یک تاپل ایجاد شد، نمی‌توانید مقادیر آن را تغییر دهید. تاپل‌ها **غیرقابل تغییر** یا به اصطلاح «**تغییرناپذیر**» **هستند.**
اما یک راه حل وجود دارد. می‌توانید تاپل را به یک لیست تبدیل کنید، لیست را تغییر دهید و دوباره لیست را به یک تاپل تبدیل کنید.

❖ تاپل را به یک لیست تبدیل کنید تا بتوانید آن را تغییر دهید:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

```
("apple", "kiwi", "cherry")
```

به‌روز رسانی تاپل‌ها

از آنجایی که تاپل‌ها تغییرناپذیر هستند، متد داخلی ندارند append()، اما روش‌های دیگری برای اضافه کردن آیتم‌ها به یک تاپل وجود دارد.

۱. تبدیل به لیست : درست مانند راه‌حل تغییر یک تاپل، می‌توانید آن را به یک لیست تبدیل کنید، آیتم(های) خود را اضافه کنید و دوباره آن را به یک تاپل تبدیل کنید.

❖ تاپل را به یک لیست تبدیل کنید، "نارنجی" را اضافه کنید و دوباره آن را به یک تاپل تبدیل کنید:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
print(thistuple)
```

```
('apple', 'banana', 'cherry', 'orange')
```

به‌روز رسانی تاپل‌ها

۲. **اضافه کردن چندتایی به یک چندتایی** . شما مجاز به اضافه کردن چندتایی به چندتایی‌ها هستید، بنابراین اگر می‌خواهید یک (یا چند) آیتم اضافه کنید، یک چندتایی جدید با آیتم(ها) ایجاد کنید و آن را به چندتایی موجود اضافه کنید:

- یک تاپل جدید با مقدار " orange " ایجاد کنید و آن تاپل را اضافه کنید:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)
```

```
('apple', 'banana', 'cherry', 'orange')
```

نکته: هنگام ایجاد یک تاپل با تنها یک آیتم، به یاد داشته باشید که بعد از آیتم یک کاما قرار دهید، در غیر این صورت به عنوان یک تاپل شناسایی نخواهد شد.

به‌روز رسانی تاپل‌ها

تاپل‌ها **غیرقابل تغییر** هستند ، بنابراین نمی‌توانید آیتم‌ها را از آنها حذف کنید، اما می‌توانید از همان راهکاری که برای تغییر و اضافه کردن آیتم‌های تاپل استفاده کردیم، استفاده کنید:

- تاپل را به یک لیست تبدیل کنید، "سیب" را حذف کنید و دوباره آن را به یک تاپل تبدیل کنید:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
print(thistuple)
```

```
('banana', 'cherry')
```

- یا می‌توانید تاپل را به طور کامل حذف کنید:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error
because the tuple no longer exists
```

```
Traceback (most recent call last):
  File "demo_tuple_del.py", line 3, in <module>
    print(thistuple) #this will raise an error because the
NameError: name 'thistuple' is not defined
```

Python - Unpack Tuples

وقتی یک تاپل ایجاد می‌کنیم، معمولاً به آن مقادیری اختصاص می‌دهیم. به این کار «بسته‌بندی» یک تاپل می‌گویند:

بسته‌بندی یک تاپل:

```
fruits = ("apple", "banana", "cherry")
print(fruits)
```

```
('apple', 'banana', 'cherry')
```

اما در پایتون، ما همچنین مجاز به استخراج مقادیر به متغیرها هستیم. به این کار "باز کردن" (unpacking) گفته می‌شود:

نکته: تعداد متغیرها باید با تعداد مقادیر موجود در تاپل مطابقت داشته باشد، در غیر این صورت، باید از علامت ستاره برای جمع‌آوری مقادیر باقی‌مانده به صورت یک لیست استفاده کنید.

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

```
apple
banana
cherry
```

تاپل‌های پایتون

استفاده از ستاره*

❖ اگر تعداد متغیرها کمتر از تعداد مقادیر باشد، می‌توانید یک *به نام متغیر اضافه کنید و مقادیر به صورت یک لیست به متغیر اختصاص داده می‌شوند

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits
print(green)
print(yellow)
print(red)
```

بقیه مقادیر را به عنوان لیستی به نام " red " اختصاص دهید:

```
apple
banana
['cherry', 'strawberry', 'raspberry']
```

❖ اگر علامت ستاره به نام متغیر دیگری غیر از آخرین نام اضافه شود، پایتون مقادیر را به متغیر اختصاص می‌دهد تا زمانی که تعداد مقادیر باقی مانده با تعداد متغیرهای باقی مانده برابر شود.

❖ لیستی از مقادیر را به متغیر " tropic " اضافه کنید:

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")
(green, *tropic, red) = fruits
print(green)
print(tropic)
print(red)
```

```
apple
['mango', 'papaya', 'pineapple']
cherry
```

تاپل های حلقه

- می‌توانید با استفاده از یک حلقه for ، آیتم‌های تاپل را پیمایش کنید.

