Name:_____ **Partner:** _____

Python Activity 38: Classes - Inheritance

When subclasses inherit features from parent classes, it reduces code redundancy!

Learning Objectives

Students will be able to:

Content:

- Explain how **parent** and **child classes** are relevant to **class inheritance**
- Predict how methods and attributes will be inherited by child/sub-classes

Process:

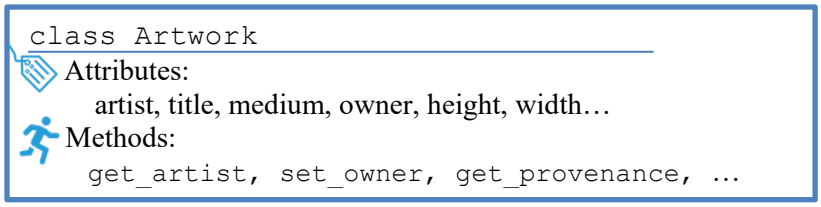
- Build sub-classes that inherit attributes and methods from the parent class.

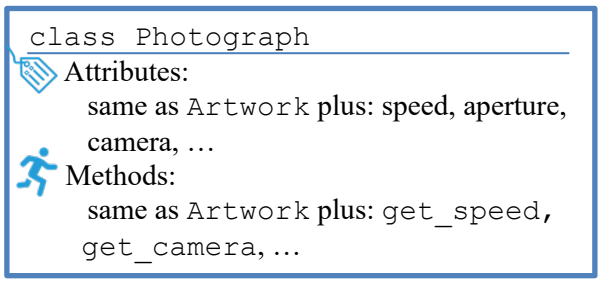
Prior Knowledge

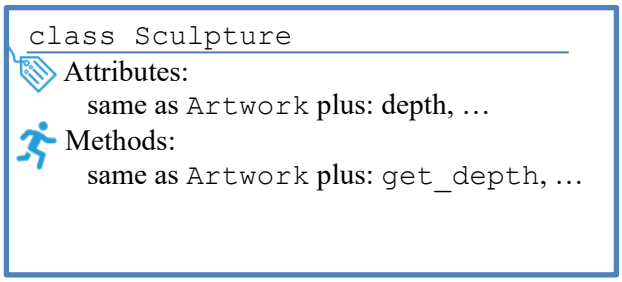
- Python concepts: user-defined classes, attributes, methods, special methods

Concept Model:

We want to build a data structure to represent the artworks at the local art museum. Before we begin programming, we should think about the different kinds of artwork, and how they are similar or different from one another:







CM1. What are some of the common *attributes* shared by all three of these classes?

_____

CM2. Does the Sculpture class have a `speed` attribute? _____

CM3. What are some of the common *methods* shared by all three of these [related] classes?

_____

CM4. Does the Artwork class have a `get_depth()` method? _____

CM5. If we were to implement these three classes with our current concepts so far, how would we have to implement the shared attributes & methods in each of our classes?

_____

Critical Thinking Questions:

1. Examine the following code below, which represents two of the classes from our *Concept Model*.

artwork.py

```
0  class Artwork:

1      def __init__ self, title):
2          self._title = title


3      def get_title(self):
4          return self._title

5      def __str__(self):
6          return self._title
```

```
7   class Sculpture(Artwork):

8      def __init__(self, title, depth):
9          super().__init__(title)
10         self._depth = depth

11     def get_depth(self):
12        return self._depth

13     def __str__(self):
14         return self._title + str(self._depth)
```

```
24  if __name__ == "__main__":
25     tt = Sculpture("The Thinker", 140)
26     print(tt.get_depth())
27     print(tt.get_title())
28     print(tt)
```

a. Circle the class syntax that is new to us.

b. Fill in the blank: `tt` is an *instance* of a _____ object.

c. What do you think will be displayed by line 26?

_____

d. What do you think will be displayed by line 27?

_____

Line 27 displays `The Thinker`. How might Python know `tt`'s `_title` attribute?

_____

e. What do you think `super().__init__(title)` on line 9 is doing?

_____

What might happen if we remove the reference to `(Artwork)` on line 7?

_____

f. What do you think will be displayed by line 28? _____

Line 28 displays `The Thinker140`. What method must `print(tt)` be calling?

_____

How might python know which `__str__` method to execute?

_____

2. Examine the following code below, that extends our previous code:

artwork.py (continued)

```
15 class Photograph(Artwork):

16     def __init__(self, title, cam, spd):
17         super().__init__(title)
18         self._camera = cam
19         self._speed = spd

20     def get_camera(self):
21         return self._camera

22     def get_speed(self):
23         return self._speed
```

a. Write a line of code to create a new instance of a `Photograph` object:

_____

b. If we were to `print` this new `Photograph` object, what might the output be?

_____

The output would be *just the Photograph object's title*. Which method is likely being called when we `print` this new `Photograph` object?

_____

Compare your response to the response in Q1f. Why are they different?

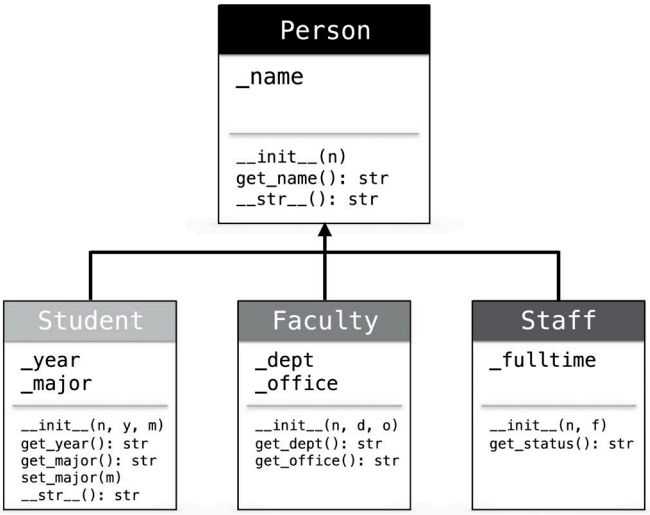
_____

c. Explain how Python chooses when to execute a method from a *child class* versus its *parent class*:

_____

Application Questions: Use Python to check your work

1. Implement a new set of classes and sub-classes that inherit from one another, according to the following class diagram:



a. Create a new instance object for each of the newly defined classes:

b. Write some lines of code to use some of the shared and some of the specific-to-subclasses methods: