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.

Python Lists

Lists are used to store multiple items in a single variable. Lists are created using square brackets:[]

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

List Items

List items are ordered, changeable, and allow duplicate values.
List items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
If you add new items to a list, the new items will be placed at the end of the list.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

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

List Length

To determine how many items a list has, use the `len()` function:

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

List Items - Data Types

List items can be of any data type:

```
list1 = ["apple", "banana", "cherry"] #STRING
list2 = [1, 5, 7, 9, 3]                #INT
list3 = [True, False, False]           #BOOLEAN
```

A list with strings, integers and boolean values:

```
list1 = ["abc", 34, True, 40, "male"]
```

type()

From Python's perspective, lists are defined as objects with the data type 'list':

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

The list() Constructor

It is also possible to use the `list()` constructor when creating a new list.

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

Access Items

List items are indexed and you can access them by referring to the index number:

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

# For Last Item
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])

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

# Negative Indexing
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])


# Check if "apple" is present in the list:
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Change List Items

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

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


# Insert value without changing
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Append Items

To add an item to the end of the list, use the `append()` method:

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

Insert Items

To insert a list item at a specified index, use the `insert()` method.

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

Extend List

To append elements from *another list* to the current list, use the `extend()` method.

```
# Add the elements of tropical to thislist:
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Add elements of a tuple to a list:

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Remove Specified Item

The `remove()` method removes the specified item.

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

Remove Specified Index

The `pop()` method removes the specified index.

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

The `del` keyword also removes the specified index:

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

The `del` keyword can also delete the list completely.

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

Clear the List

The `clear()` method empties the list.

The list still remains, but it has no content.

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

Loop Through a List

You can loop through the list items by using a `for` loop:

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

Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.
Use the `range()` and `len()` functions to create a suitable iterable.

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

The iterable created in the example above is `[0, 1, 2]`.

Using a While Loop

You can loop through the list items by using a `while` loop.
Use the `len()` function to determine the length of the list, then start at 0 and loop your way through the list items by referring to their indexes.
Remember to increase the index by 1 after each iteration.

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

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Without list comprehension you will have to write a `for` statement with a conditional test inside:

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

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

print(newlist)
```

With list comprehension you can do all that with only one line of code:

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

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

print(newlist)
```

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

The return value is a new list, leaving the old list unchanged.

Condition

The *condition* is like a filter that only accepts the items that evaluate to `True`.

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

newlist = [x for x in fruits if x != "apple"]

print(newlist)
```

The condition `if x != "apple"` will return `True` for all elements other than "apple", making the new list contain all fruits except "apple".
The *condition* is optional and can be omitted:

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

newlist = [x for x in fruits]

print(newlist)
```

Python Tuples

Tuples are used to store multiple items in a single variable. A tuple is a collection which is ordered and unchangeable.

Tuples are written with round ’()’ brackets.

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

Tuple items are ordered, unchangeable, and allow duplicate values.
Tuple items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.

Allow Duplicates

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

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", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

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

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

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
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)
```

Tuple Methods

COUNT()

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

INDEX()

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

Dictionary

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

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

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

Dictionaries cannot have two items with the same key:

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

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Get the value of the "model" key:

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

 # There is also a method called `get()` that will give you the same result:

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

Get Keys

The `keys()` method will return a list of all the keys in the dictionary.

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

x = thisdict.keys()

print(x)
```

Add a new item to the original dictionary, and see that the keys list gets updated as well:

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

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

Get Values

The `values()` method will return a list of all the values in the dictionary.

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

x = thisdict.values()

print(x)
```

Make a change in the original dictionary, and see that the values list gets updated as well:

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

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

Get Items

The `items()` method will return each item in a dictionary, as tuples in a list.

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

x = thisdict.items()

print(x)
```

Make a change in the original dictionary, and see that the items list gets updated as well:

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

x = car.items()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```