Transitioning to Object Oriented Programming

Do Now

- Login
- Create a new folder in your `Classes` folder called `Lesson 1`
- Move `rational.py` into the `Lesson 1` folder
- Create another new folder in `Classes` called `Lesson 2`
- Open Thonny
- Make a new file called `animal.py` and save it in the `Lesson 2` folder.

Quick Review

- In the previous lesson, we created our first custom class and made instances of that class.
- Instances of a class are called ***objects***.
- The programming technique of using objects is called *object-oriented* programming.
- Python is an object-oriented programming language.

Some Vocabulary

- A data type with *states* and *behaviors* is an **object**.
- A state is called a **field**.
- A behavior is represented by a **method**.

Important 3 Rules of Objects

1. Objects are described by their fields.
2. Objects interact with other objects with methods.
   - Methods can also be used to change an object's state (fields).
3. Objects interact with program users through an interface.

Object Fields

- Object fields can be any Python prepackaged object such as string, ints, booleans, floats, or they may be another object that is an instance of a custom made class.
- For example, a dog can have a state of `brown_hair` and a state of `barking`.
- When a state is describing whether or not an action is taking place we use identifier names like `is_alive`, `is_running`, `is_snoring`, etc.

Naming convention: States that describe whether a behavior is occurring can be named with an “is” and an “ing” like `is_barking`. Behaviors (methods) are ALWAYS PRESENT TENSE VERBS like `bark`.

Object Fields

- Object fields can be any Python prepackaged object such as string, ints, booleans, floats, or they may be another object that is an instance of a custom made class.
- For example, a dog can have a state of `brown_hair` and a state of `barking`.
- When a state is describing whether or not an action is taking place we use identifier names like `is_alive`, `is_running`, `is_snoring`, etc.

Naming convention: States that describe whether a behavior is occurring can be named with an “is” and an “ing” like `is_barking`. Behaviors (methods) are ALWAYS PRESENT TENSE VERBS like `bark`.

Some “Real Life” objects with states and behaviors

Object	Fields	Methods
Dog	• name • breed • color • weight • is_barking • is_running	• bark • run • sit • eat
Student	• name • schedule • age • gender	• study • write_code
Pencil	• color • is_sharp	• sharpen • write

Come up with one of your own.

More Vocabulary

- Objects are made from a special block of code called a **class**.
- A file that contains functions and/or classes is called a **module**.
- An object's states are special variables called **instance fields**.
  - The words "state" and "instance field" are synonymous.
- An object's behaviors are special functions called **instance methods**.
  - The words "instance method" and "behavior" are synonymous
- Every instance method must have at least one parameter.
  - The first parameter is always `self`.
- A special method in a class for initializing objects is called an **initializer** or a **constructor**.

The Animal Module

- Create a new module and name it `animals`.
- Add the following class to your module:

```
class Dog:
    def __init__(self):
        self.breed = "Lab"
        self.gender = "Female"
        self.age = 2
```

The `__init__()` method is a very special method. It is called the *initializer.* `self` must be the first parameter in the initializer.

To declare a class, simply write the word `class`, followed by the class name and a colon.

Remember that the `def` keyword is used to define a function or to define a method.

Remember: Functions are defined in a module, but methods are defined in a class.

The Animal Module

- Now we will create some instances of our Dog class.
- An object is often called an *instance of a class* and the act of creating it is called *instantiation*.
- Add the following main function to your `animal` module

```
def main():
    my_dog = Dog()   #instantiating a Dog object
    print(my_dog)    #printing our Dog object


if __name__ == "__main__":
    main()
```

When you run the module, the output should be something like:

```
<__main__.Dog object at 0x02F4AA10>
```

The __main__ is the current program thread. The hexadecimal number is a description of the object's location in memory.

The Animal Module

- Add the following to your main function in your `animal` module

```
my_dog2 = Dog()  #instantiating another Dog
print(my_dog2)
```

When you run the module, the output should be something like:

```
<__main__.Dog object at 0x02A99350>
<__main__.Dog object at 0x030AF8B0>
```

Your numbers will be different from mine, but notice that the *memory locations are different* because they are *different objects*.

The Animal Module

- Now lets add code to print each of the states of our Dog objects.
- Make your main function the same as the one shown:

```
def main():
    my_dog = Dog()    #instantiating a Dog object
    my_dog2 = Dog()   #instantiating another Dog
    print(my_dog.breed)  #print the breed
    print(my_dog2.age)  #print the age
```

- Output:

```
Lab
2
```

Changing States

- To access an instance field use the *dot operator*.

```
identifier_name.state
```

- Update your main method to the one shown. Pay extra close attention to the fact that each dog object has its own age that can be different.

```
def main():
    my_dog = Dog()
    my_dog2 = Dog()
    print(“My first dog's age:",my_dog.age)
    my_dog.age += 1 #First dog gets older
    print("My first dog’s age now:",my_dog.age)
    print("My other dog’s age:“,my_dog2.age)
```

Parameters in __init__

- Currently our Dog class in the animal module has only *default* fields.
- When we instantiate our dogs, we want to be able to control what state each dog has.
- Not all dogs are female labs after all!

