Experiment No : 3

Dated : 28th Jan 2022

Aim

Exploring Python - Exception Handling & Inheritance

Theory

- **Exception Handling**
  - Difference between Syntax Error & Exception (at least 2)
  - Common types of Built-In Exceptions. (enlist 5)
  - Syntax for Try-except-else-Finally clause
  - Syntax for Raising Exceptions
  - User Defined Exceptions (explain)
- **Inheritance**
  - Significance of Inheritance (at least 3)
  - Types of Inheritance (enlist them with diagrams)

Handwritten Theory

Theory:

- Exception Handling.

① Syntax error - Mistakes done while writing statements.
Compilers can detect these type of errors.

Exceptions - Mostly occur due to improper user entries or usage of system resources by program.

② @ Base Exception
ⓑ Exception
ⓒ Arithmetic Exception
ⓓ Buffer Error
ⓔ Lookup error

③
```
try:
    # Statements
except:
    # optional statements
    # Handling
else:
    # execute if no exception
finally:
    # statements (always executed)
```

④ try:
raise "Exception"
except:
optional statements.
raise # to determine whether exception was raised or not.

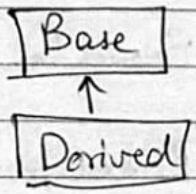
⑤ To create customized error messages, we use userdefined exceptions. We can create user defined exceptions as checked or unchecked exceptions. We can create user defined exceptions that extend Exception class or subclasses of checked exceptions so that undefined exception becomes checked.

• Inheritance.

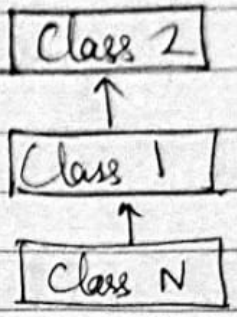
① Helps developers to reuse program/objects.
Low development time & cost.
Reduces redundancy of code & helps improve extensibility.

② Types of inheritance are:

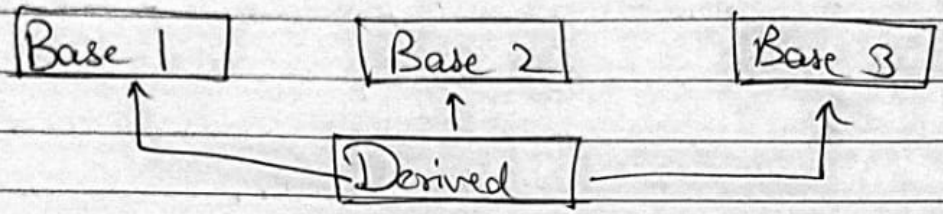
ⓐ Single inheritance



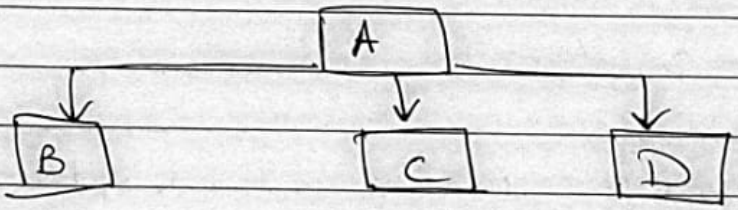
(b) Multi-level inheritance



(c) Multiple inheritance



(d) Hierarchical inheritance



Programs to be performed :

Exceptions (attempt any 1)

1.

- Create User Defined Exception Class.
- Write a new class (says MyExceptionClass) for custom exceptions and inherit it from an in-build Exception class.
- Raise ZeroDivisionError Exception.
- Incorporate scenarios to catch NameError, ValueError and Arithmetic Error using try-except-else clause.
- Display that “The program is successfully executed” via the finally clause.

```
class MyExceptionClass(Exception):
  def __init__(self,args):
    self.args = args

try:
  print("Beginning of the Program\n")
  raise ZeroDivisionError()

except ZeroDivisionError:
  print("ZeroDivisionError occured")
except NameError:
  print("NameError occured")
except ValueError:
  print("ValueError occured")
except ArithmeticError:
  print("ArithmeticError occured")

else:
  print("No Error found")
finally:
  print("The program is successfully executed")
```

```
Beginning of the Program

ZeroDivisionError occured
The program is successfully executed
```

2.

