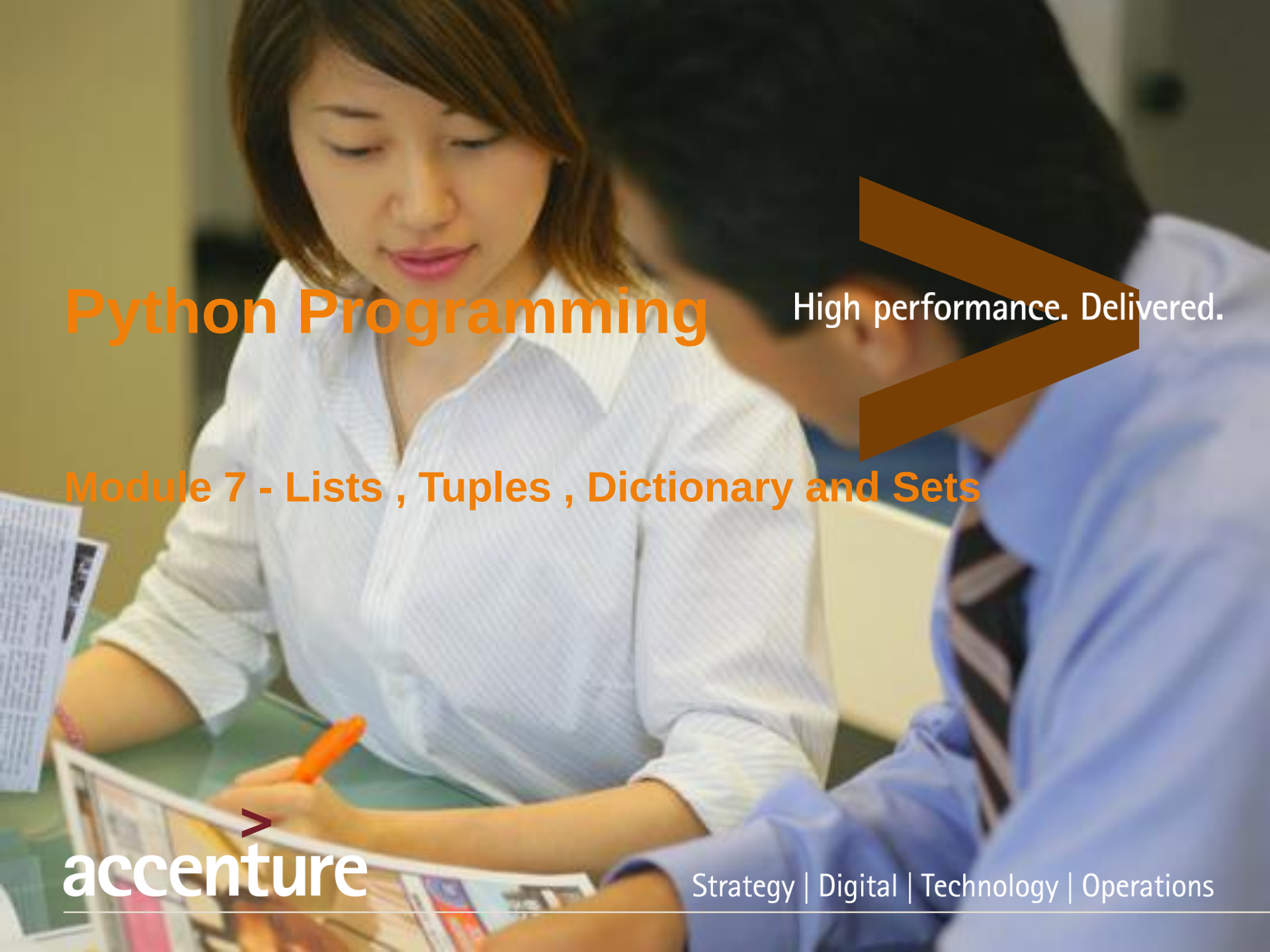
Python Programming
High performance. Delivered.
Module 7 - Lists , Tuples , Dictionary and Sets
accenture
Strategy | Digital | Technology | Operations

Agenda

- Lists
- Tuples
- Dictionary
- Sets

Module Objectives

At the end of this module, you will be able to understand

- Lists and its operations
- Tuples and its operations
- Dictionary and its operations
- Sets and its operations.

Sequences in Python

Python offers a range of compound datatypes often referred to as sequences.

Python provides six sequence (or sequential) data types:

- strings
- byte sequences
- byte arrays
- lists
- tuples
- range objects

- Strings, lists, tuples, bytes and range objects may look like different things, but they have some underlying concepts in common:
- The items or elements of strings, lists and tuples are ordered in a defined sequence
- The elements can be accessed via indices

Lists in Python

- List is one of the most frequently used and very versatile datatype in Python.

Creating a List in Python

- In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).
- A list can even have another list as an item. These are called nested list.

```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Accenture", 3.4]

# nested list
my_list = ["Accenture", [8, 4, 6]]
```

Accessing Elements in a List

There are various ways in which we can access the elements of a list.

