Noida Institute of Engineering and Technology, Greater Noida

Object Oriented Concepts

Unit: 2

Problem Solving Using Advanced Python

Course Details
(B Tech 2nd Sem /1st Year)

CONTENTS

- Introduction to the Specialization
- Inheritance
- Types of Inheritance
- Invoking the parent Class Method
- Method Overriding
- Abstract Class
- MRO & Super()
- Polymorphism
- Introspection: Introspection types , Introspection objects
- Introspection scopes, Inspect modules, Introspect tools

COURSE OBJECTIVES

After you have read and studied this module, you should be able to:

- To learn the Object-Oriented Concepts in Python.
- To learn the concept of reusability through inheritance polymorphism and Introspection tools.
- To impart the knowledge of functional programming.
- To learn the concepts of designing graphical user interfaces.
- To explore the knowledge of standard Python libraries.

COURSE OUTCOME

Course Outcome (CO)	At the end of course , the student will be able :	Bloom's Knowledge Level (KL)
CO1	Define classes and create instances in python.	K1, K2
CO2	Implement concept of inheritance and polymorphism using python.	**K3**
CO3	Implement functional programming in python.	K3
CO4	Create GUI based Python application.	K2
CO5	Apply the concept of Python libraries to solve real world problems.	K3,K6

CO-PO and PSO Mapping

Mapping of Course Outcomes and Program Outcomes:

Programming in Advanced Python												
CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO 12
CO1	3	3	3	3	2	2	1	-	1	-	2	2
CO2	**3**	**3**	**3**	**3**	**2**	**2**	**1**	**-**	**1**	**1**	**2**	**2**
CO3	3	3	3	3	3	2	2	-	2	1	2	3
CO4	3	3	3	3	3	2	2	1	2	1	2	3
CO5	3	3	3	3	3	2	2	1	2	1	2	2
Average	3	3	3	3	2.6	2	1.6	0.4	1.6	0.8	2	2.2

Mapping of Course Outcomes and Program Specific Outcomes

Programming in Advanced Python				
CO.K	PSO1	PSO2	PSO3	PSO4
CO1	2	3	2	2
CO2	**2**	**3**	**2**	**2**
CO3	2	3	3	3
CO4	3	3	3	3
CO5	3	3	3	3
Average	2.4	5	2.6	2.6

Prerequisite and Recap(CO2)

- Operators
- Loop
- Method
- Variables
- Class , Objects
- Constructor

Topic Objectives(CO2)

After you have read and studied this topic, you should be able to:

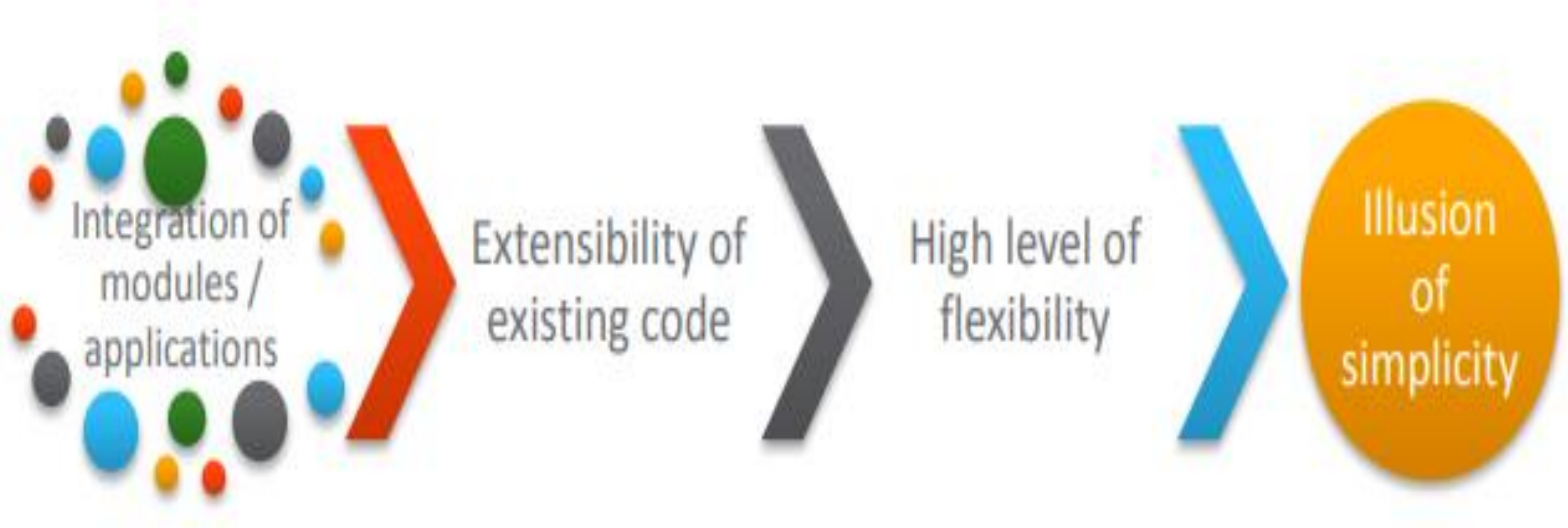
- Understand the concept of Object-Oriented Systems.
- Create class and objects.
- Understand the concept of Inheritance and its types.
- Able to know about Constructor.

Introduction to the Specialization(CO2)

- Python is used in Data Science Field.
- Python is implemented in IOT Field.
- Python Specialization in Artificial Intelligence tools.
- The python tools is also used in web Designing ,web crawling.
- Python is easily implemented in Machine Learning.
- We can specialized in Python Data Science tools.
- It is mostly used in Cloud Computing platform also.
- The specialization of Python is storing Data.
- The Specialization of Python is in Libraries of python.

Need for Object Oriented Approach (CO2)

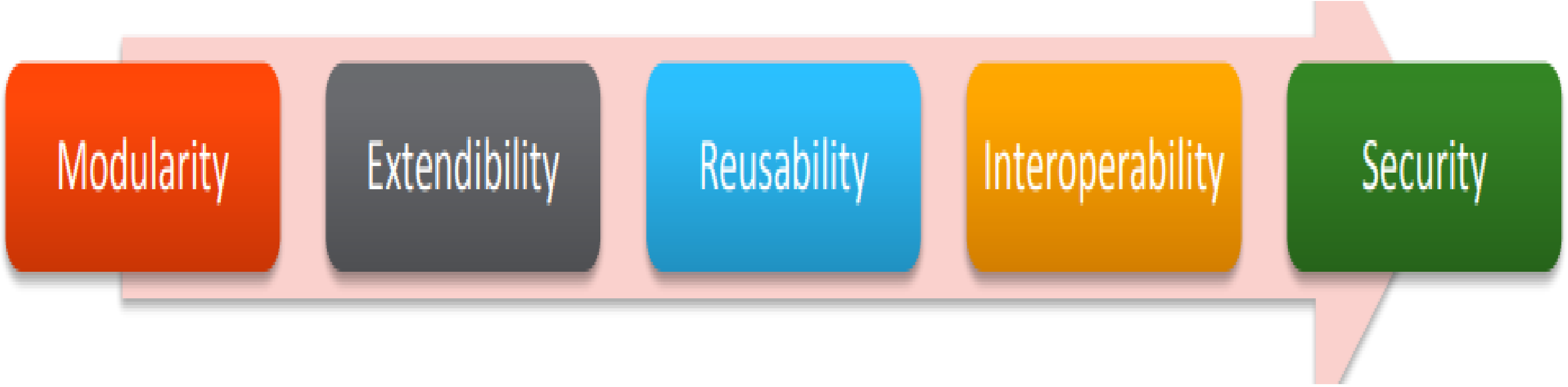
- Challenges in developing a business application.


Integration of modules / applications
Extensibility of existing code
High level of flexibility
Illusion of simplicity

- If these challenges are not addressed, it may lead to software crisis.

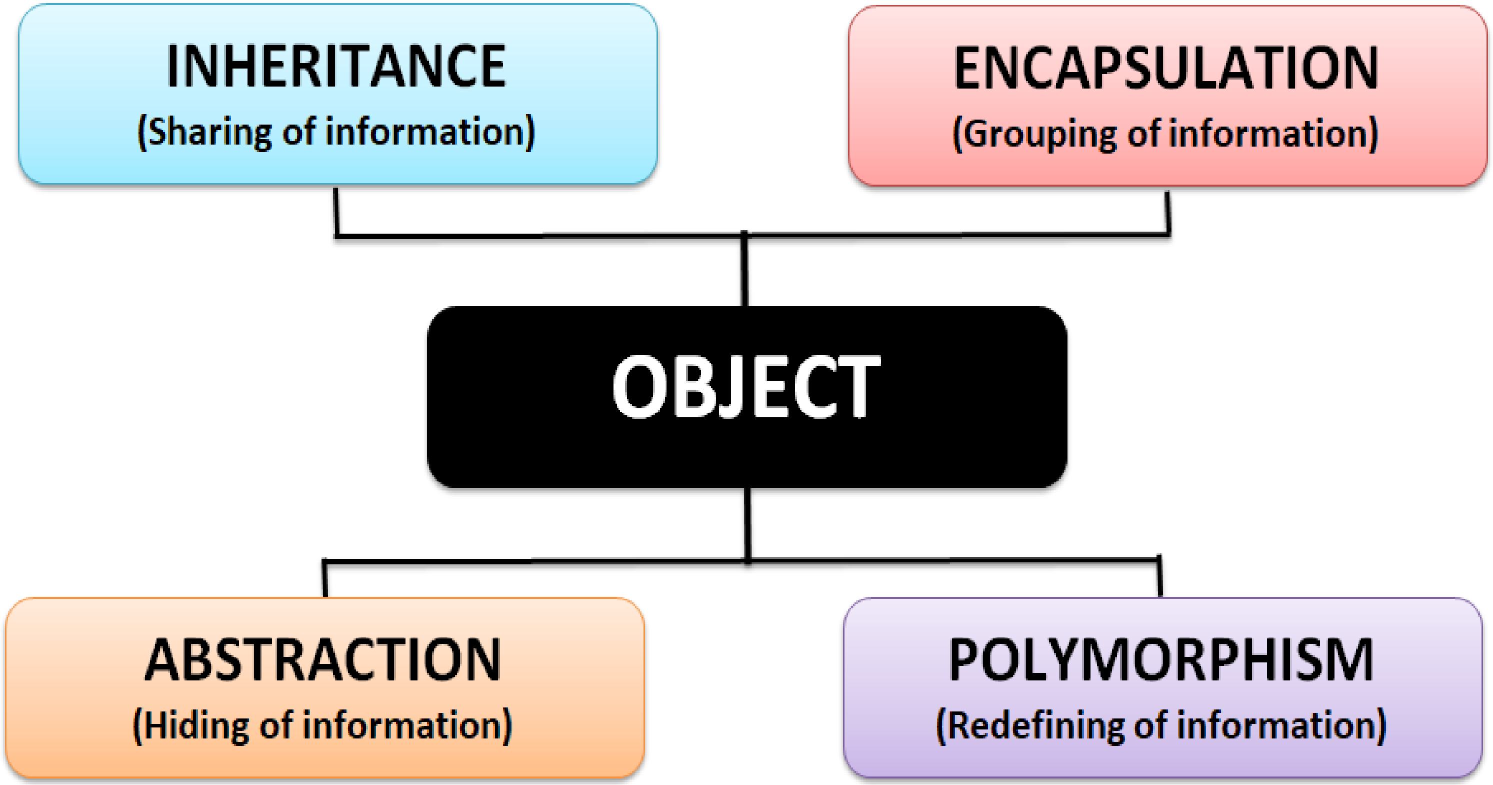
Need for Object Oriented Approach (CO2)

- Features needed in the business application to meet these challenges.



- Challenges can be addressed using object-oriented approach.

Pillars of OOPS (CO2)



INHERITANCE
(Sharing of information)

ENCAPSULATION
(Grouping of information)

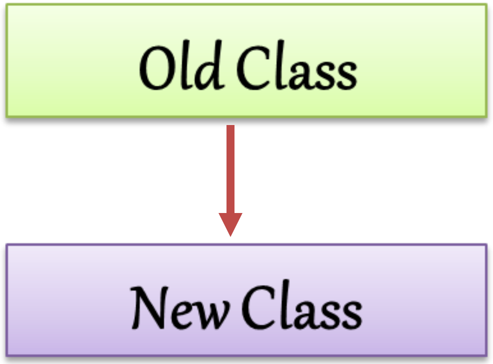
OBJECT

ABSTRACTION
(Hiding of information)

POLYMORPHISM
(Redefining of information)

Inheritance (CO2)

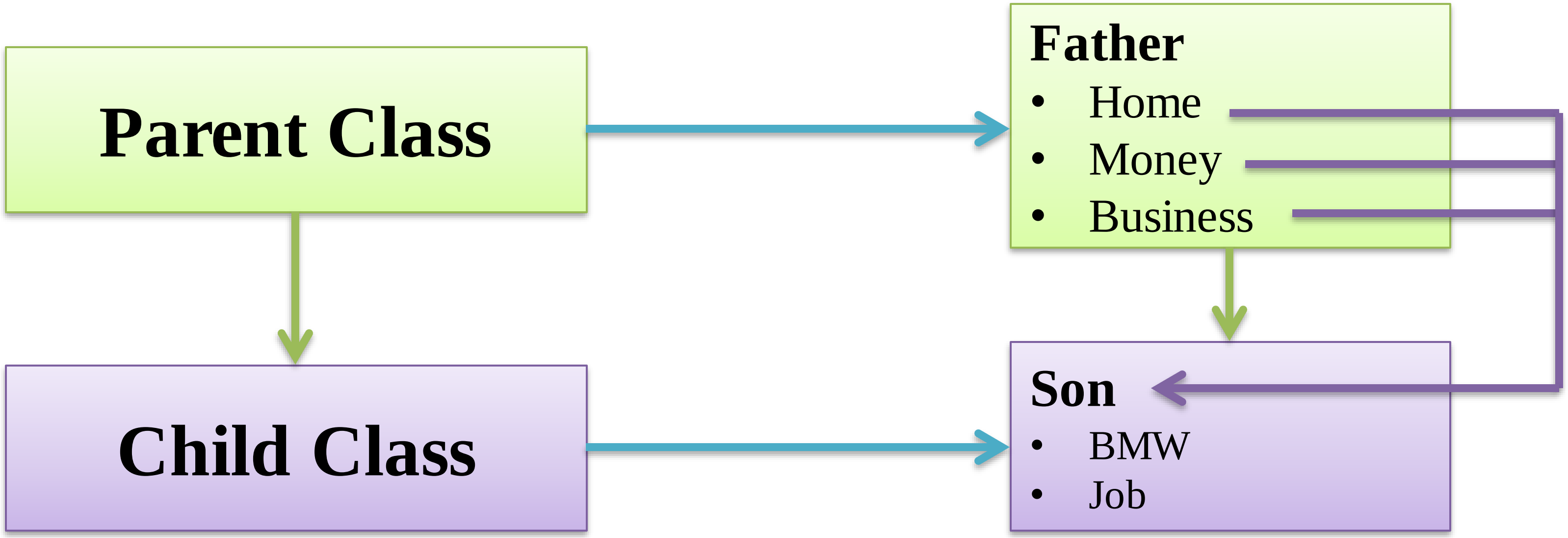
- The mechanism of deriving a new class from an old one (existing class) such that the new class inherit all the members (variables and methods) of old class is called inheritance or derivation.
- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class.



Super Class and Sub Class (CO2)

The old class is referred to as the Super class and the new one is called the Sub class.

- Parent Class - Base Class or Super Class
- Child Class - Derived Class or Sub Class



Advantage of Inheritance (CO2)

- All classes in python are built from a single super class called ‘object’ so whenever we create a class in python, object will become super class for them internally.

```
class Mobile(object):

class Mobile:
```

- The main advantage of inheritance is code reusability.
- Python also allows the classes to inherit commonly used attributes and methods from other classes through inheritance.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Type of Inheritance (CO2)

- Single Inheritance

- Multi-level Inheritance

- Hierarchical Inheritance

- Multiple Inheritance

Declaration of Child Class(CO2)

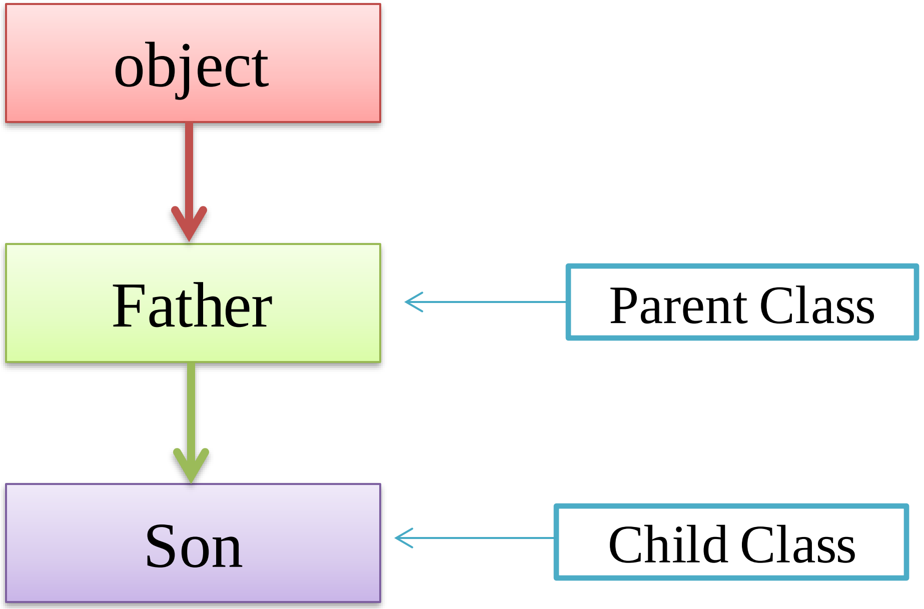
```
class ChildClassName (ParentClassName) :
        members of Child class
```

```
class Mobile (object) :
        members of Child class
```

```
class Mobile :
        members of Child class
```

Single Inheritance(CO2)

If a class is derived from one base class (Parent Class), it is called Single Inheritance.



