**Unit II**

**String, List, Tuple ,Set& Dictionary**

**Introduction:-**

A **string** is a sequence of one or more characters (letters, numbers, symbols) that can be either a constant or a variable. Made up of Unicode, strings are immutable sequences, meaning they are unchanging.Because text is such a common form of data that we use in everyday life, the string data type is a very important building block of programming.This Python will go over how to create and print strings, how to concatenate and replicate strings, and how to store strings in variables.

**Creating strings:-**

Creating strings is as simple as assigning a value to a variable.

**For example −**

var1 = 'Hello World!'

var2 = "Python Programming"

**Accessing Values in Strings**

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring.

**For example −**

var1 = 'Hello World!'

var2 = "Python Programming"

print "var1[0]: ", var1[0]

print "var2[1:5]: ", var2[1:5]

**Output:-**

var1[0]: H

var2[1:5]: ytho

**Updating Strings:-**

You can "update" an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether.

**For example −**

var1 = 'Hello World!'

print "Updated String :- ", var1[:6] + 'Python'

**Output:-**

Updated String :- Hello Python

1. **String Concatenation:-**

Concatenation means joining strings together end-to-end to create a new string. To concatenate strings, we use the + operator. Keep in mind that when we work with [numbers, + will be an operator for addition](https://www.digitalocean.com/community/tutorials/how-to-do-math-in-python-3-with-operators#addition-and-subtraction), but when used with strings it is a joining operator.

Let’s combine the strings "Sammy" and "Shark" together with concatenation through a print()statement:

print("Sammy" + "Shark")

**Output**

SammyShark

If we would like a whitespace between the two strings, we can simply include the whitespace within a string, like after the word “Sammy”:

print("Sammy " + "Shark")

**Output**

Sammy Shark

Be sure not to use the + operator between two different data types. We can’t concatenate strings and integers together, for instance. So, if we try to write:

print("Sammy" + 27)

We will receive the following error:

TypeError: Can't convert 'int' object to str implicitly

If we wanted to create the string "Sammy27", we could do so by putting the number 27 in quotes ("27") so that it is no longer an integer but is instead a string.

## String Replication:-

There may be times when you need to use Python to automate tasks, and one way you may do this is through repeating a string several times. You can do so with the \* operator. Like the + operator, [the \*operator has a different use when used with numbers](https://www.digitalocean.com/community/tutorials/how-to-do-math-in-python-3-with-operators#multiplication-and-division), where it is the operator for multiplication. When used with one string and one integer, \* is the **string replication operator**, repeating a single string however many times you would like through the integer you provide.

**Example:-**

print("Sammy" \* 9)

**Output**

SammySammySammySammySammySammySammySammySammy

With string replication, we can repeat the single string value the amount of times equivalent to the integer value.

1. **string comparison:-**

To compare two strings in python, you have to ask from user to enter any two string to check whether the two string are equal or not as shown in the program given below.

**Example:-**

string1 =raw\_input("Enter string1: \n")

string2 =raw\_input("Enter string2: ")

if string1 == string2:

print("Both Strings are Equal.\n")

else:

print("Strings are Not Equal.\n")

**Example:-**

word = "zebra"

if word == "banana":

print "string is equal"

else:

print "string is not equal"

1. **String length:-**

The method **len()** returns the length of the string.

**Syntax:-**

**len(str )**

**Example:-**

str = "this is string example....wow!!!";

print "Length of the string: ", len(str)

**Output:-**

Length of the string: 32

1. **String Reverse:-**

Python string library does’nt support the in-built “reverse()” as done by other python containers like list, hence knowing other methods to reverse string can prove to be useful. This article discusses several ways to achieve it.

|  |
| --- |
| **Example:-**   a=str(raw\_input("Enter a string: "))  **print**("Reverse of the string is: ")  **print**(a[::-1]) |

1. **String capitalize :-**

It returns a copy of the string with only its first character capitalized.

## Syntax

str.capitalize()

**Example:-**

str = "this is string example....wow!!!";

print "str.capitalize() : ", str.capitalize()

**Output:-**

This is string example....wow!!!;

# String islower:-

The method **islower()** checks whether all the case-based characters (letters)

of the string are lowercase.

