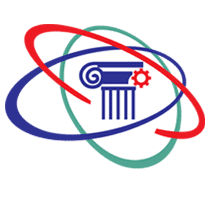
Acropolis Institute of Technology and Research



Dept. of C.S.I.T

PYTHON LAB WORK

Submitted By:- Submitted To:

Pratik Patidar Prof. Vandana

0827CI191046 Kate Ma’am

**Index**

1.Write a program to demonstrate List in Python and List comprehension

2. Write a program to demonstrate Tuple in Python

3. Write a program to demonstrate String in Python

4. Write at least 10-15  problem codes   to demonstrate usage of Lambda functions in Python (with use of map, filter , reduce etc.)

5. Write a program to demonstrate input output commands, use of format, and split commands

6. Write a program to demonstrate usage of  functions , passing list as argument , recursion, local and global scope using  in Python

7. Write at least 10 program to demonstrate usage of  classes and Objects in Python( Stack , Queue , dequeue implementation already demonstrated in lab)

1.Write a program to demonstrate List in Python and List comprehension

#define list

items = [2,3,8,7,8,4,13,9]

print(items)

#change element in list

items[1] = 85

items[3] = 65

items[5] = 45

print(items)

# List indexing

print(items[0])

print(items[2])

print(items[4])

#output

[2, 3, 8, 7, 8, 4, 13, 9]

[2, 85, 8, 65, 8, 45, 13, 9]

2

8

8

|  |
| --- |
| >>> mylist = ["apple", "banana", "cherry"] |
|  | >>> print(mylist) |
|  | ['apple', 'banana', 'cherry'] |
|  | >>> mylist = ["apple", "banana", "cherry", "apple", "cherry"] |
|  | >>> print(mylist) |
|  | ['apple', 'banana', 'cherry', 'apple', 'cherry'] |
|  | >>> mylist = ["apple", "banana", "cherry"] |
|  | >>> print(len(mylist)) |
|  | 3 |
|  | >>> list1 = ["apple", "banana", "cherry"] |
|  | >>> list2 = [1, 5, 7, 9, 3] |
|  | >>> list3 = [True, False, False] |
|  | >>> print(list1) |
|  | ['apple', 'banana', 'cherry'] |
|  | >>> print(list2) |
|  | [1, 5, 7, 9, 3] |
|  | >>> print(list3) |
|  | [True, False, False] |
|  | >>> mylist = ["apple", "banana", "cherry"] |
|  | >>> print(type(mylist)) |
|  | <class 'list'> |
|  | >>> mylist = list(("apple", "banana", "cherry")) |
|  | >>> print(mylist) |
|  | ['apple', 'banana', 'cherry'] |
|  | >>> mylist = ["apple", "banana", "cherry"] |
|  | >>> print(mylist[1]) |
|  | banana |
|  | >>> mylist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] |
|  | >>> print(mylist[2:5]) |
|  | ['cherry', 'orange', 'kiwi'] |
|  | >>> mylist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] |
|  | >>> print(mylist[:4]) |
|  | ['apple', 'banana', 'cherry', 'orange'] |
|  | >>> mylist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] |
|  | >>> print(mylist[2:]) |
|  | ['cherry', 'orange', 'kiwi', 'melon', 'mango'] |
|  | >>> mylist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] |
|  | >>> print(mylist[-4:-1]) |
|  | ['orange', 'kiwi', 'melon'] |
|  | >>> mylist = ["apple", "banana", "cherry"] |
|  | >>> if "apple" in mylist: |
|  | ... print("Yes, 'apple' is in the fruits list") |
|  | File "<stdin>", line 2 |
|  | print("Yes, 'apple' is in the fruits list") |
|  | ^ |
|  | IndentationError: expected an indented block |
|  | >>> if "apple" in mylist: |
|  | ... print("Yes, 'apple' is in the fruits list") |
|  | ... |
|  | Yes, 'apple' is in the fruits list |
|  | >>> |

2. Write a program to demonstrate Tuple in Python

# Empty tuple

tup1 = ()

print(tup1)

# Tuple having integers

tup1 = (1, 2, 3)

print(tup1)

# tuple with mixed datatypes

tup1 = (1, "Hi", 2, 3)

print(tup1)

# nested tuple

tup1 = ("mouse", [6, 5, 4], (1, 2, 3))

print(tup1)

tup1 = 3, 4, "ball"

print(tup1)

# tuple unpacking is also possible

a, b, c = tup1

print(a)

print(b)

print(c)

#Access Tuple Elements

#1. Indexing

tup1 = ('p','r','a','t','i','k')

print(tup1[0])

print(tup1[5])

Output:

()

(1, 2, 3)

(1, 'Hi', 2, 3)

('mouse', [6, 5, 4], (1, 2, 3))

(3, 4, 'ball')