Single Inheritance Syntax(CO2)

Syntax:-

```
class ParentClassName(object):
        members of Parent Class

class ChildClassName(ParentClassName):
        members of Child Class
```

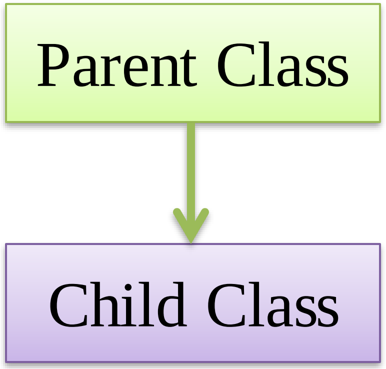
Example:-

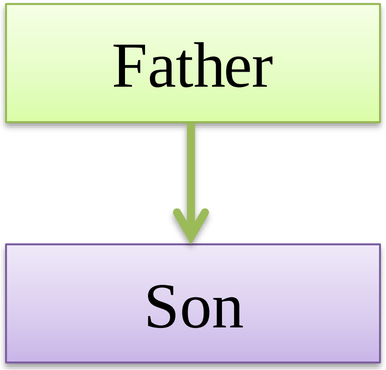
```
class Father:
        members of class Father
```

Parent Class

```
class Son (Father):
        members of class Son
```

Child Class





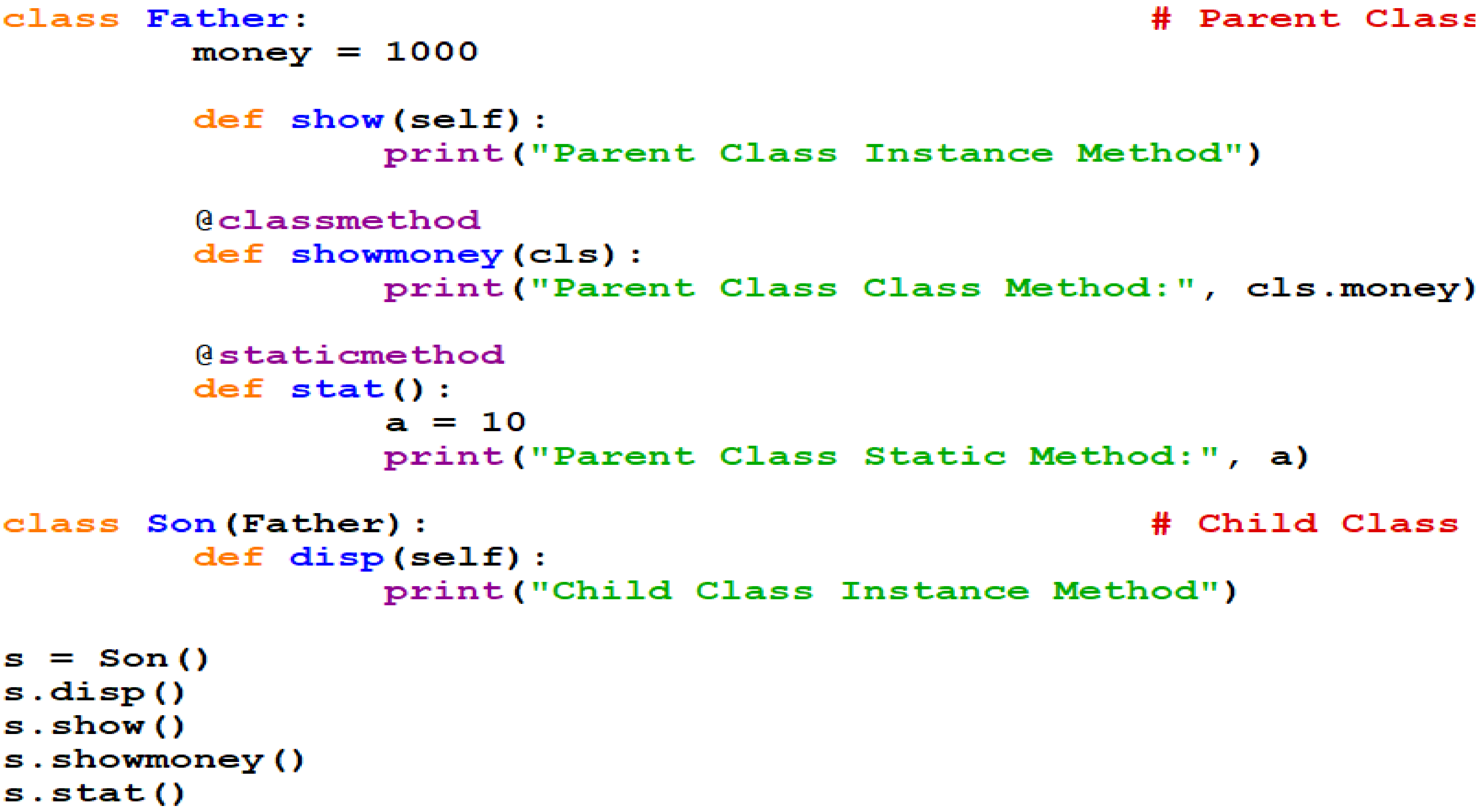
Single Inheritance(CO2)

- We can access Parent Class Variables and Methods using Child Class Object

- We can also access Parent Class Variables and Methods using Parent Class Object

- We can not access Child Class Variables and Methods using Parent Class Object

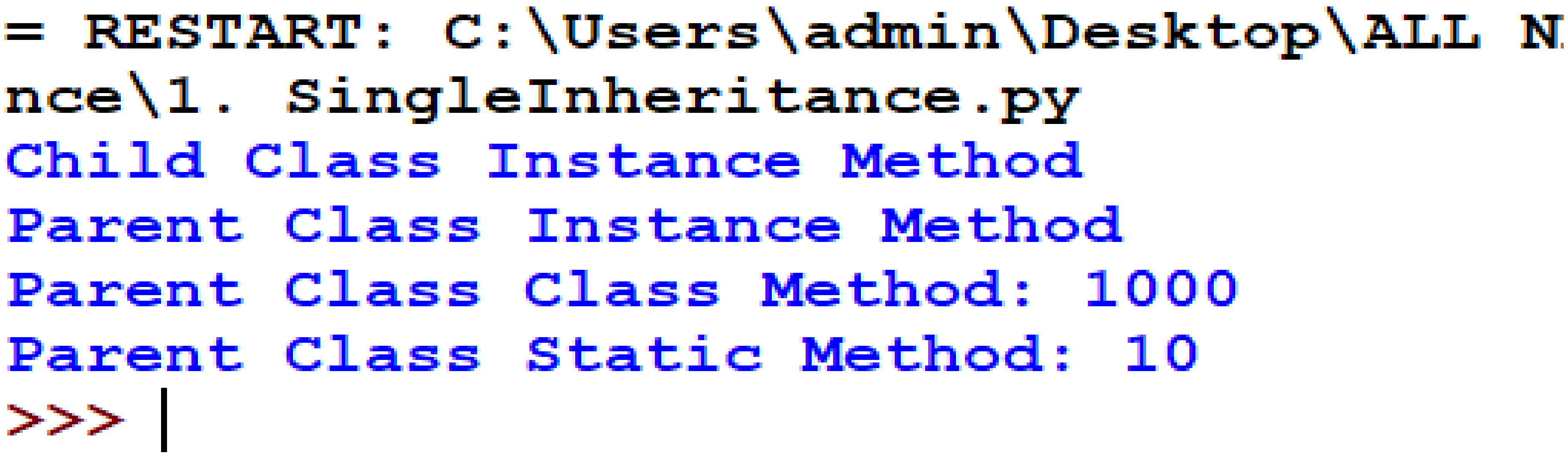
Single Inheritance Program (CO2)



```
class Father:                                   # Parent Class
        money = 1000

        def show(self):
                print("Parent Class Instance Method")

        @classmethod
        def showmoney(cls):
                print("Parent Class Class Method:", cls.money)

        @staticmethod
        def stat():
                a = 10
                print("Parent Class Static Method:", a)

class Son(Father):                              # Child Class
        def disp(self):
                print("Child Class Instance Method")

s = Son()
s.disp()
s.show()
s.showmoney()
s.stat()
```

Single Inheritance Program Output (CO2)

Output



```
= RESTART: C:\Users\admin\Desktop\ALL N
nce\1. SingleInheritance.py
Child Class Instance Method
Parent Class Instance Method
Parent Class Class Method: 1000
Parent Class Static Method: 10
>>> |
```

Daily MCQs

1. A class can serve as base class for many derived classes.

A. True

B. False

Answer: A

2. Which of the following is not a type of inheritance?

A. Double-level

B. Multi-level

C. Single-level

D. Multiple

Answer: A

3. What does single-level inheritance mean?

A. A subclass derives from a class which in turn derives from another class

B. A single superclass inherits from multiple subclasses

C. A single subclass derives from a single superclass

D. Multiple base classes inherit a single derived class

Answer: C

The self Parameter (CO2)

- The self parameter is a reference to the current instance of the class and is used to access variables that belongs to the class.
- self represents the instance of the class. By using the “self” keyword we can access the attributes and methods of the class in python. It binds the attributes with the given arguments.

the __init__() Function(CO2)

- We want to add the __init__() function to the child class (instead of the pass keyword).

Example

- Add the __init__() function to the Student class:

```
class Student(Person):
        def __init__(self, fname, lname):
                #add properties etc.
```

- When you add the __init__() function, the child class will no longer inherit the parent's __init__() function.
- **Note:** The child's __init__() function **overrides** the inheritance of the parent's __init__() function.

Example of the __init__() Function(CO2)

- To keep the inheritance of the parent's __init__() function, add a call to the parent's __init__() function:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)

x = Student(“Abhijit", “Niet")
x.printname()
```

Output

```
Nidhi Niet
```

Daily MCQs

1. What will be the output of below Python code?

```
class Student:
    def __init__(self,name,id):
         self.name=name
         self.id=id
          print(self.id)
std=Student("Simon",1)
std.id=2
print(std.id)
```

A. 1 1 B. 1 2 C. 2 1 D. 2 2

Answer: B

2. Which of the following is correct with respect to OOP concept in Python?

A. Objects are real world entities while classes are not real.
B. Classes are real world entities while objects are not real.
C. Both objects and classes are real world entities.
D. Both object and classes are not real.

Answer: A

Constructor in Inheritance(CO2)

By default, The constructor in the parent class is available to the child class.

```
class Father:
      def __init__(self):
        self.money = 2000
        print("Father Class Constructor")

class Son (Father):
      def disp(self):
        print(“Son Class Instance Method:”,self.money)

s = Son( )
s.disp()
```

What will happen if we define constructor in both classes ?

Constructor Overriding(CO2)

- If we write constructor in the both classes, parent class and child class then the parent class constructor is not available to the child class.
- In this case only child class constructor is accessible which means child class constructor is replacing parent class constructor.
- Constructor overriding is used when programmer want to modify the existing behavior of a constructor.

Constructor Overriding Program(CO2)

```
class Father:
    def __init__(self):
        self.money = 2000
        print("Father Class Constructor")
class Son(Father):
    def __init__(self):
        self.money = 5000
        print("Son Class Constructor")
    def disp(self):
        print(self.money)
s = Son()
s.disp()
```

How can we call parent class constructor ?

Constructor with super( ) Method(CO2)

- If we write constructor in the both classes, parent class and child class then the parent class constructor is not available to the child class.
- In this case only child class constructor is accessible which means child class constructor is replacing parent class constructor.
- **super ( )** method is used to call parent class constructor or methods from the child class.

Daily MCQs

1. When a child class inherits from only one parent class, it is called?

A. single inheritance

B. singular inheritance

C. Multiple inheritance

D. Multilevel inheritance

Answer: A

2. The child's __init__() function overrides the inheritance of the parent's __init__() function.

A. TRUE

B. FALSE

C. Can be true or false

D. Can not say

Answer: A

3. __________function that will make the child class inherit all the methods and properties from its parent.

A. self

B. __init__()

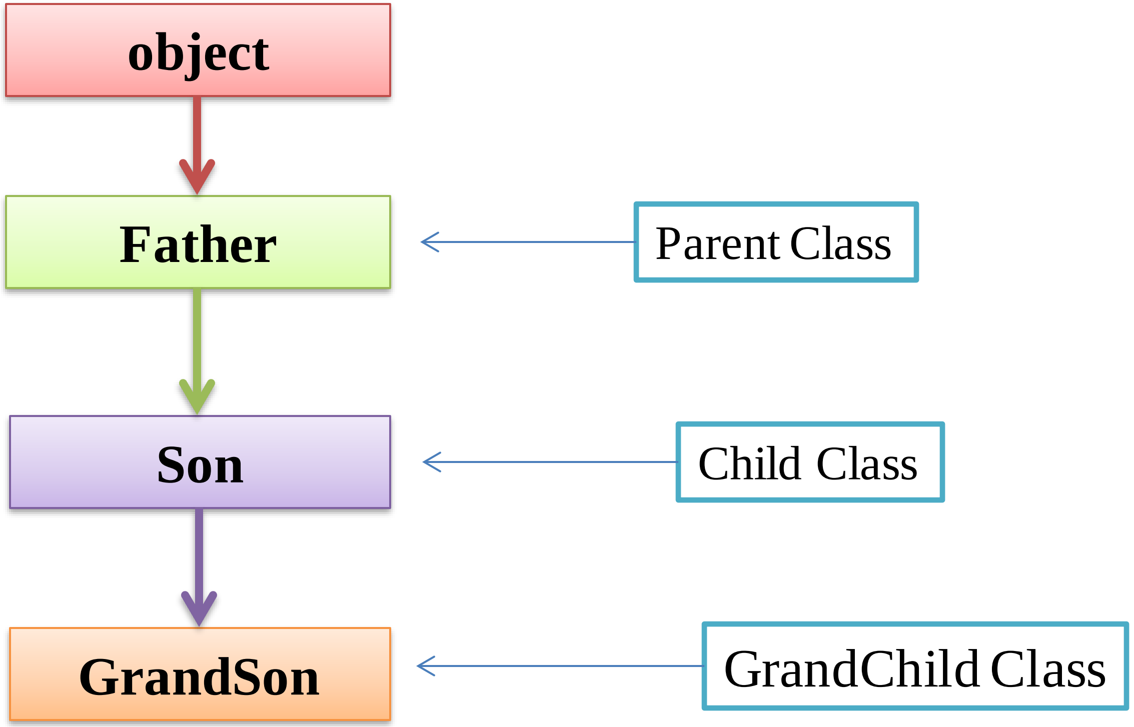
C. super

D. pass

Answer: C

Multi-level Inheritance(CO2)

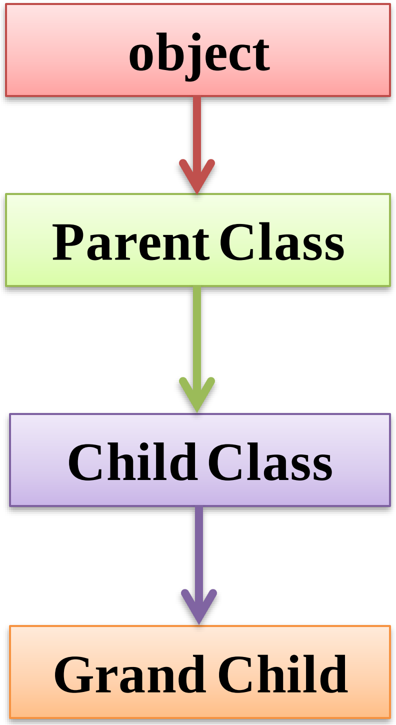
In multi-level inheritance, the class inherits the feature of another derived class (Child Class).



Multi-level Inheritance Syntax(CO2)

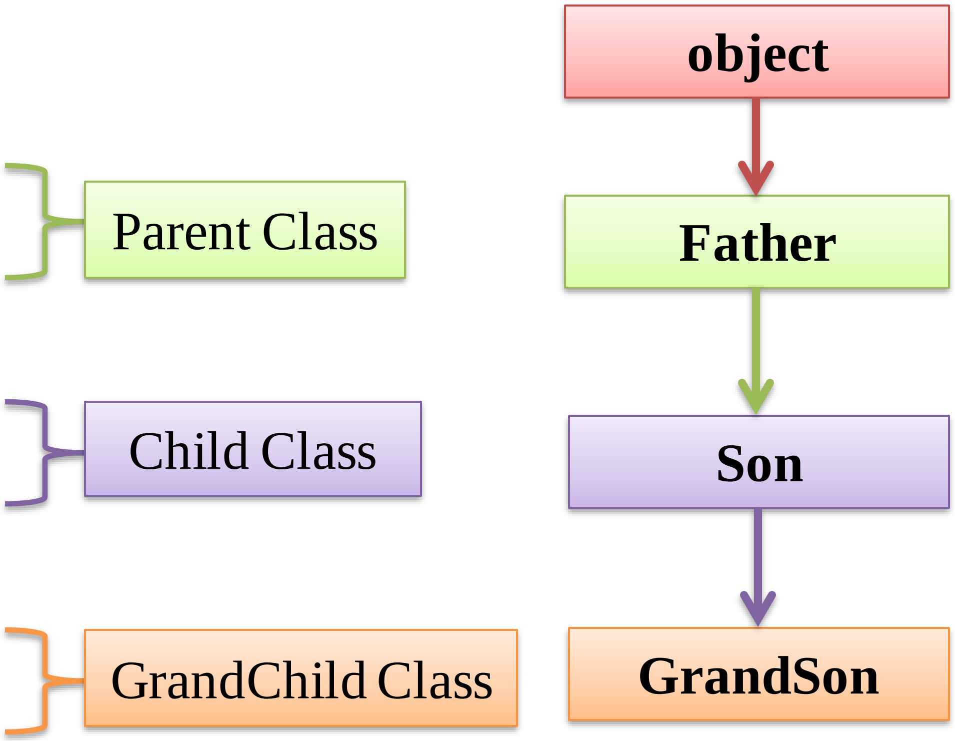
Syntax:-

