Lecture 17 - Objects and State

Motivation and History

State, Collections of data.

Separation of Data from Operations on data.

We could store data in lists

```
kirk = ["James Kirk", 34, "Captain", 2265]
spock = ["Spock", 35, "Science Officer", 2254]
mccoy = ["Leonard McCoy", "Chief Medical Officer", 2266]

print ( kirk[1] )
print ( spock[1] )
print ( mccoy[1] )
```

From: https://realpython.com/python3-object-oriented-programming/

Python’s Implementation is Objects

Objects are created as a package that defines the layout and names for data and combines that with functions that operate on the data. These functions ore called “methods”.

Create a Class:

```
class Dog:
    pass
```

Initialization is important - so with a class we have a special function called `__init__` that setups the data and initializes it.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.food = ""
```

This introduces the term `self` - in other languages it is often called `this`. `self` refers to the set of data that this class is going to work on. In the constructor, the special function called `__init__`, `self.name` creates a variable called `name` that is tied to the instance data for this class.

By doing the `__init__` in this way we require that creating a `Dog` means that you must pass 2 values, then name and the age.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.food = ""

spot = Dog('Spot',2)
kitty = Dog('Kitty',5)
```

And...

Access some values in the data.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.food = ""

spot = Dog('Spot',2)
kitty = Dog('Kitty',5)

print ( spot )
```

We can create functions that change parts of the class data.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.food = ""

    def setFood(self, food):
        self.food = food

    def getName(self):
        return self.name
```

```

spot = Dog('Spot',2)
kitty = Dog('Kitty',5)

spot.setFood ( "Kibbles" )
kitty.setFood ( "Purena Dog Chow" )

print ( "{}'s favorite food is {}".format(spot.name, spot.food) )
```

You can compare to see if they are euqal.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.food = ""

    def setFood(self, food):
        self.food = food

    def getFood(self):
        return self.food

    def getName(self):
        return self.name


spot = Dog('Spot',2)
kitty = Dog('Kitty',5)

print ( "are they the same? {}".format( spot == kitty ) )
```

We can put the format of name/age into the class itself.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.food = ""

    def setFood(self, food):
        self.food = food

    def getFood(self):
        return self.food

    def getName(self):
        return self.name

    def desc(self):
        return "{} is {} years old.".format(self.name, self.age)

spot = Dog('Spot',2)
kitty = Dog('Kitty',5)

print ( kitty.desc() )
```

We control how the values are changed inside the class.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def setFood(self, food):
        self.food = food

    def getFood(self):
        return self.food

    def birthDay(self):
        self.age = self.age + 1

    def getAge(self):
        return self.age
```