Python Data Structures | Lists, Tuples and Sets

Lists in Python

Lists are one of the most powerful tools in python. They are just like the arrays declared in other languages. But the most powerful thing is that list need not be always homogenous. A single list can contain strings, integers, as well as objects. Lists can also be used for implementing stacks and queues. Lists are mutable, i.e., they can be altered once declared.

```
# Declaring a list
L = [1, "a" , "string" , 1+2]
print(L)
L.append(6)
print(L)
L.pop()
rint(L)
print(L[1])
```

The output is :

```
[1, 'a', 'string', 3]

[1, 'a', 'string', 3, 6]

[1, 'a', 'string', 3]

a
```

```
# Creating a List with
# the use of Numbers
# (Having duplicate values)
List = [1, 2, 4, 4, 3, 3, 3, 6, 5]
print("\nList with the use of Numbers: ")
print(List)

# Creating a List with
# mixed type of values
# (Having numbers and strings)
List = [1, 2, 'Geeks', 4, 'For', 6, 'Geeks']
print("\nList with the use of Mixed Values: ")
print(List)
```

Output:

```
List with the use of Numbers:

[1, 2, 4, 4, 3, 3, 3, 6, 5]
```

```
List with the use of Mixed Values:

[1, 2, 'Geeks', 4, 'For', 6, 'Geeks']
```

Knowing the size of List

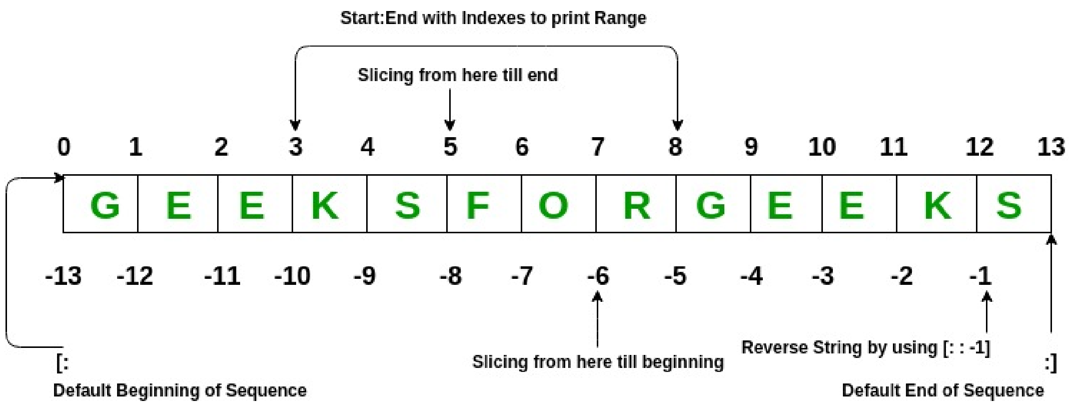
```python
# Creating a List
List1 = []
print(len(List1))

# Creating a List of numbers
List2 = [10, 20, 14]
print(len(List2))
```

Output:

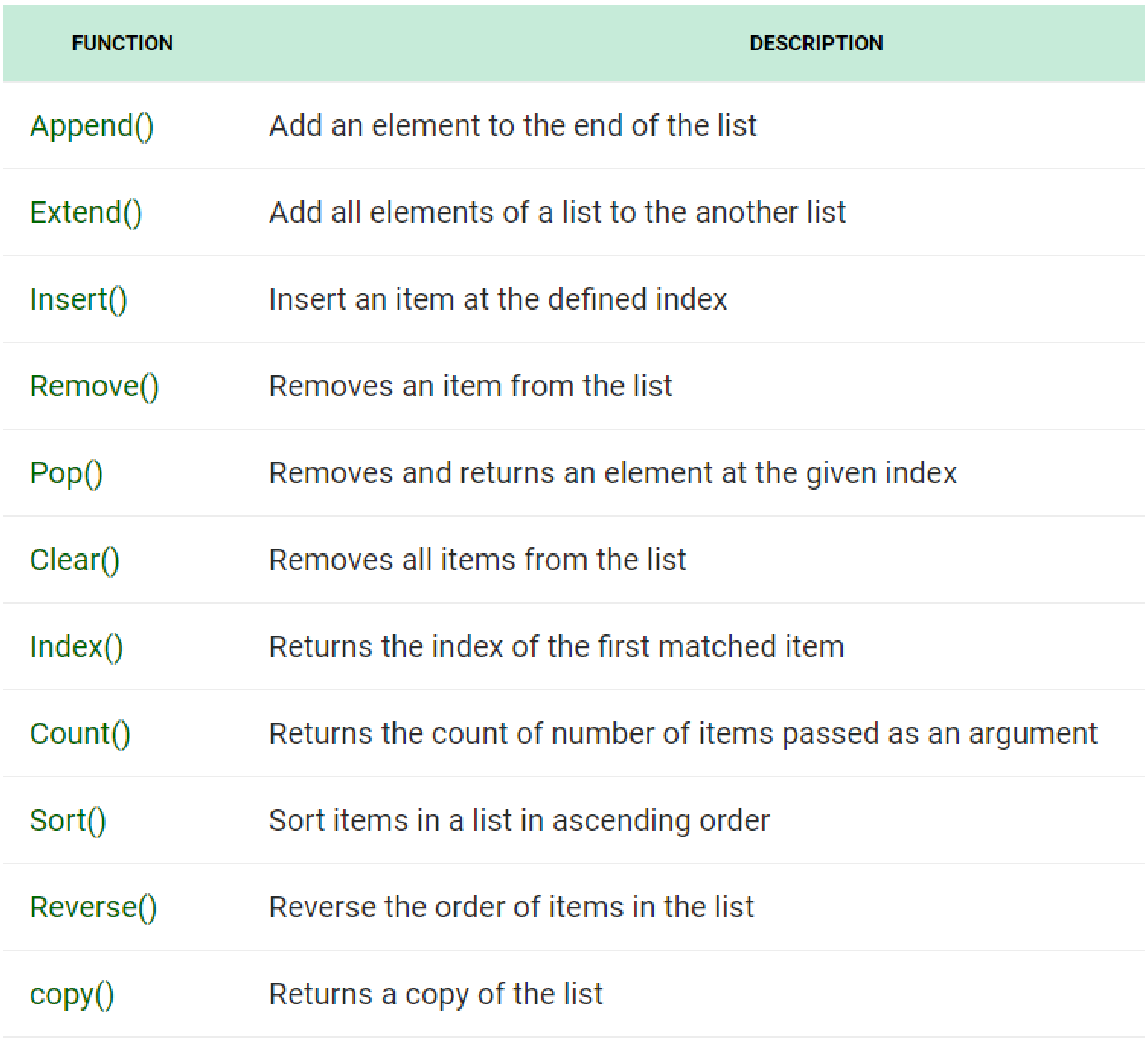
```
0

3
```

Slicing of a List

In Python List, there are multiple ways to print the whole List with all the elements, but to print a specific range of elements from the list, we use Slice operation. Slice operation is performed on Lists with the use of colon(:). To print elements from beginning to a range use [:Index], to print elements from end use [:-Index], to print elements from specific Index till the end use [Index:], to print elements within a range, use [Start Index:End Index] and to print whole List with the use of slicing operation, use [:]. Further, to print whole List in reverse order, use [::-1].
Note – To print elements of List from rear end, use Negative Indexes.


Start:End with Indexes to print Range
Slicing from here till end
0 1 2 3 4 5 6 7 8 9 10 11 12 13
G E E K S F O R G E E K S
-13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
[:
Default Beginning of Sequence
Slicing from here till beginning
Reverse String by using [: : -1]
:]
Default End of Sequence

List Methods



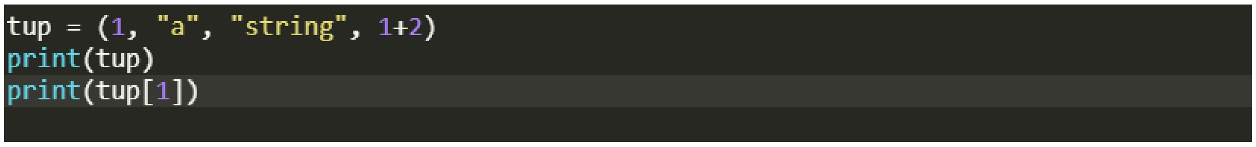
FUNCTION	DESCRIPTION
Append()	Add an element to the end of the list
Extend()	Add all elements of a list to the another list
Insert()	Insert an item at the defined index
Remove()	Removes an item from the list
Pop()	Removes and returns an element at the given index
Clear()	Removes all items from the list
Index()	Returns the index of the first matched item
Count()	Returns the count of number of items passed as an argument
Sort()	Sort items in a list in ascending order
Reverse()	Reverse the order of items in the list
copy()	Returns a copy of the list

Built-in functions with List

FUNCTION	DESCRIPTION
reduce()	apply a particular function passed in its argument to all of the list elements stores the intermediate result and only returns the final summation value
sum()	Sums up the numbers in the list
ord()	Returns an integer representing the Unicode code point of the given Unicode character
cmp()	This function returns 1, if first list is “greater” than second list
max()	return maximum element of given list
min()	return minimum element of given list
all()	Returns true if all element are true or if list is empty
any()	return true if any element of the list is true. if list is empty, return false
len()	Returns length of the list or size of the list
enumerate()	Returns enumerate object of list
accumulate()	apply a particular function passed in its argument to all of the list elements returns a list containing the intermediate results
filter()	tests if each element of a list true or not
map()	returns a list of the results after applying the given function to each item of a given iterable
lambda()	This function can have any number of arguments but only one expression, which is evaluated and returned.

Tuples in Python

A tuple is a sequence of immutable Python objects. Tuples are just like lists with the exception that tuples cannot be changed once declared. Tuples are usually faster than lists.



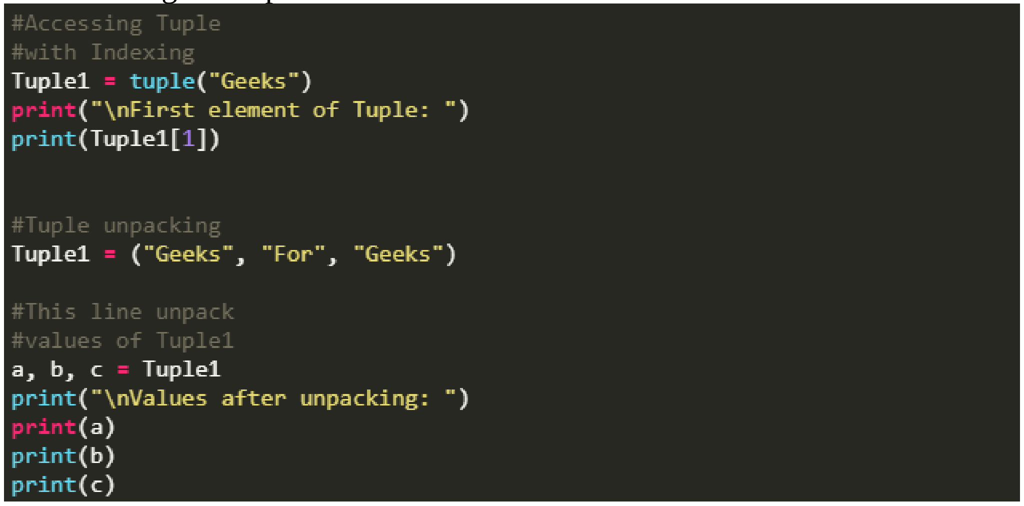
```
tup = (1, "a", "string", 1+2)
print(tup)
print(tup[1])
```

