219116/117 Programming I

Lab 1

Answer the following 3 questions in a file named StudentID_Firstname_lab1_ans.pdf, where StudentID is your KU ID and Firstname is your given name

1. **OOP Review**

Without using computers, analyze the following code and predict the outcome:

```
class Student:
    students = 0 # this is a class attribute
    def __init__(self, name, ta):
        self.name = name # this is an instance attribute
        self.understanding = 0
        Student.students += 1
        print("There are now", Student.students, "students")
        ta.add_student(self)

    def visit_office_hours(self, staff):
        staff.assist(self)
        print("Thanks, " + staff.name)

class Professor:
    def __init__(self, name):
        self.name = name
        self.students = {}

    def add_student(self, student):
        self.students[student.name] = student

    def assist(self, student):
        student.understanding += 1
```

What will the following lines output?

>>> snape = Professor("Snape")

>>> harry = Student("Harry", snape)

Your answer: There are now 1 students

>>> harry.visit_office_hours(snape)

Your answer: Thanks, Snape

>>> harry.visit_office_hours(Professor("Hagrid"))

Your answer: Thanks, Hagrid

>>> harry.understanding

Your answer: 2

>>> for name in snape.students:
>>> print(name)

Your answer: harry

>>> x = Student("Hermione",Professor("McGonagall")).name

Your answer: There are now 2 students

>>> x

Your answer: Hermione

>>> for name in snape.students:
>>> print(name)

Your answer: harry

2. **Inheritance**

Without using computers, analyze the following code and fill in the blanks. Consider the following Dog and Cat classes: s

```
class Dog():
    def __init__(self, name, owner):
        self.is_alive = True
        self.name = name
        self.owner = owner
    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")
    def talk(self):
        print(self.name + " says woof!")

class Cat():
    def __init__(self, name, owner, lives=9):
        self.is_alive = True
        self.name = name
        self.owner = owner
        self.lives = lives
    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")
    def talk(self):
        print(self.name + " says meow!")
```

Notice that because dogs and cats share a lot of similar qualities, there is a lot of repeated code! To avoid redefining attributes and methods for similar classes, we can write a single superclass from which the similar classes inherit. For example, we can write a class called Pet and redefine Dog as a subclass of Pet:

```
class Pet():
    def __init__(self, name, owner):
        self.is_alive = True    # It's alive!!!
        self.name = name
        self.owner = owner
    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")
    def talk(self):
        print(self.name)

class Dog(Pet):
    def talk(self):
        print(self.name + ' says woof!')
```



```
class Cat(Pet):
```

```
def   init  (self, name, owner, lives=9):
    super().__init__(name,owner)

    self.lives = lives
```

```
# fill me in
```

```
def talk(self):
""" Print out a cat's greeting.

"""
# fill me in
    print(self.name + ' says meow! ')
```

```
  def lose_life(self):
   """Decrements a cat's life by 1. When lives reaches zero, 'is_alive'
  becomes False. If this is called after lives has reached zero, print
  out that the cat has no more lives to lose.
   """
```

```
# fill me in


    if self.lives > 0:


        self.lives -= 1


        if self.lives == 0:


            self.is_alive = False
```

```
    else:

        print('The cat has no more lives to lose. ')
```

>>> Cat('Thomas', 'Tammy').talk()

Thomas says meow!


```
class NoisyCat(Cat):
 """A Cat that repeats things twice."""

 def talk(self):
 """Talks twice as much as a regular cat."""   #

fill me in
```

```
        super().talk
        super().talk
```

```
>>> NoisyCat('Magic', 'James').talk()
Magic says meow!
Magic says meow!
```

3. More inheritance

Study the code in the file car.py and predict the outcome of the following code.

```
>>> deneros_car = Car('Tesla', 'Model S')
>>> deneros_car.model
```

Your answer: Model S

```
>>> deneros_car.gas = 10
>>> deneros_car.drive()
```

Your answer: Tesla Model S goes vroom!

```
>>> deneros_car.drive()
```

Your answer: Cannot drive!


```
>>> deneros_car.fill_gas()
```

Your answer: Gas level: 20

```
>>> deneros_car.gas
```

Your answer: 20

>>> Car.gas

Your answer: 30

>>> deneros_car = Car('Tesla', 'ModelS')
>>> deneros_car.wheels = 2
>>> deneros_car.wheels

Your answer: 2

>>> Car.num_wheels

Your answer: 4

>>> deneros_car.drive()

Your answer: Cannot drive!

>>> Car.drive()

Your answer: Error

>>> Car.drive(deneros_car)

Your answer: Cannot drive!

Verify your answer by executing the code with car.py. You can do this with Python interactive mode:

python3 -i car.py

```
>>> deneros_car = Car('Tesla', 'Model S’)
>>> deneros_car.model
```

...

Complete the following coding questions. Use the code skeleton provided with this lab.

4. **mint.py**

Complete the Mint and Coin classes so that the coins created by a mint have the correct year and worth.

- Each Mint instance has a year stamp. The update method sets the year stamp to the current_year class attribute of the Mint class.
- The create method takes a subclass of Coin and returns an instance of that class stamped with the mint's year (which may be different from Mint.current_year if it has not been updated.) • A Coin's worth method returns the cents value of the coin plus one extra cent for each year of age beyond 50. A coin's age can be determined by subtracting the coin's year from the current_year class attribute of the Mint class.

The expected outcome is given in the Doctest for Mint class

Submission:

- ***Create StudentID_Firstname_lab1 folder, where StudentID is your KU ID and Firstname is your given name***
- ***Put the files to submit, StudentID_Firstname_lab1_ans.pdf and mint.py, into this folder***
- ***Zip the folder and submit the zip file to the course's Google Classroom before the due date***