- Indexing
- Negative indexing
- Slicing

Indexing

- We can use the index operator [] to access an item in a list.
- Index starts from 0.
- Trying to access an element with wrong index number will raise an IndexError.
- The index must be an integer. (trying to user other types as index, will result into TypeError) Nested list are accessed using nested indexing.

```
mylist = ['A','c', 'c', 'e', 'n', 't', 'u', 'r', 'e']
print("mylist[0] = ",mylist[0])   # Displays 'A'
print("mylist[2] = ",mylist[2])   # Displays 'c'
print("mylist[4] = ",mylist[4])   # Displays 'n'
print("mylist[4.1] = ",mylist[4.1])  #TypeError: list indices must be integers, not float
print("mylist[9] = ",mylist[9])   # IndexError: list index out of range

mylist = ["Happy", [2,0,1,5]]
print("mylist[0][1] = ",mylist[0][1]);   # nested indexing    Displays 'a'
print("mylist[1][3]   ",mylist[1][3]);   # nested indexing    Displays '5'
```

Negative indexing

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

```
mylist = ['A','c', 'c', 'e', 'n', 't', 'u', 'r', 'e']

print("mylist[-1] = ",mylist[-1]);          # Displays 'e'
print("mylist[-5] = ",mylist[-5]);          # Displays 'n'
```

Slicing

We can access a range of items in a list by using the slicing operator (colon).

```
mylist = ['A','c','c','e','n','t','u','r','e']

print("mylist[2:5] = ",mylist[2:5]);    # elements 3rd to 5th
print("mylist[:-5] = ",mylist[:-5]);    # elements beginning to 4th
print("mylist[5:]  = ",mylist[5:]);     # elements 6th to end
print("mylist[:]   = ",mylist[:]);      # elements beginning to end.
```

OUTPUT

```
mylist[2:5] =  ['c', 'e', 'n']
mylist[:-5] =  ['A', 'c', 'c', 'e']
mylist[5:]  =  ['t', 'u', 'r', 'e']
mylist[:]   =  ['A', 'c', 'c', 'e', 'n', 't', 'u', 'r', 'e']
```

Changing or Adding Elements to a List

- Lists are mutable, meaning, their elements can be changed unlike string or tuple.
- We can use assignment operator (=) to change an item or a range of items.

```
evennumbers = [2, 4, 5, 8, 10, 12]    # mistake values
print("EVENNUMBERS = ",evennumbers);
evennumbers[2] = 6            # change the 3rd item
print("EVENNUMBERS = ",evennumbers);
evennumbers [1:4] = [3, 5, 7]  # change 2nd to 4th items
print("EVENNUMBERS = ",evennumbers);
```

OUTPUT

```
EVENNUMBERS =  [2, 4, 5, 8, 10, 12]
EVENNUMBERS =  [2, 4, 6, 8, 10, 12]
EVENNUMBERS =  [2, 3, 5, 7, 10, 12]
```

Adding Elements using append() or extend() methods

We can add one item to a list using append() method or add several items using extend() method.

```
evennumbers =  [2, 3, 5, 7, 10, 12]
evennumbers.append(7)
print("EVENNUMBERS = ",evennumbers);

evennumbers.extend([9, 11, 13])
print("EVENNUMBERS = ",evennumbers);
```

```
OUTPUT
EVENNUMBERS =  [2, 3, 5, 7, 10, 12, 7]
EVENNUMBERS =  [2, 3, 5, 7, 10, 12, 7, 9, 11, 13]
```

+ and * Operators

- The + operator is used to combine two lists. This is also called concatenation.
- The * operator repeats a list for the given number of times.

```
evennumbers = [2, 3, 5]
print("EVENNUMBERS = ",evennumbers);

evennumbers = evennumbers + [9, 7, 5]
print("EVENNUMBERS = ",evennumbers);

evennumbers = evennumbers * 2
print("EVENNUMBERS = ",evennumbers);

print(['Acc','Acc','Acc'] * 3);
```

```
OUTPUT
EVENNUMBERS =  [2, 3, 5]
EVENNUMBERS =  [2, 3, 5, 9, 7, 5]
EVENNUMBERS =  [2, 3, 5, 9, 7, 5, 2, 3, 5, 9, 7, 5]
['Acc', 'Acc', 'Acc', 'Acc', 'Acc', 'Acc', 'Acc', 'Acc', 'Acc']
```

insert() Method

Insert() method can be used to insert one item at a desired location or insert multiple items by squeezing it into an empty slice of a list.

```
MyList = [10,20,30,40,50]
print("MyList = ",MyList);
MyList.insert(1,3)            # inserts 3 at index 1
print("MyList = ",MyList);
MyList[2:2] = [555, 777]      # inserts the two elements 555 and 777 starting index 2.
print("MyList = ",MyList);
```

OUTPUT

```
MyList =  [10, 20, 30, 40, 50]
MyList =  [10, 3, 20, 30, 40, 50]
MyList =  [10, 3, 555, 777, 20, 30, 40, 50]
```

Deleting or Removing Elements from a List

Del Keyword

- We can delete one or more items from a list using the keyword del.
- It can even delete the list entirely.

```
MyList = [10,20,30,40,50,60,70,80,90,100]
print("MyList = ",MyList);
del MyList [2]              # delete one item
print("MyList = ",MyList);
del MyList [1:5]            # delete multiplte items
print("MyList = ",MyList);
del MyList                  # delete entire list
print("MyList = ",MyList);   # Displays NameError: name 'MyList' is not defined
```

OUTPUT

```
MyList =  [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
MyList =  [10, 20, 40, 50, 60, 70, 80, 90, 100]
MyList =  [10, 70, 80, 90, 100]
NameError: name 'MyList' is not defined
```

remove() , pop() and clear() methods

- remove() method removes the given item
- pop() method removes an item at the given index.
- The pop(i) method removes and returns the last item if index(i) is not provided.
- This helps us implement lists as stacks (first in, last out data structure).
- We can also use the clear() method to empty a list.

```
MyList =  [10, 70, 80, 90, 100]
print("MyList = ",MyList);
MyList.remove(80)                    # Removes the element 80
print("MyList = ",MyList);
print(MyList.pop(1));                # Removes the element at index 1
print("MyList = ",MyList);
print("MyList.pop() = ",MyList.pop());    # Removes the Last element and Returns that
element
MyList.clear()                              # Clears all the elements
print("MyList = ",MyList);
```

