MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

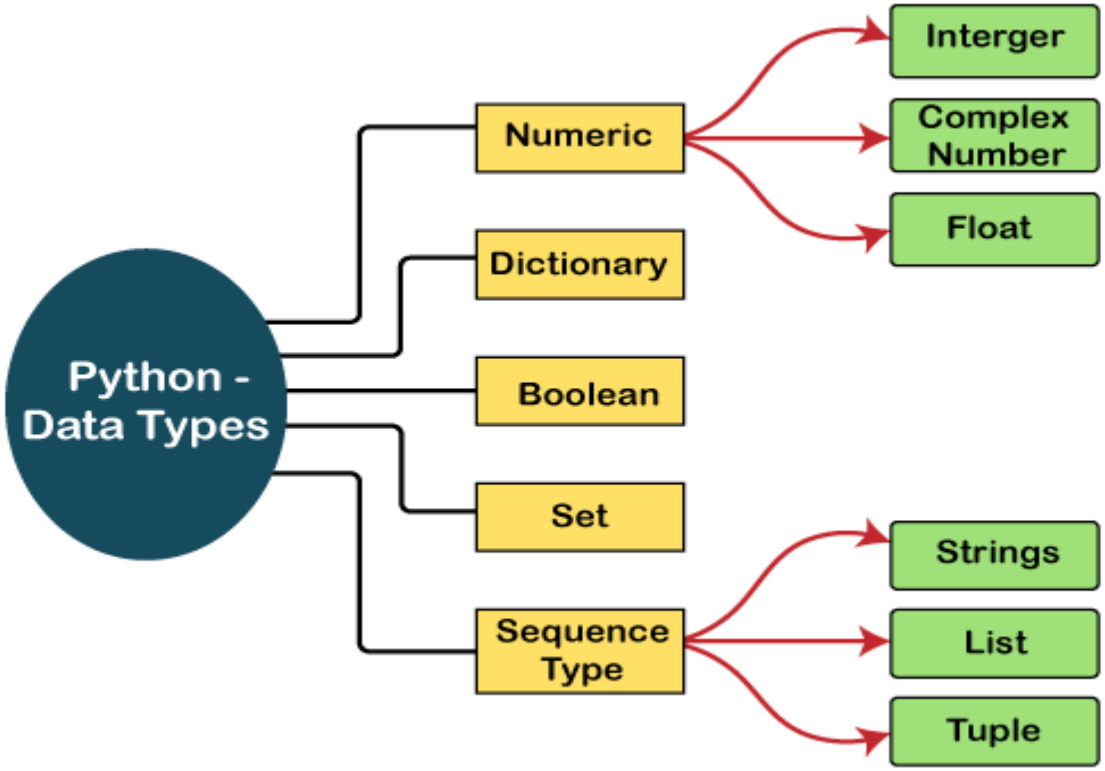
School of Computer Engineering and Technology

