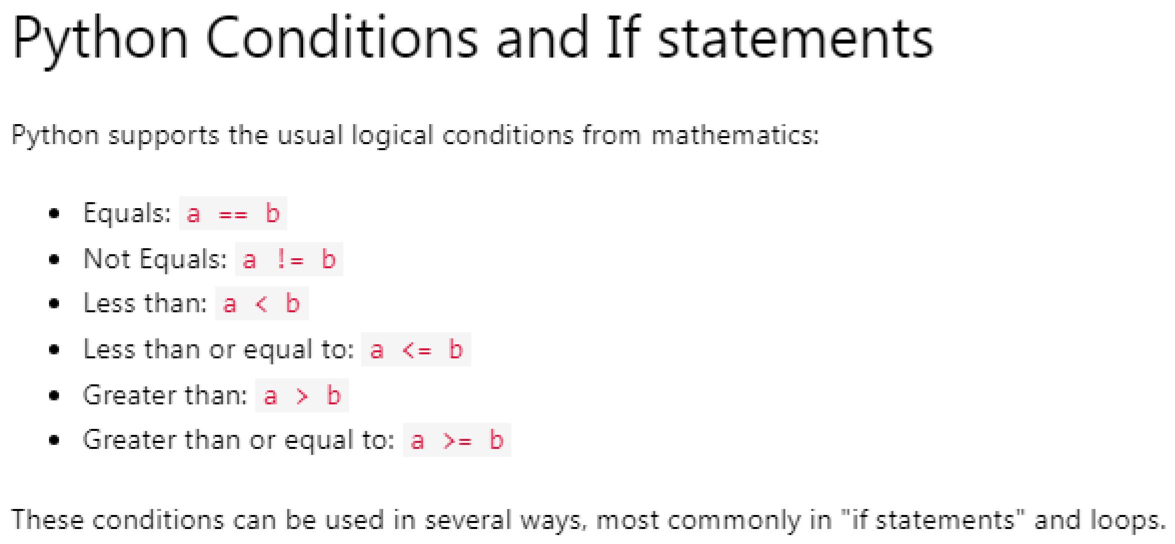
دستورات شرطی

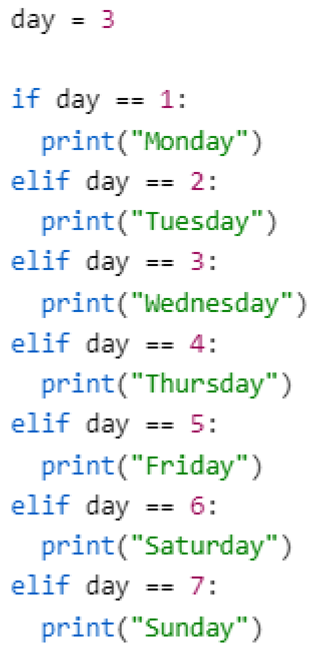


Python Conditions and If statements

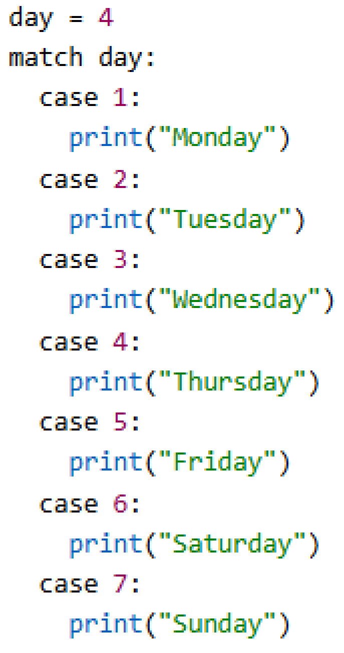
Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "if statements" and loops.

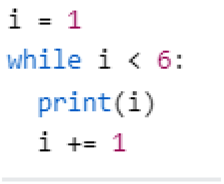


```python
day = 3

if day == 1:
  print("Monday")
elif day == 2:
  print("Tuesday")
elif day == 3:
  print("Wednesday")
elif day == 4:
  print("Thursday")
elif day == 5:
  print("Friday")
elif day == 6:
  print("Saturday")
elif day == 7:
  print("Sunday")
```

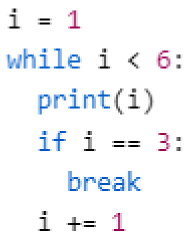


```python
day = 4
match day:
  case 1:
    print("Monday")
  case 2:
    print("Tuesday")
  case 3:
    print("Wednesday")
  case 4:
    print("Thursday")
  case 5:
    print("Friday")
  case 6:
    print("Saturday")
  case 7:
    print("Sunday")
```

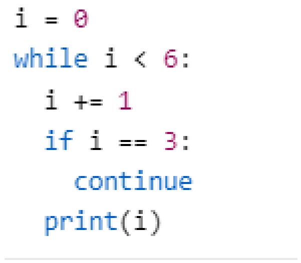
ساختارهای تکرار در پایتون



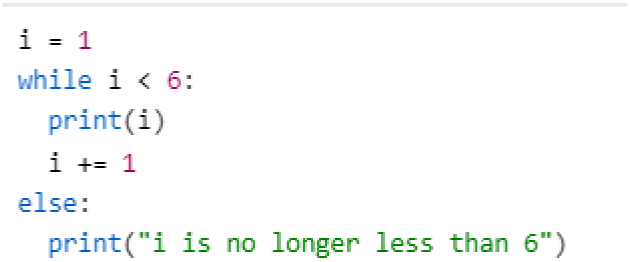
```python
i = 1
while i < 6:
  print(i)
  i += 1
```



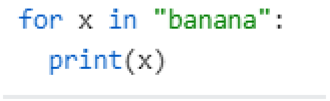
```python
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```



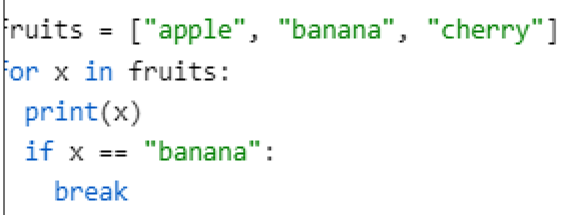
```python
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```



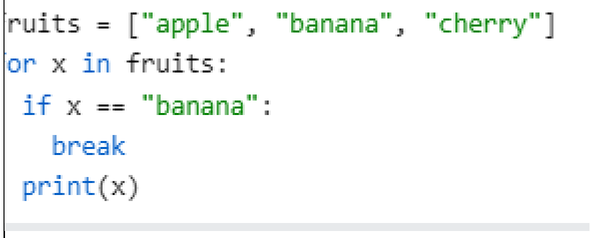
```python
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```



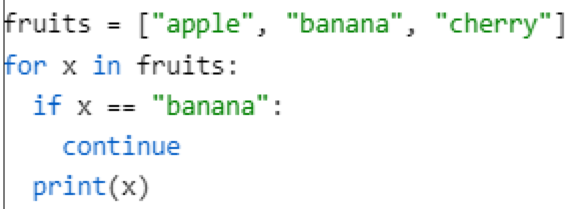
```python
for x in "banana":
  print(x)
```



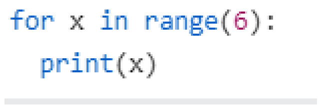
```python
ruits = ["apple", "banana", "cherry"]
or x in fruits:
  print(x)
  if x == "banana":
    break
```



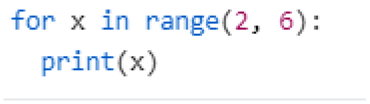
```python
ruits = ["apple", "banana", "cherry"]
or x in fruits:
  if x == "banana":
    break
  print(x)
```



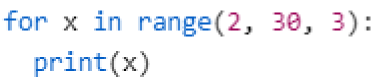
```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```



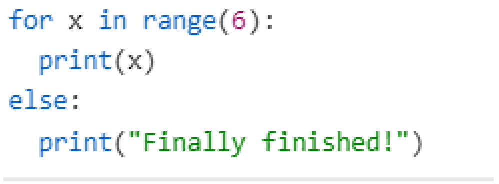
```python
for x in range(6):
  print(x)
```



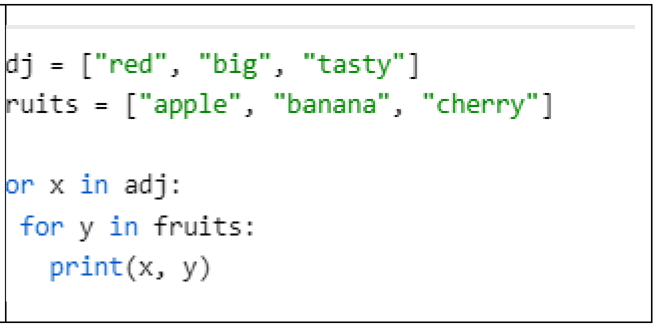
```python
for x in range(2, 6):
  print(x)
```



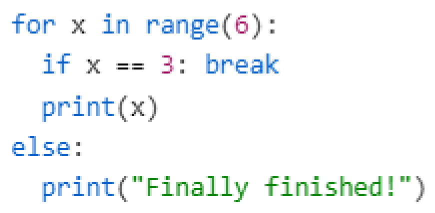
```python
for x in range(2, 30, 3):
  print(x)
```



```python
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```



```
dj = ["red", "big", "tasty"]
ruits = ["apple", "banana", "cherry"]

or x in adj:
 for y in fruits:
   print(x, y)
```



```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

سوالات برای درک بهتر ساختارها

سوال یک-برنامه بنویسید که تشخصیص دهد عدد زوج هست یا فرد.

سوال دوم-برنامه ای بنویسید که یک عدد از کاربر گرفته و مقسوم علیه های آن عدد را نشان دهد.

