**Batch: C1\_1 Roll No.: 12**

**Experiment / assignment / tutorial No. 2**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:**  Basic Data structure in python |

**AIM:** Use suitable methods to get output for given input.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:** Use of basic data structure in Python.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Resource Needed: Python IDE**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Theory:**

## Python Collections (Arrays)

There are four collection data types in the Python programming language:

* **List** is a collection which is ordered and changeable. Allows duplicate members.
* Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
* Set is a collection which is unordered and unindexed. No duplicate members.
* Dictionary is a collection which is unordered and changeable. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and it could mean an increase in efficiency or security.

**List:** Lists are used to store multiple items in a single variable. Lists are created using square brackets. e.g. mylist = ["apple", "banana", "cherry"]

## List Methods

Python has a set of built-in methods that you can use on lists. L:list, e:element, i:index

|  |  |
| --- | --- |
| **Method** | **Description** |
| L.append(e) | Adds an element at the end of the list |
| L.clear() | Removes all the elements from the list |
| L.copy() | Returns a copy of the list |
| L.count(e) | Returns the number of elements with the specified value |
| L.extend(L2) | Add the elements of a list (or any iterable), to the end of the current list |
| L.index(e) | Returns the index of the first element with the specified value |
| L.insert(i,e) | Adds an element at the specified position |
| L.pop(i) | Removes the element at the specified position |
| L.remove(e) | Removes the item with the specified value |
| L.reverse() | Reverses the order of the list |
| L.sort() | Sorts the list |

## Tuple

Tuples are used to store multiple items in a single variable. A tuple is a collection which is ordered and **unchangeable**. Tuples are written with round brackets.

e.g. mytuple = ("apple", "banana", "cherry")

## Tuple Methods

Python has two built-in methods that you can use on tuples. T:tuple, e:element

|  |  |
| --- | --- |
| **Method** | **Description** |
| T.count(e) | Returns the number of times a specified value occurs in a tuple |
| T.index(e) | Searches the tuple for a specified value and returns the position of where it was found |

## Set

Sets are used to store multiple items in a single variable. A set is a collection which is both ***unordered*** and ***unindexed***. Sets are written with curly brackets.

e.g. myset = {"apple", "banana", "cherry"}

## Set Methods

Python has a set of built-in methods that you can use on sets.

|  |  |
| --- | --- |
| **Method** | **Description** |
| S.add(e) | Adds an element to the set |
| S.clear() | Removes all the elements from the set |
| S.copy() | Returns a copy of the set |
| S1.difference(S2) | Returns a set containing the difference between two or more sets |
| S1.difference\_update(S2) | Removes the items in this set that are also included in another, specified set |
| S1.discard(e) | Remove the specified item |
| S1.intersection(S2) | Returns a set, that is the intersection of two other sets |
| S1.intersection\_update(S2) | Removes the items in this set that are not present in other, specified set(s) |
| S1.isdisjoint(S2) | Returns whether two sets have a intersection or not |
| S1.issubset(S2) | Returns whether another set contains this set or not |
| S1.issuperset(S2) | Returns whether this set contains another set or not |
| S.pop() | Removes an element from the set |
| S.remove(e) | Removes the specified element |
| S1.symmetric\_difference(S2) | Returns a set with the symmetric differences of two sets |
| S1.symmetric\_difference\_update(S2) | inserts the symmetric differences from this set and another |
| S1.union(S2) | Return a set containing the union of sets |
| S1.update(L1) | Update the set with the union of this set and others |

## Dictionary

Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is **ordered (3.7 version onward)**, **changeable** and **does not allow duplicates**.

Dictionaries are written with curly brackets, and have keys and values.

e.g. thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}

## Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

|  |  |
| --- | --- |
| **Method** | **Description** |
| D.clear() | Removes all the elements from the dictionary |
| D.copy() | Returns a copy of the dictionary |
| D.get(k) | Returns the value of the specified key |
| D.items() | Returns a list containing a tuple for each key value pair |
| D.keys() | Returns a list containing the dictionary's keys |
| D.pop(k) | Removes the element with the specified key |
| D.popitem() | Removes the last inserted key-value pair |
| D.setdefault(k,v) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| D.update({k:v}) | Updates the dictionary with the specified key-value pairs |
| D.values() | Returns a list of all the values in the dictionary |

