UNIT 3

Python String

- A String is a data structure in Python that represents a sequence of characters.
- It is an immutable data type, meaning that once you have created a string, you cannot change it.
- Strings are used widely in many different applications, such as storing and manipulating text data, representing names, addresses, and other types of data that can be represented as text.

What is a String in Python?

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. Python does not have a character data type, a single character is simply a string with a length of 1.

Creating String in Python

We can create a string by enclosing the characters in single-quotes or double- quotes. Python also provides triple-quotes to represent the string, but it is generally used for multiline string or **docstrings**.

```
#Using single quotes
str1 = 'Hello Python'
print(str1)
#Using double quotes
str2 = "Hello Python"
print(str2)

#Using triple quotes
str3 = '''Triple quotes are generally used for
```

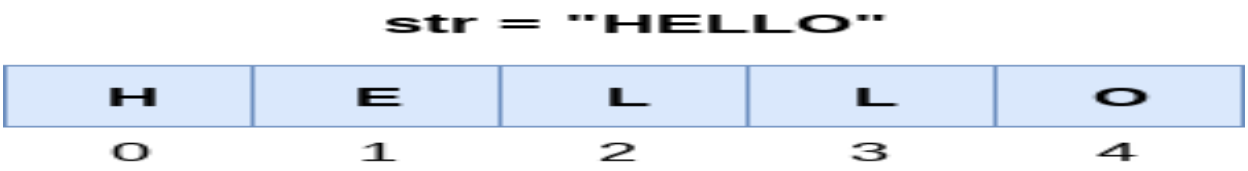
```
represent the multiline or
docstring'''
print(str3)
```

Output –

```
Hello Python
Hello Python
Triple quotes are generally used for
    represent the multiline or
    docstring
```

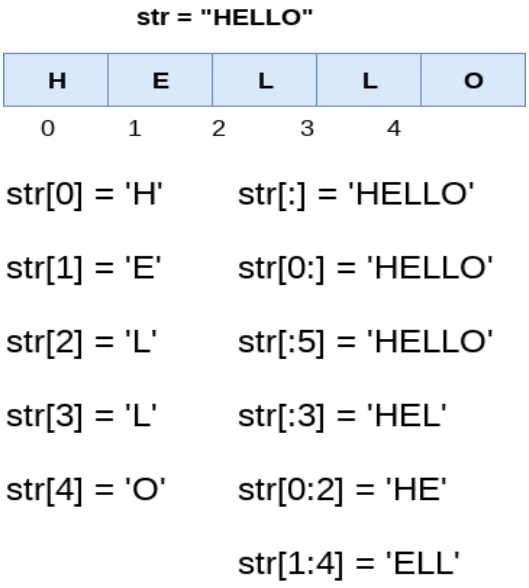
Accessing characters in Python String

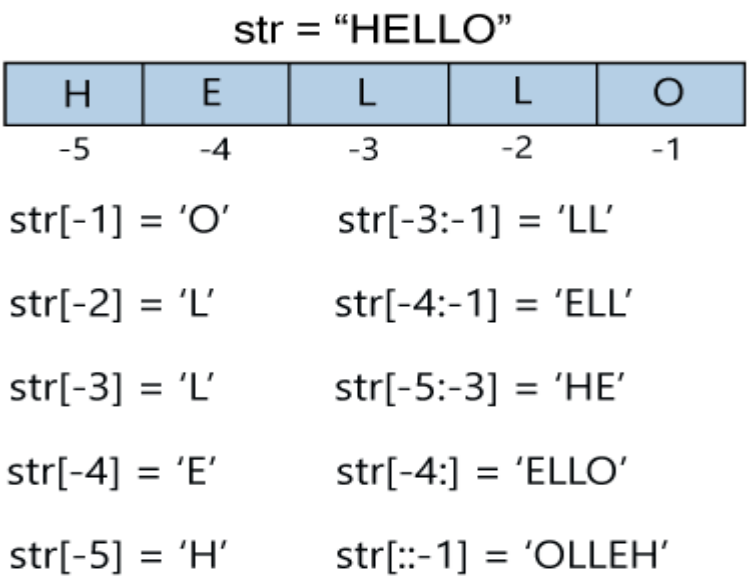
- In Python, individual characters of a String can be accessed by using the method of Indexing.
- Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.
- While accessing an index out of the range will cause an **IndexError**. Only Integers are allowed to be passed as an index, float or other types that will cause a **TypeError**.



