Homework 13

Building a car factory in Python

Pick up where we left off in lecture with the definition of the `car` class, with five attributes and three methods.

```
class car:
  """ Create a new car """
  def __init__(self, make = "honda", model = "civic",
               year = 2007, mpg = 30, gas = 12):
    """ Create a new car with attributes """
    self.make = make
    self.model = model
    self.year = year
    self.mpg = mpg
    self.gas = gas

  def drive(self, distance):
    """ Drive a distance (in miles) and deplete the gas """
    self.gas = self.gas  - distance / self.mpg

  def age(self, years):
    """ Age a car by a number of years and decrease mpg """
    self.mpg += -years / 4

  def greet(self, name):
    """ Greet the user with car characteristics """
    print("Hello, I am a " + self.make.capitalize() +
          " " + self.model.capitalize() + " and my name is " +
          name + ".")
```

1. The `.age()` function is currently problematic: its possible to age your car into negative mpg. It also doesn't keep track of the age explicitly. Modify the `.__init__` method to add an age attribute and modify the `.age()` method so that mpg is not a strictly decreasing linear function of years aged.
2. Modify the `.drive()` method to respond more usefully if the user specifies the car to drive farther than is possible given the gas in the tank. If this happens, prevent the gas from going negative and print a message when the tank is empty. Be sure your method has a docstring.
3. Add a `.fill()` method that can be used to fill the tank back up with gas. To make

this realistic add an attribute for the size of the gas tank and print message to `.fill()` for when the tank is full. Be sure your method has a docstring.

4. Right now, the notion of driving only involves the spending of gas. More importantly, though, it changes the location of the car. Add an attribute that stores the distance of the car from Cal. Modify the `.drive()` method to take an additional argument `to_cal` that takes boolean values and will affect the distance from Cal attribute. Add a message that prints whenever the car has returned to Cal.

Building a car factory in R

R's S3 approach to OOP is a bit different. The new `sloop` package has some useful tools for understanding how it works.

5. Use `s3_methods_generic()` to print out all of the methods that stem from the generic function `summary()`. Which ones have you used before?

6. Use `s3_methods_class()` to print out all of the methods have been written for the `lm()` class. Which ones have you used before?

In lecture we created an S3 R class called "car" by writing a *constructor* function that sets the attributes of an instance of that object. Instead of including the methods in the definition of the class, they're written separately, are are associated with a *generic* version of the function. Here, we replaced `.greet()` with `summary.car()`.

```
new_car <- function(make = "honda", model = "civic",
                    year = 2007, mpg = 30, gas = 12) {
  out <- list(make = make,
              model = model,
              year = year,
              mpg = mpg,
              gas = gas)
  structure(out, class = c("car", "list"))
}

summary.car <- function(obj, ...) {
  cat(paste0("Hello, I am a ", obj$make, " ", obj$model, "."))
}
```

7. Run the above code then call the same two `sloop` functions to check which methods the `car` class has to check in `summary.car()` shows up as one of the methods for the `summary()` generic.

8. Implement a `age.car()`, `drive.car()`, and `fill.car()` method with the same functionality as in Python, adding attributes to your constructor as necessary. Writing each of these methods actually requires writing two functions: a generic function, e.g. `drive()`, as well as the class-specific methods. These generics are very simple pass-through functions that serve to be sure the correct method is called. See https://adv-r.hadley.nz/s3.html#s3-methods to learn their structure. Note that a method must have the same arguments as its generic. There is one exception to this rule: if the generic has `...`, the method can contain a superset of

the arguments. This allows methods to take arbitrary additional arguments (see: https://adv-r.hadley.nz/functions.html?q=dot-dot-#fun-dot-dot-dot).

9. Use `sloop` to verify the methods that are available for an object of class `car`.

10. This exercise is to be done twice: first in R then in Python. Construct a car that gets 20 mpg and starts 210 miles from Cal with 10 gallons in its gas tank. Write lines of code that describe each of the following:

    - The car drives straight back to Cal.
    - The car drives for awhile back to Cal, stops for gas, then continues back to Cal.

    Each line of Python code should be a single line using dot notation to pass methods. Each line of R code should be concatenate the methods using pipe operator `%>%`.

11. An alternative to writing R code as methods for an S3 object is to just write independent functions `age()`, `drive()` and `fill()`. What are the advantages and disadvantages of the object-oriented approach?