OUTPUT

```
MyList =  [10, 70, 80, 90, 100]
MyList =  [10, 70, 90, 100]
70
MyList =  [10, 90, 100]
MyList.pop() =  100
MyList =  []
```

Deleting by slicing

- We can also delete items in a list by assigning an empty list to a slice of elements.

```
MyList =  [10, 70, 80, 90, 100]
print("MyList = ",MyList);
MyList[2:3] = [];                    # Assigns an empty
list at index 2
print("MyList = ",MyList);
```

```
OUTPUT
MyList =  [10, 70, 80, 90, 100]
MyList =  [10, 70, 90, 100]
```

Python List Methods

Methods that are available with list object are displayed below

Method	Description
append(x)	Add item x at the end of the list
extend(L)	Add all items in given list L to the end
insert(i, x)	Insert item x at position i
remove(x)	Remove first item that is equal to x, from the list
pop([i])	Remove and return item at position i (last item if i is not provided)
clear()	Remove all items and empty the list
index(x)	Return index of first item that is equal to x
count(x)	Return the number of items that is equal to x
sort()	Sort items in a list in ascending order
reverse()	Reverse the order of items in a list
copy()	Return a shallow copy of the list

Python List Comprehension

List comprehension is an elegant and concise way to create new list from an existing list in Python. List comprehension consists of an expression followed by for statement inside square brackets. Here is an example to make a list with each item being increasing power of 2.

```
MyList = [2 ** x for x in range(10)]
print("MyList = ",MyList);
```

```
OUTPUT
MyList =  [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

This code is equivalent to

```
MyList = []
for x in range(10):
   MyList.append(2 ** x)
```

Python List Comprehension (Contd)

A list comprehension can optionally contain more for or if statements.

An optional if statement can filter out items for the new list.

```
MyList = [2 ** x for x in range(10) if x % 3 == 0 ]
print("MyList = ",MyList);

MyList = [x+y for x in ['Accenture ', 'Consulting '] for y
in ['Bangalore','Tokyo']]
print("MyList = ",MyList);
```

```
OUTPUT
MyList =  [1, 8, 64, 512]
MyList =  ['Accenture Bangalore', 'Accenture Tokyo', 'Consulting Bangalore', 'Consulting Tokyo']
```

Other List Operations – List Membership Test

We can test whether an item exists in a list or not, using the keyword **'in'.**

```
MyList = [10,20,33,44,55,60,70,88];
print("MyList = ",MyList);
print("Check whether 44 exists in the List?  ",44 in MyList);
print("Check whether 99 exists in the List?  ",99 in MyList);
print("Check whether 44 exists in the List?  ",44 not in MyList);
print("Check whether 44 exists in the List?  ",56 not in MyList);
```

OUTPUT

```
MyList =  [10, 20, 33, 44, 55, 60, 70, 88]
Check whether 44 exists in the List?   True
Check whether 99 exists in the List?   False
Check whether 44 exists in the List?   False
Check whether 44 exists in the List?   True
```

Iterating Through a List

Using a for loop we can iterate though each item in a list.

```
MyList = ['Apple','Banana','Mango']
for fruit in MyList:
        print("I like ",fruit)
```

```
OUTPUT
I like  Apple
I like  Banana
I like  Mango
```

Built-in Functions with List

Following are the commonly used methods with lists

Function	Description
all()	Return True if all elements of the list are true (or if the list is empty).
any()	Return True if any element of the list is true. If the list is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of list as a tuple.
len()	Return the length (the number of items) in the list.
list()	Convert an iterable (tuple, string, set, dictionary) to a list.
max()	Return the largest item in the list.
min()	Return the smallest item in the list
sorted()	Return a new sorted list (does not sort the list itself).
sum()	Retrun the sum of all elements in the list.

Tuples in Python

- A tuple is a sequence of immutable Python objects.
- Tuples are sequences, just like lists.

Difference between tuples and lists

Tuples	Lists
Tuples cannot be changed	Lists can be changed
Tuples use parenthesis ()	Lists use square brackets []

Creating a Tuple

- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma.
- The parentheses are optional but is a good practice to write it.
- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).

• Example:

mytuple = ()	**# empty tuple**
mytuple = (11, 22, 33)	# tuple having integers
mytuple = (45, "Hello", 9.5, “Accenture)	# tuple with mixed datatypes
mytuple = ("mouse", [8, 4, 6], (1, 2, 3))	# nested tuple
mytuple = 3, 4.6, "dog"	# tuple can be created without parentheses ,also called tuple packing
a, b, c = mytuple	# tuple unpacking is also possible

Creating a tuple with one element

Creating a tuple with one element is a bit tricky.

It needs a trailing comma to indicate that it is tuple and not a string.

Program	OutPut	Remarks
mytuple = ("hello"); **print("Type = ",type(mytuple));**	Type = <class 'str'>	Treated as a String
mytuple = ("hello",); **print("Type = ",type(mytuple));**	Type = <class 'tuple'>	Treated as a Tuple. Note the extra comma at the end.
mytuple = "hello",; **print("Type = ",type(mytuple));**	Type = <class 'tuple'>	Treated as a Tuple.Note the extra comma at the end. Parenthesis is optional.

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

Accessing Elements in a Tuple

There are various ways in which we can access the elements of a tuple.

- Indexing
- Negative Indexing
- Slicing

Indexing

- Use the index operator [] to access an item in a tuple.
- Index starts from 0.
- Trying to access an element other that this will raise an IndexError.
- The index must be an integer. Trying to use float or other types, will result in TypeError.
- Nested tuple are accessed using nested indexing