String Slicing

- In Python, the String Slicing method is used to access a range of characters in the String. Slicing in a String is done by using a Slicing operator, i.e., a colon (:).
- One thing to keep in mind while using this method is that the string returned after slicing includes the character at the start index but not the character at the last index.





Reassigning Strings

As Python strings are immutable in nature, we cannot update the existing string. We can only assign a completely new value to the variable with the same name.

Consider the following example.

Example 1

```
1. str = "HELLO"
2. str[0] = "h"
3. print(str)
```

Output:

```
Traceback (most recent call last):
  File "12.py", line 2, in <module>
    str[0] = "h";
TypeError: 'str' object does not support item assignment
```

A character of a string can be updated in Python by first converting the string into a Python List and then updating the element in the list. As lists are mutable in nature, we can update the character and then convert the list back into the String.

```
String1 = "Hello, I'm a Geek"
print("Initial String: ")
print(String1)
   #Method 1
list1 = list(String1)
list1[2] = 'p'
String2 = ''.join(list1)
print("\nUpdating character at 2nd Index: ")
print(String2)

 #Method 2
String3 = String1[0:2] + 'p' + String1[3:]
print(String3)
```

Output

```
Initial String:
Hello, I'm a Geek
Updating character at 2nd
Index:
Heplo, I'm a Geek
Heplo, I'm a Geek
```

Example 2

```
1. str = "HELLO"
2. print(str)
3. str = "hello"
4. print(str)
```

Output:

```
HELLO
hello
```

Deleting the String

As we know that strings are immutable. We cannot delete or remove the characters from the string. But we can delete the entire string using the **del** keyword.

```
1. str = "JAVATPOINT"
2. del str[1]
```

Output:

```
TypeError: 'str' object doesn't support item deletion
```

Now we are deleting entire string.

```
1. str1 = "JAVATPOINT"
2. del str1
3. print(str1)
```

Output:

```
NameError: name 'str1' is not defined
```

Escape Sequencing in Python

- While printing Strings with single and double quotes in it causes **SyntaxError** because String already contains Single and Double Quotes and hence cannot be printed with the use of either of these.
- Escape sequences start with a backslash and can be interpreted differently. If single quotes are used to represent a string, then all the single quotes present in the string must be escaped and the same is done for Double Quotes.

Example

```
String1= "C:\\Python\\Geeks\\"
print(String1)
```

```
C:\Python\Geeks\
```

Formatting of Strings

Strings in Python can be formatted with the use of format( ) . Format method in String contains curly braces {} as placeholders which can hold arguments according to position or keyword to specify the order.

```
String1 = "{l} {f} {g}".format(g='Geeks', f='For', l='Life')
print("\nPrint String in order of Keywords: ")
print(String1)
```

Output

```
Print String in order of Keywords:
Life For Geeks
```

String Methods

capitalize()	Converts the first character to upper case
count()	Returns the number of times a specified value occurs in a string
endswith()	Returns true if the string ends with the specified value
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet

isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
islower()	Returns True if all characters in the string are lower case
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
split()	Splits the string at the specified separator, and returns a list
replace()	Returns a string where a specified value is replaced with a specified value
startswith()	Returns true if the string starts with the specified value
title()	Converts the first character of each word to upper case
upper()	Converts a string into upper case
lower()	Converts a string into lower case
swapcase()	Swaps cases, lower case becomes upper case and vice versa