Lab Assignment-04,05

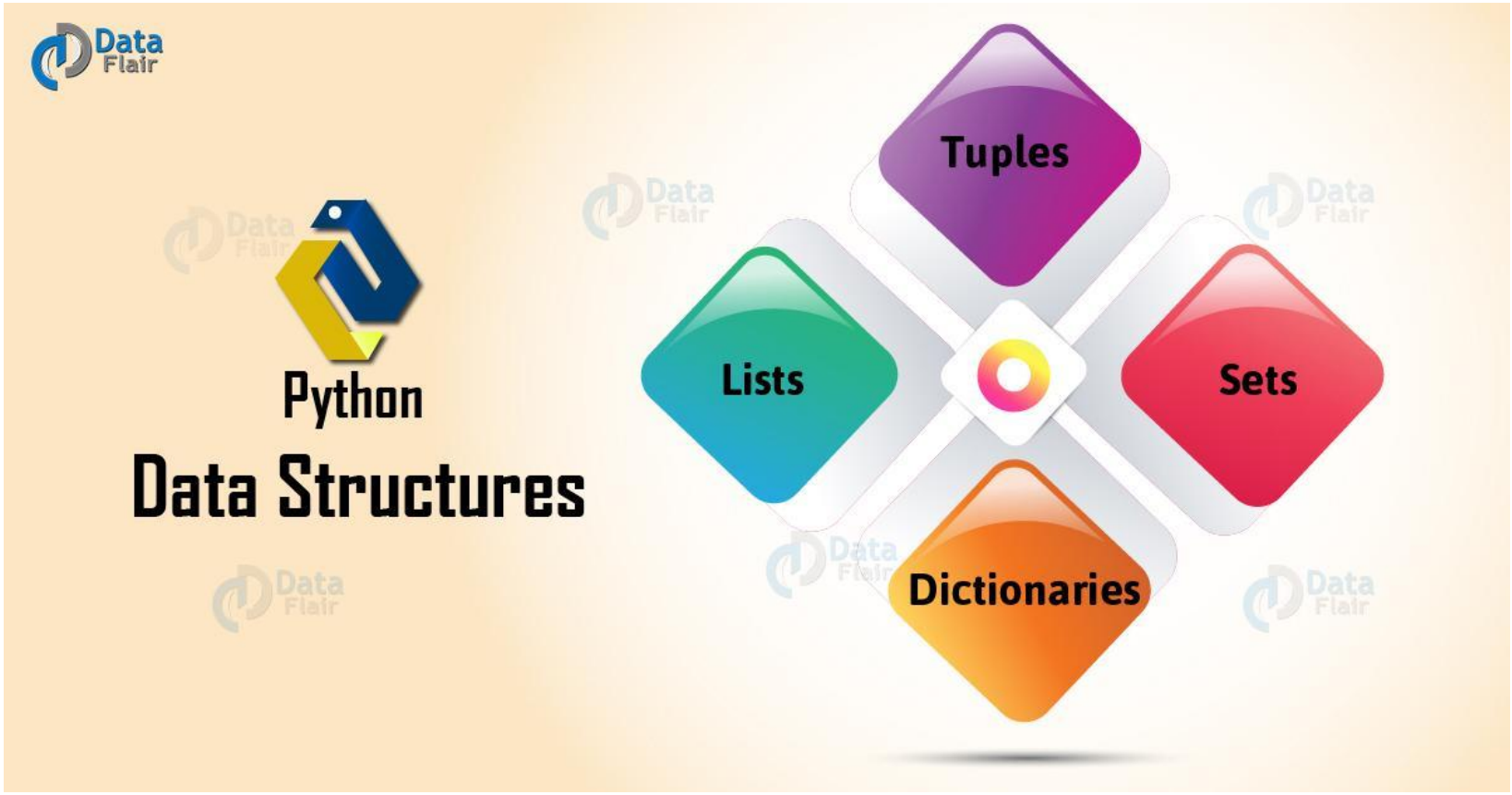
04-Write a python program to create, append and remove etc. operation on list.

05. Write a python program to create, append and remove etc. operation on Dictionary and Tuple

Data types in Python



Python Data Structure



List

- Lists are used to store data of different data types in a sequential manner.
- There are addresses assigned to every element of the list, which is called as Index.
- The index value starts from 0 and goes on until the last element called the **positive index**.
- There is also **negative indexing** which starts from -1 enabling you to access elements from the last to first.

Syntax for List operation

1. creating list[]
   - my_list = [] #create empty list
   - print(my_list)
   - my_list = [1, 2, 3, 'example', 3.132] #creating list with data
   - print(my_list)
- **Output:**

  []

  [1, 2, 3, ‘example’, 3.132]

Adding Elements to list

- Adding the elements in the list can be done using the append(), extend() and insert() functions.
  - The **append()** function adds all the elements passed to it as a single element.
  - The **extend()** function adds the elements one-by-one into the list.
  - The **insert()** function adds the element passed to the index value and increase the size of the list too.

