

Python Basics II

List functions

Python offers the following list functions:

1. `sort()`: Sorts the list in ascending order.
2. `type(list)`: It returns the class type of an object.
3. `append()`: Adds a single element to a list.
4. `extend()`: Adds multiple elements to a list.
5. `index()`: Returns the first appearance of the specified value.
6. `max(list)`: It returns an item from the list with max value.
7. `min(list)`: It returns an item from the list with min value.
8. `len(list)`: It gives the total length of the list.
9. `list(seq)`: Converts a tuple into a list.
10. `cmp(list1, list2)`: It compares elements of both lists list1 and list2.

`sort()` method

The `sort()` method is a built-in Python method that, by default, sorts the list in ascending order. However, you can modify the order from ascending to descending by specifying the sorting criteria.

Example

Let's say you want to sort the element in prices in ascending order. You would type prices followed by a `.` (period) followed by the method name, i.e., `sort` including the parentheses.



```
prices = [23.5, 4, 2.5, 9.0 ]  
prices.sort()  
print(prices)
```

type() method

For the type() function, it returns the class type of an object.



```
list1 = [2, 9.7, "word", True]  
print(type(list1))
```

append() method

The append() method will add certain content you enter to the end of the elements you select.



```
weeks = ["Monday", "Tuesday", "Wednesday"]  
weeks.append("Thursday")  
print(weeks)
```

extend() method

The extend() method increases the length of the list by the number of elements that are provided to the method, so if you want to add multiple elements to the list, you can use this method.



```
weeks = ["Monday", "Tuesday", "Wednesday"]  
weeks.extend(["Thursday", "Friday"])  
print(weeks)
```

index() method

The index() method returns the first appearance of the specified value.



```
weeks = ["Monday", "Tuesday", "Wednesday"]  
print(weeks.index("Monday"))
```

max() function

The max() function will return the highest value of the inputted values.



```
Ages = [3,5,7]  
print("{} is the highest age in the  
group".format(max(Ages)))
```

min() function


The min() function will return the lowest value of the inputted values.



```
Ages = [3,5,7]  
print("{} is the least age in the  
group".format(min(Ages)))
```

len() function

The len() function shows the number of elements in a list.



```
weeks = ["Monday", "Tuesday", "Wednesday"]  
print(len(weeks))  
list() function
```

The list() function takes an iterable construct and turns it into a list.

A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. It contains four lines of Python code:

```
string1 = "python"
list1 = list(string1)
print(list1)
print(list1[4])
```

The code demonstrates converting a string to a list and accessing an element by index. The index 4 is highlighted in red in the original image.

To learn more about list methods and functions, check the documentation below

[5. Data Structures — Python 3.9.0 documentation](#)

Slicing list

Tuples

Tuples are an ordered sequence of items, just like lists. The main difference between tuples and lists is that tuples cannot be changed (immutable) unlike lists which can be (mutable). When you try to change a value in a tuple it will throw a type error. Similarly, you cannot add or remove values either.

There are two ways to declare or initialize a tuple.

To create a tuple you use parentheses instead of square brackets, and just provide it the values separated by commas.

1. You can initialize an empty tuple by having () with no values in them.

```
empty_tuple = ()
```

2. You can also initialize an empty tuple by using the tuple function.

```
empty_tuple = tuple()
```

Accessing elements of a tuple is just like accessing lists, you use indexes. Let's take a look at an example;

```
scores = (2.4, 6.9, 3.6, 8.3, 2.0, 4.5)
```

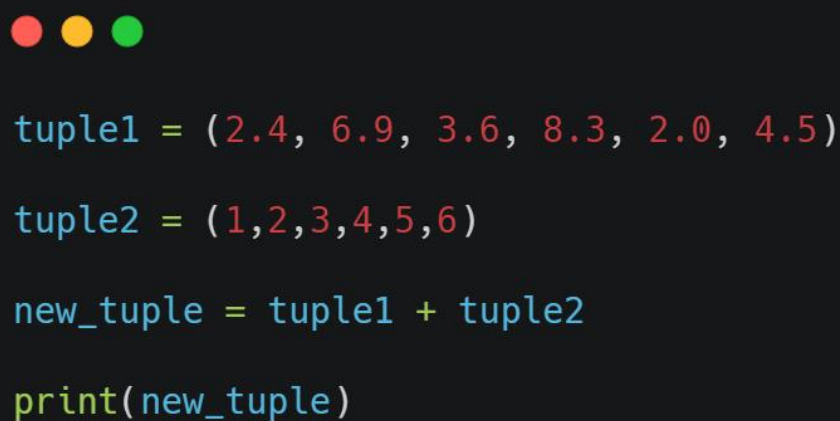
```
print(scores[4])
```

Note that you cannot add items to a tuple, it will throw an error. Let's try this



The screenshot shows a Jupyter Notebook interface. In the first cell, the code `scores = (2.4, 6.9, 3.6, 8.3, 2.0, 4.5)` and `print(scores[4])` is executed, resulting in the output `2.0`. In the second cell, the code `scores[6] = 3` is entered, which triggers a `TypeError: 'tuple' object does not support item assignment`. The error message is displayed in a red box with a traceback showing the error occurred in the current cell.