The output is :

```
(1, 'a', 'string', 3)

a
```

Accessing of Tuples

Tuples are immutable, and usually, they contain a sequence of heterogeneous elements that are accessed via unpacking or indexing (or even by attribute in the case of named tuples). Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.
NOTE : In unpacking of tuple number of variables on left hand side should be equal to number of values in given tuple a.



```
#Accessing Tuple
#with Indexing
Tuple1 = tuple("Geeks")
print("\nFirst element of Tuple: ")
print(Tuple1[1])


#Tuple unpacking
Tuple1 = ("Geeks", "For", "Geeks")

#This line unpack
#values of Tuple1
a, b, c = Tuple1
print("\nValues after unpacking: ")
print(a)
print(b)
print(c)
```

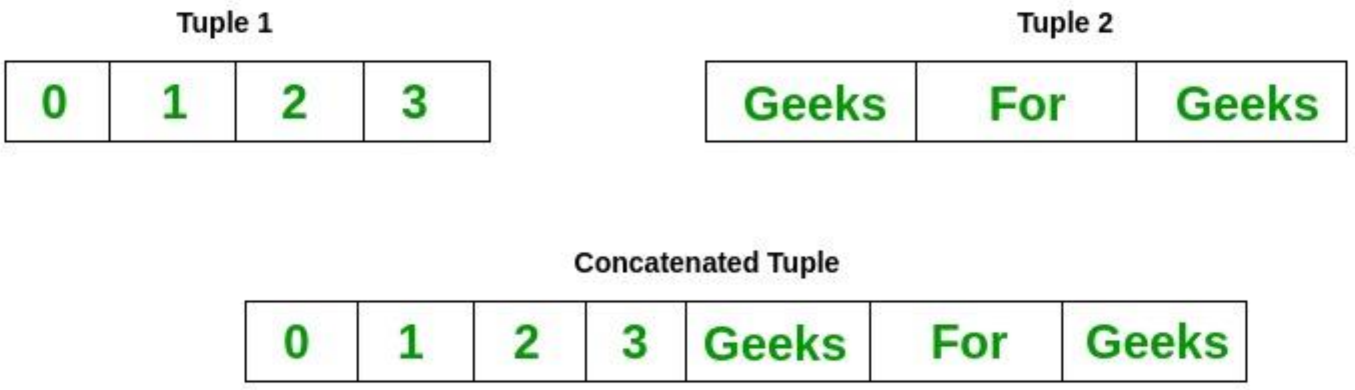
Output:

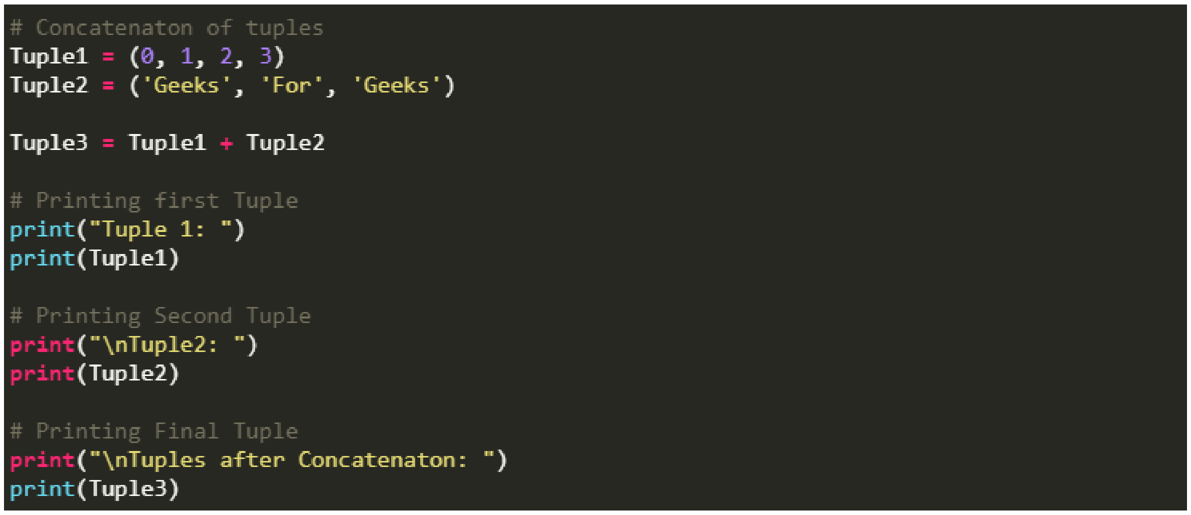
```
First element of Tuple:

e


Values after unpacking:

Geeks
```

```
For

Geeks
```

Concatenation of Tuples

Concatenation of tuple is the process of joining of two or more Tuples. Concatenation is done by the use of ‘+’ operator. Concatenation of tuples is done always from the end of the original tuple. Other arithmetic operations do not apply on Tuples.
Note- Only same datatypes can be combined with concatenation, an error arises if a list and a tuple are combined.





```python
# Concatenaton of tuples
Tuple1 = (0, 1, 2, 3)
Tuple2 = ('Geeks', 'For', 'Geeks')

Tuple3 = Tuple1 + Tuple2

# Printing first Tuple
print("Tuple 1: ")
print(Tuple1)

# Printing Second Tuple
print("\nTuple2: ")
print(Tuple2)

# Printing Final Tuple
print("\nTuples after Concatenaton: ")
print(Tuple3)
```

