Unit 8

Lesson 3 - Class Variables vs. Instance Variables and Inheritance

Learning Targets

- I can describe the differences between class and instance variables.
-

Check-In

Create a new folder called Lesson-3 and inside that folder make a new module called main.py

Create a class called Point inside this module. The init method should only take self as a parameter and set x = 0 and set y = 0. Make the str method print (x,y) where x and y are the x and y values of self

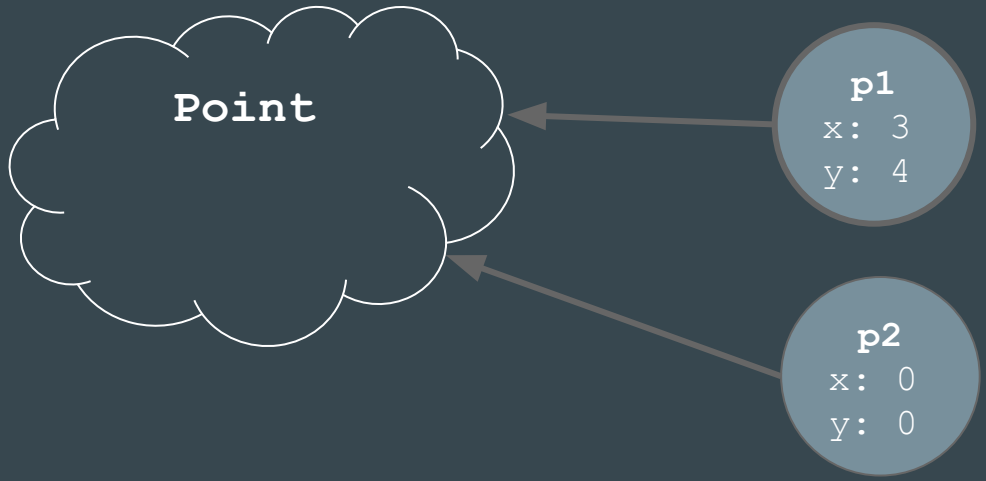
```
1  class Point:
2      def __init__(self):
3          self.x = 0
4          self.y = 0
5
6      def __str__(self):
7          return f"({self.x}, {self.y})"
8
```

Instances

Create two new Point instances. p1 with values (3,4) and p2 with values (0,0)

```
10  def main():
11      p1 = Point()
12      p1.x = 3
13      p1.y = 4
14      p2 = Point
15
```

Instance Variables



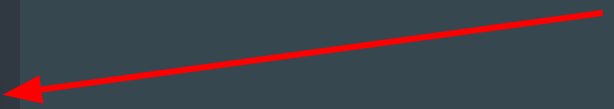
x and y are instance variables

Each instance has an x variable value and a y variable value

Class Variables

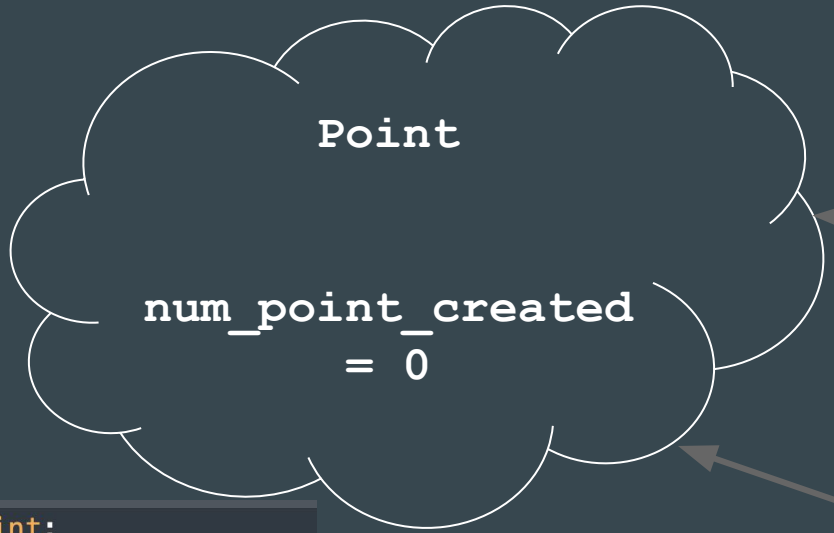
For simplicity, I'm removing the main method and the __str__ method