You can add two tuples together to create a new tuple,



```
tuple1 = (2.4, 6.9, 3.6, 8.3, 2.0, 4.5)
tuple2 = (1, 2, 3, 4, 5, 6)
new_tuple = tuple1 + tuple2
print(new_tuple)
```

Note:

You can't add elements to a tuple because of their immutable property.

There's no `append()` or `extend()` method for tuples, You can't remove elements from a tuple.

Also because of their immutability, Tuples have no `remove()` or `pop()` method,

You can find elements in a tuple, since this doesn't change the tuple.

You can also use the `in` operator to check if an element exists in the tuple.

You can delete a tuple by using the `del` keyword followed by the name of the tuple.

```
del tuple1
```

Sets


Python set is an unordered collection of unique items. They are commonly used for computing mathematical operations such as union, intersection, difference, and symmetric difference.

The important properties of Python sets are as follows:

1. Sets are unordered – Items stored in a set aren't kept in any particular order.
2. Set items are unique – Duplicate items are not allowed.
3. Sets are unindexed – You cannot access set items by referring to an index.
4. Sets are changeable (mutable) – They can be changed in place, can grow and shrink on demand.

You can create a set by placing a comma-separated sequence of items in curly brackets `{ }`.

Note that if you create a set with duplicate items, the duplicate will be removed automatically during the set creation.



```
colors = {"red", "black", "white", "green", "black"}  
print(colors)
```

You can also create a set using the `set` function, this also great for converting sequence/iterable to set. For example; let's convert a list to a set.



```
colors = set(["red", "black", "white", "green",  
"black"])  
  
print(colors)
```

This is also a great way to remove duplicates from an iterable, you can simply convert the iterable to a set which removes the duplicates, and then you convert it back to the iterable type.



```
colors = set(["red", "black", "white", "green",  
"black"])  
  
colors = list(colors)  
  
print(colors)
```

Adding items to a set



```
# using the add method

colors = set(["red", "black", "white", "green",
"black"])

colors.add("Blue")

print(colors)
```

To add an element to a set, you use the add method; and to add multiple elements you use the update method.

Remove Items from a Set

To remove a single item from a set, use remove() or discard() method.



```
# using the remove method

colors = set(["red", "black", "white", "green",
"black"])

colors.remove('red')

print(colors)
```

Find Set Size

To find how many items a set has, use len() method.



```
colors = set(["red", "black", "white", "green"])  
print(len(colors))
```

Set Operations

Sets are commonly used for computing mathematical operations such as intersection, union, difference, and symmetric difference.

Set Union

You can perform union on two or more sets using `union()` method or `|` operator.



```
A = {'red', 'green', 'blue'}  
B = {'yellow', 'red', 'orange'}  
  
# by operator  
print(A | B)  
  
# Prints {'blue', 'green', 'yellow', 'orange', 'red'}  
  
# by method  
print(A.union(B))  
  
# Prints {'blue', 'green', 'yellow', 'orange', 'red'}
```

Set Intersection

You can perform intersection on two or more sets using `intersection()` method or `&` operator.



```
A = {'red', 'green', 'blue'}  
B = {'yellow', 'red', 'orange'}  
  
# by operator  
  
print(A & B)  
  
# Prints {'red'}  
  
# by method  
  
print(A.intersection(B))  
  
# Prints {'red'}
```

Set Difference

You can compute the difference between two or more sets using `difference()` method or `-` operator.



```
A = {'red', 'green', 'blue'}  
B = {'yellow', 'red', 'orange'}  
  
# by operator  
  
print(A - B)  
  
# Prints {'blue', 'green'}  
  
# by method  
  
print(A.difference(B))  
  
# Prints {'blue', 'green'}
```

You can compute symmetric difference between two or more sets using `symmetric_difference()` method or `^` operator.

```

A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}

# by operator

print(A ^ B)

# Prints {'orange', 'blue', 'green', 'yellow'}

# by method

print(A.symmetric_difference(B))

# Prints {'orange', 'blue', 'green', 'yellow'}
```

Other Set Operations

Below is a list of all set operations available in Python.

Method & Their Description

union() Return a new set containing the union of two or more sets

update() Modify this set with the union of this set and other sets

intersection() Returns a new set which is the intersection of two or more sets

intersection_update() Removes the items from this set that are not present in other sets

difference() Returns a new set containing the difference between two or more sets

difference_update() Removes the items from this set that are also included in another set

symmetric_difference() Returns a new set with the symmetric differences of two or more sets

symmetric_difference_update() Modify this set with the symmetric difference of this set and other set

isdisjoint() Determines whether or not two sets have any elements in common

issubset() Determines whether one set is a subset of the other

issuperset()Determines whether one set is a superset of the other