

CSCI 127: Joy and Beauty of Data

Lecture 13: OOP

Reese Pearsall
Summer 2021

<https://reese.github.io/classes/summer2021/127/main.html>

Announcements

Program 3 due **tonight** @ 11:59 PM

Lab 7 (Dictionaries) due **tomorrow** @ 11:59 PM

Lab 8 due on **Thursday** @ 11:59 P.M.

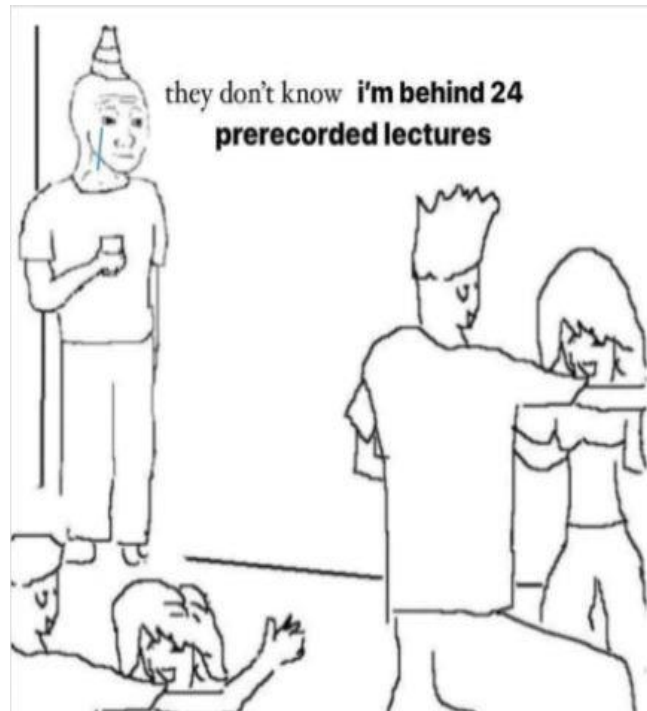
- After today, you will be able to finish it

Program 4 due on **Sunday** 6/13 @ 11:59 PM

- After today, you will be able to finish it

Everyone is eligible for full access to PyCharm!

FYI: You can use late passes on any remaining lab/program



When I meet my instructor on campus and they don't speak on 2X speed



If you have not signed up for a 1
on 1 meeting time with me yet,
**make sure to do that sometime
this week**

*Me if I have to take off 5% of your final
because you never signed up for a
time to meet with me*



Object Oriented Programming

So far, we have used **procedural programming** to solve problems. We have written **functions** that do things

Now, we will talk about a different way to solve problems...

Object Oriented Programming (OOP) is a paradigm of solving problems using objects, which represent *something*

The objects we create usually have data (**states/attributes**) and behaviors (**methods**)

Object Oriented Programming Example

There are many different kinds of cars...



Object Oriented Programming Example

There are many different kinds of cars...

However, all cars share similar features



Object Oriented Programming Example

There are many different kinds of cars...

However, all cars share similar features

All cars have:

- A color
- Wheels
- Engine
- Windshield
- Windows
- Seating
- Lights



Object Oriented Programming Example

There are many different kinds of cars...

However, all cars share similar features

All cars have:

- A color
- Wheels
- Engine
- Windshield
- Windows
- Seating
- Lights

All cars can:

- Accelerate
- Slow down
- Stop
- Turn



Object Oriented Programming Example

There are many different kinds of cars...

However, all cars share similar features

All cars have:

- A color
- Wheels
- Engine
- Windshield
- Windows
- Seating
- Lights

Attributes

All cars can:

- Accelerate
- Slow down
- Stop
- Turn

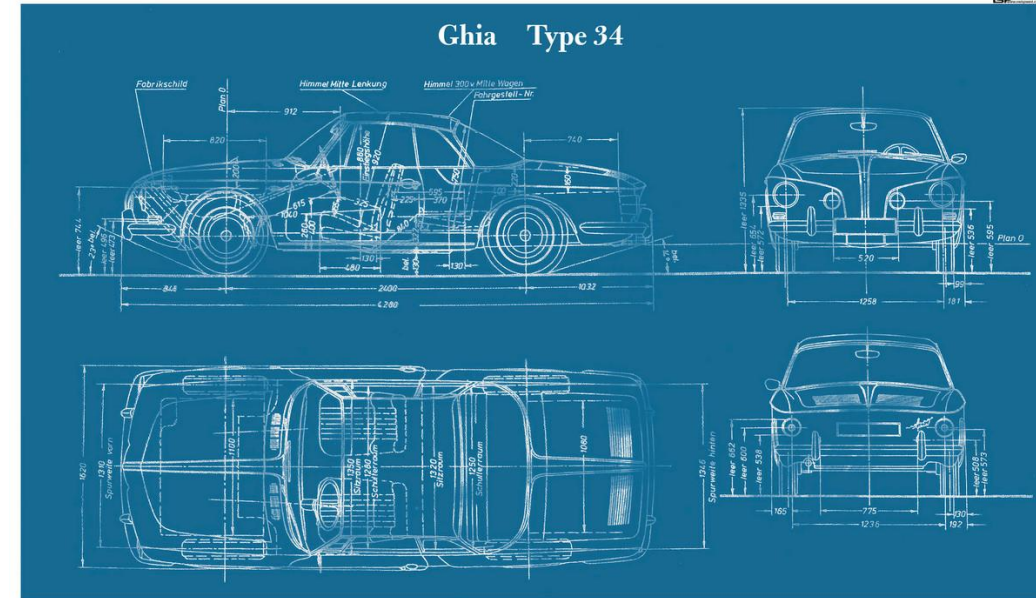
Functionality/Behavior



If we can create a **blueprint** for a generic car, then we can use that blueprint to create many different cars

When we create a car using that blue print, we can specify the different **attributes** (color, # of seats, speed, etc)

When we create a car, we give the car access to different kinds of **behavior** (accelerating, stopping, turning, etc)

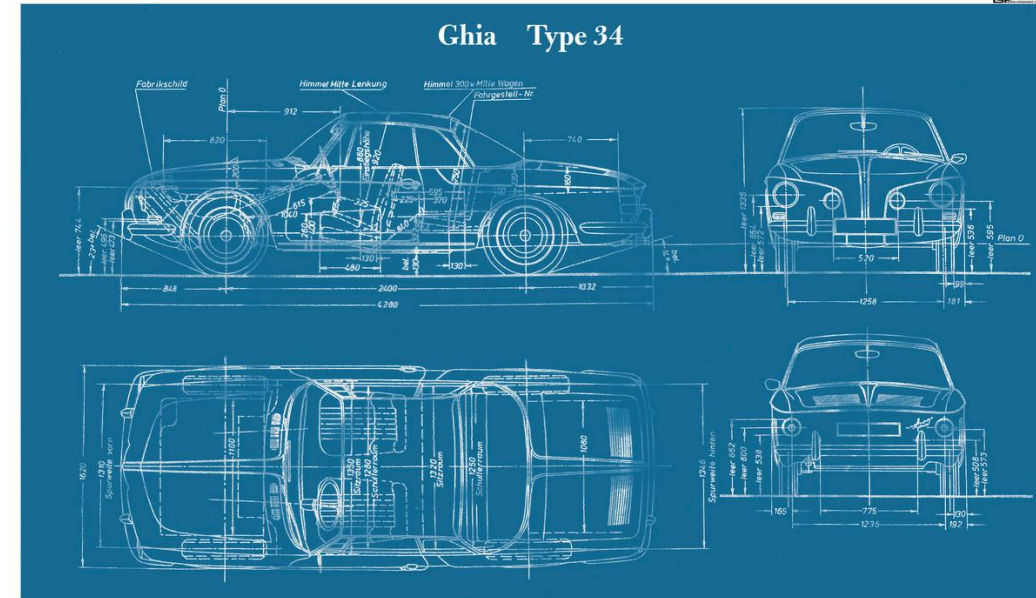


Object Oriented Programming Example

If we can create a **class** ~~blueprint~~ for a generic car, then we can use that ~~blueprint~~ **class** to create many different cars

When we create a car using that blue print, we can specify the different **attributes** (color, # of seats, speed, etc)

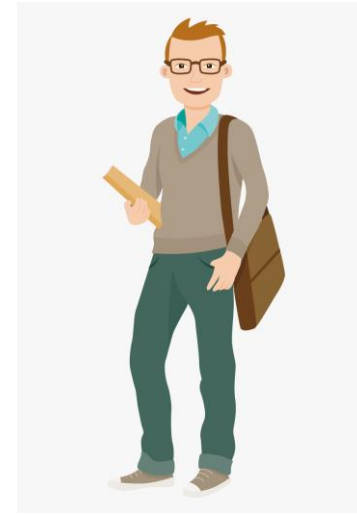
When we create a car, we give the car access to different kinds of **behavior** (accelerating, stopping, turning, etc)



Student Example

Consider a college student at MSU...

