**Python**

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Python is an interpreted language, which can save you considerable time during program development because no compilation and linking is necessary. The interpreter can be used interactively, which makes it easy to experiment with features of the language, to write throw-away programs, or to test functions during bottom-up program development. It is also a handy desk calculator.

Python enables programs to be written compactly and readably. Programs written in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons: the high-level data types allow you to express complex operations in a single statement; statement grouping is done by indentation instead of beginning and ending brackets; no variable or argument declarations are necessary.

Python is extensible: if you know how to program in C it is easy to add a new built-in function or module to the interpreter, either to perform critical operations at maximum speed, or to link Python programs to libraries that may only be available in binary form (such as a vendor-specific graphics library). Once you are really hooked, you can link the Python interpreter into an application written in C and use it as an extension or command language for that application.

**List**

The list data type has some more methods. Here are all of the methods of list objects:

list.append(*x*)

Add an item to the end of the list. Equivalent to a[len(a):] = [x].

list.extend(*iterable*)

Extend the list by appending all the items from the iterable. Equivalent to a[len(a):] = iterable.

list.insert(*i*, *x*)

Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.remove(*x*)

Remove the first item from the list whose value is *x*. It is an error if there is no such item.

list.pop([*i*])

Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.clear()

Remove all items from the list. Equivalent to del a[:].

list.index(*x*[, *start*[, *end*]])

Return zero-based index in the list of the first item whose value is *x*. Raises a [ValueError](../library/exceptions.html) if there is no such item.

The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

list.count(*x*)

Return the number of times *x* appears in the list.

list.sort(*key=None*, *reverse=False*)

Sort the items of the list in place (the arguments can be used for sort customization, see [sorted()](../library/functions.html) for their explanation).

list.reverse()

Reverse the elements of the list in place.

list.copy()

Return a shallow copy of the list. Equivalent to a[:].

An example that uses most of the list methods:

>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

>>> fruits.count('apple')

2

>>> fruits.count('tangerine')

0

>>> fruits.index('banana')

3

>>> fruits.index('banana', 4) # Find next banana starting a position 4

6

>>> fruits.reverse()

>>> fruits

['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']

>>> fruits.append('grape')

>>> fruits

['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']

>>> fruits.sort()

>>> fruits

['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']

>>> fruits.pop()

'pear'

You might have noticed that methods like insert, remove or sort that only modify the list have no return value printed – they return the default None. This is a design principle for all mutable data structures in Python.

**Tuple**

A tuple is just like a list of a sequence of immutable python objects. The difference between list and tuple is that list are declared in square brackets and can be changed while tuple is declared in parentheses and cannot be changed. However, you can take portions of existing tuples to make new tuples.

Tuple Syntax

Tup = ('Jan','feb','march')

To write an empty tuple, you need to write as two parentheses containing nothing-

tup1 = ();

**Tuple Assignment**

Python has tuple assignment feature which enables you to assign more than one variable at a time. In here, we have assigned tuple 1 with the persons information like name, surname, birth year, etc. and another tuple 2 with the values in it like number (1,2,3,….,7).

For Example,

(name, surname, birth year, favorite movie and year, profession, birthplace) = Robert

Here is the code,

tup1 = ('Robert', 'Carlos','1965','Terminator 1995', 'Actor','Florida');

tup2 = (1,2,3,4,5,6,7);

print(tup1[0])

print(tup2[1:4])

Tuple 1 includes list of information of Robert

Tuple 2 includes list of numbers in it

* We call the value for [0] in tuple and for tuple 2 we call the value between 1 and 4
* Run the code- It gives name Robert for first tuple while for second tuple it gives number (2,3 and 4)

**Packing and Unpacking**

In packing, we place value into a new tuple while in unpacking we extract those values back into variables.

x = ("Guru99", 20, "Education") # tuple packing

(company, emp, profile) = x # tuple unpacking

print(company)

print(emp)

print(profile)

**Comparing tuples**

A comparison operator in Python can work with tuples.

The comparison starts with a first element of each tuple. If they do not compare to =,< or > then it proceed to the second element and so on.

