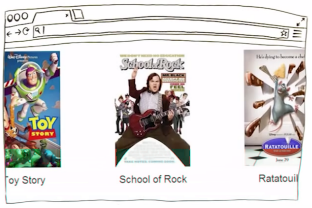
Programming Foundations with Python:
Learn Object Oriented Programming

Lesson 03b Notes

Advanced Ideas in OOP

So we've come a long way in this course, from drawing shapes, to designing a movie website. And what we're going to do here in this lesson is talk about some advanced ideas in object-oriented programming. The first of which is called class variables. Let's talk about it next.



Class Variables

So let's begin by recalling this thing called the instance variables. In the case of class Movie, there were several of them, title, storyline, poster_image_url, and trailer_youtube_url. Now, further recall that these variables are associated with every instance that we create. For example, both Toy Story and Avatar have their own copies of these variables. So I could print out Toy Story's storyline and I could also print out Avatar's storyline.

Sometimes however, we need variables that we want all of our instances to share. So consider the variable valid_ratings for a movie. This is an array or a list, of all possible ratings a movie could have. Now, it would not quite make sense to say, hey, here are Toy Story's valid ratings, and here are Avatar's valid ratings. They would essentially be the same for all instances. Thus, this variable is really associated with a class Movie, and is therefore called a Class Variable. Let's see it in action. So here is the code for our Class Movie. And behind this Python file is the other Python file we created, where we made a bunch of movie instances.

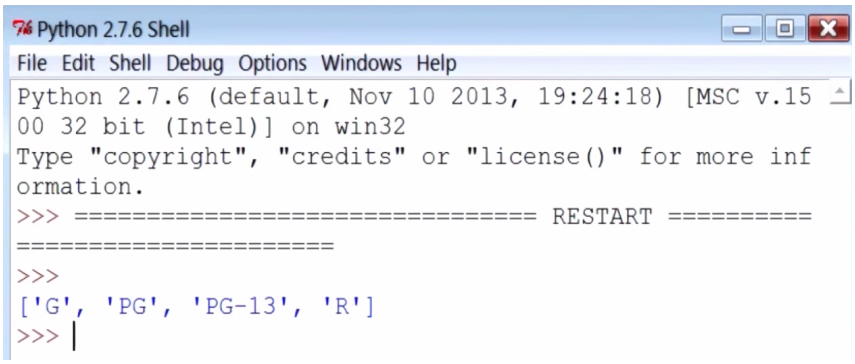
media.py - C:/OOP/movies/media.py

File Edit Format Run Options Windows Help

```
import webbrowser

class Movie():
    VALID_RATINGS = ["G", "PG", "PG-13", "R"]

    def __init__(self, movie_title, movie_storyline, poster_image,
                 trailer_youtube):
        self.title = movie_title
        self.storyline = movie_storyline
        self.poster_image_url = poster_image
        self.trailer_youtube_url = trailer_youtube

    def show_trailer(self):
        webbrowser.open(self.trailer_youtube_url)
```

So I'm going to begin by making changes to my class Movie. Now here I will define a variable called valid_ratings. Now notice that this variable valid ratings is defined at the level of the class and is outside the init function. In order to start using this variable, we will begin by saving this file first.

So next I'm going to go to my other Python file. By the way, this is the file where we are defining a bunch of movie instances. And here, I will scroll down all the way to the bottom. And comment out any sort of print or output statements. Now, I'm doing this so I can focus primarily on valid_ratings. Now, here, I will try to print out the value of the variable valid ratings by saying, print, and the name of my class, which is media.Movie, followed by the name of the variable, which is valid_ratings. There. Let me save and run this program.



