Object Oriented Programming

Aditya Iyer

Ice Breaker: What’s something good that happened to you recently?

Procedural Programming

- This is what you’ve been doing until now!
- Storing data in the form of variables
- Creating functions that act on the data

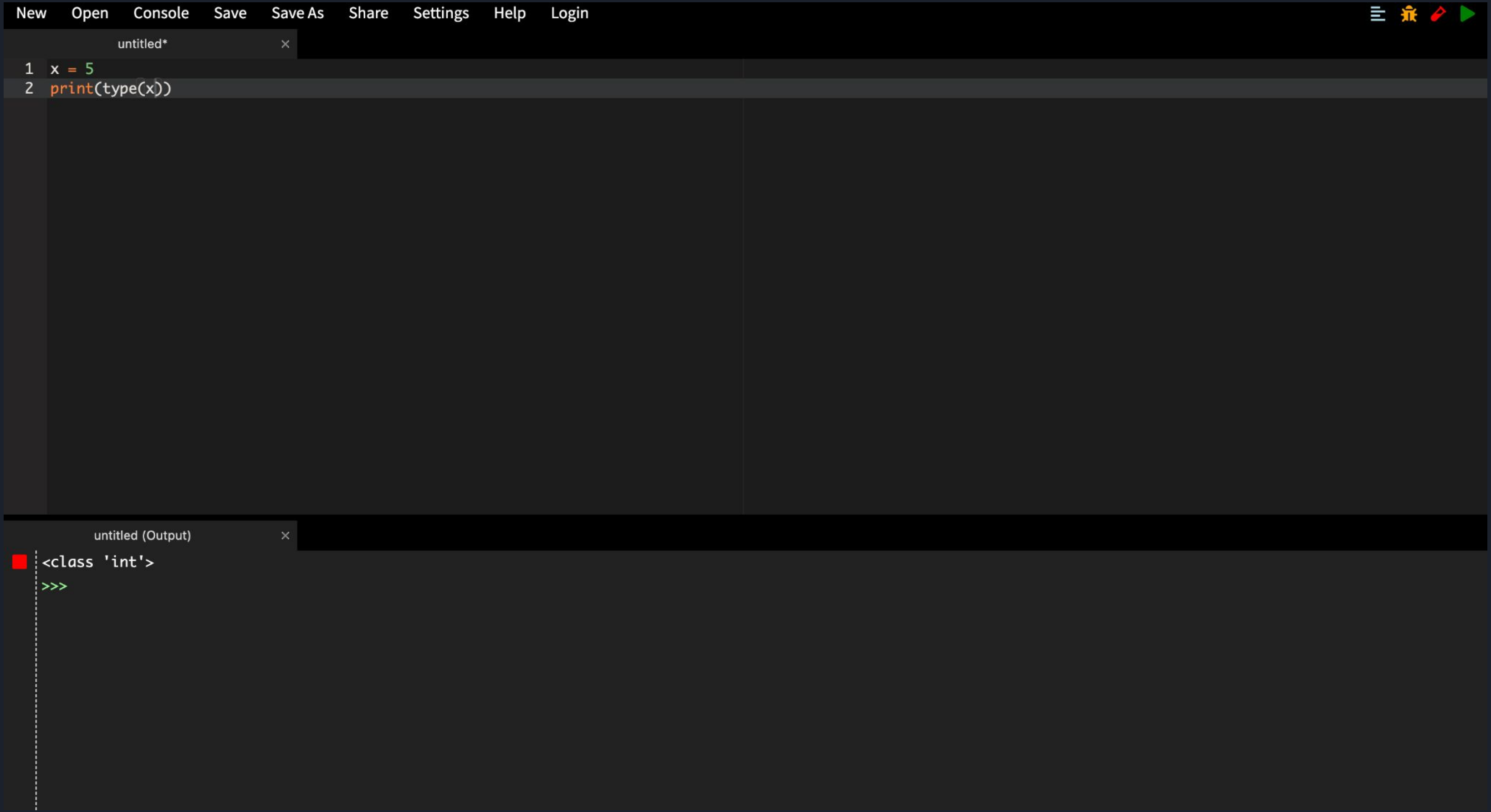
The problem with procedural programming?

Say I’m trying to create a program that replicates a zoo. If I was to do this with procedural programming, it would work but it would be very tedious.

However, the bigger issue is that we’d have a lot of similar code and a very long code file (which also increases the chance of bugging out)

Remember: Complexity is the enemy!

But what if I told you we've already been working with objects?



```
x = 5
print(type(x))
```

```
<class 'int'>
>>>
```

So what is a class?