Interactive Calculator : User input is assumed to be a formula that consists of a number, an operator (at least + and -), and another number, separated by white space (e.g. 1 + 1). Split user

input using str.split(), and check whether the resulting list is valid:

- If the input does not consist of 3 elements, raise a FormulaError, which is a custom Exception.
- Try to convert the first and third input to a float (like so: float_value = float(str_value)). Catch any ValueError that occurs, and instead raise a FormulaError
- If the second input is not '+' or '-', again raise a FormulaError
- If the input is valid, perform the calculation and print out the result. The user is then prompted to provide new input, and so on, until the user enters quit.

Inheritance (attempt any 2 out of 3)

1.

- Create a Vehicle class with name, max_speed, capacity and mileage instance attributes.
- Create a Bus child class that inherits from the Vehicle class.
- Give the capacity argument of Bus.seating_capacity() a default value of 50. Define a class attribute ”color” with a default value white. i.e, Every Vehicle should be white.
- The default fare charge of any vehicle is seating capacity * 100.

> 1. If Vehicle is Bus instance, we need to add an extra 10% on full fare as a maintenance charge. So total fare for bus instance will become the final amount = total fare + 10% of the total fare. Override the fare() method of a Vehicle class in Bus class.
> 2. Write a code snippet to determine which class a given Bus object belongs to.
> 3. Create an object named School_bus and determine if School_bus is also an instance of the Vehicle class.

```
#Create a Vehicle class with name, max_speed, capacity and mileage instance attributes.
class Vehicle:
  max_speed = 0
  seating_capacity = 0
  mileage = 0
  fare = 0
  def __init__(self, max_speed = 0, seating_capacity = 0, mileage = 0):
    self.max_speed = max_speed
    self.seating_capacity = seating_capacity
    self.mileage = mileage

  def fare_calc(self):
    self.fare = 100 * self.seating_capacity
    return self.fare
```

```
#Create a Bus child class that inherits from the Vehicle class. Give the capacity argum
class Bus(Vehicle):
  capacity = 50
  color = 'white' #Define a class attribute ”color” with a default value white. i.e, Ev
  bus_fare = 0
  def fare_calc(self):
    super().fare_calc()
    self.bus_fare = 1.1*self.fare
    return self.bus_fare
```

```
#Write a code snippet to determine which class a given Bus object belongs to.
Commercial = Bus( 50 , 50, 50)
print(Commercial.fare_calc())

print(type(Commercial))
```

```
5500.0
<class '__main__.Bus'>
```

```
#Create an object named School_bus and determine if School_bus is also an instance of t
School_Bus = Bus()
print(isinstance(School_Bus, Vehicle) )
```

```
True
```

2. Implement the any of the following scenarios in Python

Two Photos of Hierarchichal Inheritances

3. Implement the following Scenario :

- class Shoe:
  - Attributes: self.color, self.brand
- class Converse(Shoe): # Inherits from Shoe
  - Attributes: self.lowOrHighTop, self.tongueColor, self.brand = "Converse"
- class CombatBoot(Shoe): # Inherits from Shoe
  - Attributes: self.militaryBranch, self.DesertOrJungle
- class Sandal(Shoe): # Inherits from Shoe
  - Attributes: self.openOrClosedToe, self.waterproof

```
class Shoe:
  def __init__(self, color = 'NA', brand = 'NA'):
    self.color = color
    self.brand = brand

class Converse(Shoe): # Inherits from Shoe
  def __init__(self, color = 'BrownMunde', lowOrHighTop = 'Low', tongueColor = 'Red', b
    super().__init__(color, brand)
    self.lowOrHighTop = lowOrHighTop
    self.tongueColor = tongueColor


class CombatBoot(Shoe): # Inherits from Shoe
  def __init__(self, color='Camoflague', militaryBranch='Marine' , DesertOrJungle = 'Ju
    super().__init__(color, brand)
    self.militaryBranch = militaryBranch
    self.DesertOrJungle = DesertOrJungle


class Sandal(Shoe): # Inherits from Shoe
  def __init__(self, color='Black', brand = "Sandals"):
    super().__init__(color, brand)


ok = Converse(color='blue')
print(type(ok))
```

```
<class '__main__.Converse'>
```