And boom. There it is. A list of all of my valid ratings. Notice how we use the class name movie to access this variable. This means that all instances of this class movie, Toy Story, Avatar and others, they can share this list. They can share this list to see if their individual rating is a valid one or not. Okay, so the one last thing I want to do is go back to my class Movie. Now notice that the value of this variable valid_strings is probably a constant. By that I mean, that the value of this variable is probably not going change every now and then. When we define a constant like this, the Google Style Guide for Python recommends that we use all caps or an upper case to define a variable like that. I'm going to go ahead and save this file and then go back to my other Python file and change the variable name there as well. Let me save and run this program to see if it still works. And there it is, I get the correct output one more time.

Doc Strings

So now that we know a little bit about class variables, let's move on to another idea in object oriented programming, which is that in Python, all classes come with some pre-existing class variables. One of them is called __doc__. Now this variable has got underscores on both sides of its name. Let's see this variable in action.

So here I am at the Python Shell window, with the Python prompt. And I can type a program in here like 2+2 and it gives me the correct answer. Now I want us to recall this class called turtle that we had used some time back while drawing shapes. So I can just import that class turtle here. There. Then let me see what happens when I use the class name. turtle.Turtle. Remember this was the name of the module or the file and this was the class name. So if I use the class name with the prepackaged variable called doc (turtle.Turtle.__doc__). Let me see what it prints out. I get some kind of documentation on the class turtle.

media.py - C:/OOP/movies/media.py

File Edit Format Run Options Windows Help

```
import webbrowser

class Movie():
    """ This class provides a way to store movie related information"""

    VALID_RATINGS = ["G", "PG", "PG-13", "R"]

    def __init__(self, movie_title, movie_storyline, poster_image,
                 trailer_youtube):
        self.title = movie_title
        self.storyline = movie_storyline
        self.poster_image_url = poster_image
        self.trailer_youtube_url = trailer_youtube
```

Now I wonder if I can use this variable doc with my class Movie. So, I'm back to the code for the class movie and behind this file is my other Python file where, I'm defining a whole bunch of instances of class Movie. Let me go back to the code for class Movie here for a second. Now, here, I'm going to add some documentation. At the beginning of the class Movie. You may have noticed that I use triple codes, to define my documentation. Now what I can do with triple codes, is I can create documentation in multiple lines. For now I just have the one line. So I'm going to save this file, and

entertainment_center.py - C:/OOP/movies/entertainment_center.py

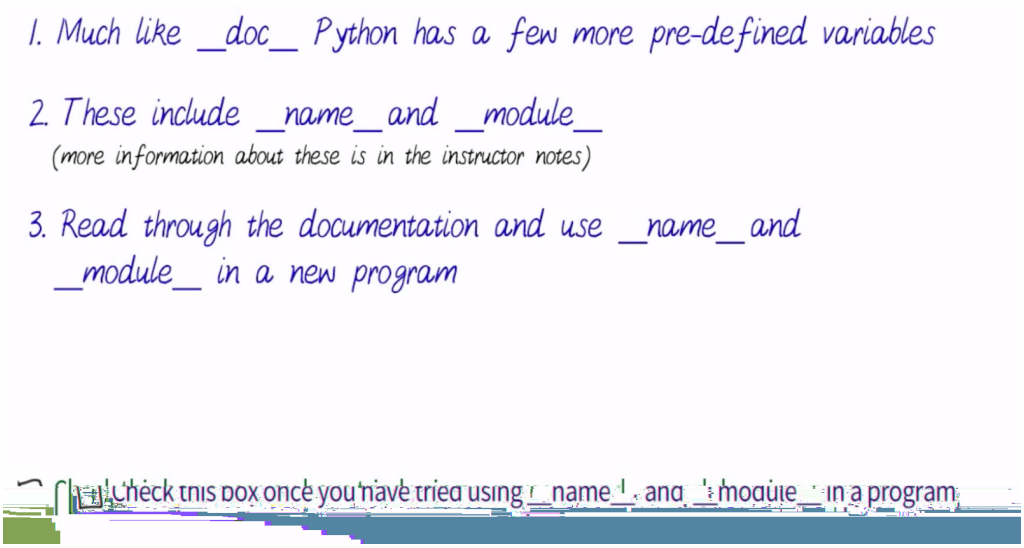
File Edit Format Run Options Windows Help

```
            "https://www.youtube.com/watch?v=c3sBBRxDAqk")

midnight_in_paris = media.Movie("Midnight in Paris", "Going back in time to meet a
            "http://upload.wikimedia.org/wikipedia/en/9/9f/Midnight_in_Paris_Poste
            "https://www.youtube.com/watch?v=atLg2wQQxvU")

hunger_games = media.Movie("Hunger Games", "A really real reality show",
            "http://upload.wikimedia.org/wikipedia/en/4/42/HungerGamesPoster.jpg",
            "https://www.youtube.com/watch?v=PbA63a7H0bo")

movies = [toy_story, avatar, school_of_rock, ratatouille, midnight_in_paris, hunge
#fresh_tomatoes.open_movies_page(movies)
#print(media.Movie.VALID_RATINGS)
print(media.Movie.__doc__)
```