```
class ParentClassName(object):
        members of Parent Class

class ChildClassName(ParentClassName):
        members of Child Class

class GrandChildClassName(ChildClassName):
        members of Grand Child Class
```

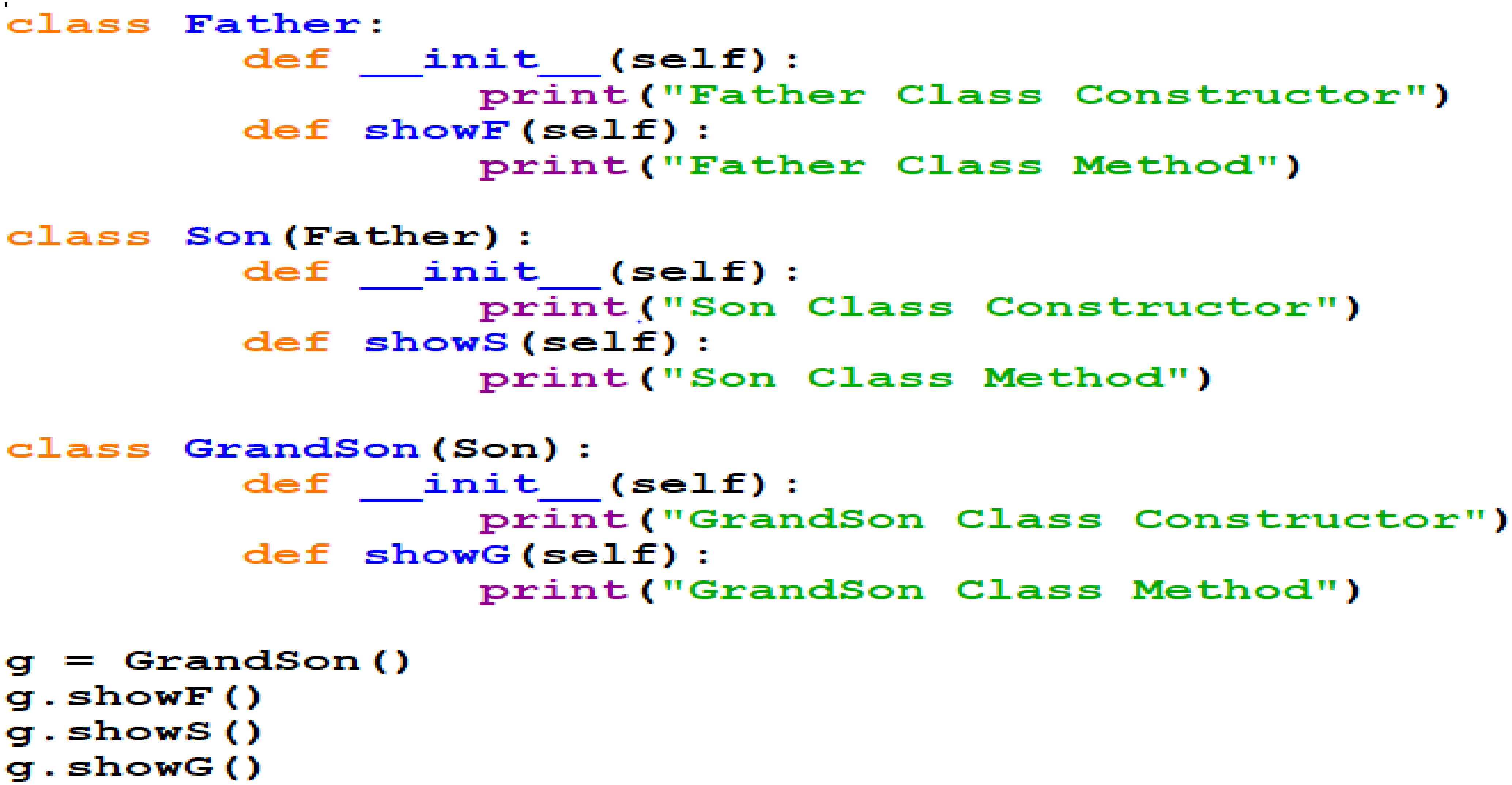


Multi-level Inheritance Example(CO2)

```
class Father (object):
        members of class Father

class Son (Father):
        members of class Son

class GrandSon (Son):
        members of class GrandSon
```



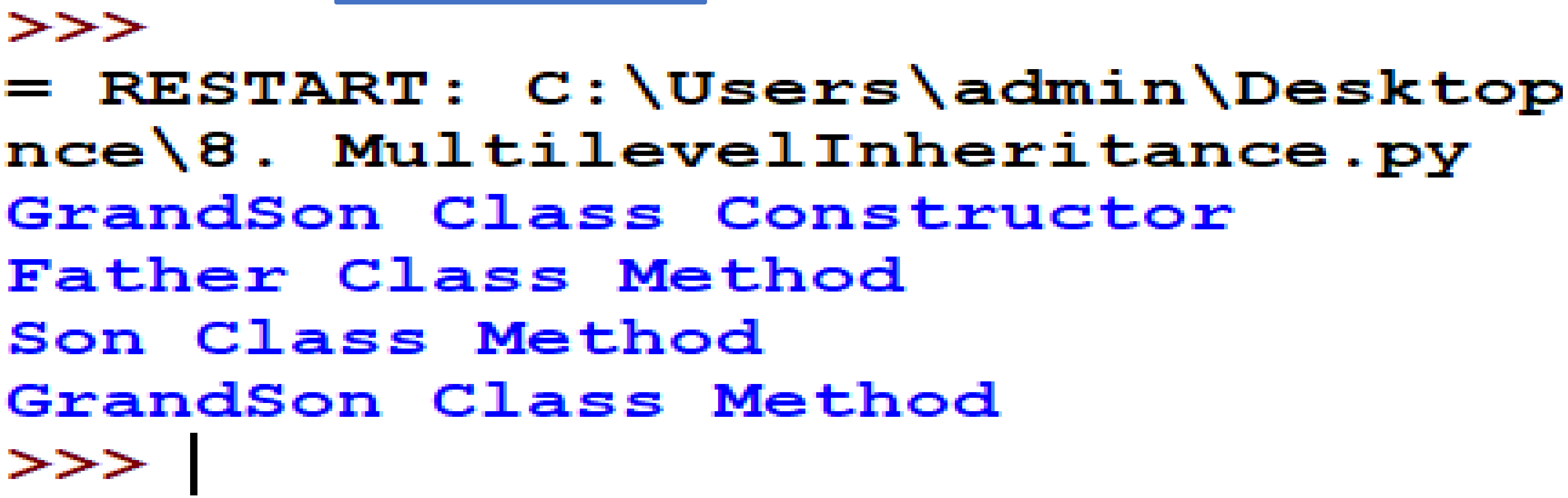
Multi-level Inheritance Program(CO2)



```
class Father:
    def __init__(self):
        print("Father Class Constructor")
    def showF(self):
        print("Father Class Method")

class Son(Father):
    def __init__(self):
        print("Son Class Constructor")
    def showS(self):
        print("Son Class Method")

class GrandSon(Son):
    def __init__(self):
        print("GrandSon Class Constructor")
    def showG(self):
        print("GrandSon Class Method")

g = GrandSon()
g.showF()
g.showS()
g.showG()
```

Multi-level Inheritance Program(CO2)

OUTPUT



```
>>>
= RESTART: C:\Users\admin\Desktop
nce\8. MultilevelInheritance.py
GrandSon Class Constructor
Father Class Method
Son Class Method
GrandSon Class Method
>>> |
```

Daily MCQs

1. When two or more classes serve as base class for a derived class, the situation is known as __________.

A. multiple inheritance

B. polymorphism

C. encapsulation

D. hierarchical inheritance

E. none of these

Answer: A

2. Multiple inheritance leaves room for a derived class to have _______ members.

A. dynamic

B. private

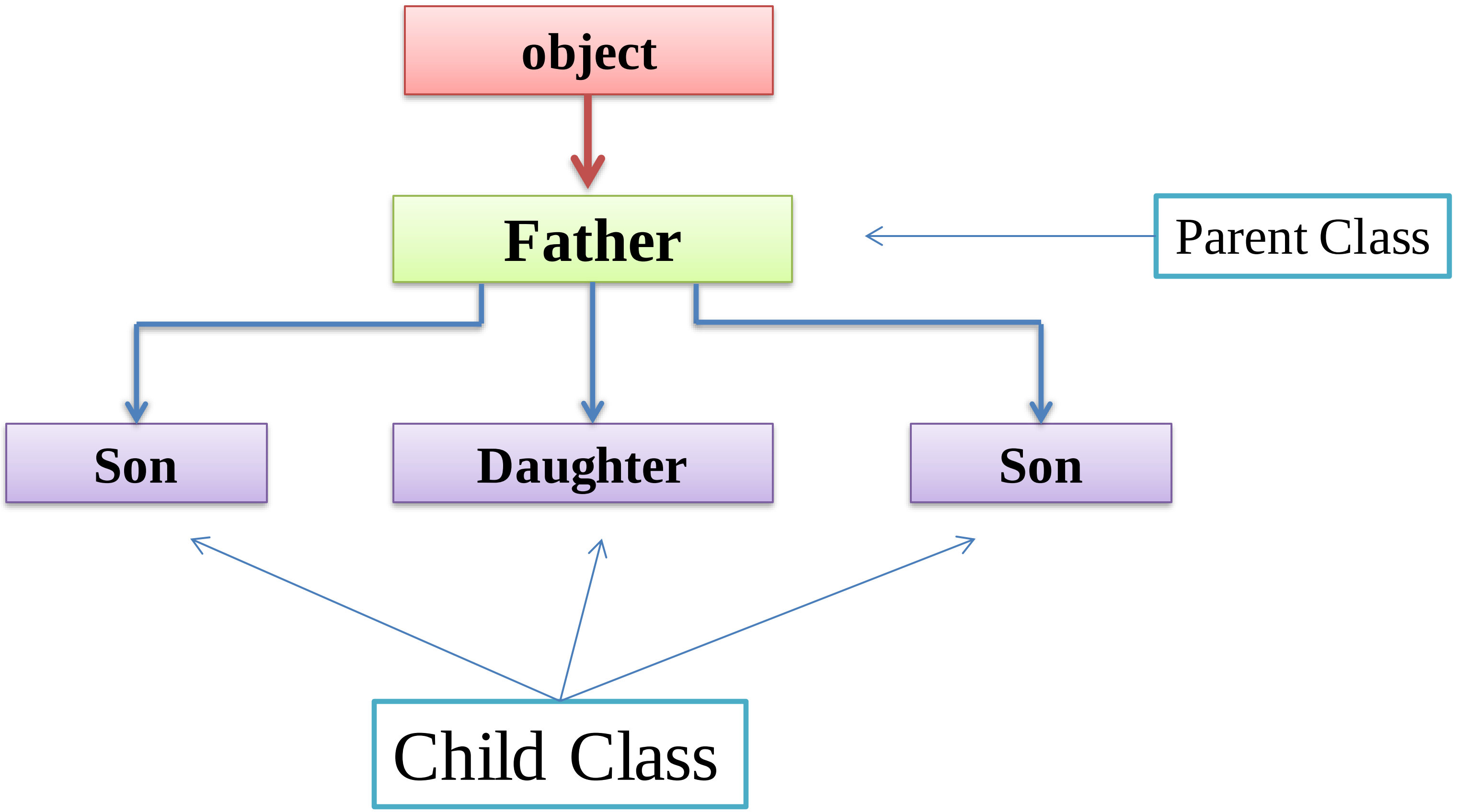
C. public

D. ambiguous

E. none of these

Answer: d

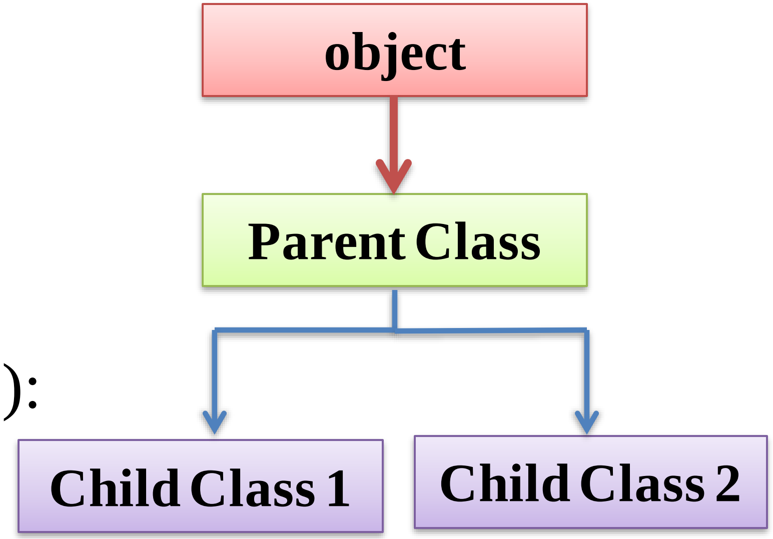
Hierarchical Inheritance(CO2)



Hierarchical Inheritance Syntax(CO2)

Syntax:-

```
class ParentClassName(object):
        members  of Parent Class

class ChildClassName1(ParentClassName):
        members  of Child Class 2

class ChildClassName2(ParentClassName):
        members  of Child Class 2
```



Hierarchical Inheritance(CO2)

class Father (object):

members of class Father

Parent Class

class Son (Father):

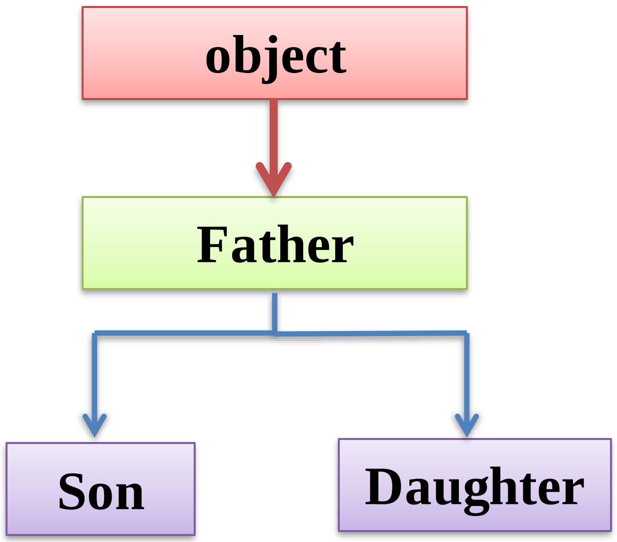
members of class Son

Child Class

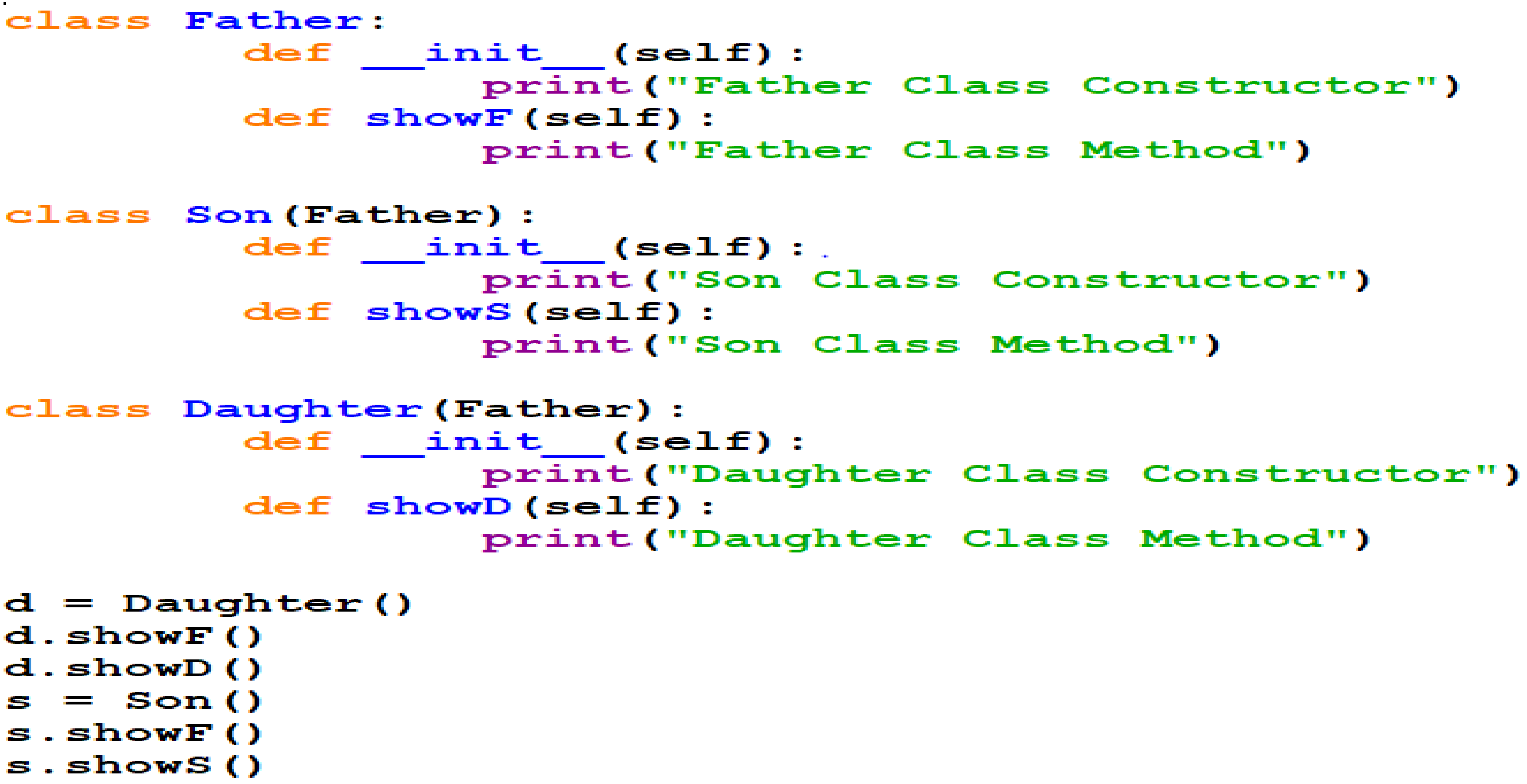
class Daughter (Father):

members of class Daughter

Child Class



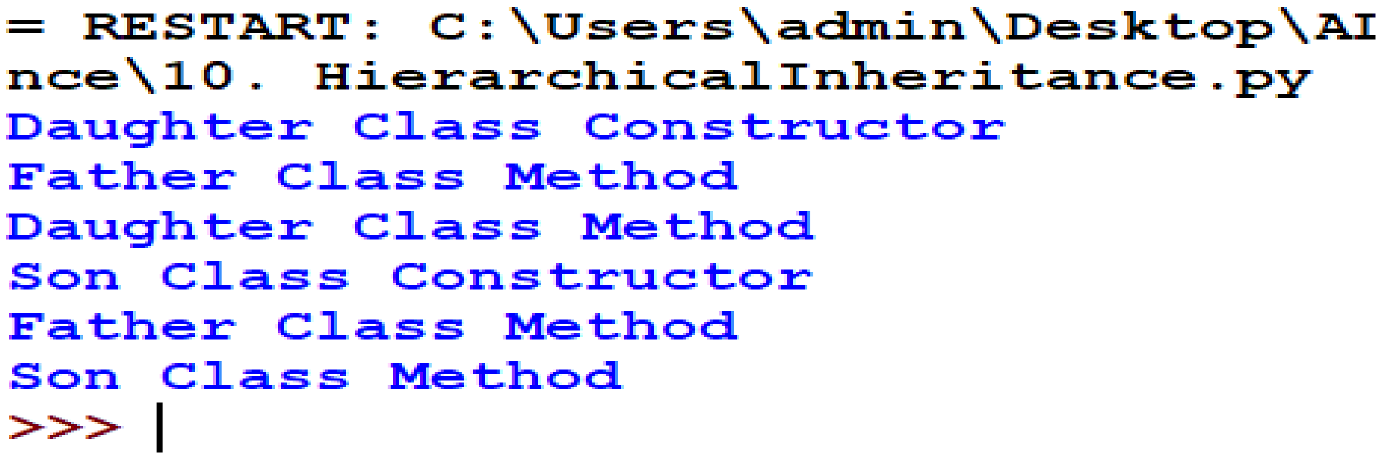
Hierarchical Inheritance Program(CO2)



