Advanced Python
Lesson 9: Classes

A ***Class*** is a template for making an object. It's really important in object-orientated programming (OOP) which most video game companies use for their games.

A class defines what properties and function an object can have. Think of classes as a definition for an object.

PART 1: Defining a Class and Initializing an Object

Open a new program and save it as **classes.py**

The first part to learning about classes is to make a class. The **definition of a class** looks like this:

```
class classname:

    #variables and functions for the class go indented here
```

Try typing the following code into your program:

```
class Cat:

    lives = 9
```

Now we have the definition for a Cat object that holds an int called `lives` equal to 9.

We still haven't made a Cat object though. We only made the definition for the object. The **initialization of a class object** looks like this:

```
objectname = classname()
```

Try typing the following code into your program:

```
class Cat:

    lives = 9

Cat1 = Cat()

Cat2 = Cat()
```

We now created two Cat objects called Cat1 and Cat2. They both have 9 lives.

What if we wanted to print the `lives` value for Cat1 and Cat2?

What if we wanted to change the `lives` value of Cat2 to 8?

For these questions we need to know how to **access a class variable**, which looks like this:

```
objectname.variablename
```

Try typing the following code at the BOTTOM of your program:

```
Cat2 = Cat()
Cat2.lives = 8
print("Cat1 has", Cat1.lives, "lives")
print("Cat2 has", Cat2.lives, "lives")
```

Now save your code and run it. It should say Cat1 has 9 lives and Cat2 has 8. Changing the variable value for one class object will not change the variable value for another one.

PART 2: Class Functions

Classes can also contain functions within them that only class objects can use.

A **class function definition** looks any other function except it's indented into the class and needs a variable self, which is a reference to the class object calling the function.

```
def functionname(self):
    #Code for function
```

Calling a class function is like accessing a class variable, it looks like this:

```
objectname.functionname()
```

Let's make a new class for a Chameleon to practice with functions. In case you didn't know, chameleons can blend in to the background and become invisible by changing their skin color. We will have a variable and function in our class to mimic that.

Try typing the following code at the BOTTOM of your program:

```
class Chameleon:
    invisible = False
    def blend_in_with_background(self):
        self.invisible = True
```

Notice: we need to pass **`self`**, an object reference, into our function or else we will see an error message.

We put `self.invisible` because we only want the Chameleon that called the function to become invisible. If we put `invisible = True`, then all Chameleon objects would become invisible if only one called the function `blend_in_with_background()`.

Try typing the following code at the BOTTOM of your program:

```
Chameleon1 = Chameleon()
Chameleon2 = Chameleon()
Chameleon1.blend_in_with_background()
print("Chameleon1 is invisible:", Chameleon1.invisible)
print("Chameleon2 is invisible:", Chameleon2.invisible)
```

Chameleon1

Chameleon2

Save your code and run it. Since we only had Chameleon1 blend in with background, it should say Chameleon1 is invisible : True and Chameleon2 is invisible : False.

Class Initializing Function

When we initialize a class object, we are actually calling the built in class initializing function. It usually just returns a reference to the class object, but we can customize it to take in variables when we create the object.

The **class initializing function** looks like this:

```
def __init__(self):
    #Code for function
```

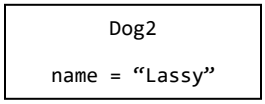
Notice that it needs two underscores "_" on both sides of the function name `init`.

Let's make a new class Dog that will take in a name string for our dog when we create the class object.

Try typing the following code at the BOTTOM of your program:

```
class Dog:
    def __init__(self, dogname):
        self.name = dogname
```

So to create a Dog object, we need to pass in a `dogname` variable. Each dog will then have a `name` variable that will be whatever `dogname` was used when it was created.



Try typing the following code at the BOTTOM of your program:

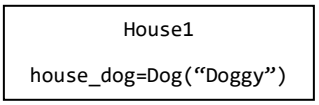
```
Dog1 = Dog("Fido")
Dog2 = Dog("Lassy")
print("Dog1 has a name of", Dog1.name)
print("Dog2 has a name of", Dog2.name)
```

Save your code and run it. Each dog should print their own name.

PART 3: Classes within a Class

A class can have other class objects passed in to it from the class functions. Or you can create a class object within a different class.

Let's say we want to make a House class that creates a Dog object.



Try typing the following code at the BOTTOM of your program:

```
class House:
    house_dog = Dog("Doggy")
House1 = House()
print("The house dog's name is", House1.house_dog.name)
```

Notice that we have to access the Dog object first within the House class, and then access the name variable within the Dog class.

Let's see another example of classes within a class, when we pass a class object through the initialization function of another class.

We will make a Zoo class that requires a Cat, Chameleon, and Dog object to be created. We will also require a city name. Then we'll create a class function that prints all the animal information and city name.

Try typing the following code at the BOTTOM of your program:

```
class Zoo:
    def __init__(self, cat, chameleon, dog, city_name):
        self.cat = cat
        self.chameleon = chameleon
        self.dog = dog
        self.city_name = city_name
    def print_animals(self):
        print("The cat has", self.cat.lives, "lives")
        print("The chameleon is invisible:", Chameleon1.invisible)
        print("The dog has a name of", self.dog.name)
    def print_zoo(self):
        print("The", self.city_name, "Zoo has 3 animals")
        print("Here is some information about them:")
        self.print_animals()
```

To call a class function `print_animals()` from another class function `print_zoo()`, we need to use the class object reference **`self`** or else we get an error.

Now try making two Zoo objects and then calling the print_zoo() function for each zoo.

Hint: `Zoo1 = Zoo(Cat1, Chameleon1, Dog1, "San Francisco")`