then go to my other Python file. And here, I'm going to scroll all the way to the bottom, and comment out any sort of print statements. There. Now, I will try to print out the documentation for my class Movie, which is accessed by saying, media.Movie. Then I will try to print out its documentation. There. Let me save and run this file. And there it is. The documentation for my class Movie. Notice how I accessed it though. I accessed it using my class name, which is Movie and a predefined class variable, called __doc__. So, now that we have successfully used this variable called doc, I have a question for you.

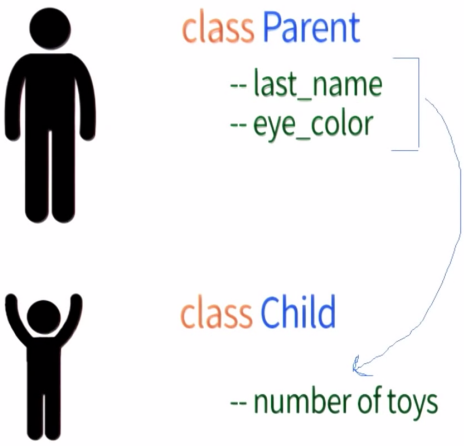
You know much like the variable __doc__, Python has a few more predefined variables these include the variable __name__ and __module__. By the way more information about these is also available

through links in the instructor notes. What I want you to do is read through the documentation in the instructor notes and then I want you to use these variables __name__ and __module__, in a new program. Once you have done that, please come back to the screen and check this box.