```
class Father:
        def __init__(self):
                print("Father Class Constructor")
        def showF(self):
                print("Father Class Method")

class Son(Father):
        def __init__(self):
                print("Son Class Constructor")
        def showS(self):
                print("Son Class Method")

class Daughter(Father):
        def __init__(self):
                print("Daughter Class Constructor")
        def showD(self):
                print("Daughter Class Method")

d = Daughter()
d.showF()
d.showD()
s = Son()
s.showF()
s.showS()
```

Hierarchical Inheritance Program(CO2)

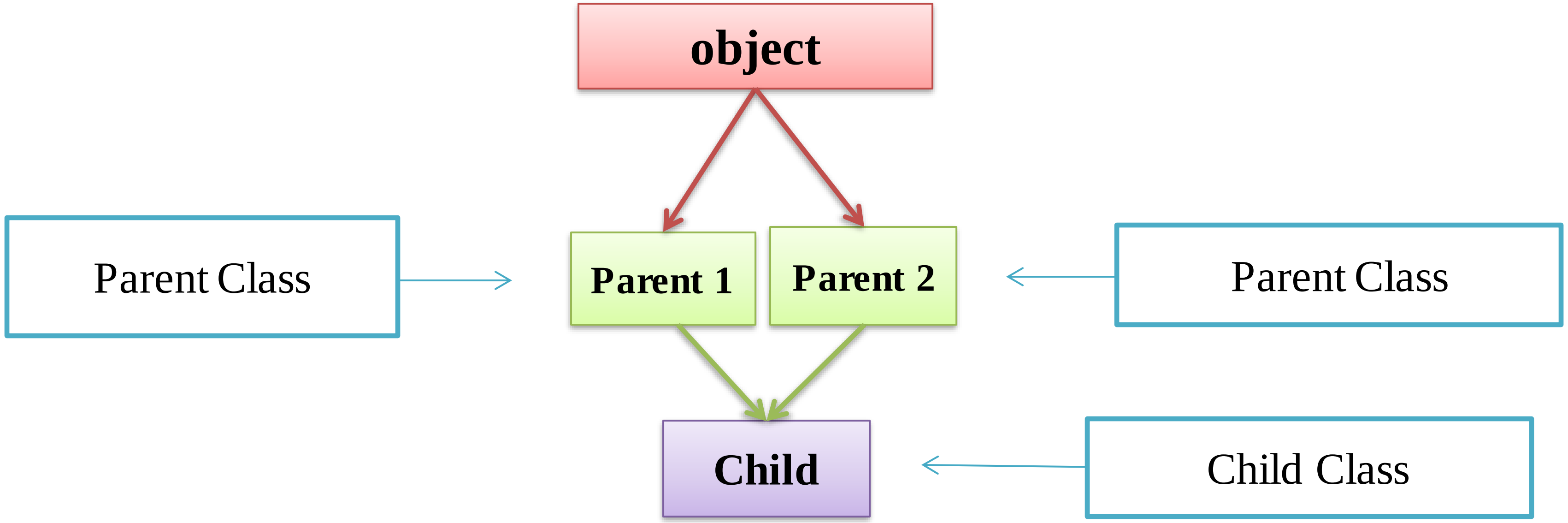
OUTPUT



```
= RESTART: C:\Users\admin\Desktop\AI
nce\10. HierarchicalInheritance.py
Daughter Class Constructor
Father Class Method
Daughter Class Method
Son Class Constructor
Father Class Method
Son Class Method
>>> |
```

Multiple Inheritance(CO2)

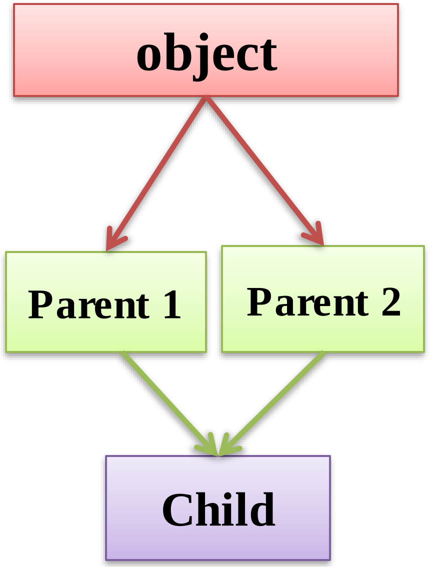
If a class is derived from more than one parent class, then it is called multiple inheritance.



Multiple Inheritance Syntax(CO2)

Syntax:-

```
class ParentClassName1(object):
        members of Parent Class

class ParentClassName2(object):
        members of Parent Class

class ChildClassName(ParentClassName1, ParentClassName2):
        members of Child Class
```



Multiple Inheritance(CO2)

```
class Father (object):
     members of class Father
```

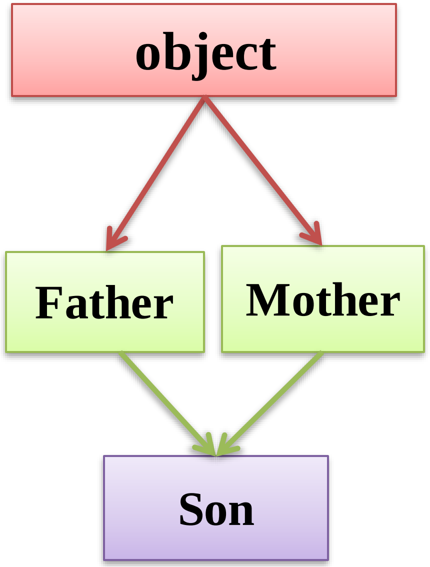
Parent Class

```
class Mother (object):
     members of class Mother
```

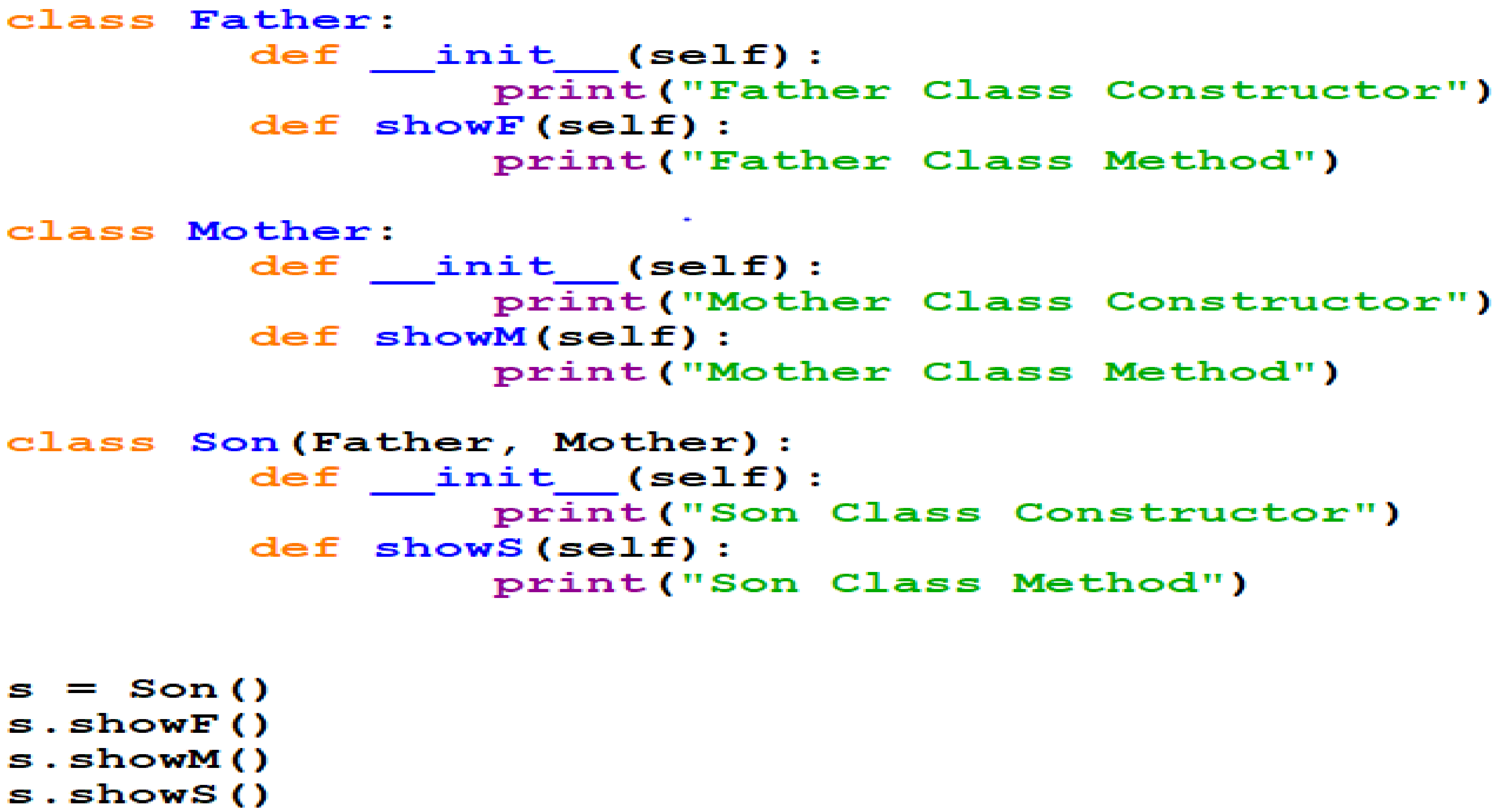
Parent Class

```
class Son (Father, Mother):
    members of class Son
```

Child Class



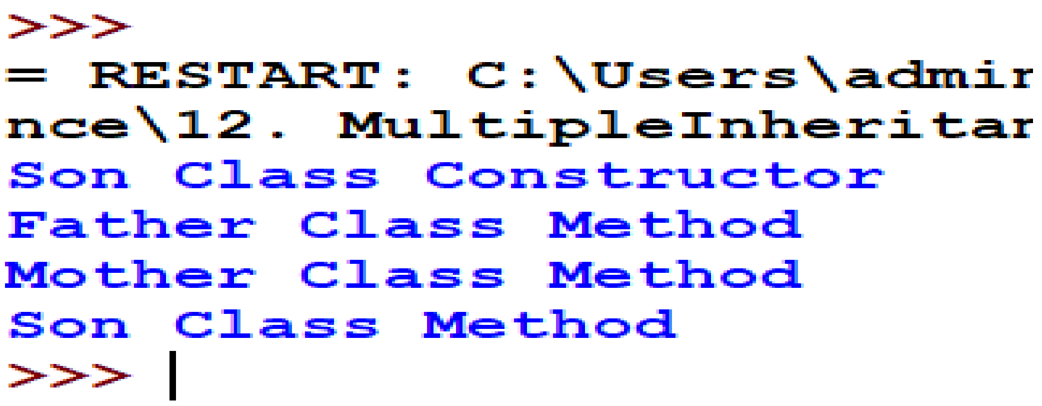
Multiple Inheritance Program(CO2)



```
class Father:
    def __init__(self):
        print("Father Class Constructor")
    def showF(self):
        print("Father Class Method")

class Mother:
    def __init__(self):
        print("Mother Class Constructor")
    def showM(self):
        print("Mother Class Method")

class Son(Father, Mother):
    def __init__(self):
        print("Son Class Constructor")
    def showS(self):
        print("Son Class Method")


s = Son()
s.showF()
s.showM()
s.showS()
```

Multiple Inheritance Program(CO2)

OUTPUT



```
>>>
= RESTART: C:\Users\admir
nce\12. MultipleInheritar
Son Class Constructor
Father Class Method
Mother Class Method
Son Class Method
>>> 
```

Daily Quiz

1.Which of the following best describes inheritance?

a) Ability of a class to derive members of another class as a part of its own definition.

b) Means of bundling instance variables and methods in order to restrict access to certain class members

c) Focuses on variables and passing of variables to functions

d) Allows for implementation of elegant software that is well designed and easily modified.

Answer: a

2. All subclasses are a subtype in object-oriented programming.

a) True

b) False

Answer: b

3. When defining a subclass in Python that is meant to serve as a subtype, the subtype Python keyword is used.

a) True

b) False

Answer: b

Daily Quiz

4. What will be the output of the following Python code?

```
class Test:
    def __init__(self):
      self.x = 0
class Derived_Test(Test):
    def __init__(self):
      self.y = 1
def main():
      b = Derived_Test()
   print(b.x,b.y)
 main()
```

Answer:c

a) 0 1

b) 0 0

c) Error because class B inherits A but variable x isn’t inherited

d) Error because when object is created, argument must be passed like Derived_Test(1)

Daily Quiz

5.**What will be the output of the following Python code?**

```
 class A():
        def disp(self):
print("A disp()")
            class B(A):
        pass
obj = B()
obj.disp()
```

Answer: d

a) Invalid syntax for inheritance

b) Error because when object is created, argument must be passed

c) Nothing is printed

d) A disp()

Weekly Assignment

Q1. A) Create child class Bus that will inherit all of the variables and methods of the Vehicle class

Given:

```
class Vehicle:


  def __init__(self, name, max_speed, mileage):
    self.name = name
    self.max_speed = max_speed
    self.mileage = mileage
```

B) Create a Bus object that will inherit all of the variables and methods of the Vehicle class and display it.

Q2. How to check if a class is subclass of another?

Q3. How to access parent members in a subclass?

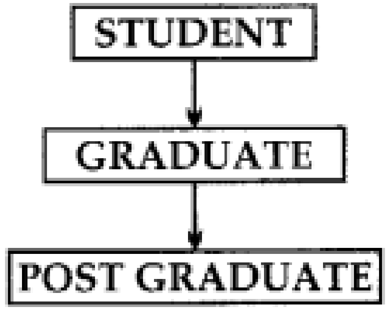
Weekly Assignment

Q4. What will be the output of the following Python code?

```
class A:
def __init__(self):
self.__i = 1
self.j = 5
def display(self):
print(self.__i, self.j)
class B(A):
def __init__(self):
super().__init__()
self.__i = 2
self.j = 7
c = B()c.display()
```

Weekly Assignment

Q5. Consider the figure given below and answer the questions that follows:



- Name the base class and the derived class.
- Which concept of OOP is implemented in the figure given above?

Old Question Papers

Q1. Predict the output of the following program. Also state which concept of OOP is being implemented?