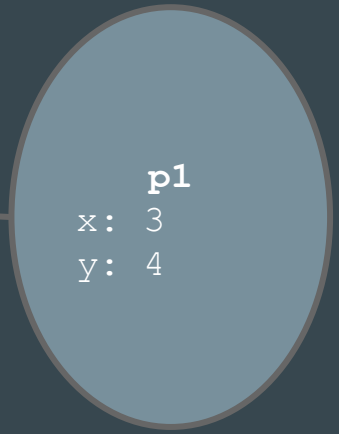
```
1   class Point:
2
3       num_points_created = 0
4
5       def __init__(self):
6           self.x = 0
7           self.y = 0
8
9
10  p1 = Point()
11  p2 = Point()
```

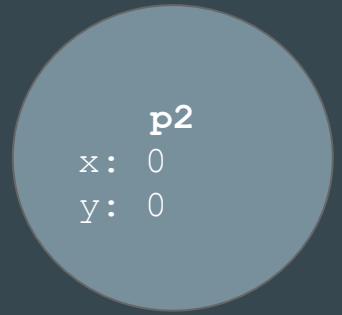


A class variable is a variable that belongs to the the class rather than an instance of the class

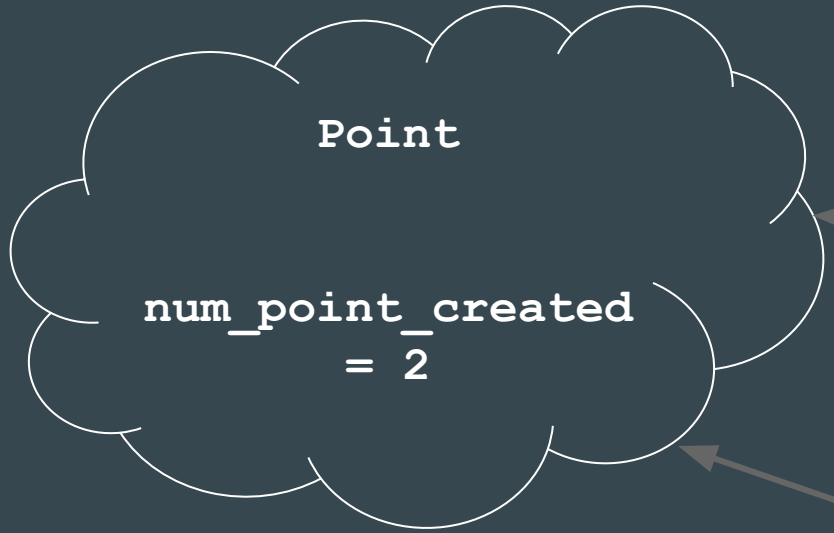
It has some unique behaviors that we need to explore