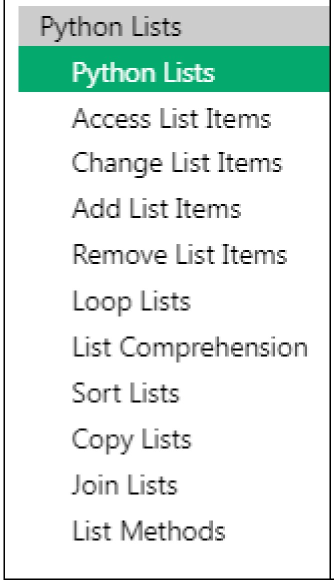
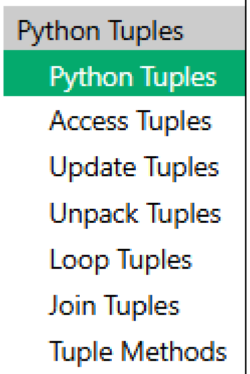
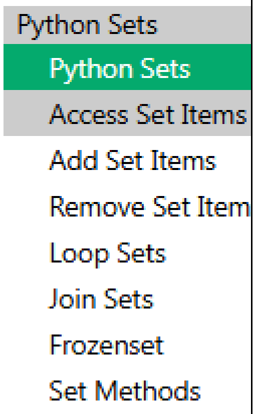
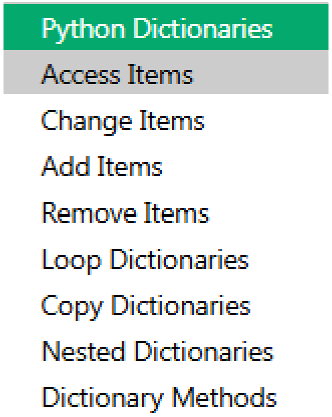
سوال سوم-برنامه ای بنویسید که یک عدد از ورودی گرفته و تشخصیص بدهد عدد کامل هست یا خیر.

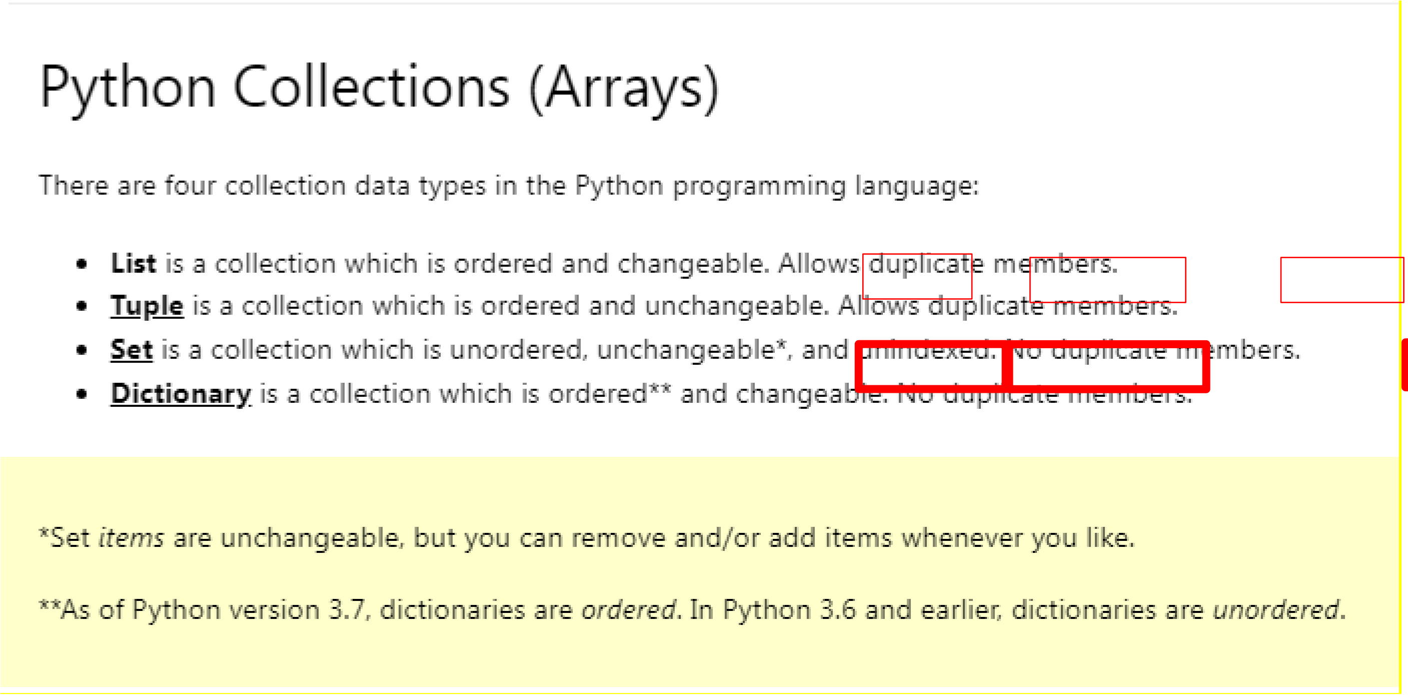
سوال چهارم-فرض امروز پنج شنبه هست صد روز بعد چند شنبه است؟

سوال پنجم-برنامه ای بنویسید که یک جدول ضرب ۵*۵ ایجاد کند.

سوال ششم-برنامه ای بنویسید که ده هزار عدد از ورودی گرفته و تشخیص بدهد عدد کامل هست یا خیر.

Python Lists-Tuples-Sets-Dictionaries

Lists	**Tuples**	**Sets**	**Dictionaries**
 Python Lists Python Lists Access List Items Change List Items Add List Items Remove List Items Loop Lists List Comprehension Sort Lists Copy Lists Join Lists List Methods	 Python Tuples Python Tuples Access Tuples Update Tuples Unpack Tuples Loop Tuples Join Tuples Tuple Methods	 Python Sets Python Sets Access Set Items Add Set Items Remove Set Item Loop Sets Join Sets Frozenset Set Methods	 Python Dictionaries Access Items Change Items Add Items Remove Items Loop Dictionaries Copy Dictionaries Nested Dictionaries Dictionary Methods



Python Collections (Arrays)

There are four collection data types in the Python programming language:

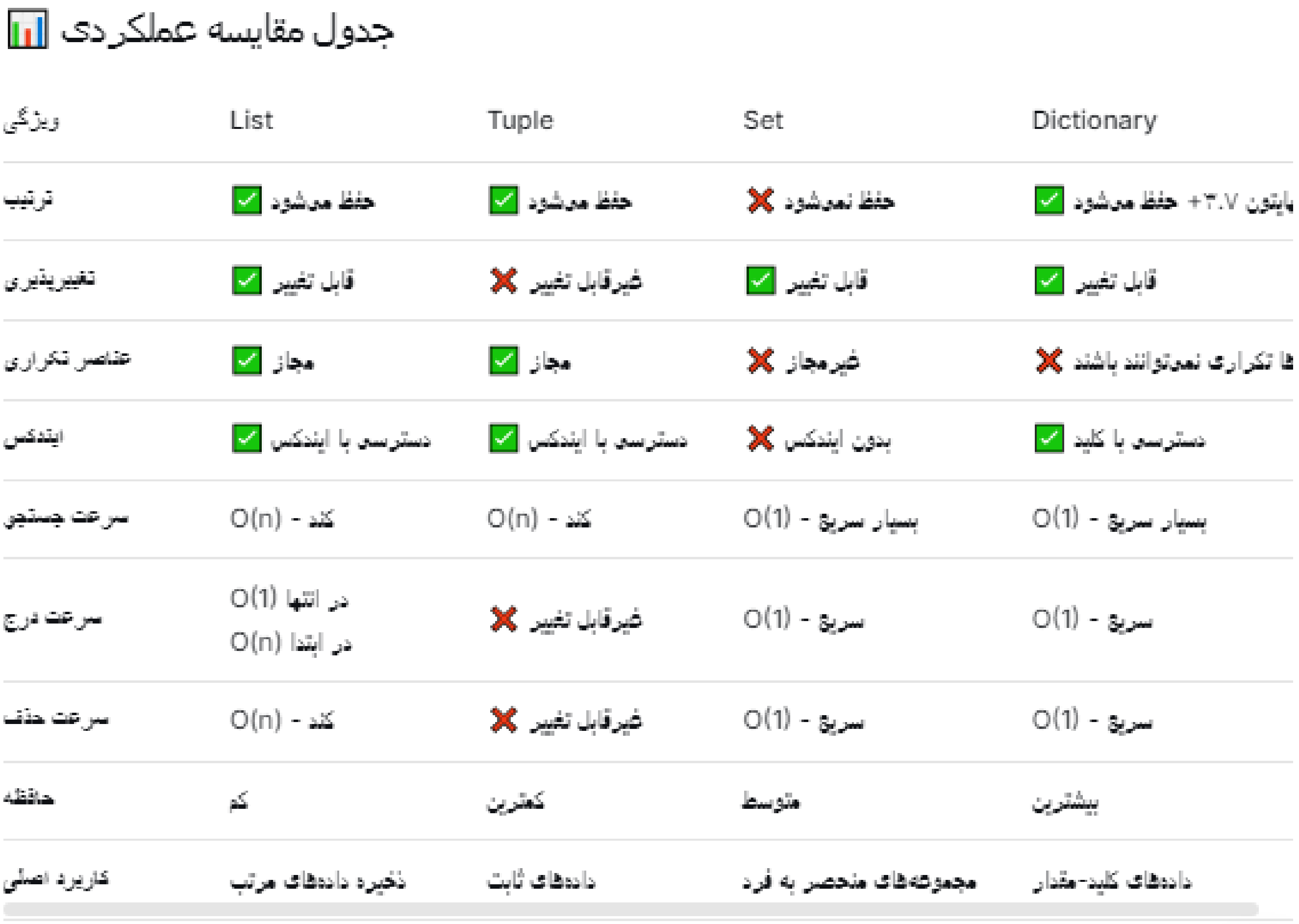
- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

*Set *items* are unchangeable, but you can remove and/or add items whenever you like.