Inheritance

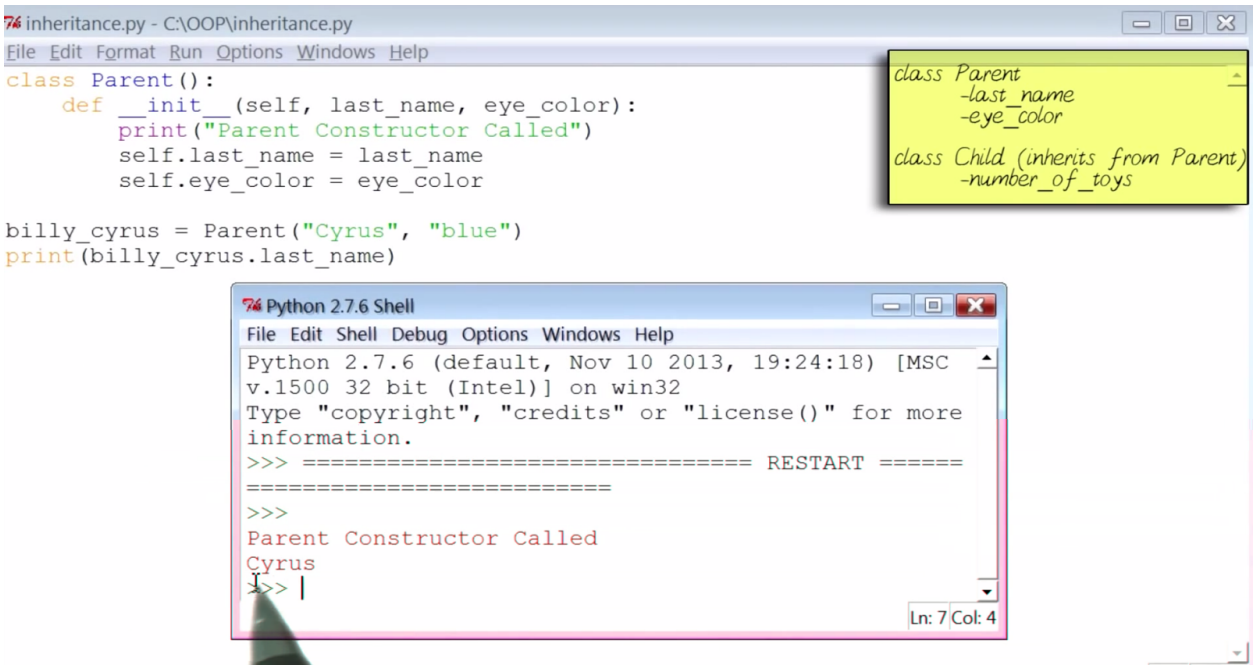
So, now that we know a little bit about class variables, both the ones that we created and the ones that come by default or predefined in Python classes, I want to take a moment to talk about the next big idea in object-oriented programming which is called inheritance. Now, let's think about the meaning of this word inheritance and how we use it in the English language. You may have heard someone say that a child can inherit some traits from their parents. Things like the last name, the eye color, some money, if the kid is lucky. So, if we were to take this meaning of inheritance, like we commonly use it in the English language, and turn it into code or model this in terms of programming, we would create a class called Parent. This would have some attributes like last name and eye color. And then we would make a class called class Child which would inherit these two things from the class Parent. Additionally, the class Child could have another variable of its own called number of toys. Now, you may notice that if we design code like this, we are already beginning to reuse code, which is a huge advantage of object oriented programming. Okay, so the next thing we are going to do is take our design and convert it into actual code. Let's do that next.



Class Parent

Okay, so here we have our design of the classes we are going to build up on the top right hand corner, and what I've done thus far is created a new Python file, and I called it inheritance.py. Now, based on our design, I'm going to go ahead and create a class called Parent. There. The next thing to do is to initialize the variables of class Parent, variables like last_name and eye_color. So, to do that, I will define this class' init method or constructor. The first argument for this function, we know, is self. And the two instance variables for class Parent, we know, are self.last_name and self.eye_color. Now the init function we know will receive a couple of values as arguments. So, let me add them in here, and we will use these two arguments to initialize our instance variables. Let me do that next. Alright. You may notice that this piece of code is pretty similar to the code for class Movie that we have written previously. By the way, one new thing I will do here, is add a quick print statement inside the init method. It will print out, Parent Constructor Called. So, this print statement will explicitly tell us, each time the init method or the constructor of class Parent is called.

Alright, to make sure that this code that we have written thus far actually works, let's go ahead and use this class. I will define an instance of class Parent and call it billy_cyrus, and will provide the two arguments that are necessary. The first of which is last_name and that happens to be Cyrus, and the next one is eye_color which is, let's say, blue. Now, a quick word of caution. Ordinarily, I would have kept these two things, the definition of the class Parent and creating its instances. I would have kept them in separate Python files. But I have kept them both here in the same file for ease of demonstration. Okay, so to demonstrate that the instance actually works, I'll print out its last name.



Let me save and then run this file. And there's my output. It says the parent constructor was called. Which seems appropriate, because we created an instance called billy_cyrus. And as soon as we did that, the class Parent's _init_ method got called, which had a print statement inside it. And then, we printed out Billy Cyrus's last name, which it printed out correctly.