```
mytuple = ('A','C','C','E','N','T','U','R','E');
```

Program	OutPut
print("First Element = ",mytuple[0]);	First Element = A
#print("Sixth Element = ",mytuple[5]);	Sixth Element = T
#print("Tenth Element = ",mytuple[9]);	IndexError: tuple index out of range
print("Wrong Index = ",mytuple[2.0]);	TypeError: tuple indices must be integers, not float
newTuple=("mouse", [8, 4, 6], (1, 2, 3)); **print("newtuple[0][3] = ",newTuple[0][3]);**	newtuple[0][3] = s (Nested Index)

Negative Indexing

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

Example:

```
mytuple = ('A','C','C','E','N','T','U','R','E');
print("Last Element = ",mytuple[0]);
print("Third Last Element = ",mytuple[-3]);
newTuple =  ("PYTHON", [8, 4, 6], (1, 2, 3));
print("newtuple[-1][-3] = ",newTuple[-1][-3]);  # nested index
print("newtuple[-3][-4] = ",newTuple[-3][-4]);  # nested index
```

OUTPUT

```
Last Element =  A
Third Last Element =  U
newtuple[-1][-3] =  1
newtuple[-3][-4] =  T
```

Slicing

A range of items in a tuple can be accessed by using the slicing operator (colon).

- Example:

```
mytuple = ('A','C','C','E','N','T','U','R','E');
print("mytuple[1:5] Elements = ",mytuple[1:5]);
print("mytuple[-2:-6] Elements = ",mytuple[-2:-6]);
print("mytuple[:-6] Elements = ",mytuple[:-6]);  # Beginning to Until -6( -6 not included)
print("mytuple[-6:] Elements = ",mytuple[-6:]);  # Starting -6 till end

newTuple =  ("PYTHON", [8, 4, 6],"PROGRAMMING", (1, 2, 3),"HELLO");
print("newtuple[1:4] = ",newTuple[1:4]);  # From Second element until the 4th Element
```

OUTPUT

```
mytuple[1:5] Elements =  ('C', 'C', 'E', 'N')
mytuple[-2:-6] Elements =  ()
mytuple[:-6] Elements =  ('A', 'C', 'C')
mytuple[-6:] Elements =  ('E', 'N', 'T', 'U', 'R', 'E')
newtuple[1:4] =  ([8, 4, 6], 'PROGRAMMING', (1, 2, 3))
```

Updating Tuples

Tuples are immutable which means you cannot update or change the values of tuple elements.

It is possible to take portions of existing tuples to create new tuples.

```
tuple1 = (30, 40 ,50 ,60 ,70);
tuple[3]=200;     # This is an invalid operation.

Example:
tuple1 = (30, 40 ,50 ,60 ,70);
tuple2 = ('abc', 'def', 'lmn', 'xyz');

tuple3 = tuple1 + tuple2;   # creating a new tuple from the existing two
tuples
print("Tuple1 = ",tuple1);
print("Tuple2 = ",tuple2);
print("Tuple3 = ",tuple3);
```

```
OUTPUT
Tuple1 =  (30, 40, 50, 60, 70)
Tuple2 =  ('abc', 'def', 'lmn', 'xyz')
Tuple3 =  (30, 40, 50, 60, 70, 'abc', 'def', 'lmn', 'xyz')
```

Delete Tuple Elements

Tuples are immutable.

Individual Elements cannot be removed from the tuple.

Entire tuple can be deleted

We can remove the unwanted element by creating another tuple with the undesired elements discarded.

Example

```
mytuple = (10,20,30, 40 ,50 ,60 ,70, 80, 90, 100);
# Suppose we want to delete the 6th Element i.e. 60
tuple1 = mytuple[:5];
tuple2 = mytuple[6:];
mynewtuple = tuple1 + tuple2; # creating a new tuple discarding the unwanted element
print("MYTUPLE = ",mytuple);
print("Tuple1 = ",tuple1);
print("Tuple2 = ",tuple2);
print("MYNEWTUPLE = ",mynewtuple);
```

```
OUTPUT
MYTUPLE =  (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
Tuple1 =  (10, 20, 30, 40, 50)
Tuple2 =  (70, 80, 90, 100)
MYNEWTUPLE =  (10, 20, 30, 40, 50, 70, 80, 90, 100)
```

Deleting the Entire Tuple - Example

Deleting the entire tuple is done by using the ***del*** keyword

```
mytuple = (10,20,30, 40 ,50 ,60 ,70, 80, 90, 100);
print("MYTUPLE = ",mytuple);
del mytuple          # Deleting the entire Tuple
print("MYTUPLE = ",mytuple);   # Trying to access the Tuple after deleting
```

OUTPUT

```
MYTUPLE =  (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
print("MYTUPLE = ",mytuple);   # Trying to access the Tuple
after deleting
NameError: name 'mytuple' is not defined
```

Basic Tuples Operations

Tuples respond to the + (Concatenation) , * (repetition) and other operators.

Expression	Results	Description
len((10, 20, 30))	3	Length
(11, 22, 33) + (44, 55, 66)	(11, 22, 33, 44, 55, 66)	Concatenation
('Bye!',) * 4	(' Bye!', ' Bye!', ' Bye!', ' Bye!')	Repetition
33 in (11, 22, 33)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Built-in Tuple Functions

Python includes the following tuple functions

Function	Description
cmp(tuple1, tuple2)	Compares elements of both tuples.
len(tuple)	Gives the total length of the tuple.
max(tuple)	Returns item from the tuple with max value.
min(tuple)	Returns item from the tuple with min value.
tuple(seq)	Converts a list into tuple.

Tuple Membership Test

Keyword ‘in is used to test if an item exists in a tuple or not.