روی آیتم‌ها پیمایش کن و مقادیر را چاپ کن:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

```
apple
banana
cherry
```

- می‌توانید با استفاده از یک حلقه while ، آیتم‌های تاپل را پیمایش کنید.

از این تابع برای تعیین طول تاپل استفاده کنید ، سپس از ۰ شروع کنید و با مراجعه به اندیس‌های آیتم‌های تاپل، به صورت حلقه‌ای آنها را پیمایش کنید. len()
به یاد داشته باشید که بعد از هر تکرار، اندیس را ۱ واحد افزایش دهید.

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

```
apple
banana
cherry
```

اتصال تاپل‌ها

برای اتصال دو یا چند تاپل می‌توانید از عملگر + زیر استفاده کنید:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

اگر می‌خواهید محتوای یک تاپل را به تعداد مشخصی ضرب کنید، می‌توانید از عملگر * زیر استفاده کنید:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2
print(mytuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

متدهای تاپل

- متد `count()` تعداد دفعاتی که یک مقدار مشخص شده در تاپل ظاهر می‌شود را برمی‌گرداند.

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.count(5)
print(x)
```

```
2
```

- متد `index()` اولین رخداد مقدار مشخص شده را پیدا می‌کند.
- `index()`اگر مقدار مورد نظر پیدا نشود، متد یک استثنا ایجاد می‌کند

- اولین مورد یافت شده با مقدار ۸ را جستجو کن و موقعیت آن را برگردان:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.index(8)
print(x)
```

```
3
```

مجموعه‌های پایتون

- مجموعه‌ها برای ذخیره چندین آیتم در یک متغیر واحد استفاده می‌شوند.
- Set یکی از ۴ نوع داده‌ی داخلی در پایتون است که برای ذخیره‌ی مجموعه‌ای از داده‌ها استفاده می‌شود.
- یک مجموعه، مجموعه‌ای است که نامرتب ، غیرقابل تغییر* و بدون اندیس است .
- **توجه:** اقلام مجموعه غیرقابل تغییر هستند، اما می‌توانید اقلام را حذف و اقلام جدید اضافه کنید.
- مجموعه‌ها با براکت‌های مجعد نوشته می‌شوند.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

```
{'banana', 'apple', 'cherry'}
```

- منظور از بی‌نظمی این است که اقلام موجود در یک مجموعه، ترتیب مشخصی ندارند.
- آیتم‌های مجموعه می‌توانند هر بار که از آنها استفاده می‌کنید، به ترتیب متفاوتی ظاهر شوند و نمی‌توان به آنها با اندیس یا کلید اشاره کرد.
- اقلام مجموعه غیرقابل تغییر هستند، به این معنی که ما نمی‌توانیم اقلام را پس از ایجاد مجموعه تغییر دهیم.

مجموعه‌های پایتون

❖ set() Constructor

❖ همچنین می‌توان از سازنده‌ی `set()` برای ایجاد یک مجموعه استفاده کرد.

```
thisset = set(("apple", "banana", "cherry")) #
note the double round-brackets
print(thisset)
```

```
{'banana', 'cherry', 'apple'}
```

❖ از دیدگاه پایتون، مجموعه‌ها به عنوان اشیاء با نوع داده ' set ' تعریف می‌شوند:

نوع داده یک مجموعه چیست؟

```
myset = {"apple", "banana", "cherry"}
print(type(myset))
```

```
<class 'set'>
```

دسترسی به آیتم‌های مجموعه

شما نمی‌توانید با ارجاع به یک اندیس یا یک کلید به آیتم‌های یک مجموعه دسترسی پیدا کنید.
اما می‌توانید با استفاده از یک حلقه for ، آیتم‌های مجموعه را پیمایش کنید، یا با استفاده از کلمه کلیدی in ، وجود یک مقدار مشخص در یک مجموعه را جویا شوید.

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

```
cherry
apple
banana
```

❖ وقتی یک مجموعه ایجاد می‌شود، نمی‌توانید موارد آن را تغییر دهید، اما می‌توانید موارد جدیدی اضافه کنید.

❖برای اضافه کردن یک آیتم به یک مجموعه از متد add() استفاده کنید.