What sort of attributes may a college student have?



Student Example

Consider a college student at MSU...

What sort of attributes may a college student have?

- Name
- Major
- GPA
- Student ID Number
- Year (freshman, sophomore, junior, senior)

And much more



Student Example

Consider a college student at MSU...

What sort of attributes may a college student have?

- Name
- Major
- GPA
- Student ID Number
- Year (freshman, sophomore, junior, senior)

And much more

Lets create our blueprint!



OOP in Python

Define classes using the **class** keyword

- All class names should be capitalized

All classes need a constructor. A constructor is the method that will create the object

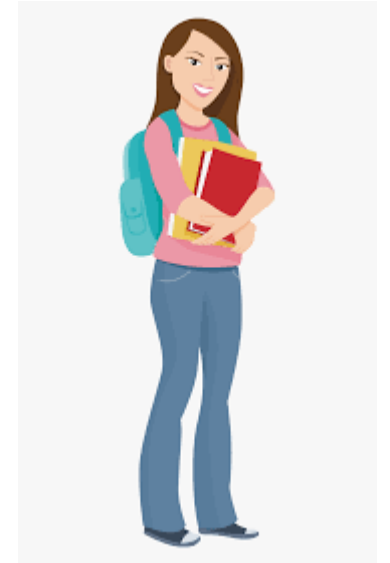
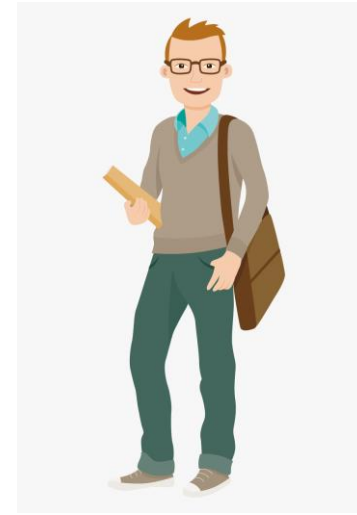
- Constructor will **always** be:

```
def __init__(<insert parameters here>):
```

All methods need to go inside of the class

Reader methods: getName(), getMajor(), etc

Writer methods: setName(), setMajor(), etc



OOP Review

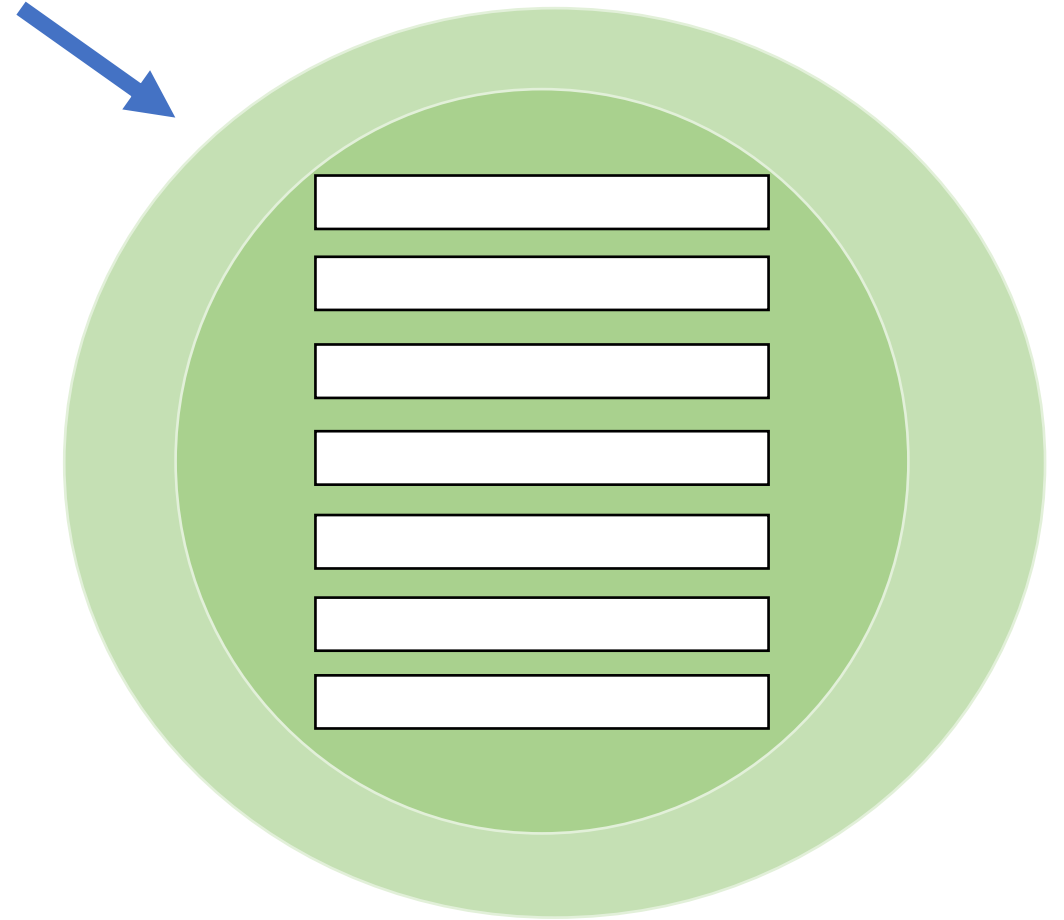
student1

We create and use objects using **classes**

```
student1 = Student("James","Computer Science","04293401",4.0,"Junior")
```

We start off in our **constructor**

```
def __init__(self,name,major,student_id,gpa="Undefined",year="Freshman"):  
    self.name = name  
    self.major = major  
    self.student_id = student_id  
    self.gpa = gpa  
    self.year = year  
    self.champ_change = 0  
    self.minor = "N/A"
```



OOP Review

student1

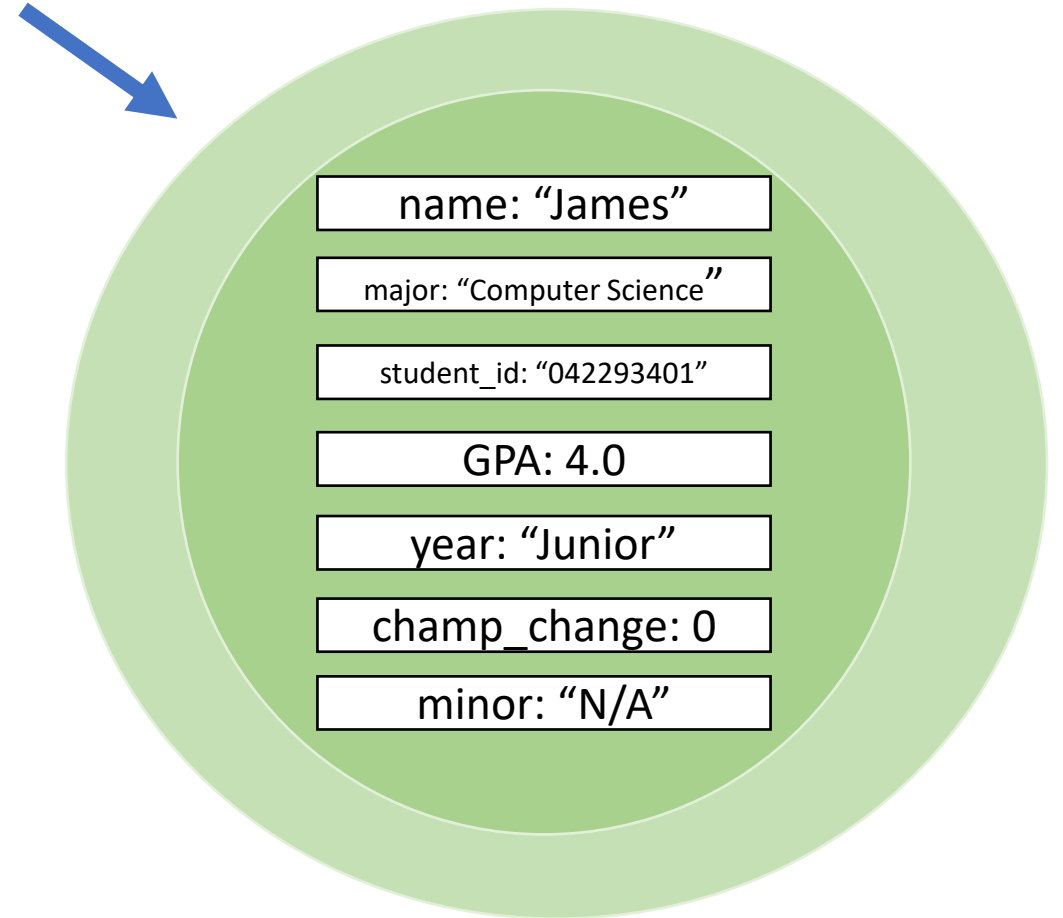
Student object

We create and use objects using **classes**

```
student1 = Student("James","Computer Science","04293401",4.0,"Junior")
```

