Week 5 Video Programs

**What is Object-oriented programming? (Optional)**

 Attributes are characteristics associated with a type. Methods are:



Lists associated with a type



Instances of a class



**Functions associated with a type**

**Correct**

**Right on! Remember, a method defines what you do with an object.**

**Functions associated with a type**

**is selected.This is correct.**

**Right on! Remember, a method defines what you do with an object.**



Characteristics associated with a type

#### Classes and Objects in Python (Optional)

**You want to find more information about the integer (int) class. What’s the best way to do this?**



**Use the command help(int)**

**Correct**

**Nailed it! Using the help command can be useful for finding quick documentation about the methods in a class.**

**Use the command help(int)**

**is selected.This is correct.**

**Nailed it! Using the help command can be useful for finding quick documentation about the methods in a class.**



Use the command type(int)



Type ‘q’



Use the len() function

#### Defining New Classes (Optional)

Want to give this a go? Fill in the blanks in the code to make it print a poem.



**class Flower:**

**color = 'unknown'**

**rose = Flower()**

**rose.color = "red"**

**violet = "blue"**

**a="this\_pun\_is\_for\_you"**

**print("Roses are {},".format(rose.color))**

**print("violets are {},".format(violet))**

**print(a)**

**Here is your output:**

**Roses are red,**

**violets are blue,**

**this\_pun\_is\_for\_you**

**Awesome! Very nice poem, if not a little cliche!**

#### Instance Methods (Optional)

OK, now it’s your turn! Have a go at writing methods for a class. Create a Dog class with dog\_years based on the Piglet class shown before (one human year is about 7 dog years).



class Dog:

years = 0

def dog\_years(self):

return self.years \* 7

fido=Dog()

fido.years=3

print(fido.dog\_years())

**Here is your output:**

**21**

**Awesome! You've now learned about writing your own methods!**

#### Constructors and Other Special Methods (Optional)

Want to see this in action? In this code, there's a Person class that has an attribute name, which gets set when constructing the object. Fill in the blanks so that 1) when an instance of the class is created, the attribute gets set correctly, and 2) when the greeting() method is called, the greeting states the assigned name.



**class Person:**

**def \_\_init\_\_(self, name):**

**self.name = name**

**def greeting(self):**

**# Should return "hi, my name is " followed by the name of the Person**

**.**

**return "hi, my name is {}".format(self.name)**

**# Create a new instance with a name of your choice**

**some\_person = Person("Sherwin")**

**# Call the greeting method**

**print(some\_person.greeting())**

**RunReset**

**Here is your output:**

**hi, my name is Sherwin**

**Right on! You have successfully learned to assign attributes**

**to instances of class objects!**

#### Documenting Functions, Classes, and Methods (Optional)

Remember our Person class from the last video? Let’s add a docstring to the greeting method. How about, “Outputs a message with the name of the person”.



**class Person:**

**def \_\_init\_\_(self, name):**

**self.name = name**

**def greeting(self):**

**"""Outputs a message with the name of the person"""**

**print("Hello! My name is {name}.".format(name=self.name))**

**help(Person)**

**Here is your output:**

**Help on class Person in module submission:**

**class Person(builtins.object)**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self, name)**

**| Initialize self. See help(type(self)) for accurate signature.**

**|**

**| greeting(self)**

**| Outputs a message with the name of the person**

**|**

**| ----------------------------------------------------------------------**

**| Data descriptors defined here:**

**|**

**| \_\_dict\_\_**

**| dictionary for instance variables (if defined)**

**|**

**| \_\_weakref\_\_**

**| list of weak references to the object (if defined)**

**Excellent! You’ve mastered the art of providing info using**

**docstrings!**

#### Inheritance (Optional)

Let’s create a new class together and inherit from it. Below we have a base class called Clothing. Together, let’s create a second class, called Shirt, that inherits methods from the Clothing class. Fill in the blanks to make it work properly.



**class Clothing:**

**material = ""**

**def \_\_init\_\_(self,name):**

**self.name = name**

**def checkmaterial(self):**

**print("This {} is made of {}".format(self.name,self.material))**

**class Shirt(Clothing):**

**material="Cotton"**

**polo = Shirt("Polo")**

**polo.checkmaterial()**

**Here is your output:**

**This Polo is made of Cotton**

**Nice work! You used the existing Clothing class to make a**

**Shirt class that inherited from it!**

#### Composition (Optional)

Let’s expand a bit on our Clothing classes from the previous in-video question. Your mission: Finish the "Stock\_by\_Material" method and iterate over the amount of each item of a given material that is in stock. When you’re finished, the script should add up to 10 cotton Polo shirts.



**class Clothing:**

**stock={ 'name': [],'material' :[], 'amount':[]}**

**def \_\_init\_\_(self,name):**

**material = ""**

**self.name = name**

**def add\_item(self, name, material, amount):**

**Clothing.stock['name'].append(self.name)**

**Clothing.stock['material'].append(self.material)**

**Clothing.stock['amount'].append(amount)**

**def Stock\_by\_Material(self, material):**

**count=0**

**n=0**

**for item in Clothing.stock['material']:**

**if item == material:**

**count += Clothing.stock['amount'][n]**

**n+=1**

**return count**

**class shirt(Clothing):**

**material="Cotton"**

**class pants(Clothing):**

**material="Cotton"**

**polo = shirt("Polo")**

**sweatpants = pants("Sweatpants")**

**polo.add\_item(polo.name, polo.material, 4)**

**sweatpants.add\_item(sweatpants.name, sweatpants.material, 6)**

**current\_stock = polo.Stock\_by\_Material("Cotton")**

**print(current\_stock)**

**Here is your output:**

**10**

**Nice job! You successfully used composition to reuse the**

**Clothing.stock attribute and stock\_by\_material() function of**

**the Clothing class to take stock of the Cotton shirts!**

#### Python Modules (Optional)

Let’s say we want to use the Keras deep learning module. Upon running the script, an error is printed stating the Keras module could not be found. What might we have missed?



We need to initialize the timedelta class



We need to define Keras functions



We need to define Keras attributes



**We need to import the Keras module**

**Correct**

**You got it! We must use the import keyword to import the module before it can be used.**