## Syntax:-

str.islower()

**Example:**

str="THIS is string example....wow!!!";

printstr.islower()

str="this is string example....wow!!!";

printstr.islower()

**Output:-**

**-**False

True

1. **String isupper:-**

The method isupper() checks whether all the case-based characters (letters) of the string are uppercase.

**Syntax:-**

str.isupper()

**Example:-**

str = "THIS IS STRING EXAMPLE....WOW!!!";

printstr.isupper()

str = "THIS is string example....wow!!!";

printstr.isupper()

**Output:-**

True

False

1. **String lower:-**

The method **lower()** returns a copy of the string in which all case-based characters have been lowercased.

## Syntax:-

str.lower()

## Example:-

str="THIS IS STRING EXAMPLE....WOW!!!";

printstr.lower()

**Output:-**

this is string example....wow!!!

# String upper:-

The method **upper()** returns a copy of the string in which all case-based characters have been uppercased.

**Syntax**

str.upper()

**Example**

str = "this is string example....wow!!!";

print "str.capitalize() : ", str.upper()

**Output:-**

str.capitalize() : THIS IS STRING EXAMPLE....WOW!!!

**11)split():-**

Splits the string at the specified separator, and returns a list.

**Example:-**

txt = "welcome to the jungle"  
x = txt.split()  
print(x)

**12)splitlines():-**

Splits the string at line breaks and returns a list.

**Example:-**

txt = "Thank you for the music\nWelcome to the jungle"  
x = txt.splitlines()  
print(x)

**13)join():-**

Join all items in a tuple into a string, using a hash character as separator.

**Example:-**

myTuple = ("John", "Peter", "Vicky")

x = "#".join(myTuple)

print(x)

**List:-**

**Introduction:-**

The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.

Python has six built-in types of sequences, but the most common ones are lists and tuples, which we would see in this tutorial.

There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.

**Python Lists**

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets.

**For example −**

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5 ];

list3 = ["a", "b", "c", "d"]

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

**Accessing Values in Lists**

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

**Example −**

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5, 6, 7 ];

print "list1[0]: ", list1[0]

print "list2[1:5]: ", list2[1:5]

**Output:-**

list1[0]: physics

list2[1:5]: [2, 3, 4, 5]

**Updating Lists**

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method.

**Example −**

#"Value available at index 2 : "

list=['physics', 'chemistry', 1997, 2000]

print list[2]

list[2] = 2001

print "New value available at index 2 : "

print list

**Delete List Elements**

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.

**For example −**

list1 = ['physics', 'chemistry', 1997, 2000];

print list1

del list1[2];

print "After deleting value at index 2 : "

print list1

**Output:-**

['physics', 'chemistry', 1997, 2000]

After deleting value at index 2 :

['physics', 'chemistry', 2000]

**List Operations:-**

List is a type of data structuring method that allows storing of the integers or the characters in an order indexed by starting from 0. List operations are the operations that can be performed on the data in the list data structure. A few of the basic list operations used in Python programming are extend(), insert(), append(), remove(), pop(), slice, reverse(), min() & max(), concatenate(), count(), multiply(), sort(), index(), clear(), etc.

# 1. append()-

# The append() method is used to add elements at the end of the list. This method can only add a single element at a time. To add multiple elements, the append() method can be used inside a loop.

# Code:

myList = ['physics', 'chemistry', 1997, 2000]

# myList.append(4)

# myList.append(5)

# myList.append(6)

print myList

# 2. extend()-

# The extend() method is used to add more than one element at the end of the list. Although it can add more than one element, unlike append(), it adds them at the end of the list like append().

# Code:

# myList = ['physics', 'chemistry', 1997, 2000]

# myList.extend([4, 5, 6])

# for i in range(7, 9):

# myList.append(i)

# print(myList)

# 3. insert()

# The insert() method can add an element at a given position in the list. Thus, unlike append(), it can add elements at any position, but like append(), it can add only one element at a time. This method takes two arguments. The first argument specifies the position, and the second argument specifies the element to be inserted.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola']

# myList.insert(3, 400)

# myList.insert(4, 500)

# myList.insert(5, 600)

# print(myList))

# 4. remove()

# The remove() method is used to remove an element from the list. In the case of multiple occurrences of the same element, only the first occurrence is removed.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola',2000]

