Object Oriented Programming Notes

These notes were adapted from Jon Kotker's OOP lecture from Summer 2012. Thanks Jon!

- Objects are data structures that are combined with associated behaviors.
- They are "smart bags" of data that have state and can interact.
- Functions can do one thing; objects can do many related things.
- Previously, functions "disappear" after they were called. Now, with OOP, we can call functions and store the values they return within the object itself.

The basics:

Every **person** (Object) is a **Human** (Class)
    Classes are basic templates for objects
    An **object** is an instance *of a* **class**
    A **person** is an instance *of a* **human**
A **single person** (instance) has a **name** (Instance variable) and **age** (Instance variable)
    Objects and instance variables have a "*has-a*" relationship
    An instance variable is an attribute specific to an instance.
    An **object** *has an* **instance variable**.
    A **person** *has a* **brain**
A single person can **eat** (method) and **sleep** (method)
    Methods describe a certain "behavior" of an object
The **population** (class variable) of the Earth is 7 billion
    Class variable: attributes for the class as a whole, shared by all instances of a class

```
class Human:

    population = 7,000,000 #class variable

    def __init__(self, name, age):
        self.name = name #instance variable
        self.age = age #instance variable

    def eat(self, food): #method!
        print("mmm, I love eating " + str(food))

    def sleep(self): #method
        print("zzZZZzzzZZZ")
```

Pokemon

```
Class Pokemon:
	total_pokemon = 0 #_______________
	def __init__(self, name, owner, hit_pts):
		self.name = name #_______________
		self.owner = owner
		self.hp = hit_pts
		Pokemon.total_pokemon += 1
		#  Class variables are referenced using the name of the class since they don't belong to a specific instance

	def increase_hp(self, amount):
		self.hp += amount

	def decrease_hp(self, amount):
		self.hp -= amount
		if self.hp < 0:
			self.hp = 0

	def get_name(self): #selector
		return self.name

	def get_owner(self):
		return self.owner

	def get_hit_pts(self):
		return self.hp
```

Note: Every method needs self as an argument! This allows you to reference a specific instance of the class
Think of it like this: how else will you reference the instance uniquely?

```
>>> ashs_pikachu = Pokemon('Pikachu', 'Ash', 300)

>>> mistys_togepi = Pokemon('Togepi', 'Misty', 245)
```

The above two statements *instantiate* new objects
When you instantiate a new object, the __init__ method of the class is called.
Objects can only be created by the constructor
We've created two new objects! Each of which have their own set instance variables (name, owner, hp) and **bound** methods (increase_hp, decrease_hp, get_name, get_owner, get_hit_pts)

- Bound methods are methods **bound** to the instance

```
>>> mistys_togepi.get_owner() # Alternatively, Pokemon.get_owner(mistys_togepi)


>>> ashs_pikachu.get_hit_pts()


>>> ashs_pikachu.increase_hp(150) # Alternatively, Pokemon.increase_hp(ashs_pikachu, 150)
>>> ashs_pikachu.get_hit_pts()
```

Write a method `attack` that takes another Pokemon object as an argument.

When the method is called on a Pokemon object, the object screams (prints!) its name and reduces the HP of the opposing Pokemon by 50.

```
def attack(                              ):
        """
        >>> mistys_togepi.get_hp()
        >>> ashs_pikachu.attack(mistys_togepi)
        Pikachu!
        >>> mistys_togepi.get_hp()
        195
        """
        # YOUR CODE HERE
```

Inheritance

There are several different types of Pokemon which differ in the amount of points lost by its opponent in an attack.

The only method that changes from one type of Pokemon to another is the attack method. Everything else stays the same! We want to avoid duplicating code.

The key idea of inheritance is that classes can inherit methods and instance variables from other classes.

```
class WaterPokemon(Pokemon): # the Pokemon class is the superclass of the WaterPokemon class

	def __init__(                          ):



	# the Pokemon class already has an attack method, this attack method in the WaterPokemon subclass
	overrides the previous attack method
	def attack(self, other):
		other.decrease_hp(75)


class ElectricPokemon(Pokemon):
	def __init__(self, name, owner, hp, origin):
		# YOUR CODE HERE


	def attack(self, other):
		other.decrease_hp(60)

>>> ashs_squirtle = WaterPokemon('Squirtle', 'Ash', 314)
>>> mistys_togepi = Pokemon('Togepi', 'Misty', 245)
>>> ashs_squirtle.get_hit_pts() # WaterPokemon doesn't have a get_hit_pts method defined?!


>>> ashs_squirtle.attack(mistys_togepi)


>>> mistys_togepi.get_hit_pts()
```