**Problem Definition:**

1. In below table input variable, python code and output column is given. You have to complete blank cell in every row.

|  |  |  |  |
| --- | --- | --- | --- |
| **List** | | | |
| **Input** | **Python Code** | **Output** | |
| thislist=["apple","banana","cherry","orange","kiwi","melon","mango"] | print(len(thislist))  print(type(thislist))  print(thislist[1])  print(thislist[-1])  print(thislist[2:5])  print(thislist[:4])  print(thislist[2:]) | 7  <class 'list'>  banana  mango  ['cherry', 'or ange', 'kiwi']  ['apple', 'banana', 'cherry', 'or ange']  ['cherry', 'or ange', 'kiwi', 'melon', 'mango'] | |
|  | | | |
| thislist = ["orange", "mango", "kiwi", "pineapple", "apple"] | if "apple" in thislist:    print("Yes, 'apple' is in the fruits list")  for x in thislist:  print(x)  for i in range(len(thislist)):  print(thislist[i])  thislist.sort()  print(thislist) | | Yes,'apple' is in the fruits list  orange  mango  kiwi  pineapple  apple  orange  mango  kiwi  pineapple  apple  ['apple', 'kiwi', 'mango',  'orange', 'pineapple'] |
| thislist=["apple","banana","cherry"] | thislist[1]=  "blackcurrent"  print(thislist) | | ['apple','blackcurrant','cherry'] |
| thislist=["apple", "banana", "cherry"] | thislist.insert(2,"watermelon")  print(thislist) | | ['apple','banana','watermelon', 'cherry'] |
| thislist=["apple","banana","cherry"] | thislist.append("orange")  print(thislist) | | ['apple', 'banana', 'cherry', 'orange'] |
| thislist=["apple", "banana", "cherry"] tropical=["mango", "pineapple"] | thislist.extend(tropical) print(thislist) | | ['apple', 'banana', 'cherry', 'mango', 'pineapple'] |
| thislist = ["apple", "banana", "cherry"] | thislist.pop(1)  print(thislist) | | ['apple', 'cherry'] |
| thislist = ["apple", "banana", "cherry"] | del thislist  print(thislist) | | Error |
| thislist = ["apple", "banana", "cherry"] | thislist.clear()  print(thislist) | | [] |
| thislist = ["apple", "banana", "cherry"] | x=thislist  y= thislist.copy()  thislist.clear()  print(x)  print(y) | | []  ['apple', 'banana', 'cherry'] |
| list1 = [5, 6, 7]  list2 = [1, 2, 3] | list3 = list1 + list2  print(list3) | | [5, 6, 7, 1, 2, 3] |

|  |  |  |
| --- | --- | --- |
| **Tuple** | | |
| **Input** | **Python Code** | **Output** |
| x = ("apple",)  y = ("apple") | print(type(x))  print(type(y)) | <class 'tuple'>  <class 'str'> |
| thistuple=("apple","banana","cherry") | print(thistuple[-1]) | cherry |
| x = ("apple", "banana", "cherry") | x[1] = "kiwi"  print(x) | Error |
| x = ("apple", "banana", "cherry") | y = list(x)  y[1] = "kiwi"  x = tuple(y)  print(x) | ('apple', 'kiwi', 'cherry') |
| fruits = ("apple", "banana", "cherry", "strawberry", "raspberry") | (green, yellow, \*red) = fruits  print(green)  print(yellow)  print(red)  print(type(red)) | apple  banana  ['cherry',  'strawberry',  'raspberry']  <class 'list'> |
| fruits = ("apple", "banana", "cherry") | mytuple = fruits \* 2  print(mytuple.count("apple"))  print(mytuple.index("banana")) | 2  1 |

|  |  |  |
| --- | --- | --- |
| **Set** | | |
| **Input** | **Python Code** | **Output** |
| myset = {"abc", 34, True, 40.5} | print(myset)  print(len(myset))  print(type(myset))  print(34 in thisset)  myset.add("orange")  print(myset) | {40.5, 'abc', 34, True}  4  <class 'set'>  Error  {True, 34, 'abc', 40.5, 'orange'} |
| thisset = {"apple", "mango", "cherry"}  tropical={"papaya", "mango"} | thisset=thisset+tropical  print(thisset) | Error |
| thisset.update(tropical)  print(thisset) | {'cherry', 'apple', 'mango', 'papaya'} |
| thisset.intersection\_update (tropical)  print(thisset) | {'mango'} |
| thisset.symmetric\_difference\_update(tropical)  print(thisset) | {'papaya', 'cherry', 'apple'} |

