

PyQB

Monga

Abstracting

Procedural

encapsulation

PyQB

Monga

Abstracting

similarities

Procedural

## Programming in Python<sup>1</sup>

Mattia Monga

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

### Academic year 2023/24, I semester

<sup>1</sup>⊛⊕⊛ 2023 M. Monga. Creative Commons Attribuzione — Condividi allo stesso modo 4.0 Internazionale. http://creativecommons.org/licenses/by-sa/4.0/deed.it

# PyQB Monga Abstracting Procedural encapsulation Lecture X: Encapsulation encapsulation

Procedural abstraction

Procedural abstraction is key for our thinking process (remember the power of recursion, for example): giving a name to a procedure/function enhances our problem solving skills.

def sum\_range(a: int, b: int) -> int: """Sum integers from a through b.

```
>>> sum_range(1, 4)
10
>>> sum_range(3, 3)
3
.....
assert b >= a
result = 0
for i in range(a, b+1):
   result = result + i
return result
```

### Another "sum"

### This is very similar... def sum\_range\_cubes(a: int, b: int) -> int: """Sum the cubes of the integers from a through b. similarities >>> sum\_range\_cubes(1, 3) 36 >>> sum\_range\_cubes(-2, 2) 0 assert b >= a result = 0for i in range(a, b+1): result = result + cube(i) # cube(i: int) -> $\rightarrow$ int defined elsewhere return result

59

PyQB

Monga

Abstracting

Procedural

encapsulation

encapsulation

### Another "sum"

This is also very similar $\frac{1}{a \cdot (a+2)} + \frac{1}{(a+4) \cdot (a+6)} + \frac{1}{(a+8) \cdot (a+10)} + \dots + \frac{1}{(b-2) \cdot (b)}$
(Leibniz: $\frac{1}{1\cdot 3} + \frac{1}{5\cdot 7} + \frac{1}{9\cdot 11} + \cdots = \frac{\pi}{8}$ )
<pre>def pi_sum(a: int, b: int) -&gt; float:     """Sum 1/(a(a+2)) terms until (a+2) &gt; b.</pre>
>>> from math import pi >>> abs(8*pi_sum(1, 1001) – pi) < 10e-3 True
assert $b \ge a$ result = 0.0
<pre>result = 0.0 for i in range(a, b+1, 4):     result = result + (1 / (i * (i + 2))) return result</pre>

### The huge value of procedural abstraction

62

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)

a = 67 # another concern

for i in range(0, x):

• the piece of functionality is not easily to distinguish

it could be intertwined with other unrelated code

s = s + iprint(a) # another concern • the goal is not explicit, which data are needed, what

x = 45

s = 0

computes

• it's hard to reuse even in slightly different contexts

**PvQB** 

Monga

Abstracting

encapsulation

encapsulation

similarities

# Can we abstract the similarity?



#### PyQB from typing import Callable Monga Num = int | float # same as Num = Union[int, float] Abstracting def gen\_sum(a: int, b: int, fun: Callable[[int], Num], step: int = 1) -> Num: similarities """Sum terms from a through b, incrementing by step. >>> gen\_sum(1, 4, lambda x: x) encapsulation 10 encapsulation >>> gen\_sum(1, 3, lambda x: x\*\*3) 36 >>> from math import pi >>> abs(8\*gen\_sum(1, 1000, lambda x: 1 / (x \* (x + 2)), 4) - pi) < 10e-3 True ...... assert b >= a result = 0.0for i in range(a, b+1, step): result = result + fun(i) if isinstance(result, float) and result.is\_integer(): return int(result) return result 63

**PyQB** 

Monga

Abstracting

Procedural

encapsulation

encapsulation

64

Encapsulate the functionality

```
def sum to(x: int) -> int:
    assert \mathbf{x} \ge 0
    r = 0
    for i in range(0, x):
       \mathbf{r} = \mathbf{r} + \mathbf{i}
    return r
```

PyQB Monga

Procedural encapsulation

 $s = sum_to(45)$ 

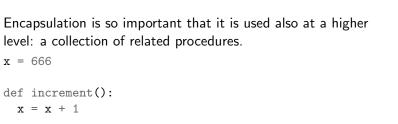
- It gives to our mind a "piece of functionality", the interpreter we are programming is now "able" to do a new thing that can be used without thinking about the internal details
- It makes clear which data it needs (an integer,  $\geq 0$  if we add also an assertion or a docstring)
- It makes clear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



# Lecture XI: OOP



### **Object Oriented encapsulation**



### 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)

67

PyQB

Monga

Abstracting

similarities

Procedural

00

encapsulation

encapsulation

### 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()



66



00 encapsulation

68