We start off in our **constructor**

```
def __init__(self,name,major,student_id,gpa="Undefined",year="Freshman"):  
    self.name = name  
    self.major = major  
    self.student_id = student_id  
    self.gpa = gpa  
    self.year = year  
    self.champ_change = 0  
    self.minor = "N/A"
```



OOP Review

We create and use objects using **classes**

```
student1 = Student("James", "Computer Science", "04293401", 4.0, "Junior")
```

We start off in our **constructor**

```
def __init__(self, name, major, student_id, gpa="Undefined", year="Freshman"):  
    self.name = name  
    self.major = major  
    self.student_id = student_id  
    self.gpa = gpa  
    self.year = year  
    self.champ_change = 0  
    self.minor = "N/A"
```

```
print(student1)
```



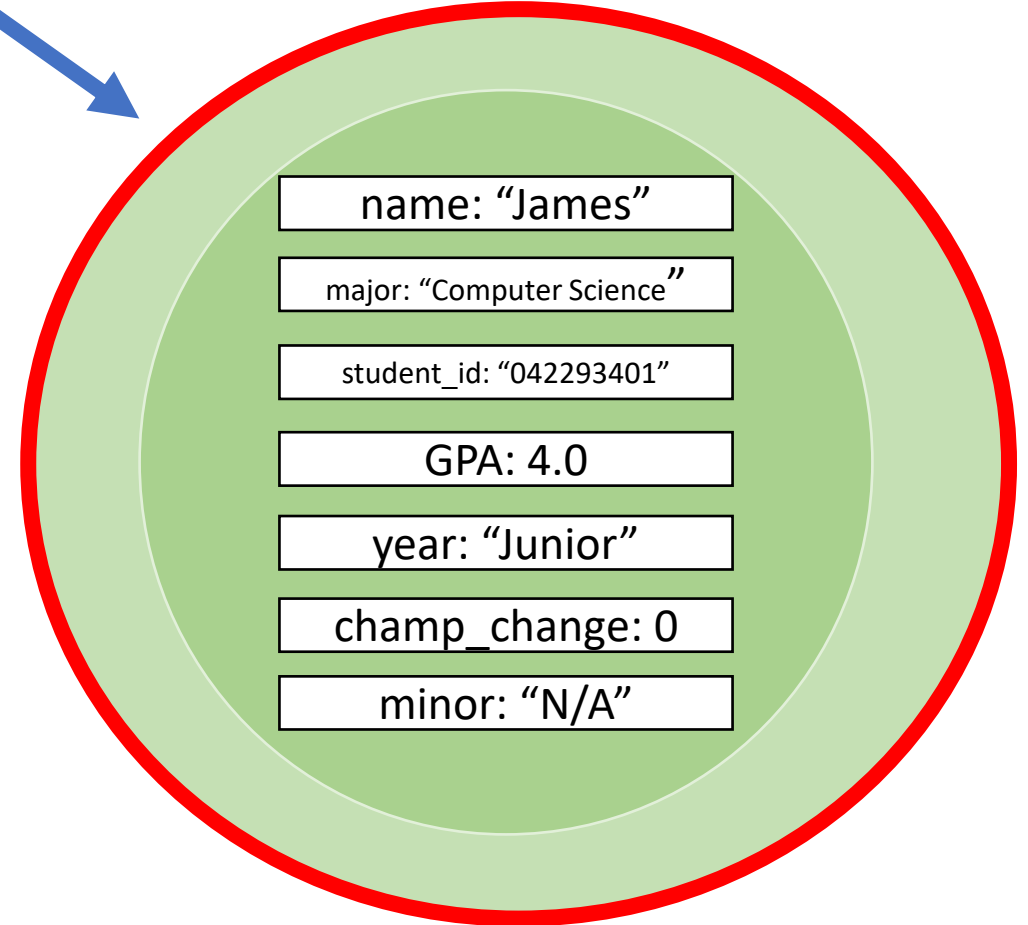
<__main__.Student object at 0x03242D78>

Object's Location in Memory

student1



Student object



OOP Review

We create and use objects using **classes**

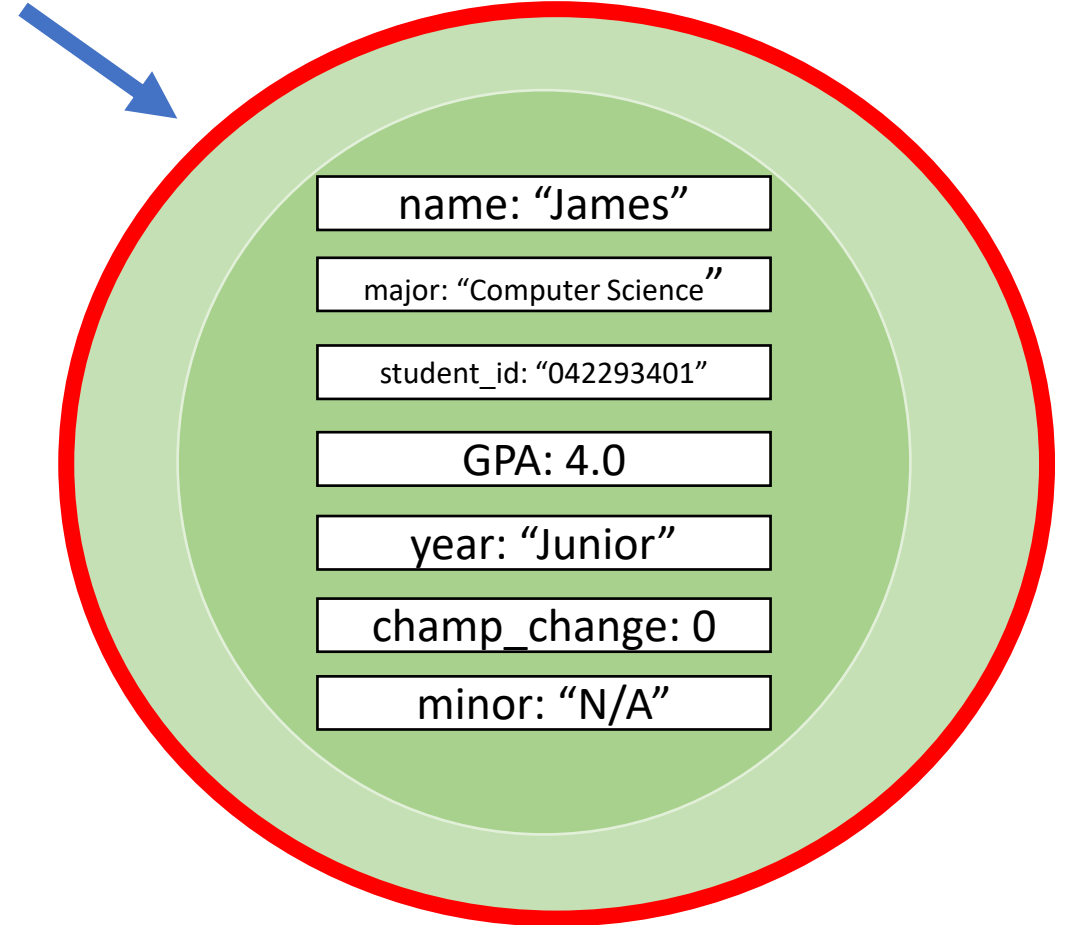
```
student1 = Student("James", "Computer Science", "04293401", 4.0, "Junior")
```

We start off in our **constructor**

```
def __init__(self, name, major, student_id, gpa="Undefined", year="Freshman"):  
    self.name = name  
    self.major = major  
    self.student_id = student_id  
    self.gpa = gpa  
    self.year = year  
    self.champ_change = 0  
    self.minor = "N/A"
```

student1

Student object



Solution:

Overwrite what gets printed out using the `__str__` method

```
print(student1)
```



```
<__main__.Student object at 0x03242D78>
```