# print(myList)

# myList.remove(2000)

# print(myList)

# 5. pop()

# The method pop() can remove an element from any position in the list. The parameter supplied to this method is the index of the element to be removed.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola',20]

# print(myList)

# myList.pop(4)

# print(myList)

# 6. slice

# The slice operation is used to print a section of the list. The slice operation returns a specific range of elements. It does not modify the original list.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola',20]

# print(myList[:4]) # prints from beginning to end index

# print(myList[2:]) # prints from start index to end of list

# print(myList[2:4]) # prints from start index to end index

# print(myList[:])

# 7. reverse()

# The reverse() operation is used to reverse the elements of the list. This method modifies the original list. To reverse a list without modifying the original one, we use the slice operation with negative indices. Specifying negative indices iterates the list from the rear end to the front end of the list.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola',20]

# print(myList[::-1]) # does not modify the original list

# #myList.reverse() # modifies the original list

# #print(myList)

# 8. len()

# The len() method returns the length of the list, i.e. the number of elements in the list.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola',20]

# print(len(myList))

# 9. min() & max()

# The min() method returns the minimum value in the list. The max() method returns the maximum value in the list. Both the methods accept only homogeneous lists, i.e. lists having elements of similar type.

# Code:

# print(min([1, 2, 3]))

# print(max([1, 2, 3]))

# 10. count()

# The function count() returns the number of occurrences of a given element in the list.

# Code:

# myList = ['physics', 'chemistry', 1999, 20,'sangola',20]

# print(myList.count(20)

# 11. concatenate

# The concatenate operation is used to merge two lists and return a single list. The + sign is used to perform the concatenation. Note that the individual lists are not modified, and a new combined list is returned.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola',20]

# yourList = [4, 5, 'Python', 'is fun!']

# print(myList+yourList)

# 12. multiply

# Python also allows multiplying the list n times. The resultant list is the original list iterated n times.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola',20]

# print(myList\*2)

# 13. index()

# The index() method returns the position of the first occurrence of the given element. It takes two optional parameters – the beginning index and the end index. These parameters define the start and end position of the search area on the list. When supplied, the element is searched only in the sub-list bound by the begin and end indices. When not supplied, the element is searched in the whole list.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola',20]

# print(myList.index('sngola')) # searches in the whole list

# print(myList.index('EduCBA', 0, 2)) # searches from 0th to 2nd position

# 14. sort()

# The sort method sorts the list in ascending order. This operation can only be performed on homogeneous lists, i.e. lists having elements of similar type.

# Code:

# yourList = [4, 2, 6, 5, 0, 1]

# yourList.sort()

# print(yourList)

# 15. clear()

# This function erases all the elements from the list and empties them.

# Code:

# myList = ['physics', 'chemistry', 1999, 2000,'sangola',20]

# myList.clear()

# print(myList)

# Tuples :-

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.

**Example −**

tup1 = ('physics', 'chemistry', 1997, 2000)

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

tup3 = "a", "b", "c", "d";

The empty tuple is written as two parentheses containing nothing −

tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value −

tup1 = (50,);

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

**Accessing Values in Tuples**

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.

**Example:-**

#!/usr/bin/python

tup1 = ('physics', 'chemistry', 1997, 2000);

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

print "tup1[0]: ", tup1[0]

print "tup2[1:5]: ", tup2[1:5]

When the above code is executed, it produces the following result −

tup1[0]: physics

tup2[1:5]: [2, 3, 4, 5]

**Updating Tuples**

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to

**Example :-**

tup1 = (12, 34.56)

tup2 = ('abc', 'xyz')

tup3 = tup1 + tup2

print tup3

When the above code is executed, it produces the following result −

(12, 34.56, 'abc', 'xyz')

**Delete Tuple Elements**

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement.

**Example :-**

#!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000)

print tup

del tup;

print "After deleting tup : "

print tup

**Output:-**

('physics', 'chemistry', 1997, 2000)

After deleting tup :

Traceback (most recent call last):

File "test.py", line 9, in <module>

printtup;

NameError: name 'tup' is not defined

**Basic Tuples Operations**

Tuples respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

|  |  |  |
| --- | --- | --- |
| **Python Expression** | **Results** | **Description** |
| len((1, 2, 3)) | 3 | Length |
| (1, 2, 3) + (4, 5, 6) | (1, 2, 3, 4, 5, 6) | Concatenation |
| ('Hi!',) \* 4 | ('Hi!', 'Hi!', 'Hi!', 'Hi!') | Repetition |
| 3 in (1, 2, 3) | True | Membership |
| for x in (1, 2, 3): print x, | 1 2 3 | Iteration |

**Indexing, Slicing, and Matrixes**

Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings.

