IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:

Python

COASTAL GATEWAY LIMITED™

One company Unlimited Solutions...

"Thank you so much for giving us the opportunity to explore more, be more fearless, and adapt to change better!"

Sri Harsha
CEO
Coastal Gateway Pvt Ltd
Hyderabad

Vinod Kumar Pattagari
Software Engineer
Coastal Gateway Pvt Ltd
Hyderabad

Agenda

- Introduction of Python Programming Language
- Features of Python
- Why Python is Dynamic Language
- Slice Function in python
- List in Python
- Dictionary in Python

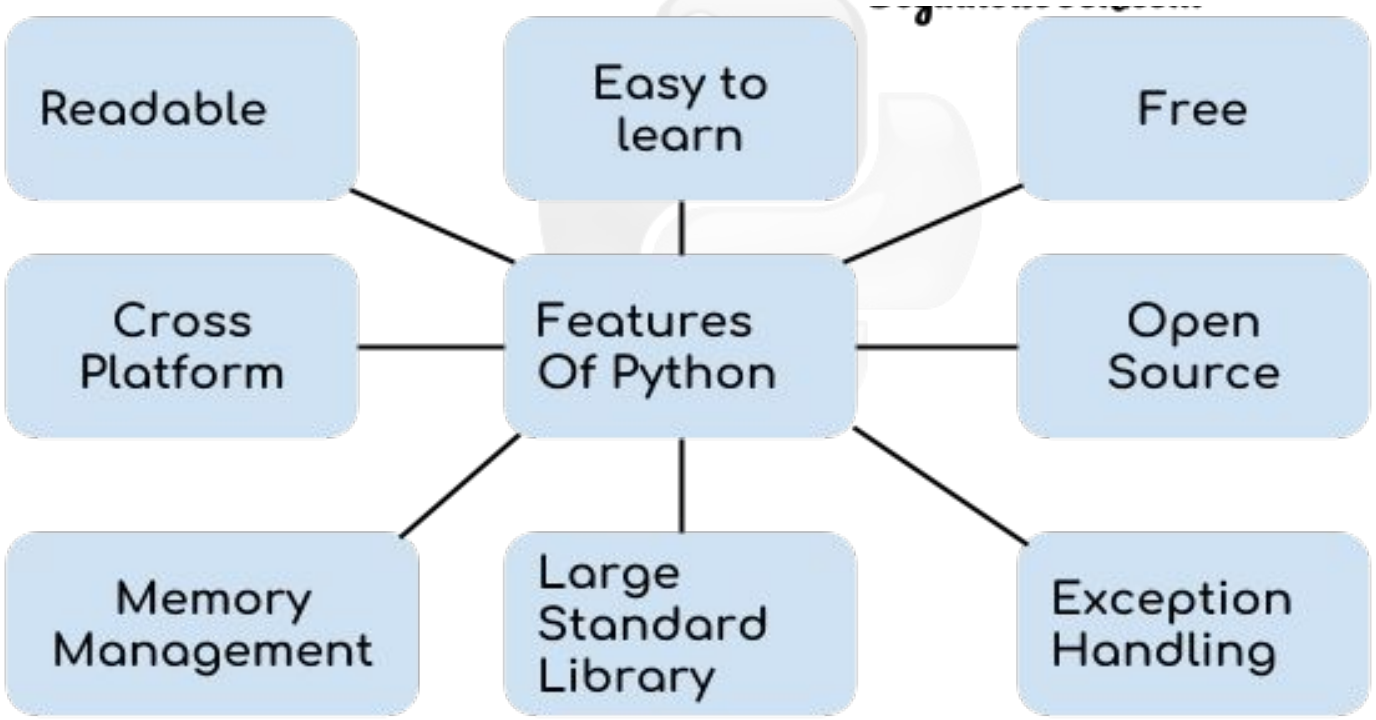
Introduction of Python Programming Language

Python is developed by **Guido van Rossum**. Guido van Rossum started implementing Python in 1989. Python is a very simple programming language so even if you are new to programming,

What Can You Do with Python?

- Web development
- Data Analysis
- Scripting
- Game development
- You can develop Embedded applications in Python.
- Desktop applications

Features of Python



Why Python is Dynamic Language?

Python is strongly typed as the interpreter keeps track of all variables types. It's also very dynamic as it rarely uses what it knows to limit variable usage. In Python, it's the program's responsibility to use built-in functions like isinstance() and issubclass() to test variable types and correct usage.

When Compared to Other language let's say “JAVA:”

Java syntax to create a variable:	Python syntax to create a variable:
Int num = 10:	num=10
Statically type	Dynamically Type

Slice Function in python

The slice() function is used to get a slice object representing the set of indices specified by range(start, stop, step).