Programs on Python Strings

1. **WAP to display unique words from the string**

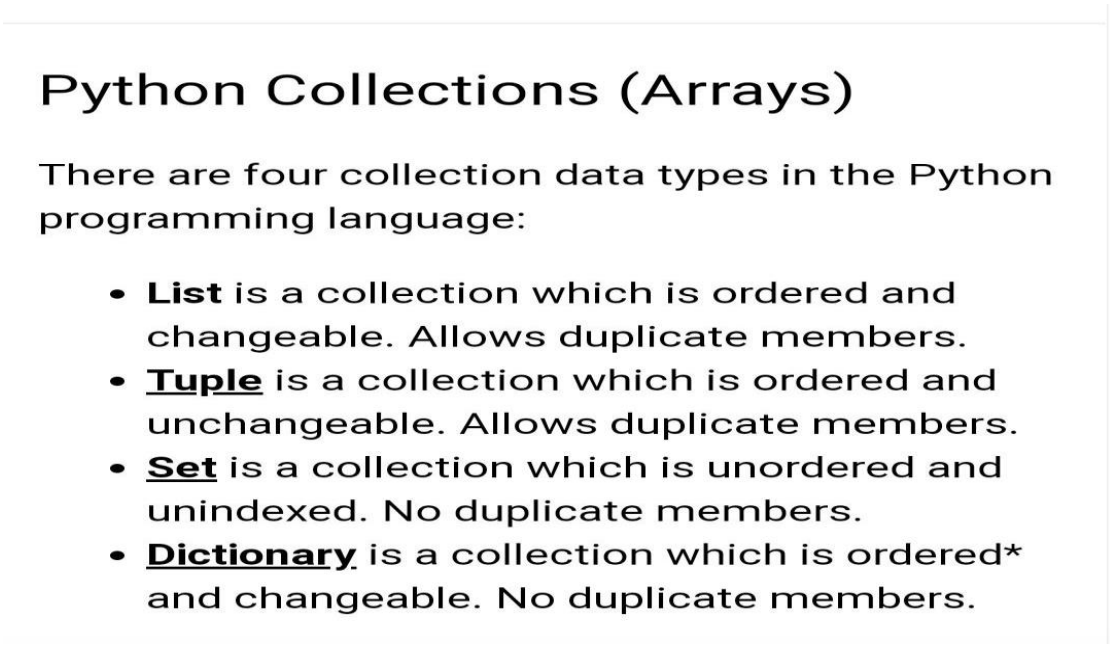
```
str=input("Enter string:")
l=str.split()
l1=[]
for i in l:
    if i not in l1:
        l1.append(i)
str=" ".join(l1)
print(str)
```

2. **WAP to accept a string & replace all spaces by # without using string method.**

```
str=input("Enter string:")
str1=""
for i in str:
    if i.isspace():
        str1+="#"
    else:
        str1+=i
print(“output is”,str1)
```

3. **WAP to accept two strings and then display the common words**

```
str1=input("Enter string1:")
str2=input("Enter string2:")
for i in str1.split():
    if i in str2.split():
        print(i)
```



Python List

- The list is a sequence data type which is used to store the collection of data.
- In lists the comma (,)and the square brackets [enclose the List's items] serve as separators.
- Lists need not be homogeneous always which makes it the most powerful tool in Python.
- A single list may contain DataTypes like Integers, Strings, as well as Objects.
- Lists are mutable, and hence, they can be altered even after their creation.

Creating a List in Python

Lists in Python can be created by just placing the sequence inside the square brackets[].

A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

Example-

List1=[“physics”, “Maths”,34,15.5]

List2=[2,3,5,10]

List Indexing and Splitting

The indexing procedure is carried out similarly to string processing. The slice operator [] can be used to get to the List's components.

The index ranges from 0 to length -1.

List = [0, 1, 2, 3, 4, 5]

0	1	2	3	4	5

List[0] = 0

List[1] = 1

List[2] = 2

List[3] = 3

List[4] = 4

List[5] = 5

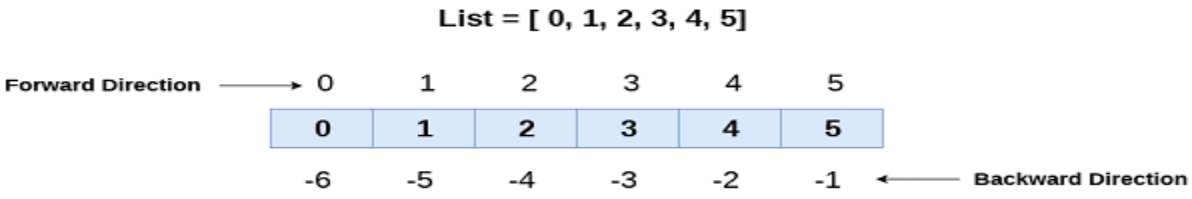
List[0:] = [0,1,2,3,4,5]

List[:] = [0,1,2,3,4,5]

List[2:4] = [2, 3]

List[1:3] = [1, 2]

List[:4] = [0, 1, 2, 3]



We can get the sub-list of the list using the following syntax.

```
1. list_varible(start:stop:step)
```

- The beginning indicates the beginning record position of the rundown.
- The stop signifies the last record position of the rundown.
- Within a start, the step is used to skip the nth element: stop.