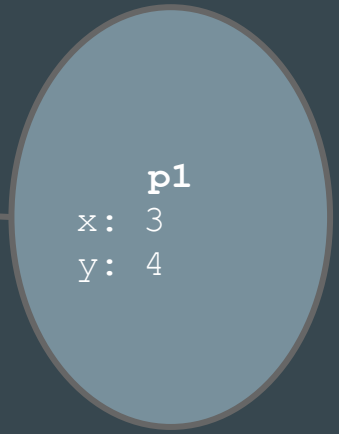




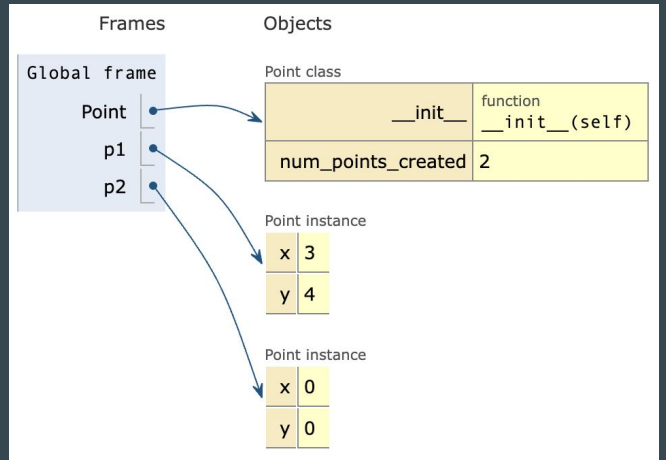



```python
class Point:

    num_points_created = 0

    def __init__(self):
        self.x = 0
        self.y = 0

p1 = Point()
p2 = Point()

p1.x = 3
p1.y = 4

