Announcements

- Homework
  - I'm going to get HW8 posted this weekend in case you want to get a jump start on it over break
  - 3 fairly short problems
  - We are going to make it due that Friday we get back from break
- Plan is for the test the Friday after break as well
- Credit/Non Credit option has also been extend to that Friday
- There is a lab today, so I'll be hanging out in the Zoom meeting afterwards if questions come up that you want to ask!
  - You can always share your screen in there to let me see what you are seeing as well!
- Polling: `rembold-class.ddns.net`

Review Question

We are wanting to move into learning about how to define our own types in Python. Doing so though means understanding what a Python type entails. What two main ideas should be included in any definition of a new type?

A) An internal representation and an interface to manipulate type objects
B) The ability to store non-scalar objects and an interface to manipulate those objects
C) The ability to store non-scalar objects and to be garbage collected by the interpreter
D) An internal representation and the ability to be garbage collected

Solution: An internal representation and an interface to manipulate type objects

Defining our Type

- Use the `class` keyword to define a new type

```
class Coordinate:
    # all attributes go here
```

- Python convention is to start class names with a capital letter
- All internal parts of the class definition should be indented
- Can occasionally have a term in parentheses after the type name. We'll talk about that later when we discuss inheritance.

Attribute who?

- Attributes are data and procedures/interactions that "belong" to a particular class
- Data Attributes
  - Data as other objects that make up the class
  - Commonly the pieces you need to construct your desired class
  - Example: A coordinate is made up of two numbers
- Methods (Procedures/Interfaces)
  - Methods are functions that only work with the class
  - How you get to interact with the class objects
  - Example: You could define a distance function between two coordinate points. Such a function would have no meaning outside of working with coordinates.

Class Construction: Representation

- Step 1: Defining how we construct on new object of this type
  - What makes up the object? How is it stored? What values do we pass in and what values are constant?
  - Define a special method called `__init__` to initialize some data attributes

```
class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Class Construction: Representation

- Step 1: Defining how we construct on new object of this type
  - What makes up the object? How is it stored? What values do we pass in and what values are constant?
  - Define a special method called `__init__` to initialize some data attributes

```
class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

`__init__`

Special method to create an instance. Starts and stops with a double underscore.

Class Construction: Representation

- Step 1: Defining how we construct on new object of this type
  - What makes up the object? How is it stored? What values do we pass in and what values are constant?
  - Define a special method called `__init__` to initialize some data attributes

```
class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

self

parameter with refers to an instance of the class

Class Construction: Representation

- Step 1: Defining how we construct on new object of this type
  - What makes up the object? How is it stored? What values do we pass in and what values are constant?
  - Define a special method called `__init__` to initialize some data attributes

```
class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

`x,y`

Parameters which get passed to the Coordinate object upon initialization

Class Construction: Representation

- Step 1: Defining how we construct on new object of this type
  - What makes up the object? How is it stored? What values do we pass in and what values are constant?
  - Define a special method called `__init__` to initialize some data attributes

```
class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

`.x, .y`

Each coordinate object gets two data attributes

Class Construction: Initialization

- Like functions, the class definition *only* defines the class
- Still need to use it in your code somewhere by initializing an object of your new type!

```
c1 = Coordinate(2,4)
c2 = Coordinate(1,0)

print(c1.x)
print(c2.y)
```

- We use the dot notation to access attributes of any instance
- You do not need to provide an argument for `self`. Python does this automatically.

Understanding Check

What is printed out on the final line of the code to the right?

A) `Honda red 2006`

B) `Honda blue 2006`

C) `Toyota blue 2008`

D) `Honda red 2008`

```
class Car:
  def __init__(self, color, year):
    self.color = color
    self.year = year
    self.make = 'Toyota'

A = Car('blue', 2008)
B = Car('red', 2006)
A.make = 'Honda'
A.year = B.year
print(A.make, A.color, A.year)
```

Solution: `Honda blue 2006`

What's your Method?

- ☑ We can create data attributes now to represent our type
- ☐ Still need a way to interface and manipulate our objects
- Need to include methods or procedural attributes to our classes
  - Basically a function that only works with this class
  - Python always passes the object as the first argument!
    - Convention to is always use `self` to refer to the instanced object within the class definition
- Still access or call methods using dot notation
- Other than `self` and dot notation, methods act just like functions

Class Construction: Adding Methods

```
def method(self, other_inputs):
    return self.data_attribute + other_inputs
```

Class Construction: Adding Methods

name of method

```
def method(self, other_inputs):
    return self.data_attribute + other_inputs
```

Class Construction: Adding Methods

use to refer to any instance

```
def method(self, other_inputs):
    return self.data_attribute + other_inputs
```

Class Construction: Adding Methods

any other parameters the method might need

```
def method(self, other_inputs):
    return self.data_attribute + other_inputs
```

Class Construction: Adding Methods

```
def method(self, other_inputs):
    return self.data_attribute + other_inputs
```

normal function parts

Accessing and Using Methods

- After definition, you have two main ways to access and use the method
- Dot Notation (Conventional):

```
c = Coordinate(3,4)
O = Coordinate(0,0)
print(c.distance(O))
```

- Function Notation:

```
c = Coordinate(3,4)
O = Coordinate(0,0)
print(Coordinate.distance(c,O))
```