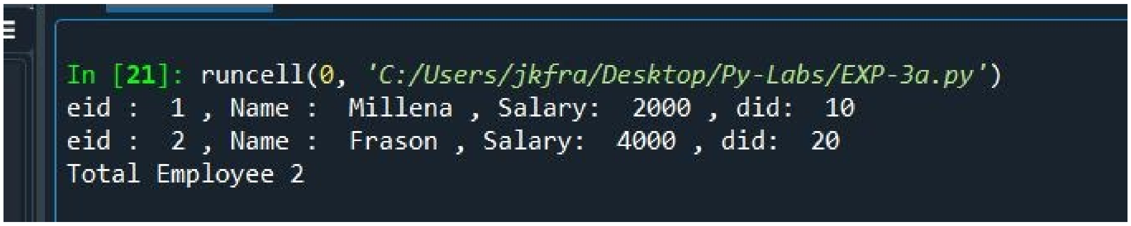
EXPERIMENT-3

Frason Francis/SE-IT/201903020

(A) Classes,Objects, Constructors, Inner class and Static method.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: jkfrason
"""
class Employee():
    'Common base class for all employees'
    empCount = 0

    #constructor
    def __init__(self, eid, name, salary, did):
        self.eid = eid
        self.name = name
        self.salary = salary
        self.did = did
        Employee.empCount += 1

     #instance method
    def displayEmployee(self):
        print("eid : ", self.eid,", Name : ", self.name,  ", Salary: ", self.salary,  ", did: ", self.did)

    @staticmethod
    def info(msg):
        print("Total Employee %d" % Employee.empCount)

"This would create first object of Employee class"
emp1 = Employee(1,"Millena", 2000,10)
"This would create second object of Employee class"
emp2 = Employee(2,"Frason", 4000,20)
emp1.displayEmployee()
emp2.displayEmployee()

Employee.info("calling the static method")
```



```
In [21]: runcell(0, 'C:/Users/jkfra/Desktop/Py-Labs/EXP-3a.py')
eid :  1 , Name :  Millena , Salary:  2000 , did:  10
eid :  2 , Name :  Frason , Salary:  4000 , did:  20
Total Employee 2
```

Theory:

Static Methods

The third method, MyClass.staticmethod was marked with a @staticmethod decorator to flag it as a *static method*. This type of method takes neither a self nor a cls parameter (but of course it's free to accept an arbitrary number of other parameters). Therefore a static method can neither modify object state nor class state. Static methods are restricted in what data they can access - and they're primarily a way to namespace your methods.

Instance Methods

The first method on MyClass, called method, is a regular *instance method*. That's the basic, no-frills method type you'll use most of the time. You can see the method takes one parameter, self, which points to an instance of MyClass when the method is called (but of course instance methods can accept more than just one parameter). Through the self parameter, instance methods can freely access attributes and other methods on the same object. This gives them a lot of power when it comes to modifying an object's state.

Inner class

A class defined in another class is known as inner class or nested class. If an object is created using child class means inner class then the object can also be used by parent class or root class. A parent class can have one or more inner class but generally inner classes are avoided.

(B) Different types of Inheritance

Code:

person.py

```
class Person:
   def __init__(self, first_name, last_name, age):
      self.first_name = first_name
      self.last_name = last_name
```

```
        self.age = age

    def introduce(self):
        return f"Hi. I'm {self.full_name}. I'm {self.age} years old."

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, value):
        if value <= 0:
            raise ValueError('Age is not valid')

