Python Object Orientation

Intro

OO Review

- class
  - blueprint for new objects, defines attributes & methods
- method
  - function defined on class, can see/change attributes on instance
- class method
  - function defined on class, called on class, not individual instance

Instances

Like in JS, you make an instance by calling the class:

```
from collections import Counter

# make instance of a counter
counts = Counter("hello world")

type(counts)     # 'collections.Counter'

isinstance(counts, Counter)     # True
```

Get/set attributes or find methods with `.` (like JS):

```
# get most common letter
counts.most_common(1)
```

JavaScript:

- get/set attribute of object: `o.name` *or* `o['name']`
- call method: `o.method()` *or* `o['method']()`

Python:

- get/set attribute of object: `o.name`
- call method: `o.method()`
- retrieve value from dictionary: `o['my-key']`
  - not the same thing!

What Can I Do With This Object?

`help(obj)`
Show help about object and methods

`dir(obj)`
List methods/attributes of object

Classes

Making classes is similar to JS:

```
class Triangle:
    "Right triangle."

    def __init__(self, a, b):
        "Create triangle from a and b sides."
        self.a = a
        self.b = b

    def get_hypotenuse(self):
        "Get hypotenuse (length of 3rd side)."
        return math.sqrt(self.a ** 2 + self.b ** 2)

    def get_area(self):
        "Get area of triangle."
        return (self.a * self.b) / 2

    def describe(self):
        return f"My area is {self.get_area()}"
```

Self

self is similar to ***this***

- ***this*** is a bit magical: it automatically gets created
- ***self*** is explicit: you must list it as the first argument of methods
  - It's just a normal variable, otherwise

Inheritance

Like in JS, classes can subclass other objects:

```
class ColoredTriangle(Triangle):
    """Triangle that has a color."""

    def __init__(self, a, b, color):
        # get parent class [`super()`], call its `__init__()`
        super().__init__(a, b)

        self.color = color

    def describe(self):
        msg = super().describe() + f" I am {self.color}"
```

Super

Like in JS, ***super*** finds parent class:

- JS: `super` is parent, `super(...)` calls parent constructor function
- Python: `super()` is parent, `super().__init__(...)` is parent initializer

Multi-Level Inheritance

Like in JS, you can have multiple levels of inheritance

Documenting Classes

As always, good style to have comment explaining purpose of class & methods:

```
class Triangle:
    "Right triangle."

    def __init__(self, a, b):
        "Create triangle from a and b sides."
        self.a = a
        self.b = b

    def get_hypotenuse(self):
        "Get hypotenuse (length of 3rd side)."
        return math.sqrt(self.a ** 2 + self.b ** 2)

    def get_area(self):
        "Get area of triangle."
        return (self.a * self.b) / 2
```

Documenting Instance

When you print an instance/examine in Python shell, often not helpful:

```
>>> tri = Triangle(3, 4)

>>> tri
<__main__.Triangle object at 0x1012a6358>
```

Would be nicer to see values for ***a*** and ***b***

We can do this by making a ***__repr__*** *(representation)* method:

```
class Triangle:
    "Right triangle."

    def __init__(self, a, b):
        "Create triangle from a and b sides."
        self.a = a
        self.b = b

    def __repr__(self):
        return f"<Triangle a={self.a} b={self.b}>"

    def get_hypotenuse(self):
        "Get hypotenuse (length of 3rd side)."
        return math.sqrt(self.a ** 2 + self.b ** 2)

    def get_area(self):
        "Get area of triangle."
        return (self.a * self.b) / 2
```

```
>>> tri = Triangle(3, 4)

>>> tri
<Triangle a=3 b=4>
```