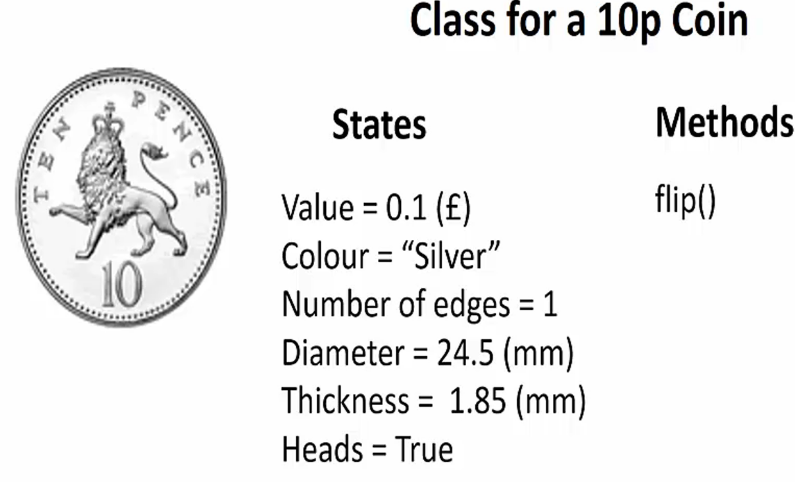
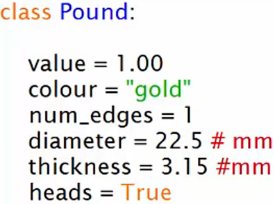
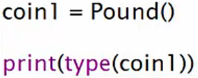
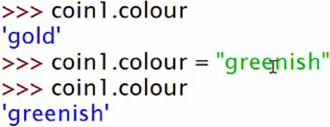
**Section Overview**  
\* **OOP is an incredibly powerful and popular programming discipline that helps us make codified representations of real world objects**.

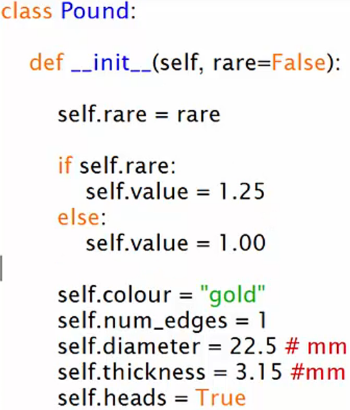
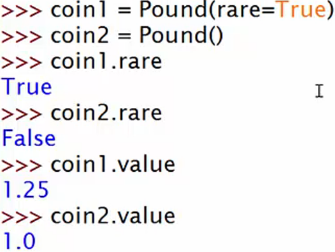
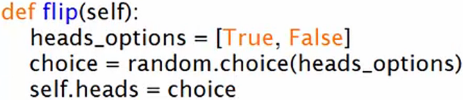
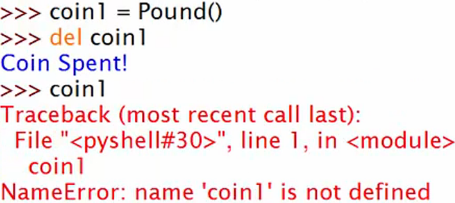
**Objects and Classes: What are they?**  
\* **Every object is just a physical copy - instance - of a class**.  
\* **Classes are just templates - we instantiate a class to make an object -> an object is an instance of a class**.  
**State** => **Class Variable**  
**Method** => **Class Function**  


**PROJECT 9 - Make your own Coin! - Part 1**  
**class**  
\* **Class names in Python start with an uppercase letter by convention**.  


