**CA Lab-V LAB on Python Programming**

**Assignment 1. Develop programs to understand the control structures of python**

print("Select your Choice:")

print("1. For Loop")

print("2. While Loop")

choice = int( input() )

if( choice == 1 ):

for x in range(1,8,2):

print(x)

elif( choice == 2 ):

count=1;

while( count< 8):

print(count)

count+=2;

else:

print( "Ok" )

**Output: -**

Select your Choice:

1. For Loop

2. While Loop

1

1

3

5

7

**Assignment 2. Develop programs to learn different types of structures (list, dictionary, tuples) in python**

print("Select your Choice:")

print("1. List")

print("2. Dictionary ")

print("3. Tuple")

choice **=** int( input() )

if( choice == 1 ):

my\_list **=** [1, 2, 3, 'example', 3.132] *#creating list with data*

print(my\_list)

my\_list**.**append([555, 12]) *#add as a single element*

print(my\_list)

my\_list**.**extend([234, 'more\_example']) *#add as different elements*

print(my\_list)

my\_list**.**insert(1, 'insert\_example') *#add element*

print(my\_list)

del my\_list[5] *#delete element at index 5*

print(my\_list)

my\_list**.**remove('example') *#remove element with value*

print(my\_list)

a **=** my\_list**.**pop(1) *#pop element from list*

print('Popped Element: ', a, ' List remaining: ', my\_list)

my\_list**.**clear() *#empty the list*

print(my\_list)

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*

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 c*

elif( choice == 2 ):

my\_dict **=** {1: 'Python', 2: 'Java'} *#dictionary with elements*

print(my\_dict)

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)

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(my\_dict)

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*

elif( choice == 3 ):

my\_tuple = (1, 2, 3) *#create tuple*

print(my\_tuple)

my\_tuple2 = (1, 2, 3, 'edureka') *#access elements*

for x in my\_tuple2:

print(x)

print(my\_tuple2)

print(my\_tuple2[0])

print(my\_tuple2[:])

print(my\_tuple2[3][4])

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']))

else:

print( "Ok" )

**Output: -**

Select your Choice:

1. List

2. Dictionary

3. Tuple

3

(1, 2, 3)

1

2

3

edureka

(1, 2, 3, 'edureka')

1

(1, 2, 3, 'edureka')

e

(1, 2, 3, 4, 5, 6)

(1, 2, 3, ['english', 'python'])

1

3

**Assignment 3. Develop programs to learn concept of functions scoping, recursion and list mutability.**

total = 0; *# This is global variable. # Function definition is here*

def sum( arg1, arg2 ):

*# Add both the parameters and return them."*

total = arg1 + arg2; *# Here total is local variable.*

print ("Inside the function local total : ", total)

return total;

def factorial(x):

if x == 1:

return 1

else:

return (x \* factorial(x-1))

print("Select your Choice:")

print("1. Functions Scoping")

print("2. Recursion")

print("3. List Mutability")

choice = int( input() )

if( choice == 1 ):

*# Now you can call sum function*

sum( 10, 20 );

print ("Outside the function global total : ", total)

elif( choice == 2 ):

num = 4

print("The factorial of", num, "is", factorial(num))

elif( choice == 3 ):

my\_list = [1, 2, 3, 'example', 3.132] *#creating list with data*

print(my\_list)

my\_list.append([555, 12]) *#add as a single element*

print(my\_list)

my\_list.extend([234, 'more\_example']) *#add as different elements*

print(my\_list)

my\_list.insert(1, 'insert\_example') *#add element*

print(my\_list)

del my\_list[5] *#delete element at index 5*

print(my\_list)

my\_list.remove('example') *#remove element with value*

print(my\_list)

a = my\_list.pop(1) *#pop element from list*

print('Popped Element: ', a, ' List remaining: ', my\_list)

my\_list.clear() *#empty the list*

print(my\_list)

else:

print( "Ok" )

**Output: -**

Select your Choice:

1. Functions Scoping

2. Recursion

3. List Mutability

3

[1, 2, 3, 'example', 3.132]

[1, 2, 3, 'example', 3.132, [555, 12]]

[1, 2, 3, 'example', 3.132, [555, 12], 234, 'more\_example']

[1, 'insert\_example', 2, 3, 'example', 3.132, [555, 12], 234, 'more\_example']

[1, 'insert\_example', 2, 3, 'example', [555, 12], 234, 'more\_example']