Object's Location in Memory

OOP Review

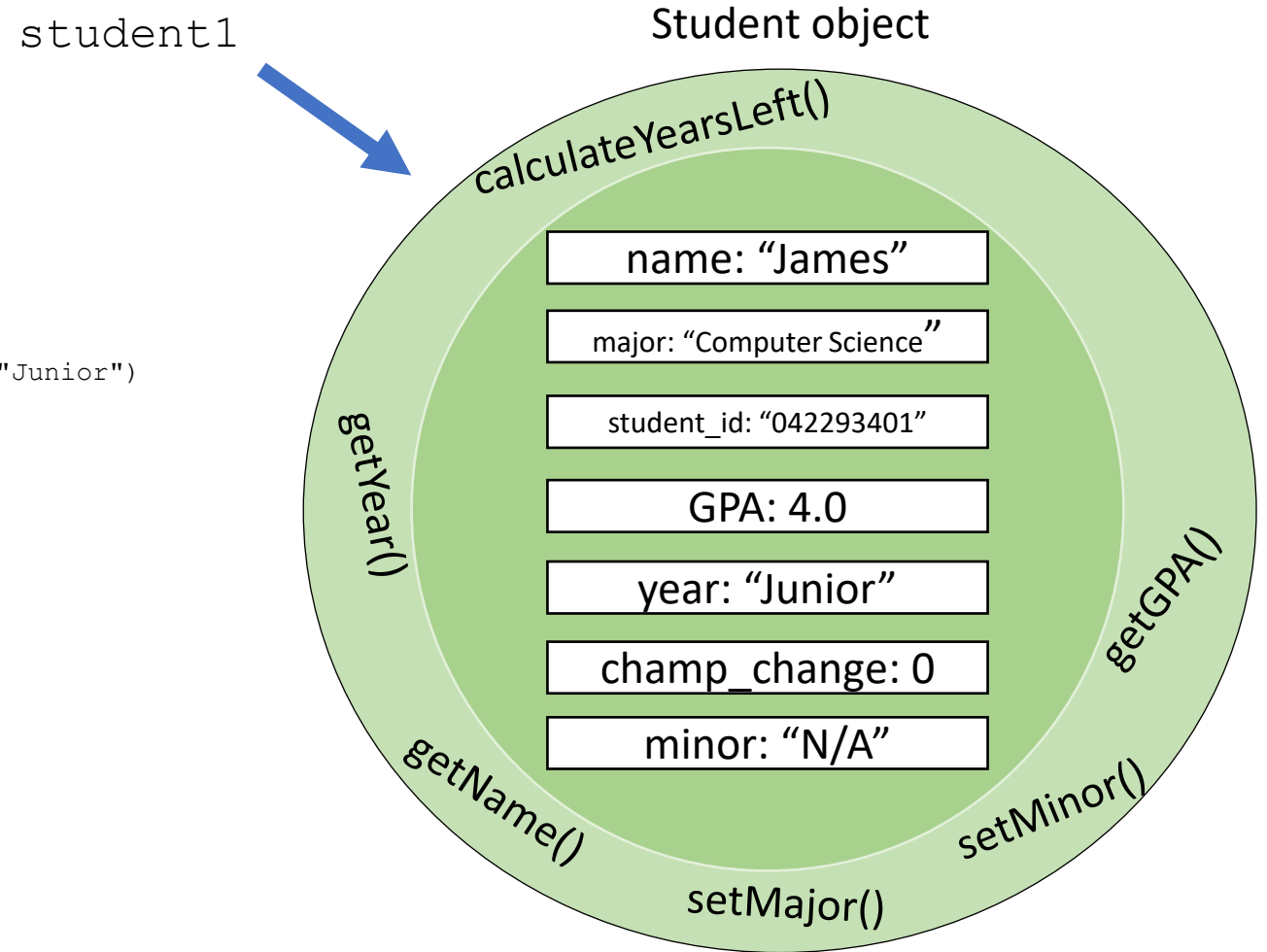
We create and use objects using **classes**

```
student1 = Student("James", "Computer Science", "04293401", 4.0, "Junior")
```

We start off in our **constructor**

Our objects also have functionality (**methods**)

```
print(student1.getName())
```



OOP Review

We create and use objects using **classes**

```
student1 = Student("James", "Computer Science", "04293401", 4.0, "Junior")
```

We start off in our **constructor**

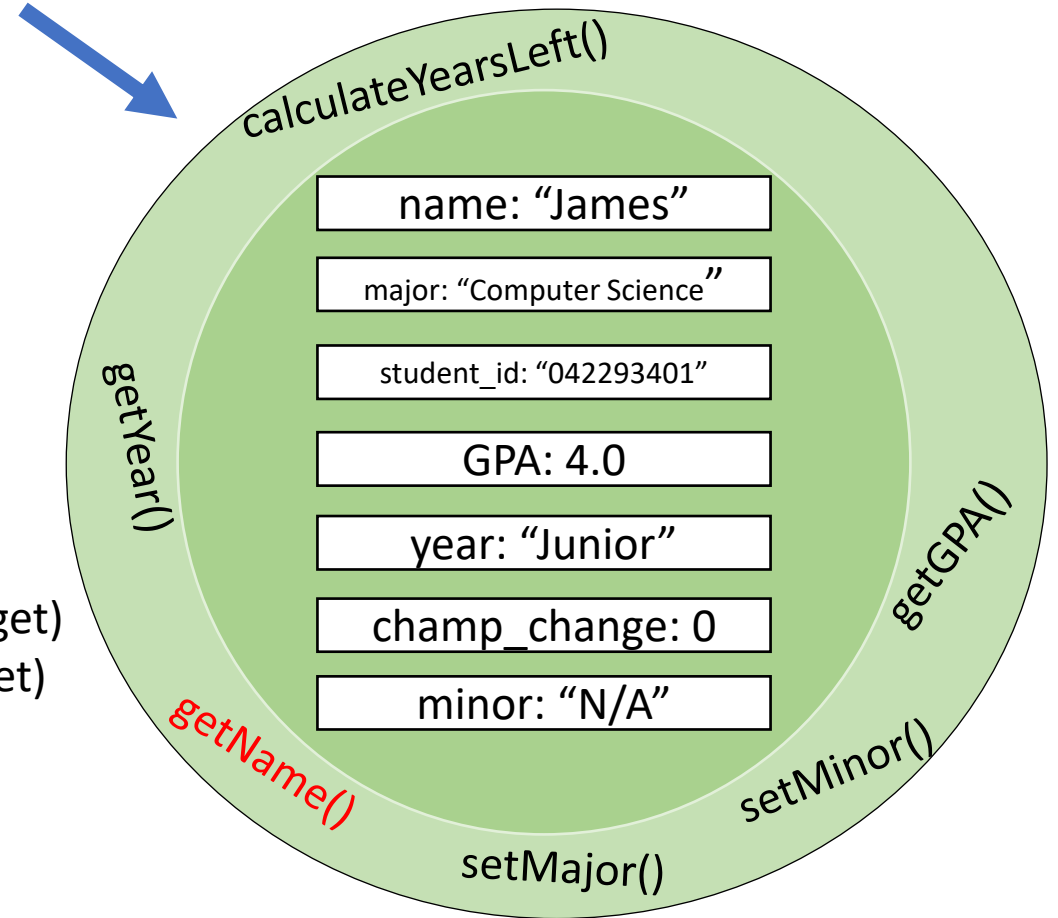
Our objects also have functionality (**methods**) Reader Methods (get)
Writer Methods (set)

```
def getName(self):  
    return self.name
```

```
print(student1.getName())
```