```
mytuple = ('A','c','c','e','n','t','u','r','e')
print("Is 'A' Present in mytuple?  : ",'A' in mytuple);
print("Is 'b' Present in mytuple?  : ",'b' in mytuple);
print("Is 'c' NOT Present in mytuple?  : ",'c' not in mytuple);
```

OUTPUT

```
Is 'A' Present in mytuple?  :  True
Is 'b' Present in mytuple?  :  False
Is 'c' NOT Present in mytuple?  :  False
Python Tuple Methods
```

Tuple Methods

Method	Description
Count('p')	Return the number of items that is equal to 'p'
index(x)	Return index of first item that is equal to x

Iterating Through a Tuple

A for loop is used to iterate though each item in a tuple

```
Locations = ('Accenture','Bangalore');
for name in Locations:
    print("Hello",name)
```

OUTPUT

```
Hello Accenture
Hello Bangalore
```

Built-in Functions with Tuple

Function	Description
all()	Return True if all elements of the tuple are true (or if the tuple is empty).
any()	Return True if any element of the tuple is true. If the tuple is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of tuple as pairs.
len()	Return the length (the number of items) in the tuple.
max()	Return the largest item in the tuple.
min()	Return the smallest item in the tuple
sorted()	Take elements in the tuple and return a new sorted list (does not sort the tuple itself).
sum()	Retrun the sum of all elements in the tuple.
tuple()	Convert an iterable (list, string, set, dictionary) to a tuple.
count(x)	Return the number of items that is equal to x
index(x)	Return index of first item that is equal to x
cmp(tuple1, tuple2)	Compares elements of both tuples.

Advantage of Tuple over List

- Tuples and list look quite similar except the fact that one is immutable and the other is mutable.
- We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Dictionary in Python

- It is an unordered collection of items.
- Every element in a Dictionary has a key and the corresponding value expressed as a key: value pair.
- It is optimized to retrieve values when the key is known.
- Values can be of any datatype and can repeat, but keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

Creating a Dictionary

- A Dictionary is created
- by items inside curly braces {} separated by commas.
- using the built-in function ***dict().***

Creating a Dictionary - Example

```
mydictionary = {}      # empty dictionary
print("Empty Dictionary = ",mydictionary);

mydictionary = {1: 'apple', 2: 'ball'}     # dictionary with integer keys
print("Dictionary with Integer Keys = ",mydictionary);

mydictionary = {'name': 'John', 1: [2, 4, 3]} # dictionary with mixed keys
print("Dictionary with Mixed Keys= ",mydictionary);

mydictionary = dict({1:'apple', 2:'ball'})     # using dict() function
print("Dictionary Created Using function = ",mydictionary);

mydictionary = dict([(1,'apple'), (2,'ball')]) # from sequence having each item as a pair
print("Dictionary Created from Sequence having item as a pair = ",mydictionary);
```

```
OUTPUT
Empty Dictionary =  {}
Dictionary with Integer Keys =  {1: 'apple', 2: 'ball'}
Dictionary with Mixed Keys=  {1: [2, 4, 3], 'name': 'John'}
Dictionary Created Using function =  {1: 'apple', 2: 'ball'}
Dictionary Created from Sequence having item as a pair =  {1: 'apple', 2: 'ball'}
```

Accessing Elements in a Dictionary

While indexing is used with other container types to access values, dictionary uses keys.

Key can be used either inside square brackets or with the get() method.

The difference between get() and [] is that [] throws KeyError if the key is not found where as get() returns “None”, if the key is not found.

Program	OutPut
print("MyDictionary[2] = ",mydictionary[2]);	MyDictionary[2] = Orange
print("MyDictionary[4] = ",mydictionary.get(4));	MyDictionary[4] = Pineapple
print("MyDictionary[5] = ",mydictionary[5]);	**print("MyDictionary[5]= ",mydictionary[5]);** **KeyError: 5**
print("MyDictionary[5] = ",mydictionary.get(5));	MyDictionary[5] = None

Changing or Adding Elements in a Dictionary

- Adding new items and changing the value of the existing items can be done the assignment operator.
- If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

```
mydictionary = {1: 'Apple', 2: 'Orange', 3: 'Banana', 4: 'Pineapple'}
print("Dictionary Elements = ",mydictionary);

mydictionary[5]= "Grapes";
print("Dictionary Elements After Adding Grapes = ",mydictionary);

mydictionary[3]= "Strawberry";
print("Dictionary Elements After Modifying Key 3 = ",mydictionary);
```

OUTPUT