Okay, so far so good. Now, you'll notice that there really isn't anything new that we've done here thus far. The new thing we're going to try is called inheritance, which will happen when we create the class called Child. Let's do that next.

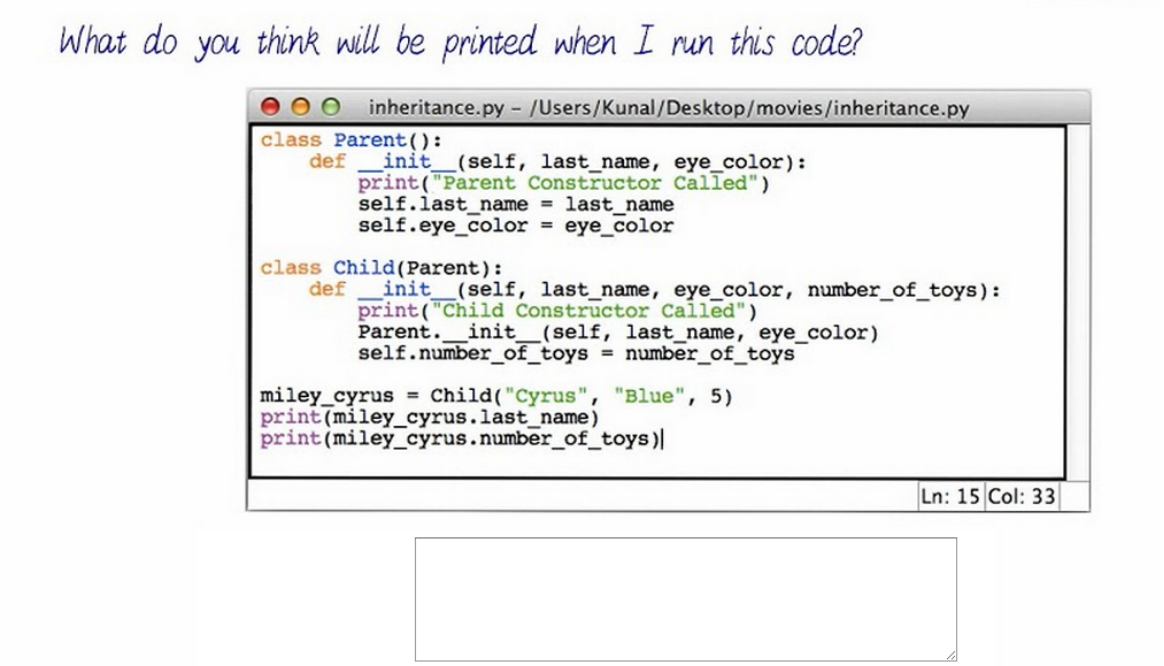
What's the Output

Okay, so here is the code for class Parent and now I will start to define class Child. Now, I know class Child inherits from class Parent and the way to indicate that in Python is to say this. The syntax here means that class Child will now inherit or reuse everything that is publicly available in class Parent.

Now, things will get really interesting when we begin to define the init method, or the constructor for this class. So, we know that class Child has three variables. last_name and eye_color that are inherited from class Parent. And this other variable called number_of_toys. So, in addition to the keyword self, I will receive those values as arguments right here in the init function; there they are.

Now, to initialize the variables I'm inheriting from class Parent, variables like last_name and eye_color, I will actually reuse class Parent's init method. This is how we do it. The last instance variable, number_of_toys, will be initialized next. Okay, finally, I will add a print statement to the very beginning of the init method.

