Class Customization

We have seen already that a very common thing to do is to print the object information. It's super useful. Well, Python has built in a cool way to do this.

```
def __str__(self):
    return (f"..... ")
```

So, we can do things in the main program like:

```
moes = Restaurant("moes", "Mex")
print(moes)
```

and it will automagically call the "__str__" method of the class, if it exists !!

N.B. in the __str__ method, we return the thing to be sent to a single print statement.

Operator Overloading

Algebra => al gebra Arabic

the way/method

→ a set of rules/definitions for how to do +, −, ÷, *, ( ), log( ), e^, etc.

→ Question: what does it mean to add two objects together?

```
moes = Restaurant()
panera = Restaurant()
my_fusion_restaurant = moes + panera
```

??

→ Well, maybe for a given class we can come up for our own set of rules that make sense for that class 🙂

Operators: +, -, *, /, ...

logical operators: <, >, <=, >=, !, ...

==

important !!

Time class.

Example:

```
class Time:
    def __init__(self, hours, minutes):
        self.hours = hours
        self.minutes = minutes

    def __str__(self):
        return f'{self.hours}:{self.minutes}'

    def __lt__(self, other):
        if self.hours < other.hours:
            return True
        elif self.hours == other.hours:
            if self.minutes < other.minutes:
                return True
        return False

time1 = Time(10, 40)
```

this object

the other object

```
time2 = Time(12, 13)
time3 = Time(9, 15)

min_time = time1
if time2 < min_time:
        min_time = time2
if time3 < min_time:
        min_time = time3

print(f'Earliest time is {min_time}')
```

What are the other interesting "Rich comparison" logical operators?

- `__lt__` ⇨ `<`
- `__le__` ⇨ `<=`
- `__gt__` ⇨ `>`
- `__ge__` ⇨ `>=`

--eq-- ⇨ ==

--ne-- ⇨ != These are defined by default, but may not be what we want.

BE CAREFUL

one cannot say for sure !!!