student1

Student object



OOP Review

We create and use objects using **classes**

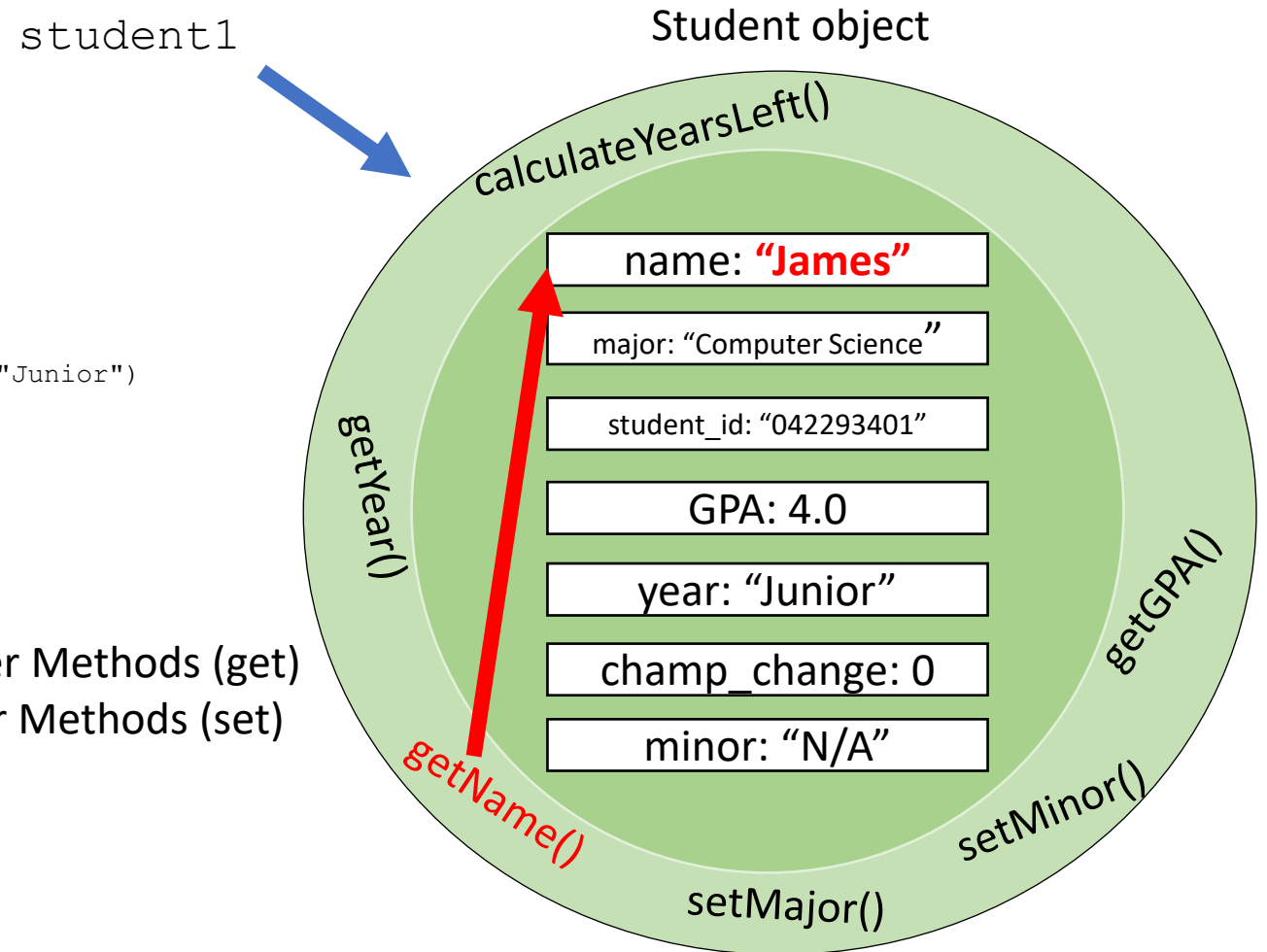
```
student1 = Student("James", "Computer Science", "04293401", 4.0, "Junior")
```

We start off in our **constructor**

Our objects also have functionality (**methods**) Reader Methods (get)
Writer Methods (set)

```
def getName(self):  
    return self.name
```

```
print(student1.getName())
```



OOP Review

We create and use objects using **classes**

```
student1 = Student("James","Computer Science","04293401",4.0,"Junior")
```

We start off in our **constructor**

Our objects also have functionality (**methods**) Reader Methods (get)
Writer Methods (set)

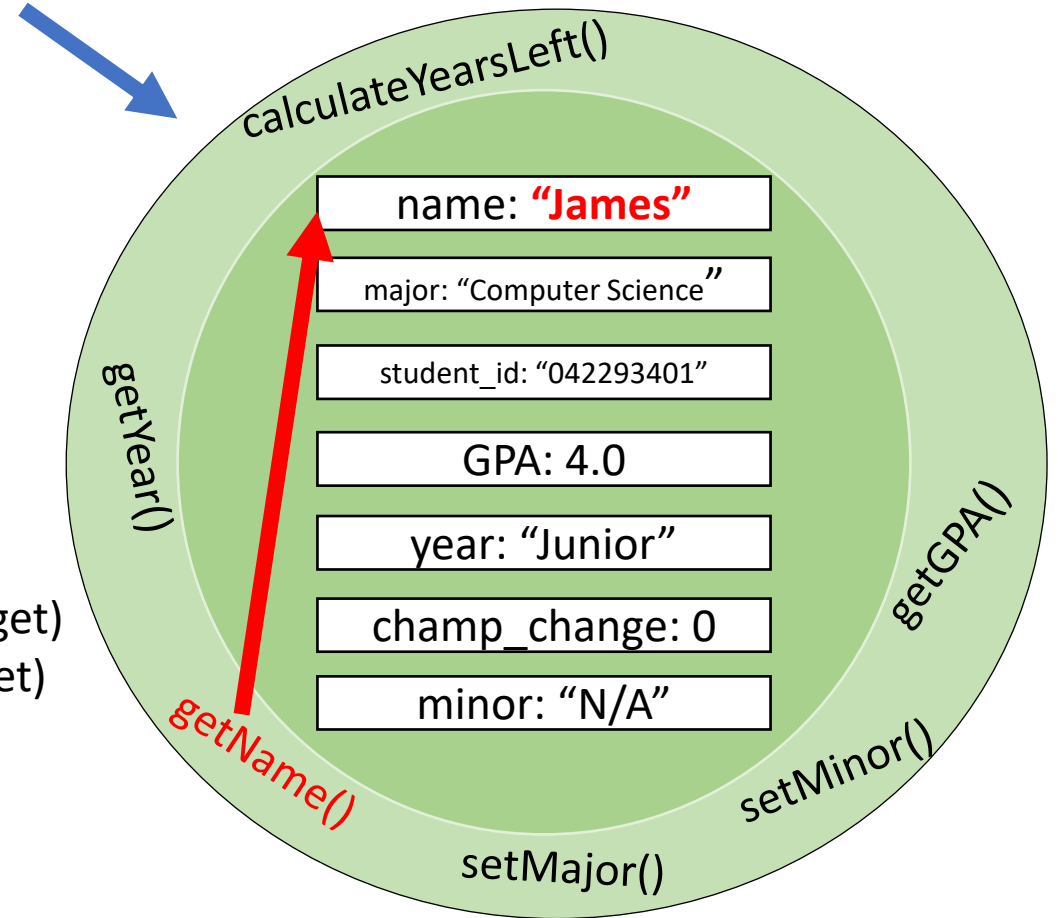
```
def getName(self):  
    return self.name
```

```
print(student1.getName())
```

 → James

student1

Student object



OOP Review

We create and use objects using **classes**

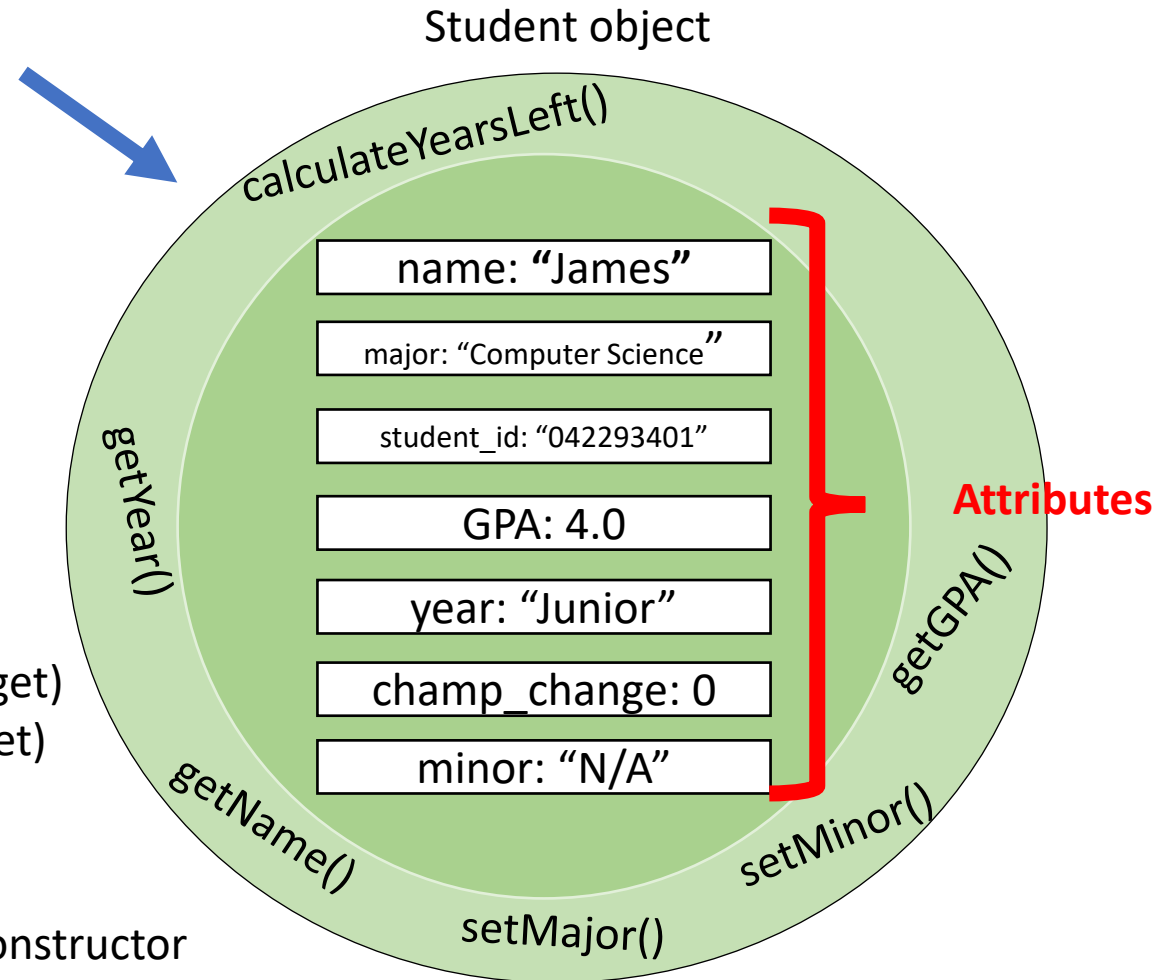
```
student1 = Student("James", "Computer Science", "04293401", 4.0, "Junior")
```