All right, now that we have defined our class Child, the next thing to do is to create an instance of this class Child. I will call it, appropriately enough, miley_cyrus. And here, I will provide it the three necessary arguments. And then, I will print out two things. Her last name, and the number of toys. Also, so I can focus on these print statements in my output, I will comment out the previous instance for now. There. Let me save this file. Now, before I run this program, I want you to think about what the output of this program will be. So, here is the code one more time. Here is class Parent, and here is class Child, and here, we are creating an instance of class Child, and we named it miley_cyrus. So, what do you think will be printed when I run this piece of code? Enter your responses in this box.



Class Child

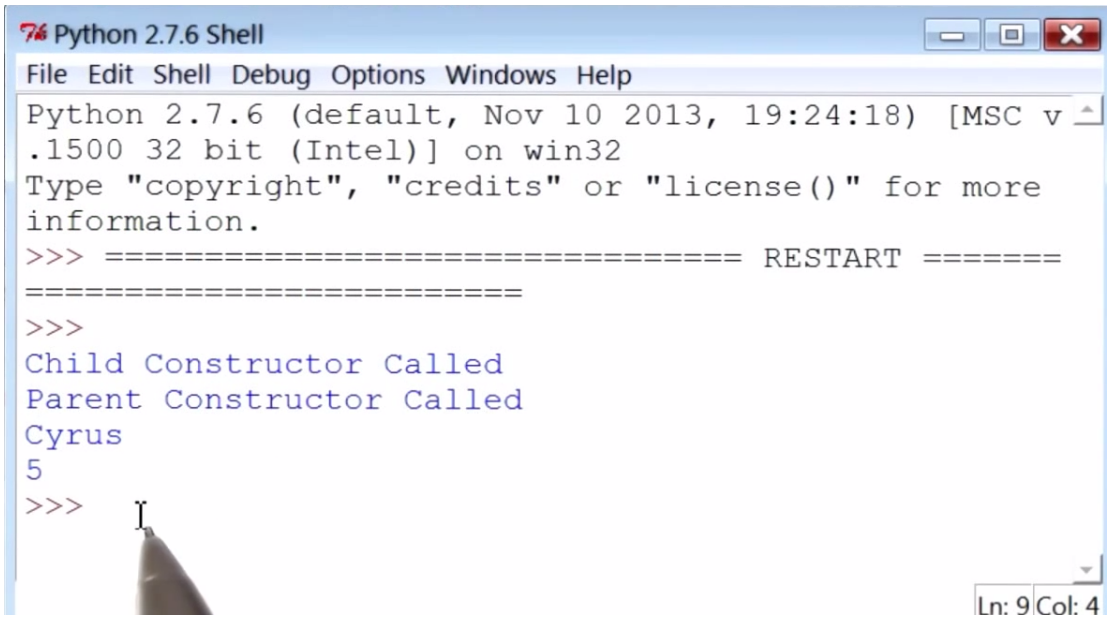
So before I run this program, let me share with you my hypothesis of what will happen when I run this piece of code.

Now one of the first things we are doing here in this program, is that we're creating an instance of class Child we called it miley_cyrus. As soon as that line of code runs, the init method inside class Child will get called. The first line within the init method, is this print statement. So my hypothesis is, that this print statement will be printed out first in our output.

After that the constructor for the class Parent is going to get called. So the control will move from here up to here. When the init method for class Parent is called, this statement is going to get executed. So my hypothesis is that this print statement “Parent constructor called” will be the second thing that gets printed.

Then the instance variables last_name and eye_color will be appropriately initialized. Once the init method for class Parent has successfully run, the control will come back here. At that point, the instance variable number_of_toys will successfully get initialized. So at that point, the init method for class child is done, which means that the instance miley_cyrus has been created successfully.

Then, the following two print statements which are trying to print the last_name and number_of_toys for miley_cyrus will get executed in that order. Alright, this time let me actually run this piece of code. And there's my output. And it seems pretty close to what I thought it would be. Now, I want you to pause the video here for a second, and I want you to see if the output here matches your hypothesis.