**As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

list	**set**
ترتیب عناصر مهم است	**حذف خودکار عناصر تکراری مورد نیاز است**
ممکن است عناصر تکراری وجود داشته باشد	**عملگرهای مجموعه (اجتماع، اشتراک، تفاضل) نیاز دارید**
نیاز به دسترسی بر اساس ایندکس دارید	**جستجوی سریع نیاز است(O(۱))**
می‌خواهید داده‌ها را مرتب کنید	**ترتیب عناصر مهم نیست**

tuple	**dictionary**
وقتی داده‌ها ثابت هستند	وقتی داده‌های ساختاریافته دارید
وقتی می‌خواهید از تغییر داده‌ها جلوگیری کنید	وقتی نیاز به دسترسی سریع با کلید دارید
وقتی از داده‌ها به عنوان کلید استفاده می‌کنید	وقتی می‌خواهید داده‌ها را گروه‌بندی کنید
وقتی چندین مقدار از تابع برمی‌گردانید	وقتی نیاز به نگاشت کلید به مقدار دارید



جدول مقایسه عملکردی 📊

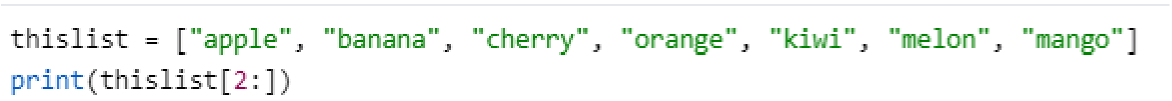
ویژگی	List	Tuple	Set	Dictionary
ترتیب	حفظ می‌شود ✅	حفظ می‌شود ✅	حفظ نمی‌شود ❌	پایتون ۳.۷+ حفظ می‌شود ✅
تغییرپذیری	قابل تغییر ✅	غیرقابل تغییر ❌	قابل تغییر ✅	قابل تغییر ✅
عناصر تکراری	مجاز ✅	مجاز ✅	غیرمجاز ❌	ها تکراری نمی‌توانند باشند ❌
ایندکس	دسترسی با ایندکس ✅	دسترسی با ایندکس ✅	بدون ایندکس ❌	دسترسی با کلید ✅
سرعت جستجو	O(n) - کند	O(n) - کند	O(1) - بسیار سریع	O(1) - بسیار سریع
سرعت درج	O(1) در انتها O(n) در ابتدا	غیرقابل تغییر ❌	O(1) - سریع	O(1) - سریع
سرعت حذف	O(n) - کند	غیرقابل تغییر ❌	O(1) - سریع	O(1) - سریع
حافظه	کم	کمترین	متوسط	بیشترین
کاربرد اصلی	ذخیره داده‌های مرتب	داده‌های ثابت	مجموعه‌های منحصر به فرد	داده‌های کلید-مقدار

Access Items list

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

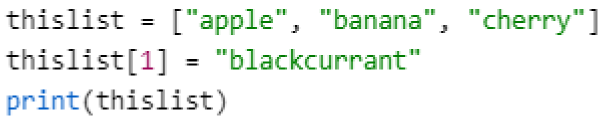
```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

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

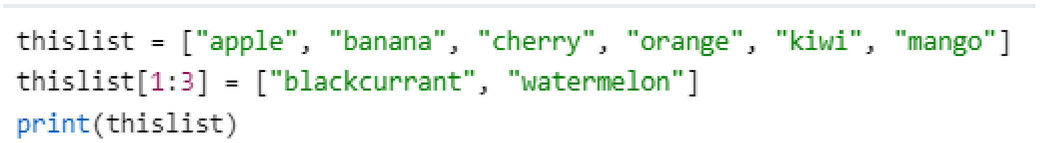


```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

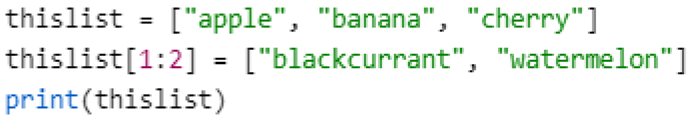
Change Item Value list



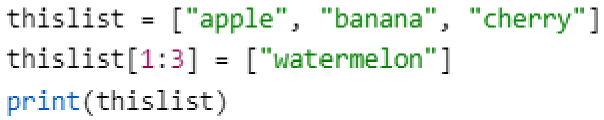
```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```



```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

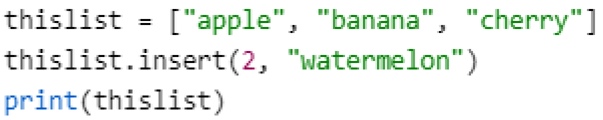


```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```



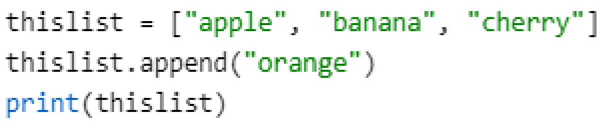
```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

Insert Items list

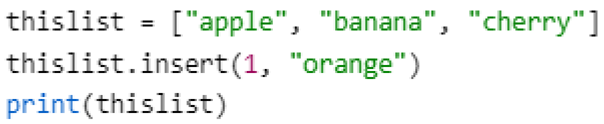


```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Append Items list

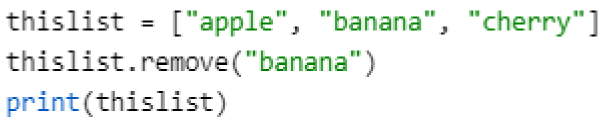


```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

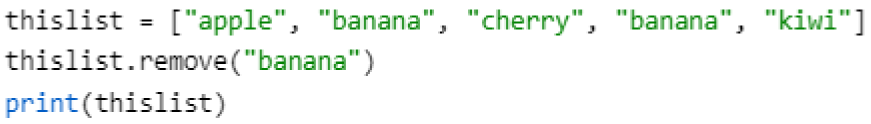


```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

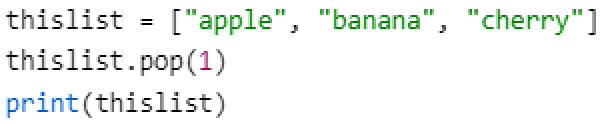
Remove List Items



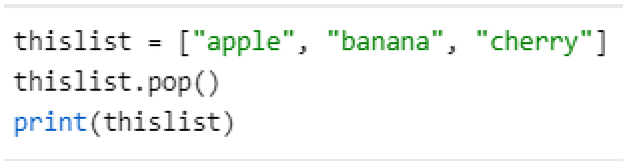
```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```



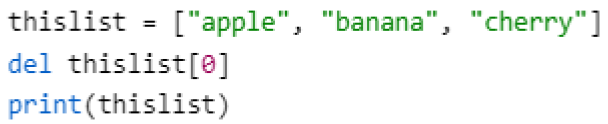
```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```



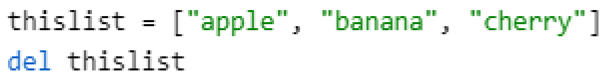
```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```



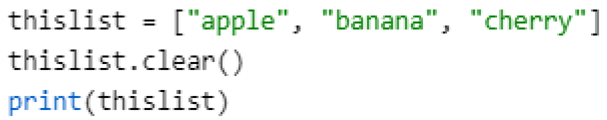
```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```



```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

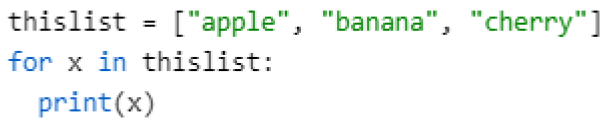


```
thislist = ["apple", "banana", "cherry"]
del thislist
```

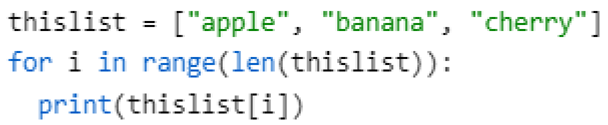


```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

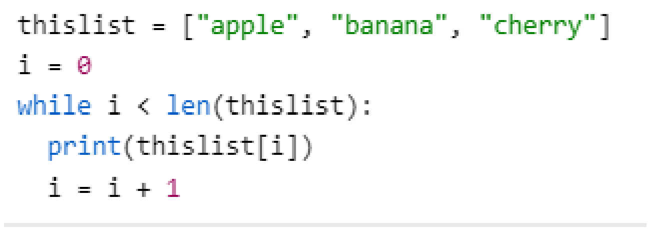
Loop Through a List



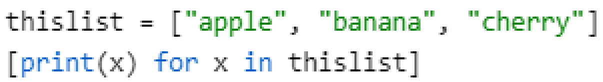
```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```



```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

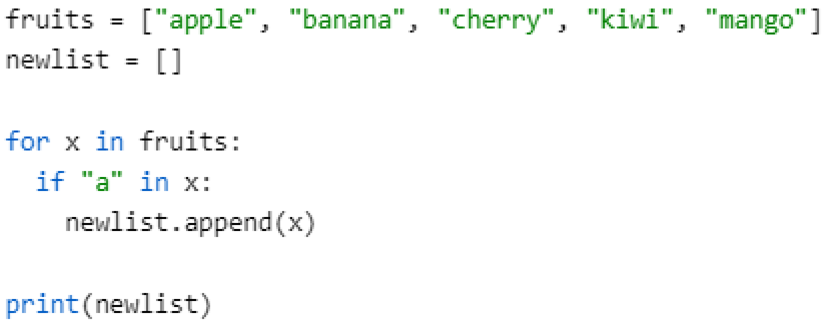


```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

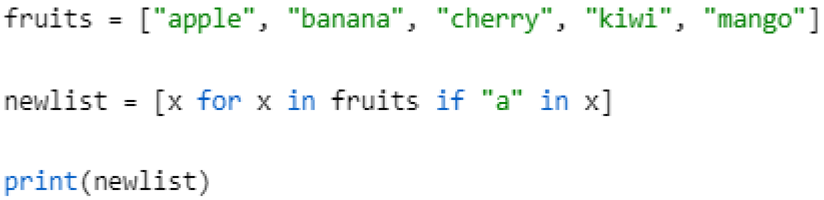


```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

List Comprehension

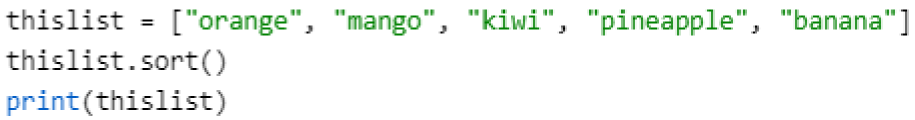