```

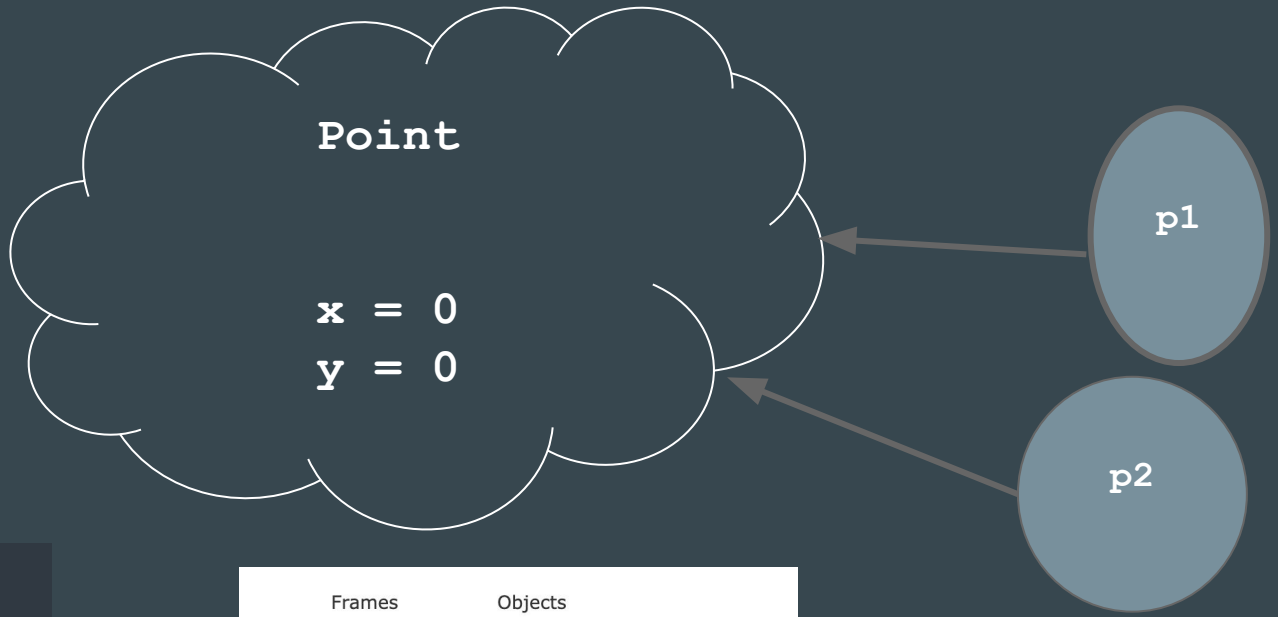




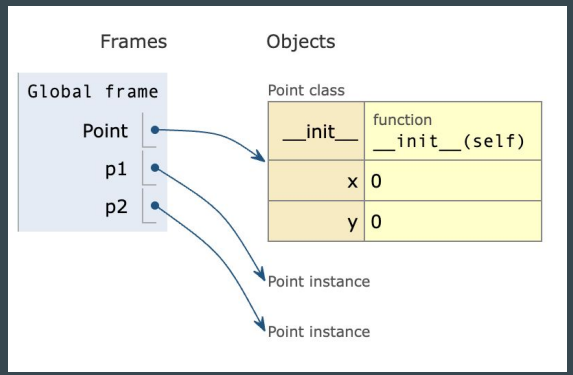

```python
class Point:
    num_points_created = 0

    def __init__(self):
        self.x = 0
        self.y = 0
        Point.num_points_created += 1


p1 = Point()
p2 = Point()

p1.x = 3
p1.y = 4
print(Point.num_points_created)
print(p1.num_points_created)
print(p2.num_points_created)
```



What if we just used class variables only?

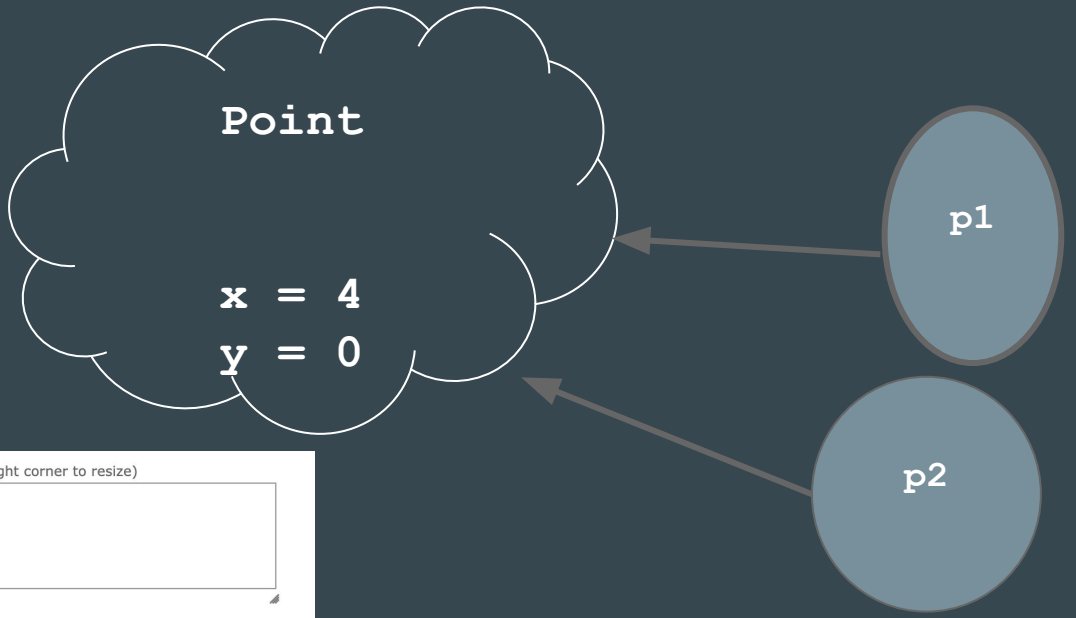


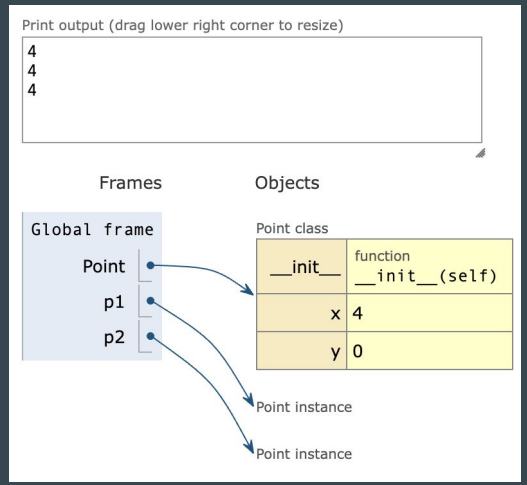

```
1  class Point:
2      x = 0
3      y = 0
4
5      def __init__(self):
6          pass
7
8  p1 = Point()
9  p2 = Point()
10
```