It starts with comparing the first element from each of the tuples

Let's study this with an example-

**#case 1**

a=(5,6)

b=(1,4)

if (a>b):print("a is bigger")

else: print("b is bigger")

**#case 2**

a=(5,6)

b=(5,4)

if (a>b):print("a is bigger")

else: print ("b is bigger")

**#case 3**

a=(5,6)

b=(6,4)

if (a>b):print("a is bigger")

else: print("b is bigger")

Case1: Comparison starts with a first element of each tuple. In this case 5>1, so the output a is bigger

Case 2: Comparison starts with a first element of each tuple. In this case 5>5 which is inconclusive. So it proceeds to the next element. 6>4, so the output a is bigger

Case 3: Comparison starts with a first element of each tuple. In this case 5>6 which is false. So it goes into the else loop prints "b is bigger."

**Using tuples as keys in dictionaries**

Since tuples are hashable, and list is not, we must use tuple as the key if we need to create a composite key to use in a dictionary.

Example: We would come across a composite key if we need to create a telephone directory that maps, first-name, last-name, pairs of telephone numbers, etc. Assuming that we have declared the variables as last and first number, we could write a dictionary assignment statement as shown below:

directory[last,first] = number

Inside the brackets, the expression is a tuple. We could use tuple assignment in a for loop to navigate this dictionary.

for last, first in directory:

print first, last, directory[last, first]

This loop navigates the keys in the directory, which are tuples. It assigns the elements of each tuple to last and first and then prints the name and corresponding telephone number.

**Tuples and dictionary**

Dictionary can return the list of tuples by calling items, where each tuple is a key value pair.

a = {'x':100, 'y':200}

b = list(a.items())

print(b)

**Deleting Tuples**

Tuples are immutable and cannot be deleted, but deleting tuple entirely is possible by using the keyword "del."

**Slicing of Tuple**

To fetch specific sets of sub-elements from tuple or list, we use this unique function called slicing. Slicing is not only applicable to tuple but also for array and list.

x = ("a", "b","c", "d", "e")

print(x[2:4])

The output of this code will be ('c', 'd').

**Here is the Python 2 Code for all above example**

tup1 = ('Robert', 'Carlos','1965','Terminator 1995', 'Actor','Florida');

tup2 = (1,2,3,4,5,6,7);

print tup1[0]

print tup2[1:4]

#Packing and Unpacking

x = ("Guru99", 20, "Education") # tuple packing

(company, emp, profile) = x # tuple unpacking

print company

print emp

print profile

#Comparing tuples

#case 1

a=(5,6)

b=(1,4)

if (a>b):print "a is bigger"

else: print "b is bigger"

#case 2

a=(5,6)

b=(5,4)

if (a>b):print "a is bigger"

else: print "b is bigger"

#case 3

a=(5,6)

b=(6,4)

if (a>b):print "a is bigger"

else: print "b is bigger"

#Tuples and dictionary

a = {'x':100, 'y':200}

b = a.items()

print b

#Slicing of Tuple

x = ("a", "b","c", "d", "e")

print x[2:4]

**Built-in functions with Tuple**

To perform different task, tuple allows you to use many built-in functions like all(), any(), enumerate(), max(), min(), sorted(), len(), tuple(), etc.

**Advantages of tuple over list**

Iterating through tuple is faster than with list, since tuples are immutable.

Tuples that consist of immutable elements can be used as key for dictionary, which is not possible with list

If you have data that is immutable, implementing it as tuple will guarantee that it remains write-protected

**Dictionary**

A dictionary is used to map or associate things you want to store the keys you need to get them. A dictionary in Python is just like a dictionary in the real world. Python Dictionary are defined into two elements Keys and Values.

Keys will be a single element. Values can be a list or list within a list, numbers, etc.

**Syntax for Dictionary:**

Dict = { ' Tim': 18, xyz,.. }

Example:

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print (Dict['Tiffany'])

**Properties of Dictionary Keys**

There are two important points while using dictionary keys. More than one entry per key is not allowed ( no duplicate key is allowed)

* The values in the dictionary can be of any type while the keys must be immutable like numbers, tuples or strings.
* Dictionary keys are case sensitive- Same key name but with the different case are treated as different keys in Python dictionaries.

**Updating Dictionary**

You can also update a dictionary by adding a new entry or a key-value pair to an existing entry or by deleting an existing entry. Here in the example we will add another name "Sarah" to our existing dictionary.