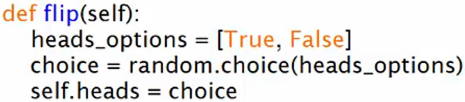


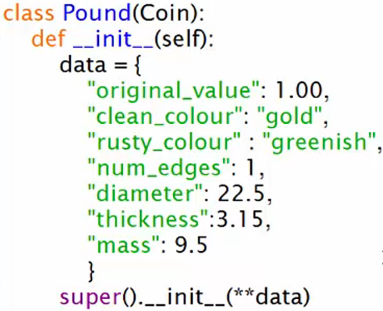
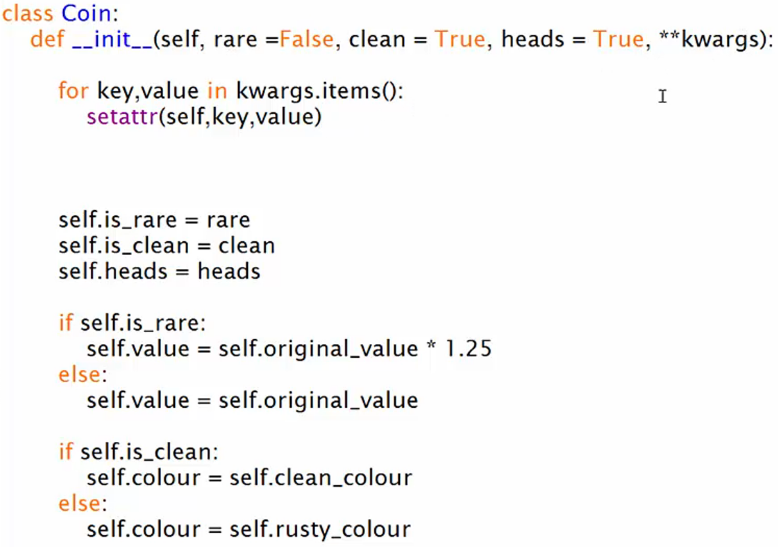


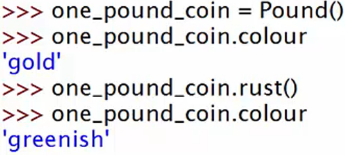


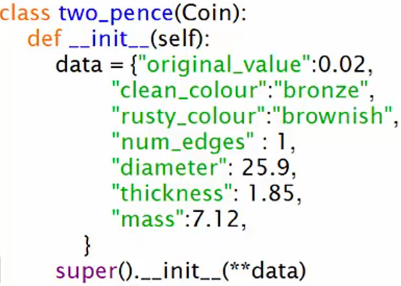
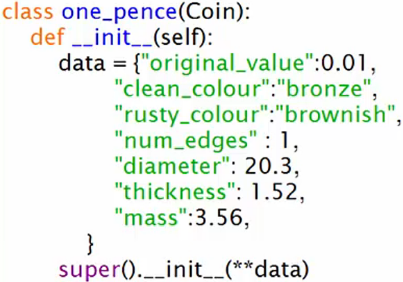
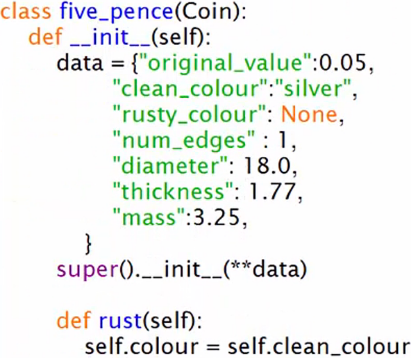
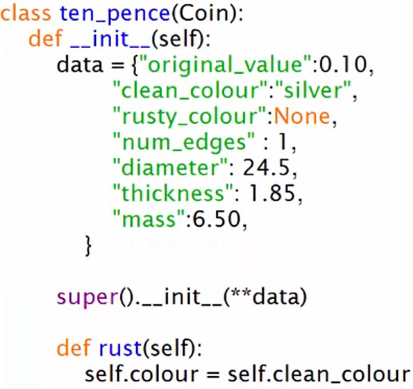
**PROJECT 9 - Make your own Coin! - Part 2**  
**constructor** **def \_\_init\_\_(self)**  
**self** => **refers to a specific instance of the class when we write the class code, it’s a convention to call it `self`**.  
=> **Constructor doesn’t return anything**.  
   