Example

- my_list = [1, 2, 3]
- print(my_list)

#add as a single element
- my_list.append([555, 12])
- print(my_list)

#add as different elements
- my_list.extend([234, 'more_example'])
- print(my_list)

#add element i
- my_list.insert(1, 'insert_example')
- print(my_list)

- **Output:**
  [1, 2, 3]
  [1, 2, 3, [555, 12]]
  [1, 2, 3, [555, 12], 234, ‘more_example’]
  [1, ‘insert_example’, 2, 3, [555, 12], 234, ‘more_example’]

Deleting Elements from List

- To delete elements, use the ***del*** keyword which is built-in into Python but this does not return anything back to us.
- If you want the element back, you use the **pop()** function which takes the index value.
- To remove an element by its value, you use the **remove()** function.

Example

- my_list = [1, 2, 3, 'example', 3.132, 10, 30]
  #delete element at index 5
- del my_list[5]
- print(my_list)

  #remove element with value
- my_list.remove('example')
- print(my_list)

  #pop element from list
- a = my_list.pop(1)
- print('Popped Element: ', a, ' List remaining: ', my_list)

  #empty the list
- my_list.clear()
- print(my_list)

- **Output:**
  [1, 2, 3, 'example', 3.132, 30]
  [1, 2, 3, 3.132, 30]
  Popped Element: 2 List remaining: [1, 3, 3.132, 30]
  []

Accessing Elements

- Accessing elements is the same as accessing Strings in Python. You pass the index values and hence can obtain the values as needed.
  - my_list = [1, 2, 3, 'example', 3.132, 10, 30]
  - for element in my_list: #access elements one by one
    print(element)
  - print(my_list) #access all elements
  - print(my_list[3]) #access index 3 element
  - print(my_list[0:2]) #access elements from 0 to 1 and exclude 2
  - print(my_list[::-1]) #access elements in reverse

- **Output:**
  1
  2
  3
  example
  3.132
  10
  30
  [1, 2, 3, 'example', 3.132, 10, 30]
  example
  [1, 2]
  [30, 10, 3.132, 'example', 3, 2, 1]

Length(),count(),index(),sorted()

- my_list = [1, 2, 3, 10, 30, 10]
- print(len(my_list)) #find length of list
- print(my_list.index(10)) #find index of element that occurs first
- print(my_list.count(10)) #find count of the element
- print(sorted(my_list)) #print sorted list but not change original
- my_list.sort(reverse=True) #sort original list
- print(my_list)

- **Output:**

  6
  3
  2
  [1, 2, 3, 10, 10, 30]
  [30, 10, 10, 3, 2, 1]

Dictionary

- Dictionaries are used to store **key-value** pairs
- in Python, this structure is stored using Dictionaries.
  - my_dict = {} #empty dictionary
  - print(my_dict)
  - my_dict = {1: 'Python', 2: 'Java'} #dictionary with elements
  - print(my_dict)
- **Output:**
  {}
  {1: ‘Python’, 2: ‘Java’}

Changing and Adding key, value pairs in Dictionary

- my_dict = {'First': 'Python', 'Second': 'Java'}
- print(my_dict)
- my_dict['Second'] = 'C++' #changing element
- print(my_dict)
- my_dict['Third'] = 'Ruby' #adding key-value pair
- print(my_dict)

- **Output:**
  {‘First’: ‘Python’, ‘Second’: ‘Java’}
  {‘First’: ‘Python’, ‘Second’: ‘C++’}
  {‘First’: ‘Python’, ‘Second’: ‘C++’, ‘Third’: ‘Ruby’}

```
Dict = {'Name': [‘Riya','pranav', 'Vaishali'], 1: [1, 2, 3, 4]}
#Dict.items()
#Dict.keys()
Dict.values()
```

output:

dict_values([[Riya, 'pranav', 'Vaishali'], [1, 2, 3, 4]])

Deleting key, value pairs in dictionary

- To delete the values, you use the **pop()** function which returns the value that has been deleted.
- To retrieve the key-value pair, you use the **popitem()** function which returns a tuple of the key and value.
- To clear the entire dictionary, you use the **clear()** function.

