David Brady

March 8th, 2021

Foundations of Programming: Python

Assignment 08

https://github.com/D500844/GitAssignments/tree/main/Assignment08

**Objects and Classes**

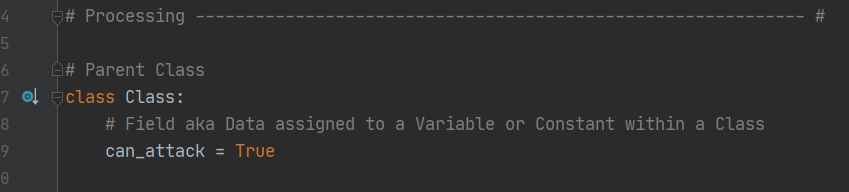
**Overview**

Classes and Objects, as well as their encompassing parts that we have gone over, are huge language-agnostic topics. I have spent some time writing them before but have never had to understand them until now, and it proved to be quite a challenge. I will go over briefly some of the features of Classes, such as its pattern (typically containing fields, constructors, attributes, properties, and methods) how they are different from the instances of objects they create. We will go over the significance of static methods and talk briefly about using the git hub desktop application. This is an extensive topic, and I have utilized both code in the assignment and separate code to create analogies that will make it easier to understand.

**The Blueprint of Object-Oriented-Programming**

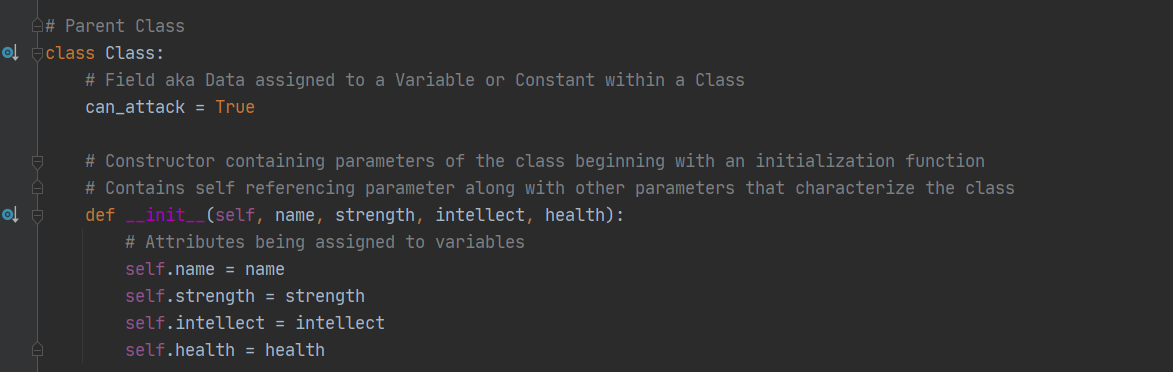
When speaking about OOP, we are talking about code written for, about, within, around, and/or utilizing objects- and there is much more than that that can be done with them. Almost idea or concept can be talked about as a class made up of discernible parts- it’s descriptive elements- constant and static or changing, even moving and interacting. It’s good to *say* we are creating a class, but perhaps easier to *think* we are describing the nature of an object- for in fact both are true at once- and it is not called Class Oriented Programming.

I have begun the work of writing this by creating a blueprint class for a standard Dungeons and Dragons type character in **figure 1** in. We have initially added some confusion by naming our class type “Class”, as this is typically the name of a character-types specialty, so let us hurry and introduce some attributes that will begin make sense of our decision.



**Figure. 1**

So here we have declared class Class and added a field, which is data we use to describe our class. In this case our field tells our class “Class” to always be ready to attack. Let us look deeper at some more elements that can be used to make up classes as we add a constructor and attributes in figure 2. Hopefully my docstring adds clarity.



