Introduction to Object Oriented Programming in Python

Aleksander Fabijan

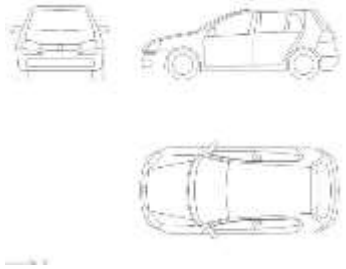
Why OOP (from last time)?

- Often we describe real-life objects in code (which is easier with OOP),
- Code get’s more manageable,
- With OPP, it is easy to reuse previously written code,
- …
- Every bigger real-life project will be written in OOP.

Object oriented programming

- Object in Python is a representation of a person, a place, a bank account, a car, or any item that the program should handle.

- Object Oriented Programming Recipe:
  - (1) define **classes** (these are descriptions)
  - (2) **make object instances** out of classes.

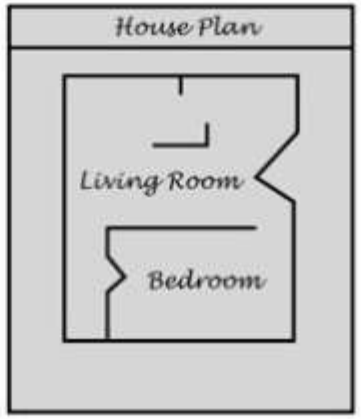


Class Car

Car instances

Class

Blueprint that describes a house



Instances of the house described by the blueprint

3 objects /
instances /
individuals

OOP with a Taxi Example

- To learn OOP, we will use an example of a Taxi.

Example Object - Taxi

Every object has two main components:

Data (the attributes about it)

Behavior (the methods)

DATA

- DriverName
- OnDuty
- NumPassenger
- Cities

BEHAVIOR

- PickUpPassenger
- DropOffPassenger
- SetDriverName
- GetDriverName

Taxi
• DriverName: **string** • OnDuty: **Boolean** • NumPassenger: **int** • Cities:**list**
• PickUpPassenger():**int** • DropOffPassenger(): **int** • SetDriverName(**string**) • GetDriverName:**string**

Creating a simple class in Python

- In a class, we describe (1) how the object will look like, and (2) what behavior it will have.

- Classes are the blueprints for a new data type in your program!

- A class should have at minimum*:
  - A name (e.g. **Class Taxi**)
  - A constructor method (that describes how to create a new object, **__init__)**

Example Class

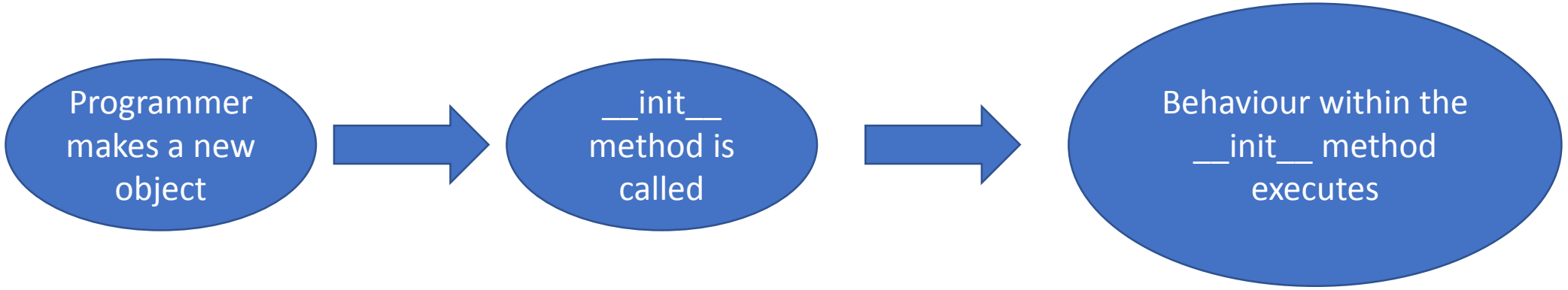
CLASS NAME

```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
```

Object Variables

The __init__ method

- __init__ is a special method in Python classes,
- The __init__ method is the constructor method for a class
- __init__ is called when ever an object of the class is constructed.



myTaxi = Taxi(“Aleks”)

…
Self.taxiDriver = “Aleks”
…

Example Object - Taxi

DATA

- DriverName
- OnDuty
- NumPassenger
- Cities

Creating an object from the class

- From one class, you make objects (instances).

```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0


ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
print ourFirstTaxi.cities
```

Object Creation Flow

- To create a new object, use the class name

```
ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
```

Object Creation Flow

- To create a new object, use the class name

```
ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
```

- When you create a new object, the __init__ method from the class is called with the parameters that were passed.

Object Creation Flow

```
Taxi("Aleks", True, ['Lund', 'Malmo'])
```

- The __init__method Is called

```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0


ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
print ourFirstTaxi.cities
```

Example Object - Taxi

DATA

- DriverName
- OnDuty
- NumPassenger
- Cities

BEHAVIOR

- PickUpPassenger
- DropOffPassenger
- SetDriverName
- GetDriverName

Behavior of a class

- classes/objects can have methods just like functions except that they have an extra **self** variable at the beginning.

- An object method takes as the first parameter the object (self) and can accept any number of other parameters.

Example Object Method

```
#We Describe the beaviour of the class with methods
def changeDriverName(self, newDriverName):
    ''' a simple method that updates the name of the driver'''
    self.dname = newDriverName
```

This method changes the name of the taxi driver for the passed object (self).

