



PyQB

Monga

Abstracting similarities

Procedural encapsulation

OO encapsulation

Programming in Python¹

Mattia Monga

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

Academic year 2022/23, I semester

¹© 2022 M. Monga. Creative Commons Attribution — Condividi allo stesso modo 4.0 Internazionale. <http://creativecommons.org/licenses/by-sa/4.0/deed.it>



PyQB

Monga

Abstracting similarities

Procedural encapsulation

OO encapsulation

Lecture VIII: Encapsulation



PyQB

Monga

Abstracting similarities

Procedural encapsulation

OO 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
```



PyQB

Monga

Abstracting similarities

Procedural encapsulation

OO encapsulation

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.
```

```
>>> sum_range_cubes(1, 3)
36
```

```
>>> sum_range_cubes(-2, 2)
0
```

```
"""
assert b >= a
result = 0
for i in range(a, b+1):
    result = result + cube(i) # cube(i: int) ->
    ↪ int defined elsewhere
return result
```

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: $\frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \dots = \frac{\pi}{8}$)

```
def pi_sum(a: int, b: int) -> float:
    """Sum 1/(a(a+2)) terms until (a+2) > b.

    >>> from math import pi
    >>> abs(8*pi_sum(1, 1001) - pi) < 10e-3
    True

    """
    assert b >= a
    result = 0.0
    for i in range(a, b+1, 4):
        result = result + (1 / (i * (i + 2)))
    return result
```

PyQB

Monga

Abstracting similarities

Procedural encapsulation

OO encapsulation

57

Can we abstract the similarity?



```
from typing import Callable

Num = int | float # same as Num = Union[int, float]

def gen_sum(a: int, b: int, fun: Callable[[int], Num], step: int = 1) -> Num:
    """Sum terms from a through b, incrementing by step.

    >>> gen_sum(1, 4, lambda x: x)
    10

    >>> 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 = 0.0
    for i in range(a, b+1, step):
        result = result + fun(i)
    if isinstance(result, float) and result.is_integer():
        return int(result)
    return result
```

PyQB

Monga

Abstracting similarities

Procedural encapsulation

OO encapsulation

58

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

PyQB

Monga

Abstracting similarities

Procedural encapsulation

OO encapsulation

59

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

PyQB

Monga

Abstracting similarities

Procedural encapsulation

OO encapsulation

60

Object Oriented encapsulation



PyQB

Monga

Abstracting
similarities

Procedural
encapsulation

OO
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)

61

Classes



PyQB

Monga

Abstracting
similarities

Procedural
encapsulation

OO
encapsulation

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()
```

62