|  |  |  |
| --- | --- | --- |
| **Dictionaries** | | |
| **Input** | **Python Code** | **Output** |
| thisdict={"brand":"Ford","model": "Mustang","year": 1964, "year": 2020} | print(thisdict)  print(type(thisdict))  print(len(thisdict))  print(thisdict["brand"])  print(thisdict["year"])  x = thisdict.get("model")  print(x)  y = thisdict.keys()  print(y)  z = thisdict.values()  print(z)  thisdict["color"] = "white"  print(thisdict)  if "model" in thisdict:  print("Yes”) | {'brand':'Ford','model':'Mustang','year': 2020}  <class 'dict'>  3  Ford  2020  Mustang  dict\_keys(['brand', 'model', 'year'])  dict\_values(['Ford', 'Mustang', '2020'])  {'brand': 'Ford', 'model': 'Mustang', 'year': '2020', 'color': 'white'}  Yes |
| thisdict["year"] = 2018  print(thisdict) | {'brand': 'Ford', 'model': 'Mustang', 'year': 2018} |
| thisdict.pop("model")  print(thisdict) | {'brand': 'Ford', 'year': '2020'} |
| for x in thisdict:  print(x)  print(thisdict[x]) | brand  Ford  model  Mustang  year  2020 |
| for x, y in thisdict.items():  print(x, y) | brand Ford  model Mustang  year 2020 |

2. Write a python program to take list values as input parameters and returns another list without any duplicates.

3. Write a program that takes a string as input from the user and computes the frequency of each letter. Use a variable of dictionary type to maintain the count.

**Books/ Journals/ Websites referred:**

1. Reema Thareja, *Python Programming: Using Problem Solving Approach*, Oxford University Press, First Edition 2017, India
2. Sheetal Taneja and Naveen Kumar, *Python Programming: A modular Approach*, Pearson India, Second Edition 2018,India

**Implementation details:**

**2.**

new\_list = input("Enter a list of values separated by spaces: ").split()

print(new\_list)

new\_set = set(new\_list)

print(new\_set)

**3.**

# Initialize an empty dictionary to store letter frequencies

letter\_frequency = {}

# Get input from the user

input\_string = input("Enter a string: ")

# Convert the input string to lowercase to make the counting case- insensitive

input\_string = input\_string.lower()

# Iterate through each character in the input string

for char in input\_string:

 # Check if the character is a letter

 if char.isalpha():

 # If the letter is already in the dictionary, increment its count by 1

  if char in letter\_frequency:

   letter\_frequency[char] += 1

   # If the letter is not in the dictionary, add it with a count of 1

  else:

   letter\_frequency[char] = 1

# Print the letter frequencies

print("Letter frequencies:")

for letter, frequency in letter\_frequency.items():

 print(f"{letter}: {frequency}")

**Output(s):**

**2.**

Enter a list of values separated by spaces: 1 3 5 7 5 3 1

['1', '3', '5', '7', '5', '3', '1']

{'7', '3', '1', '5'}

**3.**

Enter a string: python programming

Letter frequencies:

p: 2

y: 1

t: 1

h: 1

o: 2

n: 2

r: 2

g: 2

a: 1

m: 2

i: 1

**Conclusion:**

**Post Lab Descriptive Questions**

1. List out Mutable and Immutable Data Types in Python.

**Mutable Data Types :** Lists, Dictionaries, Sets, etc.

**Immutable Data Types:** Tuple, Integer, Float, String, etc.

1. What do you mean by indexed and ordered data type in python

**Indexed Data Type** refers to data structures like lists and strings where each element has a unique index or position, allowing for direct access to individual elements based on their position within the sequence.

**Ordered Data Type** means that the elements in the data structure have a defined order, which is maintained. Lists, tuples, and strings are examples of ordered data types, and they preserve the sequence of elements as they were added or defined.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**