Parameters in __init__

- Change the dog class __init__ method (constructor) to allow customization.

```
def __init__(self, b, g, a):
    self.breed = b
    self.gender = g
    self.age = a
```

- Change the calls to the constructor to include the necessary information.

```
def main():
    my_dog = Dog("Boxer","Male",1)
    my_dog2 = Dog("Great Dane", "Female", 8)
    print("My first dog:",my_dog.breed,my_dog.gender,my_dog.age)
    print("My second dog:",my_dog2.breed,my_dog2.gender,my_dog2.age)
```

Output:

```
My first dog: Boxer Male 1
My second dog: Great Dane Female 8
```

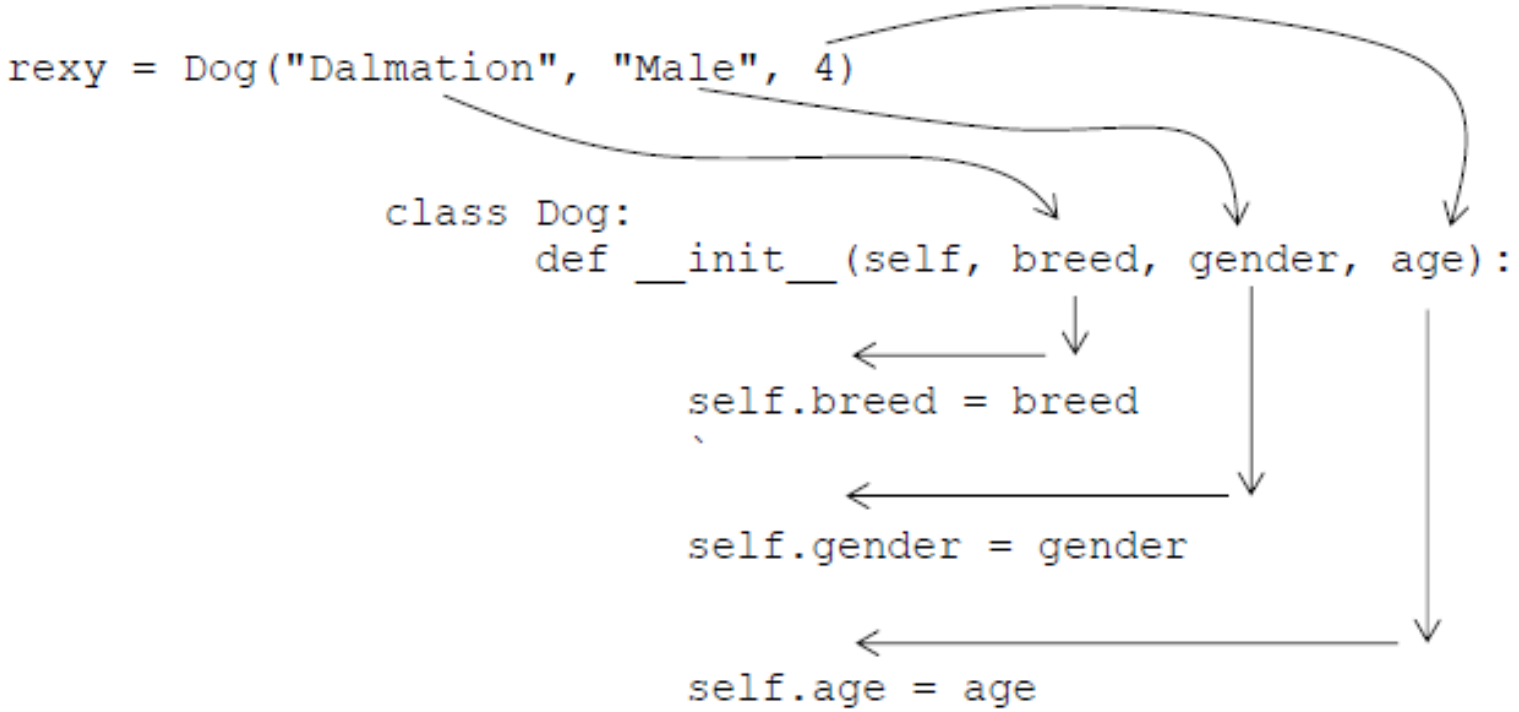
Parameters in __init__

- When someone else uses our class, they will see the identifiers (names) of the parameters. Therefore, it makes sense to actually use the same name for the constructor parameters as for the field names:

```
def __init__(self, breed, gender, age):
    self.breed = breed
    self.gender = gender
    self.age = age
```

Parameters in __init__

- To better understand how this works, consider the path of each variable given this call:



Important: Each instance of a class gets its own copy of an instance field.

Therefore, when we create a `Dog` with identifier `rexy`, `self.age` becomes `rexy.age`.