**Assuming following input −**

L = ('spam', 'Spam', 'SPAM!')

|  |  |  |
| --- | --- | --- |
| **Python Expression** | **Results** | **Description** |
| L[2] | 'SPAM!' | Offsets start at zero |
| L[-2] | 'Spam' | Negative: count from the right |
| L[1:] | ['Spam', 'SPAM!'] | Slicing fetches sections |

**No Enclosing Delimiters**

Any set of multiple objects, comma-separated, written without identifying symbols, i.e., brackets for lists, parentheses for tuples, etc., default to tuples, as indicated in these short

**Examples −**

print 'abc', -4.24e93, 18+6.6j, 'xyz'

x, y = 1, 2;

print "Value of x , y : ", x,y

**Output:-**

abc -4.24e+93 (18+6.6j) xyz

Value of x , y : 1 2

**Built-in Tuple Functions :-**

|  |  |
| --- | --- |
| **Sr.No.** | **Function with Description** |
| 1 | [**cmp(tuple1, tuple2)**](https://www.tutorialspoint.com/python/tuple_cmp.htm)  Compares elements of both tuples. |
| 2 | [**len(tuple)**](https://www.tutorialspoint.com/python/tuple_len.htm)  Gives the total length of the tuple. |
| 3 | [**max(tuple)**](https://www.tutorialspoint.com/python/tuple_max.htm)  Returns item from the tuple with max value. |
| 4 | [**min(tuple)**](https://www.tutorialspoint.com/python/tuple_min.htm)  Returns item from the tuple with min value. |
| 5 | [**tuple(seq)**](https://www.tutorialspoint.com/python/tuple_tuple.htm)  Converts a list into tuple. |

**1)[cmp(tuple1, tuple2)](https://www.tutorialspoint.com/python/tuple_cmp.htm):-**

The method cmp() compares elements of two tuples.

**Syntax:-**

cmp(tuple1, tuple2)

Parameters-

**T1 − This is the first tuple to be compared**

**T2 − This is the second tuple to be compared**

# Return Value

# If elements are of the same type, perform the compare and return the result. If elements are different types, check to see if they are numbers.

# if T1 > T2, then cmp(T1, T2) returns 1

# if T1 = T2, then cmp(T1, T2) returns 0

# if T2 > T1, then cmp(T2, T1) returns -1

# Example:-

# tuple1, tuple2 = (123, 'xyz'), (456, 'abc')

# print cmp(tuple1, tuple2)

# print cmp(tuple2, tuple1)

# Output:-

# -1

# 1

# 2)[len(tuple)](https://www.tutorialspoint.com/python/tuple_len.htm):-

The method **len()** returns the number of elements in the tuple.

## Syntax

len(tuple)

**Parameters**

* **tuple** − This is a tuple for which number of elements to be counted.

## Return Value

This method returns the number of elements in the tuple.

**Example**

tuple1, tuple2 = (123, 'xyz', 'zara'), (456, 'abc')

print "First tuple length : ", len(tuple1)

print "Second tuple length : ", len(tuple2)

**Output:-**

First tuple length : 3

Second tuple length : 2

**3)[max(tuple)](https://www.tutorialspoint.com/python/tuple_max.htm):-**

The method **max()** returns the elements from the tuple with maximum value.

**Syntax**

max(tuple)

**Parameters**

* **tuple** − This is a tuple from which max valued element to be returned.

**Return Value**

This method returns the elements from the tuple with maximum value.

**Example**

tuple1, tuple2 = (123, 'xyz', 'zara', 'abc'), (456, 700, 200)

print "Max value element : ", max(tuple1)

print "Max value element : ", max(tuple2)

**Output:-**

Max value element :zara

Max value element : 700

**4)[min(tuple)](https://www.tutorialspoint.com/python/tuple_min.htm):-**

The method **min()** returns the elements from the tuple with minimum value.