[1, 'insert\_example', 2, 3, [555, 12], 234, 'more\_example']

Popped Element: insert\_example List remaining: [1, 2, 3, [555, 12], 234, 'more\_example']

[]

**Assignment 4. Develop programs to understand object oriented programming using python.**

class Parent: *# define parent class*

parentAttr = 100

def \_\_init\_\_(self):

print ("Calling parent constructor")

def parentMethod(self):

print ("Calling parent method")

def setAttr(self, attr):

self.parentAttr = attr

def getAttr(self):

print ("Parent attribute :", self.parentAttr)

class Child(Parent): *# define child class*

def \_\_init\_\_(self):

print ("Calling child constructor")

def childMethod(self):

print ("Calling child method")

c = Child()

c.childMethod()

c.parentMethod()

c.setAttr(200)

c.getAttr()

**Output:-**

Calling child constructor

Calling child method

Calling parent method

Parent attribute : 200

**Assignment 5. Develop programs for data structure algorithms using python – searching, sorting and hash tables.**

def LinearSearch(lys, element):

for i in range (len(lys)):

if lys[i] == element:

return i

return -1

def BinarySearch(lys, val):

first = 0

last = len(lys)-1

index = -1

while (first <= last) and (index == -1):

mid = (first+last)//2

if lys[mid] == val:

index = mid

else:

if val<lys[mid]:

last = mid -1

else:

first = mid +1

return index

def bubblesort(list):

# Swap the elements to arrange in order

for iter\_num in range(len(list)-1,0,-1):

for idx in range(iter\_num):

if list[idx]>list[idx+1]:

temp = list[idx]

list[idx] = list[idx+1]

list[idx+1] = temp

def merge\_sort(unsorted\_list):

if len(unsorted\_list) <= 1:

return unsorted\_list

# Find the middle point and devide it

middle = len(unsorted\_list) // 2

left\_list = unsorted\_list[:middle]

right\_list = unsorted\_list[middle:]

left\_list = merge\_sort(left\_list)

right\_list = merge\_sort(right\_list)

return list(merge(left\_list, right\_list))

# Merge the sorted halves

def merge(left\_half,right\_half):

res = []

while len(left\_half) != 0 and len(right\_half) != 0:

if left\_half[0] < right\_half[0]:

res.append(left\_half[0])

left\_half.remove(left\_half[0])

else:

res.append(right\_half[0])

right\_half.remove(right\_half[0])

if len(left\_half) == 0:

res = res + right\_half

else:

res = res + left\_half

return res

def insertionSort(arr):

if (n := len(arr)) <= 1:

return

for i in range(1, n):

key = arr[i]

# Move elements of arr[0..i-1], that are

# greater than key, to one position ahead

# of their current position

j = i-1

while j >=0 and key < arr[j] :

arr[j+1] = arr[j]

j -= 1

arr[j+1] = key

def selectionSort(array, size):

for ind in range(size):

min\_index = ind

for j in range(ind + 1, size):

# select the minimum element in every iteration

if array[j] < array[min\_index]:

min\_index = j

# swapping the elements to sort the array

(array[ind], array[min\_index]) = (array[min\_index], array[ind])

print("Select your Choice:")

print("1. Linear Search")

print("2. Binary Search")

print("3. Bubble Sort")

print("4. Merge Sort")

print("5. Insertion Sort")

print("6. Selection Sort")

print("7. Hash Table")

choice = int( input() )

if( choice == 1 ):

print(LinearSearch([1,2,3,4,5,2,1], 5))

elif( choice == 2 ):

print(BinarySearch([10,20,30,40,50], 40))

elif( choice == 3 ):

list = [19,2,31,45,6,11,121,27]

bubblesort(list)

print(list)

elif( choice == 4 ):

unsorted\_list = [64, 34, 25, 12, 22, 11, 90]

print(merge\_sort(unsorted\_list))

elif( choice == 5 ):

arr = [12, 11, 13, 5, 6]

insertionSort(arr)

print(arr)

elif( choice == 6 ):

arr = [-2, 45, 0, 11, -9,88,-97,-202,747]

size = len(arr)

selectionSort(arr, size)

print(arr)

elif( choice == 7 ):

# Declare a dictionary

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

# Accessing the dictionary with its key

print ("dict['Name']: ", dict['Name'])

print ("dict['Age']: ", dict['Age'])

dict['Age'] = 8; # update existing entry

dict['School'] = "DPS School"; # Add new entry

print ("dict['Age']: ", dict['Age'])

print ("dict['School']: ", dict['School'])

