COMP 110

Classes in Memory + Magic Methods

```python
class Pizza:
    """This is my class to represent pizza!"""

    size: str
    toppings: int
    gluten_free: bool

    def __init__(self, size_input: str, toppings_input: int, gf_input: bool):
        """Constructor"""
        self.size = size_input
        self.toppings = toppings_input
        self.gluten_free = gf_input
        # returns self

    def price(self) -> float:
        """Method to compute price of pizza"""
        if self.size == "large":
            cost: float = 6.25
        else:
            cost: float = 5.00
        return cost

    def add_toppings(self, num_toppings: int):
        """Update exisiting pizza order with num_toppings"""
        self.toppings += num_toppings

pie: Pizza = Pizza("medium", 2, False)
pie.add_toppings(2)
print(pie.price())
```

```python
"""Challenge Question Class"""
from __future__ import annotations

class Point:

    x: float
    y: float

    def __init__(self, init_x: float, init_y: float):
        self.x = init_x
        self.y = init_y

    def scale_by(self, factor: int) -> None:
        self.x *= factor
        self.y *= factor

    def scale(self, factor: int) -> Point:
        return Point(self.x * factor, self.y * factor)

my_point: Point = Point(1.0, 2.0)
my_point.scale_by(3.0)
new_point: Point = my_point.scale(2.0)
print(new_point.x)
print(new_point.y)
```

Magic Methods

- Methods with functionality already built-in: you don't *directly* call them, but rather they are invoked by a different action
    - One example: calling Point() really called __init__()
    - Other example: calling print(x)
- Always starts and ends with two underscores (e.g. __init__)

An example in VS Code…

```python
"""Challenge Question Class"""
from __future__ import annotations

class Point:

    x: float
    y: float

    def __init__(self, init_x: float, init_y: float):
        self.x = init_x
        self.y = init_y

    def __str__(self) -> str:
        return f"({self.x},{self.y})"

    def scale(self, factor: int) -> Point:
        return Point(self.x * factor, self.y * factor)

my_point: Point = Point(1.0, 2.0)
new_point: Point = my_point.scale(2.0)
print(my_point)
print(f"My new point is: {new_point}")
```