```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

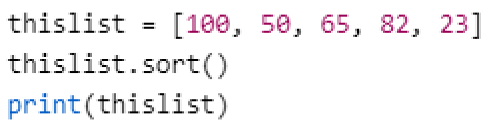


```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

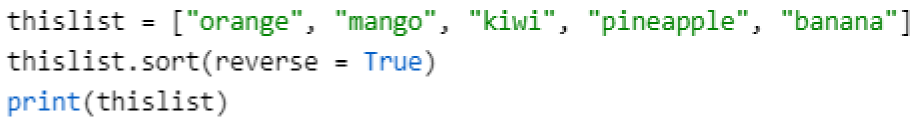
Sort Lists



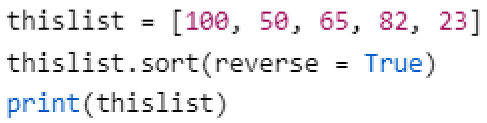
```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```



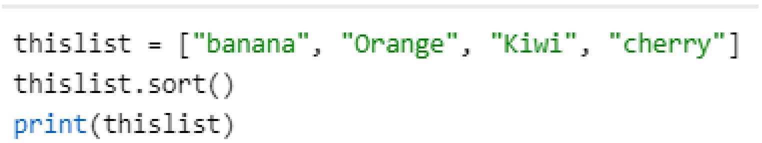
```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```



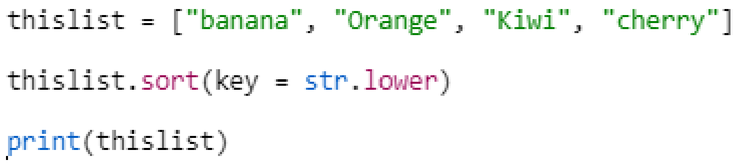
```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```



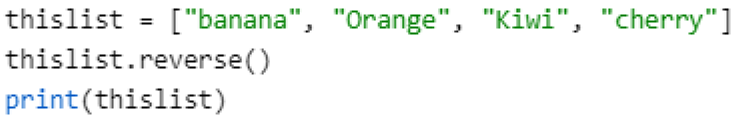
```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
```



```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
```

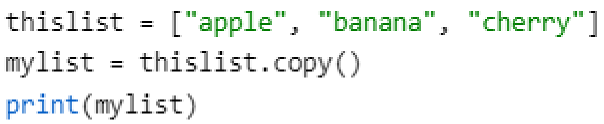


```
thislist = ["banana", "Orange", "Kiwi", "cherry"]

thislist.sort(key = str.lower)

print(thislist)
```

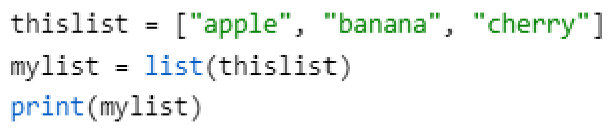


```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

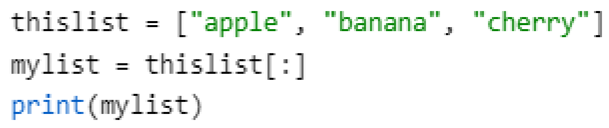
Copy a List



```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

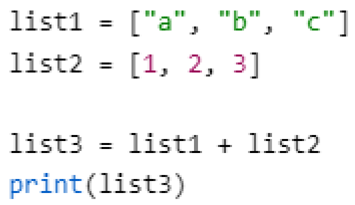


```python
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

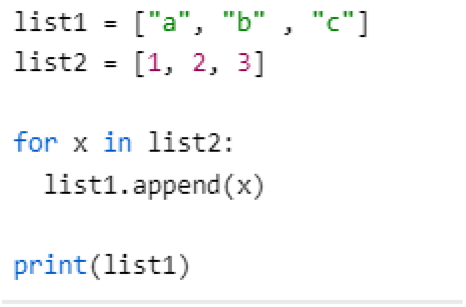


```python
thislist = ["apple", "banana", "cherry"]
mylist = thislist[:]
print(mylist)
```

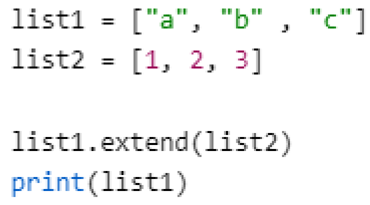
Join Two Lists



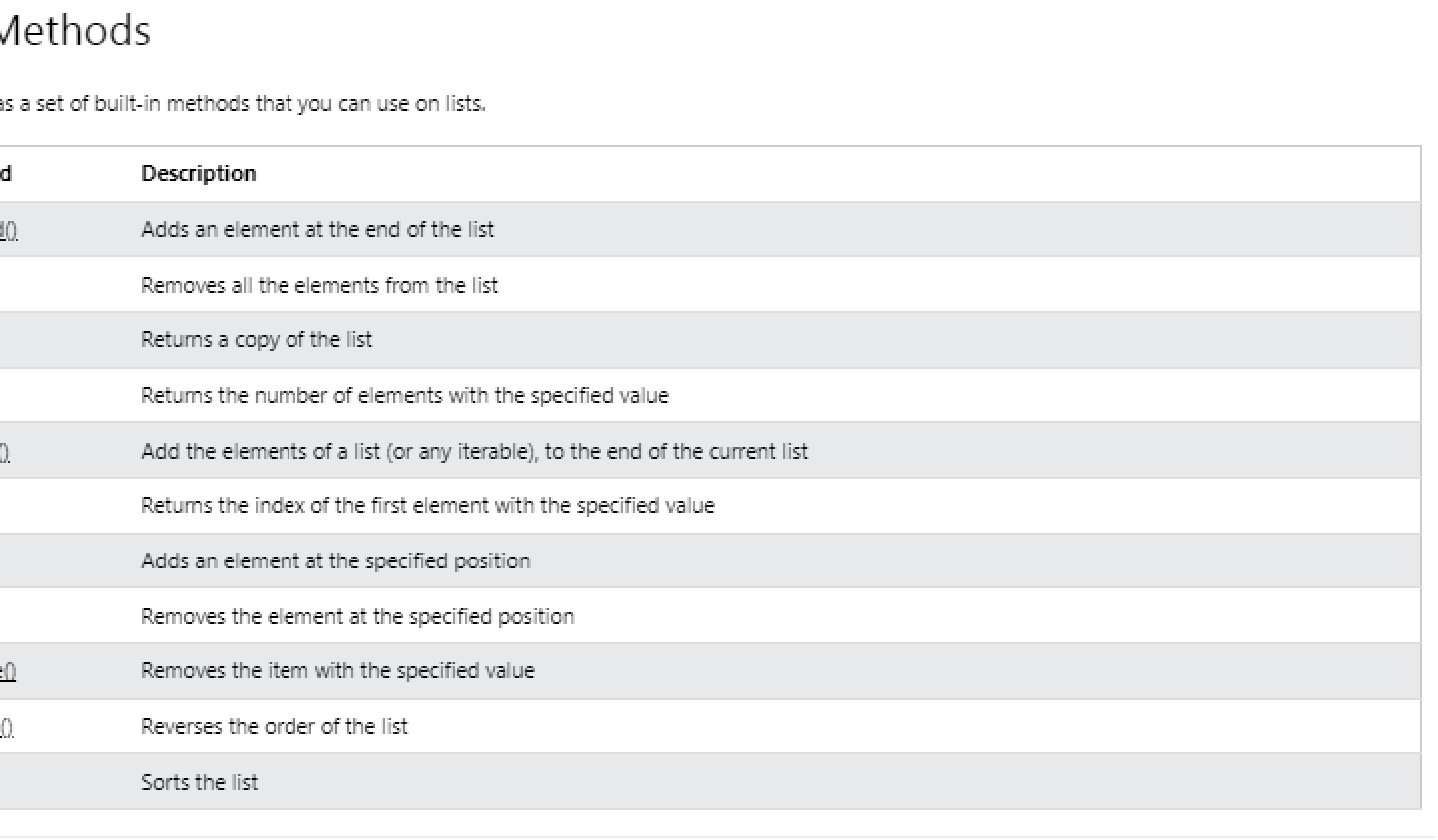
```python
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```



```python
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```



```python
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

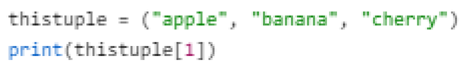


Methods

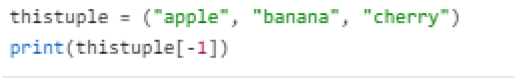
as a set of built-in methods that you can use on lists.

d	Description
d()	Adds an element at the end of the list
	Removes all the elements from the list
	Returns a copy of the list
	Returns the number of elements with the specified value
()	Add the elements of a list (or any iterable), to the end of the current list
	Returns the index of the first element with the specified value
	Adds an element at the specified position
	Removes the element at the specified position
e()	Removes the item with the specified value
()	Reverses the order of the list
	Sorts the list

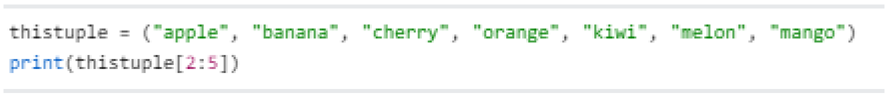
Access Tuple Items



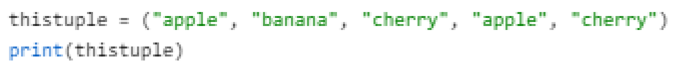
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```