```
def sum(x,y,z):
print “sum= ”, x+y+z
def sum(a,b):
print “sum= ”, a+b
sum(10,20)
sum(10,20,30)
```

Q2. Write a program that uses an area() function for the calculation of area of a triangle or a rectangle or a square. Number of sides (3, 2 or 1) suggest the shape for which the area is to be calculated.

Old Question Papers

Q3. Give a suitable example using Python code to illustrate single level inheritance considering COUNTRY to be BASE class and STATE to be derived class.

Q4. What is the difference between Multilevel inheritance and multiple inheritance? Give suitable examples to illustrate.

Q5. What are the different ways of overriding function call in derived class of python? Illustrate with example.

Expected Questions for University Exam

Q1.What is the need of object-oriented systems? Explain with the help of classes and objects.

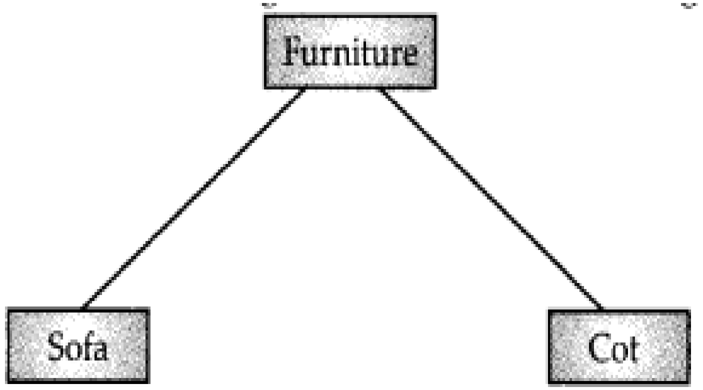
Q2. What is Inheritance? How code reusability is achieved using inheritance? Explain with the help of a program.

Q3. Write short note on hybrid inheritance.

Q4. What is the difference between Multilevel inheritance and multiple inheritance? Give suitable examples to illustrate.

Expected Questions for University Exam

Q5. Based on the diagram, answer the following:



- Write the name of the base class and the derived classes.
- Write the type of inheritance depicted in the above diagram.

SUMMARY

Inheritance allows us to define a class that **inherits** all the methods and properties from another class. Parent class is the class being **inherited** from, also called base class.

Class, methods and polymorphism

CONTENTS

- Class Method
- Concrete method
- Abstract Class
- Instance Method
- MRO & Super()
- Polymorphism
- Method Overriding

Prerequisite and Recap(CO2)

- Operators
- Loop
- Method

- Variables
- Class , Objects
- Constructor

Topic Objectives(CO2)

After you have read and studied this topic, you should be able to

- Understand the class types and method.
- Understand how Polymorphism is done.
- Understand how method overriding is done.

Type of Methods(CO2)

- Instance Methods
  - Accessor Methods
  - Mutator Methods

- **Class Methods**

- Static Methods Unit-1 Read already

Instance Method(CO2)

- Instance methods are the methods which act upon the instance variables of the class.
- Instance method need to know the memory address of the instance which is provided through *self* variable by default as first parameter for the instance method.

Syntax:-

```
def method_name(self):
        function body
```

Instance Method without Parameter/Formal Arguments

```
def method_name(self, f1, f2):
        function body
```

Instance Method with Parameter/Formal Arguments

Calling Instance Method w/o Argument(CO2)

Instance methods are bound to object of the class so we call instance method with object name.

Syntax:- object_name.method_name()

Ex:- realme.show_model()

```
class Mobile:
    def show_model(self):
            print(“RealMe X”)
realme = Mobile( )
realme.show_model()
```

Calling Instance Method w/o Argument

Calling Instance Method with Argument(CO2)

Syntax:- object_name.method_name(Actual_argument)

Ex:- realme.show_model(1000)

```
class Mobile:
     def __init__(self):
            self.model = 'RealMe X'
    def show_model(self, p):
            self.price = p
            print(self.model, self.price)

realme = Mobile( )
realme.show_model(1000)
```

Calling Method with argument

Accessor Method(CO2)

This method is used to access or read data of the variables. This method do not modify the data in the variable. This is also called as getter method.

Ex:-

```
def get_value(self):
def get_result(self):
def get_name(self):
def get_id(self):
```

```
class Mobile:
    def __init__(self):
        self.model = ‘RealMe X’
    def get_model(self):
        return self.model

realme = Mobile( )
m = realme.get_model()
print(m)
```

Mutator Method(CO2)

This method is used to access or read and modify data of the variables. This method modify the data in the variable. This is also called as setter method.

Ex:-

```
def set_value(self):
def set_result(self):
def set_name(self):
def set_id(self):
```

```
class Mobile:
   def __init__(self):
        self.model =
‘RealMe X’

   def set_model(self):
        self.model =
‘RealMe 2’
realme = Mobile( )
realme.set_model()
```

```
class Mobile:

   def set_model(self,
m):
        self.model = m

realme = Mobile( )
realme.set_model(‘Real
Me X’)
```

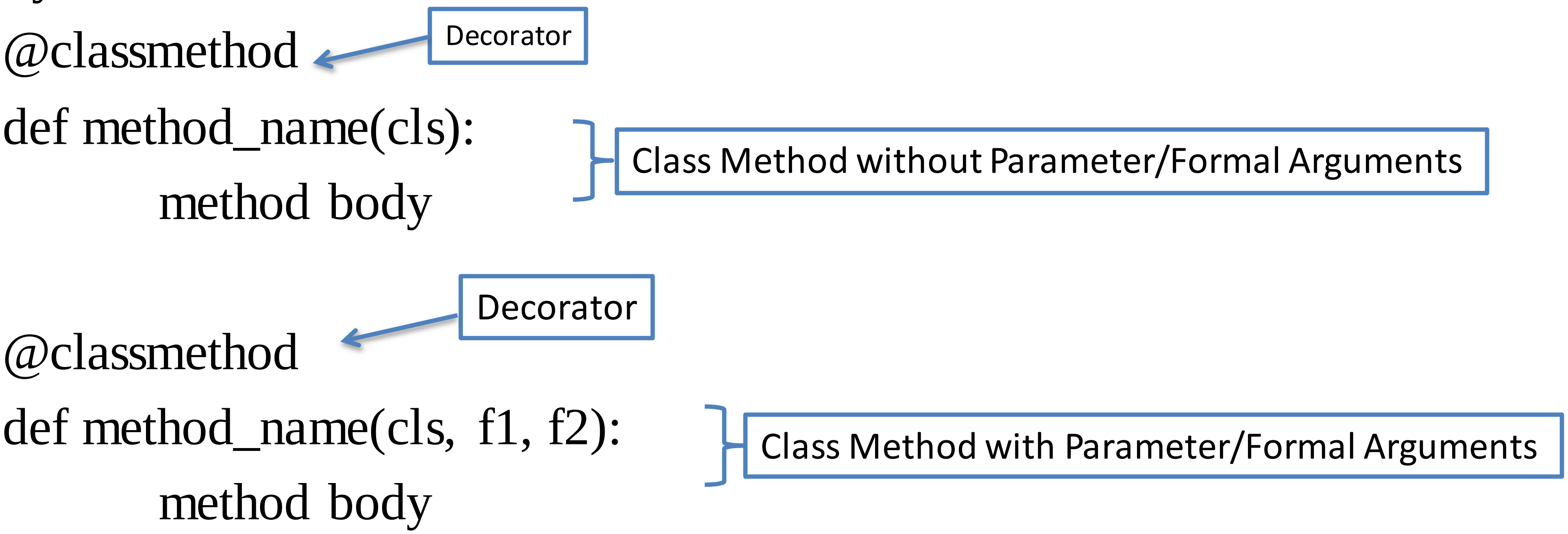
Class Methods(CO2)

Class methods are the methods which act upon the class variables or static variable of the class.

Decorator @classmethod need to write above the class method.

By default, the first parameter of class method is cls which refers to the class itself.

Syntax:-



```
@classmethod
def method_name(cls):
        method body

@classmethod
def method_name(cls, f1, f2):
        method body
```

Decorator
Class Method without Parameter/Formal Arguments
Decorator
Class Method with Parameter/Formal Arguments

Class Method without Parameter(CO2)

```
class Mobile:
    @classmethod
    def show_model(cls):
        print(“RealMe X”)

realme = Mobile( )
```

Decorator

Class Method

```
class Mobile:
    fp = ‘Yes’
    @classmethod
    def show_model(cls):
        print(cls.fp)

realme = Mobile( )
```

Class Variable

Decorator

Class Method

Accessing Class variable
Inside Class Method

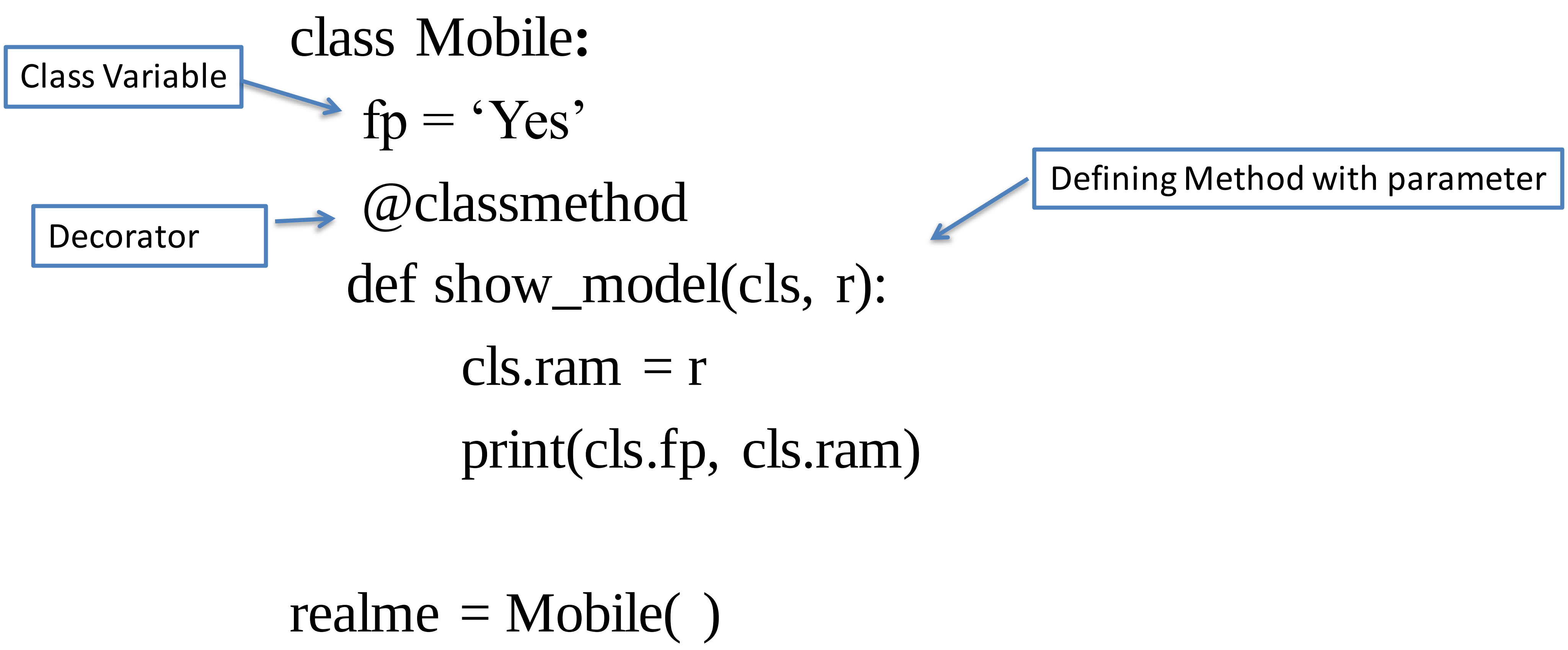
Calling Class Method without Argument(CO2)

Syntax:- Classname.method_name()

```
class  Mobile:
    @classmethod
    def show_model(cls):
            print(“RealMe  X”)

realme  =  Mobile(  )
Mobile.show_model()
```

Calling Class Method w/o Argument

Class Method with Parameter(CO2)



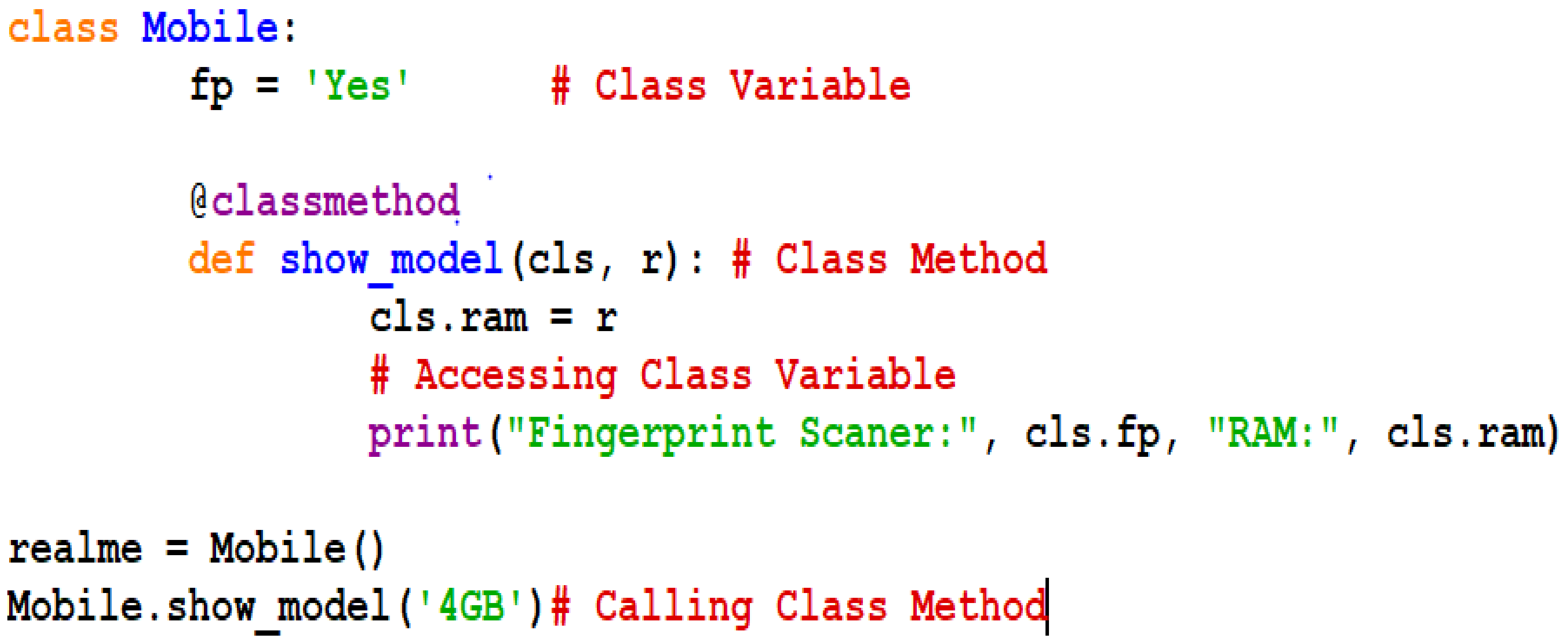
Class Variable

Decorator

Defining Method with parameter

```
class Mobile:
  fp = 'Yes'
  @classmethod
  def show_model(cls, r):
        cls.ram = r
        print(cls.fp, cls.ram)

realme = Mobile( )
```

Class Method with Parameter Program(CO2)



```python
class Mobile:
        fp = 'Yes'      # Class Variable

        @classmethod
        def show_model(cls, r): # Class Method
                cls.ram = r
                # Accessing Class Variable
                print("Fingerprint Scaner:", cls.fp, "RAM:", cls.ram)

realme = Mobile()
Mobile.show_model('4GB')# Calling Class Method
```

Calling Class Method with Argument(CO2)

Syntax:- Classname.method_name(Actual_argument)

Ex:- Mobile.show_model(‘4GB’)

```
class  Mobile:
      fp = ‘Yes’
      @classmethod
      def show_model(cls,  r):
              cls.ram  = r
              print(cls.fp,  cls.ram)
realme  = Mobile(  )
Mobile.show_model(101)
```

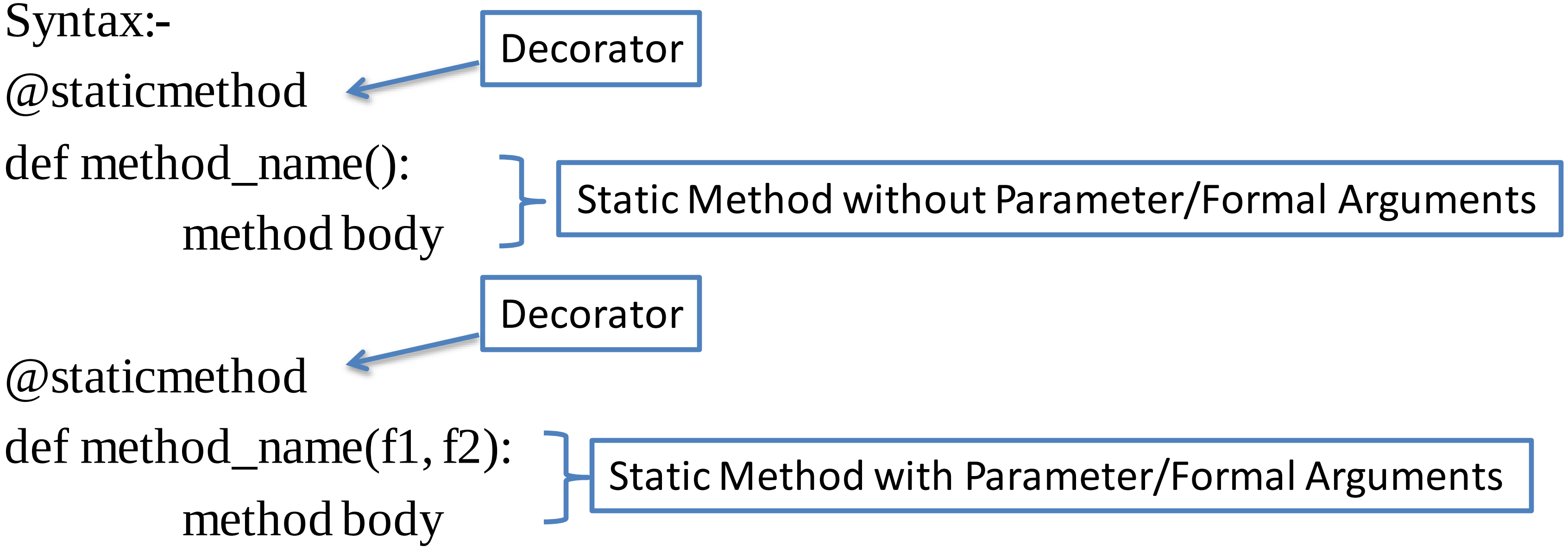
Calling Method with argument

Static Methods(CO2)

- Static Methods are used when some processing is related to the class but does not need the class or its instances to perform any work.
- We use static method when we want to pass some values from outside and perform some action in the method.

Decorator @staticmethod need to write above the static method.

Syntax:-



```
@staticmethod
def method_name():
        method body

@staticmethod
def method_name(f1, f2):
        method body
```

Decorator
Static Method without Parameter/Formal Arguments
Decorator
Static Method with Parameter/Formal Arguments

Static Method without Parameter(CO2)

```
class  Mobile:
    @staticmethod
    def show_model():
            print(“RealMe  X”)

realme  = Mobile(  )
```

Decorator

Static Method

```
class  Mobile:
     fp = ‘Yes’
    @staticmethod
    def show_model():
            print(Mobile.fp)

realme  = Mobile(  )
```

Static Method

Calling Static Method without Argument(CO2)

Syntax:- Classname.method_name()

```
class Mobile:
    @staticmethod
    def show_model():
            print(“RealMe  X”)

realme = Mobile( )
Mobile.show_model()
```

Calling Static Method w/o Argument

Static Method with Parameter(CO2)

Decorator

Defining Method with parameter