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

```
{'banana', 'cherry', 'apple', 'orange'}
```

Imh.Torkashvand@gmail.com

دسترسی به آیتم‌های مجموعه

❖ برای اضافه کردن آیتم ها از یک مجموعه دیگر به مجموعه فعلی، از این متد update() استفاده کنید.

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

```
{'apple', 'mango', 'cherry', 'pineapple', 'banana', 'papay
```

❖ اضافه کردن هر Iterable

شیء موجود در متد update() لازم نیست حتماً یک مجموعه باشد، می‌تواند هر شیء قابل تکرار (تاپل‌ها، لیست‌ها، دیکشنری‌ها و غیره) باشد .

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]
thisset.update(mylist)
print(thisset)
```

```
{'banana', 'cherry', 'apple', 'orange', 'kiwi'}
```

دسترسی به آیتم‌های مجموعه

❖ برای حذف یک آیتم در یک مجموعه، از متد ()remove، یا ()discard استفاده کنید.

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

```
{'apple', 'cherry'}
```

نکته: اگر آیتمی که باید حذف شود وجود نداشته باشد، خطا ()remove ایجاد می‌شود.

❖ با استفاده از discard() کلمه "banana" را حذف کنید:

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

```
{'cherry', 'apple'}
```

نکته: اگر آیتمی که باید حذف شود وجود نداشته باشد، خطایی ایجاد discard() نخواهد **شد.**

مجموعه‌های پایتون

شما همچنین می‌توانید از این pop() روش برای حذف یک آیتم استفاده کنید، اما این روش یک آیتم تصادفی را حذف می‌کند، بنابراین نمی‌توانید مطمئن باشید که کدام آیتم حذف می‌شود.
مقدار بازگشتی این متد pop() ، آیتم حذف شده است.

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)
print(thisset)
```

```
cherry
{'apple', 'banana'}
```

نکته: مجموعه‌ها نامرتب هستند ، بنابراین هنگام استفاده از روش pop() ، نمی‌دانید کدام آیتم حذف می‌شود.

مجموعه‌های پایتون

❖ متد clear() مجموعه را خالی می‌کند:

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

```
set()
```

❖ کلمه del کلیدی مجموعه را به طور کامل حذف می‌کند:

```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

```
Traceback (most recent call last):
  File "demo_set_del.py", line 5, in <module>
    print(thisset) #this will raise an error because the s
NameError: name 'thisset' is not defined
```

Imh.Torkashvand@gmail.com

مجموعه‌های پایتون

می‌توانید با استفاده از یک حلقه for ، آیتم‌های مجموعه را پیمایش کنید:

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

```
banana
cherry
apple
```

روش‌های مختلفی برای اتصال دو یا چند مجموعه در پایتون وجود دارد.

- متدهای union() and update() تمام آیتم‌های هر دو مجموعه را به هم متصل می‌کنند.
- متد intersection() فقط مقادیر تکراری را نگه می‌دارد.
- متد difference() ، آیتم‌هایی از مجموعه اول که در مجموعه(های) دیگر نیستند را نگه می‌دارد.
- متد symmetric_difference() همه آیتم‌ها به جز موارد تکراری را نگه می‌دارد.

Imh.Torkashvand@gmail.com

مجموعه‌های پایتون

❖ متد union() یک مجموعه جدید با تمام آیتم‌های هر دو مجموعه را برمی‌گرداند.

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

```
{'c', 'b', 2, 3, 'a', 1}
```

❖ برای اتصال دو مجموعه استفاده کنید:

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = set1 | set2
print(set3)
```

```
{2, 'c', 'b', 1, 'a', 3}
```

❖ یک مجموعه و یک تاپل را به هم وصل کنید:

متد union() روش به شما امکان می‌دهد مجموعه‌ای را با انواع داده دیگر، مانند لیست‌ها یا تاپل‌ها، پیوند دهید.

```
x = {"a", "b", "c"}
y = (1, 2, 3)
z = x.union(y)
print(z)
```

```
{1, 'c', 3, 2, 'a', 'b'}
```

Imh.Torkashvand@gmail.com

مجموعه‌های پایتون

❖ متد update() تمام آیتم‌های یک مجموعه را در مجموعه دیگر درج می‌کند. مجموعه اصلی را تغییر می‌دهد و مجموعه جدیدی را برنمی‌گرداند.

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

```
{'b', 3, 1, 'a', 2, 'c'}
```

توجه: هر دو مورد union() و update() موارد تکراری را حذف می‌کنند.

