Class Constructors

In the previous examples, the constructor that was used was as simple as possible. It simply defined the member variables and assigned some default values.

You can imagine that we might want to be able to choose the initial values of (some of) those member variables when we create the object.

We can do this easily in Python, and it only requires a simple change:

```
def __init__(self, name, rating,
             price, cuisine):
    self.name = name
    self.rating = rating
    self.price = price
    self.cuisine = cuisine
```

Now, define objects

Notes:

① We can now define objects like:

```
moes = Restaurant("moes", 3, "**",
                  "Mexican")
```

② We use the same variable initialization names in the constructor definition as for the internal number variables.

③ It is nice to provide a "docstring" with explanations of the number variables.

```
    """
    :param name: Name of the Restaurant (string)
    :param rating: Rating (1-10) int)
    :param price: Price (*-*****, string)
    :param cuisine: Cuisine Type (string)
    """
```

We can access this using

help(Restaurant)

in the main program.

Modification 1: What if we

want to provide some default values for some of the parameters, but allow the user to specify others?

Ex.

```
def __init__(self, name, cuisine,
             rating = -1, rating = "none")
```

⋮

Now, we would initilize with:

```
moes = Restaurant("moes", "Mexican")
```

N.B. The non-default parameters must all come before the default ones !!! 

Cool Thing: We can still override the default values !!

```
panera = Restaurant("Panera", "Soups and sandwiches", 4, "$$$$")

fiveguys = Restaurant("Five Guys", "Burgers", 10, "$$$")
```

will still work !!

Best Practices:

Give default values to all of the member variables. Then,

all of the following make sense.

```
Restaurant()
Restaurant("Moes")
Restaurant("Moes", "Fake Mex")
Restaurant("Moes", "Fake Mex", 6)
Restaurant("Moes", "Fake Mex", 6, "$$")
```

Another "best practice", or maybe I should say "convention" is to use an underscore to indicate methods of the class that are internal, and which users would not typically access.

Ex.

```
class Rectangle:
    def __init__(self, length=1, width=1):
```

```
def __init__(self, length, width):
        self.length = length
        self.width = width

def _get_area(self):
    return self.length * self.width

def resize(self, new_length):
    area = self._get_area()
    self.length = new_length
    self.width = area / new_length

def print_rectangle(self):
    print(f"Length: {self.length:.1f}
          Width: {self.width:.1f}
          Area: {self._get_area():.1f}")
```

Again, this is just a convention, and is not required. We could have just called it get_area(). And, it's still a publicly available method that can be

called from the

I mention it because when we see "_something()" it makes us think that it is somehow special, and in this case, it is not.