```
class Mobile:
    @staticmethod
    def show_model(m, p):
        model = m
        price = p
        print(model, price)
realme = Mobile( )
```

Calling Static Method with Argument(CO2)

Syntax:- Classname.method_name(Actual_argument)

Ex:- Mobile.show_model(1000)

```
class Mobile:
    @staticmethod
    def show_model(m, p):
        model = m
        price = p
        print(model, price)
realme = Mobile( )
Mobile.show_model(‘RealMe X’, 1000)
```

Calling Method with argument

Daily MCQs

1. To create a class, use the keyword?

A. new

B. except

C. class

D. Object

Answer: C

2. ___is used to create an object.

A. class

B. constructor

C. user-defined functions

D. In-built functions

Answer: B

3. **_____ represents an entity in the real world with its identity and behavior.**

A. A method

B. An object

C. A class

D. An operator

Answer:B

Daily MCQs

4. Special methods need to be explicitly called during object creation.

A. TRUE

B. FALSE

Answer: B

5. Overriding means changing behaviour of methods of derived class methods in the base class.

A. TRUE

B. FALSE

Answer: B

6. Which of the following is not a class method?

A. Non-static

B. Static

C. Bounded

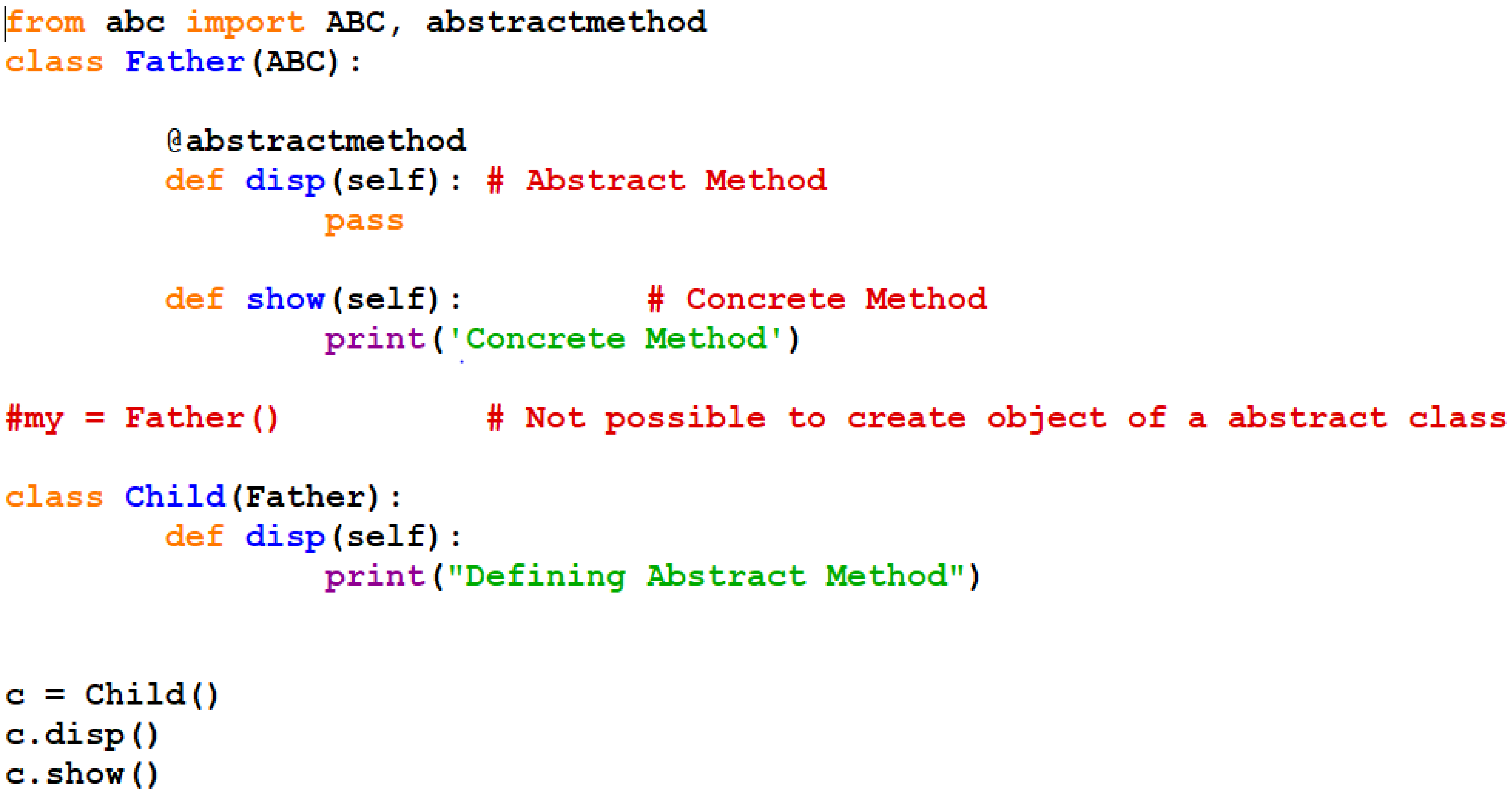
D. Unbounded

Answer:A

Abstract Class(CO2)

1. A class derived from ABC class which belongs to abc module, is known as abstract class in Python.
2. ABC Class is known as Meta Class which means a class that defines the behavior of other classes. So we can say, Meta Class ABC defines that the class which is derived from it becomes an abstract class.
3. Abstract Class can have abstract method and concrete methods.
4. Abstract Class needs to be extended and its method implemented.
5. PVM can not create objects of an abstract class.

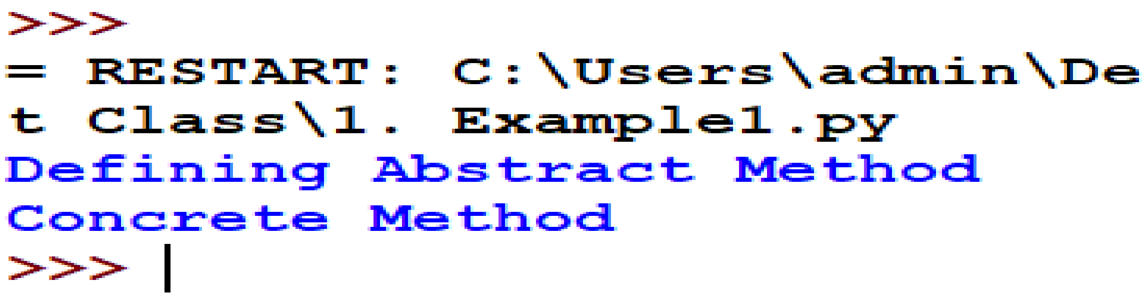
Abstract Class Program(CO2)



```
from abc import ABC, abstractmethod
class Father(ABC):

        @abstractmethod
        def disp(self): # Abstract Method
                pass

        def show(self):         # Concrete Method
                print('Concrete Method')

#my = Father()          # Not possible to create object of a abstract class

class Child(Father):
        def disp(self):
                print("Defining Abstract Method")


c = Child()
c.disp()
c.show()
```

Abstract Class Program(CO2)

OUTPUT



```
>>>
= RESTART: C:\Users\admin\De
t Class\1. Example1.py
Defining Abstract Method
Concrete Method
>>> 
```

Abstract Method(CO2)

An abstract method is a method whose action is redefined in the child classes as per the requirement of the object.

We can declare a method as abstract method by using @abstractmethod decorator.

Ex:-

```
from abc import ABC, abstractmethod
Class Father(ABC):
      @abstractmethod
       def disp(self):
              pass
```

Concrete Method(CO2)

A Concrete method is a method whose action is defined in the abstract class itself.

Ex:-

```
from abc import ABC, abstractmethod
Class Father(ABC):
      @abstractmethod
        def disp(self):
                pass
        def show(self):
                print(“Concrete Method”)
```

Abstract Method / Method Without Body

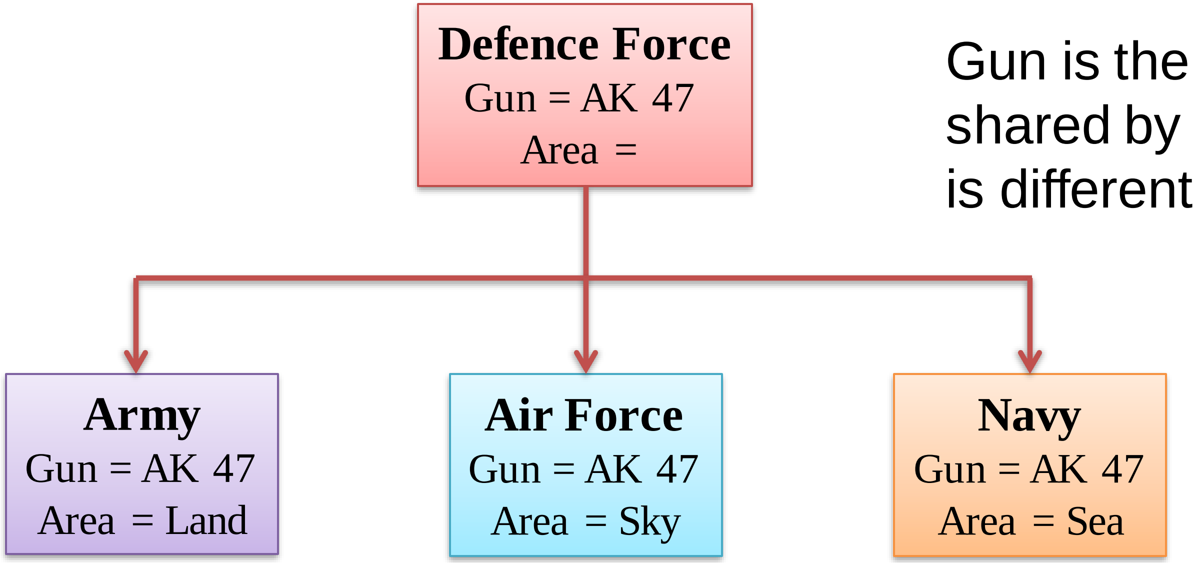
Concrete Method / Method with Body

Rules of Abstract Class (CO2)

- PVM cannot create objects of an abstract class.
- It is not necessary to declare all methods abstract in an abstract class.
- Abstract Class can have abstract method and concrete methods.
- If there is any abstract method in a class, that class must be abstract.
- The abstract methods of an abstract class must be defined in its child class/subclass.
- If you are inheriting any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

When use Abstract Class(CO2)

We use abstract class when there are some common feature shared by all the objects as they are.



Gun is the common feature shared by all Forces but area is different for them.

Daily MCQs

1. What is an abstract class?

A. An abstract class is one without any child classes.

B. An abstract class is any parent class with more than one child class.

C. An abstract class is class which cannot be instantiated, but can be a base class.

D. abstract class is another name for "base class."

Answer: C

2. Can an abstract parent class have non-abstract children?

A. No—an abstract parent must have only abstract children.

B. No—an abstract parent must have no children at all.

C. Yes—all children of an abstract parent must be non-abstract.

D. Yes—an abstract parent can have both abstract and non-abstract children.

Answer: D

Method Resolution Order (MRO)(CO2)

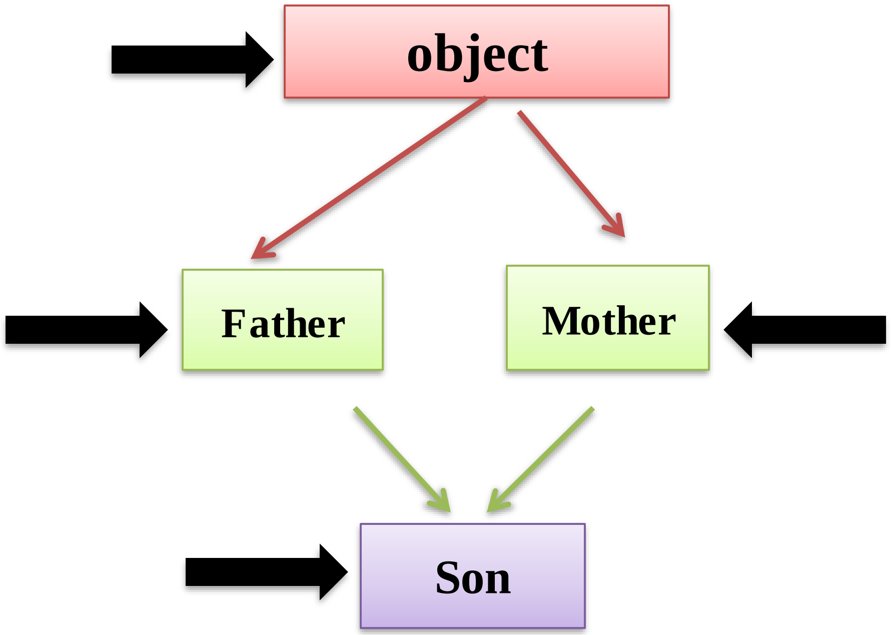
In the multiple inheritance scenario members of class are searched first in the current class. If not found, the search continues into parent classes in depth-first, left to right manner without searching the same class twice.

- Search for the child class before going to its parent class.
- When a class is inherited from several classes, it searches in the order from left to right in the parent classes.
- It will not visit any class more than once which means a class in the inheritance hierarchy is traversed only once exactly.

Method Resolution Order (MRO)(CO2)

s = Son()

- The search will start from Son. As the object of Son is created, the constructor of Son is called.
- Son has super().__init__() inside his constructor so its parent class, the one in the left side ‘Father’ class’s constructor is called.
- Father class also has super().__init__() inside his constructor so its parent ‘object’ class’s constructor is called.
- Object does not have any constructor so the search will continue down to right hand side class (Mother) of object class so Mother class’s constructor is called.
- As Mother class also has super().__inti__() so its parent class ‘object’ constructor is called but as object class already visited, the search will stop here.



Use the super() Function(CO2)

Python also has a super() function that will make the child class inherit all the methods and properties from its parent.

Example

```
class Student(Person):
    def __init__(self, fname, lname):
        super().__init__(fname, lname)
```

By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

Use the super() Function(CO2)

1. __________function that will make the child class inherit all the methods and properties from its parent.

A. self

B. __init__()

C. super

D. pass

Answer: C

2. What will be output for the folllowing code?

```
class A:
    def __init__(self, x= 1):
        self.x = x
class der(A):
    def __init__(self,y = 2):
        super().__init__()
        self.y = y
def main():
    obj = der()
    print(obj.x, obj.y)
main()
```

A. Error, the syntax of the invoking method is wrong

B. The program runs fine but nothing is printed

C. 1 0

D. 1 2

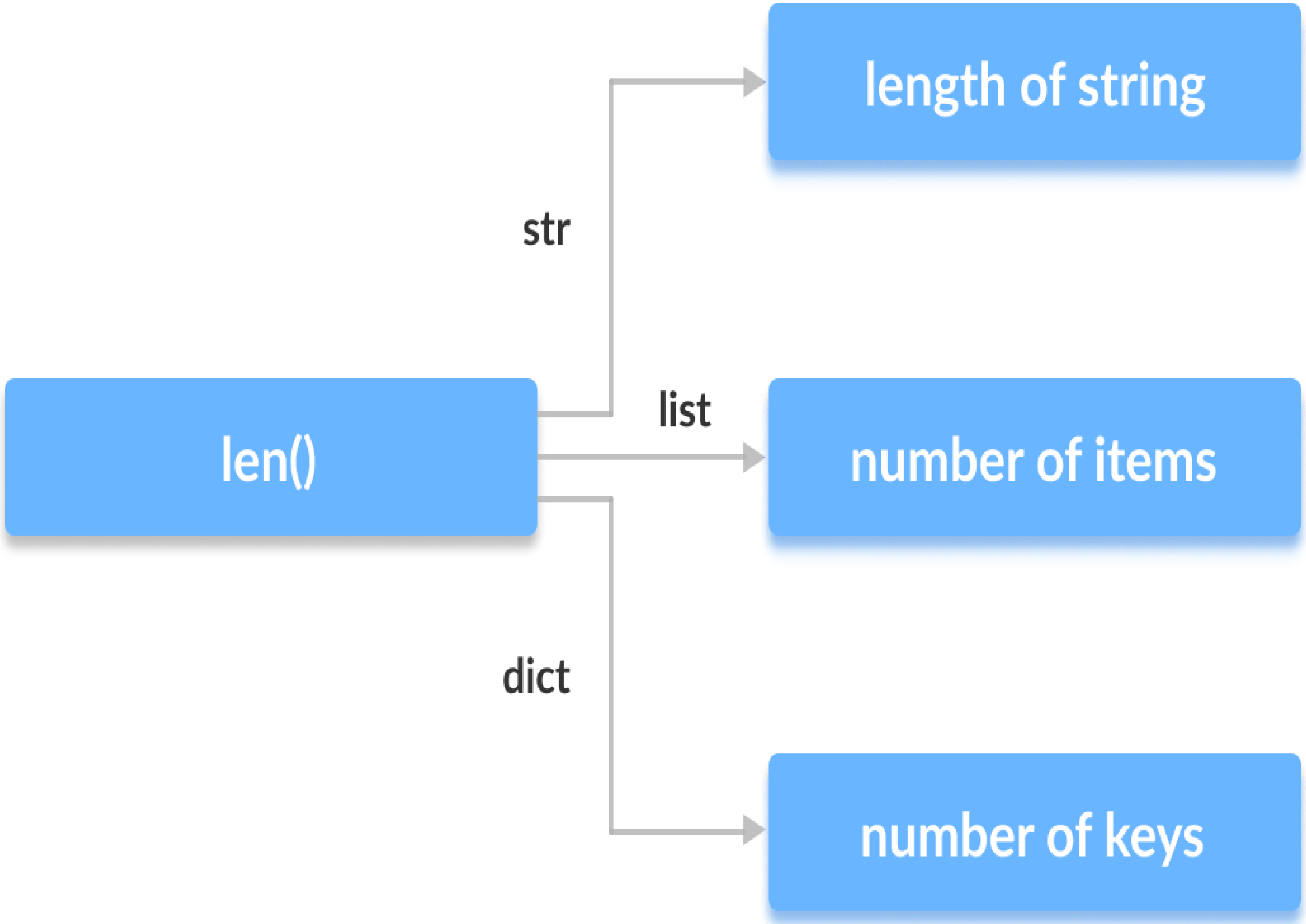
Answer: D

Polymorphism(CO2)

- Polymorphism is a word that came from two greek words, poly means many and morphos means forms.
- If a variable, object or method perform different behavior according to situation, it is called **polymorphism**.

- **Method Overriding**
- Method Overloading
- Operator Overloading

Polymorphic len() function(CO2)



Method overriding (CO2)

- The method overriding in Python means **creating two methods with the same name but differ in the programming logic.**
- The concept of Method overriding allows us to change or override the Parent Class function in the Child Class.
- You can’t override a method within the same class. It means you have to do it in the child class using the **Inheritance** concept.
- To override the Parent **Class** method, you have to create a method in the Child class with the same name and the same number of parameters.

Method overriding Syntax(CO2)