3

4

ball

p

k

|  |
| --- |
| >>> mytuple=("ram","shyam","mohan") |
|  | >>> print(mytuple) |
|  | ('ram', 'shyam', 'mohan') |
|  | >>> mytuple=("ram","shyam","mohan","ram","mohan") |
|  | >>> print(mytuple) |
|  | ('ram', 'shyam', 'mohan', 'ram', 'mohan') |
|  | >>> mytuple=("ram","shyam","mohan") |
|  | >>> print(len(mytuple)) |
|  | 3 |
|  | >>> mytuple = ("ram",) |
|  | >>> print(type(mytuple)) |
|  | <class 'tuple'> |
|  | >>> #NOT a tuple |
|  | >>> mytuple = ("ram") |
|  | >>> print(type(mytuple)) |
|  | <class 'str'> |
|  | >>> tuple1 = ("a", "b", "c") |
|  | >>> tuple2 = (1, 2, 3, 4, 5) |
|  | >>> tuple3 = (True, False, False) |
|  | >>> print(tuple1) |
|  | ('a', 'b', 'c') |
|  | >>> print(tuple2) |
|  | (1, 2, 3, 4, 5) |
|  | >>> print(tuple3) |
|  | (True, False, False) |
|  | >>> mytuple = ("abc", 20, True, 11, "male") |
|  |  |
|  | SyntaxError: unexpected indent |
|  | >>> mytuple = ("abc", 20, True, 11, "male") |
|  | >>> print(mytuple) |
|  | ('abc', 20, True, 11, 'male') |
|  | >>> mytuple=("ram","shyam","mohan") |
|  | >>> print(mytuple[1]) |
|  | >>> print(mytuple[1]) |
|  | shyam |
|  | >>> mytuple=("ram","shyam","mohan") |
|  | >>> print(mytuple[-1]) |
|  | mohan |
|  | >>> mytuple=("ram","shyam","mohan","sohan","rohan") |
|  | >>> print(mytuple[1:3]) |
|  | ('shyam', 'mohan') |
|  | >>> mytuple=("ram","shyam","mohan","sohan","rohan") |
|  | >>> print(mytuple[:1]) |
|  | ('ram',) |
|  | >>> mytuple=("ram","shyam","mohan","sohan","rohan") |
|  | >>> print(mytuple[1:]) |
|  | ('shyam', 'mohan', 'sohan', 'rohan') |
|  | >>> |

3. Write a program to demonstrate String in Python

# defining string

str1 = 'Hello'

print(str1)

str1 = "Hello"

print(str1)

str1 = '''Hello'''

print(str1)

# triple quotes string can extend multiple lines

str1 = """Hello guys"""

print(str1)

Output:-

Hello

Hello

Hello

Hello guys

|  |
| --- |
| >>> print("Hello") |
|  | Hello |
|  | >>> print('Hello') |
|  | Hello |
|  | >>> a = "Hello" |
|  | >>> print(a) |
|  | Hello |
|  | >>> a = """My Name is Piyush Dave.""" |
|  | >>> print(a) |
|  | My Name is Piyush Dave. |
|  | >>> a = '''My Name is Piyush Dave.''' |
|  | >>> print(a) |
|  | My Name is Piyush Dave. |
|  | >>> a = "Hello, World" |
|  | >>> print(a[1]) |
|  | e |
|  | >>> a = "Hello, World" |
|  | >>> print(len(a)) |
|  | 12 |
|  | >>> txt = "The best things in life are free" |
|  | >>> print("free" in txt) |
|  | True |
|  | >>> |
|  | KeyboardInterrupt |
|  | >>> txt = "The best things in life are free" |
|  | >>> if "free" in txt: |
|  | ... print("Yes, 'free' is present.") |
|  | ... |
|  | Yes, 'free' is present. |
|  | >>> txt = "The best things in life are free" |
|  | >>> print("expensive" not in txt) |
|  | True |
|  | >>> txt = "The best things in life are free" |
|  | >>> if "expensive" not in txt: |
|  | ... print("No, 'expensive' is NOT present.") |
|  | ... |
|  | No, 'expensive' is NOT present. |
|  | >>> |

4. Write at least 10-15  problem codes   to demonstrate usage of Lambda functions in Python (with use of map, filter , reduce etc.)