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

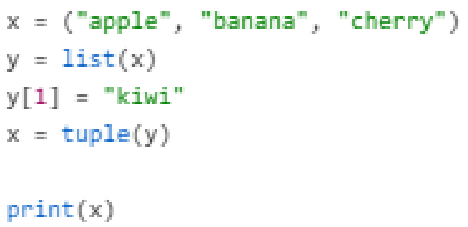


```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

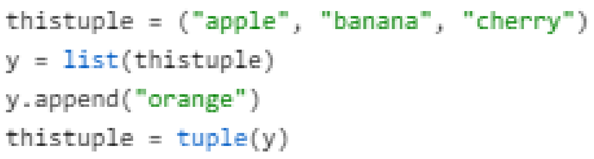


```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

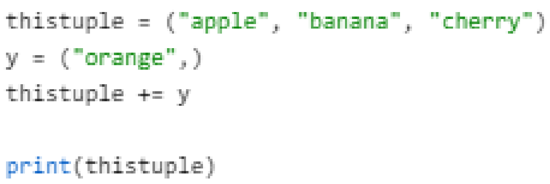
Update Tuples



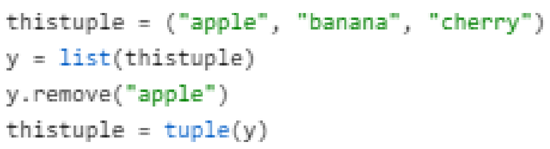
```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```



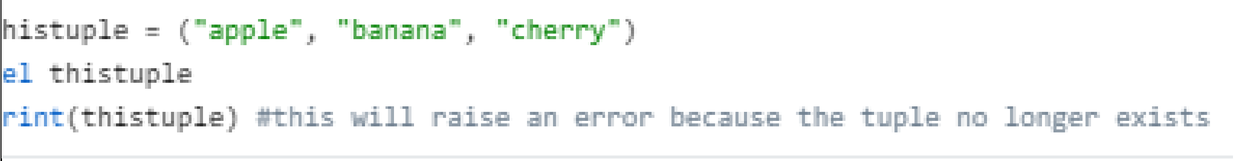
```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
```



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

print(thistuple)
```

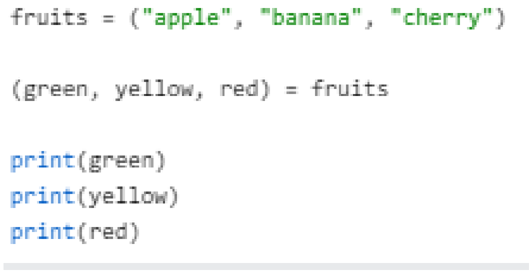


```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

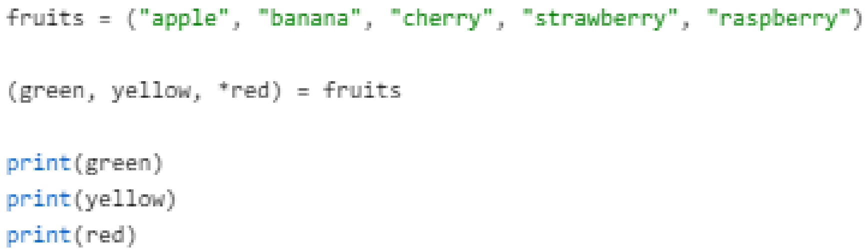


```
histuple = ("apple", "banana", "cherry")
el thistuple
rint(thistuple) #this will raise an error because the tuple no longer exists
```

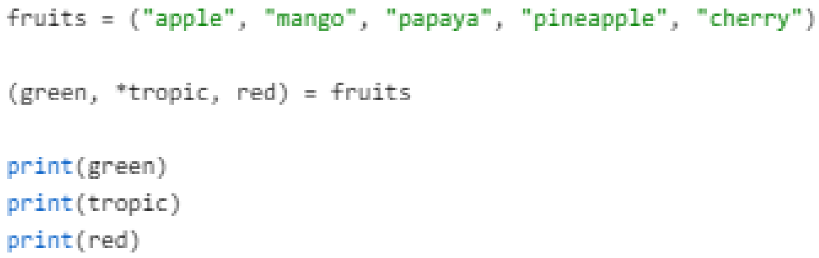
Unpacking a Tuple



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

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```



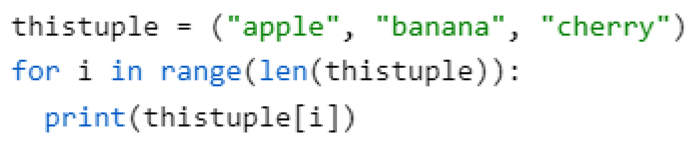
```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```



```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)
```

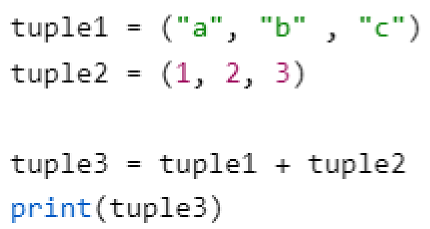
Loop Through a Tuple

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

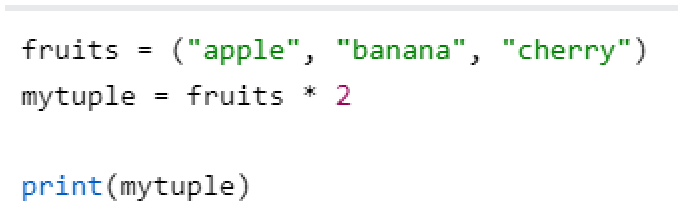


```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

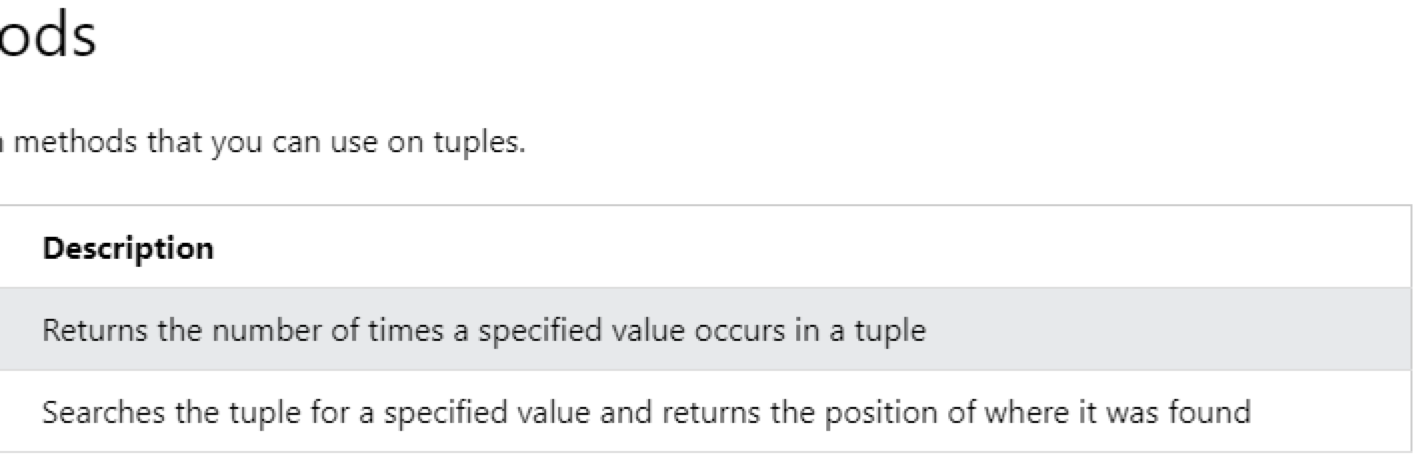
Join Two Tuples



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

tuple3 = tuple1 + tuple2
print(tuple3)
```



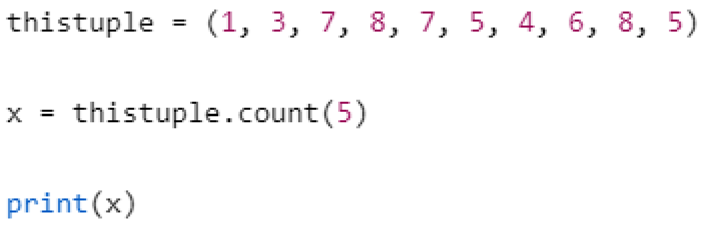
```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```



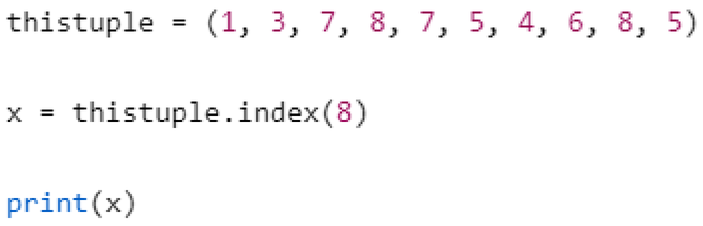
ods

methods that you can use on tuples.

Description
Returns the number of times a specified value occurs in a tuple
Searches the tuple for a specified value and returns the position of where it was found

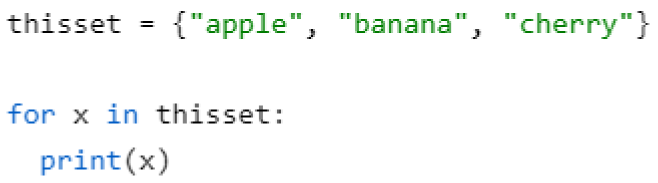


```python
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(x)
```

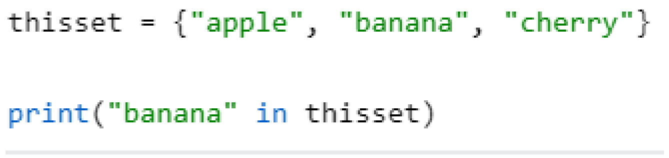