**Syntax**

min(tuple)

**Parameters**

* **tuple** − This is a tuple from which min valued element to be returned.

Return Value

This method returns the elements from the tuple with minimum value.

**Example**

tuple1, tuple2 = (123, 'xyz', 'zara', 'abc'), (456, 700, 200)

print "min value element : ", min(tuple1)

print "min value element : ", min(tuple2)

**Output:-**

min value element : 123

min value element : 200

**5)[tuple(seq)](https://www.tutorialspoint.com/python/tuple_tuple.htm):-**

The method **tuple()** converts a list of items into tuples

**Syntax**

tuple(seq )

Parameters

* **seq** − This is a sequence to be converted into tuple.

Return Value

This method returns the tuple.

**Example**

aList = [123, 'xyz', 'zara', 'abc'];

aTuple = tuple(aList)

print "Tuple elements : ", aTuple

**Output:-**

Tuple elements : (123, 'xyz', 'zara', 'abc')

**Dictionary :-**

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

**Accessing Values in Dictionary**

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

**Example −**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']

print "dict['Age']: ", dict['Age']

**Output:-**

dict['Name']: Zara

dict['Age']: 7

**Updating Dictionary**

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple **Example :-**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

dict['Age'] = 8; # update existing entry

dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

**Output:-**

dict['Age']: 8

dict['School']: DPS School

**Delete Dictionary Elements**

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement.

**Example −**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'

dict.clear(); # remove all entries in dict

del dict ; # delete entire dictionary

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

**Output:-**

dict['Age']:

Traceback (most recent call last):

File "test.py", line 8, in <module>

print "dict['Age']: ", dict['Age'];

TypeError: 'type' object is unsubscriptable

**Properties of Dictionary Keys**

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

**(a)** More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.

**For example −**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}

print "dict['Name']: ", dict['Name']

**Output:-**

dict['Name']: Manni

**(b)** Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

**Following example :-**

#!/usr/bin/python

dict = {['Name']: 'Zara', 'Age': 7}

print "dict['Name']: ", dict['Name']

**Output:-**

Traceback (most recent call last):

File "test.py", line 3, in <module>

dict = {['Name']: 'Zara', 'Age': 7};

TypeError: list objects are unhashable

**Built-in Dictionary Functions & Methods**

|  |  |
| --- | --- |
| **Sr.No.** | **Function with Description** |
| 1 | [**cmp(dict1, dict2)**](https://www.tutorialspoint.com/python/dictionary_cmp.htm)  Compares elements of both dict. |
| 2 | [**len(dict)**](https://www.tutorialspoint.com/python/dictionary_len.htm)  Gives the total length of the dictionary. This would be equal to the number of items in the dictionary. |
| 3 | [**str(dict)**](https://www.tutorialspoint.com/python/dictionary_str.htm)  Produces a printable string representation of a dictionary |
| 4 | [**type(variable)**](https://www.tutorialspoint.com/python/dictionary_type.htm)  Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type. |

1. [**cmp(dict1, dict2)**](https://www.tutorialspoint.com/python/dictionary_cmp.htm):-

The method cmp() compares two dictionaries based on key and values.

**Syntax**

cmp(dict1, dict2)

**Parameters**

* dict1 − This is the first dictionary to be compared with dict2.
* dict2 − This is the second dictionary to be compared with dict1.

**Return Value**

This method returns 0 if both dictionaries are equal, -1 if dict1 < dict2 and 1 if dict1 > dic2.

**Example**

#!/usr/bin/python

dict1 = {'Name': 'Zara', 'Age': 7};

dict2 = {'Name': 'Mahnaz', 'Age': 27};

dict3 = {'Name': 'Abid', 'Age': 27};

dict4 = {'Name': 'Zara', 'Age': 7};

print "Return Value : %d" % cmp (dict1, dict2)

print "Return Value : %d" % cmp (dict2, dict3)

print "Return Value : %d" % cmp (dict1, dict4)

**Output:-**

Return Value : -1

Return Value : 1

**Return Value : 0**

**2)[len(dict)](https://www.tutorialspoint.com/python/dictionary_len.htm)**:-

The method len() gives the total length of the dictionary. This would be equal to the number of items in the dictionary.