What if we just used class variables only?

```
1  class Point:
2      x = 0
3      y = 0
4      def __init__(self):
5          pass
6
7
8  p1 = Point()
9  p2 = Point()
10 Point.x = 4
11 print(Point.x)
12 print(p1.x)
13 print(p2.x)
```

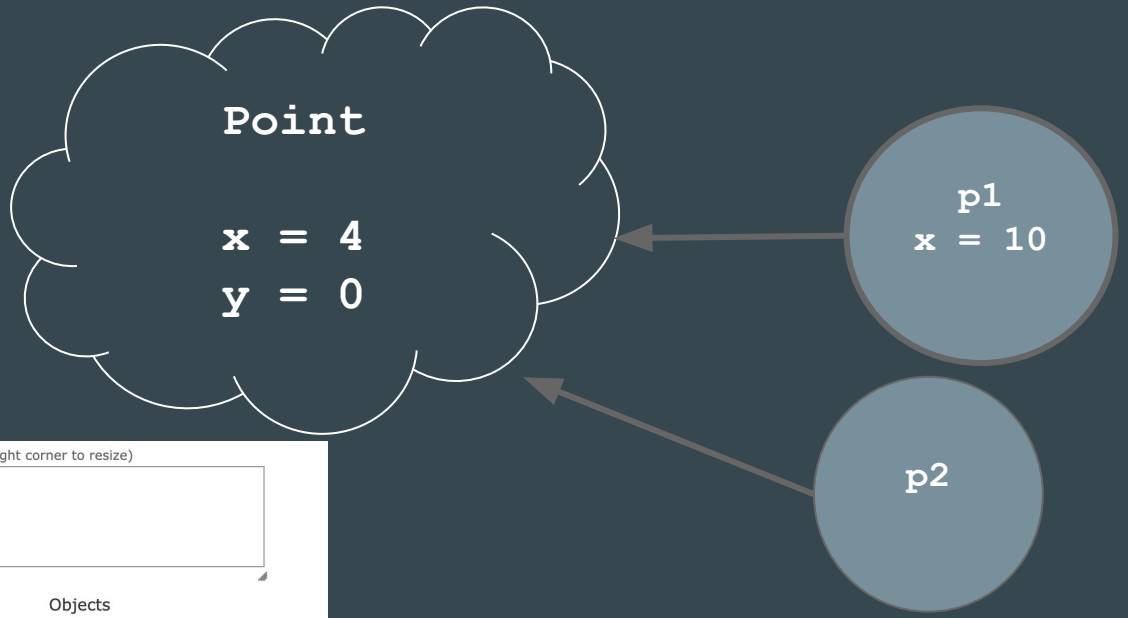


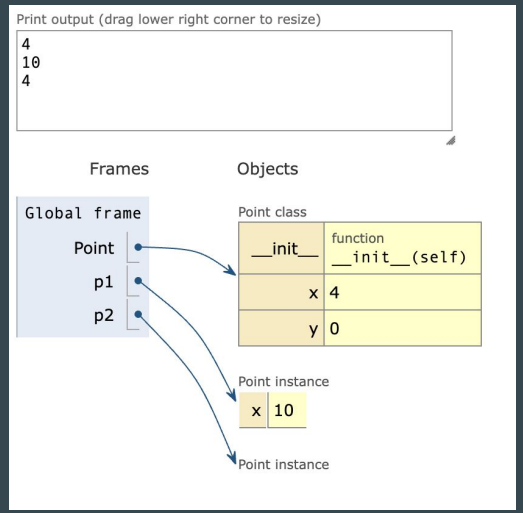


p1 and p2 do not have instance variables x and y so they go to the class variable

What if we just used class variables only?

```
1  class Point:
2      x = 0
3      y = 0
4      def __init__(self):
5          pass
6
7  p1 = Point()
8  p2 = Point()
9  Point.x = 4
10 p1.x = 10
11 print(Point.x)
12 print(p1.x)
13 print(p2.x)
```





p1 has an instance variable x with a value of 10 so 10 is printed, but p2 does not so 4 is printed

Summary

This can be really confusing - which x is which?

