Magic Methods + Operator Overloads

Practice Writing Classes

- Define a class named Profile
- It has two attributes username: str and private: bool
- Write an __init__ method that:
  - takes username_input: str as input and initializes the username attribute with that value
  - initializes the private attribute with the value True
- Write a tweet method that:
  - takes msg: str as input and if the private attribute is False, prints msg
- Instantiate the class by creating user1, a Profile with the username 110_rulez
- Change user1's private attribute to False
- Call tweet for user1 with the message "OOP is cool!"

Review

```python
"""Practice writing a class."""

# Definition
class Profile:

    username: str
    private: bool

    def __init__(self, username_input: str):
        """Create a new Profile object."""
        self.username = username_input
        self.private = True

    def tweet(self, msg: str) -> None:
        """If profile is public, print msg."""
        if self.private is False: # not self.private
            print(msg)

# Instantiation
user1: Profile = Profile("110_rulez") # calls __init__()
user1.private = False
user1.tweet("OOP is cool!")
```

Review

What are unique properties of the __init__ method? (What sets it apart from other methods?)

Magic Methods

- Methods with built in functionality!
- Not called *directly!*
- Names start and end with two underscores (__<method_name>__)

Question

When I call print(x), Python calls what magic method on x *before* printing?

__str__ Magic Method

- Gives a str representation to an object of a Class.
- Call it by calling str(<class_object>)

__str__ Magic Method

- Gives a str representation to an object of a Class.
- Call it by calling str(<class_object>)

Let's try it together!

Let's define a __str__ magic method that returns a string of information about a Profile object.

Operator Overloads

- You can write magic methods to give operators meaning!
- Think about operators you use on numbers that you'd like to use on other objects, e.g. `+`, `-`, `*`, `/`, `<`, `<=`, etc…
- This is called operator overloading

Arithmetic Operator Overloads

+	__add__(self, other)
–	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
**	__pow__(self, other)
%	__mod__(self, other)

Comparison Operator Overloads

<	__lt__(self, other)
>	__gt__(self, other)
<=	__le__(self, other)
>=	__ge__(self, other)
==	__eq__(self, other)
!=	__ne__(self, other)

For each magic method call, what is self and (if applicable) what is other?

str(a)	__str__(self)
a + b	__add__(self, other)
a – b	__sub__(self, other)
a * b	__mul__(self, other)
a < b	__lt__(self, other)
a == b	__eq__(self, other)

Bonus: Union Types

Say I have:

```
def add(x: int, y: int = 1) -> int:
    return x + y
```

and I want this function to work for ints *or* floats…

I can express this using the Union operator:

```
def add(x: int | float, y: int | float = 1) -> int | float:
    return x + y
```

Next Class: Challenge Question!

*You are going to use union types!