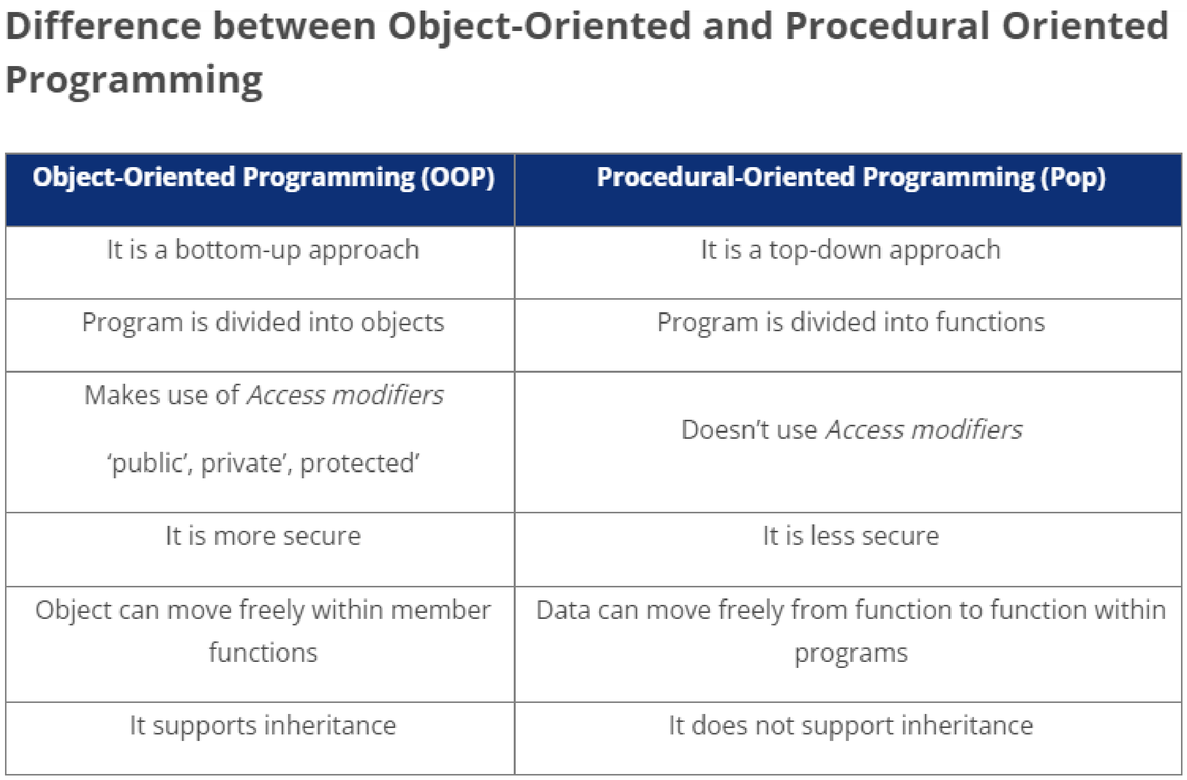
OOP (Object Oriented Programming)

- Object Oriented Programming is a way of programming that uses "objects" to represent data and methods.



Difference between Object-Oriented and Procedural Oriented Programming

Object-Oriented Programming (OOP)	Procedural-Oriented Programming (Pop)
It is a bottom-up approach	It is a top-down approach
Program is divided into objects	Program is divided into functions
Makes use of *Access modifiers* 'public', private', protected'	Doesn't use *Access modifiers*
It is more secure	It is less secure
Object can move freely within member functions	Data can move freely from function to function within programs
It supports inheritance	It does not support inheritance

OOP Concepts

- Class
- Objects
- Methods
- Inheritance
- Polymorphism
- Data abstraction
- Data encapsulation

Class:

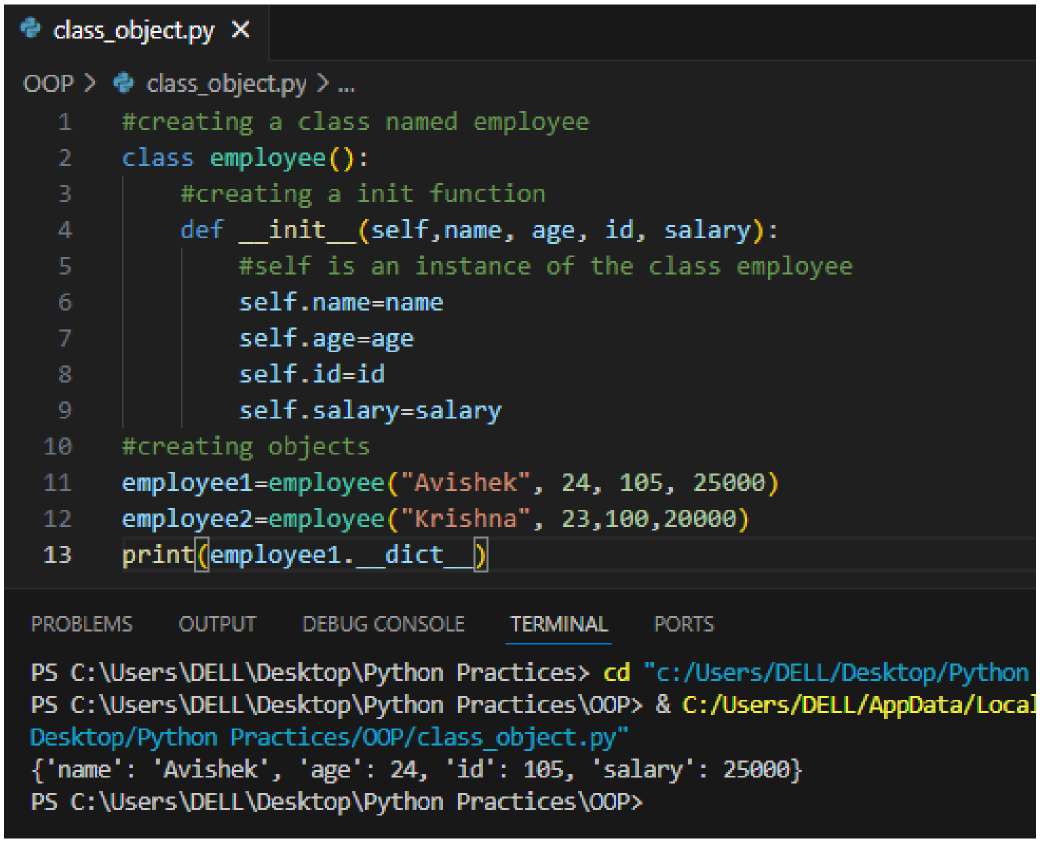
- collection of objects defining the common attributes and behaviors
- is defined under a keyword " class"
  Example:
  class Class1(): // class 1 is the name of the class

Object:

- an instance of a class
- has state and behavior and can access the data
  Syntax:
  obj = class1()

Here, obj is the object of class1.

Class and Object Creation in Python



```python
#creating a class named employee
class employee():
    #creating a init function
    def __init__(self,name, age, id, salary):
        #self is an instance of the class employee
        self.name=name
        self.age=age
        self.id=id
        self.salary=salary
#creating objects
employee1=employee("Avishek", 24, 105, 25000)
employee2=employee("Krishna", 23,100,20000)
print(employee1.__dict__)
```

```
PS C:\Users\DELL\Desktop\Python Practices> cd "c:/Users/DELL/Desktop/Python
PS C:\Users\DELL\Desktop\Python Practices\OOP> & C:/Users/DELL/AppData/Local
Desktop/Python Practices/OOP/class_object.py"
{'name': 'Avishek', 'age': 24, 'id': 105, 'salary': 25000}
PS C:\Users\DELL\Desktop\Python Practices\OOP>
```

- employee1() and employee2() are the objects instantiated against the class ”employee”.
- the word (__dict__) is a “dictionary” which prints all the values of object ‘emp1’ against the given parameter (name, age, salary).
- (__init__) acts like a constructor that is invoked whenever an object is created.

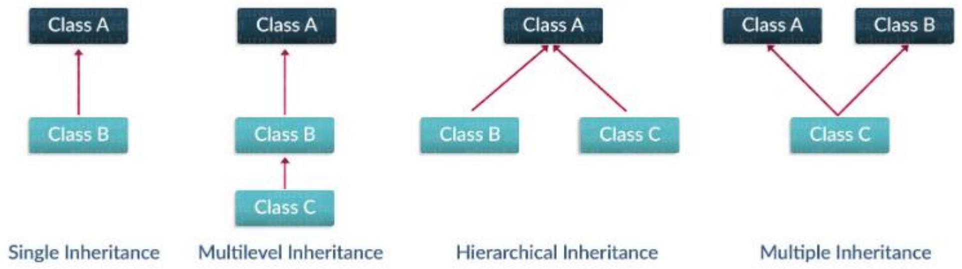
OOP Methodologies

Inheritance

- Inheriting or transfer of characteristics from parent to child class without any modification.