        self.__age = value

    @property
    def full_name(self):
        return f"{self.first_name} {self.last_name}"
```

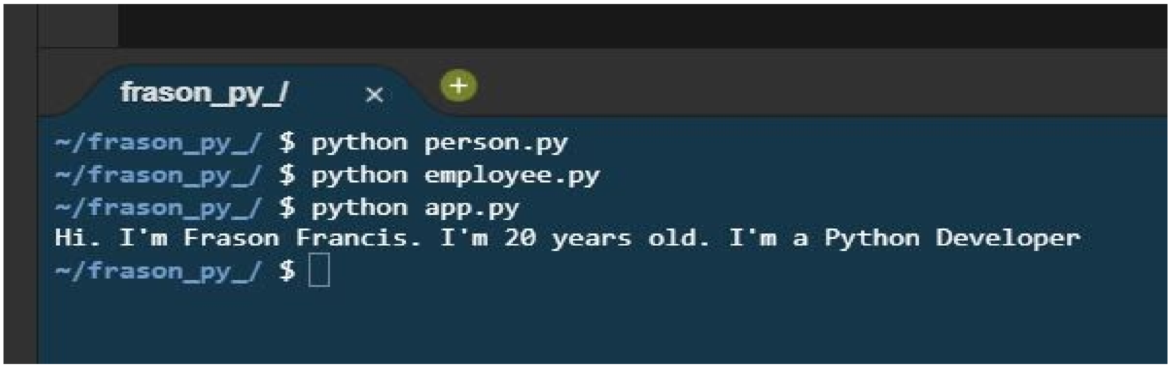
employee.py

```
#importing the class from the person for
#code resusebality.

from person import Person


class Employee(Person):

    def __init__(self, first_name, last_name, age, job_title, salary):
        super().__init__(first_name, last_name, age)

        self.job_title = job_title
        self.salary = salary

    @property
    def job_title(self):
        return self.__job_title

    @job_title.setter
    def job_title(self, value):
        self.__job_title = value
    @property
    def salary(self):
        return self.__salary

    @salary.setter
    def salary(self, value):
        if value < 0:
            raise ValueError('Salary must be greater than zero.')

        self.__salary = value

    def introduce(self):
        introduction = super().introduce()
        introduction += f" I'm a {self.job_title}"
        return introduction
```

app.py

```
from employee import Employee

employee = Employee('Frason', 'Francis', 20 , 'Python Developer', 120000)
print(employee.introduce())
```

Output:

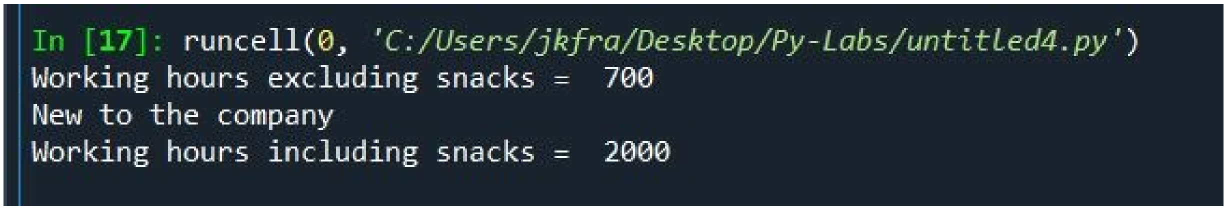


```
~/frason_py_/ $ python person.py
~/frason_py_/ $ python employee.py
~/frason_py_/ $ python app.py
Hi. I'm Frason Francis. I'm 20 years old. I'm a Python Developer
~/frason_py_/ $
```

- Used inheritance to model the is-a relationship.
- Inheritance allows a class to inherit attributes and methods from another class.
- Inheritance promotes code reusability by reusing code from an existing class.

(C) Polymorphism using Operator overloading, Method overloading, Method overriding, Abstract class, Abstract method and Interfaces in Python.

```
class Employee:
   def salary(self):
    pass

class tester(Employee):
  def job(self,hours):
    print("Working hours excluding snacks = ", 100*hours)

  def intern(self):
    return "New to the company"

class analyst(Employee):
  def analyse(self,hours,snacks):
    print("Working hours including snacks = ",100*hours*snacks)


class manager(Employee):
  def __init__(self, length):
    super().__init__("Square")
    self.length = length


obj1 = tester()
obj1.job(7) #working for 7 hours
print(obj1.intern())
obj2 = analyst()
obj2.analyse(10,2) #no. of snack taken 2
```



```
In [17]: runcell(0, 'C:/Users/jkfra/Desktop/Py-Labs/untitled4.py')
Working hours excluding snacks =  700
New to the company
Working hours including snacks =  2000
```

- Polymorphism means having vivid or different forms. In the programming world, Polymorphism refers to the ability of the function with the same name to carry different functionality altogether. It creates a structure that can use many forms of objects.
- An abstract method is a method that’s declared by the Python interface, but it may not have a useful implementation. The abstract method must be overridden by the concrete class that implements the interface in question.