

Seatwork 5.1: OOP Methodologies

This seatwork is in response to your posts in discussion 4.1 (module 4 Q&A).

Instructions:

1. Run the procedures below, note your observations.
2. Answer the given exercises/questions after the procedures.

Procedures:

```
In [1]: class class1():  
        # This is just a sample for class  
        pass
```

```
In [2]: class employee():  
        def __init__(self, name, age, emp_id, salary):  
            self.name = name  
            self.age = age  
            self.salary = salary  
            self.id = emp_id
```

```
In [3]: emp1 = employee('Roman', 22, '0001', 1234)  
emp2 = employee('Richard', 23, '0002', 2345)
```

```
In [4]: print(emp1.__dict__)  
  
{'name': 'Roman', 'age': 22, 'salary': 1234, 'id': '0001'}
```

```
In [29]: # Single Inheritance  
class employee(): # Parent Class  
    emp_id = 0  
  
    def __init__(self, name, age, salary):  
        self.name = name  
        self.age = age  
        temp_id = employee.emp_id
```

```

        employee.emp_id += 1
        self.id = temp_id

    def printDetails(self):
        print(self.name, self.id)

class partTime(employee): # Child Class

    def status_PT(self):
        self.printDetails()
        print("PART TIME EMPLOYEE")

emp1 = partTime('Roman', 22, 1234.00)
emp2 = partTime('Richard', 23, 2345.00)

emp1.status_PT()
emp2.status_PT()

```

```

Roman 0
PART TIME EMPLOYEE
Richard 1
PART TIME EMPLOYEE

```

```

In [60]: # Mutliple Inheritance
class employee(): # Parent Class
    emp_id = 0

    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        temp_id = employee.emp_id
        employee.emp_id += 1
        self.id = temp_id

    def printDetails(self):
        print(self.name, self.id)

class professional():
    prc_id = 0

    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.pro_id = self.getID()

```

```

def getID(self):
    temp_id = professional.prc_id
    professional.prc_id += 1
    self.pro_id = temp_id
    return str(self.pro_id)

class consultant(employee, professional):

    def status(self):
        print(self.name, self.age)
        print("PROFESSIONAL ID: {}".format(self.getID()))
        print("EMPLOYEE ID: {}".format(self.id), '\n')

consultant1 = consultant('Roman', 23, 1234.00)
consultant1.printDetails()
consultant1.status()

consultant2 = consultant('Richard', 29, 2345.00)
consultant2.printDetails()
consultant2.status()

```

```

Roman 0
Roman 23
PROFESSIONAL ID: 0
EMPLOYEE ID: 0

```

```

Richard 1
Richard 29
PROFESSIONAL ID: 1
EMPLOYEE ID: 1

```

```

In [90]: # Multilevel Inheritance
class employee(): # Parent Class
    emp_id = 0

    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = setEmpID()

    def printDetails(self):
        print(self.name, self.id)

```

```

def setEmpID(self):
    temp_id = employee.emp_id
    employee.emp_id += 1
    return temp_id

class middle(employee): # First Derived Class

    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = self.setEmpID()
        self.deptID = self.setDept()

    def setDept(self, newID=None):
        if newID == None:
            print("No Valid Dept ID Set.")
            self.deptID = int(input("Input new ID: "))
        else:
            self.deptID = newID
            print("Department ID Set.\n")

    def status(self):
        print("{} has ID No. {}".format(
            self.name, self.id))

class supervisor(middle):

    def supervise(self):
        print("Employee {} is now supervising.".format(self.id))

supervisor1 = supervisor('Roman', 29, 1234.00)
supervisor1.supervise()

```

No Valid Dept ID Set.

Input new ID: 2

Department ID Set.

Employee 0 is now supervising.

```

In [91]: # Hierarchical Inheritance
class employee(): # Parent Class
    emp_id = 0