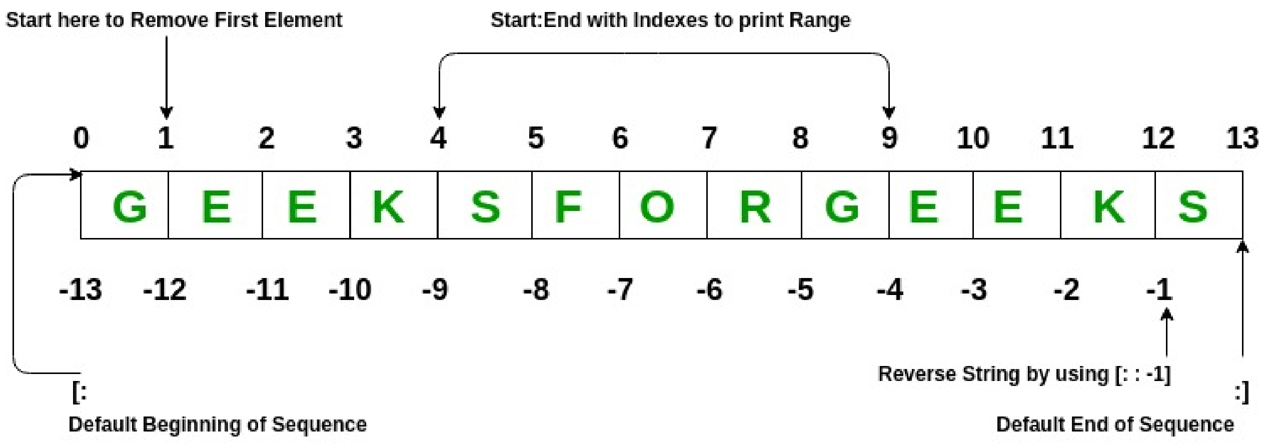
Output:

```
Tuple 1:

(0, 1, 2, 3)


Tuple2:

('Geeks', 'For', 'Geeks')
```

```
Tuples after Concatenaton:
(0, 1, 2, 3, 'Geeks', 'For', 'Geeks')
```

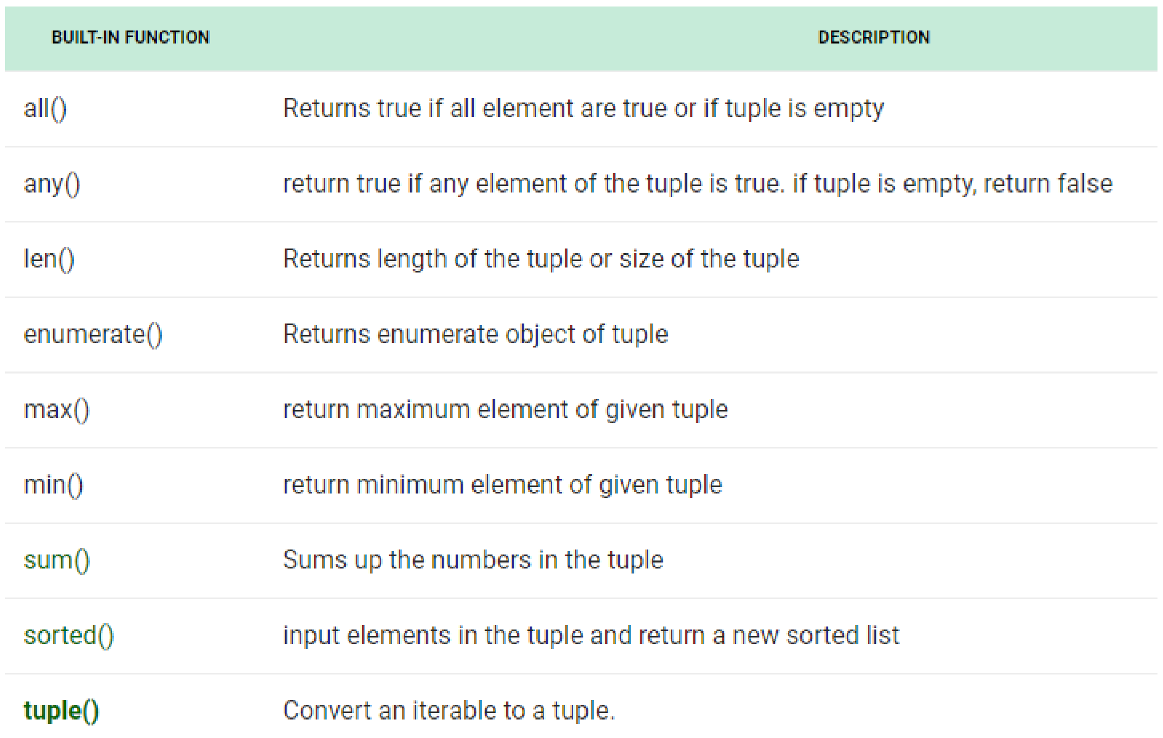
Slicing of Tuple

Slicing of a Tuple is done to fetch a specific range or slice of sub-elements from a Tuple. Slicing can also be done to lists and arrays. Indexing in a list results to fetching a single element whereas Slicing allows to fetch a set of elements.
Note- Negative Increment values can also be used to reverse the sequence of Tuples



Start here to Remove First Element
Start:End with Indexes to print Range
0 1 2 3 4 5 6 7 8 9 10 11 12 13
G E E K S F O R G E E K S
-13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
Reverse String by using [: : -1]
[:
Default Beginning of Sequence
:]
Default End of Sequence

Built-In Methods



BUILT-IN FUNCTION	DESCRIPTION
all()	Returns true if all element are true or if tuple is empty
any()	return true if any element of the tuple is true. if tuple is empty, return false
len()	Returns length of the tuple or size of the tuple
enumerate()	Returns enumerate object of tuple
max()	return maximum element of given tuple
min()	return minimum element of given tuple
sum()	Sums up the numbers in the tuple
sorted()	input elements in the tuple and return a new sorted list
tuple()	Convert an iterable to a tuple.

