ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

Academic Year: **2020-21** Subject: **Skill Base Lab Course: Python Programming**

Class/ Branch/ Sem: **SE/ CMPN A&B/ IV** PID**: 192120** Roll no.**:42**

Exp2_Aim: To create functions, classes and objects using python and to demonstrate exception handling and inheritance.

THEORY:

1. List the different features of OOP in python. Explain each of them with examples.

Ans:

1. Create a class in python:

A class is a blueprint for the object, to create a class we will use the class keyword and from the class, we construct instances.

2. Create an object in python:

In python, an object is a real-world entity that has its state and behavior, everything in python is an object. To create an object we can use the class name.

3. Instance Attribute in python:

In python, instance attribute is an attribute or properties which is attached to an instance of the class. The instance is accessed by using the dot notation.

4. Class Attribute in python:

A class attribute is an attribute whose value remains the same for all instances of a class is known as a class attribute. It is defined at a class level rather than inside the method. The value of the class attribute is shared by all objects.

5. The init function in python:

In python, init () is a built-in function, and all the classes have a function called init which is always executed when the class is initiated. We use the init_() to assign the values to object properties.

6. Method in python:

The method in python is similar to a function, a method is defined inside the body of a class and it is used to define the behavior of an object.

7. Self Parameter in Python:

The self parameter is used to reference the current instance of a class, also by using "self" we can access the attributes and methods of a class in python. We can use another parameter name instead of a "self" but it should be the first parameter of any function in class.

8. Inheritance:

Inheritance is the process in which class inherits all the properties and methods from another class. The new class is called derived class or child class and the one from which it derived is called parent class or base class.

9. Types of Inheritance in Python

Types of inheritance depend upon the number of child and parent classes involved. There are four types of inheritance in python.

i. Single Inheritance
ii. Multiple Inheritance
iii. Multilevel Inheritance
iv. Hierarchical Inheritance
v. Hybrid Inheritance

i. **Single Inheritance:** In single inheritance child class inherits only a single parent class.

ii. **Multiple Inheritance:** In multiple inheritance a child class inherits from more than one parent class.

iii. **Multilevel Inheritance:** In multilevel inheritance child class becomes a parent class for another child class

iv. **Hierarchical Inheritance:** In hierarchical inheritance more than one derived class inherits properties from the parent class.

v. **Hybrid Inheritance:** Hybrid inheritance involves multiple types of inheritance taking place in a single program.

10. Method Overriding in python:

- Python method overriding means creating two methods with the same name and the same number of parameters but the different print messages.
- Here, the method overriding allows us to change or override the parent class function in the child class. Method overriding is an example of run time polymorphism.

11. Method Overloading in python:

- Python method overloading means we can have the same name but different arguments and a method can have one or more arguments.
- Calling the same method in different ways is called method overloading in python. Method overloading is an example of compile-time polymorphism.

12. Polymorphism in Python:

- Polymorphism in Python means more than one form. Also, we can say it is a condition of occurrence in different forms.
- Polymorphism is one of the important concepts in programming. For example, we know that the "+" operator is used for adding two integer numbers, and for the string, the same " + " operator is used for string concatenation.

13. Encapsulation in python:

1. Encapsulation is the process of wrapping up variables and methods into a single unit, it is the fundamental concepts of object-oriented programming in Python.
2. In python, though there is no explicit access modifier by using ( ) double underscores we can make the variable private.

Academic Year: **2020-21** Subject: **Skill Base Lab Course: Python Programming**

Class/ Branch/ Sem: **SE/ CMPN A&B/ IV** PID**: 192120** Roll no.**:42**

14. Abstraction in python:

1. In python, abstraction is used for hiding the internal details and showing the functionalities. Abstraction means hiding the real implementation and knowing how to use it as a user and it is achieved by using abstract classes and interfaces.
2. An abstract class is a class that provides incomplete functionality and the interface provides the method names without method bodies.

(All the examples are done in the implementation part)

3. Explain importance of exception handling in Python. Explain various keywords used in exception handling with examples.

Ans:

Sr. No.	Exception with Description
1.	**SystemExit** - Raised by the sys.exit() function.
2.	**ArithmeticError** - Base class for all errors that occur for numeric calculation.
3.	**OverflowError** - Raised when a calculation exceeds maximum limit for a numeric type.
4.	**FloatingPointError** - Raised when a floating point calculation fails.
5.	**ZeroDivisionError** - Raised when division or modulo by zero takes place for all numeric types.
6.	**AttributeError** - Raised in case of failure of attribute reference or assignment.
7.	**KeyboardInterrupt** - Raised when the user interrupts program execution, usually by pressing Ctrl+c.