Example-
List1=[0,1,2,3,4]
List1[1:3:1]= [1,2]

Adding Elements to a Python List

Method 1: Using append() method -

Only one element at a time can be added to the list by using the append() method

```
List = []
print("Initial blank List: ")
print(List)

List.append(1)
List.append(2)
print("List after Addition of Three elements: ")
print(List)
```

```
Initial blank List:
[]
List after Addition of
Three elements:
[1, 2]
```

Method 2: Using insert() method-

append() method only works for the addition of elements at the end of the List, for the addition of elements at the desired position, insert() method is used.

```
List.insert(2, 12)
print("List after performing Insert Operation: ")
print(List)
```

```
List after performing
Insert Operation:
[1,2,12]
```

Method 3: Using extend() method-

Extend( ) method is used to add multiple elements at the same time at the end of the list.

```
List.extend([8, 'Geeks'])
print("List after performing Extend Operation: ")
print(List)
```

```
List after performing Extend
Operation:
[1,2,12,8,Geeks]
```

Removing Elements from the List

Method 1: Using remove() method

Elements can be removed from the List by using the built-in remove() function but an Error arises if the element doesn't exist in the list.

```
List = [1, 2, 3, 4, 5, 6,
        7, 8, 9, 10, 11, 12]
List.remove(5)
List.remove(6)
print("List after Removal of two elements: ")
print(List)
```

```
List after Removal of two elements:

[1,2,3,4,7,8,9,10,11,12]
```

Method 2: Using pop() method

pop() function can also be used to remove and return an element from the list, but by default it removes only the last element of the list, to remove an element from a specific position of the List, the index of the element is passed as an argument to the pop() method.

```
List = [1, 2, 3, 4, 5]
List.pop()
print("List after popping an element: ")
print(List)
List.pop(2)
print("\nList after popping a specific element: ")
print(List)
```

```
List after popping an element:

[1,2,3,4]

List after popping a specific element:

[1,2,4]
```

Basic List Operations

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

In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter.

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

List out some in-built function in python List

SN	Function	Description
1	cmp(list1, list2)	It compares the elements of both the lists.
2	len(list)	It is used to calculate the length of the list.
3	max(list)	It returns the maximum element of the list.
4	min(list)	It returns the minimum element of the list.
5	list(seq)	It converts any sequence to the list.

List out some in-built methods in python List

SN	Function	Description
1	list.append(obj)	The element represented by the object obj is added to the list.
2	list.clear()	It removes all the elements from the list.
3	List.copy()	It returns a shallow copy of the list.
4	list.count(obj)	It returns the number of occurrences of the specified object in the list.
5	list.extend(seq)	The sequence represented by the object seq is extended to the list.
6	list.index(obj)	It returns the lowest index in the list that object appears.
7	list.insert(index, obj)	The object is inserted into the list at the specified index.
8	list.pop(obj=list[-1])	It removes and returns the last object of the list.
9	list.remove(obj)	It removes the specified object from the list.
10	list.reverse()	It reverses the list.
11	list.sort([func])	It sorts the list by using the specified compare function if given.

List Comprehension

Python List comprehensions are used for creating new lists from other iterables like tuples, strings, arrays, lists, etc. A list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

Syntax:

newList = [expression(element) for element in oldList if condition]

Example-

```
odd_square = [x ** 2 for x in range(1, 11) if x % 2 == 1]
print(odd_square)
```

```
OUTPUT-
[1, 9, 25, 49, 81]
```

Similar to this code –

```
odd_square = []
for x in range(1, 11):
    if x % 2 == 1:
        odd_square.append(x**2)
print(odd_square)
```

Ques : Python Program to find the second largest number in a list.
Ans:

```
a=[]
n=int(input("Enter number of elements:"))
for i in range(1,n+1):
   b=int(input("Enter element:"))
   a.append(b)
a.sort()
print("Second largest element is:",a[n-2])
```

Ques: Program to remove the duplicate items from a list.
Ans:

```
a=[]
n= int(input("Enter the number of elements in list:"))
for x in range(0,n):
   element=int(input("Enter element" ))
   a.append(element)
b = []
[b.append(x) for x in a if x not in b]
print("Non-duplicate items:")
print(unique)
```

Python Tuple

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. For 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.