Python Sets

In Python, **Set** is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.
The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.

Creating a Set

Sets can be created by using the built-in `set()` function with an iterable object or a sequence by placing the sequence inside curly braces, separated by ‘comma’.

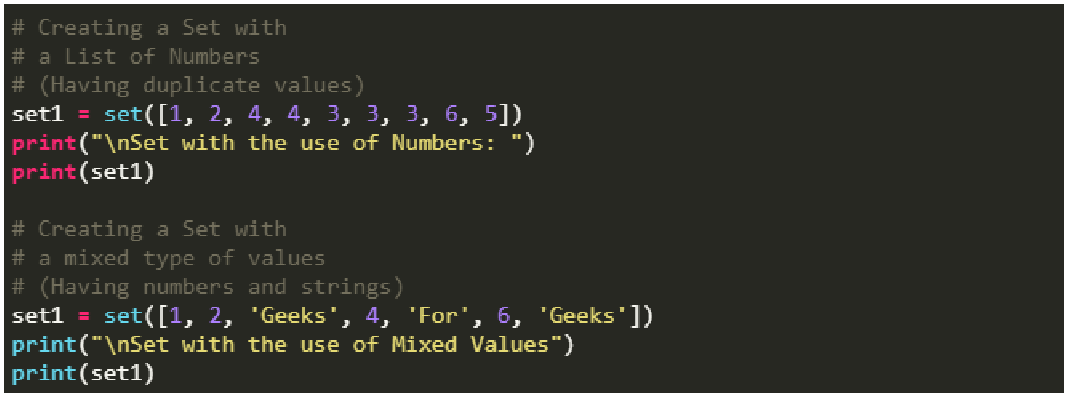
Note – A set cannot have mutable elements like a list, set or dictionary, as its elements.

```python
# Python program to demonstrate
# Creation of Set in Python

# Creating a Set
set1 = set()
print("Intial blank Set: ")
print(set1)

# Creating a Set with
# the use of a String
set1 = set("GeeksForGeeks")
print("\nSet with the use of String: ")
print(set1)

# Creating a Set with
# the use of Constructor
# (Using object to Store String)
String = 'GeeksForGeeks'
set1 = set(String)
print("\nSet with the use of an Object: " )
print(set1)

# Creating a Set with
# the use of a List
set1 = set(["Geeks", "For", "Geeks"])
print("\nSet with the use of List: ")
print(set1)
```

Output:

```
Intial blank Set:

set()


Set with the use of String:

{'e', 'r', 'k', 'o', 'G', 's', 'F'}


Set with the use of an Object:

{'r', 'o', 'e', 'F', 's', 'k', 'G'}


Set with the use of List:

{'Geeks', 'For'}
```

A set contains only unique elements but at the time of set creation, multiple duplicate values can also be passed. Order of elements in a set is undefined and is unchangeable. Type of elements in a set need not be the same, various mixed up data type values can also be passed to the set.



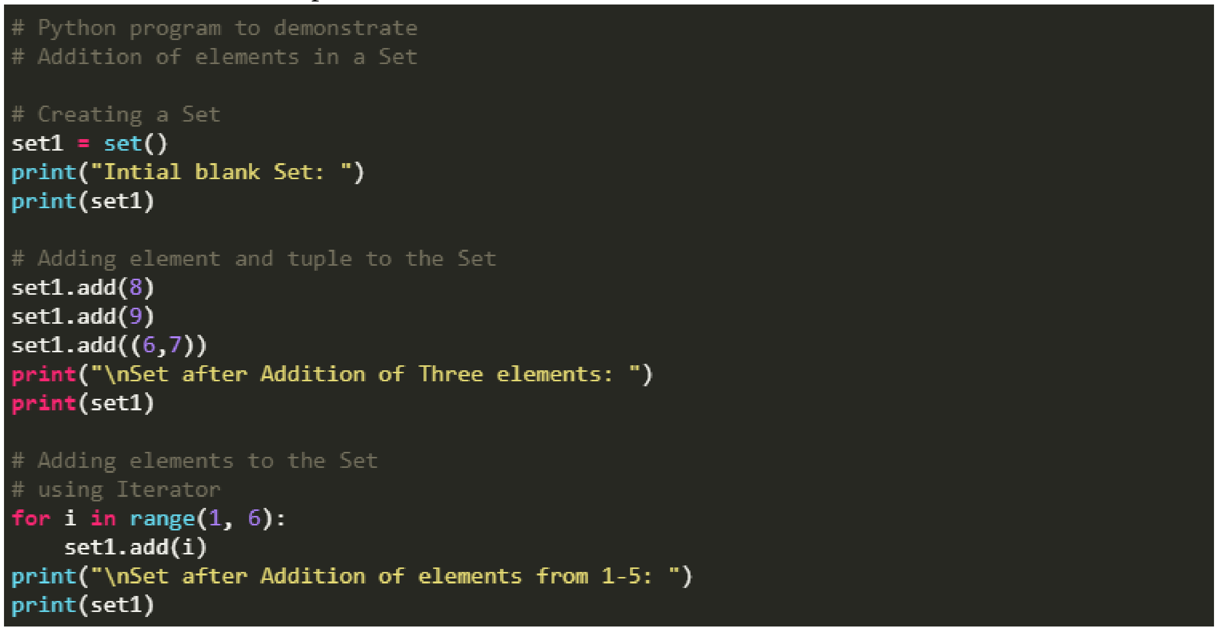
```python
# Creating a Set with
# a List of Numbers
# (Having duplicate values)
set1 = set([1, 2, 4, 4, 3, 3, 3, 6, 5])
print("\nSet with the use of Numbers: ")
print(set1)

# Creating a Set with
# a mixed type of values
# (Having numbers and strings)
set1 = set([1, 2, 'Geeks', 4, 'For', 6, 'Geeks'])
print("\nSet with the use of Mixed Values")
print(set1)
```