```
# Python Method Overriding
class Employee:
    def message(self):
        print('This message is from Employee Class')
class Department(Employee):
    def message(self):
        print('This Department class is inherited from Employee')
emp = Employee()
emp.message()
print('------------')
dept = Department()
dept.message()
```

Method overriding(CO2)

Output:

```
This message is from Employee Class
------------
This Department class is inherited from Employee
```

Method Overriding with arguments(CO2)

```
class Employee:
      def add(self, a, b):
            print('The Sum of Two = ', a + b)
class Department(Employee):
     def add(self, a, b, c):
           print('The Sum of Three = ', a + b + c)
emp = Employee()
emp.add(10, 20)
print('------------')
dept = Department()
dept.add(50, 130, 90)
```

Method Overriding with arguments(CO2)

Output:

```
The Sum of Two = 30
 ------------
'The Sum of Three = 270
```

Method Overriding(CO2)

```
class Add:
      def result(self, a, b):
        print(“Addition:”, a+b)

class Multi(Add):
    def result(self, a, b):
        print(“Multiplication:”, a*b)

m = Multi()
m.result(10, 20)
```

Method with super( ) Method(CO2)

- If we write method in the both classes, parent class and child class then the parent class’s method is not available to the child class.
- In this case only child class’s method is accessible which means child class’s method is replacing parent class’s method.

- **super ( )** method is used to call parent class’s constructor or methods from the child class.

  Syntax:- super().methodName()

Weekly Assignment

Q1.Write short notes on:

1) MRO

2) Super()

3) Abstract Class

Q2.What are the different ways of overriding function call in derived class of python? Illustrate with example.

Q3.Why use Abstract Base Classes?

Q4.How Abstract Base classes work?

Weekly Assignment

Q5. What is the output of following program:

```
class A:
    def rk(self):
        print(" In class A")
class B(A):
    def rk(self):
        print(" In class B")

r = B()
r.rk()
```

Daily MCQs

1.**Which of the following represents a template, blueprint, or contract that defines objects of the same type?**

a. A class

b. An object

c. A method

d. A data field

Answer: a

2.**Which of the following represents a distinctly identifiable entity in the real world?**

a. A class

b. An object

c. A method

d. A data field

Answer: b

Daily MCQs

1. Overriding means changing behaviour of methods of derived class methods in the base class.

A. True

B. False

Answer: B

2. What will be the output of the following Python code?

```
class A:
    def __repr__(self):
        return "1"
class B(A):
    def __repr__(self):
        return "2"
class C(B):
    def __repr__(self):
        return "3"
o1 = A()
o2 = B()
o3 = C()
print(obj1, obj2, obj3)
```

A. 1 1 1

B. 1 2 3

C. ‘1’ ‘1’ ‘1’

D. An exception is thrown

Daily Quiz

1.**Which of the following best describes polymorphism?**

a) Ability of a class to derive members of another class as a part of its own definition

b) Means of bundling instance variables and methods in order to restrict access to certain class members

c) Focuses on variables and passing of variables to functions

d) Allows for objects of different types and behaviour to be treated as the same general type

Answer: d

2.**A class in which one or more methods are only implemented to raise an exception is called an abstract class.**

a) True

b) False

Answer: a

3.**Overriding means changing behaviour of methods of derived class methods in the base class.**

a) True

b) False

Answer: b

Daily Quiz

4.**What will be the output of the following Python code?**

```
class A:
    def __repr__(self):
        return "1"
class B(A):
    def __repr__(self):
        return "2”
class C(B):
    def __repr__(self):
        return "3"
o1 = A()
o2 = B()
o3 = C()
print(obj1, obj2, obj3)
```

a) 1 1 1
b) 1 2 3
c) ‘1’ ‘1’ ‘1’
d) An exception is thrown

Answer: b

Old Question Papers

Q1. What is the output of following python code?

```
class Parent():
    def __init__(self):
        self.value = "Inside Parent"
    def show(self):
        print(self.value)
class Child(Parent):
    def __init__(self):
        self.value = "Inside Child"
    def show(self):
        print(self.value)
obj1 = Parent()
obj2 = Child()
obj1.show()
obj2.show()
```

Old Questions Papers

Q2. What should be the output of calling the parent's class method inside the overridden method?

```
  class Parent():
     def show(self):
        print("Inside Parent")
  class Child(Parent):
      def show(self):
        Parent.show(self)
        print("Inside Child")
obj = Child()
obj.show()
```

Old Question Papers

Q3.What is super method? Explain with example.

Q4. Explain method overriding with suitable example.

Q5.What is method resolution operator? Support your answer with suitable example.

Expected Questions for University Exam

Q1. Write short note on- (a) Method overriding

(b) Polymorphism

Q2. Explain method overriding with suitable example.

Q3. What is method resolution operator? Support your answer with suitable example.

Q4. Explain the concept of polymorphism by giving suitable examples.

Q5. Explain calling the parent's class method inside the overridden method .

SUMMARY

Polymorphism lets us define methods in the child class that have the same name as the methods in the parent class.

Introspection

CONTENTS

- Introspection: Introspection types , Introspection objects
- Introspection scopes, Inspect modules, Introspect tools

Prerequisite and Recap(CO2)

- Loop
- Method
- Variables
- Class , Objects
- Constructor

Topic Objectives(CO2)

After you have read and studied this topic, you should be able to:

- Understand Introspection using python.
- Understand Introspection tools using python.

Introspection(CO2)

- **Introspection** is an ability to determine the type of an object at runtime.
- Everything in **python** is an object.
- Every object in **Python** may have attributes and methods.
- Introspection reveals useful information about your program’s objects.

Code Introspection(CO2)

- Code **Introspection** is used for examining the classes, methods, objects, modules, keywords and get information about them so that we can utilize it.

- Python, being a dynamic, object-oriented programming language, provides tremendous introspection support. Python's support for introspection runs deep and wide throughout the language.

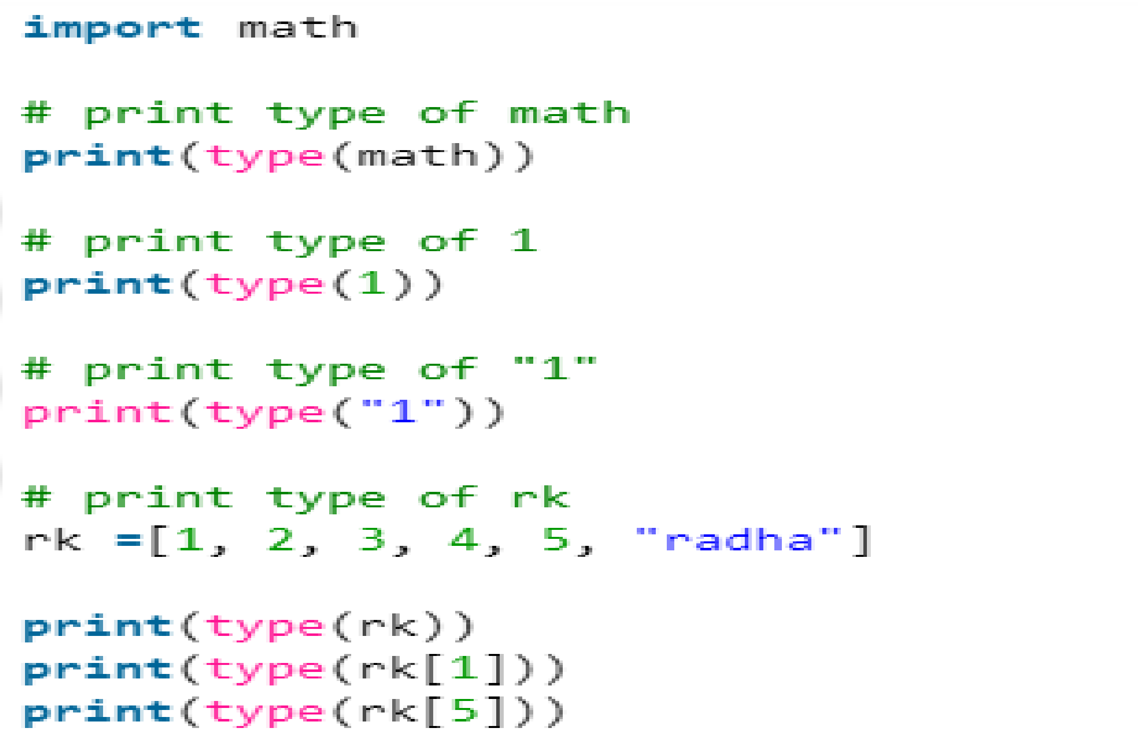
Introspection Types(CO2)

Python provides some built-in functions that are used for code introspection. **These are following:**

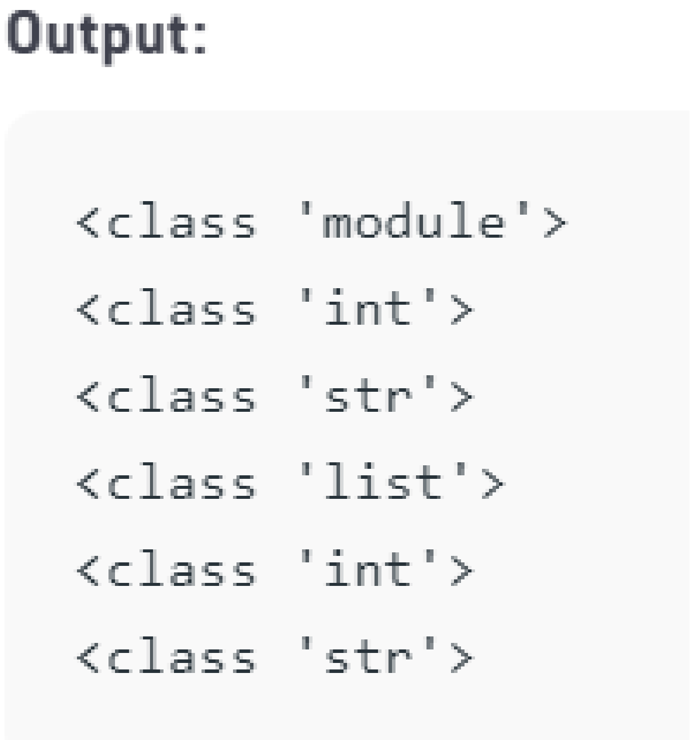
- **type() :** This function returns the type of an object
- **dir() :**This function return list of methods and attributes associated with that object.
- **str() :**This function converts everything into a string
- **id() :**This function returns a special id of an object.
- **isinstance():** Using this function, we can determine if a certain object is an instance of the specified class.
- **hasattr():** to check if it has the attribute

Code Introspection(CO2)

type() : This function returns the type of an object



```
import math

# print type of math
print(type(math))

# print type of 1
print(type(1))

# print type of "1"
print(type("1"))

# print type of rk
rk =[1, 2, 3, 4, 5, "radha"]

print(type(rk))
print(type(rk[1]))
print(type(rk[5]))
```

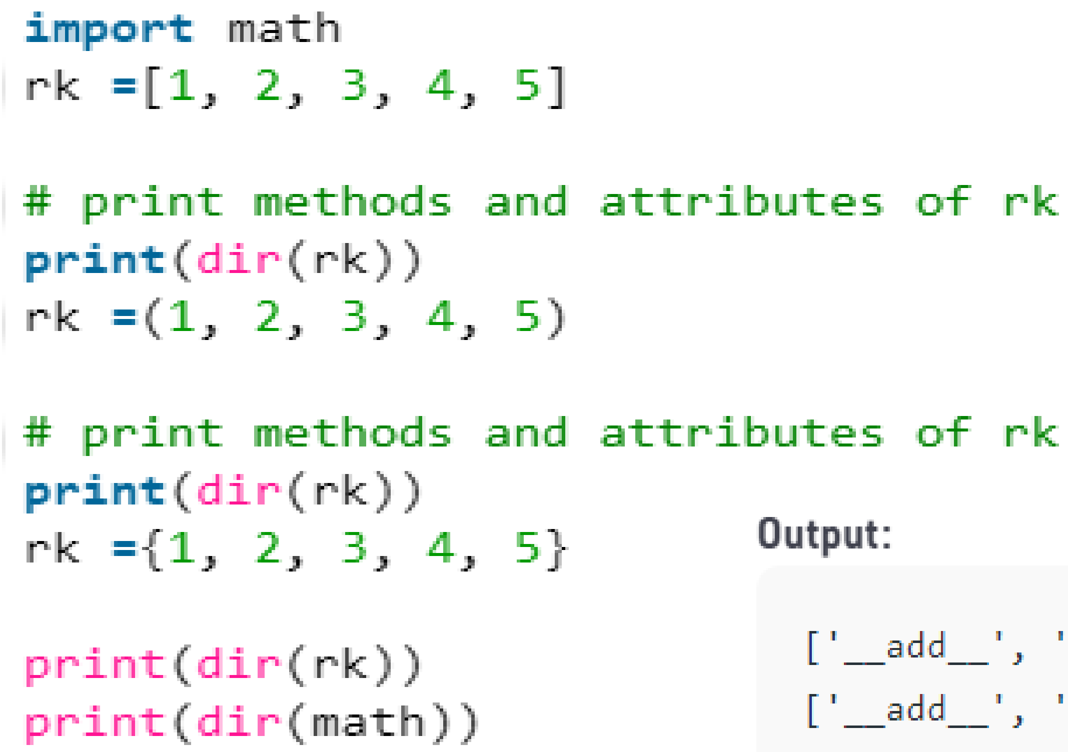


Output:

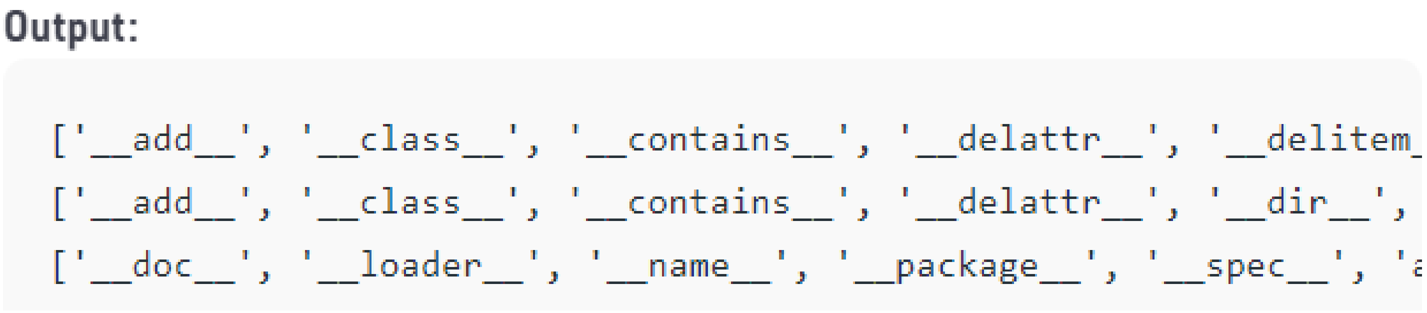
```
<class 'module'>
<class 'int'>
<class 'str'>
<class 'list'>
<class 'int'>
<class 'str'>
```

Code Introspection(CO2)

dir() : This function return list of methods and attributes associated with that object.