8.	**IndexError** - Raised when an index is not found in a sequence.
9.	**NameError** - Raised when an identifier is not found in the local or global namespace.
10.	**EnvironmentError** - Base class for all exceptions that occur outside the Python environment.
11.	**SyntaxError** - Raised when there is an error in Python syntax.
12.	**TypeError** - Raised when an operation or function is attempted that is invalid for the specified data type.

Python uses try and except keywords to handle exception. Both keywords are followed by idented blocks.

try...except blocks:

Syntax:-

```
try :
    #statement in try block
except :
    #executed when error in try block
```

Catch Specific Error Type:

```
try:
    a=5
    b='0'
    print(a+b)
except TypeError:
    print('Unsuported operation')
```

print("Out of try except blocks")

Output:

Unsuported operation

Out of try except blocks

Multiple...except blocks:

Code:

```
try:
    a=5
    b='0'
    print(a+b)
except TypeError:
    print('Unsuported operation')
except ZeroBydivision:
    print('Zero by division is not allowed')
print('Out of try except blocks')
```

Output:

Zero by division is not allowed

Out of try except blocks

Academic Year: **2020-21** Subject: **Skill Base Lab Course: Python Programming**

Class/ Branch/ Sem: **SE/ CMPN A&B/ IV** PID**: 192120** Roll no.**:42**

IMPLEMENTATION:

- Write a program in python to illustrate various features of OOP in python

Code:

```
#creating class
class plane:
   roll = 1
print(plane)
```

Output:

```
<class '__main__.plane'>
```

Code:

```
#method in python
class crew:
   def __init__ (self, name, roll):
      self.name = name
      self.roll = roll
   def function(self):
      print("Welcome to python " + self.name)

a1 = crew("Andrei", 25)
a1.function()
```

Output:

```
Welcome to python Andrei
```

Code:

```
#create object and class
class plane():
   def __init__ (self,name,id,salary):
      self.name = name
      self.id = id
      self.salary = salary

obj1 = plane("Boeing",101,12500)
print(obj1.__dict__)
```

Output:

```
{'name': 'Boeing', 'id': 101, 'salary': 12500}
```

Academic Year: **2020-21** Subject: **Skill Base Lab Course: Python Programming**

Class/ Branch/ Sem: **SE/ CMPN A&B/ IV** PID**: 192120** Roll no.**:42**

Code:

```
#single inheritance in python
class Teacher():
    def myfirst(self):
        print('This is Parent Class')

class Child(Teacher):
    def mysecond(self):
        print('This is Child Class')

obj = Child()
obj.myfirst()
obj.mysecond()
```

Output:

```
This is Parent Class
This is Child Class
```

Code:

```
#polymorphism
val1 = 30
val2 = 20
print(val1-val2)
```

Output:

```
10
```

Code:

```
#Encapsulation in python
class Plane:
    def __init__ (self, name, salary):
        self.name = name
        self. salary = salary
    def disp(self):
        print(self.name)
        print(self. salary)

plane = Plane('ANTONOV', 5000000)
plane.disp()
print(plane.name)
```

Output:

```
ANTONOV
5000000
ANTONOV
```

Academic Year: **2020-21** Subject: **Skill Base Lab Course: Python Programming**

Class/ Branch/ Sem: **SE/ CMPN A&B/ IV** PID**: 192120** Roll no.**:42**

- Write a program in python to demonstrate exception handling in python. (Hint: use multiple except block, user defined exception)

Code:

a). Division by zero:

Code:

```
try:
        a = int(input("Enter a:"))
        b = int(input("Enter b:"))
        c = a/b
except:
        print("Can't divide with zero")
```

Output:

```
Enter a:10
Enter b:0
Can't divide with zero
```

b). User Defined Exception:

Code:

```
class User_Error(Exception):
  def init(self, value):
    self.value = value
  def str(self):
    return(repr(self.value))
try:
  raise(User_Error("User Defined Error"))
except User_Error as error:
  print('A New Exception Has Spawned:',error.value)
```

Output:

```
A New Exception occured: User defined error
```

C**onclusion**:

Successfully learnt to use various OOP features and exception handling in python.