Output:

```
Set with the use of Numbers:

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



Set with the use of Mixed Values

{1, 2, 4, 'Geeks', 6, 'For'}
```

Adding Elements to a Set

Using add() method
Elements can be added to the Set by using built-in **add()** function. Only one element at a time can be added to the set by using add() method, loops are used to add multiple elements at a time with the use of add() method.
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.



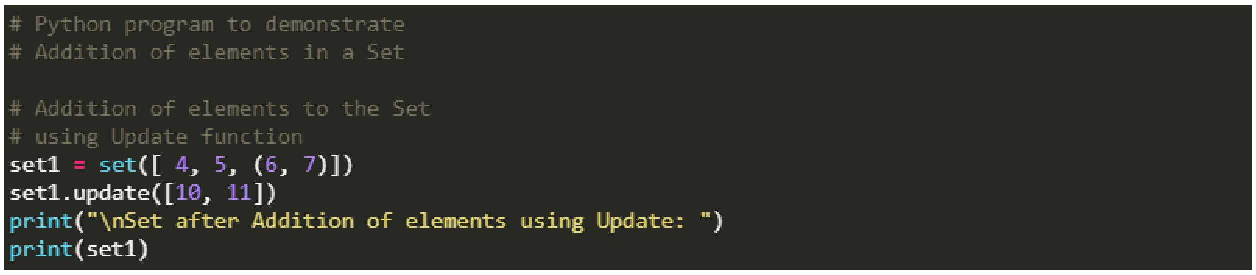
```python
# Python program to demonstrate
# Addition of elements in a Set

# Creating a Set
set1 = set()
print("Intial blank Set: ")
print(set1)

# Adding element and tuple to the Set
set1.add(8)
set1.add(9)
set1.add((6,7))
print("\nSet after Addition of Three elements: ")
print(set1)

# Adding elements to the Set
# using Iterator
for i in range(1, 6):
    set1.add(i)
print("\nSet after Addition of elements from 1-5: ")
print(set1)
```

Output:

```
Intial blank Set:

set()


Set after Addition of Three elements:

{8, 9, (6, 7)}


Set after Addition of elements from 1-5:

{1, 2, 3, (6, 7), 4, 5, 8, 9}
```

Using update() method

For addition of two or more elements Update() method is used. The update() method accepts lists, strings, tuples as well as other sets as its arguments. In all of these cases, duplicate elements are avoided.



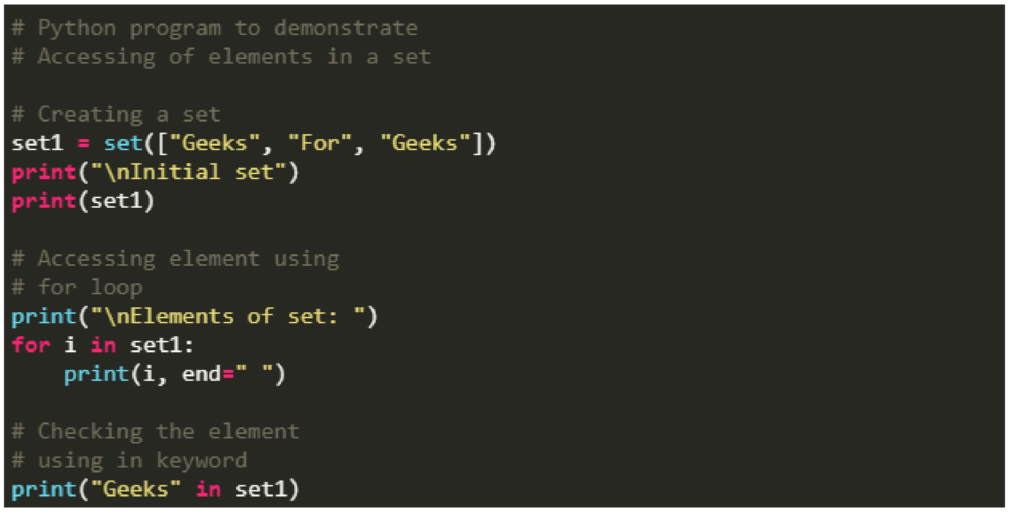
```
# Python program to demonstrate
# Addition of elements in a Set

# Addition of elements to the Set
# using Update function
set1 = set([ 4, 5, (6, 7)])
set1.update([10, 11])
print("\nSet after Addition of elements using Update: ")
print(set1)
```

Output:

```
Set after Addition of elements using Update:

{10, 11, 4, 5, (6, 7)}
```

Accessing a Set

Set items cannot be accessed 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.



