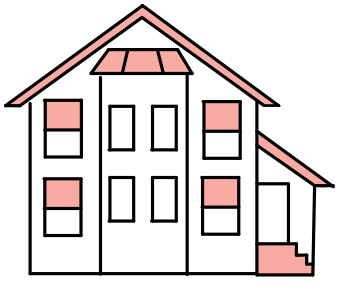
OOP! : object-oriented programming :

some Definitions:

class a template for creating objects

↳ 

instance a single actualization of a class

↳  

instance attribute a property specific to an instance

↳ accent colors

class attribute a property shared by all instances

↳ number of windows

method a function that all instances may perform

↳ turn on the lights

```
class House():
    windows = 8
    floors = 2

    def __init__(self, owner, color):
        self.owner = owner
        self.color = color
        self.lit = False

    def check_lights(self):
        if self.lit:
            print("lights on")
        else:
            print("lights off")

    def light_switch(self):
        self.lit = not self.lit
```

class attributes

instance attributes

methods

```
class House():
    windows = 8
    floors = 2

    def __init__(self, owner, color):
        self.owner = owner
        self.color = color
        self.lit = False

    def check_lights(self):
        if self.lit:
            print("lights on")
        else:
            print("lights off")

    def light_switch(self):
        self.lit = not self.lit
```

constructor

__init__

House("Jessica", "red")

↓

House instance

owner: "Jessica"
color: "red"
lit: False

Practice: What would Python do?

```
>>> jess_house = House("Jessica", "red")
>>> jess_house.owner
```

"Jessica"

```
>>> jess_house.check_lights()


>>> House.check_lights(jess_house)


>>> dan_house = House("Daniel", "yellow")
>>> dan_house.owner


>>> dan_house.check_lights()


>>> jess_house.light_switch()
>>> jess_house.check_lights()


>>> dan_house.check_lights()


>>> jess_house.windows


>>> House.windows = 9
>>> jess_house.windows


>>> dan_house.windows
```

TIPS:

- draw an environment diagram & update it as you go along

OOP AND INHERITANCE

COMPUTER SCIENCE MENTORS CS 88

March 29th to April 2nd

1 Object Oriented Programming

1. What is a class?

template to create our own data structure

2. What is an instance of a class?

a single thing / object of a class

3. What is the purpose of the __init__ method?

defines / creates a specific instance

4. What is self?

refers to our current instance that we're calling a method on

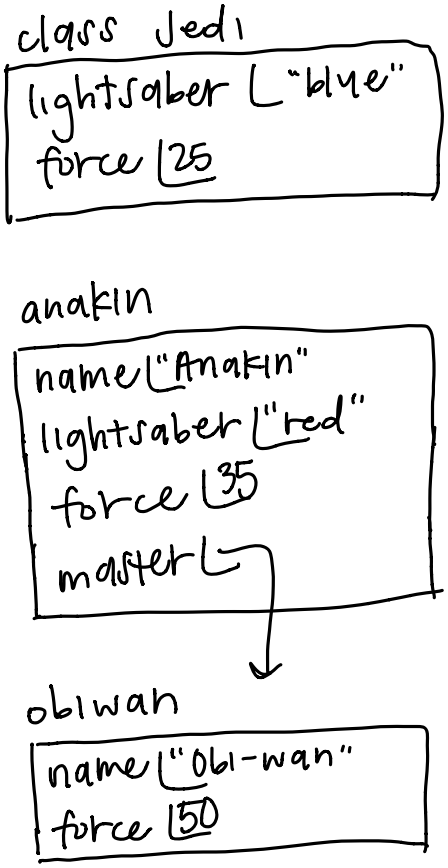
list a = [1, 2]

a ⟶ [1|2]

