



PyQB

Monga

Functions

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>

1



PyQB

Monga

Functions

Lecture IV: Functions

26

Summary



PyQB

Monga

Functions

In Python3

- Variables are names to refer to objects;
- Objects are elements of types, which define the operations that make sense on them;
- Therefore, the basic instructions are the assignment (bind a name to an object), the proper operations for each object, and the commands to ask the services of the operating system;
- One can alter the otherwise strictly sequential execution of instruction with control flow statements: `if`, `for`, `while`.

Remember that in python3, indentation matters (it is part of the syntax).

27

Proper operations



PyQB

Monga

Functions

- On objects one can apply binary and unary operators: `2 * 3` `-(-5.0)` `not True` `'foo' + 'bar'...`
- There also built-in functions like `max(8,5,6)`, the full list is here: <https://docs.python.org/3/library/functions.html>
- (syntactically, commands like `print` or `input` cannot be distinguished from other built-in functions)
- Every object has methods that can be applied with the so called **dot notation**: `(3.2).is_integer()` `'foo'.upper()` `'xxx'.startswith('z')`; the list of which methods an object has is given by `dir(object)`.

28

Definition of functions



PyQB

Monga

Functions

As variables are names for objects, one can also name fragments of code:

```
def cube(x: int) -> int:
    square = x * x
    return square * x
```

Now we have a new operation `cube`, acting on ints: `cube(3)`. Type hints are optional (and ignored, you can call `cube(3.2)` or `cube('foo')`), but **very useful** for humans (and tools like `mypy`).

Equivalent

```
def cube(x):
    square = x * x
    return square * x
```

29

A function computes a result



PyQB

Monga

Functions

- Returns a useful result

```
def concat_with_a_space(string1: str, string2: str) -> str:
    return string1 + ' ' + string2
```

```
# string1 is the _formal_ parameter
# 'foo' is the _actual_ parameter (like an assignment string1 =
↪ 'foo')
print(concat_with_a_space('foo', 'bar'))
```

- Return None

```
def repeated_print(string: str, repetitions: int) -> None:
    for i in range(0, repetitions):
        print(string)
```

```
repeatedPrint('Hello, world!', 3)
```

- Recursive call:

```
def repeatedPrint(string: str, repetitions: int) -> None:
    if repetitions > 0:
        print(string)
        repeatedPrint(string, repetitions - 1)
```

```
repeatedPrint('Hello, world!', 3)
```

30

Functions are objects too



PyQB

Monga

Functions

One can assign functions to variables:

```
def cube(x: int) -> int:
    square = x * x
    return square * x
```

```
mycube = cube
```

```
print(mycube(3))
print(type(mycube))
```

And short functions can even be expressed as literal expressions (lambda expressions)

```
cube = lambda y: y*y*y
```

31

Naming helps solving



PyQB

Monga

Functions

The tower of Hanoi

<https://www.mathsisfun.com/games/towerofhanoi.html>

32

Describe the moves for a solution



PyQB

Monga

Functions

Recursive thinking is a powerful problem solving technique and it can be translated to Python thanks to recursive calls.

Hanoi moves $A \rightarrow C$:

- In A there is just one disk: move it to C
- Otherwise in A there are n disks (> 1):
 - **leap of faith!** I suppose to know the moves needed to move $n - 1$ disk; then
 - apply this (supposed) solution to move $n - 1$ disks from A to B (leveraging on C , empty, as the third pole)
 - move the last disk from A to C
 - apply the (supposed) solution to move $n - 1$ disks from B to C (leveraging on A , now empty, as the third pole)

This implicit description solve the problem! Finding a non-recursive solution is possible but not that easy.

33

In Python



PyQB

Monga

Functions

```
def hanoi(n: int, a_from: str, c_to: str,
        → b_intermediate: str) -> None:
    if n == 1:
        print('Move 1 disk from ' + a_from + ' to ' + c_to)
    else:
        hanoi(n - 1, a_from, b_intermediate, c_to)
        print('Move 1 disk from ' + a_from + ' to ' + c_to)
        hanoi(n - 1, b_intermediate, c_to, a_from)

hanoi(3, 'A', 'C', 'B')
```

34

Homework



PyQB

Monga

Functions

- <https://classroom.github.com/a/w8eFXVbn>

35