A class is a blueprint for creating objects.

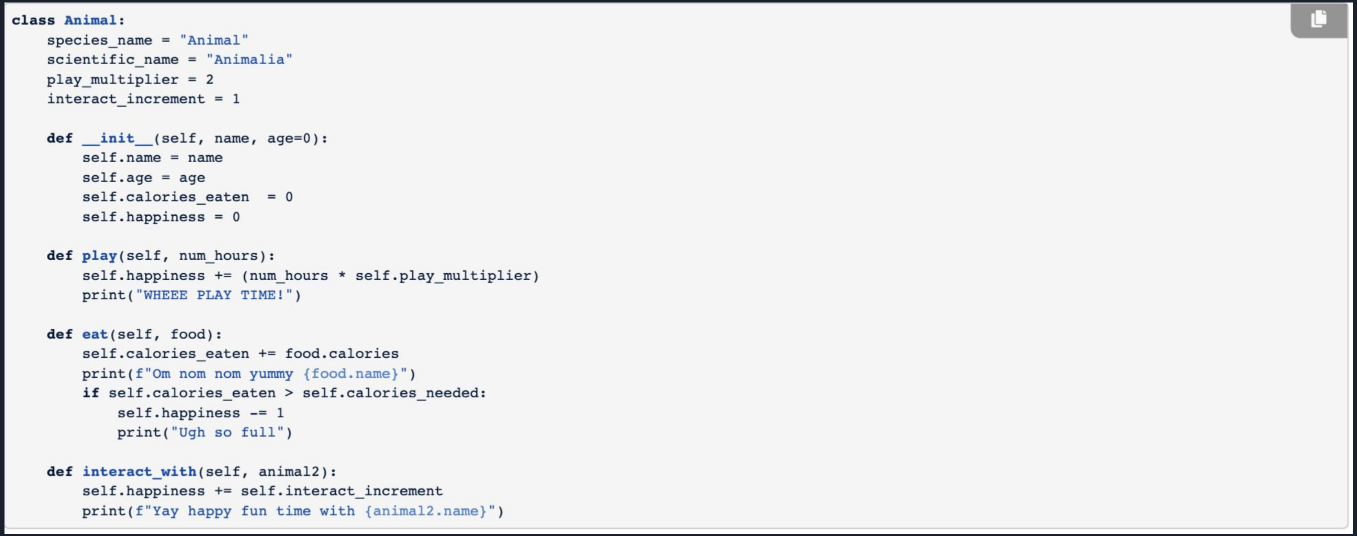
Okay but what does that really mean?

- A class helps us provide initial values for our objects (which we can then modify later)
- It provides us with the implementation of how the objects of that class should behave (methods)

Class variable: an attribute of the object that is shared by all the members of the class

For example, if I have a Shark class, I can have an class variable called animal_species=fish (because every single shark is a fish.

What does a class look like?



```python
class Animal:
    species_name = "Animal"
    scientific_name = "Animalia"
    play_multiplier = 2
    interact_increment = 1

    def __init__(self, name, age=0):
        self.name = name
        self.age = age
        self.calories_eaten  = 0
        self.happiness = 0

    def play(self, num_hours):
        self.happiness += (num_hours * self.play_multiplier)
        print("WHEEE PLAY TIME!")

    def eat(self, food):
        self.calories_eaten += food.calories
        print(f"Om nom nom yummy {food.name}")
        if self.calories_eaten > self.calories_needed:
            self.happiness -= 1
            print("Ugh so full")

    def interact_with(self, animal2):
        self.happiness += self.interact_increment
        print(f"Yay happy fun time with {animal2.name}")
```

Credits: Pamela Fox

Objects

Objects are instances of classes.

How do you create an object? You call the constructor of the class.

animal1 = Animal (“Tom”)

Instance Variables: An attribute of the object where the attributes are specific to that object.

Class Variables vs Instance Variables

The way I like to think about class variables vs instance variables is as follows:

1. Does this property apply to every single object of the class?
   a. If yes, then it’s probably a class variable. For example, every dog on earth will have the scientific name “canis lupus familiaris”, therefore its a class variable.
   b. If no, then it’s probably an instance variable. For example, is the weight of every dog on earth the same? No, therefore it should be an instance variable as it’s specific for that instance.

