OOP AND INHERITANCE

COMPUTER SCIENCE MENTORS CS 88

March 29th to April 2nd

1 Object Oriented Programming

1. What is a class?

   Solution: A class is a mechanism used to create new user-defined data structures. It contains data as well as the methods used to process that data.

2. What is an instance of a class?

   Solution: An instance is an instantiation of the class with actual values, literally an object of a specific class.

3. What is the purpose of the __init__ method?

   Solution: The __init__ method initializes an instance of a class.

4. What is self?

   Solution: In the __init__ method, self refers to the newly created object; in other class methods, it refers to the instance whose method was called. It's a pointer to the instance of the object.

5. What would Python display? Write the result of executing the following code and prompts. If nothing would happen, write "Nothing". If an error occurs, write "Error".

```
class Jedi:
    lightsaber = "blue"
    force = 25
    def __init__(self, name):
        self.name = name
    def train(self, other):
        other.force += self.force / 5
    def __repr__(self):
        return "Jedi " + self.name
```

```
>>> anakin = Jedi("Anakin")
>>> anakin.lightsaber, anakin.force
```

Solution: ("blue", 25)

```
>>> anakin.lightsaber = "red"
>>> anakin.lightsaber
```

Solution: "red"

```
>>> Jedi.lightsaber
```

Solution: "blue"

```
>>> obiwan = Jedi("Obi-wan")
>>> anakin.master = obiwan
>>> anakin.master
```

Solution: Jedi Obi-wan

```
>>> Jedi.master
```

Solution: Error

```
>>> obiwan.force += anakin.force
>>> obiwan.force, anakin.force
```

Solution: (50, 25)

```
>>> obiwan.train(anakin)
>>> obiwan.force, anakin.force
```

Solution: (50, 35)

```
>>> Jedi.train(obiwan, anakin)
>>> obiwan.force, anakin.force
```

Solution: (50, 45)

6. We now want to write three different classes, `Postman`, `Client`, and `Email` to simulate email. Fill in the definitions below to finish the implementation!

```
>>> postman = Postman() #Create a new Postman

>>> john = Client(postman, "John") #Create client named John

>>> rohan = Client(postman, "Rohan") #Create client named
   Rohan

>>> john.compose("POG", "Rohan") #John sends an email to Rohan

>>> rohan.compose("CHAMP", "John") #Rohan sends an email to
   John

>>> rohan.inbox[0].msg #Rohan's inbox
"POG"

>>>john.inbox[0].msg #John's inbox
"CHAMP"
```

```
class Email:
    """Every email object has 3 instance attributes: the
       message,
    the sender (their name), and the addressee (the
       destination's
    name).
    """
    def __init__(self, msg, sender, addressee):
```

Solution:

```
        self.msg = msg
        self.sender = sender
        self.addressee = addressee
```

```
class Postman:
    """Each Postman has an instance attribute clients, which
       is a
    dictionary that associates client names with client
       objects.
    """
    def __init__(self):
        self.clients = {}

    def send(self, email):
        """Take an email and put it in the inbox of the client
            it
        is addressed to."""
```

Solution:

```
        client = self.clients[email.addressee]
        client.receive(email)
```

```
    def register_client(self, client, client_name):
        """Takes a client object and client_name and adds it
           to the
        clients instance attribute.
        """
```

Solution:

```
        self.clients[client_name] = client
```

```
class Client:
    """Every Client has instance attributes name (which is
       used
    for addressing emails to the client), mailman (which is
    used to send emails out to other clients), and inbox (a
    list of all emails the client has received).
    """
    def __init__(self, mailman, name):
        self.inbox = []
```

Solution:

```
        self.mailman = mailman
        self.name = name
        self.mailman.register_client(self, self.name)
```

```
    def compose(self, msg, recipient):
        """Send an email with the given message msg to the
           given
        recipient."""
```

Solution:

```
        email = Email(msg, self.name, recipient)
        self.mailman.send(email)
```

```
    def receive(self, email):
        """Take an email and add it to the inbox of this
           client.
        """
```

Solution:

```
        self.inbox.append(email)
```

7. Fill in the classes `Emotion`, `Joy`, and `Sadness` below so that you get the following output from the Python interpreter.

```
>>> Emotion.num
0

>>> joy = Joy()
>>> sadness = Sadness()
>>> emotion = Emotion()
>>> Emotion.num # number of Emotion instances created
3

>>> joy.power
5

>>> joy.catchphrase() # Print Joy's catchphrase
Think positive thoughts

>>> sadness.catchphrase() #Print Sad's catchphrase
I'm positive you will get lost

>>> sadness.power
5

>>> emotion.catchphrase()
I'm just an emotion.

>>> joy.feeling(sadness) # print "Together" if same power
Together

>>> sadness.feeling(joy)
Together

>>> joy.power = 7
>>> joy.feeling(sadness) # Print the catchphrase of the more
   powerful feeling before the less powerful feeling
Think positive thoughts
I'm positive you will get lost

>>> sadness.feeling(joy)
Think positive thoughts
I'm positive you will get lost
```

```
class Emotion
```

Solution:

```
class Emotion:
    num = 0
```

```
    def __init__(self):
```

Solution:

```
        self.power = 5
        Emotion.num += 1
```

```
    def feeling(self, other):
```

Solution:

```
        if self.power > other.power:
            self.catchphrase()
            other.catchphrase()
        elif other.power > self.power:
            other.catchphrase()
            self.catchphrase()
        else:
            print("Together")
```

```
    def catchphrase(self):
```

Solution:

```
        print("I'm just an emotion.")
```

```
class Joy
```

Solution:

```
class Joy(Emotion):
```

```
    def catchphrase(self):
```

Solution:

```
        print("Think positive thoughts!")
```

```
class Sadness
```

Solution:

```
class Sadness(Emotion):
```

```
    def catchphrase(self):
```

Solution:

```
        print("I'm positive you will get lost.")
```