Lecture 11: from Procedural thinking to Object-Oriented thinking

introducing Object-Oriented programming:

- defining functions + defining & using data + calling functions

⇒ the way we've been programming so far in this course

vs.

- defining classes + instantiating objects + calling methods

⇐ the way we're going to be programming from now on in this course
Lecture 11: from Procedural thinking to Object-Oriented thinking

What do we know so far ... about *procedural thinking*?

We'll start by examining two example problems – not programming problems...

...and we'll try to provide solutions that are object-oriented!

Example 1: what if we want to open a factory?
   And build motorcycles?
   And be able to accelerate/brake on one motorcycle at a time?

Example 2: what if we want to start a bank?
   And provide bank account?
   And be able to make deposits/withdrawals, etc. on each bank account, individually?
Procedural thinking

on the other hand, what do we know so far about Procedures?

Example of a procedural *algorithm*:

A flow chart describing how to prepare ...

...some eggs for breakfast
Object-oriented thinking: start with a design – a blueprint!

What if we want to open a **factory**... ...to **build motorcycles**?

We need a plan!

We need a design!

**PS:** what you see in the picture is not a motorcycle: it's a design for a motorcycle!
Object-oriented thinking: build them in a factory

1. provide design →
    → 2. provide materials →
    → 3. build motorcycles!

If we want to...
...build motorcycles?

We need a factory!
Object-oriented thinking: using the motorcycles we built

What if we want to have a motorcycle race?

We have our motorcycles...

we need to be able to accelerate/brake on one motorcycle at a time! (obvious, right?)
Scalability for Physical objects

- We want to build 100 motorcycles.
- Each motorcycle should have the same features, but operate independently of the others.
- We could build each motorcycle by hand, one at a time…
- Or we could use a blueprint!
Scalability for Virtual objects

• We want to model 100 motorcycles for a videogame.
• Each "virtual motorcycle" should have the same features (functions and data), but operate independently of the others.
• We could write code for each "virtual motorcycle" by hand, one at a time…
• Or we could use a blueprint!
Lecture Task: from Procedures to Objects

- What if we want to open a bank account?
- And be able to make deposits/withdrawals?

Lecture task: Design it! How?

1. form groups of 2 students per team
   (please no students left alone! If # of students is not even, at most 1 team of 3 students)

2. on your index card **design** for a bank account:
   list "properties" that each bank account may need to have
   list "actions" that may be required for each account
   (both properties and actions need to be separated in two categories: general vs. unique)
Object Oriented Programming

• Our Objectives Are:

• Create **classes** to define **objects**

• Write **methods** and create **attributes** for objects

• Learn how to **instantiate** objects from classes

• Today: learn about **self** in Python
OOP : Object-Oriented Programming

• Based around **software objects** that have:
  • attributes (*variables*, i.e. **data** about the object) and
  • methods (*functions* associated with the object, i.e. **actions**).

• recall: *function definition* – defines how a function will behave when the function is called.

• **new**: *Class definition* – defines what an object’s attributes and methods are, and how they behave.
  The **class definition** is the **blueprint** we were talking about.
OOP : Object-Oriented Programming

• Like we do with function definitions, we only write the class definition (blueprint) once!

• Instantiating – creating an instance of the class (object) (Like building a motorcycle from the blueprints)

• Each object instance can have its own attributes (i.e. values that can be different for any individual object, such as the value that tells us the speed of any individual motorcycle).

• **New**: the class as a whole can also have *shared attributes* (one value shared across the class). For example, many motorcycles may share the same brand name, etc.
OOP : Object-Oriented Programming

Declaring a class looks like this – a general example:

class ClassName(object):
    """This is a blueprint for a class""

    def method1(self):
        # self refers to the object
        # do something!
        # in front of the dot (see next slide)

    def method2(self):
        # do something else!

Note about choosing names:
• class names begin with an uppercase letter.
• method names (like function names) begin with a lowercase letter.
OOP : Object-Oriented Programming

Declaring a class looks like this – a specific example:

class Motorcycle(object):
    """A virtual motorcycle""
    
    def speak(self, text):
        print("Hi, I'm a talking motorcycle!")
        print(text)

# main program here:

myBike = Motorcycle()  # instantiates an object

myBike.speak("Oil Change Time!")  # calls a method

...see the next two slides for complete script & transcript, as from examples seen at lecture time.
# we're going to define a new class named Motorcycle:
class Motorcycle(object):
   """A virtual motorcycle""

    # here we define a new method named speak()
    # the method speak() belongs to the class Motorcycle,
    # i.e. any Motorcycle object will be able to call speak()
    def speak(self, text):
        print("Hi, I'm a talking motorcycle!")
        print("And I say...." + text)
        print("By the way, while this speak() method is running,"
        print("the value of self is: " + str(self))

    # ------------------------------------------------------
    # the main program code begins here

    # proceed with instantiation of a couple of Motorcycle objects,
    # i.e. create new objects that follow the same 'blueprint'
    # of the Motorcycle class definition:

    myBike = Motorcycle()    # instantiates a Motorcycle object
    ivansBike = Motorcycle() # instantiates another Motorcycle object

    print("I created a new object named myBike, from the class Motorcycle.")
    print("While in the main program, the myBike object is: " + str(myBike))
    print()
    print("I also created a new object named ivansBike, from the class Motorcycle.")
    print("Still in the main program, the ivansBike object is: " + str(ivansBike))
    print()
    print("Now I'm going to ask myBike to say something:")
    # call the speak() method belonging to the myBike object:
    myBike.speak("Oil Change Time!")
    print()
    print("Now I'm going to ask ivansBike to say something:")
    # call the speak() method belonging to the ivansBike object:
    ivansBike.speak("I'm the other one.")
running the Python script from the previous slide (defining a class and instantiating objects from that class) looks like this – a specific example –

note these two values: they refer to the same object!
same here!

---

I created a new object named myBike, from the class Motorcycle. While in the main program, the myBike object is: `<__main__.Motorcycle object at 0x102055c18>`

I also created a new object named ivansBike, from the class Motorcycle. Still in the main program, the ivansBike object is: `<__main__.Motorcycle object at 0x102068940>`

Now I'm going to ask myBike to say something:
Hi, I'm a talking motorcycle!
And I say....Oil Change Time!
By the way, while this speak() method is running, the value of self is: `<__main__.Motorcycle object at 0x102055c18>`

Now I'm going to ask ivansBike to say something:
Hi, I'm a talking motorcycle!
And I say....I'm the other one.
By the way, while this speak() method is running, the value of self is: `<__main__.Motorcycle object at 0x102068940>`

---
Reading Assignment 11