|  |
| --- |
| >>> x = lambda a : a + 10 |
|  | >>> print(x(5)) |
|  | 15 |
|  | >>> x = lambda a, b : a \* b |
|  | >>> print(x(5, 6)) |
|  | 30 |
|  | >>> x = lambda a, b, c : a + b + c |
|  | >>> print(x(5, 6, 2)) |
|  | 13 |
|  | >>> def myfunc(n): |
|  | ... return lambda a : a \* n |
|  | ... |
|  | >>> mydoubler = myfunc(2) |
|  | >>> |
|  | >>> print(mydoubler(11)) |
|  | 22 |
|  | >>> def myfunc(n): |
|  | ... return lambda a : a \* n |
|  | ... |
|  | >>> mytripler = myfunc(3) |
|  | >>> |
|  | >>> print(mytripler(11)) |
|  | 33 |
|  | >>> def myfunc(n): |
|  | ... return lambda a : a \* n |
|  | ... |
|  | >>> mydoubler = myfunc(2) |
|  | >>> mytripler = myfunc(3) |
|  | >>> print(mydoubler(11)) |
|  | 22 |
|  | >>> print(mytripler(11)) |
|  | 33 |

|  |
| --- |
| >>> # Without using lambdas |
|  | >>> def starts\_with\_A(s): |
|  | ... return s[0] == "A" |
|  | ... |
|  | >>> fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"] |
|  | >>> map\_object = map(starts\_with\_A, fruit) |
|  | >>> print(list(map\_object)) |
|  | [True, False, False, True, False] |
|  |  |
|  | >>> fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"] |
|  | >>> map\_object = map(lambda s: s[0] == "A", fruit) |
|  | >>> print(list(map\_object)) |
|  | [True, False, False, True, False] |
|  |  |
|  | >>> # Without using lambdas |
|  | >>> def starts\_with\_A(s): |
|  | ... return s[0] == "A" |
|  | ... |
|  | >>> fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"] |
|  | >>> filter\_object = filter(starts\_with\_A, fruit) |
|  | >>> print(list(filter\_object)) |
|  | ['Apple', 'Apricot'] |
|  |  |
|  | >>> fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"] |
|  | >>> filter\_object = filter(lambda s: s[0] == "A", fruit) |
|  | >>> print(list(filter\_object)) |
|  | ['Apple', 'Apricot'] |
|  |  |
|  | # Without using lambdas |
|  | >>> from functools import reduce |
|  | >>> |
|  | >>> def add(x, y): |
|  | ... return x + y |
|  | ... |
|  | >>> list = [2, 4, 7, 3] |
|  | >>> print(reduce(add, list)) |
|  | 16 |
|  | >>> from functools import reduce |
|  | >>> |
|  | >>> list = [2, 4, 7, 3] |
|  | >>> print(reduce(lambda x, y: x + y, list)) |
|  | 16 |
|  | >>> print("With an initial value: " + str(reduce(lambda x, y: x + y, list, 10))) |
|  | With an initial value: 26 |
|  | >>> |

5. Write a program to demonstrate input output commands, use of format, and split commands

6. Write a program to demonstrate usage of  functions , passing list as argument , recursion, local and global scope using  in Python

|  |
| --- |
| def my\_function(): |
|  | print("Hello") |
|  |  |
|  | my\_function() |
|  | def evenOdd(x): |
|  | if (x % 2 == 0): |
|  | print("even") |
|  | else: |
|  | print("odd") |
|  |  |
|  | evenOdd(2) |
|  | evenOdd(3) |
|  | def student(firstname, lastname): |
|  | print(firstname, lastname) |
|  |  |
|  |  |
|  | student(firstname='Piyush', lastname='Dave') |
|  | student(lastname='Dave', firstname='Piyush') |
|  | def square\_value(num): |
|  | """This function returns the square |
|  | value of the entered number""" |
|  | return num\*\*2 |
|  |  |
|  |  |
|  | print(square\_value(2)) |
|  | print(square\_value(-4)) |
|  | def swap(x, y): |
|  | temp = x |
|  | x = y |
|  | y = temp |
|  |  |
|  | x = 2 |
|  | y = 3 |
|  | swap(x, y) |
|  | print(x) |
|  | print(y) |
|  |  |
|  | //output// |
|  |  |
|  | Hello |
|  | even |
|  | odd |
|  | Piyush Dave |
|  | Piyush Dave |
|  | 4 |
|  | 16 |
|  | 2 |
|  | 3 |

7. Write at least 10 program to demonstrate usage of  classes and Objects in Python( Stack , Queue , dequeue implementation already demonstrated in lab)

|  |
| --- |
|  |
| class Dog: |
|  | pass |
|  | class Dog: |
|  |  |
|  | attr1 = "mammal" |
|  | attr2 = "dog" |
|  | def fun(self): |
|  | print("I'm a", self.attr1) |
|  | print("I'm a", self.attr2) |
|  | Rodger = Dog() |
|  | print(Rodger.attr1) |
|  | Rodger.fun() |
|  |  |
|  |  |
|  | //output// |
|  |  |
|  | mammal |
|  | I'm a mammal |
|  | I'm a dog |
|  |  |
|  |  |
|  | ...Program finished with exit code 0 |
|  | Press ENTER to exit console. |
|  |  |
|  |  |
|  | class Person: |
|  | def \_\_init\_\_(self, name, age): |
|  | self.name = name |
|  | self.age = age |
|  |  |
|  | p1 = Person("piyush", 20) |
|  |  |
|  | print(p1.name) |
|  | print(p1.age) |
|  | class Person: |
|  | def \_\_init\_\_(self, name, age): |
|  | self.name = name |
|  | self.age = age |
|  |  |
|  | def myfunc(self): |
|  | print("Hello my name is " + self.name) |
|  |  |
|  | p1 = Person("piyush", 20) |
|  | p1.myfunc() |
|  |  |
|  | //output// |
|  |  |
|  | piyush |
|  | 20 |
|  | Hello my name is piyush |
|  |  |