Example

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Dict.update({"Sarah":9})

print Dict

* Our existing dictionary "Dict" does not have the name "Sarah."
* We use the method Dict.update to add Sarah to our existing dictionary
* Now run the code, it adds Sarah to our existing dictionary

**Delete Keys from the dictionary**

Python dictionary gives you the liberty to delete any element from the dictionary list. Suppose you don't want the name Charlie in the list, so you can delete the key element by following code.

Example

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

del Dict ['Charlie']

print Dict

* When you run this code, it should print the dictionary list without Charlie.
* We used the code del Dict
* When code executed, it has deleted the Charlie from the main dictionary

**Dictionary items() Method**

The items() method returns a list of tuple pairs (Keys, Value) in the dictionary.

Example

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print "Students Name: %s" % Dict.items()

* We use the code items() method for our Dict.
* When code was executed, it returns a list of items ( keys and values) from the dictionary

**Check if a given key already exists in a dictionary**

For a given list, you can also check whether our child dictionary exists in a main dictionary or not. Here we have two sub-dictionaries "Boys" and "Girls", now we want to check whether our dictionary Boys exist in our main "Dict" or not. For that, we use the forloop method with else if method.

Example

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Boys = {'Tim': 18,'Charlie':12,'Robert':25}

Girls = {'Tiffany':22}

for key in Boys.keys():

if key in Dict.keys():

print True

else:

print False

* he forloop in code checks each key in the main dictionary for Boys keys
* If it exists in the main dictionary, it should print true or else it should print false
* When you execute the code, it will print "True" for three times, as we got three elements in our "Boys" dictionary
* So it indicates that the "Boys" exist in our main dictionary (Dict)

**Sorting the Dictionary**

In the dictionary, you can also sort the elements. For example, if we want to print the name of the elements of our dictionary alphabetically we have to use the forloop. It will sort each element of dictionary accordingly.

Example

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

Boys = {'Tim': 18,'Charlie':12,'Robert':25}

Girls = {'Tiffany':22}

Students = Dict.keys()

Students.sort()

for S in Students:

print":".join((S,str(Dict[S])))

* We declared the variable students for our dictionary "Dict."
* Then we use the code Students.sort, which will sort the element inside our dictionary
* But to sort each element in dictionary, we run the forloop by declaring variable S
* Now, when we execute the code the forloop will call each element from the dictionary, and it will print the string and value in an order

**Dictionary Str(dict)**

With Str() method, you can make a dictionary into a printable string format.

Example

Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}

print "printable string:%s" % str (Dict)

* It will return the dictionary elements into a printable string format

**Class**

A Class is a logical grouping of data and functions. It gives the freedom to create data structures that contains arbitrary content and hence easily accessible.

For example, for any bank employee who want to fetch the customer details online would go to customer class, where all its attributes like transaction details, withdrawal and deposit details, outstanding debt, etc. would be listed out.

How to define Python classes

To define class you need to consider following points

Step 1) In Python, classes are defined by the "Class" keyword

class myClass():

Step 2) Inside classes, you can define functions or methods that are part of this class

def method1 (self):

print "Guru99"

def method2 (self,someString):

print "Software Testing:" + someString

* Here we have defined method1 that prints "Guru99."
* Another method we have defined is method2 that prints "Software Testing"+ SomeString. SomeString is the variable supplied by the calling method
* Step 3) Everything in a class is indented, just like the code in the function, loop, if statement, etc. Anything not indented is not in the class.

NOTE: About using "self" in Python

* The self-argument refers to the object itself. Hence the use of the word self. So inside this method, self will refer to the specific instance of this object that's being operated on.
* Self is the name preferred by convention by Pythons to indicate the first parameter of instance methods in Python. It is part of the Python syntax to access members of objects

Step 4) To make an object of the class

c = myClass()

Step 5) To call a method in a class

c.method1()

c.method2(" Testing is fun")

* Notice that when we call the method1 or method2, we don't have to supply the self-keyword. That's automatically handled for us by the Python runtime.
* Python runtime will pass "self" value when you call an instance method on in instance, whether you provide it deliberately or not
* You just have to care about the non-self arguments

Step 6) Here is the complete code

# Example file for working with classes

class myClass():

def method1(self):

print("Guru99")

def method2(self,someString):

print("Software Testing:" + someString)

def main():

# exercise the class methods

c = myClass ()

c.method1()

c.method2(" Testing is fun")

if \_\_name\_\_== "\_\_main\_\_":

main()