Another example of an object method

```
def pickUpPassengers(self, numOfPickedUpPassengers):
    ''' a method that increases the number of passengers'''
    self.numPassengers += numOfPickedUpPassengers
```

Exercise: Write a class that describes a **Bus**.

- A bus is created with a *number of seats*, *a color*, and is driven by a bus driver that *has a name*. New passengers can get in the bus, and existing bus passengers can leave their seat and get of the bus. Number of buss passengers can't be smaller than 0.

Instructions

- Start by drawing a class diagram
- Create a class for the bus in python
- Create two objects of your class

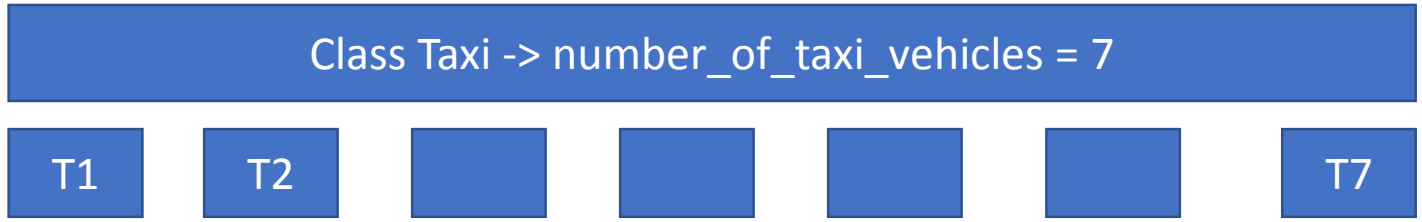
```
class Taxi:
    '''This class describes how a taxi may look like'''
    def __init__(self,driverName, onDuty, cities):
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0


ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
print ourFirstTaxi.cities
```

Object vs. Class Variables

- Most variables are **object specific** (for example, the variable number of passengers in a taxi is different for every taxi that we create).

- Some variables are **common to all objects** (for example, if we want to count the number of taxis that we have in our company)



Accessing class variables

- To access a class variable within a method, we use the @classmethod decorator, and pass the class to the method.

```
    @classmethod
    def how_many(cls):
        """returns the current TAXI inventory."""
        return cls.numberOfTaxis
```

Example use of class variable

```
class Taxi:
    '''This class describes how a taxi may look like'''

    numberOfTaxis = 0
    @classmethod
    def how_many(cls):
        """returns the current TAXI inventory."""
        return cls.numberOfTaxis

    def __init__(self,driverName, onDuty, cities):
        ''' The method that is called when a new object is created'''
        self.dname = driverName
        self.oduty = onDuty
        self.cities = cities
        self.numPassengers = 0
        Taxi.numberOfTaxis = Taxi.numberOfTaxis +1
```

```
ourFirstTaxi = Taxi("Aleks", True, ['Lund', 'Malmo'])
ourSecondTaxi = Taxi("Mladen", True, ['Lund', 'Malmo'])
ourFirstTaxi.changeDriverName("Jacob")
ourFirstTaxi.pickUpPassengers(4)
print ourFirstTaxi.dname,"is driving", ourFirstTaxi.numPassengers,
    "passenger(s). Currently, we have: ", Taxi.how_many(), " taxi(s)."
```

Exercise2: Update your class **Bus**.

- As an owner of the bus company, I wish to keep track of the number of busses that people buy (create).

Instructions

- Create a class variable
- Increment a class variable on __init__
- Create a @classmethod to print its value

Exercise 3: Hippo ZOO

- A local ZOO keeper wants to model his collection of **Hippo**s in the Zoo.
- Every hippo has a **name** and **size** (in kg). In the morning, a zoo keeper feeds the hypo so his <u>weight increases</u>. During the day the Hippo exercises, therefore his <u>weight goes down</u>.
- A zoo keeper also needs a way to keep track how many Hippos he has in the ZOO.
- Model this problem (1) with a Class diagram & (2) in Python OOP code.

Example Solution

Hippo
+Name: string +Weight: double +NumberOfHippos: int
+IncreaseWeight(Weight_DELTA) +DecreaseWeight(Weight_DELTA) +PrintNumberOfHipposInZOO()

```python
class Hippo:
    HipposInZoo = 0         # class variable
    # A class method that tells the ZOO keeper a number of Hippos in the zoo
    @classmethod
    def getNumberOfHIppos(cls):
        return cls.HipposInZoo

    def __init__(self, hName, hWeight):
        self.hippoName = hName
        self.hippoWeight = hWeight
        Hippo.HipposInZoo += 1
    def IncreaseWeight(self, wDELTA):
        self.hippoWeight += wDELTA
    def DecreaseWeight(self, wDELTA):
        self.hippoWeight -= wDELTA

newHIppo = Hippo("Jabba", 100)
newHIppo = Hippo("Luke", 32)
print Hippo.getNumberOfHIppos()
```

Takeaways

- Today, we learned how to create simple classes and objects.
  - A class is a blueprint for objects, and it contains attributes and behavior.

- We added to our classes a number of:
  - instance methods & instance variables (for example, color, size, etc.).
  - class methods & class variables (for exmaple, a number of taxis).

- We learned that new objects are created from a class through the **__init__** method.

For Next time

In our next lecture, we will learn how to reuse the code that we have written today while describing other objects. We will learn about:

- Inheritance. (TransportVehicle – Taxi)
- Encapsulation (e.g. private functions)
- Polymorphism (defining the same function for different types of objects)