#queue

|  |
| --- |
|  |
| que = [] |
|  |  |
|  | que.append('Apple') |
|  | que.append('Mango') |
|  | que.append('Papaya') |
|  |  |
|  | print(que) |
|  | print(que.pop(0)) |
|  |  |
|  | //output// |
|  | ['Apple', 'Mango', 'Papaya'] |
|  | Apple |
|  |  |
|  |  |
|  | ...Program finished with exit code 0 |
|  | Press ENTER to exit console. |
|  |  |
|  | # Initializing a queue |
|  | queue = [] |
|  |  |
|  | # Adding elements to the queue |
|  | queue.append('a') |
|  | queue.append('b') |
|  | queue.append('c') |
|  |  |
|  | print("Initial queue") |
|  | print(queue) |
|  |  |
|  | # Removing elements from the queue |
|  | print("\nElements dequeued from queue") |
|  | print(queue.pop(0)) |
|  | print(queue.pop(0)) |
|  | print(queue.pop(0)) |
|  |  |
|  | print("\nQueue after removing elements") |
|  | print(queue) |
|  |  |
|  | //output// |
|  | Initial queue |
|  | ['a', 'b', 'c'] |
|  |  |
|  | Elements dequeued from queue |
|  | a |
|  | b |
|  | c |
|  |  |
|  | Queue after removing elements |
|  | [] |
|  |  |
|  |  |
|  | ...Program finished with exit code 0 |
|  | Press ENTER to exit console. |
|  | import collections |
|  | de = collections.deque([1, 2, 3, 3, 4, 2, 4]) |
|  |  |
|  | print ("The number 4 first occurs at a position : ") |
|  | print (de.index(4,2,5)) |
|  |  |
|  | de.insert(4,3) |
|  |  |
|  | print ("The deque after inserting 3 at 5th position is : ") |
|  | print (de) |
|  | print ("The count of 3 in deque is : ") |
|  | print (de.count(3)) |
|  |  |
|  | de.remove(3) |
|  |  |
|  | print ("The deque after deleting first occurrence of 3 is : ") |
|  | print (de) |
|  |  |
|  | //output// |
|  |  |
|  | The number 4 first occurs at a position : |
|  | 4 |
|  | The deque after inserting 3 at 5th position is : |
|  | deque([1, 2, 3, 3, 3, 4, 2, 4]) |
|  | The count of 3 in deque is : |
|  | 3 |
|  | The deque after deleting first occurrence of 3 is : |
|  | deque([1, 2, 3, 3, 4, 2, 4]) |
|  |  |

#stack

|  |
| --- |
|  |
| my\_stack = [] |
|  |  |
|  | # append() function to push |
|  | # element in the my\_stack |
|  | my\_stack.append('x') |
|  | my\_stack.append('y') |
|  | my\_stack.append('z') |
|  |  |
|  |  |
|  | print(my\_stack) |
|  | # pop() function to pop |
|  | # element from my\_stack in |
|  | # LIFO order |
|  | print('\nElements poped from my\_stack:') |
|  | print(my\_stack.pop()) |
|  | print(my\_stack.pop()) |
|  | print(my\_stack.pop()) |
|  |  |
|  | print('\nmy\_stack after elements are poped:') |
|  | print(my\_stack) |
|  |  |
|  | //output// |
|  | ['x', 'y', 'z'] |
|  |  |
|  | Elements poped from my\_stack: |
|  | z |
|  | y |
|  | x |
|  |  |
|  | my\_stack after elements are poped: |
|  | [] |
|  |  |

8. Dictionary

#Empty Dictionary

dict1 = {}

print(type(dict1))

print(dict1)

#print dic key and items

dict1["Virat Kohli"] = 100

dict1["Sachin Tendulkar"] = 200

print(dict1)

print(dict1.get("Virat Kohli"))

print(dict1.items())

print(dict1.keys())

Output:

<class 'dict'>

{}

{'Virat Kohli': 100, 'Sachin Tendulkar': 200}

100

dict\_items([('Virat Kohli', 100), ('Sachin Tendulkar', 200)])

dict\_keys(['Virat Kohli', 'Sachin Tendulkar'])

>