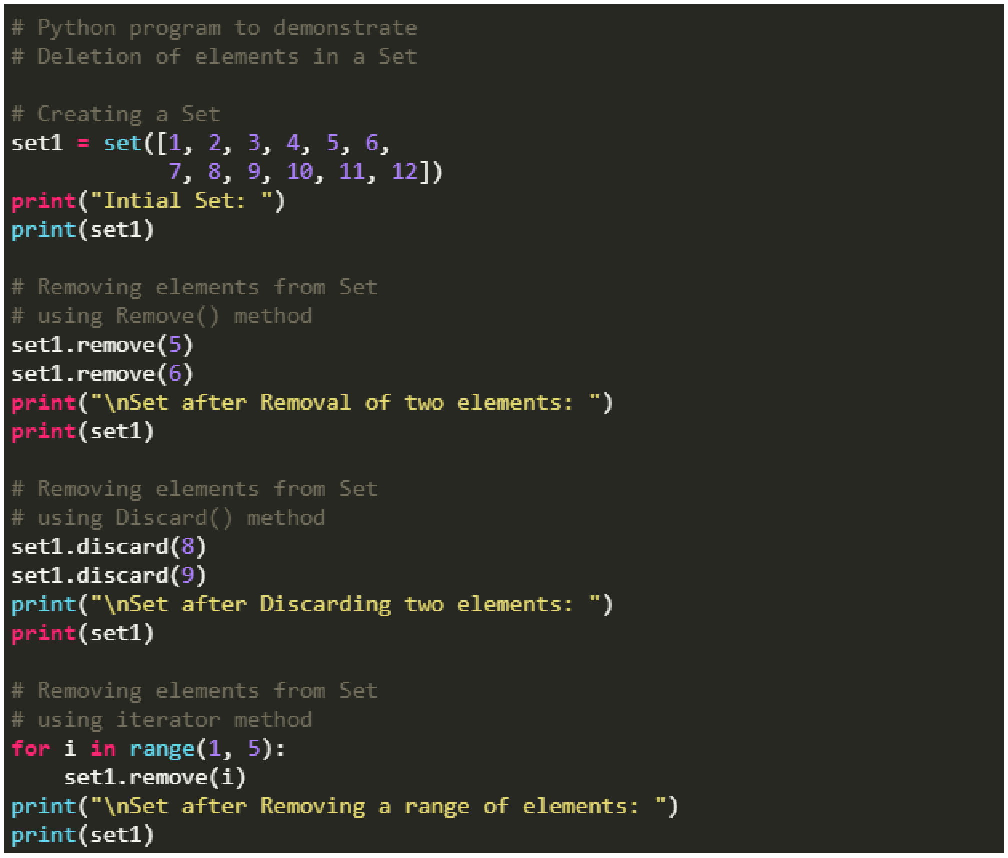
```python
# Python program to demonstrate
# Accessing of elements in a set

# Creating a set
set1 = set(["Geeks", "For", "Geeks"])
print("\nInitial set")
print(set1)

# Accessing element using
# for loop
print("\nElements of set: ")
for i in set1:
    print(i, end=" ")

# Checking the element
# using in keyword
print("Geeks" in set1)
```

Output:

```
Initial set:

{'Geeks', 'For'}


Elements of set:

Geeks For


True
```

Removing elements from the Set

Using remove() method or discard() method
Elements can be removed from the Set by using built-in remove() function but a KeyError arises if 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.



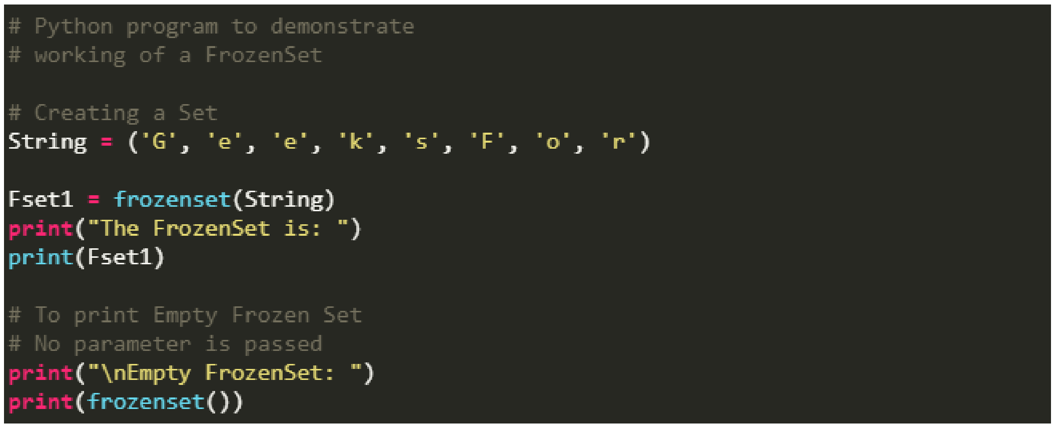
```python
# Python program to demonstrate
# Deletion of elements in a Set

# Creating a Set
set1 = set([1, 2, 3, 4, 5, 6,
            7, 8, 9, 10, 11, 12])
print("Intial Set: ")
print(set1)

# Removing elements from Set
# using Remove() method
set1.remove(5)
set1.remove(6)
print("\nSet after Removal of two elements: ")
print(set1)

# Removing elements from Set
# using Discard() method
set1.discard(8)
set1.discard(9)
print("\nSet after Discarding two elements: ")
print(set1)

# Removing elements from Set
# using iterator method
for i in range(1, 5):
    set1.remove(i)
print("\nSet after Removing a range of elements: ")
print(set1)
```

Output:

```
Intial Set:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}


Set after Removal of two elements:
{1, 2, 3, 4, 7, 8, 9, 10, 11, 12}


Set after Discarding two elements:
{1, 2, 3, 4, 7, 10, 11, 12}


Set after Removing a range of elements:
{7, 10, 11, 12}
```