We start off in our **constructor**

Our objects also have functionality (**methods**)
Reader Methods (get)
Writer Methods (set)

We can find the attributes/states of the object by looking at the constructor

student1



OOP Review

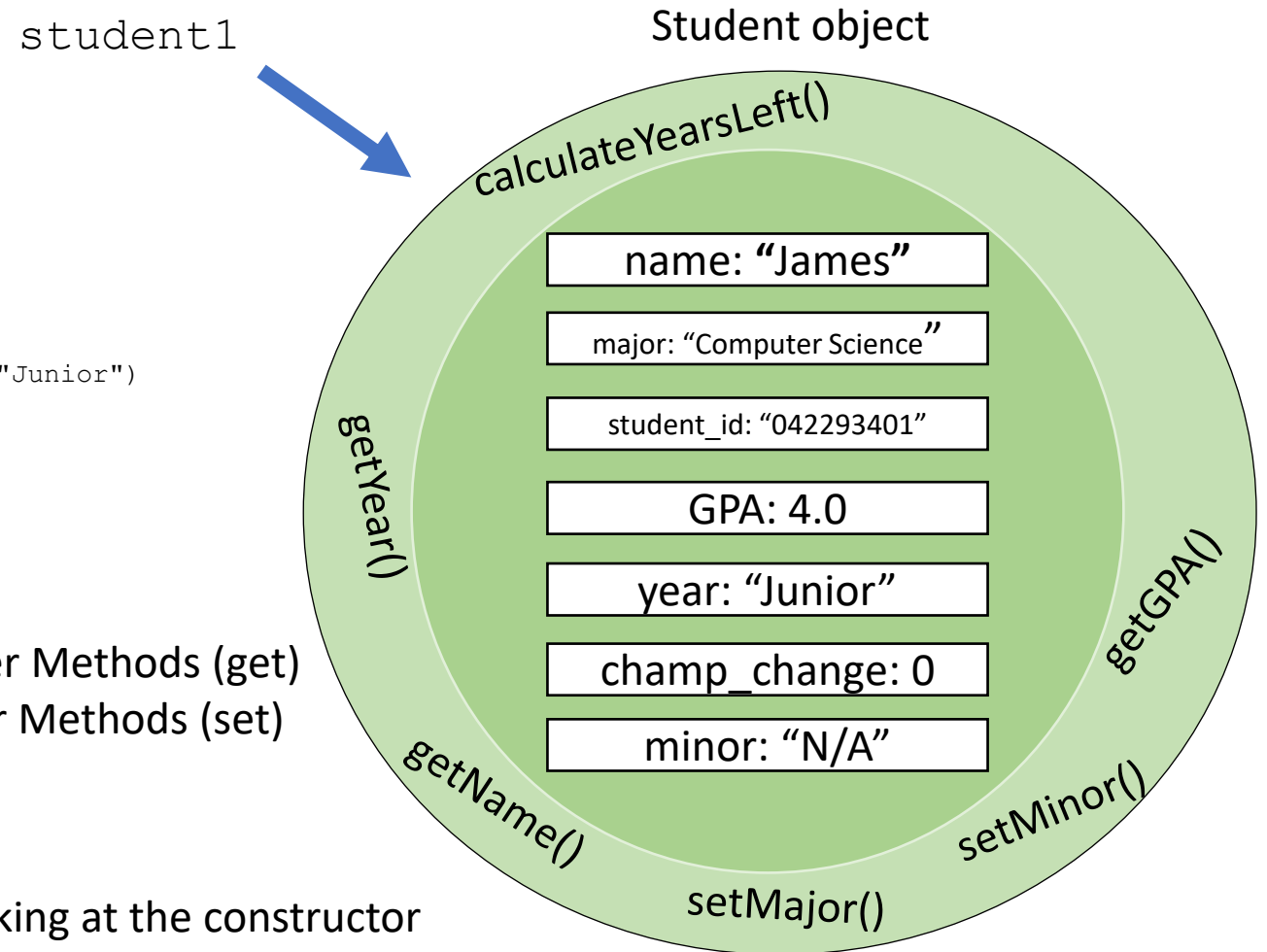
We create and use objects using **classes**

```
student1 = Student("James", "Computer Science", "04293401", 4.0, "Junior")
```

We start off in our **constructor**

Our objects also have functionality (**methods**)
Reader Methods (get)
Writer Methods (set)

We can find the attributes/states of the object by looking at the constructor



Announcements (Tuesday)

Lab 7 due **tonight** (Tuesday 11:59 PM)

Lab 8 due **Thursday** (Tuesday 11:59 PM)

Program 4 due **Sunday** @ 11:59 PM

Today:
More OOP

When you're the number 1 student in the class but your Python Professor says only the top student in the class gets an A



meme made by reese

OOP Example

Construct a **quarterback** class. Each quarterback will have:

- Name
- Attempts
- Completions
- Passing Yards
- Touchdowns
- Interceptions

The class should be able to calculate the **completion percentage**, **passing yards per attempt**, and **quarterback passer rating**

There should also be all necessary getter/setter methods

$$a = \left(\frac{\text{COMP}}{\text{ATT}} - .3 \right) \times 5$$

$$b = \left(\frac{\text{YDS}}{\text{ATT}} - 3 \right) \times 0.25$$

$$c = \left(\frac{\text{TD}}{\text{ATT}} \right) \times 20$$

$$d = 2.375 - \left(\frac{\text{INT}}{\text{ATT}} \times 25 \right)$$

$$\text{Passer Rating} = \left(\frac{a + b + c + d}{6} \right) \times 100$$

OOP Example

Let's create a Python class using `billionaires.csv` that is going to represent information about Billionaires

Each Billionaire has a

Name
Company Name
Age
Gender
Worth in Billions
Location (Continent)

Lets write some functions that can

- Search for billionaires that make more money than a certain threshold
- Print out # of male vs female billionaires
- Print out number of Billionaires based on Continent

This example will be helpful for program 4

Announcements

Lab 8 due **Thursday**

Program 4 due **Sunday**

Cutting a few lectures today and tomorrow to give you time to catch up 😊

Today:

Inheritance, Magic Methods



PYTHON OBJECT
ORIENTED
PROGRAMMING



POOP

Important Announcement

Course evaluations are **OPEN**

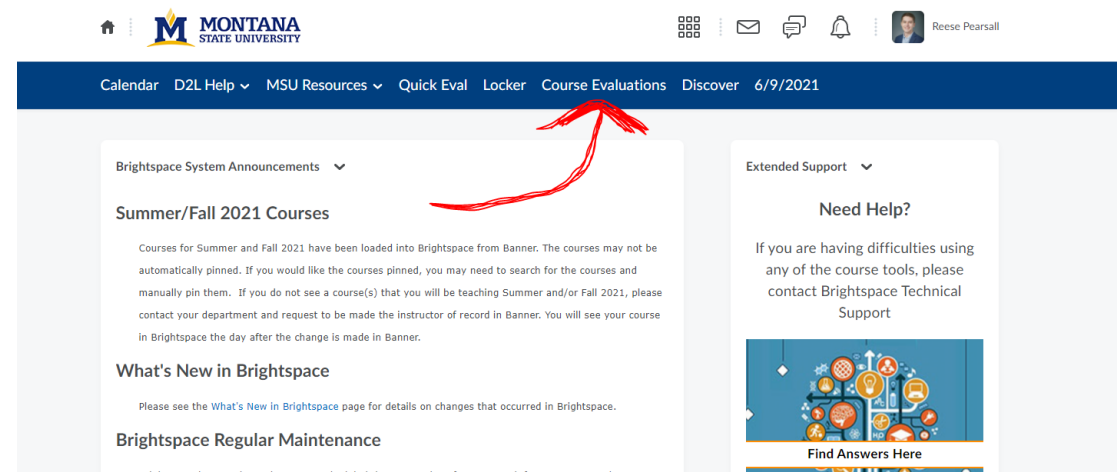
Because I am early in my teaching career,
your feedback is *very* important to me

Your feedback will help make this
class better for future students

Your feedback will also be read by the
CS department head, who is in charge
of hiring instructors 😊

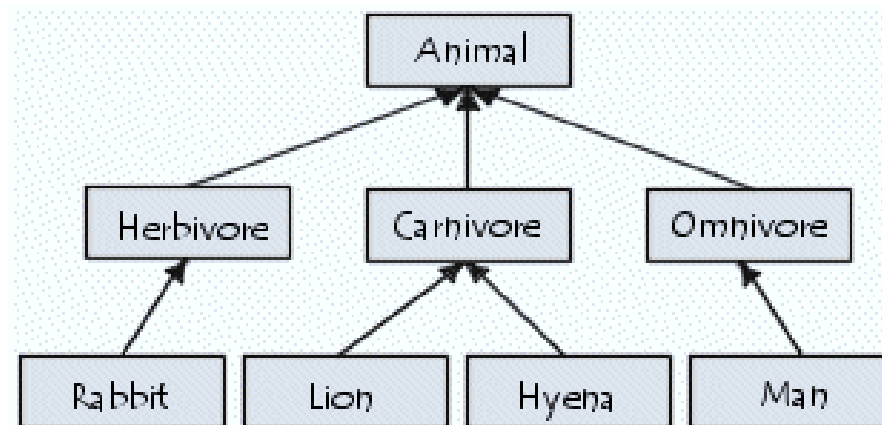
If you submit a screenshot showing you completed the
course evaluation, I will add 2% to your final exam grade

**If 90% of the class or more fills out the evaluation, I
will add on another bonus of some kind**



Inheritance is an OOP principle that allows us to create structure and hierarchy in our classes

Inheritance allows us to derive a class from another class to get access to attributes and methods. This creates a set of “shared” attributes and methods across different classes



Accountant “inherits from” Employee

To inherit from another class, you need to provide the Class name of the parent

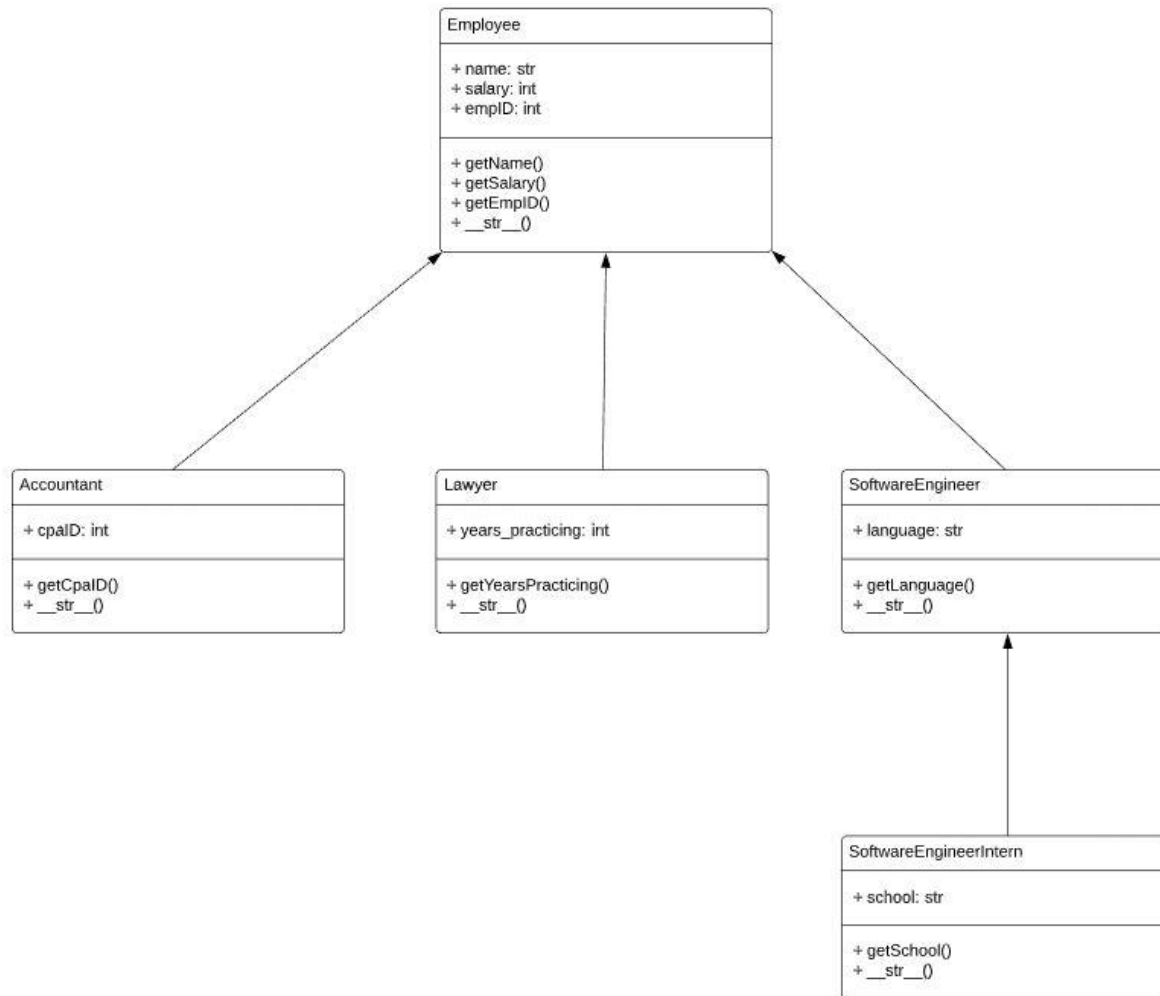
```
class Accountant(Employee):  
  
    def __init__(self, name, salary, empID, cpaID):  
        Employee.__init__(self, name, salary, empID)  
        self.cpaID = cpaID  
  
    def getCpaID(self):  
        return self.cpaID  
  
    def __str__(self):
```

Then, call the parent constructor inside of the child constructor

Another way using the super() keyword

```
class Lawyer(Employee):  
  
    def __init__(self, name, salary, empID, years_practicing):  
        #another way to call the parent class  
        super().__init__(name, salary, empID)  
        self.years_practicing = years_practicing
```

You now have access to the attributes and methods in the parent class



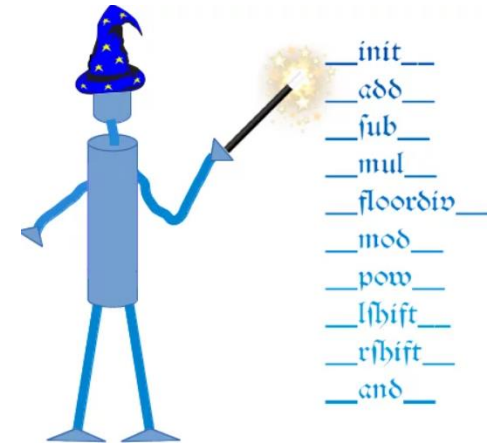
Engineers create **UML Diagrams** to illustrate how classes interact with each other

These help engineers understand how software systems are structured without needing to dive deep into the source code