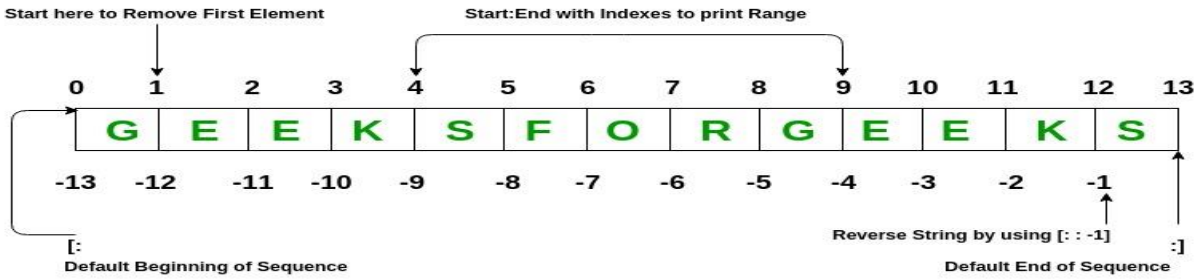
Note: Creation of Python tuple without the use of parentheses is known as Tuple Packing.

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

```
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]
```



Note: We can change any python collection like (list, tuple, set) and string into other data type like tuple, list and string. But cannot change into and float to any python collection. For example: int cannot change into list or vice-versa.

Deleting a Tuple

Tuples are immutable and hence they do not allow deletion of a part of it. The entire tuple gets deleted by the use of del() method.

Note- Printing of Tuple after deletion results in an Error.
Python

```
Tuple1 = (0, 1, 2, 3, 4)

del Tuple1

print(Tuple1)
```

Traceback (most recent call last):
File "/home/efa50fd0709dec08434191f32275928a.py", line 7, in
print(Tuple1)
NameError: name 'Tuple1' 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.

In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter −

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

Built-in Tuple Functions

Python includes the following tuple functions −

Sr.No.	Function with Description
1	cmp(tuple1, tuple2) Compares elements of both tuples.
2	len(tuple) Gives the total length of the tuple.
3	max(tuple) Returns item from the tuple with max value.
4	min(tuple) Returns item from the tuple with min value.
5	tuple(seq) Converts a list into tuple.

The zip() Function

The zip() is an inbuilt function in python. It takes items in sequence from a number of collections to make a list of tuples, where each tuple contains one item from each collection. The function is often used to group items from a list which has the same index.

Example:

```
A=[1,2,3]

B=’XYZ’

Res=list(zip(A,B))# list of tuples

print(Res)
```

Output:

```
[(1, 'X'), (2, 'Y'), (3, 'Z')]
```

We can change into tuple of tuples

```
Res=tuple(zip(A,B))# tuple of tuples
```

Note: if the sequences are not of the same length then the result of zip() has the length of the shorter sequence.

```
A=’abcd’

B=[1,2,3]

Res=list(zip(A,B))

print(Res)
```

Output:

```
[(‘a’,1),(‘b’,2),(‘c’,3)]
```

The Inverse zip(*) Function

The * operator is used within the zip() function. The * operator unpacks a sequence into positional arguments.

X=[(‘apple’,90000),(‘del’,60000),(‘hp’,50000)]

Laptop,prize=zip(*X)

print(Laptop)

print(prize)

Output:

(‘apple’,’del’,’hp’)

(90000,60000,50000)

Python program to find the maximum and minimum K elements in a tuple

```
# Python program to find maximum and minimum k elements in tuple

# Creating a tuple in python
myTuple = (4, 9, 1, 7, 3, 6, 5, 2)
K = 2

# Finding maximum and minimum k elements in tuple
sortedColl = sorted(list(myTuple))
vals = []
for i in range(K):
    vals.append(sortedColl[i])

for i in range((len(sortedColl) - K), len(sortedColl)):
    vals.append(sortedColl[i])

# Printing
print("Tuple : ", str(myTuple))
print("K maximum and minimum values : ", str(vals))
```

Output

```
Tuple :  (4, 9, 1, 7, 3, 6, 5, 2)
K maximum and minimum values :  [1, 2, 7, 9]
```

Python Sets

set is an unordered collection of data items which does not contain duplicate values. Every set element is unique and must be immutable (cannot be changed). However, a set itself is mutable. Elements of set are enclosed inside a pair of curly brackets {}.