```

```
def __init__(self, name, age, salary):
    self.name = name
    self.age = age
    self.salary = salary
    self.id = setEmpID()
```

```
def printDetails(self):
    print(self.name, self.id)
```

```
def setEmpID(self):
    temp_id = employee.emp_id
    employee.emp_id += 1
    return temp_id
```

```
class middle(employee): # First Child Class
```

```
def __init__(self, name, age, salary):
    self.name = name
    self.age = age
    self.salary = salary
    self.id = self.setEmpID()
    self.deptID = self.setDept()
```

```
def setDept(self, newID=None):
    if newID == None:
        print("No Valid Dept ID Set.")
        self.deptID = int(input("Input new ID: "))
    else:
        self.deptID = newID
    print("Department ID Set.\n")
```

```
def status(self):
    print("Middle: {} has ID No. {}".format(
        self.name, self.id))
```

```
class top(employee): # Second Child Class
```

```
def __init__(self, name, age, salary):
    self.name = name
    self.age = age
    self.salary = salary
    self.id = self.setEmpID()
    self.deptID = self.setDept()
```

```

def setDept(self, newID=None):
    if newID == None:
        print("No Valid Dept ID Set.")
        self.deptID = int(input("Input new ID: "))
    else:
        self.deptID = newID
    print("Department ID Set.\n")

def status(self):
    print("Top: {} has ID No. {}".format(
        self.name, self.id))

emp1 = middle('Roman', 29, 1234.00)
emp2 = top('Richard', 29, 2345.00)

emp1.status()
emp2.status()

```

No Valid Dept ID Set.
Input new ID: 2
Department ID Set.

No Valid Dept ID Set.
Input new ID: 2
Department ID Set.

Middle: Roman has ID No. 0
Top: Richard has ID No. 1

In [94]: *# Polymorphism*

```

class employee(): # Parent Class
    emp_id = 0

    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = setEmpID()

    def printDetails(self):
        print(self.name, self.id)

    def setEmpID(self):
        temp_id = employee.emp_id
        employee.emp_id += 1

```

```
    return temp_id
```

```
class middle(employee): # First Child Class
```

```
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
        self.id = self.setEmpID()
        self.deptID = self.setDept()

    def printDetails(self):
        print("MIDDLE:")
        print(self.name, self.id)

    def setDept(self, newID=None):
        if newID == None:
            print("No Valid Dept ID Set.")
            self.deptID = int(input("Input new ID: "))
        else:
            self.deptID = newID
            print("Department ID Set.\n")

    def status(self):
        print("Middle: {} has ID No. {}".format(
            self.name, self.id))
```

```
class top(employee): # Second Child Class
```

```
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.__salary = salary
        self.id = self.setEmpID()
        self.deptID = self.setDept()

    def printDetails(self):
        print("TOP: ")
        print(self.name, self.id)

    def setDept(self, newID=None):
        if newID == None:
            print("No Valid Dept ID Set.")
            self.deptID = int(input("Input new ID: "))
        else:
```

```

        self.deptID = newID
        print("Department ID Set.\n")

    def status(self):
        print("Top: {} has ID No. {}".format(
            self.name, self.id))

    def printSalary(self):
        print(self.__salary)

emp1 = top('Roman', 29, 1234.00)
emp1.printSalary()
emp1.salary

```

No Valid Dept ID Set.

Input new ID: 2

Department ID Set.

1234.0

```

-----
AttributeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_15720\3687215997.py in <module>
    72 emp1 = top('Roman', 29, 1234.00)
    73 emp1.printSalary()
---> 74 emp1.salary

AttributeError: 'top' object has no attribute 'salary'

```

Exercises:

Question 1: Methods can be of two types, getters and setters. Getters take values by parameter passing and setters set values in the given class.

- Create a class student with appropriate attributes.
- Create getter and setter methods for it.

Note: Take into account that the student's ID number, name and other important details **must not be publicly available**. It must only be accessed by appropriate class methods.

Question 2: A graduate student is different from a student in the undergraduate program.

- Create a class undergrad that derives from the class student.
- Create a class graduate that derives from the class student.
- Create appropriate attributes and methods for each derived class.

What type of inheritance is shown here?

Question 3: A graduate student may be in the master's program or the doctorate program. Create the appropriate class for the doctorate program that sets it apart from the rest of the graduate programs.

- Justify the type of inheritance you used to answer this question.
- How did you demonstrate polymorphism here?

Question 4: Create 3 instances of the master's class.

- Test all the methods of the class.
- Use the **del** command to delete the object after declaration and testing all the methods.
- Call the same object and test the same methods. Does it work?

