

Object Oriented Programming

Jannusch Bigge

05.12.2023

Objetc Oriented Programming

Object Oriented Programming

- Python is an object oriented programming language
- Everything is an object
- Objects have attributes and methods
- Attributes are variables
- Methods are functions

What is an object?

What is an object?

- An instance of a class

What is an object?

- An instance of a class

What is a class?

- A blueprint for an object

Object Oriented Programming

What is an object?

- An instance of a class

What is a class?

- A blueprint for an object

You can create multiple objects from one class!

Example - OOP

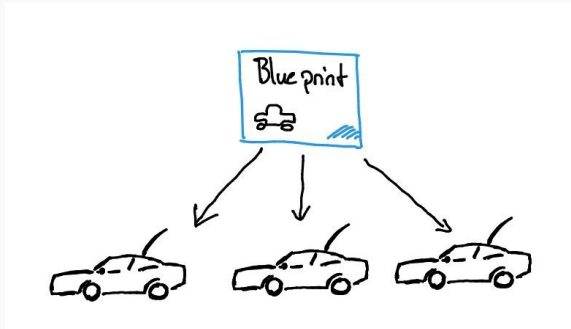
We have a blueprint for a car.

And we create three cars from this blueprint.

Example - OOP

We have a blueprint for a car.

And we create three cars from this blueprint.



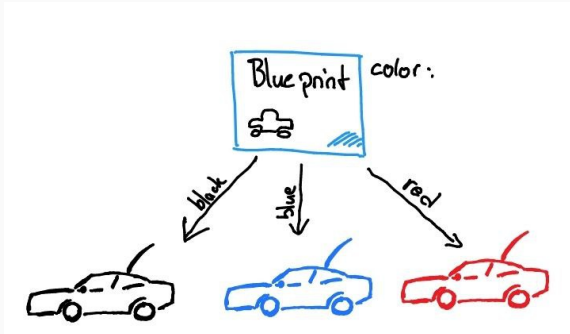
Example - OOP

```
class Car:
    def __init__(self) -> None:
        print(" Created a new car ")

car1 = Car()
car2 = Car()
car3 = Car()
```

OOP - Attributes

- Attributes are variables
- They are defined in the `__init__` method
- They are accessed with `self`



Example - OOP

```
class Car:
    def __init__(self, color: str) -> None:
        print(" Created a new car ")
        self.color = color

car1 = Car(" black ")
car2 = Car(" blue ")
car3 = Car(" red ")
```

- Methods are functions of the object
- They are defined in the class
- Accessing attributes with `getter` and `setter`
- Called by referencing the object

To get the color of the car we need a `getter` method.

Example - OOP

```
class Car:
    def __init__(self, color: str) -> None:
        print("Created a new car")
        self.color = color

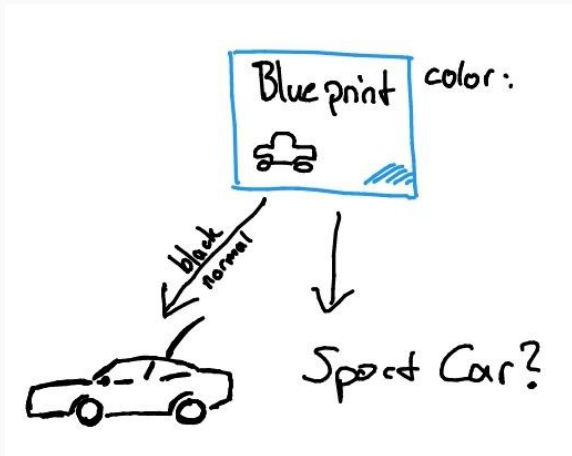
    def get_color(self) -> str:
        return self.color

car1 = Car("black")
color = car1.get_color()
print(f"The car is: {color}")
>>> The car is: black
```

We are interested in different cars.

OOP - Inheritance

We are interested in different cars.



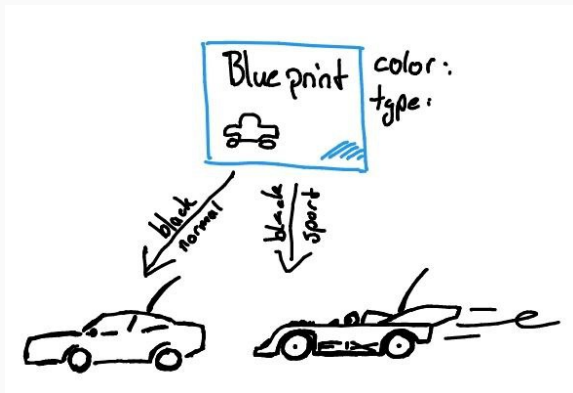
We are interested in different cars.