Transitioning To Class Movie

Ok, so we've just seen how inheritance can help us reuse code. In our example, we saw that class Child can reuse some of the code written in class Parent. Now I want to use the same technique of inheritance, to see if I can improve my design for class Movie that I wrote previously. Let's do that next.

Updating the Design for Class Movie

So in a previous lesson we created a class called Movie. Now this class had the following attributes, the movie's title, its storyline, a link to the movie's poster image, and a link to its YouTube trailer. Now in addition to these things, the class movie also had a function called show_trailer. After we had defined this class Movie we created several instances of this class. Instances like Toy Story and Avatar.

Now further imagine that we wanted to create another class called TvShow. I would think that this class would have details like the title of the show, its season and episode number. And also, the TV station that this show is aired on. Additionally, this class could also have a function called, get_local_listings. Once we've created a class called TvShow, we can create multiple instances of this class, instances like, season one, episode one of Breaking Bad, and the final episode of Seinfeld. Alright, so if we continue our thought experiment here, we can further imagine that there can be several items that both of these classes can share amongst each other. Things like title for sure, also things like duration of the movie and duration of the TV show in minutes.

So the best way to structure this code would be to create another class called Video, which would have two attributes. The video's title, and the video's duration. And the class Movie, could inherit from this class Video. To do this, we would have to add the class name Video, inside these parentheses. Now, this would mean that class Movie, would inherit title and duration, from class Video.

Additionally, the class TvShow could also inherit from class Video. To do this we would also have to add the class name Video inside these parentheses. Now this would mean that class TvShow would inherit title and duration from class Video. Now you can clearly see how we can write a piece of code, in this case, class Video, and continue to reuse it in multiple different places. Another big benefit of writing code this way, in addition to just reusing code, is that it allows us to follow an intuitive model of how things exist in our brain. So intuitively speaking, we know what videos are, we also know what TV shows and movies are. So writing code in this way allows programmers to map how things exist in our brain onto code.

Reusing Methods

Okay, so thus far, we've seen how inheritance can help us with reusing instance variables. Now I want to show you an example of how inheritance can help with reusing methods. So here we are back at the code, where we have a class called Parent and a class called Child. And if you recall, class Child inherits from class Parent.

Now, I'll begin by defining a simple instance method inside class Parent. And I will call it show_info. The first argument of this method is self, and all this method does is it prints out the last name and eye color of the parent. Alright, there are the two print statements.