Syntax:

`slice(stop)`

`slice(start, stop[, step])`

Parameter:

Start - An integer number specify where to start the slicing. Default is 0

Stop - An integer number specify where the slice will end.

Step - An integer number specify the step of the slicing. Default is .

Example for slice()

```
pyStr = 'Python'

sliceObj = slice(4)

print(pyStr[sliceObj])

hsliceObj = slice(1,5, 2)

print(pyStr[sliceObj])
```

Output:

```
Pyth

yh
```

LISTS

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.).

Example: list = [1, 2, 3] - list with same data-types.
list = [1, "Hello", 3.4] - list with mixed data-types.

ACCESS ITEMS FROM A LIST

It can be access in several ways Use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

Example: list = ['p','r','o','b','e']
print(list[2]) -Positive Indexing
print(list[-2]) -Negative Indexing
Output: o b

SLICE LISTS

Accessing a range of items in a list by using the slicing operator [ ] using (colon :). Slicing can be best visualized by considering the index to be between the elements.

Example: list = ['p','r','o','b','e'] print(list[0:4]) -Positive
print(list[-2:-1]) - Negative.
Output: [‘p’,’r’,’o’,’b’] [‘b’]

LIST METHODS

append() - Add an element to the end of the list.
count() - Returns the count of number of items passed as an argument.
extend() - Add all elements of a list to the another list.
index() - Returns the index of the first matched item.
insert() - Insert an item at the defined index.
pop() - Removes and returns an element at the given index.
copy() - Returns a shallow copy of the list
remove() - Removes an item from the list.
reverse() - Reverse the order of items in the list.
sort() - Sort items in a list in ascending order.

APPEND() METHODS

The append() method adds a single item to the existing list. The append() method only modifies the original list. It doesn't return any value. The append() method takes a single item and adds it to the end of the list. The item can be numbers, strings, another list, dictionary etc.

Example 1: Adding Element to a List list = ['hi', 'hello'] print('old list: ', list)
list.append('welcome!')
print('Updated list: ', list)
output:old list: ['hi', 'hello'] Updated list: ['hi', 'hello‘,'welcome!']
Example 2: Adding List to a List list = ['hi', 'hello'] print('old list: ', list)
list1=['welcome‘] list.append(list1)
print('Updated list: ', list)
output:old list: ['hi', 'hello'] Updated list: ['hi', 'hello', ['welcome']]

COUNT() METHODS

Count() method counts how many times an element has occurred in a list and returns it. The count() method takes a single argument: element - element whose count is to be found. The count() method returns the number of occurrences of an element in a list.

Example : Count the occurrence of tuple and list inside the list

```
list = ['a', ('h', 'i'), ('a', 'b'), [8, 4]]
count = list.count(('a', 'b'))
 print("The count of ('a', 'b') is:", count)
count = list.count([3, 4])
print("The count of [3, 4] is:", count)
Output The count of ('a', 'b') is: 1
            The count of [3, 4] is: 0
```

EXTEND() METHODS

The extend() extends the list by adding all items of a list to the end. Extend() method takes a single argument (a list) and adds it to the end. This method also add elements of other native data_types ,like tuple and set to the list,

Example : Using extend() Method

```
language = ['C', 'C++', 'Python']
 language1 = ['JAVA', 'COBOL']
language.extend(language1)
 print('Language List: ', language)
Output Language List: ['C', 'C++', 'Python', 'JAVA', 'COBOL']
```

INDEX( ) METHODS

Index() method search and find given element in the list and returns its position. However, if the same element is present more than once, index() method returns its smallest/first position. Index in Python starts from 0 not 1. The index() method returns the index of the element in the list. The index method takes a single argument: element - element that is to be searched.

Example : Find index of tuple and list inside a list

```
random = ['a', ('a', 'd'),('a','b'), [3, 4]]
index = random.index(('a', 'b'))
print("The index of ('a', 'b'):", index)
index = random.index([3, 4])
print("The index of [3, 4]:", index)
```

Output The index of ('a', 'b'): 2 The index of [3, 4]: 3

INSERT( ) METHODS

The insert() method inserts the element to the list at the given index. The insert() function takes two parameters: index - position where element needs to be inserted element - this is the element to be inserted in the list The insert() method only inserts the element to the list. It doesn't return any value.

Example : Inserting a Tuple (as an Element) to the List

```
list = [{1, 2}, [5, 6, 7]]
print('old List: ', list)
number_tuple = (3, 4)
list.insert(1, number_tuple)
print('Updated List: ', list)
```

Output old List: [{1, 2}, [5, 6, 7]] Updated List: [{1, 2}, (3, 4), [5, 6, 7]]

POP( ) METHODS

The pop() method takes a single argument (index) and removes the element present at that index from the list. If the index passed to the pop() method is not in the range, it throws IndexError: pop index out of range exception. The parameter passed to the pop() method is optional. If no parameter is passed, the default index -1 is passed as an argument. Also, the pop() method removes and returns the element at the given index and updates the list.