Animal Type

- Before leaving the Dog class for a bit, add one more instance field: `animal_type`
- For this field, it will not be determined by a parameter when calling the constructor because all Dogs are the same animal type.

```
class Dog:
    def __init__(self,breed,gender,age):
        self.breed = breed
        self.gender = gender
        self.age = age
        self.animal_type = "dog"
```

Cat Class

Now we will add a new class to our module, Cat.

```
class Cat:
    def __init__(self,color,short_hair,gender,age):
        self.color = color
        self.short_hair = short_hair
        self.gender = gender
        self.age = age
        self.animal_type = "cat"
```

Cat Class

Update the main function:

```
def main():
    my_dog = Dog(“Boxer”,”Male”,1)
    my_dog2 = Dog(“Great Dane”, “Female”, 8)
    print(“My first dog:”,my_dog.breed,my_dog.gender,my_dog.age)
    print(“My second dog:”,my_dog2.breed,my_dog2.gender,my_dog2.age)
    my_cat = Cat("Orange",True,"Female",1)
    print(“My Cat is”,my_cat.color)
```

Output:

```
My first dog: Boxer Male 1
My second dog: Great Dane Female 8
My Cat is Orange
```

Summary

- In this lesson you:
  - Created another module with two new class types.
  - Created more instance methods.
  - Practice more object oriented programming techniques.

Exercises

1. Add a `name` parameter for both the Cat and Dog initializer that sets the instance field `name` of the animal.
2. Add a Pig class to the animal module. Make the instance fields `name` and `age` that are set by the initializer (and `animal_type` that is always the same).

Exercises - #3

Add code so that when the animal module is run that the user is asked whether they want to make a Dog, Cat, or Pig. Then ask for the relevant parameters to match the initializer for the animal being created. Then print out a description of their animal.

For example, consider the following (user input is bold):

```
Dog, Cat, or Pig?  Cat
What color? grey
Short hair – yes or no? yes
Gender? male
What is the age? 12
You have a 12 year old male, grey, short haired cat.
```

Exercises - #4

Create a module called `fruit`. In the `fruit` module make 4 different fruit classes. These class names should start with a capital letter such as `Apple`, `Pear`, and so forth…

- Each class should have at least 3 unique fields and an initializer method that describe the fruit.
- In a main function, test each of the four classes by creating several different fruits. Have at least one from each fruit class plus one that is the same fruit but with different characteristics.

Exercises - #5

Create a module called `shower`. In the `shower` module make classes called `Shampoo`, `Conditioner`, `Soap`, and `ShavingCream`.

Each class should have the following fields:

- `Shampoo: brand, size, smell, is_color_safe, is_for_dry_scalp`
- `Conditioner: brand, size, smell, is_color_safe`
- `Soap: brand, size, smell`
- `ShavingCream: brand, size, is_foamy`
- Test your `shower` module with the tester program on the next slide, `shower_tester.py` saved in the same folder as your shower module. This is a different file from the shower file!

shower_tester.py

```
import shower
if __name__ == "__main__":
    pert = shower.Shampoo("Pert Plus", 24, "strawberry", True, False)
    vs = shower.Conditioner("Vidal Sasoon", 18, "strawberry", True)
    jergy = shower.Soap("Jergens", 6, "fresh scent")
    gillette = shower.ShavingCream("Gillette", 10, True)
    print("I am using", pert.brand, "to shampoo with")
    print("My conditioner", vs.brand, "is", vs.size,"ounces")
    print("My conditioner smells like a", vs.smell)
    if(gillette.is_foamy):
        print("My", gillette.brand, "brand shaving cream is foamy")
    else:
        print("My", gillette.brand, "brand shaving cream is a gel.")
    print("My soap,", jergy.brand,"is",jergy.size,"ounces of",jergy.smell,"soap")
```

Exercises - #6

Make a module called `fps`. In this module, create a class called `Hero` with the following instance fields in this order:

- `alive` – set to `True` if the hero is alive
- `weapon` – a word describing the type of weapon the hero has
- `level` – an integer showing what level of power the hero is at.

- Test your `Hero` with the module called `hero_tester` on the next page.

hero_tester.py

```
import fps
def main():
    link = fps.Hero(True, "Master Sword", 1)
    print("It is", link.alive, "that the hero is alive")
    print("The hero has a", link.weapon)
    print("The hero is at level", link.level)

if __name__ == "__main__":
    main()
```

Output:

```
It is True that the hero is alive
The hero has a Master Sword
The hero is at level 1
```

Exercises - #7

Some instance fields should have a default value for every instance of the class. `Hero` objects created with our `Hero` initializer in the fps module (exercise 6) should probably always start at level 1 with a state of being alive.

Change your `Hero` class so that the initializer only receives a parameter for weapon field.

Hint: You can set an instance field to any value, not just one passed to it.

hero_tester2.py

```
import fps
def main():
    link = fps.Hero("Master Sword")
    print("It is", link.alive, "that the hero is alive")
    print("The hero has a", link.weapon)
    print("The hero is at level", link.level)

if __name__ == "__main__":
    main()
```

Output:

```
It is True that the hero is alive
The hero has a Master Sword
The hero is at level 1
```

Submit your completed modules to Google classroom:

- animal.py
- fruit.py
- shower.py
- fps.py

DO NOT SUBMIT ANY OF THE TESTER FILES