```python
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.index(8)

print(x)
```

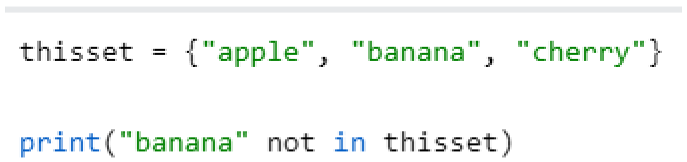
Access Set Items



```python
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

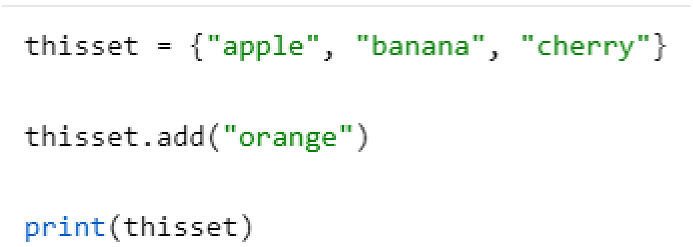


```python
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

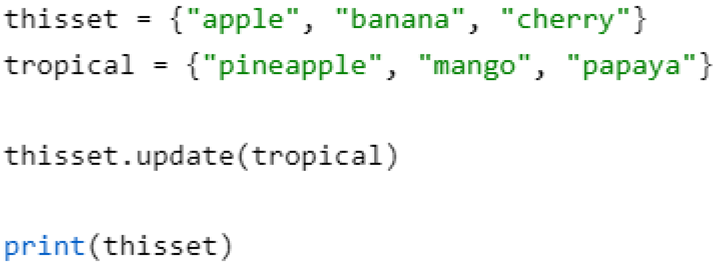


```python
thisset = {"apple", "banana", "cherry"}

print("banana" not in thisset)
```

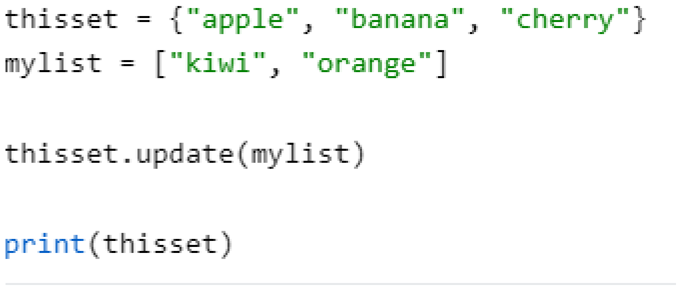
Add Set Items



```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

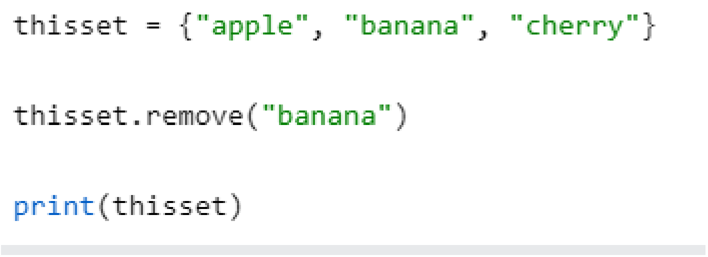


```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)
```

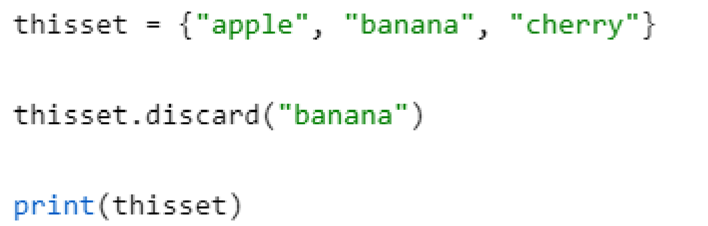


```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)
```

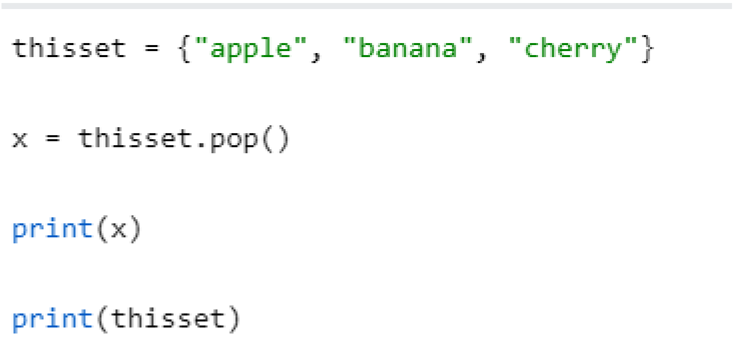
Remove Set Items



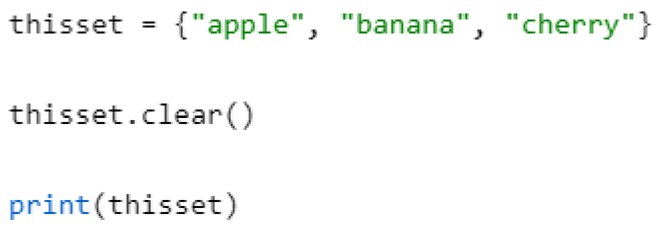
```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```



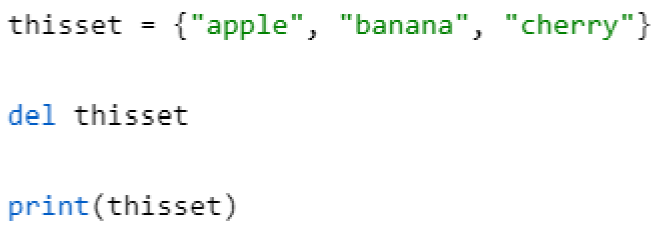
```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```



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

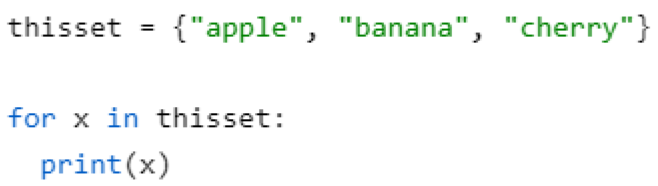


```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```



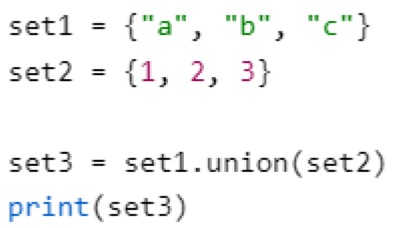
```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

Loop Sets

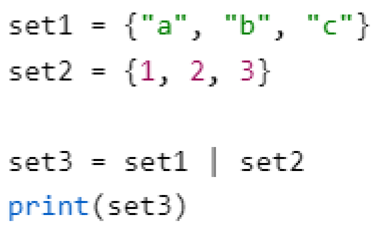


```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

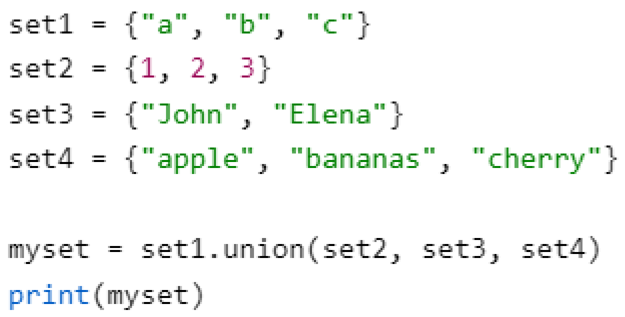
Join Sets



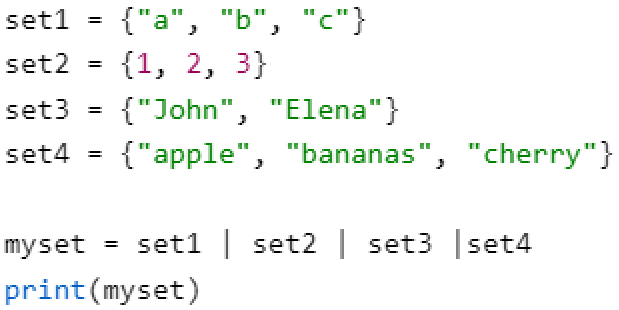
```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```



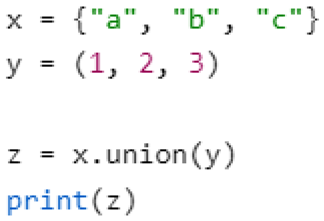
```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1 | set2
print(set3)
```



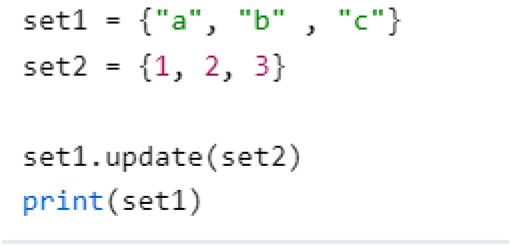
```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}

myset = set1.union(set2, set3, set4)
print(myset)
```



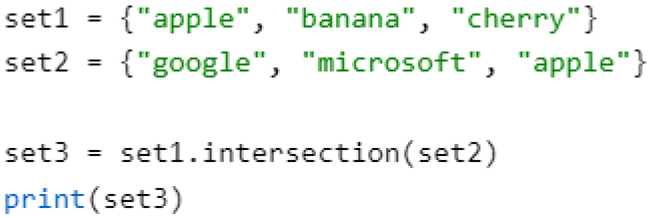
```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}

myset = set1 | set2 | set3 |set4
print(myset)
```



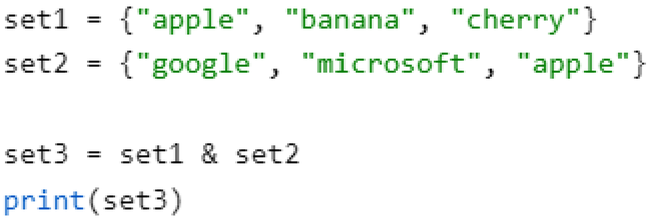
```
x = {"a", "b", "c"}
y = (1, 2, 3)

z = x.union(y)
print(z)
```



