Boolean

Data with one of two built-in values `True` or `False`. Notice that 'T' and 'F' are capital. `true` and `false` are not valid Booleans and Python will throw an error for them.

Ex.

```
print (10 > 9)
print(10 == 9)
print(10 < 9)
```

Python Set

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

```
a = {5, 2, 3, 1, 4}

# Printing set variable
print("a = ", a)

# Data type of variable a
print(type(a))
```

Output

```
a = {1, 2, 3, 4, 5}
<class 'set'>
```

We can perform set operations like union, intersection on two sets. Sets have unique values. They eliminate duplicates.

```
a = {1, 2, 2, 3,3 ,3}
print (a)
```

Output

```
{1, 2, 3}
```

Since, set are unordered collection, indexing has no meaning. Hence, the slicing operator [] does not work.

Change Items

Once a set is created, you cannot change its items, but you can add new items.

Add Items

To add one item to a set use the add() method.

To add more than one item to a set use the update() method.

Example

Add an item to a set, using the add() method:

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

thisset.add("orange")
```

```
print(thisset)
```

Update()

Example

Add multiple items to a set, using the update() method:

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

thisset.update(["orange", "mango", "grapes"])

print(thisset)
```

Get the Length of a Set

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

Example

Get the number of items in a set:

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

print(len(thisset))
```

Remove Item

To remove an item in a set, use the remove(), or the discard() method.

Example

Remove "banana" by using the remove() method:

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

thisset.remove("banana")

print(thisset)
```

Clear ()

The clear() method empties the set:

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

thisset.clear()

print(thisset)
```

Tuple

A **tuple** is a container which holds a series of comma-separated values between parentheses. A tuple is similar to a list. Since, tuples are quite similar to **lists** , both of them are used in similar situations as well. The only the difference is that list is enclosed between square bracket, tuple between parenthesis and List have **mutable objects** whereas Tuple have immutable objects.

Example

```
Tuple1 = (1,2,"Hello",3.14,"world")
print(Tuple1)
print(Tuple1 [3]) //Accessing Tuple Item
Tuple2 = (5,"six")
print(Tuple1 + Tuple2)
```

Output

```
(1, 2, 'Hello', 3.14, 'world')
3.14
(1, 2, 'Hello', 3.14, 'world', 5, 'six')
```

Negative indexing means beginning from the end, `-1` refers to the last item, `-2` refers to the second last item etc.

Example

Print the last item of the tuple:

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

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

Example

Return the third, fourth, and fifth item:

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

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

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

Loop Through a Tuple

You can loop through the tuple items by using a `for` loop.

Example

Iterate through the items and print the values:

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

```
print(x)
```

Tuple Length

To determine how many items a tuple has, use the len() method:

Example

Print the number of items in the tuple:

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

Add Items

Once a tuple is created, you cannot add items to it. Tuples are **unchangeable**.

Example

You cannot add items to a tuple:

```
thistuple = ("apple", "banana", "cherry")
thistuple[3] = "orange" # This will raise an error
print(thistuple)
```

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

Example

One item tuple, remember the commma:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Remove Items

Note: You cannot remove items in a tuple.

Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:

Example

The del keyword can delete the tuple completely:

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

Join Two Tuples

To join two or more tuples you can use the + operator:

Example

Join two tuples:

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

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

The tuple() Constructor

It is also possible to use the `tuple()` constructor to make a tuple.

Example

Using the tuple() method to make a tuple:

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

Tuple Methods

Python has two built-in methods that you can use on tuples.

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

Example

Return the number of times the value 5 appears in the tuple:

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

x = thistuple.count(5)

print(x)
```

Example

Search for the first occurrence of the value 8, and return its position:

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

x = thistuple.index(8)

print(x)
```

Dictionary

Pyhton **Dictionaries** allow you store and retrieve related information in a way that means something both to humans and computers. Dictionaries are non-ordered and contain **"keys"** and **"values"** . Each key is unique and the values can be just about anything, but usually they are string, int, or float, or a list of these things. Like lists dictionaries can easily be changed, can be shrunk and grown ad libitum at run time. Dictionaries don't support the sequence operation of the

sequence data types like strings, tuples and lists. Dictionaries belong to the built-in mapping type.

Example

```
my_Dictionay = {'ID': 1110, 'Name':'John', 'Age': 12}
print (my_Dictionay['ID'])
print (my_Dictionay['Age'])
#insert
my_Dictionay['Total Marks']=600
```

output

```
1110
12
{'Total Marks': 600, 'Age': 12, 'ID': 1110, 'Name': 'John'}
```