=> **Effectively constructing an object’s states**.  
\* **The 1st parameter to any class method has to be `self` or whatever you decide to call it in the \_\_init\_\_()**.  
  
  
  
**destructor** => **def \_\_del\_\_(self)** => it’s called automatically when our program finishes and is used to destroy an object.  
**del**  
  
  
**Encapsulation** => We saw that coin1 and coin2 could have different internal states to each other, one could be rusty and one could be normal and it doesn’t matter what the other one is doing. This is the Object Oriented Programming principle of Encapsulation - where your states can be encapsulated inside a bubble and hidden from each other.  
\* Next up, we’ll learn about Inheritance and Polymorphism.

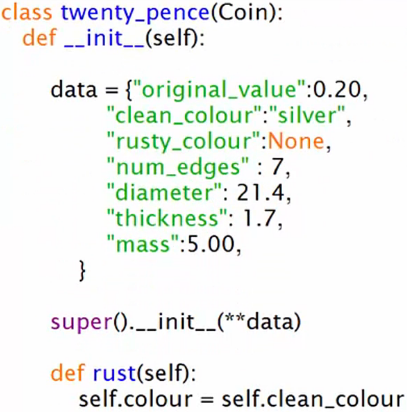
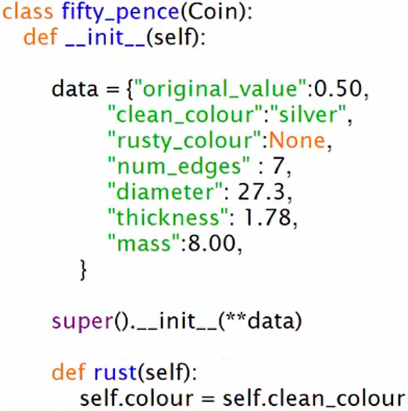
**PROJECT 10 - Make all the coins!! - Part 1**  

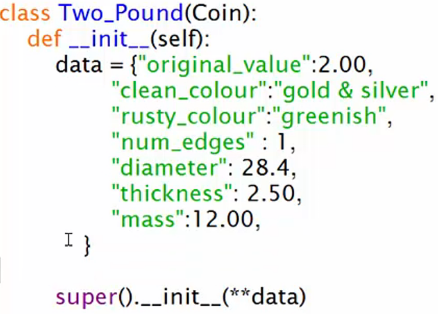
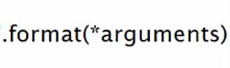
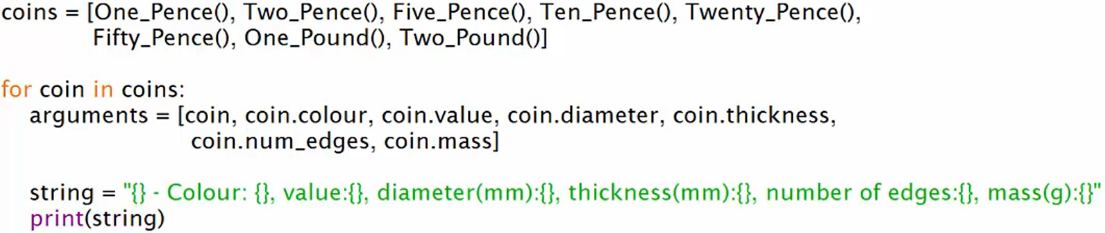
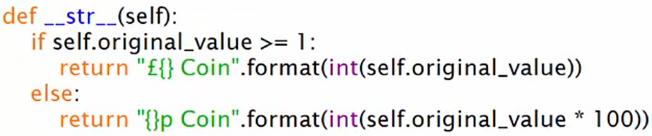
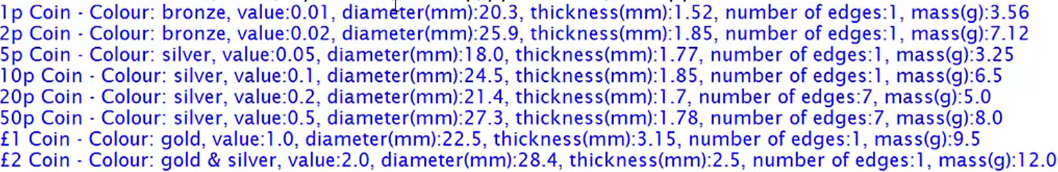



