Object-Oriented Programming In Python

By Shane Killen

What is Object-Oriented Programming

- Classes
- Objects
- Blueprint
- Attributes
- Functions(methods)

```python
class Car:
    def __init__(self, type, color, engine, doors):
        self.type = type
        self.color = color
        self.engine = engine
        self.doors = doors

    def display_car(self):
        print("Type: ", self.type, " Color: ", self.color, " Engine: ",
              self.engine, " Doors: ", self.doors)

car = Car("truck", "white", "v8", "2")
car2 = Car("sedan", "red", "I4", "4")
car.display_car()
car2.display_car()
```

What is an object

+Everything

```
car = Car(car_type, car_color,car_engine,car_doors)
car.display_car()
```

```
input("This is an object -->").lower()

list = [1, 2, 3, 4]
list.append(5)
list.pop(2)
list.sort()

x = 5
y = 10
string = "String"
num = 10.25
```

Benefit of OOP

```python
class Car:
    def __init__(self, type, color, engine, doors):
        self.type = type
        self.color = color
        self.engine = engine
        self.doors = doors

    def display_car(self):
        print("Type: ", self.type, " Color: ", self.color, " Engine: ",
              self.engine, " Doors: ", self.doors)

num_car = int(input("How many cars would you like to make"))

for car in range(num_car):
    car_type = input("enter the type of car: ")
    car_color = input("enter the color of the car: ")
    car_engine = input("enter the engine the car has: ")
    car_doors = input("enter how many doors the car has: ")
    car = Car(car_type, car_color, car_engine, car_doors)
    car.display_car()
```

```python
car_doors = 4
car_color = "red"
car_engine ="I4"
car_wheels = "winter"
car_rims = "17inch"
car_type = "sedan"

car_type2 = "truck"
car_color2 = "white"
car_engine2 = "v8"
car_wheels2 = "Off-road"
car_rims2 = "19inch"
car_doors2 = 2

print(car_type)
print(car_doors)
print(car_color)
print(car_engine)
print(car_wheels)
print(car_rims)

print(car_type2)
print(car_doors2)
print(car_color2)
print(car_engine2)
print(car_wheels2)
print(car_rims2)
```

Classes

Def __init__(self, x, y, z):
    Self.x = x
    …

What is

- __init__
- Self.x
- (self)
- Setters and getters

```python
class Car:
    def __init__(self, type, color, engine, doors):
        self._type = type
        self._color = color
        self._engine = engine
        self._doors = doors

    def get_type(self):
        return self._type

    def get_color(self):
        return self._color

    def get_engine(self):
        return self._engine

    def get_doors(self):
        return self._doors

    def set_type(self, type):
        self._type = type

    def set_color(self, color):
        self._color = color

    def set_engine(self, engine):
        self._engine = engine

    def set_doors(self, doors):
        self._doors = doors

    def display_car(self):
        print("Type: ", self._type, " Color: ", self._color, " Engine: ",
              self._engine, " Doors: ", self._doors)
```

```python
car = Car("Truck", "Red", "V8", "4")
car.set_color("Blue")
car.set_engine("V6")
print("The truck got repainted to: ", car.get_color())
print("The trucks engine is a ", car.get_engine())
car.display_car()
```

```
The truck got repainted to:  Blue
The trucks engine is a  V6
Type:  Truck  Color:  Blue  Engine:  V6  Doors:  4
```

Your Turn

- Create a pizza class
- Ask the user for
  - Number of pizzas they want
  - The size of the pizza
  - Topping of the pizza

Next Step in OOP

+Class Libraries

+Special Methods

+Super class

+Inheritance

+Polymorphism

__str__	__nonzero__	__getitem__	__add__	__neg__
__repr__	__getattr__	__setitem__	__sub__	__abs__
__lt__	__setattr__	__delitem__	__mul__	__int__
__le__	__delattr__	__reversed__	__floordiv__	__long__
__eq__	__get__	__contains__	__mod__	__float__
__ne__	__set__	__setslice__	__divmod__	__oct__
__gt__	__delete__	__delslice__	__pow__	__hex__
__ge__	__call__	__lshift__	__and__	__index__
__cmp__	__len__	__rshit__	__xor__	__enter__
__hash__	__iter__	__div__	__or__	__exit__