Example : language = ['Python', 'Java', 'C++‘, 'C']
print('Return Value: ', language.pop())
print('Updated List: ', language)
print('Return Value: ', language.pop(-1))
print('Updated List: ', language)
Output Return Value: C Updated List: ['Python', 'Java', 'C++'] Return Value: C++ Updated List: ['Python', 'Java']

COPY( ) / ALIASING METHODS

The copy() method returns a shallow copy of the list. A list can be copied with = operator. old_list = [1, 2, 3] new_list = old_list The problem with copying the list in this way is that if you modify the new_list, the old_list is also modified.

Example 1: Copying a List

```
old_list = [1, 2, 3]
 new_list = old_list new_list.append('a')
print('New List:', new_list )
print('Old List:', old_list )
```

Output New List: [1, 2, 3, 'a'] Old List: [1, 2, 3, 'a']

REMOVE( ) METHODS

The remove() method searches for the given element in the list and removes the first matching element. The remove() method takes a single element as an argument and removes it from the list. If the element(argument) passed to the remove() method doesn't exist, valueError exception is thrown.

Example 1 :Remove Element From The List

```
list = [5,78,12,26,50]
print('Original list: ', list)
list.remove(12)
print('Updated list elements: ', list)
```

Output Original list: [5, 78, 12, 26, 50]

Updated list: [5, 78, 26, 50]

REVERSE( ) METHODS

The reverse() method reverses the elements of a given list. The reverse() function doesn't return any value. It only reverses the elements and updates the list.

Example 1 :Reverse a List Using Slicing Operator list = [1,5,8,6,11,55] print('Original List:', list) reversed_list = list[::-1] print('Reversed List:', reversed_list)
Output Original List: [1, 5, 8, 6, 11, 55] Reversed List: [55, 11, 6, 8, 5, 1]
Example 2: Reverse a List list = [1,5,8,6,11,55] print('Original List:', list) list.reverse() print('Reversed List:', list)
Output Original List: [1, 5, 8, 6, 11, 55] Reversed List: [55, 11, 6, 8, 5, 1] 2

SORT( ) METHODS

The sort() method sorts the elements of a given list. The sort() method sorts the elements of a given list in a specific order - Ascending or Descending

Example 1 :Sort a given List in ascending order list = [1,15,88,6,51,55] print('Original List:', list) list.sort() print('sorted List:', list)

Output Original List: [1, 15, 88, 6, 51, 55] sorted List: [1, 6, 15, 51, 55, 88]

Example 2 : Sort the list in Descending order list = [1,15,88,6,51,55] print('Original List:', list) list.sort(reverse=True) print('sorted List:', list)

Output Original List: [1, 15, 88, 6, 51, 55] sorted List: [88, 55, 51, 15, 6, 1] 31

PYTHON DICTIONARY

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

Creating a dictionary

Creating a dictionary is as simple as placing items inside curly braces {} separated by comma. An item has a key and the corresponding value expressed as a pair, key: value. While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique

ACCESS ELEMENTS FROM 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.

```
d = {'name':'Ram', 'age': 26}
print("name is :",d['name']) d['age']=33
d['Initial']='S‘ print("Updated dict :",d) del d['Initial']
print("Updated dict :",d)
```

Output name is : Ram Updated dict : {'name': 'Ram', 'age': 33, 'Initial': 'S'} Updated dict : {'name': 'Ram', 'age': 33}

Access Dictionary Elements

Once a dictionary is created, you can access it using the variable to which it is assigned during creation. For example, in our case, the variable myDict can be used to access the dictionary elements.

Example

```
>>> myDict["A"]
'Apple'
>>> myDict["B"]
'Boy'
>>> myDict["C"]
'Cat'
```

Output:

```
>>> myDict
{'A': 'Apple', 'C': 'Cat', 'B': 'Boy'}
```

Update Dictionary Elements

Just the way dictionary values are accessed using keys, the values can also be modified using the dictionary keys. Here is an example to modify python dictionary element:

Example

```
>>> myDict["A"] = "Application"
>>> myDict["A"]
'Application'
>>> myDict
  {'A': 'Application', 'C': 'Cat', 'B': 'Boy'}
```

Delete Dictionary Elements

Individual elements can be deleted easily from a dictionary. Here is an example to remove an element from dictionary.

Example

```
>>> myDict
{'A': 'Application', 'C': 'Cat', 'B': 'Boy'}
>>> del myDict["A"]
>>> myDict
{'C': 'Cat', 'B': 'Boy'}
```

Thank You