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'

dict.clear(); # remove all entries in dict

del dict ; # delete entire dictionary

print ("dict['Age']: ", dict['Age'])

print ("dict['School']: ", dict['School'])

else:

print( "Ok" )

**Output:-**

Select your Choice:

1. Linear Search

2. Binary Search

3. Bubble Sort

4. Merge Sort

5. Insertion Sort

6. Selection Sort

7. Hash Table

3

[2, 6, 11, 19, 27, 31, 45, 121]

**Assignment 6. Develop programs to learn regular expressions using python.**

import re

#Return a list containing every occurrence of "ai":

txt1 = "The rain in Spain"

x1 = re.findall("ai", txt1)

print(x1)

txt2 = "The rain in Spain"

x2 = re.search("\s", txt2)

print("The first white-space character is located in position:", x2.start())

#Split the string at every white-space character:

txt3 = "The rain in Spain"

x3 = re.split("\s", txt3)

print(x3)

#Replace all white-space characters with the digit "9":

txt4 = "The rain in Spain"

x4 = re.sub("\s", "9", txt4)

print(x4)

txt5 = "The rain in Spain"

x5 = re.search("ai", txt5)

print(x5)

#Search for an upper case "S" character in the beginning of a word, and print the word:

txt6 = "The rain in Spain"

x6 = re.search(r"\bS\w+", txt6)

print(x6.group())

**Output:-**

['ai', 'ai']

The first white-space character is located in position: 3

['The', 'rain', 'in', 'Spain']

The9rain9in9Spain

<re.Match object; span=(5, 7), match='ai'>

Spain

**Assignment 7. Demonstrate the concept of exception handling using try/except/else Statement, Unified try/except/finally, try/finally Statement, raise Statement, assert Statement, catch multiple specific exceptions**

# Python code to illustrate

# working of try()

def divide(x, y):

try:

# Floor Division : Gives only Fractional

# Part as Answer

result = x // y

except ZeroDivisionError:

print("Sorry ! You are dividing by zero ")

else:

print("Yeah ! Your answer is :", result)

finally:

# this block is always executed

# regardless of exception generation.

print('This is always executed')

# Look at parameters and note the working of Program

divide(3, 2)

divide(3, 0)

# A python program to create user-defined exception

# class MyError is derived from super class Exception

class MyError(Exception):

# Constructor or Initializer

def \_\_init\_\_(self, value):

self.value = value

# \_\_str\_\_ is to print() the value

def \_\_str\_\_(self):

return(repr(self.value))

try:

raise(MyError(3\*2))

# Value of Exception is stored in error

except MyError as error:

print('A New Exception occurred: ', error.value)

x = "hello"

#if condition returns False, AssertionError is raised:

assert x == "goodbye", "x should be 'hello'"

**Output: -**

Yeah ! Your answer is : 1

This is always executed

Sorry ! You are dividing by zero

This is always executed

A New Exception occurred: 6

**Assignment8. Demonstrate the concept of String-Based Exceptions, Class-Based Exceptions and Nesting Exception handlers.**

#String-Based Exceptions

try:

print(1 + '3')

except Exception as e:

error\_message = str(e)

print(error\_message)

print(type(error\_message))

#Class-Based Exceptions

class LowAgeError(Exception):

def \_\_init\_\_(self):

pass

def \_\_str\_\_(self):

return 'The age must be greater than 18 years'

class Employee:

def \_\_init\_\_(self, name, age):

self.name = name

if age < 18:

raise LowAgeError

else:

self.age = age

def display(self):

print('The name of the employee: ' + self.name + ', Age: ' + str(self.age) +' Years')

try:

e1 = Employee('Subhas', 25)

e1.display()

e2 = Employee('Anupam', 12)

e1.display()

except LowAgeError as e:

print('Error Occurred: ' + str(e))

#Nested

x = 10

y = 0

try:

print("outer try block")

try:

print("nested try block")

print(x / y)

except TypeError as te:

print("nested except block")

print(te)

except ZeroDivisionError as ze:

print("outer except block")

print(ze)

**Output: -**

unsupported operand type(s) for +: 'int' and 'str'

<class 'str'>

The name of the employee: Subhas, Age: 25 Years

Error Occurred: The age must be greater than 18 years

outer try block

nested try block

outer except block

division by zero