```
import math
rk =[1, 2, 3, 4, 5]

# print methods and attributes of rk
print(dir(rk))
rk =(1, 2, 3, 4, 5)

# print methods and attributes of rk
print(dir(rk))
rk ={1, 2, 3, 4, 5}

print(dir(rk))
print(dir(math))
```

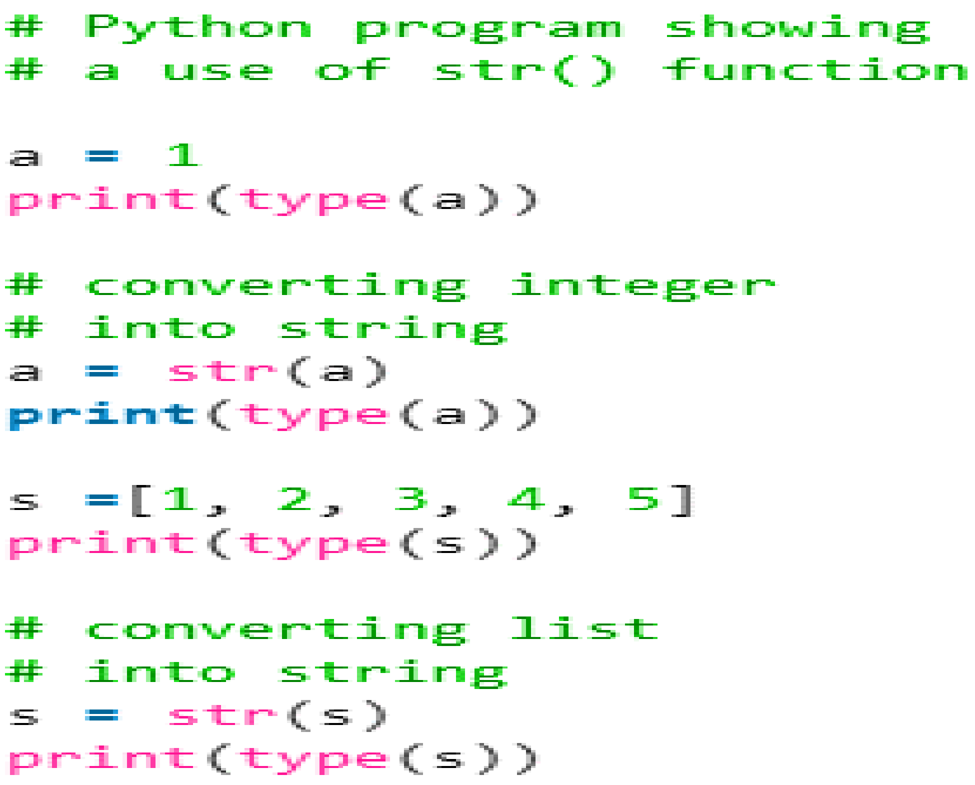


Output:

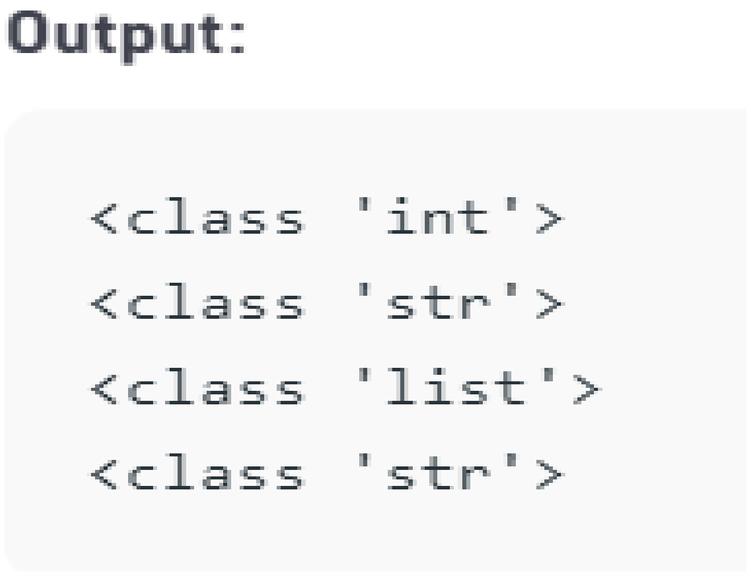
```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem_
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
['__doc__', '__loader__', '__name__', '__package__', '__spec__', '
```

Code Introspection(CO2)

str() : This function converts everything into a string



```
# Python program showing
# a use of str() function

a = 1
print(type(a))

# converting integer
# into string
a = str(a)
print(type(a))

s =[1, 2, 3, 4, 5]
print(type(s))

# converting list
# into string
s = str(s)
print(type(s))
```

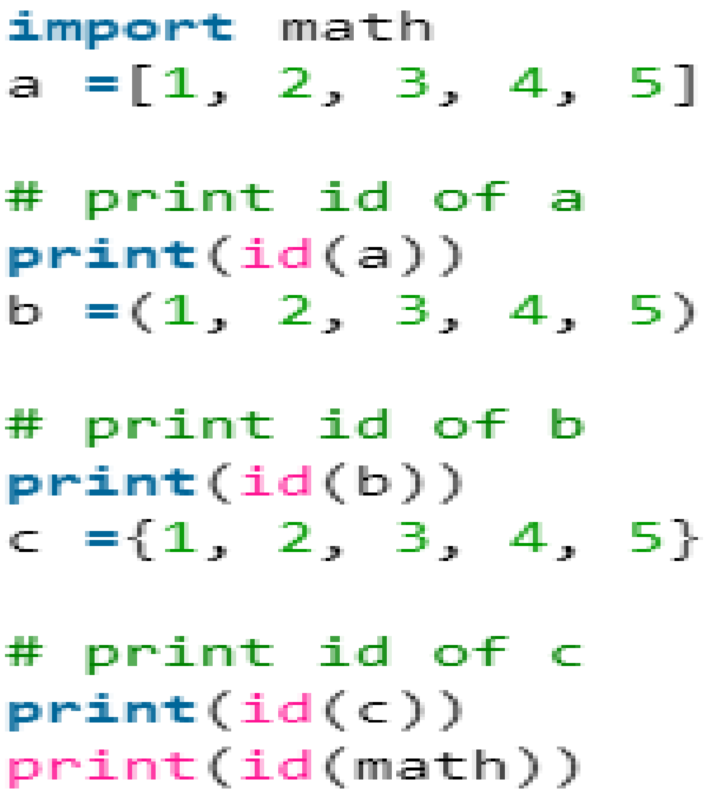


Output:

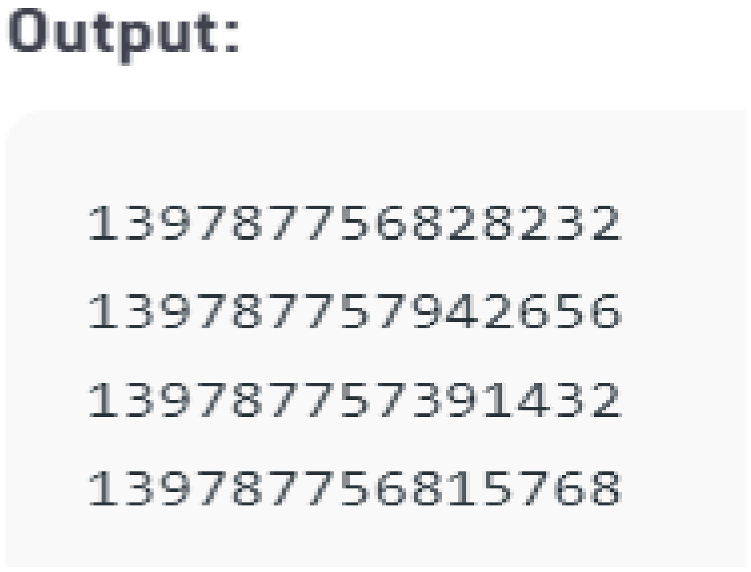
```
<class 'int'>
<class 'str'>
<class 'list'>
<class 'str'>
```

Code Introspection(CO2)

id() : This function returns a special id of an object.



```
import math
a =[1, 2, 3, 4, 5]

# print id of a
print(id(a))
b =(1, 2, 3, 4, 5)

# print id of b
print(id(b))
c ={1, 2, 3, 4, 5}

# print id of c
print(id(c))
print(id(math))
```



Output:

```
139787756828232
139787757942656
139787757391432
139787756815768
```

Method Of Code Introspection(CO2)

Function	Description
help()	It is used to find what other functions do.
hasattr()	Checks if an object has an attribute.
getattr()	Returns the contents of an attribute if there are some.
repr()	Return the string representation of object.
callable()	Checks if an object has a callable object or not.
Issubclass()	Checks if a specific class is a derived class of another class.
Isinstance()	Checks if an object is an instance of a specific class.
sys()	Give access to system specific variables and functions.

Introspection Objects(CO2)

1. In computer programming, **introspection** is the ability to determine the type of an **object** at runtime.

2. It is one of Python's strengths. Everything in Python is an **object** and we can examine those **objects:**

- type()
- dir()
- id()
- getattr()
- hasattr()
- globals()
- locals()

Introspection Scopes(CO2)

- Python contains two built-in functions for examining the content of scopes.
- The first function is globals().
- This returns a dictionary which represents the global namespace.
- Let’s define a variable a = 42 and call globals() again, and we can see that the binding of the name 'a' to the value of 42 has been added to the namespace.
- In fact, the dictionary returned by globals() is the global namespace. Let's create a variable tau and assign value 6.283185.
- We can now use this variable just like any other variables. The second function is locals().

Introspection Scopes(CO2)

- To really see locals() in action, we're are going to create another local scope, which we can do by defining a function that accepts a single argument, defines a variable X to have a value of 496, and then prints the locals() dictionary with a width of 10 characters.

- When run, we see that this function has the expected three entries in its local namespace. By using locals() to provide the dictionary, we can easily refer to local variables in format strings.

Inspect Module (CO2)

- The inspect module helps in checking the objects present in the code that we have written.

- As Python is an OOP language and all the code that is written is basically an interaction between these objects, hence the inspect module becomes very useful in inspecting certain modules or certain objects.

- We can also use it to get a detailed analysis of certain function calls or tracebacks so that debugging can be easier.

Methods to verify the type of token(CO2)

- The inspect module provides a lot of methods, these methods can be classified into two categories i.e. methods to verify the type of token and methods to retrieve the source of token.

- **The most commonly used methods of both categories are mentioned below.**

- **isclass():** The isclass() method returns True if that object is a class or false otherwise.

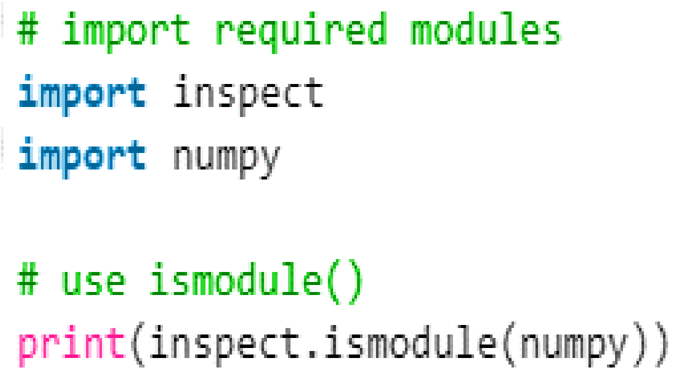
- When it combined with the getmembers() functions it shows the class and its type. It is used to inspect live classes.

- **ismodule():** This returns true if the given argument is an imported module.

Methods to verify the type of token(CO2)

The most commonly used methods of both categories are mentioned below.

- **isfunction():** This method returns *true* if the given argument is an inbuilt function name.
- **ismethod():** This method is used to check if the argument passed is the name of a method or not.



```
# import required modules
import inspect
import numpy

# use ismodule()
print(inspect.ismodule(numpy))
```

o/p---True

Methods to retrieve the source of token(CO2)

- **getclasstree()**: The getclasstree() method will help in getting and inspecting class hierarchy. It returns a tuple of that class and that of its preceding base classes. That combined with the getmro() method which returns the base classes helps in understanding the class hierarchy.

- **getmembers():** This method returns the member functions present in the module passed as an argument of this method.

- **stack():** This method helps in examining the interpreter stack or the order in which functions were called.

Methods to retrieve the source of token(CO2)

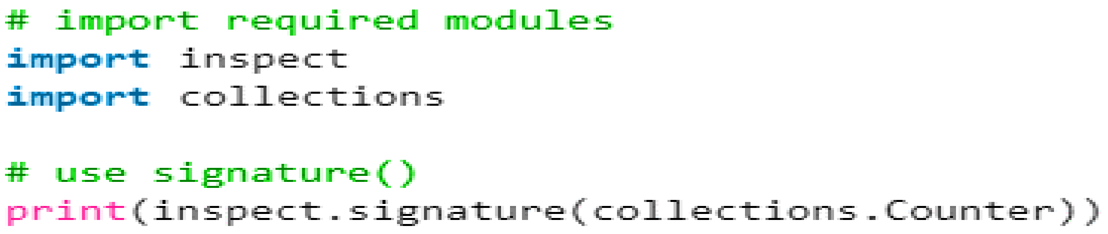
- **getsource():** This method returns the source code of a module, class, method, or a function passes as an argument of getsource() method.

- **getmodule():** This method returns the module name of a particular object pass as an argument in this method.

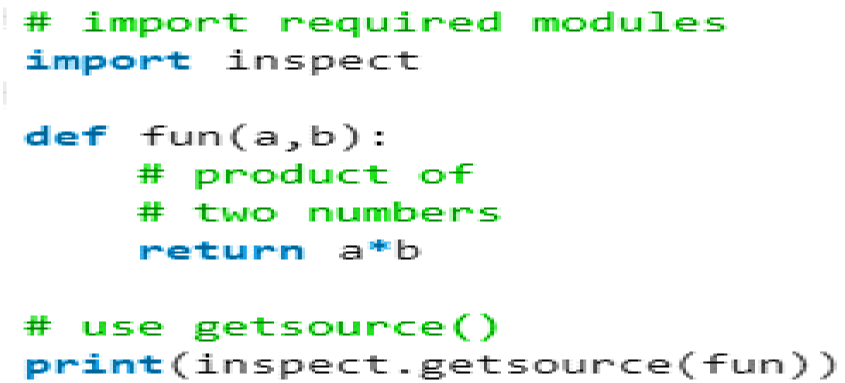
- **getdoc():** The getdoc() method returns the documentation of the argument in this method as a string.

- **signature():** The signature() method helps the user in understanding the attributes which are to be passed on to a function.

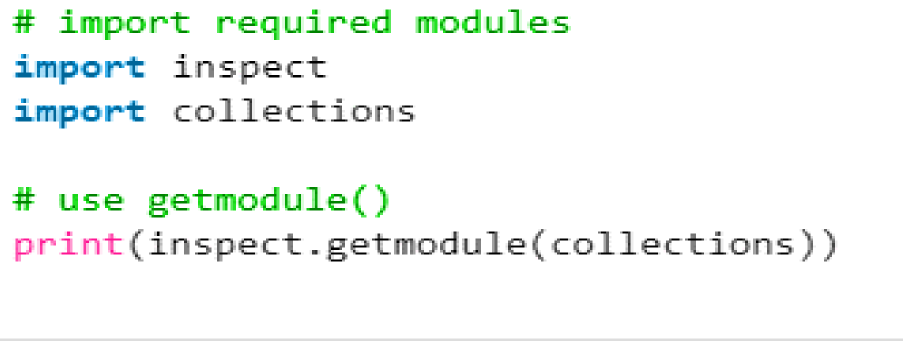
Program of Inspect(CO2)



```python
# import required modules
import inspect
import collections

# use signature()
print(inspect.signature(collections.Counter))
```



```python
# import required modules
import inspect

def fun(a,b):
      # product of
      # two numbers
      return a*b

# use getsource()
print(inspect.getsource(fun))
```



```python
# import required modules
import inspect
import collections

# use getmodule()
print(inspect.getmodule(collections))
```

Introspect Tools(CO2)

- We'll now build a tool to introspect objects by leveraging the interesting techniques we've discussed and bring them together into a useful program.

- Our objective is to create a function, which when passed a single object prints out a nicely formatted dump of that object's attributes with rather more clarity.

- This is a small tool that we are going to create and the main use case of this tool is to help us identify a different aspect of the objects with respect to its type, methods, attributes, and documentation.

Weekly Assignment

Q1. Python provides some built-in functions that are used for code introspection. Explain them with examples.

Q2. Write short note on following functions:

Type()

Dir()

Str()

Q3. What are methods of code introspection? Explain.

Q4. What does isinstance() and issubclass() describes?

Q5. What does getattr() and hasattr() method are used for?

MCQs

1. Which is not introspection method?

a) help()

b) sys()

c) repr()

d) remove()

Answer: d

2.Which of the following is not an introspection object?

a) type()

b) id()

c) dir()

d) Unbounded()

Answer: d

3.Special methods need to be explicitly called during object creation.

a) True

b) False

Answer: b

Daily Quiz

1. Is the following Python code valid?

```
class B(object):
def first(self):
print("First method called")
def second():
print("Second method called")
 ob = B()
 B.first(ob)
```

a) It isn’t as the object declaration isn’t right

b) It isn’t as there isn’t any __init__ method for initializing class members

c) Yes, this method of calling is called unbounded method call

d) Yes, this method of calling is called bounded method call

Answer: c

Old Question Papers

Q1. In the following program find out which objects are callable.

Callable.py

```
class Car(object):
    def setName(self,  name):
      self.name  = name
def fun():
   pass
c = Car()
print(callable(fun))
print(callable(c.setName))
print(callable([]))
print(callable(1))
```

Q2. Explain various methods used for introspection in python.

Q3.What does callable() describes?

Old Question Papers

Q4. Explain various ways of inspecting Python objects.

Q5. What should be the output of the following program?

```
# subclass.py
class Object(object):
    def __init__(self):
      pass
class Wall(Object):
    def __init__(self):
      pass
print(issubclass(Object, Object))
print(issubclass(Object, Wall))
print(issubclass(Wall, Object))
print(issubclass(Wall, Wall))
```

Expected Questions for University Exam

Q1.What does callable() describes?

Q2. Explain various ways of inspecting Python objects.

Q3. What are introspection tools? Explain them briefly.

Q4. Explain with example issubclass() and getattr() functions.

Q5. Explain inspect modules with example.

Q6. What should be the output of the following program?

```
class Object(object):
    def __init__(self):
      pass
class Wall(Object):
    def __init__(self):
      pass
print(issubclass(Object, Object))
print(issubclass(Object, Wall))
print(issubclass(Wall, Object))
print(issubclass(Wall, Wall))
```

Video Links/You Tube Links

- – https://www.python-course.eu/python3_inheritance.php
- – https://www.youtube.com/watch?v=u9x475OGj_U
- – https://www.youtube.com/watch?v=byHcYRpMgI4
- – https://www.youtube.com/watch?v=FsAPt_9Bf3U

TEXT & REFERENCE BOOKS

Text books:

(1) Magnus Lie Hetland, "Beginning Python-From Novice to Professional"—Third Edition, Apress

(2) Peter Morgan, Data Analysis from Scratch with Python, AI Sciences

(3) Allen B. Downey, “Think Python: How to Think Like a Computer Scientist”, 2nd edition, Updated for Python 3, Shroff/O‘Reilly Publishers, 2016

(4) Miguel Grinberg, Developing Web applications with python, OREILLY

Reference Books:

(1) Dusty Phillips, Python 3 Object-oriented Programming - Second Edition, O’Reilly

(2) Burkhard Meier, Python GUI Programming Cookbook - Third , Packt

(3) DOUG HELLMANN, THE PYTHON 3 STANDARD LIBRARY BY EXAMPLE, :Pyth 3 Stan Libr Exam _2 (Developer's Library) 1st Edition, Kindle Edition.

(4) Kenneth A. Lambert, ―Fundamentals of Python: First Programs‖, CENGAGE Learning, 2012.

SUMMARY

Introspection is an ability to determine the type of an object at runtime. Everything in **python** is an object. Every object in **Python** may have attributes and methods.

THANK YOU