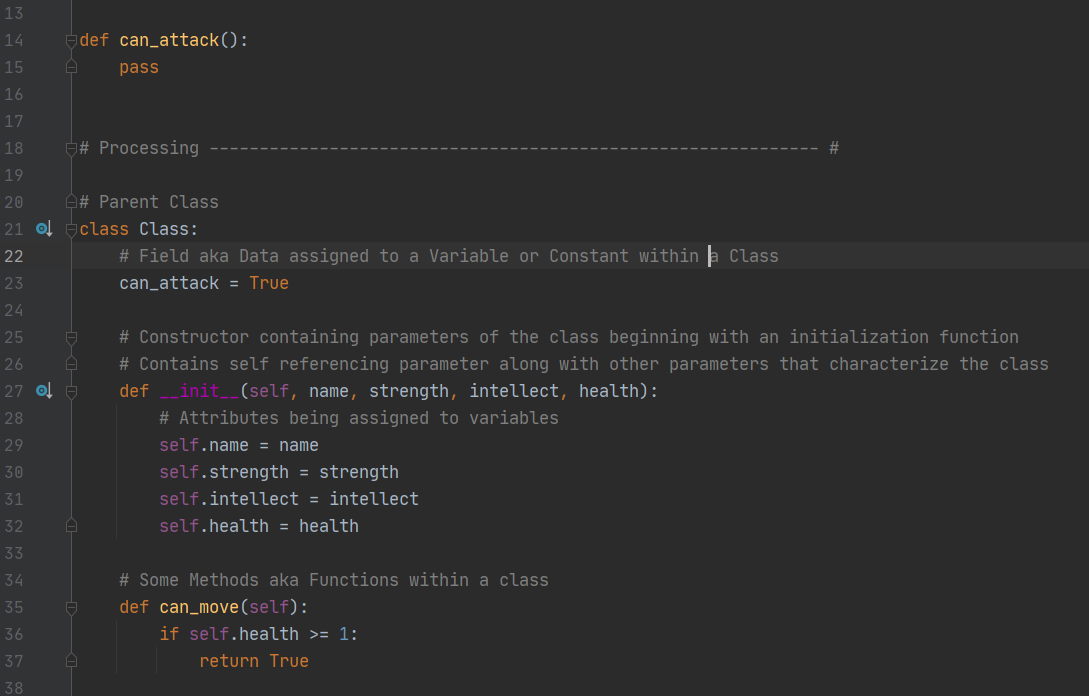
**Figure. 2**

**Constructors, Self-referencing, and Attributes**

Let’s first get the most complicated stuff out of the way. If we remember \_\_init\_\_ represents initialization- then when we use it to declare our functions Attributes, we can think of this as a type of organization and upkeep. Like creating a folder with a group of files inside of it, we create this def \_\_init\_\_() and all the things we write in the parentheses will help us add character and form to our Class. Here we will first add “self”, as this is a necessary part of initialization, and will be critical for referencing our class. Think of this as the class referencing its ”self” when accessing its own characteristics or defining them. It is a difficult concept to describe, but as you use it to form classes it begins to make more sense.  
 Classes \_\_init\_\_ constructor will contain parameters along with self. In this case I have selected parameters that I think will be most suitable to describe types of data that will make up my class “Class”, and in this case since my theme is Dungeons and Dragons, I have chosen a “Class” to be described via quantities of Strength, Intellect, Health and a name. Once we have defined these as being parameters of the class inside the def\_\_init\_\_().

**Functions, Constructors, Methods**

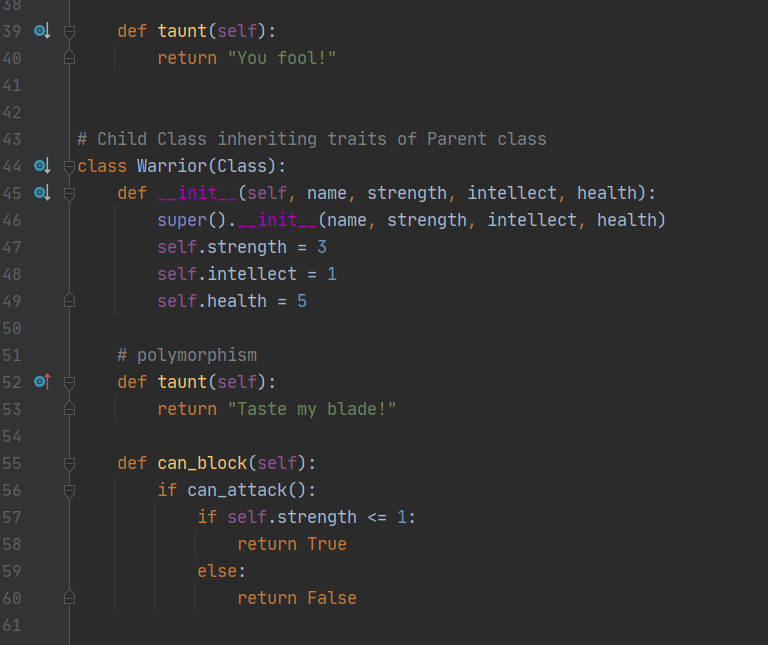
Briefly it is extremely important to create the distinction between these concepts. We have gone over functions but let us add one to our code above our class just to solidify the differences. With this we have a new action that can be performed, a function that can be called and used. The only thing more convenient than this might be creating a function inside of a class that can be ready to use as soon as the class is called and THAT is called a method. Let us create a method inside of our class in figure 3.

