# Python Advanced Course

*Part IV*

Stefano Alberto Russo

# Outline

- Part I: Object Oriented Programming
  - What is OOP?
  - Logical Example
  - Attributes and methods
  - Why to use objects
  - Defining objects
- Part II: Improving your code
  - Extending objects
  - Lambdas
  - Comprehensions
  - Iterables
  - Properties
- Part III: Exceptions
  - What are exceptions?
  - Handling exceptions
  - Raising exceptions
  - Creating custom exceptions
- Part IV: logging and testing
  - The Python logging module
  - Basics about testing
  - The Python unit-testing module
  - Test-driven development

# Logging and testing

## ➔ *What is logging?*

```
[INFO] timeseria.models.forecasters: Using a window of "63"
[DEBUG] timeseria.models.forecasters: Processed 1000 items
[INFO] timeseria.models.anomaly_detectors: Using 3 standard deviations as anomaly threshold:
0.5914733390853167
[DEBUG] timeseria.utilities: Detected encoding "ascii" with "1.0" confidence (streaming=False)
[DEBUG] timeseria.storages: Processing line #0: "epoch,temperature"
[DEBUG] timeseria.storages: Set column indexes = "[0, 1]"  and column labels = "['epoch', 'temperature']"
[DEBUG] timeseria.storages: Processing line #0: "1546477200,23.71"
[DEBUG] timeseria.storages: Set time column index = "0"
[DEBUG] timeseria.storages: Will process timestamp "1546477200"
[DEBUG] timeseria.storages: Auto-detecting timestamp format
[DEBUG] timeseria.storages: Auto-detected timestamp format: epoch
[DEBUG] timeseria.storages: Set data_label_indexes="[1]" and data_label_names="['temperature']"
[DEBUG] timeseria.storages: Set data to "{'temperature': 23.71}"
[DEBUG] timeseria.storages: Processing line #1: "1546477800,23.82
```

# Logging and testing

## ➔ *What is logging?*

Logging must be *verbose* and give as much context as possible:

- `Error`
  vs
- `Cannot convert element #32 of the list of type "str" and value "ciao"`

Logging levels (if using the logging modules)

- `CRITICAL` is for errors which crash the entire program
- `ERROR` is for errors you can deal with
- `WARNING` is for  particular conditions to be notified
- `INFO` is for giving informations about the execution
- `DEBUG` is for debug messages

# Logging and testing

## ➔ *What is testing?*

- By testing we mean the testing, generally in an automatic way, of things.
- From software, to a pen, to your mobile phone.
- Software testing is easier than hardware testing (where specialized equipment is needed), because you just need to write more software
- There are those who test the testing software :)

# Logging and testing

## ➔ *What is testing?*

Pseudocode example

given a *known* **input** and **output**

*known* **input**→ CODE → **output**

if **output** != *known* **output**:

    error!

# Logging and testing

## ➔ *What is testing?*

Python code example

```
# Sum function
def sum(a,b):
    return a+b

# Testing
if not sum(1,1) == 2:
    raise Exception('Test 1+1 not passed')

if not sum(1.5,2.5) == 4:
    raise Exception('Test 1.5+2.5 non passed')
```

# Logging and testing

## ➔ *Testing vs unit-testing*

Il testing generico può anche essere effettuato su tutto il codice /programma.

input → **programma o funzione molto grossa** → output

Se invece testo le “minime” unità testabili, allora si parla di *unit testing*:

input → **funzione piccola** → output
input → **oggetto piccolo**→ output
input → **altra funzione piccola**→ output

In questo modo sono molto più granulare nel capire dove è andato storto cosa.

# Logging and testing

## ➜ *The unittest module*

my_code.py

```python
# Sum function
def sum(a,b):
    return a+b
```

test_my_code.py

```python
import unittest
from my_code import sum

# Testing
class TestSum(unittest.TestCase):

    def test_sum(self):
        self.assertEqual(sum(1,1), 2)
        self.assertEqual(sum(1.5,2.5), 4)
```

```
~/ProgrammingLab2021$ python -m unittest discover
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

# Logging and testing

## ➔ *Test-driven development*

***FIRST WRITE THE TESTS, THEN THE CODE***

→ focus on the outcome and the interfaces, not the implementation