PROGRAMACIÓN I

OBJECT ORIENTED PROGRAMMING

Business Analytics

OBJECT ORIENTED PROGRAMMING

- *Classes and instances*

Object Oriented Programming

Object Oriented Programming (or OOP): programming paradigm (pattern, model)

Approach to structuring programs/applications so that the **data held**, and the **operations performed on that data**, are **bundled** together into **classes** and **accessed via objects**.

Classes can be used to represent real world entities (an employee, a product, a vehicle), but also abstract (financial transaction, a purchase order).

https://docs.python.org/3/tutorial/classes.html

Classes and instances

Classes as templates used to constructed instances or examples of a class of things.

Instances have same data structure (attributes) but contain their own values.

A class should accomplish one specific purpose; it should capture only one idea.

Terminology

- **Class**: defines a combination of data and behavior that operates on that data.
  - A class acts as a template when creating new instances.
- **Instance** or **object**: an example of a class.
  - All instances of a class possess the same data fields/attributes but contain their own data values. Each instance of a class responds to the same set of requests.
- **Attribute/field/instance variable**:
  - The data held by an object is represented by its attributes.
  - The "state" of an object at any particular moment relates to the current values held by its attributes.
- **Method**: a procedure defined within an class.

Class definition

class nameOfClass:

<statement 1>

<statement 2>

Class definitions, like function definitions (def statements) must be executed before ”use"

In practice, the statements inside a class definition will usually be function definitions (that we will call methods), but other statements are allowed, and sometimes useful

A class can be a subclass of another class (superclass), but we will see that later on.

Basic instantiation

Class *instantiation* uses function notation. Just pretend that the class object is a parameterless function that returns a new instance of the class.

For example:

```
class MyClass:
    """A simple example class"""
    def f(self):
        return 'hello world'


x = MyClass()
x.f()
```

Here we (1) define a class called `MyClass`, with a single method `f`. We then (2) create an instance of `MyClass` named `x`, and finally (3) we call the f method of the instance `x`

Instances and memory

Universidad Francisco de Vitoria
UFV Madrid

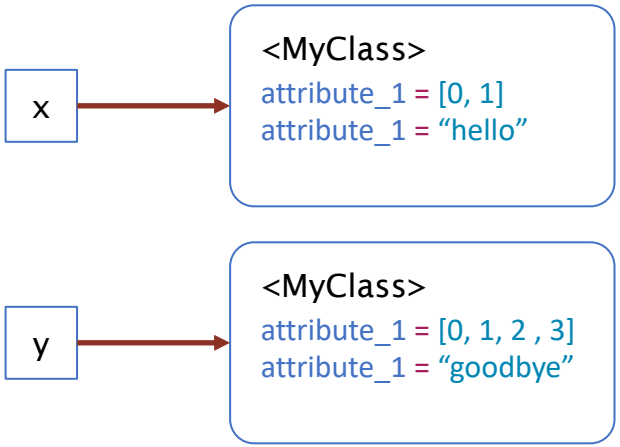
Two instances of the same class with occupy different positions in memory

```
class MyClass:
    """A simple example class"""
    def f(self):
        return 'hello world'
```

```
x = MyClass()
x.f()

y = MyClass()

id(x)
id(y)
```



id(x): 43424424234
id(y): 44303030333

Instantiation with __init__ (constructor)

The instantiation operation (“calling” a class object) creates an empty object.

Many classes like to create objects with instances customized to a specific **initial state**.

For this purpose we define a special method named __init__(), like this:

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart


x = Complex(3.0, -4.5)
print(x.r, x.i)
```

Instantiation with __init__ (constructor)

When a class defines an __init__() method, class instantiation automatically invokes __init__() for the newly-created class instance.

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart


x = Complex(3.0, -4.5)
print(x.r, x.i)
```

Note that this __init__() method has two parameters `realpart` and `imagpart`. During instantiation, we are passing the arguments 3.0 and -4.5, so the object x will be created with an initial state (values for the instance variables)

Methods, and `self`

Remember that we create classes to bundle together **data** and **operations performed on that data**

Methods are the “functions inside classes”, typically performing operations on the object

The special thing about methods is that the instance object is passed as the first argument of the function.

x.f() is exactly equivalent to MyClass.f(x).

You can have more parameters in the method, but the first one is a pointer to the instance, and the convention is to use the name `self`

Simple example


```
class Dog:

    def __init__(self, name):
        self.name = name
        self.tricks = []  # creates a new empty list for each dog

    def add_trick(self, trick):
        self.tricks.append(trick)


d = Dog('Fido')
e = Dog('Buddy')

d.add_trick('roll over')
e.add_trick('play dead')

print(d.tricks)
# ['roll over']
print(e.tricks)
# ['play dead']
```

Complex number with an absolute value method

Universidad Francisco de Vitoria

UFV Madrid

Create a class that defines:

1. two attributes (instance variables), x and y, for the real and imaginary parts of a complex number
2. A method that returns the absolute value $|z|$ (call it abs) of the complex number. Remember:

$$|z| = \sqrt{x^2 + y^2}$$

3. Create an instance with real part 3 and imaginary part 4
4. Call the method abs and print the output