We can add and remove items from the set. But cannot replace item as it does not support indexing of data items.

Empty set is created by:

A=set()

print(A)

Output:

Set()

Access Items

You cannot access items in a set by referring to an index, since sets are unordered the items has no index.

But you can loop through the set items using a `for` loop, or ask if a specified value is present in a set, by using the `in` keyword.

Example

Loop through the set, and print the values:

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

for x in thisset:
  print(x)
```

Output:
apple
banana
cherry

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)
```

NOTE: Lists cannot be added to a set as elements because Lists are not hashable whereas Tuples can be added because tuples are immutable and hence Hashable.

Removing elements from the Set

Using remove() method or discard() method:

- Elements can be removed from the Set by using the built-in remove() function but a KeyError arises if the element doesn't exist in the set.
- To remove elements from a set without KeyError, use discard(), if the element doesn't exist in the set, it remains unchanged.

Note: *If the set is unordered then there's no such way to determine which element is popped by using the pop() function.*

Python Set in -built functions:

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Examples:

```
S1={1,2,3,4,5}
S1.remove(2)
print(S1)
```

Output:

{1,3,4,5}

```
S1={1,2,3,4}
S2={1,2,3,4,5}

print(S1.issubset(S2))
```

Output:

True

```
print(S2.issuperset(S1))
```

Output:

True

```
>>>
S1={1,2,3,4,5}
S2={5,6,7}
print(S1.union(S2)) # print(S1|S2)
```

Output:

{1,2,3,4,5,6,7}

```
>>>
print(S1.intersection(S2)) # print(S2&S2)
```

Output:

{5}

```
>>>
print(S1.difference(S2)) # print(S1-S2)
```

Output:

{1,2,3,4}

>>>
print(S1.symmetric_difference(S2)) # print(S1^S2)

Output:
{1,2,3,4,6,7}

Python Dictionary

- **Dictionary in Python** is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.
- 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.

Properties of Dictionary 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.

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

Accessing value to a dictionary

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets. . Following is a simple example –

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print("dict['Name']:",dict['Name'])
print("dict['Age']: ", dict['Age'])
```

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

```
dict['Name']:Zara
dict['Age']: 7
```

There is also a method called **get()** that will also help in accessing the element from a dictionary.This method accepts key as argument and returns the value.

```
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}
print("Accessing a element using get:")
print(Dict.get(3))
```

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 –

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

car.update({"color": "White"})

print(car)
```

```
{'brand': 'Ford', 'model':
'Mustang', 'year': 1964,
'color': 'White'}
```

Deleting Elements using del Keyword

The items of the dictionary can be deleted by using the del keyword

```
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

print("Dictionary =")
print(Dict)
#Deleting some of the Dictionar data
del(Dict[1])
```

```
print("Data after deletion Dictionary=")
print(Dict)
```

Output

```
Dictionary    ={1:    'Geeks',    'name':    'For',    3:    'Geeks'}
Data after deletion Dictionary={'name': 'For', 3: 'Geeks'}
```

Built-in Dictionary Functions & Methods

Python includes the following dictionary functions −

Sr.No.	Function with Description
1	**cmp(dict1, dict2)** Compares elements of both dict.
2	**len(dict)** Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	**str(dict)** Produces a printable string representation of a dictionary
4	**type(variable)** Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Dictionary methods

Method	Description
dic.clear()	Remove all the elements from the dictionary
dict.copy()	Returns a copy of the dictionary
dict.get(key, default = “None”)	Returns the value of specified key

dict.items()	Returns a list containing a tuple for each key value pair
dict.keys()	Returns a list containing dictionary's keys
dict.update(dict2)	Updates dictionary with specified key-value pairs
dict.values()	Returns a list of all the values of dictionary
pop()	Remove the element with specified key
popItem()	Removes the last inserted key-value pair
dict.setdefault(key,default= "None")	set the key to the default value if the key is not specified in the dictionary
dict.has_key(key)	returns true if the dictionary contains the specified key.
dict.get(key, default = "None")	used to get the value specified for the passed key.