Classes and Objects

Object Oriented Programming:

Concept 1: Encapsulation

→ hide internal variables from the user!! this allows us to change the way we do things internally (like in a function) without affecting user code.

GOOD

- ☆ Maintenance.
- ☆ Team / Agile Develop mat models.

"BAD"

→ code is harder to write, and makes good design a premium.

Objects → a grouping of variables and functions which act on those variables.

Classes → a "template" for how to make and operate on / manage objects.

↑ The DNA of object oriented programming.

Example: Suppose we wanted to build a restaurant rating system, like YELP. We would want to store data for many different restaurants.

For each restaurant, we might want to record:

string ① name

int ② rating (1-10)

string ③ price (* ... *****)

string ④ Cuisine type

In addition, we might want some functions that would act on these variables:

Set_name("....")

get_name()

Set_rating(7)

get_rating()

⋮

etc.

print_report()

We seek to define a template, or a CLASS, that defines those variables and the functions that act on those variables.

"instantiate"

Then we can create objects of this class. Each object would correspond to a different restaurant.

Key Idea: we design the internal variables and the functions. The user only interacts with the objects through the functions!!! That is the idea of encapsulation

OKAY. That's a lot of words. Let's now see how we could actually accomplish this; in Python.

```
class Restaurant:

    def __init__(self):
        self.name = "none"
        self.rating = -1
```

```
        self.rating = "none"
        self.price = "none"
        self.cuisine = "none"

    def set_name(self, user_name):
        self.name = user_name
    def get_name(self):
        return self.name

    def set_rating(self, user_rating):
        self.rating = user_rating
    def get_rating(self):
        return self.rating

    def set_price(self, user_price):
        self.price = user_price
    def get_price(self):
        return self.price

    def set_cuisine(self, user_cuisine):
        self.cuisine = user_cuisine
    def get_cuisine(self):
        return self.cuisine

    def get_info(self):
```

```
def print_info(self):
    print("Restaurant Information")
    print(f"    Name: {self.name}")
    print(f"    Rating: {self.rating}")
    print(f"    Price: {self.price}")
    print(f"    Cuisine Type: {self.cuisine}")

if __name__ == "__main__":
    user_name = input()
    user_rating = int(input())
    user_price = input()
    user_cuisine = input()

    Moes = Restaurant()
    moes.print_info()
    moes.set_name(user_name)
    moes.set_rating(user_rating)
    moes.set_price(user_price)
    moes.set_cuisine(user_cuisine)
    moes.print_info()
```

Notes:

def __init__(self):

→ This is called the default Constructor of the class, and is called whenever an object of this class is initialized.

→ all functions of the class are typically written with the first argument as "self". However, this argument is not included in calls to the class functions from the main

class ...
program.

Ex.
```
def set_name (self, user_name):
    self.name = user_name
```

In main
```
my_name = "Moes"
set_name (my_name)
```

"self" refers to "this object here", or "myself". It gives the function internal access to all of the objects internal functions and variables.

→ As written, the internal member variables of the class are actually public. So, it is technically possible to write:

print (moes.name)

in the main program. But, this is very bad practice, because it violates the principles of Encapsulation!!! Don't do this. Use setter and getter methods.

Class Attributes vs. Instance Attributes.

The internal variables like name, rating, price, cuisine are referred to as instance attributes. This is they apply to each separate object, or instance, of the class.

It's possible, though, to assign class attributes that apply to all members of the class, as a whole.

Example:

```
class MarathonRunner:
    race_distance = 42.195

    def __init__(self):
        self.name = "none"
        self.time = 0.0

    def get_speed(self):
        .
        .
        .
        etc.

runner1 = MarathonRunner()
runner2 = MarathonRunner()

print(runner1.race_distance)
print(runner2.race_distance)
```

```
print (runner2.
print (MarathonRunner.race_distance)
```

```
42.195
42.195
42.195
```

Class attributes are sort of like global variables for the class... or semi-global, anyway.