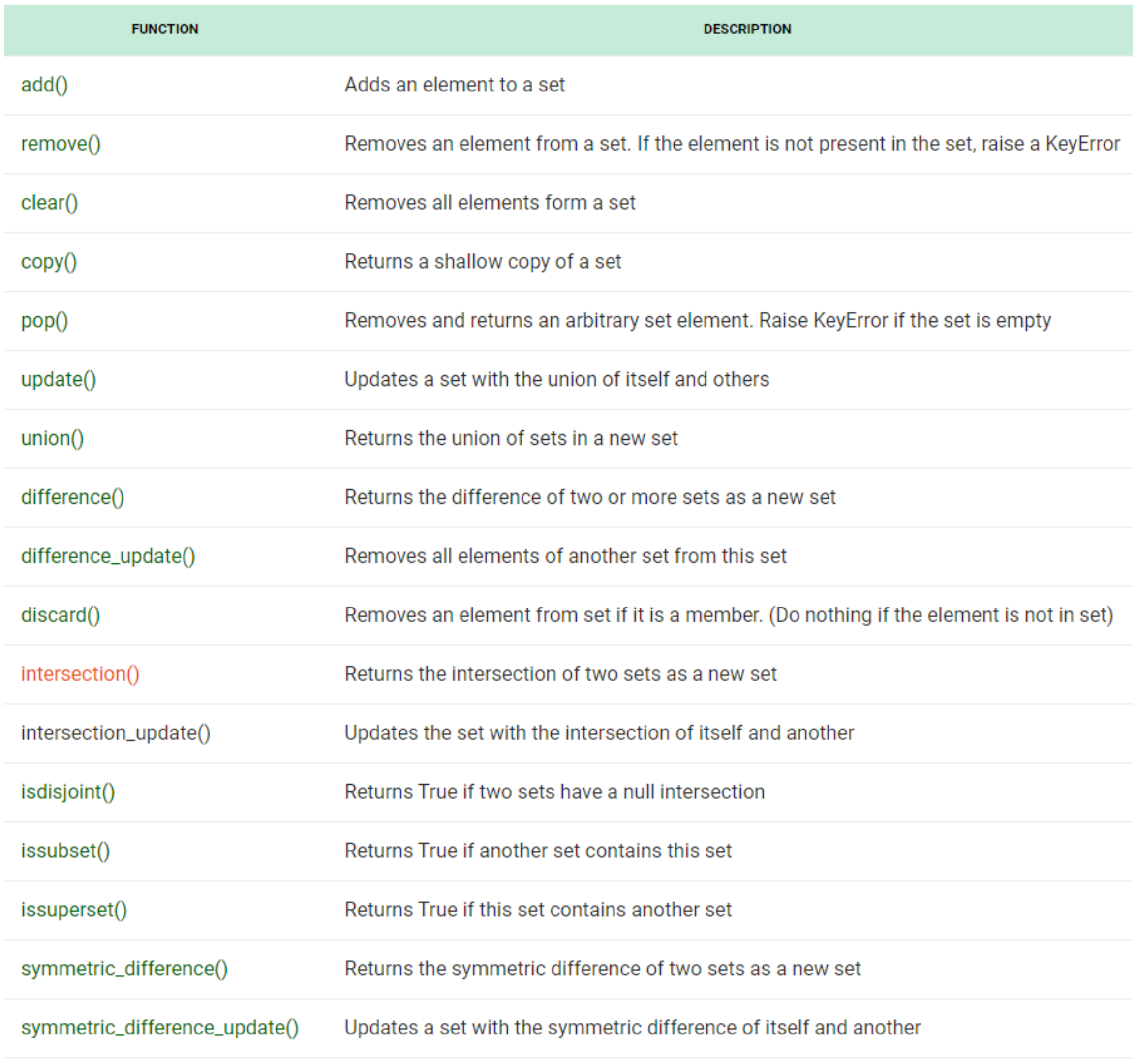
Frozen sets in Python are immutable objects that only support methods and operators that produce a result without affecting the frozen set or sets to which they are applied. While elements of a set can be modified at any time, elements of the frozen set remain the same after

creation.
If no parameters are passed, it returns an empty frozenset.



```python
# Python program to demonstrate
# working of a FrozenSet

# Creating a Set
String = ('G', 'e', 'e', 'k', 's', 'F', 'o', 'r')

Fset1 = frozenset(String)
print("The FrozenSet is: ")
print(Fset1)

# To print Empty Frozen Set
# No parameter is passed
print("\nEmpty FrozenSet: ")
print(frozenset())
```

Set Methods



FUNCTION	DESCRIPTION
add()	Adds an element to a set
remove()	Removes an element from a set. If the element is not present in the set, raise a KeyError
clear()	Removes all elements form a set
copy()	Returns a shallow copy of a set
pop()	Removes and returns an arbitrary set element. Raise KeyError if the set is empty
update()	Updates a set with the union of itself and others
union()	Returns the union of sets in a new set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns True if two sets have a null intersection
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another

Revision questions

1. Write a Python program to sum all the items in a list. Go to the editor
Click me to see the sample solution

2. Write a Python program to multiplies all the items in a list. Go to the editor
Click me to see the sample solution

3. Write a Python program to get the largest number from a list. Go to the editor
Click me to see the sample solution

4. Write a Python program to create a tuple. Go to the editor

Click me to see the sample solution

5. Write a Python program to create a tuple with different data types. Go to the editor. Click me to see the sample solution

6. Write a Python program to create a tuple with numbers and print one item. Go to the editor.
Click me to see the sample solution