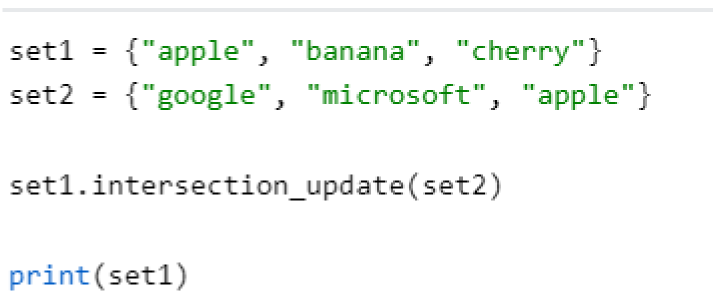
```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```



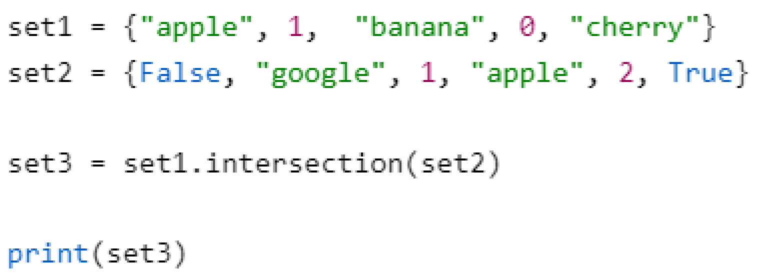
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.intersection(set2)
print(set3)
```



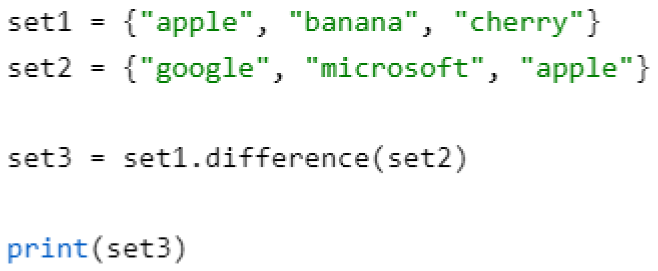
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 & set2
print(set3)
```



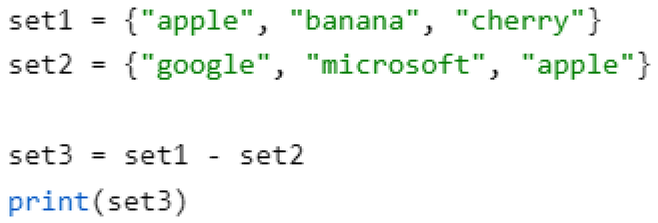
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.intersection_update(set2)

print(set1)
```



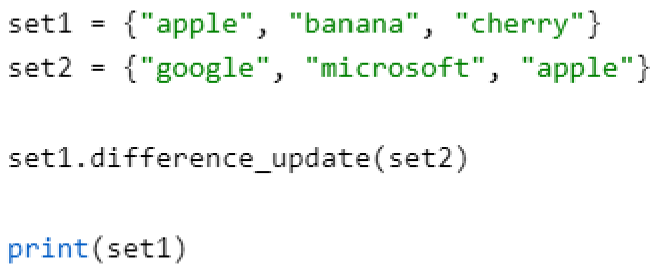
```
set1 = {"apple", 1,  "banana", 0, "cherry"}
set2 = {False, "google", 1, "apple", 2, True}

set3 = set1.intersection(set2)

print(set3)
```



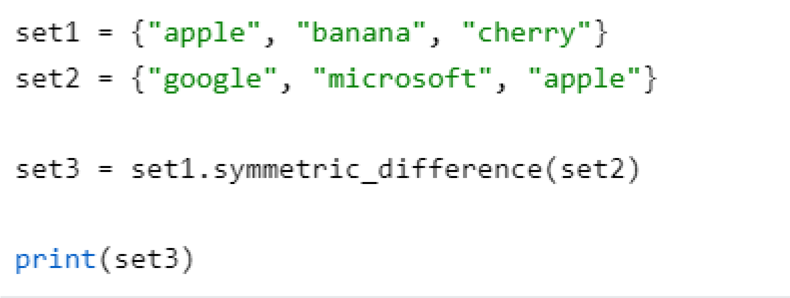
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.difference(set2)

print(set3)
```



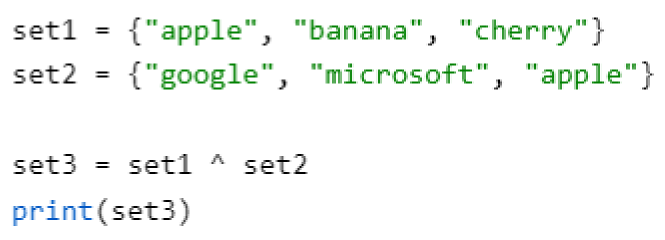
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 - set2
print(set3)
```



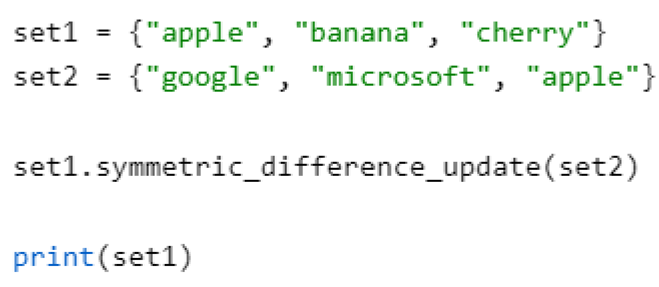
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.difference_update(set2)

print(set1)
```



```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.symmetric_difference(set2)

print(set3)
```



```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 ^ set2
print(set3)
```



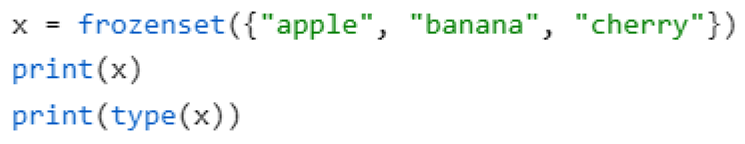
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.symmetric_difference_update(set2)

print(set1)
```

Python frozenset

frozenset is an immutable version of a set.

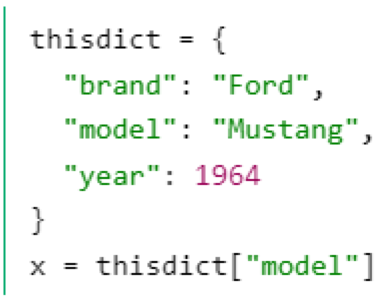
Like sets, it contains unique, unordered, unchangeable elements.

Unlike sets, elements cannot be added or removed from a frozenset.

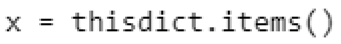


```
x = frozenset({"apple", "banana", "cherry"})
print(x)
print(type(x))
```

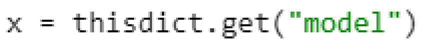
Access **Dictionary** Items



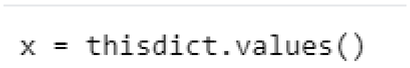
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```



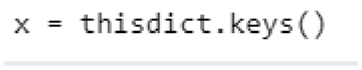
```
x = thisdict.items()
```



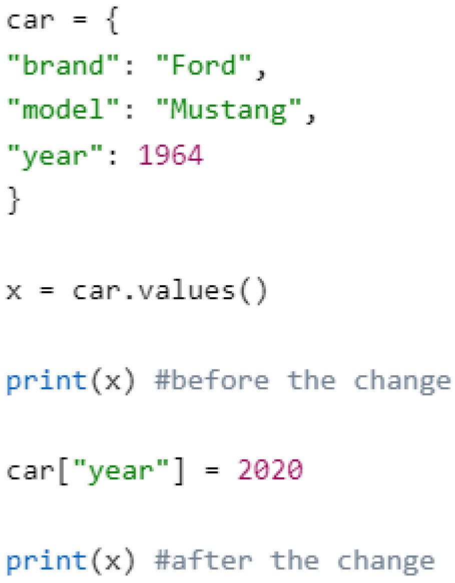
```
x = thisdict.get("model")
```



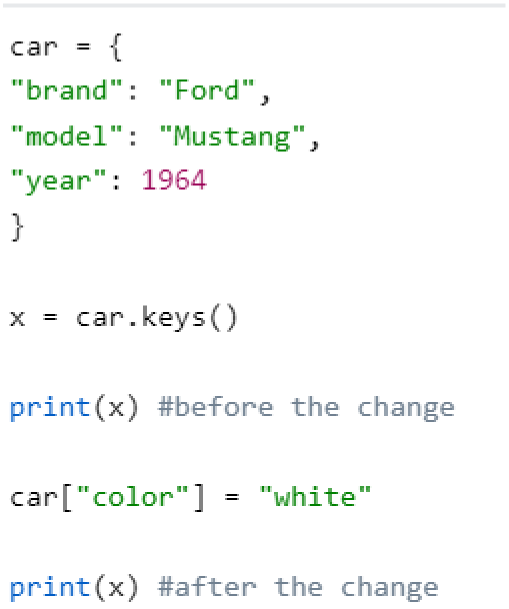
```
x = thisdict.values()
```



```
x = thisdict.keys()
```

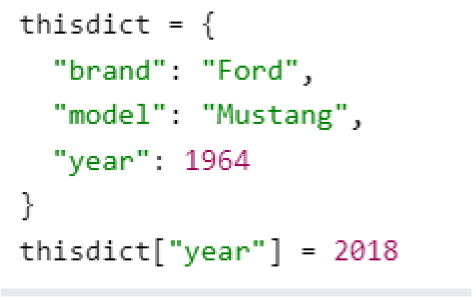


```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

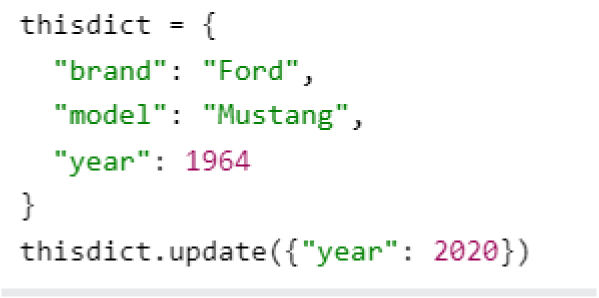