**Syntax**

len(dict)

**Parameters**

* dict − This is the dictionary, whose length needs to be calculated.

**Return Value**

This method returns the length.

**Example**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7};

print "Length : %d" % len (dict)

**Output:-**

Length : 2

**3)[str(dict)](https://www.tutorialspoint.com/python/dictionary_str.htm)**:-

The method str() produces a printable string representation of a dictionary.

**Syntax**

str(dict)

**Parameters**

dict − This is the dictionary.

**Return Value**

This method returns string representation.

**Exampl:-**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7};

print "Equivalent String : %s" % str (dict)

**Output:-**

Equivalent String : {'Age': 7, 'Name': 'Zara'}

**4)[type(variable)](https://www.tutorialspoint.com/python/dictionary_type.htm)**:-

The method type() returns the type of the passed variable. If passed variable is dictionary then it would return a dictionary type.

**Syntax**

type(dict)

**Parameters**

* dict − This is the dictionary.

**Return Value**

This method returns the type of the passed variable.

**Example**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7};

print "Variable Type : %s" % type (dict)

**Output:-**

Variable Type : <type 'dict'>

**Python includes following dictionary methods −**

|  |  |
| --- | --- |
| **Sr.No.** | **Methods with Description** |
| 1 | [**dict.clear()**](https://www.tutorialspoint.com/python/dictionary_clear.htm)  Removes all elements of dictionary *dict* |
| 2 | [**dict.copy()**](https://www.tutorialspoint.com/python/dictionary_copy.htm)  Returns a shallow copy of dictionary *dict* |
| 3 | [**dict.fromkeys()**](https://www.tutorialspoint.com/python/dictionary_fromkeys.htm)  Create a new dictionary with keys from seq and values*set* to *value*. |
| 4 | [**dict.get(key, default=None)**](https://www.tutorialspoint.com/python/dictionary_get.htm)  For *key* key, returns value or default if key not in dictionary |
| 5 | [**dict.has\_key(key)**](https://www.tutorialspoint.com/python/dictionary_has_key.htm)  Returns *true* if key in dictionary *dict*, *false* otherwise |
| 6 | [**dict.items()**](https://www.tutorialspoint.com/python/dictionary_items.htm)  Returns a list of *dict*'s (key, value) tuple pairs |
| 7 | [**dict.keys()**](https://www.tutorialspoint.com/python/dictionary_keys.htm)  Returns list of dictionary dict's keys |
| 8 | [**dict.setdefault(key, default=None)**](https://www.tutorialspoint.com/python/dictionary_setdefault.htm)  Similar to get(), but will set dict[key]=default if *key* is not already in dict |
| 9 | [**dict.update(dict2)**](https://www.tutorialspoint.com/python/dictionary_update.htm)  Adds dictionary *dict2*'s key-values pairs to *dict* |
| 10 | [**dict.values()**](https://www.tutorialspoint.com/python/dictionary_values.htm)  Returns list of dictionary *dict*'s values |

1. [**dict.clear()**](https://www.tutorialspoint.com/python/dictionary_clear.htm)**:-**

The method **clear()** removes all items from the dictionary.

**Syntax**

dict.clear()

**Return Value**

This method does not return any value.

**Example**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7};

print "Start Len : %d" % len(dict)

dict.clear()

print "End Len : %d" % len(dict)

**Output:-**

Start Len : 2

End Len : 0

2)**[dict.copy()](https://www.tutorialspoint.com/python/dictionary_copy.htm)**:-

The method copy() returns a shallow copy of the dictionary.

**Syntax**

dict.copy()

Return Value

This method returns a shallow copy of the dictionary.

**Example**

#!/usr/bin/python

dict1 = {'Name': 'Zara', 'Age': 7};

dict2 = dict1.copy()

print "New Dictionary : %s" % str(dict2)

**Output:-**

New Dictionary : {'Age': 7, 'Name': 'Zara'}

3)**[dict.fromkeys()](https://www.tutorialspoint.com/python/dictionary_fromkeys.htm)**:-

The method **fromkeys()** creates a new dictionary with keys from *seq* and *values* set to value.

**Syntax**

dict.fromkeys(seq[, value])

**Parameters**

* **seq** − This is the list of values which would be used for dictionary keys preparation.
* **value** − This is optional, if provided then value would be set to this value

**Return Value**

This method returns the list.