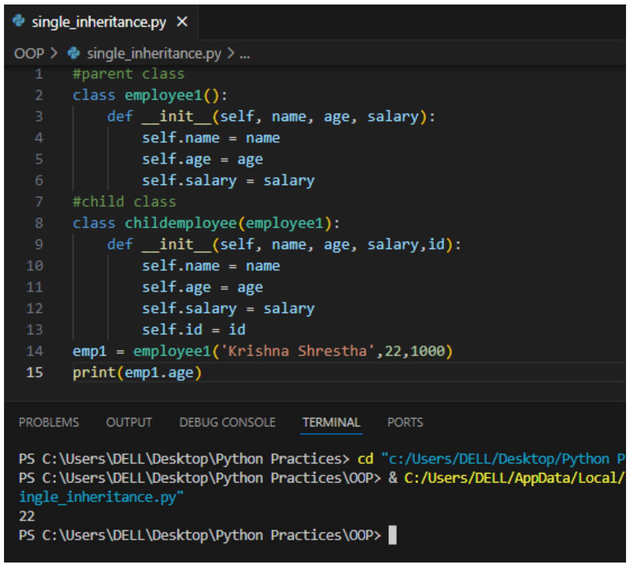
- The new class is called the **derived/child** class and the one from which it is derived is called a **parent/base** class

Types of Inheritance:



Single Inheritance:

- Enables a derived class to inherit characteristics from a single parent class



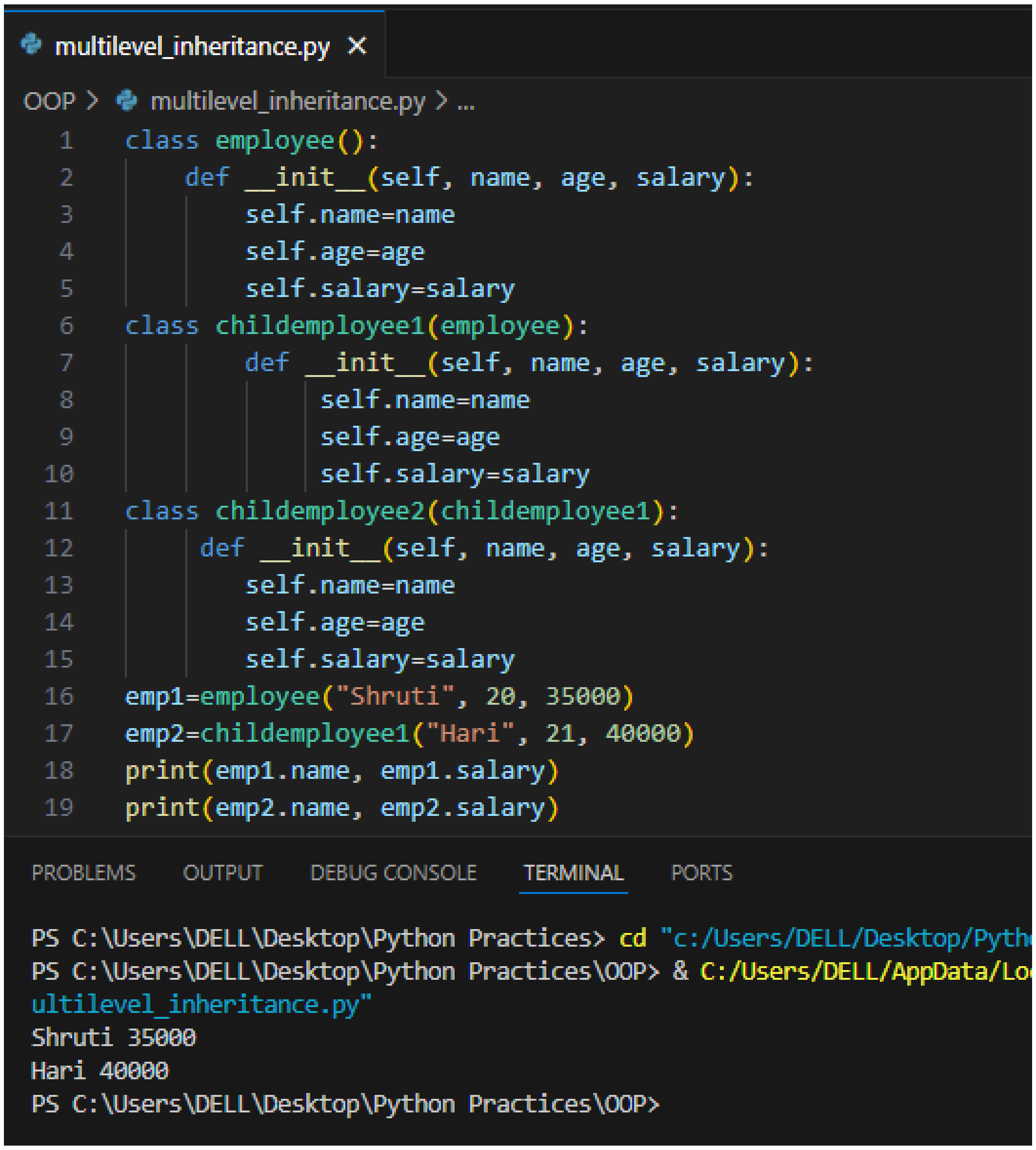
```python
#parent class
class employee1():
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
#child class
class childemployee(employee1):
    def __init__(self, name, age, salary,id):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = id
emp1 = employee1('Krishna Shrestha',22,1000)
print(emp1.age)
```

```
PS C:\Users\DELL\Desktop\Python Practices> cd "c:/Users/DELL/Desktop/Python P
PS C:\Users\DELL\Desktop\Python Practices\OOP> & C:/Users/DELL/AppData/Local/
ingle_inheritance.py"
22
PS C:\Users\DELL\Desktop\Python Practices\OOP>
```

