St. Francis Institute of Technology SE EXTC A & B Skill Lab: Python programming

**Experiment – 6: Inheritance and Linked List**

|  |
| --- |
| **Aim:** To demonstrate inheritance and Linked list in python.  **Theory:** |

|  |
| --- |
| ***Inheritance*** allows to create a hierarchy of classes that share a set of properties and methods byderiving a class from another class. Inheritance is the capability of one class to derive or inheritthe properties from another class. It provides the reusability of a code. Different types ofInheritance are: |

|  |
| --- |
| ● Single inheritance: When a child class inherits from only one parent class, it is calledsingle inheritance.  ● Multiple inheritances: When a child class inherits from multiple parent classes, it is calledmultiple inheritances.  ● Multilevel inheritance: When a child class is derived from another child class it is knownas multilevel inheritance.  ● Hierarchical inheritance More than one derived class are created from a single base. ● Hybrid inheritance: This form combines more than one form of inheritance. Basically, itis a blend of more than one type of inheritance. |

|  |  |
| --- | --- |
| A ***linked list*** is a linear data structure that includes a series of connected nodes. Linked lists can be of multiple types: | |
|  | ● Singly: Each node has data and a pointer to the next node.  ● Doubly: Pointer for the previous node is added and thus it can traverse in forward as well as backward direction.  ● Circular: The last element is linked to the first element. |

|  |
| --- |
| Linked List allows Dynamic memory allocation and can be used in the Implementation of stackand queue. |

**Conclusion:**

Skill Lab: Python Programming Lab Manual S.E. EXTC 2022-23

St. Francis Institute of Technology SE EXTC A & B Skill Lab: Python programming

**Task for submission:**

(Write comments for every statement of the program)

|  |  |  |
| --- | --- | --- |
|  | 1. Write a program to demonstrate single inheritance to calculate cube of a number. 2. Write a program to demonstrate inheritance to find the area of various shapes 3. Write a program to demonstrate addition of two numbers using multiple inheritance 4. Write a program to insert a node in the middle of a linked list.  5. Explain the difference between pop( ), push( ) and peek( ).  6. Write a program to implement stack data structure using linked list in python using Last In First Out technique. | |
|  |  |  |
|  | | |
|  |  | |

Skill Lab: Python Programming Lab Manual S.E. EXTC 2022-23

**1.**

class Number:

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

self.num = num

class Cube(Number):

def find\_cube(self):

return self.num \*\* 3

num = int(input("Enter a number: "))

c = Cube(num)

print(f"The cube of {num} is {c.find\_cube()}")

**OUTPUT:**

****

**2.**

class Shape:

def \_\_init\_\_(self, shape\_name):

self.shape\_name = shape\_name

def get\_name(self):

return self.shape\_name

class Rectangle(Shape):

def \_\_init\_\_(self, width, height):

super().\_\_init\_\_("Rectangle")

self.width = width

self.height = height

def get\_area(self):

return self.width \* self.height

class Circle(Shape):

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

super().\_\_init\_\_("Circle")

self.radius = radius

def get\_area(self):

return 3.14 \* self.radius \* self.radius

class Triangle(Shape):

def \_\_init\_\_(self, base, height):

super().\_\_init\_\_("Triangle")

self.base = base

self.height = height

def get\_area(self):

return 0.5 \* self.base \* self.height

# create objects of Rectangle, Circle and Triangle class and call their get\_area() methods

rect = Rectangle(5, 10)

print(f"The area of {rect.get\_name()} is {rect.get\_area()}")

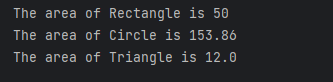
circle = Circle(7)

print(f"The area of {circle.get\_name()} is {circle.get\_area()}")

tri = Triangle(4, 6)

print(f"The area of {tri.get\_name()} is {tri.get\_area()}")

**OUTPUT:**

****

**3.**

class Num1:

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

self.num1 = None

def input1(self):

self.num1 = 10

class Num2:

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

self.num2 = None

def input2(self):

self.num2 = 2

class Result(Num1, Num2):

def printresult(self):

super().input1()

super().input2()

print(self.num1 + self.num2)

r = Result()

r.printresult()

**OUTPUT:**

****

**4.**

class Node:

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

self.data = data

self.next = None

class LinkedList:

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

self.head = None

def add\_node(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

else:

current = self.head

while current.next is not None:

current = current.next

current.next = new\_node

def print\_list(self):

current = self.head

while current is not None:

print(current.data, end=' ')

current = current.next

print()

def insert\_middle(self, data):

if self.head is None:

self.head = Node(data)

else:

# Find the middle node using slow and fast pointers

slow = self.head

fast = self.head

while fast is not None and fast.next is not None:

fast = fast.next.next

slow = slow.next

# Insert the new node after the middle node

new\_node = Node(data)

new\_node.next = slow.next

slow.next = new\_node

linked\_list = LinkedList()

linked\_list.add\_node(1)

linked\_list.add\_node(2)

linked\_list.add\_node(4)

linked\_list.add\_node(5)

linked\_list.print\_list()

linked\_list.insert\_middle(3)

linked\_list.print\_list()

**OUTPUT:**

****

**6.**

class Node:

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

self.data = data

self.next = None

class Stack:

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

self.head = None

def is\_empty(self):

return self.head is None

def push(self, data):

new\_node = Node(data)

new\_node.next = self.head

self.head = new\_node

def pop(self):

if self.head is None:

return None

else:

popped\_node = self.head

self.head = self.head.next

return popped\_node.data

def peek(self):

if self.head is None:

return None

else:

return self.head.data

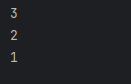
stack = [1, 2, 3]

print(stack.pop()) # remove and print the top item (3)

print(stack.pop()) # remove and print the top item (2)

print(stack.pop()) # remove and print the top item (1)

**OUTPUT:**

****