Self and Dot Notation

- All class methods take in `self` to specify which instance of an object we are calling the method on
  - In the method, `self` refers to that instance
- We can pass in `self` either implicitly or explicitly:
  - Implicitly: `my_cat.eat_food()`
  - Explicitly: `Cat.eat_food(my_cat)`
- Common error: `my_cat.eat_food(my_cat)`
  - `Error: eat_food takes 1 positional argument but 2 were given`

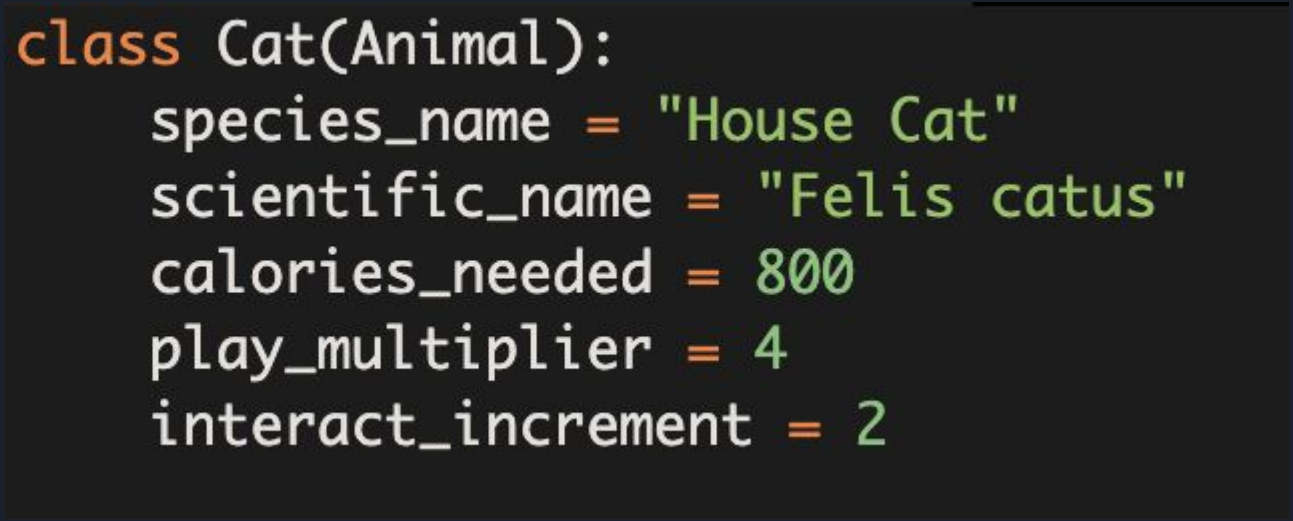
Credits: Marie Chorpita

Inheritance

Now what if I wanted to create a cat object and a panda object? The cat and panda are different animals and they have different behaviours and attributes.

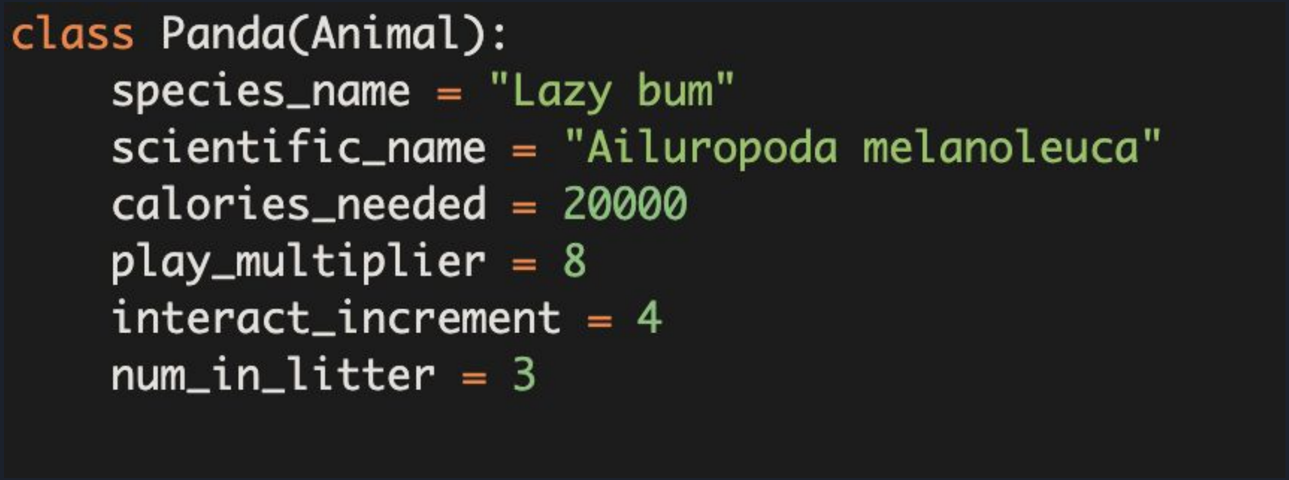
In such an instance, we’d create a cat class that is a subclass of Animal (same thing for panda).

