10_classes

June 1, 2023

[ ]:
```
# Object-oriented programming.  Motivation.

# Let us suppose that we take some measurements that we want to anlyze.

xx = [3.0, 4.0, 7.0, 2.0]
nx = 4

def media(yy, ny):
    m = 0.0
    for y in yy:
        m = m + y
    m = m / ny
    return m

def varianza(yy, ny):
    m = media(yy, ny)
    v = 0.0
    for y in yy:
        v = v + (y - m)**2.0
    v = v / ny
    return v

m = media(xx, nx)
v = varianza(xx, nx)

# We can certainly define e.g. the functions above.  However, we have to
# separately manage the measured data and the results of the calculation.
# Moreover, we have to remember that these functions are meant to work with
# this kind of data, and we have to copy the definitions in the scripts when
# we take new measures.

# A possibly better approach is to group data, properties, analysis and results
# together in a structure.  Dictionaries would work well for static data
# but not for functions.  The appropriate construct is called a "class".
# Think of it as a cookie stamp.  A class allows us to define objects, that
# are called "instances" of the class.  Think of each object as a cookie.
# They all have the same shape but are distinct and can have different
```

```
# flavors.

# Define a class.
class Misure():

    # A class contains the function __init__ which is executed when the
    # class is called to produce and instance.
    def __init__(self, yy, ny, s):
        # The variable "self" refers to the instance of the class itself.
        # At this stage in the code, the instance is still not there.
        # The dot symbol indicates variables which are saved in the
        # class instance.  They are not the same between different instances.
        # Think of instance variables as the cookie flavor.
        self.yy = yy
        self.ny = ny
        self.studente = s

    # Functions defined in the class are called "methods".  These are the
    # same for all the class instances.  Think of them as the cookie shape,
    # impressed by the cookie stamp.
    # The first argument of each function has to be "self", the variable
    # referring to the instance itself.  This argument is automatically
    # passed to the function when it is invoked as instance.method()
    def calcola_media(self):
        m = 0.0
        for y in self.yy:
             m = m + y
        m = m / self.ny
        self.m = m

    def calcola_varianza(self):
        m = self.m
        v = 0.0
        for y in self.yy:
            v = v + (y - m)**2.0
        v = v / self.ny
        self.v = v


# Instances of the class are created using the class name, which in turns
# invokes the __init__ function.  All arguments passed to the class name
# are passed to the __init__ function.  Notice that the "self" variable is
# passed automatically.

# Create a class instance.
misura1 = Misure(xx, nx, "Mario")
# Invoke class methods.
```

```
misura1.calcola_media()
misura1.calcola_varianza()
# Retrieve instance variables.
print("La media è %8.5f" % misura1.m)
print("La varianza è %8.5f" % misura1.v)

# Create a different instance with new data.
xx2 = [8.0, 3.0, 1.0, 1.0]
misura2 = Misure(xx2, nx, "Maria")

# Among the many benefits of "encapsulating" data, results and methods
# in the same object, there is easiness of iteration.
lm = [misura1, misura2]
for mis in lm:
    mis.calcola_media()
    print("La media di %s vale %8.5f" % (mis.studente, mis.m))
```

```
La media è  4.00000
La varianza è  3.50000
La media di Mario vale  4.00000
La media di Maria vale  3.25000
```

RIASSUNTO

La programmazione a oggetti (classi) permette di raggruppare dati e risultati insieme in un'unica struttura. - Quando si definisce una classe, la prima funzione da definire è **init**, in cui la variabile "self" si riferisce all'oggetto stesso; - poi si possono definire le altre funzioni, che sono chiamate metodi; - per creare quindi un oggetto, basta assegnare dei parametri alla classe, che vengono passati come parametri della funzione init - uno dei maggiori benefici è quello della semplicità con cui si possono eseguire i cicli