- I am taking the parent class and created a constructor (__init__), class itself is initializing the attributes with parameters(‘name’, ‘age’ and ‘salary’).
- Created a child class ‘childemployee’ which is inheriting the properties from a parent class and finally instantiated objects ’emp1′ and ’emp2′ against the parameters.
- Finally, I have printed the age of emp1. Well, you can do a hell lot of things like print the whole dictionary or name or salary.

Multilevel Inheritance

- Enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.


multilevel_inheritance.py
OOP > multilevel_inheritance.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

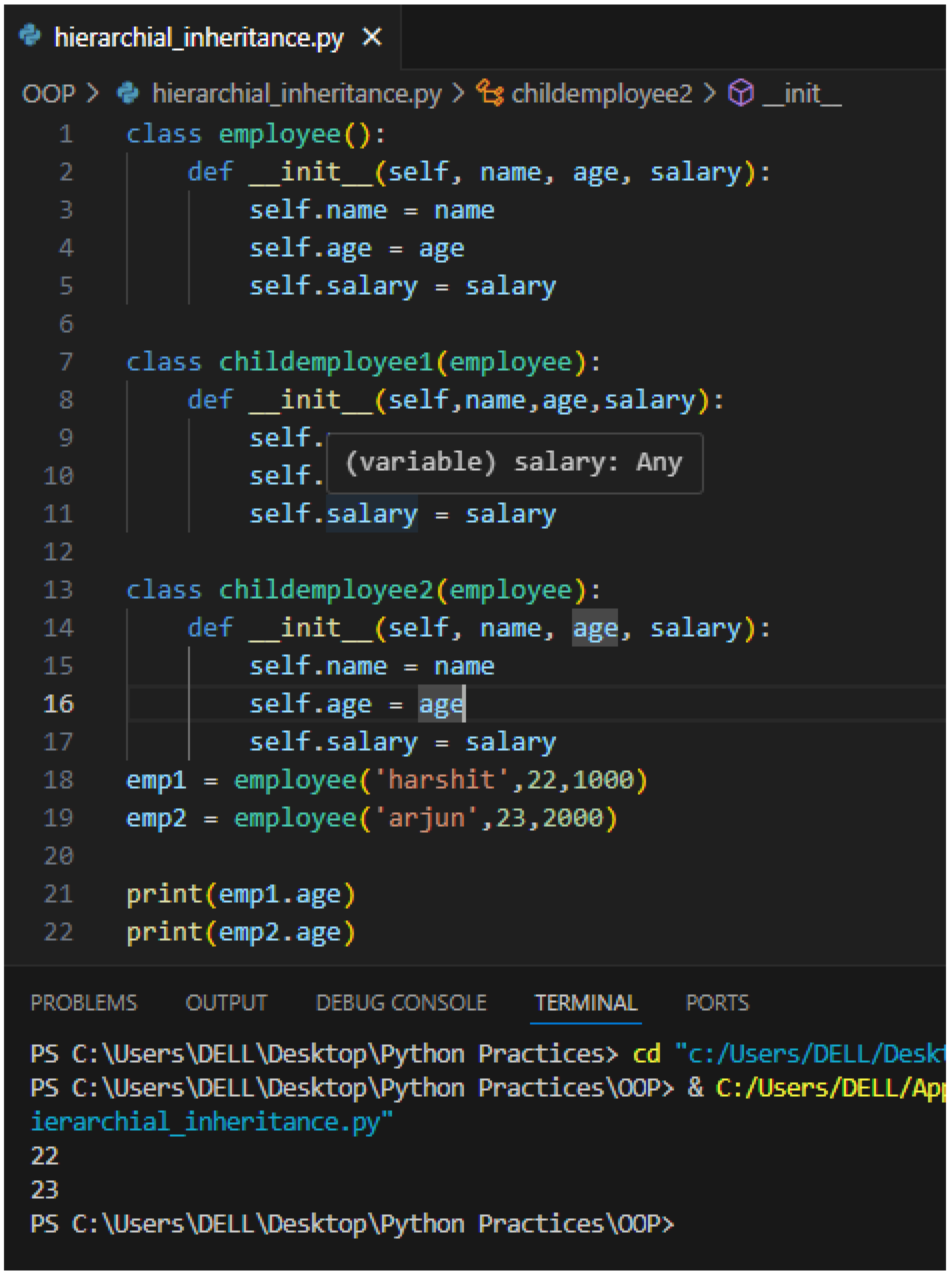
```python
class employee():
    def __init__(self, name, age, salary):
        self.name=name
        self.age=age
        self.salary=salary
class childemployee1(employee):
        def __init__(self, name, age, salary):
            self.name=name
            self.age=age
            self.salary=salary
class childemployee2(childemployee1):
     def __init__(self, name, age, salary):
        self.name=name
        self.age=age
        self.salary=salary
emp1=employee("Shruti", 20, 35000)
emp2=childemployee1("Hari", 21, 40000)
print(emp1.name, emp1.salary)
print(emp2.name, emp2.salary)
```

```
PS C:\Users\DELL\Desktop\Python Practices> cd "c:/Users/DELL/Desktop/Pyth
PS C:\Users\DELL\Desktop\Python Practices\OOP> & C:/Users/DELL/AppData/Lo
ultilevel_inheritance.py"
Shruti 35000
Hari 40000
PS C:\Users\DELL\Desktop\Python Practices\OOP>
```

- In the above program, employee is the super class, childemployee1 is the child class. The childemployee1 class acts as the parent class for class childemployee2.
- Two objects emp1 and emp2 are instantiated from superclass and parent class respectively by passing the parameters like name, age and salary.

Hierarchical Inheritance

Hierarchical level inheritance enables more than one derived class to inherit properties from a parent class.