**Example**

#!/usr/bin/python

seq = ('name', 'age', 'sex')

dict = dict.fromkeys(seq)

print "New Dictionary : %s" % str(dict)

dict = dict.fromkeys(seq, 10)

print "New Dictionary : %s" % str(dict)

**Output:-**

New Dictionary : {'age': None, 'name': None, 'sex': None}

New Dictionary : {'age': 10, 'name': 10, 'sex': 10}

4) [**dict.get(key, default=None)**](https://www.tutorialspoint.com/python/dictionary_get.htm):-

The method **get()** returns a value for the given key. If key is not available then returns default value None.

**Syntax**

dict.get(key, default = None)

**Parameters**

* **key** − This is the Key to be searched in the dictionary.
* **default** − This is the Value to be returned in case key does not exist.

**Return Value**

This method return a value for the given key. If key is not available, then returns default value None.

**Example**

#!/usr/bin/python

dict = {'Name': 'Zabra', 'Age': 7}

print "Value : %s" % dict.get('Age')

print "Value : %s" % dict.get('Education', "Never")

**Output:-**

Value : 7

Value : Never

5)**[dict.has\_key(key)](https://www.tutorialspoint.com/python/dictionary_has_key.htm)**:-

The method **has\_key()** returns true if a given *key* is available in the dictionary, otherwise it returns a false.

**Syntax**

dict.has\_key(key)

**Parameters**

* **key** − This is the Key to be searched in the dictionary.

**Return Value**

This method return true if a given key is available in the dictionary, otherwise it returns a false.

**Example**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}

print "Value : %s" % dict.has\_key('Age')

print "Value : %s" % dict.has\_key('Sex')

**Output:-**

Value : True

Value : False

**6)[dict.items()](https://www.tutorialspoint.com/python/dictionary_items.htm)**:-

The method items() returns a list of dict's (key, value) tuple pairs

**Syntax**

dict.items()

**Return Value**

This method returns a list of tuple pairs.

**Example**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}

print "Value : %s" % dict.items()

**Output:-**

Value : [('Age', 7), ('Name', 'Zara')]

**7)[dict.keys()](https://www.tutorialspoint.com/python/dictionary_keys.htm)**:-

The method keys() returns a list of all the available keys in the dictionary.

**Syntax**

dict.keys()

**Return Value**

This method returns a list of all the available keys in the dictionary.

**Example**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}

print "Value : %s" % dict.keys()

**Output:-**

Value : ['Age', 'Name']

**8)[dict.update(dict2)](https://www.tutorialspoint.com/python/dictionary_update.htm)**:-

The method **update()** adds dictionary dict2's key-values pairs in to dict. This function does not return anything.

**Syntax**

dict.update(dict2)

**Parameters**

* **dict2** − This is the dictionary to be added into dict.

**Return Value**

This method does not return any value.

**Example**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}

dict2 = {'Sex': 'female' }

dict.update(dict2)

print "Value : %s" % dict

**Output:-**

Value : {'Age': 7, 'Name': 'Zara', 'Sex': 'female'}

**9)[dict.values()](https://www.tutorialspoint.com/python/dictionary_values.htm)**:-

The method values() returns a list of all the values available in a given dictionary.

**Syntax**

dict.values()

**Return Value**

This method returns a list of all the values available in a given dictionary.

**Example**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7}

print "Value : %s" % dict.values()

**Output:-**

Value : [7, 'Zara']

**Set:-**

A Python set is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements. Sets are mutable which means we can modify it after its creation.

Unlike other collections in Python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together, or we can get the list of elements by looping through the set.

**Creating a set**

The set can be created by enclosing the comma-separated immutable items with the curly braces {}.

Python also provides the set() method, which can be used to create the set by the passed sequence.

**Example 1: Using curly braces**

Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}

**print**(Days)

**print**(type(Days))

**print**("looping through the set elements ... ")

**for** i **in** Days:

**print**(i)

**Example 2: Using set() method**

Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])

**print**(Days)

**print**(type(Days))

**print**("looping through the set elements ... ")

**for** i **in** Days:

**print**(i)

### Different methods in Python sets:-

#### 1. add():-

it is used to add a new element in the set, which means you are increasing the number of elements set by one. Here one very important piece of knowledge about the set that needs to be kept in mind is that the element only gets added if it is not already present in the set assets do not take duplicate elements. The add method also does not return any value.