Cat Subclass



```
class Cat(Animal):
    species_name = "House Cat"
    scientific_name = "Felis catus"
    calories_needed = 800
    play_multiplier = 4
    interact_increment = 2
```

Panda Subclass



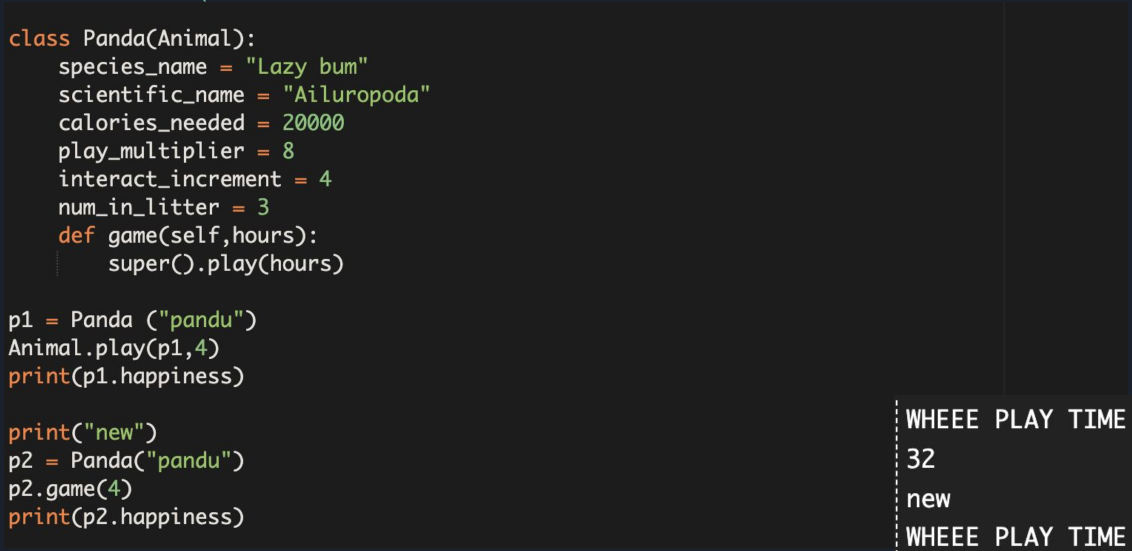
```python
class Panda(Animal):
    species_name = "Lazy bum"
    scientific_name = "Ailuropoda melanoleuca"
    calories_needed = 20000
    play_multiplier = 8
    interact_increment = 4
    num_in_litter = 3
```

Super()

Calling super() gives us the parent class of our current instance (self). For example, if I’m a panda object and I call super() it’ll take me to the Animal class.

- Calling super().play() is the same as calling Animal.play() and self is implicitly passed

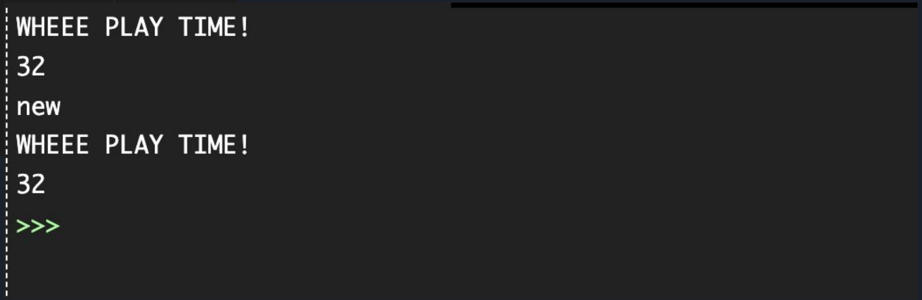
Some code to understand super



```python
class Panda(Animal):
    species_name = "Lazy bum"
    scientific_name = "Ailuropoda"
    calories_needed = 20000
    play_multiplier = 8
    interact_increment = 4
    num_in_litter = 3
    def game(self,hours):
        super().play(hours)

p1 = Panda ("pandu")
Animal.play(p1,4)
print(p1.happiness)

print("new")
p2 = Panda("pandu")
p2.game(4)
print(p2.happiness)
```



```
WHEEE PLAY TIME!
32
new
WHEEE PLAY TIME!
32
>>>
```