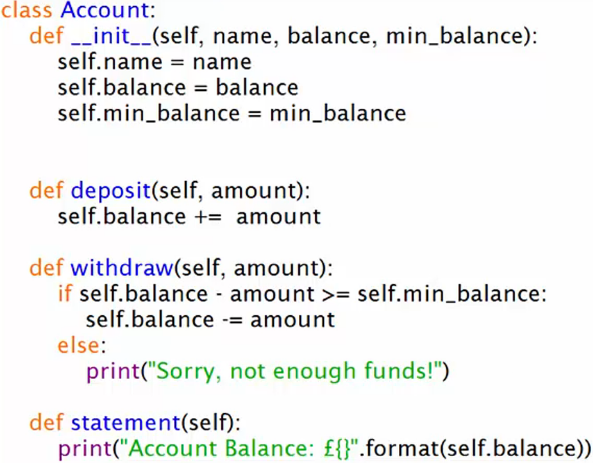
**Inheritance** => **class Child(Parent):**  
**super() => the parent class.**  
\* **The only thing that’s specific about each coin is its data**.  
**setattr(self, key, value)**  
 

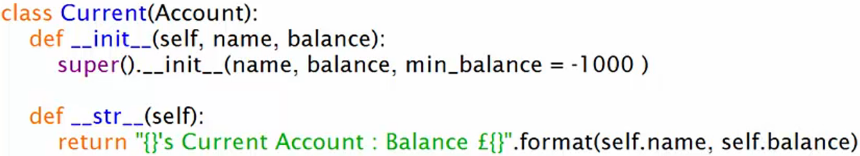


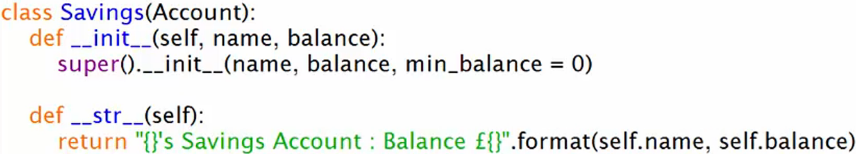
**PROJECT 10 - Make all the coins!! - Part 2**  
\* **Polymorphism after Inheritance, Abstraction and Encapsulation is the final principle of OOP.  
=> Polymorphism to change little pieces of a coin’s behavior and allow some coins to rust and others not.**  
  
\* For any silver coin, we need to override the standard behavior for the rust and clean methods because **silver coins don’t rust**.  
=> **We need to override the standard behavior that we’re getting from our parent class**.  
**Polymorphism** => **when you have the same function name but it has different forms, different shapes, it has different versions of itself, that’s called Polymorphism - Poly means multiple and Morph means form, so when a method has multiple forms inside a class, that’s Polymorphism**.  
 

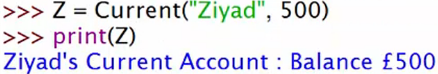
 

**!!! The classes should’ve had names capitalized => One\_Pence, Two\_Pence  
!!! The classes should’ve had names capitalized => One\_Pence, Two\_Pence  
!!! The classes should’ve had names capitalized => One\_Pence, Two\_Pence**  
\* **Just by setting up an Abstract Class - one that’s really general, all we’ve got to do to create a new coin is to give a different dictionary and we can have completely different behavior**.  
  
  
=> The weird object stuff is what happens when you just print the object.  
**\_\_str\_\_()** => **defines what comes out when you print that object**.  
  


**PROJECT 11 - Make Your Own Bank!**  
\* Account will be our **Abstract Parent Class**  







**Section Review**