Question 1:



Question 1

Getters and Setters

```
In [4]: class students():
        def __init__(self, name, age, stud_id, bday):
            self.name = name
            self.age = age
            self.stud_id = stud_id
            self.bday = str(bday)

        def get_info(self):
            access = input('Input Birthday of Student in MMDDYYYY: ')


            if access == self.bday:
                print(f"Name: {self.name}, Age: {self.age}, Student ID: {self.stud_id}")
            else:
                print("Access Denied")

student1 = students('Ivan', 20, 'TIP-0001', '06051985')
student2 = students('Navi', 21, 'TIP-0002', '102898')

student1.get_info()
student2.get_info()
```

```
Input Birthday of Student in MMDDYYYY: 06051985
Name: Ivan, Age: 20, Student ID: TIP-0001
Input Birthday of Student in MMDDYYYY: 102898
Name: Navi, Age: 21, Student ID: TIP-0002
```

Question 2:

jupyter Assignment 3.2_Villena Last Checkpoint: 2 hours ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Question 2: Single Inheritance, the under_students and grad_students inherit from the parent class (students)

```
In [40]: class students():
    def __init__(self, name, age, stud_id):
        self.name = name
        self.age = age
        self.stud_id = stud_id

    class under_students(students):
        def __init__(self, name, age, stud_id, under_prog):
            self.name = name
            self.age = age
            self.stud_id = stud_id
            self.under_prog = under_prog

    class grad_students(students):
        def __init__(self, name, age, stud_id, grad_prog):
            self.name = name
            self.age = age
            self.stud_id = stud_id
            self.grad_prog = grad_prog

        def get_program_level(self):
            if 'MS' in self.grad_prog:
                return 'Master's Program'
            elif 'PHD' in self.grad_prog:
                return 'Doctorate Program'

undergrad1 = under_students('Ivan', 20, 'BS-0001', 'BS Computer Science')
grad_student1 = grad_students('Navi', 25, 'MS-0001', 'MS Computer Science')
grad_student2 = grad_students('Earvin', 25, 'MS-0001', 'PHD Computer Science')

print(f"Undergraduate Student: {undergrad1.name}, Age: {undergrad1.age}, ID: {undergrad1.stud_id}, Program: {undergrad1.under_prog}")
print(f"Graduate Student: {grad_student1.name}, Age: {grad_student1.age}, ID: {grad_student1.stud_id}, Program: {grad_student1.grad_prog}")
print(f"Graduate Student: {grad_student2.name}, Age: {grad_student2.age}, ID: {grad_student2.stud_id}, Program: {grad_student2.grad_prog}, Level: {grad_student2.get_program_level()}")
```

Undergraduate Student: Ivan, Age: 20, ID: BS-0001, Program: BS Computer Science
Graduate Student: Navi, Age: 25, ID: MS-0001, Program: MS Computer Science
Graduate Student: Earvin, Age: 25, ID: MS-0001, Program: PHD Computer Science, Level: Doctorate Program

Question 3 AND 4:

→ ↻ ⓘ localhost:8888/notebooks/Assignment%203.2/Assignment%203.2_Villena%20.ipynb#

jupyter Assignment 3.2_Villena Last Checkpoint: 2 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Undergraduate Student: Ivari, Age: 20, ID: MS-0001, Program: MS Computer Science
Graduate Student: Navi, Age: 25, ID: MS-0001, Program: MS Computer Science
Graduate Student: Earvin, Age: 25, ID: MS-0001, Program: PHD Computer Science, Level: Doctorate Program

Question 3 and 4: Single Inheritance and polymorphism was demonstrated here by calling the status() on different instances and provides different values for each object because the parameters in the classess are unique for each instance

```
In [41]: class students():
    def __init__(self, name, age, stud_id):
        self.name = name
        self.age = age
        self.stud_id = stud_id

    class grad_students(students):
        def __init__(self, name, age, stud_id, grad_prog, year):
            self.name = name
            self.age = age
            self.stud_id = stud_id
            self.grad_prog = grad_prog
            self.year = year

        def printDetails(self):
            print("Graduate Student:")
            print("self.name, self.grad_prog")

        def status(self):
            print("Graduate Student Status: {} is a {} year {} student".format(self.name, self.year, self.grad_prog))

        def delete_student(self):
            print("\nDeleting {}".format(self.name))
            del self

student1 = grad_students('Tanggol', 29, 'MS-0001', 'MS Computer Science', '1st')
student2 = grad_students('Rigor', 29, 'MS-0002', 'MS Data Science', '2nd')

print("Testing methods for student1:")
student1.status()
student1.delete_student()

print("\nTesting methods for student2:")
student2.status()
student2.delete_student()

Testing methods for student1:
Graduate Student Status: Tanggol is a 1st year MS Computer Science student

Deleting Tanggol

Testing methods for student2:
Graduate Student Status: Rigor is a 2nd year MS Data Science student

Deleting Rigor
```