If you are a CS major, you will make plenty of these 😊

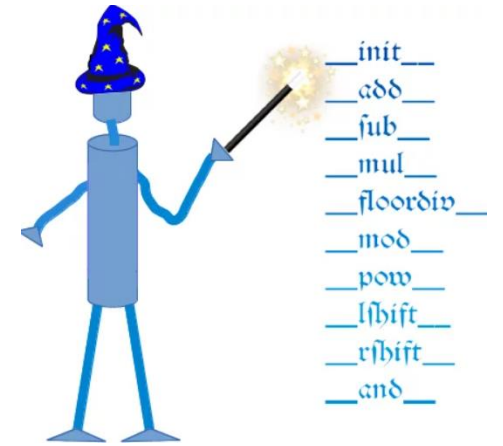
Magic Methods

Allows us to control the behavior of our program when doing an operation on an object



Magic Methods

Allows us to control the behavior of our program when doing an operation on an object



```
food_order = Order("McDonalds",10.99)
```

```
drink_order = Order("Starbucks",7.50)
```

```
total = food_order + drink_order
```

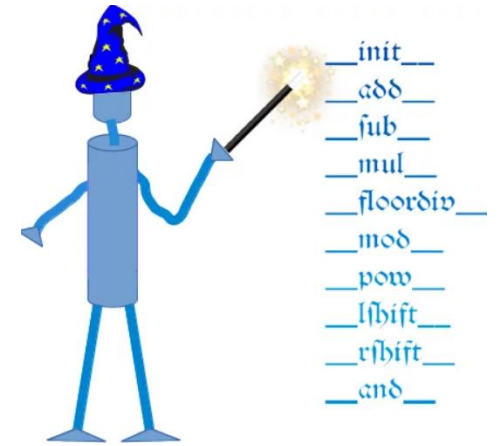
```
print(total)
```



What happens when we try to add two objects together??

Magic Methods

Allows us to control the behavior of our program when doing an operation on an object



```
food_order = Order("McDonalds",10.99)
```

```
drink_order = Order("Starbucks",7.50)
```

```
total = food_order + drink_order
```

```
print(total)
```

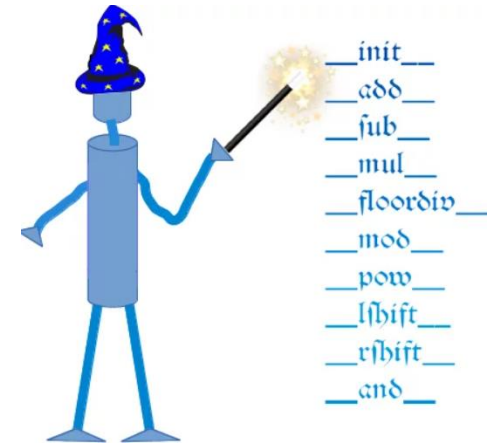


What happens when we try to add two objects together??

```
TypeError: unsupported operand type(s) for +: 'Order' and 'Order'
```

Magic Methods

Allows us to control the behavior of our program when doing an operation on an object



```
food_order = Order("McDonalds",10.99)
```

```
drink_order = Order("Starbucks",7.50)
```

```
total = food_order + drink_order
```

```
print(total)
```

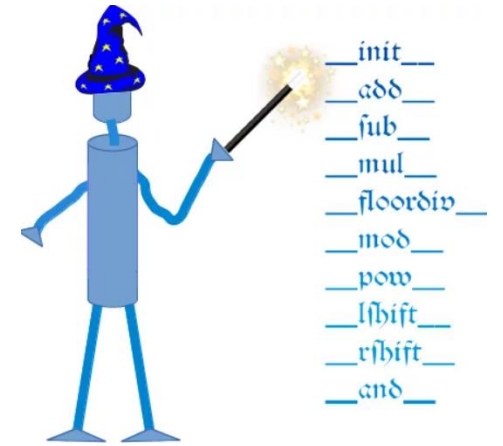


What happens when we try to add two objects together??

We can tell Python and control what we want to happen if we try to add two objects !!

Magic Methods

Allows us to control the behavior of our program when doing an operation on an object



```
food_order = Order("McDonalds",10.99)
```

```
drink_order = Order("Starbucks",7.50)
```

```
total = food_order + drink_order
```

```
print(total)
```



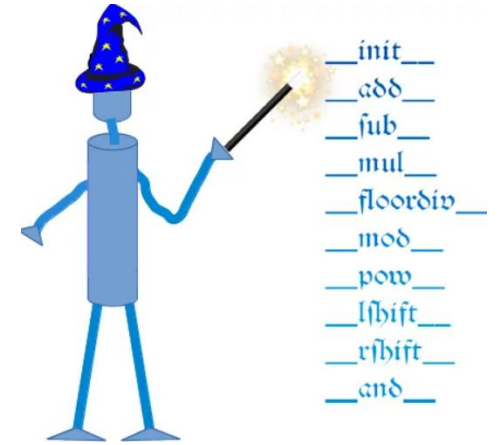
What happens when we try to add two objects together??

We can tell Python and control what we want to happen if we try to add two objects !!

using **Magic Methods**

Magic Methods

Allows us to control the behavior of our program when doing an operation on an object



```
food_order = Order("McDonalds",10.99)
drink_order = Order("Starbucks",7.50)

total = food_order + drink_order

print(total)
```

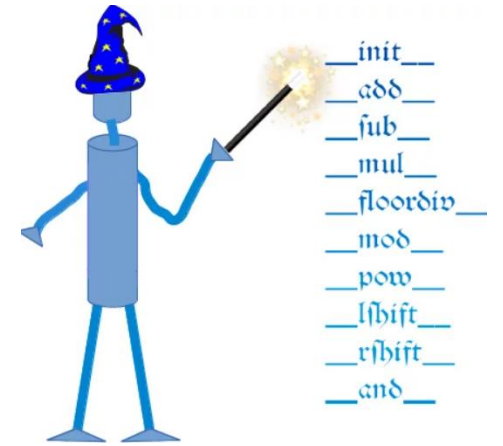
Controls behavior of the + operator



```
def __add__(obj1, obj2):
    return obj1.price + obj2.price
```

Magic Methods

Allows us to control the behavior of our program when doing an operation on an object



```
food_order = Order("McDonalds", 10.99)
drink_order = Order("Starbucks", 7.50)

total = food_order + drink_order
print(total)
```

Controls behavior of the + operator

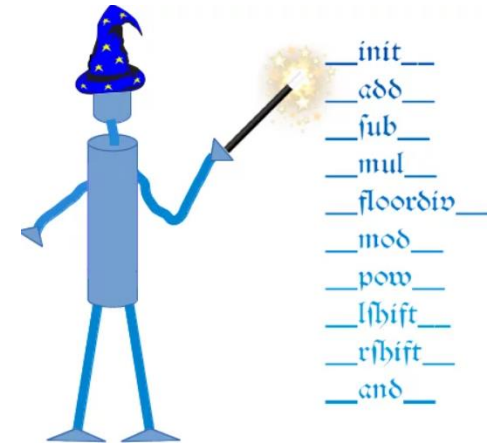
↓

```
def __add__(obj1, obj2):
    return obj1.price + obj2.price
```

When we use the + operator, we are always doing an operator between **two** objects

Magic Methods

Allows us to control the behavior of our program when doing an operation on an object



```
food_order = Order("McDonalds", 10.99)
```

```
drink_order = Order("Starbucks", 7.50)
```

```
total = food_order + drink_order  
print(total)
```

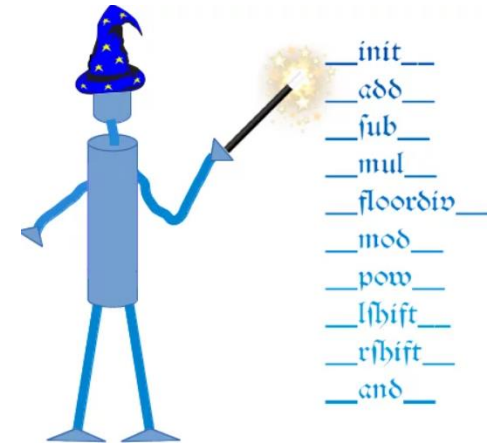
Controls behavior of the + operator

```
def __add__(obj1, obj2):  
    return obj1.price + obj2.price
```

When we use the + operator, we are always doing an operator between **two** objects

Magic Methods

Allows us to control the behavior of our program when doing an operation on an object



```
food_order = Order("McDonalds", 10.99)
drink_order = Order("Starbucks", 7.50)

total = food_order + drink_order
print(total)
```

Controls behavior of the + operator

↓

```
def __add__(obj1, obj2):
    return obj1.price + obj2.price
```

When we use the + operator, we are always doing an operator between **two** objects

Now that we have a magic method defined, we can get an answer!

```
18.490000000000002
```

Magic Methods

List of magic methods

https://www.python-course.eu/python3_magic_methods.php