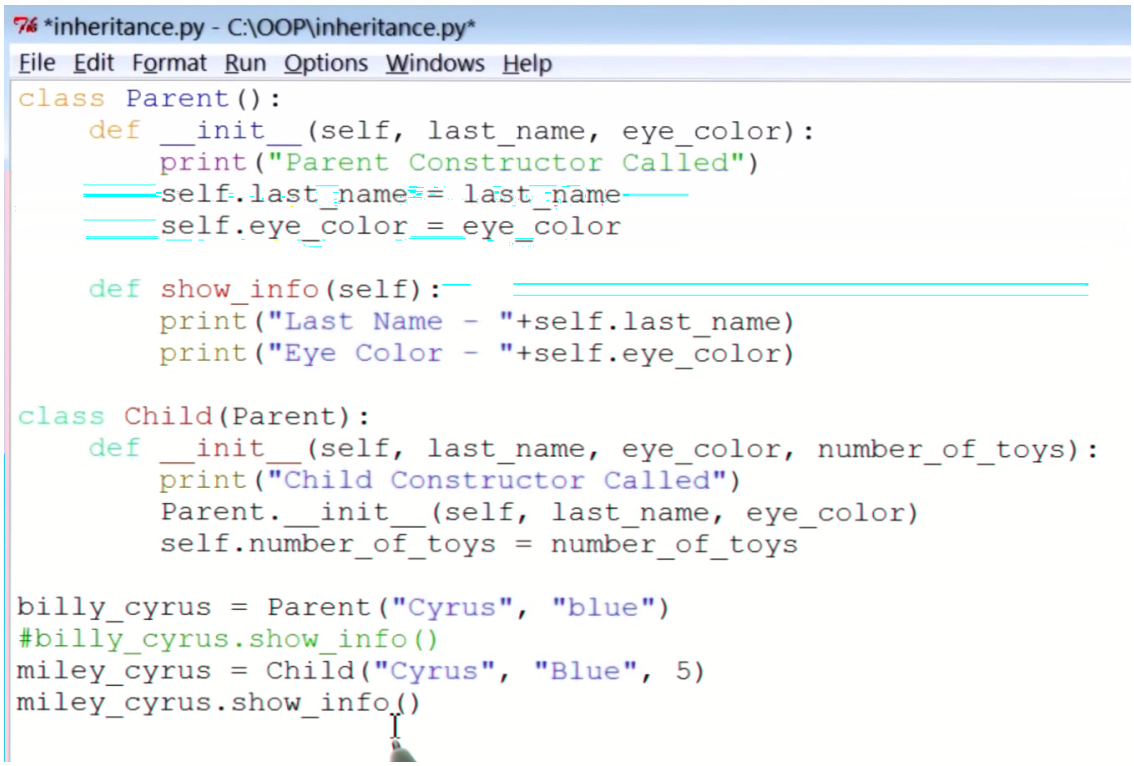
Now to test to see if this method actually works, I'm going to call this method show_info using the parent's instance billy_cyrus. So let me do that next. So there is that method. Now all I've done thus far is created a new method called show_info inside class Parent. And then used an instance of class Parent, instance called billy_cyrus to call that method. Now to be able to focus on this statement's output, I am going to comment out the other instance statement for now. There. Let me go ahead and save and run this program. Alright. So the program prints out the correct values of billy_cyrus' last name and eye color. So far so good.

Method Overriding

Alright so I'm going to come back to the code, and this time I'm going to do something new. Now because this class Child inherits from class Parent this method show_info is also inherited. Now this means that instances of class Child instances like miley_cyrus, they can also call the show_info method, even though this method is not explicitly defined inside class Child. Let me do that next.

So the first thing I will do is uncomment this line of code where I'm creating the instance called miley_cyrus, and then use that instance, to call the method show_info. There. Now because I want to focus on the output of this show_info method, I'm going to comment out the previous show_info method for now. There. Now before I run this program, I want to highlight one more time that I'm using the instance miley_cyrus to call the method show_info. Now this method does not explicitly

exist inside class Child. But because class Child inherits from class Parent this method show_info is actually available to the instances of class Child. So let me go ahead and save and run this program. And there is my output. The program did not throw any errors, and the last name and eye color of the instance miley_cyrus did get printed out. Okay. So the next thing I am going to do is create a method called show_info inside class Child, and see what that does.



So I'm going to create a new method called show_info inside class Child. Now this method show_info is going to print out the three things associated with a child. It's last name, eye color, and number of toys. Here we are printing the last name of the child. Now we are printing the child's eye color, and finally, we are printing the number of toys. In order to print this number correctly, we had to wrap it around the string function.

Now, if I save and run this program, my hypothesis is that when I call the show_info method using miley_cyrus, which by the way is an instance of class Child. This show_info method is going to get called and not the show_info method in the class Parent. So let me save and run this program. And there's the output. We're printing out the last name, eye color and number of toys of the instance of class Child. So really what we have done here is defined the method by the same name show_info inside class Child, and also inside class Parent. This ability of a subclass, in this case the class Child, to override a method that it inherited from its parent class is called Method Overriding in programming.

Next Stop - Final Project

So we've talked about several different advanced object oriented programming ideas thus far, ideas like inheritance, and also method overriding. I want to thank you for your patience thus far, and also for all of your hard work. And I really think that we are ready to attempt the final project now. And that final project is next.