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
    ("blue", 25)
>>> anakin.lightsaber = "red"
>>> anakin.lightsaber
    "red"
>>> Jedi.lightsaber
    "blue"
>>> obiwan = Jedi("Obi-wan")
>>> anakin.master = obiwan
>>> anakin.master
    Jedi Obi-wan
>>> Jedi.master
    Error
>>> obiwan.force += anakin.force
>>> obiwan.force, anakin.force
    (50, 25)
>>> obiwan.train(anakin)
>>> obiwan.force, anakin.force
    (50, 35)
>>> Jedi.train(obiwan, anakin)
>>> obiwan.force, anakin.force
    (50, 45)
```



obiwan.force = obiwan.force + anakin.force

25 25

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):
        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."""
        c_name = email.addressee
        c_obj = self.clients[c_name]
        c_obj.receive(email)
    def register_client(self, client, client_name):
        """Takes a client object and client_name and adds it
           to the
        clients instance attribute.
        """
        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 = []
        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."""
        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.
        """
        self.inbox += [email]

                or
        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

    def __init__(self):



    def feeling(self, other):









    def catchphrase(self):


class Joy
    def catchphrase(self):
```

```
class Sadness
    def catchphrase(self):
```

SOLUTIONS

OOP AND INHERITANCE

COMPUTER SCIENCE MENTORS CS 88

March 29th to April 2nd

1 Object Oriented Programming

1. What is a class?

   mechanism (blueprint) to create user defined data structures

2. What is an instance of a class?

   an object of a specific class (house)

3. What is the purpose of the __init__ method?

   create a new object

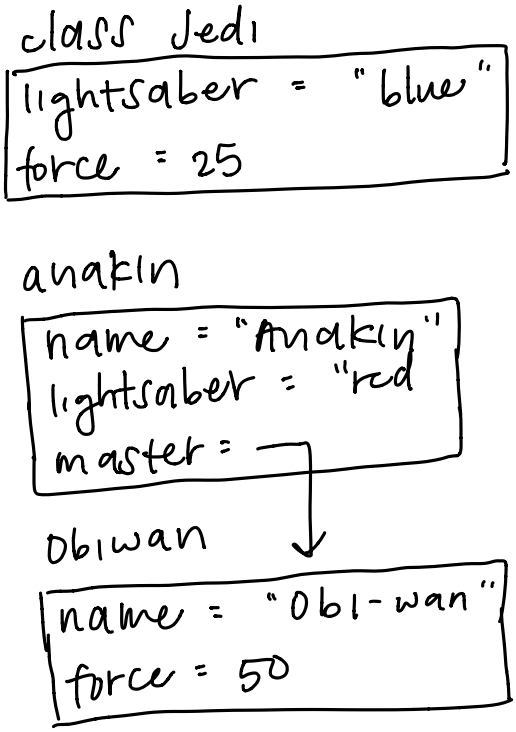
4. What is self?

   refers to the object whose method is being called

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
    "blue", 25
>>> anakin.lightsaber = "red"
>>> anakin.lightsaber
    "red"
>>> Jedi.lightsaber
    "blue"
>>> obiwan = Jedi("Obi-wan")
>>> anakin.master = obiwan
>>> anakin.master
    Jedi Obi-wan
>>> Jedi.master
    Error
>>> obiwan.force += anakin.force
>>> obiwan.force, anakin.force
    50, 25
>>> obiwan.train(anakin)
>>> obiwan.force, anakin.force
    50, 35
>>> Jedi.train(obiwan, anakin)
>>> obiwan.force, anakin.force
    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):
        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."""
        client_obj = self.clients[email.addressee]
        client_obj.receive(email)

    def register_client(self, client, client_name):
        """Takes a client object and client_name and adds it
           to the
        clients instance attribute.
        """
        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 = []
        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."""
        em = new Email(msg, self, recipient)
        self.mailman.send(em)

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

WILL BE WORKING ON THIS PROBLEM NEXT WEEK 4/9 SO DON'T SCROLL IF YOU DON'T WANT TO SPOIL THE SOLUTION

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:
    num = 0

    def __init__(self):
        self.power = 5


    def feeling(self, other):
        if (self.power > other.power):
            self.catchphrase();
            other.catchphrase();
        elif (other.power > self.power):
            other.catchphrase();
            self.catchphrase();
        else:
            print("Together")
    def catchphrase(self):
        print("I'm just an emotion.")

class Joy(Emotion):
    def catchphrase(self):
        print("Think positive thoughts!")
```

```
class Sadness (Emotion):
    def catchphrase(self):
        print("I'm positive you will get lost.")
```