❖ متد intersection() یک مجموعه جدید برمی‌گرداند که فقط شامل آیتم‌هایی است که در هر دو مجموعه وجود دارند.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1.intersection(set2)
print(set3)
```

```
{'apple'}
```

نکته: می‌توانید به جای متد **intersection()** از عملگر **&** استفاده کنید و همان نتیجه را خواهید گرفت.

تابع frozenset() در پایتون

❖ فریز کردن لیست و غیر قابل تغییر کردن آن:

تابع frozenset() یک شیء frozenset در خروجی برمیگرداند که مشابه set است با این تفاوت که نمی تواند تغییر کند .

```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
print(x)
```

```
frozenset({'banana', 'apple', 'cherry'})
```

اگر بخواهیم مقدار یک frozenset را تغییر بدهیم، با خطا روبرو می شویم:

```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
x[1] =  "strawberry"
```

```
Exception has occurred: TypeError ×
'frozenset' object does not support item assignment
  File "C:\Users\Administrator\Desktop\frozenset.py", line 3, in <module>
    x[1] =  "strawberry"
    ~^^^
TypeError: 'frozenset' object does not support item assignment
```

Imh.Torkashvand@gmail.com

دیکشنری‌های پایتون

- از دیکشنری‌ها برای ذخیره مقادیر داده در جفت‌های کلید: مقدار استفاده می‌شود. یک دیکشنری مجموعه‌ای است که مرتب، قابل تغییر و غیرقابل تکرار است. دیکشنری‌ها با براکت‌های نوشته می‌شوند و دارای کلید و مقدار هستند.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

- آیتم‌های دیکشنری مرتب و قابل تغییر هستند و موارد تکراری را مجاز نمی‌دانند.
- آیتم‌های دیکشنری به صورت جفت‌های کلید: مقدار نمایش داده می‌شوند و می‌توان با استفاده از نام کلید به آنها اشاره کرد.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

```
Ford
```

دیکشنری‌های پایتون

❖ dict() Constructor

همچنین می‌توان از سازنده‌ی **dict()** برای ساخت دیکشنری استفاده کرد.

```
thisdict = dict(name = "John",
age= 36,country= "Norway")
print(thisdict)
```

```
{'name': 'John', 'age': 36, 'country': 'Norway'}
```

❖ شما می‌توانید با اشاره به نام کلید یک دیکشنری، داخل کروشه، به آیتم‌های آن دسترسی پیدا کنید:

مقدار کلید " model " را دریافت می‌کند:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

```
Mustang
```

دیکشنری‌های پایتون

```
thisdict =    {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)
```

❖ همچنین روشی به نام get() وجود دارد که همان نتیجه را به شما می‌دهد:

```
Mustang
```

❖ متد keys() لیستی از تمام کلیدهای موجود در دیکشنری را برمی‌گرداند.

```
thisdict =    {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.keys()
print(x)
```

```
dict_keys(['brand', 'model', 'year'])
```

Imh.Torkashvand@gmail.com

دیکشنری‌های پایتون

❖ فهرست کلیدها، نمایی از دیکشنری است، به این معنی که هر تغییری که در دیکشنری انجام شود، در فهرست کلیدها منعکس خواهد شد.

مثال:
یک آیتم جدید به دیکشنری اصلی اضافه کنید و ببینید که لیست کلیدها نیز به‌روزرسانی می‌شود:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = car.keys()
print(x) #before the change
car["color"] = "white"
print(x) #after the change

print (car)
```

```
dict_keys(['brand', 'model', 'year'])
dict_keys(['brand', 'model', 'year', 'color'])
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'white'}
```

دیکشنری‌های پایتون

❖ متد values() لیستی از تمام مقادیر موجود در دیکشنری را برمی‌گرداند.

```
thisdict =    {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.values()
print(x)
```

```
dict_values(['Ford', 'Mustang', 1964])
```

❖ متد items() هر آیتم را در یک دیکشنری، به عنوان تاپل در یک لیست، برمی‌گرداند.

```
thisdict =    {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.items()
print(x)
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
```

Imh.Torkashvand@gmail.com

دیکشنری‌های پایتون

❖ برای تعیین اینکه آیا یک کلید مشخص شده در یک دیکشنری وجود دارد یا خیر، از کلمه in کلیدی زیر استفاده کنید:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

دیکشنری‌های پایتون

- متد update() ، دیکشنری را با آیتم‌های موجود در آرگومان داده شده به‌روزرسانی می‌کند.
- آرگومان باید یک دیکشنری یا یک شیء قابل تکرار با جفت‌های کلید:مقدار باشد.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

دیکشنری‌های پایتون

اضافه کردن یک آیتم به دیکشنری با استفاده از یک کلید اندیس جدید و اختصاص دادن یک مقدار به آن انجام می‌شود:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Imh.Torkashvand@gmail.com