**Figure. 3**

Hopefully this is starting to make some sense if you have played a video game in the last 30 years. Our method we have created ((can\_move) aka a function within a class) now sits inside our class (unlike our non-method function (can\_attack)) which will need to be specifically called or changed outside of the class if objects are going to be using it, at least for now, but as it stands- not all objects created of the class-type “Class” can\_attack, but all of them can\_move as long as they have 1 point of health or more.

Constructors are a special form of methods, and there are many of them. We will look only at \_\_init\_\_() for now since it is easy enough to understand special-methods like Constructors in how they fit with classes for now, as they could be a whole topic on their own and a deep rabbit hole very quickly. We used a constructor to define our parameters and self-reference, but this is still very similar to a function inside a class- which is why it is called a special-method.

Now let’s see how class “ inheritance” works by creating a new child class of our parent class “Class”: the “Warrior”, in figure 4. We will also add an additional method to our parent class called “taunt” which all classes of the class-type “Class” will be able to perform. However take note that in this example our “Warrior” child class has a different return for it’s method “taunt” which appears to take the same name.



**Figure. 4**

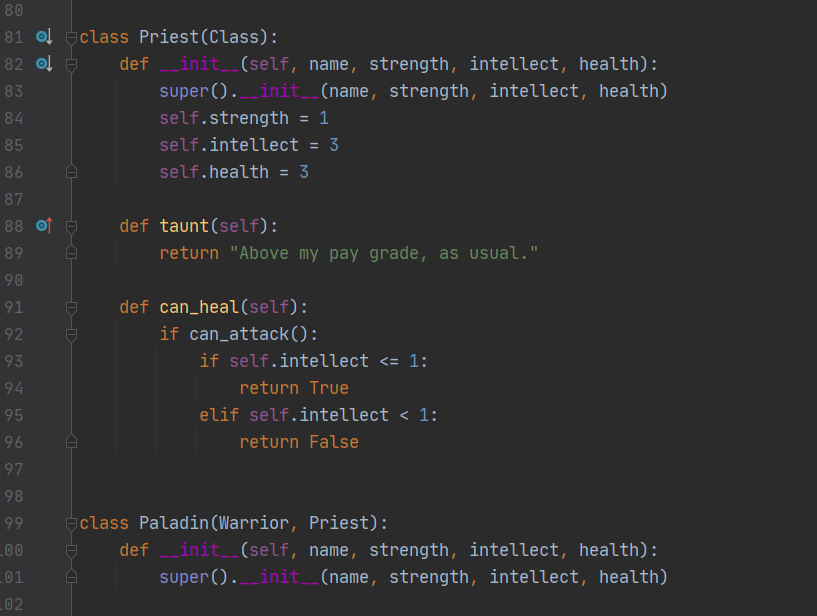
In this case what we are seeing is called “polymorphism”. In inheritance, the child class inherits the attributes and methods of the parent class- but the children classes can change how these methods will return information to make them more unique and suited for their niche’ functionality. In this case, a Warrior will carry a sword- so they might be able to taunt with the classic “taste my blade!”, but a wizard might not have a sword, so such a taunt does not fit every class type we can think of- unlike our parent class default taunt “you fool!” which just about anyone could shout!

**Parent classes and Children classes**

Inheritance can be very tricky to grasp initially but powerful the moment you understand it as it allows us to create huge classes with many layers and possibilities- and ideally a lot of options. In this case I have displayed by child class Warrior(Class), and in parentheses I fill in it’s parent class “Class” to signify that I want to inherit the concepts that define the class type “Class” while describe my “Warrior”.

In this case I know my warrior will have some stats that will be different than other types of characters, so I have filled them in. I also know my warrior will have a shield so I want to make sure they can block and have written a Warrior-specific method for them to use which I have called “can\_block”, though it will need to be able to attack and have enough Strength to do it. Our warrior has a surprising amount of definition already thanks to our parent and children classes, and we can keep building and adding until it becomes whatever we want.

Before moving on I will demonstrate some more versatility within inheritance by creating a child class of class type “Class” called the “Priest” and then I will take the warriors qualities and the priests’ qualities and create one more called “Paladin” which will inherit the descriptions and behaviors of both Priests and Warriors.

**Figure. 5**

**Object creation, demonstration, and Conclusion**

While I understand this will not cover the entirety of OOP, I also must admit it is a shocking rabbit hole of great breadth and depth that consumed over a week of study for me and could easily consume years more. I plan to continue to study it by seeking more books and videos as well as creating more classes for fun- and eventually for work, for as far as I can tell OOP understanding is so beyond rewarding but requiring diligence and respect to comprehend.  
 We will finish this up by creating an instance object of our class type warrior(class) and seeing what it can do by checking its capabilities and reading its information. Below we can print its strength and health and we can give it a name and more. We can do everything with it that we defined inside our classes, and we can make as many as we want. Extremely fun, and only the first few steps into OOP.

**Figure. 6**