Example

- my_dict = {'First': 'Python', 'Second': 'Java', 'Third': 'Ruby'}
- a = my_dict.pop('Third') #pop element
- print('Value:', a)
- print('Dictionary:', my_dict)
- b = my_dict.popitem() #pop the key-value pair
- print('Key, value pair:', b)
- print('Dictionary', my_dict)
- my_dict.clear() #empty dictionary
- print('n', my_dict)

- **Output:**

  Value: Ruby
  Dictionary: {‘First’: ‘Python’, ‘Second’: ‘Java’}
  Key, value pair: (‘Second’, ‘Java’)
  Dictionary {‘First’: ‘Python’}
  {}

Example

```
my_dict = {'First': 'Python', 'Second': 'Java'}
print(my_dict['First'])                    #access elements using keys
print(my_dict.get('Second'))
my_dict = {'First': 'Python', 'Second': 'Java', 'Third': 'Ruby'}
print(my_dict.keys())                      #get keys
print(my_dict.values())                    #get values
print(my_dict.items())                     #get key-value pairs
print(my_dict.get('First'))
```

- **Output:**

```
Python
Java
dict_keys(['First', 'Second', 'Third'])
dict_values(['Python', 'Java', 'Ruby'])
dict_items([('First', 'Python'), ('Second', 'Java'), ('Third', 'Ruby')])
Python
```

Tuple

- **Tuple**
  - are the same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what.
  - The only exception is when the data inside the tuple is mutable, only then the tuple data can be changed.
  - my_tuple = (1, 2, 3) #create tuple
  - print(my_tuple)
- **Output:**
  (1, 2, 3)

Accessing Elements

- my_tuple2 = (1, 2, 3, 'SYBTECH') #access elements
- for x in my_tuple2:
  print(x)
- print(my_tuple2)
- print(my_tuple2[0])
- print(my_tuple2[:])
- print(my_tuple2[3][4])

- **Output:**
  1
  2
  3
  SYBTECH
  (1, 2, 3, 'SYBTECH')
  1
  (1, 2, 3, 'SYBTECH')

Example

- To append the values, you use the ‘+’ operator which will take another tuple to be appended to it.
  - my_tuple = (1, 2, 3)
  - my_tuple = my_tuple + (4, 5, 6) #add elements
  - print(my_tuple)
  - my_tuple = (1, 2, 3, ['hindi', 'python'])
  - my_tuple[3][0] = 'english'
  - print(my_tuple)
  - print(my_tuple.count(2))
  - print(my_tuple.index(['english', 'python']))

- **Output:**
  (1, 2, 3, 4, 5, 6)
  (1, 2, 3, [‘english’, ‘python’])
  1
  3

Exercise

- Define a list called **list_1** with four integer members, and find the output of the following

  - Access the first three elements from **list_1** using forward indices:
    - list_1[1:3] # [12, 89]
  - Access the last element from **list_1** using the **len** function:
    - list_1[len(list_1) - 1] #1
  - Access the last two elements from **list_1** by slicing:
    - list_1[-2:] # [89, 1]
  - Access the first two elements using backward indices
    - list_1[:-2] # [34, 12]
  - Reverse the elements in the string:
    - list_1[-1::-1] # [1, 89, 12, 34]

- Create a dictionary subject and access a particular key in a dictionary:
- Assign a new value to the key:
- Create a tuple to demonstrate how tuples are immutable. Unpack it to read all elements

References

- https://www.javatpoint.com/python-features
- https://www.w3schools.com/python/python_intro.asp
- https://www.geeksforgeeks.org/python-data-types/
- https://www.tutorialsteacher.com/python/python-data-types
- https://www.programiz.com/python-programming/variables-datatypes
- http://sthurlow.com/python/lesson06/
- https://subscription.packtpub.com/book/application_development/9781789800111/1/ch01lvl1sec04/lists-sets-strings-tuples-and-dictionaries
- https://www.edureka.co/blog/data-structures-in-python/
- https://www.guru99.com/if-loop-python-conditional-structures.html
- https://beginnersbook.com/2018/01/python-functions/