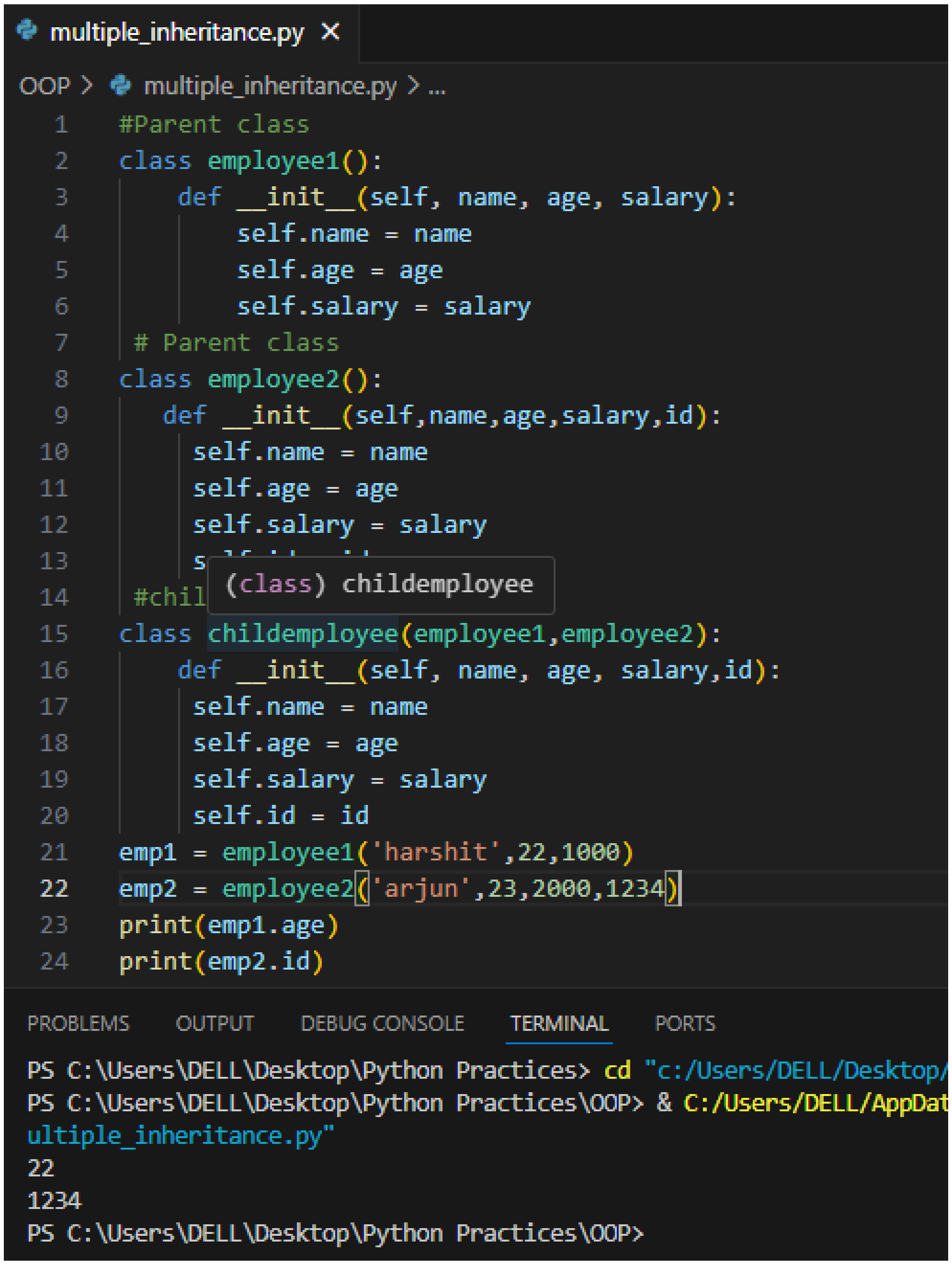

hierarchial_inheritance.py
OOP > hierarchial_inheritance.py > childemployee2 > __init__
(variable) salary: Any
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```python
class employee():
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary

class childemployee1(employee):
    def __init__(self,name,age,salary):
        self.
        self.
        self.salary = salary

class childemployee2(employee):
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
emp1 = employee('harshit',22,1000)
emp2 = employee('arjun',23,2000)

print(emp1.age)
print(emp2.age)
```

```
PS C:\Users\DELL\Desktop\Python Practices> cd "c:/Users/DELL/Deskt
PS C:\Users\DELL\Desktop\Python Practices\OOP> & C:/Users/DELL/Ap
ierarchial_inheritance.py"
22
23
PS C:\Users\DELL\Desktop\Python Practices\OOP>
```

Multiple Inheritance

Multiple level inheritance enables one derived class to inherit properties from more than one base class.



multiple_inheritance.py
OOP > multiple_inheritance.py > ...
(class) childemployee
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```python
#Parent class
class employee1():
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
 # Parent class
class employee2():
   def __init__(self,name,age,salary,id):
      self.name = name
      self.age = age
      self.salary = salary
      s
 #chil
class childemployee(employee1,employee2):
    def __init__(self, name, age, salary,id):
      self.name = name
      self.age = age
      self.salary = salary
      self.id = id
emp1 = employee1('harshit',22,1000)
emp2 = employee2('arjun',23,2000,1234)
print(emp1.age)
print(emp2.id)
```

```
PS C:\Users\DELL\Desktop\Python Practices> cd "c:/Users/DELL/Desktop/
PS C:\Users\DELL\Desktop\Python Practices\OOP> & C:/Users/DELL/AppDat
ultiple_inheritance.py"
22
1234
PS C:\Users\DELL\Desktop\Python Practices\OOP>
```