Programming in Python[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Academic year 2020/21, II semester

Lecture VIII: Object Oriented encapsulation

The huge value of procedural abstraction

It is worth to emphasize again the huge value brought by **procedural abstraction**. In Python it is not mandatory to use procedures/functions: the language is designed to be used also for *on the fly* calculations.

```
x = 45
s = 0
for i in range(0, x):
  s = s + i
```

This is ok, but it is not encapsulated (in fact, since encapsulation is so important you can at least consider it encapsulated in file which contains it)

- the piece of functionality is not easily to distinguish

  it could be intertwined with other unrelated code

```
x = 45
a = 67 # another concern
s = 0
for i in range(0, x):
  s = s + i
print(a) # another concern
```

- the goal is not explicit, which data are needed, what computes
- it's hard to reuse even in slightly different contexts

Encapsulate the functionality

```
def sum_to(x: int) -> int:
    assert x >= 0
    r = 0
    for i in range(0, x):
      r = r + i
    return r

s = sum_to(45)
```

- It gives to our mind a "piece of functionality", the interpreter we are programming is now "able" to do a new thing that can be used **without thinking about the internal details**
- It makes clear which data it needs (an integer, ≥ 0 if we add also an assertion or a docstring)
- It makes clear that the interesting result is another integer produced by the calculation
- It can be reused easily and safely

Object Oriented encapsulation

Encapsulation is so important that it is used also at a higher level: a collection of related procedures.

```
x = 666

def increment():
  x = x + 1

def decrement():
  x = x - 1
```

Again: this is correct Python code, but it has problems:

- Both the functions depends on `x` but this is not clear from their signature: a user must look at the internal details
- The two functions cannot be reused individually, but only together with the other (and `x`)

Classes

A class is a way to package together a collection of related functions. The class is a "mold" to instance new objects that encapsulated the related functionalities.

```
class Counter:
   def __init__(self, start: int):
     self.x = start

   def increment(self):
     self.x = self.x + 1

   def decrement(self):
     self.x = self.x - 1

c = Counter(666)
c.decrement()
d = Counter(999)
d.increment()
```


Homework

- `https://classroom.github.com/a/JMlHieUy`
- Optional: GitHub has a new assignment on `git` and GitHub basics; try it here `https://classroom.github.com/a/KLoZ8Qxl`