You can tell by looking at the class - sort of

If it is Point.x you know it is the the class variable since the class cannot access instance variables, but if it is an instance.variable, you cannot always be certain

Use class attributes to define properties that should have the same value for every class instance.

Use instance attributes for properties that vary from one instance to another.

Other interesting Pythonic behaviors

```
1  class Point:
2      def __init__(self):
3          self.x = 0
4          self.y = 0
5
6
7  p1 = Point()
8  p2 = Point()
9  print(p1.z)
10
```

```
Traceback (most recent call last):
  File "/Users/brandon/Documents/RandomPython/point.
    print(p1.z)
          ^^^^
AttributeError: 'Point' object has no attribute 'z'
```

Actually not that interesting...but the next slide is

Other interesting Pythonic behaviors

```python
class Point:
    def __init__(self):
        self.x = 0
        self.y = 0


p1 = Point()
p2 = Point()
p1.z = 5
print(f"z = {p1.z}")

```

```
z = 5
```

You are able to dynamically add a new attribute at runtime in Python. Python objects are dynamic since they use dictionaries to store their attributes. Each instance has its own attribute dictionary, __dict__, which stores its attributes and their corresponding values. When you assign p1.z = 5, Python just adds the 'z' key to the p1.__dict__ dictionary with the value 5.

Inheritance

is-a relationship vs has-a relationship

attributes are an example of has-a relationships in a class

a Point has-a x value

a Point has-a y value

a Point has-a distance between another Point

is-a relationship vs has-a relationship

We can also have is-a relationships and this comes from inheritance

A class can inherit attributes (methods and properties) from another class.

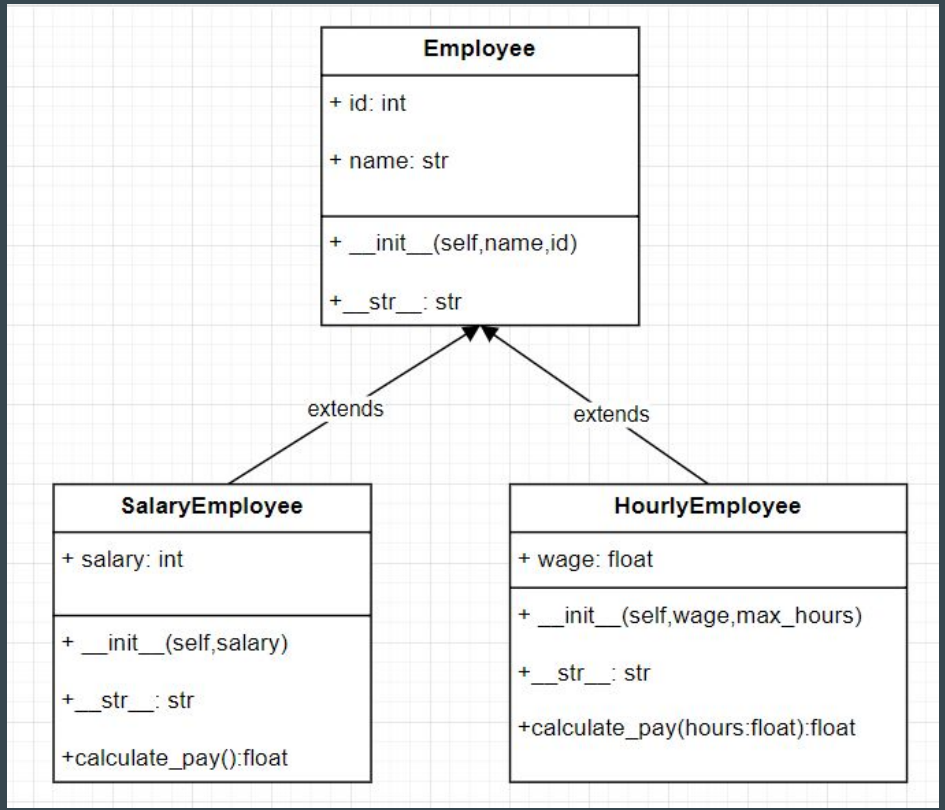
For example, in the code below, the Dog class inherits attributes from the Canine class.