**Code:**

firstset = {"Johnny", "Nilanjan", "Rupa"}

firstset.add("Sepoy")

print("The new word is",firstset)

firstset.add("Sepoy")

print("The new word is",firstset)

#### 2. clear()

it removes all the elements from the set. It neither takes any parameter nor does it return any value. We simply just have to call the clear method and execute it.

**Code:**

firstset = {"Johnny", "Nilanjan", "Rupa"}

print("Before clear",firstset)

firstset.clear()

print("After clear",firstset)

#### 3. copy()

# This method is used to create a shallow copy of a set. The term shallow copy means that if you add new elements in the set or remove elements from the set, the original set does not change. It is the basic advantage of using the copy function. We will see an example to understand the shallow copy concept.

**Code:**

originalset = {"Johnny", "Nilanjan", "Rupa"}

copiedset = originalset.copy()

print("originalset:: ",originalset)

print("copiedset:: ",copiedset)

copiedset.add("Rocky")

print("originalset:: ",originalset)

print("copiedset:: ",copiedset)

**4. difference()**

This is a very important function inset. This function returns a set which is the difference between two sets. Keep in mind that here difference does not mean subtraction because here, it is the difference between the number of elements in two sets and not the values of elements. Here, for example, set A1 – set A2 means it returns a set with elements present in A1 but not in A2 and vice versa in the case of set A2 – set A1 (present in A2 but not in A1).

**Code:**

A1= {24, 35, 34, 45}

A2= {24, 56, 35, 46}

print(A1.difference(A2))

print(A2.difference(A1))

**5. intersection()**

In this case, only the elements which are common in both sets or in multiple sets (in case of more than two sets) are returned in the form of a set.

**Code:**

A1= {24, 35, 34, 45}

A2= {24, 56, 35, 46}

A3= {24, 35, 47, 56}

print(A1.intersection(A2, A3))

**6. union()**

It is a function that returns a set with all elements of the original set and also the specified sets. Since it returns a set so all the items will have only one appearance. If two sets contain the same value, then the item will appear only once.

**Code:**

A1= {24, 35, 34, 45}

A2= {24, 56, 35, 46}

A3= {24, 35, 47, 56}

print(A1.union(A2, A3))

**7. issubset()**

This function returns Boolean values that are true or false. If all the elements of one set are present in another set, then it returns true otherwise false. We will see an example of the same to understand better.

**Code:**

A1 ={3,6,8}

A2 ={45,87,3,67,6,8}

print(A1.issubset(A2))

print(A2.issubset(A1))

**8. issuperset()**

This function returns Boolean values that are true or false. If a set contains all elements of another set, that set can be called a superset of the other set value returned by the function is true otherwise false. We will see an example of the same to understand better.

**Code:**

A1 = {3, 6, 8}

A2 = {45, 87, 3, 67, 6, 8}

print(A1.issuperset(A2))

print(A2.issuperset(A1))

**9. remove()**

This function is used to remove elements from the set. The elements to be removed are passed as arguments. The function removes the element if it is present in the set; otherwise, it returns an error.

**Code:**

firstset = {"Johnny", "Nilanjan", "Rupa"}

firstset.remove("Nilanjan")

print(firstset)

# to check error

firstset.remove("Rocky")

**10. discard()**

This built-in method is also used to remove elements from the set, but it is different from the remove method, which we discussed earlier. If the element is present in the set it removes the element, but if it is present, it returns no error and normally just prints the set.

**Code:**

firstset = {"Johnny", "Nilanjan", "Rupa"}

firstset.discard("Nilanjan")

print(firstset)

firstset.discard("Rocky")

print(firstset)

**11.update()**

To add more than one item in the set, Python provides the **update()** method. It accepts iterable as an argument.

### Example -

Months = set(["January","February", "March", "April", "May", "June"])

**print**("\nprinting the original set ... ")

**print**(Months)

**print**("\nupdating the original set ... ")

Months.update(["July","August","September","October"]);

**print**("\nprinting the modified set ... ")

**print**(Months);