```
Dictionary Elements =  {1: 'Apple', 2: 'Orange', 3: 'Banana', 4: 'Pineapple'}
Dictionary Elements After Adding Grapes =  {1: 'Apple', 2: 'Orange', 3: 'Banana', 4: 'Pineapple', 5: 'Grapes'}
Dictionary Elements After Modifying Key 3 =  {1: 'Apple', 2: 'Orange', 3: 'Strawberry', 4: 'Pineapple', 5: 'Grapes'}
```

Deleting or Removing Elements from a Dictionary

- Elements from a Dictionary can be removed using the following methods.
- pop(key)
- This method removes as item with the provided key and returns the value.
- popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary.
- clear() Removes all the items from the Dictionary.
- del keyword can be used to remove individual items or the entire dictionary itself.

Function	Description
pop(key)	Removes as item with the provided key and returns the value.
popitem()	remove and return an arbitrary item (key, value) form the dictionary
clear()	Removes all the items from the Dictionary
del dictionary_name[key]	Removes an individual item from the Dictionary
del <dictionary name>	Deletes the entire Dictionary

Deleting or Removing Elements from a Dictionary - Example

```
mydictionary = {1: 'Apple', 2: 'Orange', 3: 'Banana', 4: 'Pineapple', 5: 'Grapes'}
print("Dictionary Elements = ",mydictionary);

print("Element Returned after pop(3) Operation on the Dictionary = ",mydictionary.pop(3));
print("Dictionary Elements After POPing the item 3 = ",mydictionary);

print("Element Returned after popitem() Operation on the Dictionary = ",mydictionary.popitem());
print("Dictionary Elements After popitem operation = ",mydictionary);

del mydictionary[2];
print("Dictionary Elements After deleting the item with the key[2] = ",mydictionary);

mydictionary.clear();
print("Element Returned after clear() Operation on the Dictionary = ",mydictionary);

del mydictionary;
print("Dictionary Elements After deleting the entire Dictionary = ",mydictionary);
```

OUTPUT

```
Dictionary Elements =  {1: 'Apple', 2: 'Orange', 3: 'Banana', 4: 'Pineapple', 5: 'Grapes'}
Element Returned after pop(3) Operation on the Dictionary =  Banana
Dictionary Elements After POPing the item 3 =  {1: 'Apple', 2: 'Orange', 4: 'Pineapple', 5: 'Grapes'}
Element Returned after popitem() Operation on the Dictionary =  (1, 'Apple')
Dictionary Elements After popitem operation =  {2: 'Orange', 4: 'Pineapple', 5:'Grapes'}
Dictionary Elements After deleting the item with the key[2] =  {4: 'Pineapple',5: 'Grapes'}
Element Returned after clear() Operation on the Dictionary =  {}

print("Dictionary Elements After deleting the entire Dictionary = ",mydictionary);
NameError: name 'mydictionary' is not defined
```

Dictionary Methods

Method	Description
clear()	Remove all items form the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v (defaults to None).
get(key[,d])	Return the value of key. If key doesnot exit, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(key[,d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[,d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Return a new view of the dictionary's values

Dictionary Comprehension

- Dictionary comprehension is an elegant and concise way to create new dictionary from an iterable in Python. Dictionary comprehension consists of an expression pair (key: value) followed by for statement inside curly braces {}.

```
mydictionary  = {x: x*x*x for x in range(6)}
print("Elements of MyDictionary = ",mydictionary);
#OUTPUT
Elements of MyDictionary =  {0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125}

#This above code is equivalent to
mydictionary = {}
for x in range(6):
   mydictionary[x] = x*x*x;

print("Elements of MyDictionary = ",mydictionary);
```

```
OUTPUT
Elements of MyDictionary =  {0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
Elements of MyDictionary =  {0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
```

Dictionary Comprehension – (Contd)

A dictionary comprehension can optionally contain an if statements. An optional if statement can filter out items to form the new dictionary. Here are some examples to make dictionary with only odd items.

```
mydictionary  = {x: x*x*x for x in range(11) if x%2 == 1}
print("Elements of MyDictionary = ",mydictionary);
```

OUTPUT

Elements of MyDictionary = {1: 1, 3: 27, 9: 729, 5: 125, 7: 343}

Other Dictionary Operations

in Keyword (Membership Test)

Keyword in is use to test whether a key is in the Dictionary or not.

Note : Membership test is for keys only, not for values.

Example:

```
mydictionary = {1: 'Apple', 2: 'Orange', 3: 'Banana', 4: 'Pineapple'}
print("Dictionary Elements = ",mydictionary);
print("Is 2  present in MyDictionary as a key?  ",2 in mydictionary);
print("Is 7  present in MyDictionary as a key?  ",7 in mydictionary);
```

OUTPUT

```
Dictionary Elements =  {1: 'Apple', 2: 'Orange', 3: 'Banana', 4: 'Pineapple'}
Is 2 is present in MyDictionary as a key?   True
Is 7 is present in MyDictionary as a key?   False
```

Iterating Through a Dictionary

A for loop can be used to iterate though each key in a dictionary.

Example:

```
mydictionary = {1: 'Apple', 2: 'Orange', 3: 'Banana', 4: 'Pineapple'}
for i in mydictionary:
        print(mydictionary[i]);
```

OUTPUT
Apple
Orange
Banana
Pineapple

Built-in Functions with Dictionary

Function	Description
all()	Return True if all keys of the dictionary are true (or if the dictionary is empty).
any()	Return True if any key of the dictionary is true. If the dictionary is empty, return False.
len()	Return the length (the number of items) in the dictionary.
cmp()	Compares items of two dictionaries.
sorted()	Return a new sorted list of keys in the dictionary.

Sets in Python

- A Set is an unordered collection of items.
- Every element is unique (no duplicates) and must be immutable. i.e. a set cannot have a mutable element, like list, set or dictionary, as its element.
- However, the set itself is mutable (we can add or remove items).
- Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.
- A Set can have any number of items and they may be of different types (integer, float, tuple, string etc.).

Creating a Set in Python

A Set can be created by

- Placing all the items (elements) inside curly braces {}, separated by comma.
- using the built-in function set()
- But a set cannot have a mutable element, like list, set or dictionary, as its element

Various ways of creating a Set	OUTPUT
myset = {1, 2, 3, 4, 5, 6} # set of integers **print("SET of Integer type Elements = ",myset);**	SET of Integer type Elements = {1, 2, 3, 4, 5, 6}
myset = {1.0, "Hello", (1, 2, 3), 4.5 , 55} # set of mixed datatypes **print("SET Elements = ",myset);**	SET Elements = {(1, 2, 3), 1.0, 4.5, 'Hello', 55}
myset = {1,2,3,4,3,2} # set donot have duplicates **print("SET Cannot have Duplicate Elements = ",myset);**	SET Cannot have Duplicate Elements = {1, 2, 3, 4}
myset = set([1,2,3,2]) **# Creating a set from a list using the built-in method set()** **print("SET Created from a List using the built-in method set() = ",myset);**	SET Created from a List using the built-in methos set() = {1, 2, 3}
#myset = {1, 2, [3, 4]} # set cannot have mutable items **#print("SET having elements = ",myset);**	myset = {1, 2, [3, 4]} # set cannot have mutable items TypeError: unhashable type: 'list'

Creating an empty set

An Empty Set is created using the set() function without any argument

Note: Creating an empty set using curly braces {} will make an empty dictionary in Python.

Example:

```
a = {}
print("Type of a (a = {})   = ",type(a));
b = set()
print("Type of b (b = set() = ",type(b));
```

```
OUTPUT
Type of a (a = {})   =  <class 'dict'>
Type of b (b = set() =  <class 'set'>
```

Changing the elements in a Set

- Sets are mutable.
- Elements cannot be accessed or changed by indexing or slicing. Set does not support it.
- A Single element can be added using the method add().
- Multiple elements can be added using the update() method.
- The update() method can take tuples, lists, strings or other sets as its argument.
- In all cases, duplicates are avoided.

Changing the elements in a Set - Example

Program	OUTPUT
myset = {10,30} **print("Elements of MySet = ",myset);**	Elements of the MySet = {10, 30}
#myset[0]	# TypeError: 'set' object does not support indexing
myset.add(2) **print("Elements of MySet After adding the element 2 = ",myset);**	Elements of MySet After adding the element 2 = {10, 2, 30}
myset.update([2,3,4]) **print("Elements of MySet after adding the elements 2,3,4 = ",myset);**	Elements of MySet after adding the elements 2,3,4 = {10, 2, 3, 4, 30}
myset.update([4,5], {1,6,8}) **print("Elements of the MySet after adding two lists [4,5], {1,6,8} = ",myset);**	Elements of the MySet after adding two lists [4,5], {1,6,8} = {1, 2, 3, 4, 5, 6, 8, 10, 30}

Removing Elements from a Set

Following methods can be used to remove elements from a set.

Method	Description
discard()	Removes 1 element
remove()	Removes 1 element
pop()	Removes 1 item arbitrarily and returns the same.
clear()	Removes all elements

Note: If the item doesnot exists, discard() function does nothing , whereas remove() function throws an error.

Removing Elements from a Set - Example

```
myset = {11, 33, 44, 55, 66 , 88,77}
print("Elements of the MySet = ",myset);
myset.discard(44)
print("Elements of the MySet after discard(44) is called = ",myset);
myset.remove(66)
print("Elements of the MySet after remove(66) is called = ",myset);
myset.discard(22);
print("Elements of the MySet after discard(22) is called = ",myset);
print("Element removed and returned by pop function = ",myset.pop());
print("Elements of the MySet after popping is  = ",myset);
print("Elements of the MySet clear method is called = ",myset.clear());
print("Elements of the MySet after discard(22) is called = ",myset.remove(22));
```

OUTPUT

```
Elements of the MySet =  {33, 66, 11, 44, 77, 55, 88}
Elements of the MySet after discard(44) is called =  {33, 66, 11, 77, 55, 88}
Elements of the MySet after remove(66) is called =  {33, 11, 77, 55, 88}
Elements of the MySet after discard(22) is called =  {33, 11, 77, 55, 88}
Element removed and returned by pop function =  33
Elements of the MySet after popping is =  {11, 77, 55, 88}
Elements of the MySet clear method is called =  None
print("Elements of the MySet after discard(22) is called = ",myset.remove(22));
KeyError: 22
```

Set Operations

Mathematical set operations can be performed on Sets in Python

Following are the Set Operations

- union
- intersection
- difference
- symmetric difference

All the above operations can be done using operators or methods.

UNION

Union of *A* and *B* is a set of all elements from both sets.

Union is performed by using

- | operator.
- union() method.

Example:

```
A = {11, 22, 33, 44, 55}
B = {44, 55, 66, 77, 88}
print("Elements of Set A = ",A);
print("Elements of Set B = ",B);
C = A | B;
print("Result of the UNION Operation using '|' Operator.i.e. (A | B) = ",C);
C = A.union(B);
print("Result of the UNION Operation Using union() method i.e. A.union(B)= ",C);
```

OUTPUT:

```
Elements of Set A =  {33, 11, 44, 22, 55}
Elements of Set B =  {88, 66, 44, 77, 55}
Result of the UNION Operation using '|' Operator.i.e. (A | B) =
{33, 66, 11, 44, 77, 22, 55, 88}
Result of the UNION Operation Using union() method i.e. A.union(B)=
{33, 66, 11, 44, 77, 22, 55, 88}
```

INTERSECTION

Intersection of *A* and *B* is a set of elements that are common in both sets.

Intersection is performed by using

- & operator
- intersection()

Example:

```
A = {11, 22, 33, 44, 55}
B = {44, 55, 66, 77, 88}
print("Elements of Set A = ",A);
print("Elements of Set B = ",B);
C = A & B;
print("Result of the INTERSECTION Operation using '&' Operator.i.e. (A & B) = ",C);
C = A.intersection(B);
print("Result of the INTERSECTION Operation Using intersection() method i.e.
A.intersection(B) = ",C);
```

OUTPUT:

```
Elements of Set A =  {33, 11, 44, 22, 55}
Elements of Set B =  {88, 66, 44, 77, 55}
Result of the INTERSECTION Operation using '&' Operator.i.e. (A & B) =  {44, 55}
Result of the INTERSECTION Operation Using intersection() method i.e.
A.intersection(B) =  {44, 55}
```

DIFFERENCE

Difference of *A* and *B* (*A* - *B*) is a set of elements that are only in *A* but not in *B*.

Difference is performed using

- ‘-‘ operator
- difference() method.

Example:

```
A = {11, 22, 33, 44, 55}
B = {44, 55, 66, 77, 88}
print("Elements of Set A = ",A);
print("Elements of Set B = ",B);
C = A - B;
print("Result of the DIFFERENCE Operation i.e. using '-' Operator.(A - B) = ",C);
C = A.difference(B);
print("Result of the DIFFERENCE Operation Using difference() method i.e.
A.difference(B) = ",C);
```

```
OUTPUT
Elements of Set A =  {33, 11, 44, 22, 55}
Elements of Set B =  {88, 66, 44, 77, 55}
Result of the DIFFERENCE Operation i.e. using '-' Operator.(A - B) =  {33, 11, 22}
Result of the DIFFERENCE Operation Using difference() method i.e.
A.difference(B) =  {33, 11, 22}
```

SYMMETRIC DIFFERENCE

Symmetric Difference of *A* and *B* is a set of element in both *A* and *B* except those common in both.

Symmetric difference is performed using

- ^ operator
- symmetric_difference() method

```
A = {11, 22, 33, 44, 55}
B = {44, 55, 66, 77, 88}
print("Elements of Set A = ",A);
print("Elements of Set B = ",B);
C = A ^ B;
print("Result of the SYMMETRIC DIFFERENCE Operation using '^' Operator. i.e. (A ^ B) = ",C);
C = A.symmetric_difference(B);
print("Result of the SYMMETRIC DIFFERENCE Operation Using difference() method i.e.
A.symmetric_difference(B) = ",C);
```

OUTPUT:

```
Elements of Set A =  {33, 11, 44, 22, 55}
Elements of Set B =  {88, 66, 44, 77, 55}
Result of the SYMMETRIC DIFFERENCE Operation using '^' Operator. i.e. (A ^ B) = {33, 66, 11, 77, 22, 88}
Result of the SYMMETRIC DIFFERENCE Operation Using difference() method i.e. A.symmetric_difference(B) =
{33, 66, 11, 77, 22, 88}
```

Python Set Methods

- Following methods are from the set objects.

Method	Description
add()	Add an element to a set
clear()	Remove all elemets form a set
copy()	Return a shallow copy of a set
difference()	Return the difference of two or more sets as a new set
difference_update()	Remove all elements of another set from this set
discard()	Remove an element from set if it is a member. (Do nothing if the element is not in set)
intersection()	Return the intersection of two sets as a new set
intersection_update()	Update the set with the intersection of itself and another
isdisjoint()	Return True if two sets have a null intersection
issubset()	Return True if another set contains this set
issuperset()	Return True if this set contains another set
pop()	Remove and return an arbitary set element. Raise KeyError if the set is empty
remove()	Remove an element from a set. It the element is not a member, raise a KeyError
symmetric_difference()	Return the symmetric difference of two sets as a new set
symmetric_difference_update()	Update a set with the symmetric difference of itself and another
union()	Return the union of sets in a new set
update()	Update a set with the union of itself and others

Other Set Operations

Membership Test

Keyword **'in'** is used to check the existence of an item in set.

Example:

```
myset = set("ACCENTURE")
print("Elements of MySet = ",myset);
print("Check whether 'A' Exists in  MySet? = ",'A' in myset);
print("Check whether 'M' Exists in  MySet? = ",'M' in myset);
```

```
OUTPUT
Elements of MySet =  {'N', 'T', 'A', 'E', 'C', 'U', 'R'}
Check whether 'A' Exists in  MySet? =  True
Check whether 'M' Exists in  MySet? =  False
```

Iterating Through a Set

A for loop can be used to iterate though each item in a set.

Example:

```
myset = set("ACCENTURE")
print("Iterating throught the Set using FOR statement...");
for letter in myset:
        print(letter)
```

OUTPUT

```
Iterating throught the Set using FOR statement...
C
N
A
U
T
R
E
```

Built-in Functions with Set

Function	Description
all()	Return True if all elements of the set are true (or if the set is empty).
any()	Return True if any element of the set is true. If the set is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of set as a pair.
len()	Return the length (the number of items) in the set.
max()	Return the largest item in the set.
min()	Return the smallest item in the set.
sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
sum()	Return the sum of all elements in the set.

Questions