We could say that a Dog **is-a** Canine

```
1  class Dog(Canine):
2      # details not shown
3
```

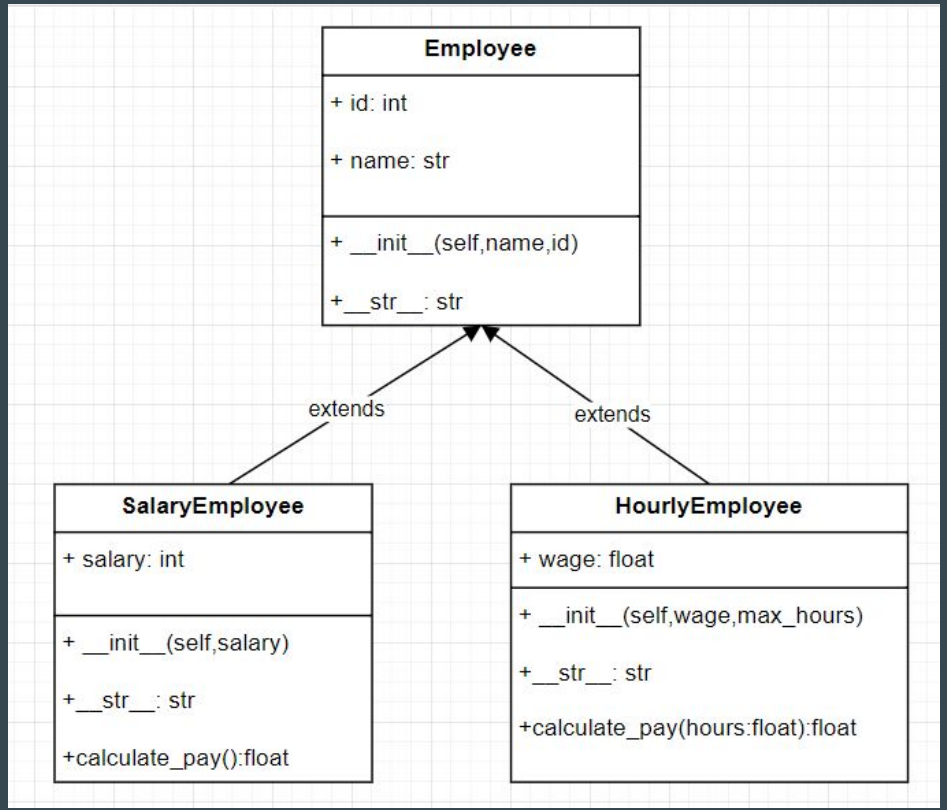
Unified Modeling Language (UML)

- Standardized way of showing the design of a system
- Often used to show class hierarchies



Unified Modeling Language (UML)

- + and - denote public and private, respectively
  - Python doesn't support formal access modifiers so we mark every member as public (+)
- Sometimes put the data type or return types after the attribute



Creating the basic employee class and subclasses

Finishing the Employee Classes

- Create a way to calculate a weekly paycheck for each employee and a method to print their paycheck. Print both regular and overtime hours and pay
  - Assume salary people have a yearly salary and are paid for a 52 week year
    - Hint: don't need a parameter
  - Assume hourly employees can work over the max
    - Hint: need a parameter to check this
- Print their employee type
- Create a new class CommissionEmployee that is a subclass of SalaryEmployee. They earn a salary but get more money based on a number of sales