Solutions:

- Create a attribute "type"

OOP - Inheritance

We are interested in different cars.



We are interested in different cars.

Solutions:

- Create a attribute "type"

But what if we want to add a attribute only for one type?

e.g. race license

We are interested in different cars.

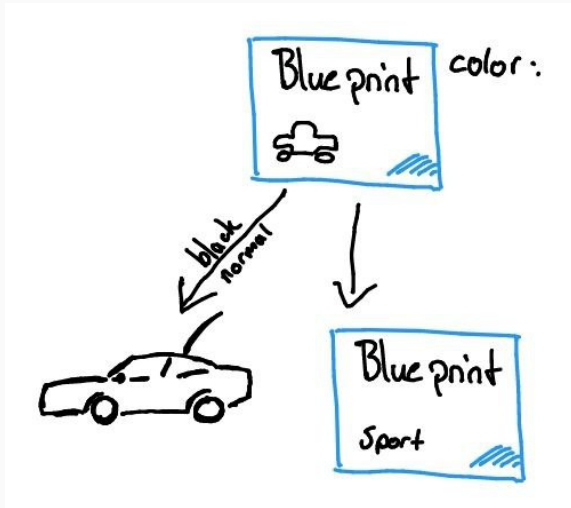
Solutions:

- Create a attribute "type"
- Create a new class "sort car"

OOP - Inheritance

We are interested in different cars.

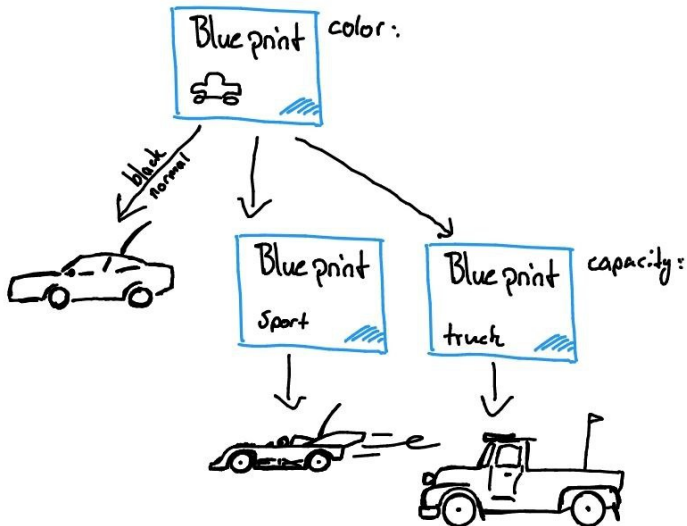
Create a new class "sport car" that inherits from "car"



OOP - Inheritance

- Inheritance is a way to create a new class from an existing class
- The new class is called child class
- The existing class is called parent class
- The child class inherits the attributes and methods of the parent class
- The child class can overwrite and add attributes and methods of the parent class

OOP - Inheritance



Example - OOP

```
class Car:
    def __init__(self, color: str) -> None:
        print(" Created a new car")
        self.color = color

    def get_color(self) -> str:
        return self.color

class SportCar(Car):
    def __init__(self, color: str, license: bool):
        super().__init__(color)
        self.license = license

car1 = Car(" black")
car2 = SportCar(" blue", True)
```


Magic Methods

Magic Methods

- Magic methods are special methods that are called by special syntax
- They are used to implement operator overloading
- They are defined with two underscores before and after the name
- Normally they are called by the interpreter and only implicitly by the user

We already used the `__init__` method.

```
def __gt__(self, other):  
    return self.hp > other.hp
```

List Comprehension

List Comprehensions

- List comprehensions are a way to create lists
- They are more compact than for loops
- They are faster than for loops
- They are more readable than for loops (sometimes)

List Comprehensions

```
newlist = [expression for item in iterable if condition == True]
```

```
# Create a list with the numbers from 0 to 9
```

```
#old way
```

```
numbers = []
```

```
for i in range(10):
```

```
    numbers.append(i)
```

```
# with list comprehensions
```

```
numbers = [i for i in range(10)]
```

```
odd = [i for i in range(10) if i % 2 == 1]
```

Task