دیکشنری‌های پایتون

روش‌های مختلفی برای حذف عناصر از یک دیکشنری وجود دارد:

❖ متد pop() ، آیتمی را که نام کلید مشخص شده را دارد، حذف می‌کند:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

❖ متد popitem() آخرین آیتم درج شده را حذف می‌کند (در نسخه‌های قبل از ۳,۷، یک آیتم تصادفی به جای آن حذف می‌شود):

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang'}
```

دیکشنری‌های پایتون

❖ کلمه کلیدی del، آیتمی را که نام کلید مشخص شده را دارد، حذف می‌کند:

❖کلمه کلیدی del همچنین می‌تواند دیکشنری را به طور کامل حذف کند.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

❖ متد clear()، دیکشنری را خالی می‌کند:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

```
{}
```

حلقه زدن در دیکشنری

- با استفاده از حلقه For می‌توانید در یک دیکشنری حلقه بزنید.
- هنگام حلقه زدن در یک دیکشنری، مقدار برگشتی کلیدهای دیکشنری هستند، اما متدهایی برای برگرداندن مقادیر نیز وجود دارد.

✓ کلید و مقادیر موجود در دیکشنری را یکی یکی چاپ می‌کند:

```
thisdict = {
  "brand":"Ford",
  "model":"Mustang",
  "year":1964
 }
 for x in thisdict:
    print(x,thisdict[x])
```

```
brand Ford
model Mustang
year 1964
```

✓ تمام نام‌های کلید موجود در دیکشنری را یکی یکی چاپ می‌کند:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

```
brand
model
year
```

✓ تمام مقادیر موجود در دیکشنری را یکی یکی چاپ می‌کند:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

```
Ford
Mustang
1964
```

Imh.Torkashvand@gmail.com

دیکشنری‌های پایتون

❖ از متد values() برای برگرداندن مقادیر یک دیکشنری استفاده کنید:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)
```

```
Ford
Mustang
1964
```

❖ از متد keys() برای برگرداندن کلیدهای یک دیکشنری استفاده کنید:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.keys():
  print(x)
```

```
brand
model
year
```

❖ با استفاده از متدitems() ، روی کلیدها و مقادیرحلقه بزنید:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

```
brand Ford
model Mustang
year 1964
```

Imh.Torkashvand@gmail.com

کپی از دیکشنری

شما نمی‌توانید یک دیکشنری را صرفاً با تایپ کردن کپی کنید dict2 = dict1، زیرا: فقط یک ارجاع به dict2 خواهد بود و تغییرات ایجاد شده در آن به طور خودکار در آن نیز ایجاد می‌شوند .

روش‌هایی برای ایجاد کپی از دیکشنری وجود دارد:

❖ با استفاده از متد copy()، یک کپی از دیکشنری تهیه کنید

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

❖ راه دیگر برای ایجاد کپی، استفاده از متد dict() است.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Imh.Torkashvand@gmail.com

Nested Dictionaries—دیکشنری‌های تو در تو

یک دیکشنری می‌تواند شامل دیکشنری‌های دیگری هم باشد، به این نوع دیکشنری، دیکشنری‌های تو در تو می‌گویند.

مثال:

یک دیکشنری ایجاد کنید که شامل سه دیکشنری باشد:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
print(myfamily)
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

lmh.Torkashvand@gmail.com

Nested Dictionaries—دیکشنری‌های تو در تو

یا اگر می‌خواهید سه دیکشنری را به یک دیکشنری جدید اضافه کنید:

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
print(myfamily)
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

دیکشنری‌های تو در تو—Nested Dictionaries

❖ دسترسی به آیتم‌ها در دیکشنری‌های تو در تو

برای دسترسی به آیتم‌های یک دیکشنری تودرتو، از نام دیکشنری‌ها استفاده می‌کنید و از دیکشنری بیرونی شروع می‌کنید:

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
  }
}
print( myfamily  ["child2"]["name"])
```

```
Tobias
```

Nested Dictionaries—دیکشنری‌های تو در تو

❖ **حلقه در دیکشنری‌های تو در تو**

می‌توانید با استفاده از items() روی یک دیکشنری حلقه بزنید:

مثال:

پیمایش کلیدها و مقادیر همه دیکشنری‌های تو در تو:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
for x, obj in myfamily.items():
   print(x)
   for y in obj:
     print(y + ':', obj[y])
```

```
child1
name: Emil
year: 2004
child2
name: Tobias
year: 2007
child3
name: Linus
year: 2011
```

پایان

Imh.Torkashvand@gmail.com