```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

Change Dictionary Item

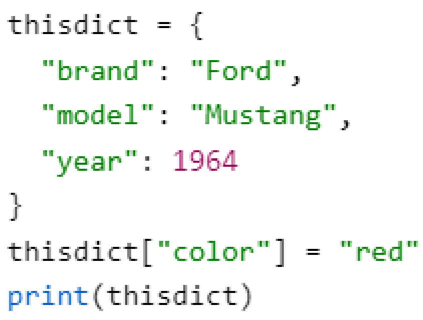


```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

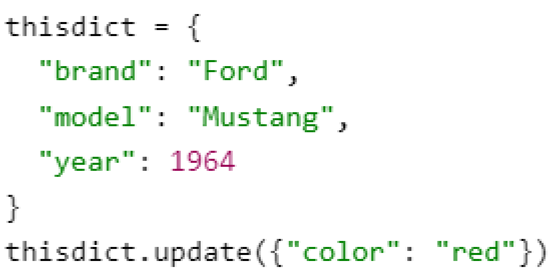


```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})
```

Add Dictionary Items

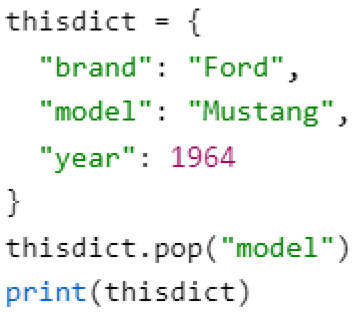


```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

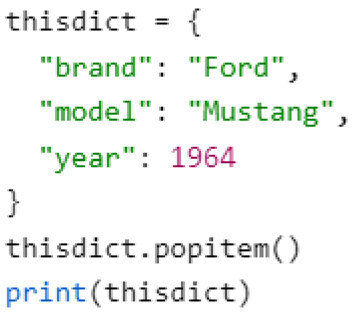


```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"color": "red"})
```

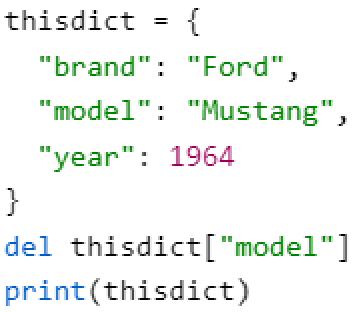
Remove Dictionary Items



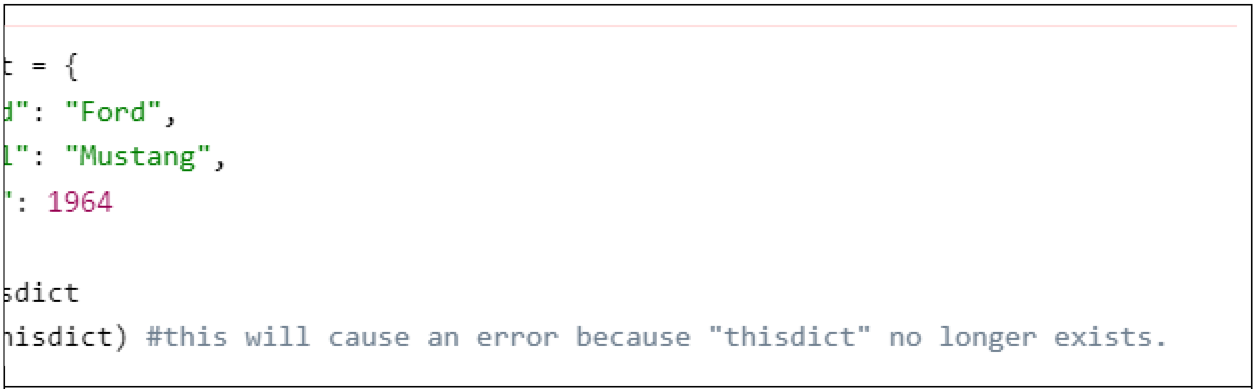
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```



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



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



```
t = {
d": "Ford",
l": "Mustang",
': 1964

sdict
isdict) #this will cause an error because "thisdict" no longer exists.
```

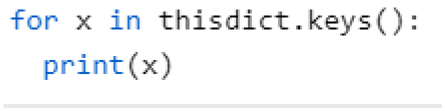
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

Loop Dictionaries

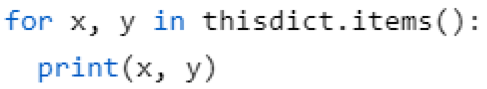
```
for x in thisdict:
  print(x)
```

```
for x in thisdict:
  print(thisdict[x])
```

```
for x in thisdict.values():
  print(x)
```

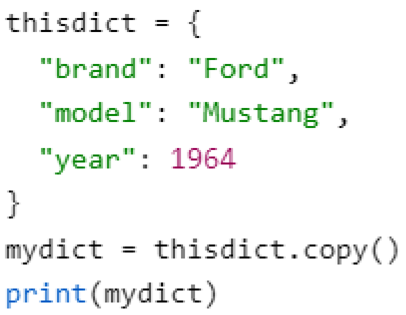


```python
for x in thisdict.keys():
  print(x)
```

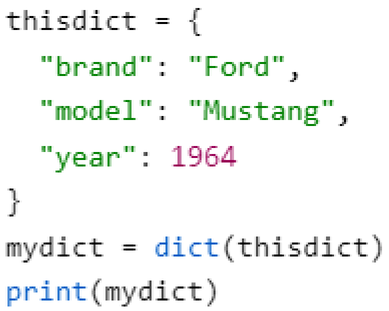


```python
for x, y in thisdict.items():
  print(x, y)
```

Copy a Dictionary



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



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

Nested Dictionaries

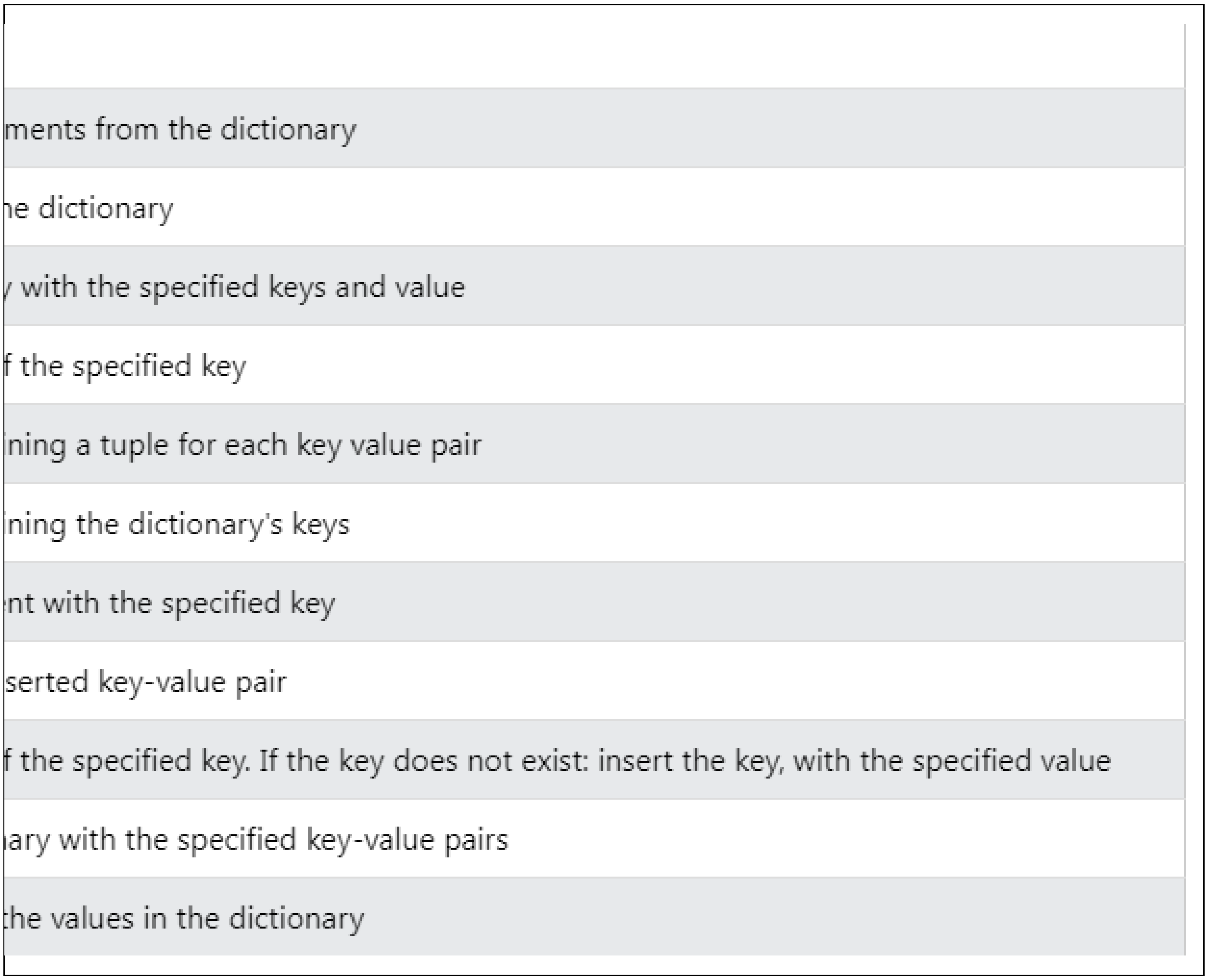
```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

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

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

```
print(myfamily["child2"]["name"])
```

```
for x, obj in myfamily.items():
  print(x)

  for y in obj:
    print(y + ':', obj[y])
```

Dictionary Methods



ments from the dictionary
ne dictionary
y with the specified keys and value
f the specified key
ning a tuple for each key value pair
ning the dictionary's keys
nt with the specified key
serted key-value pair
f the specified key. If the key does not exist: insert the key, with the specified value
ary with the specified key-value pairs
the values in the dictionary