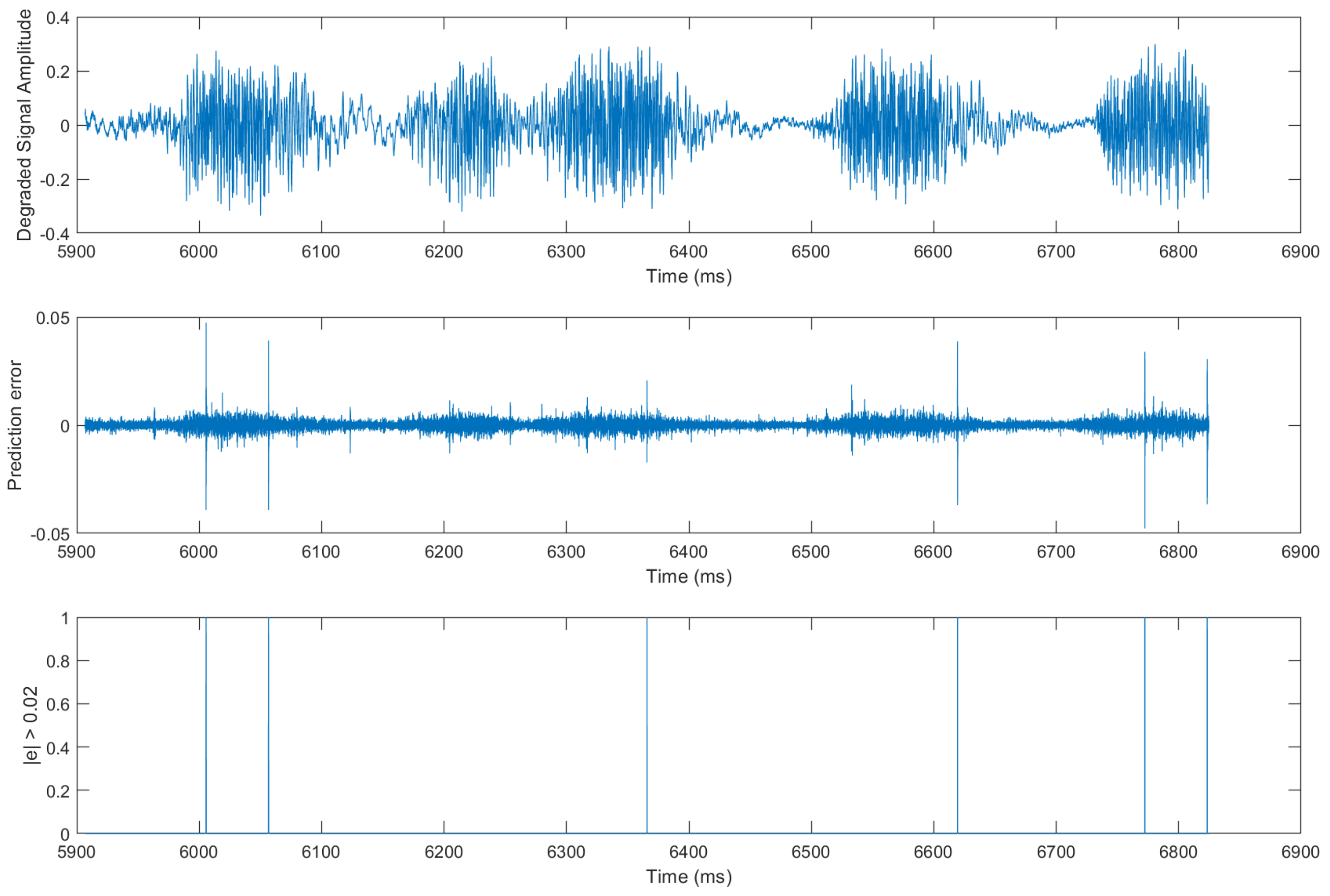
EEP5C22 Computational Methods

- Assignment II
- Unittests in Python
- Comments or Docstrings in Python
- How to write a Readme File

Introduction to Assignment II

Audio Restoration using different interpolation methods

Reminder

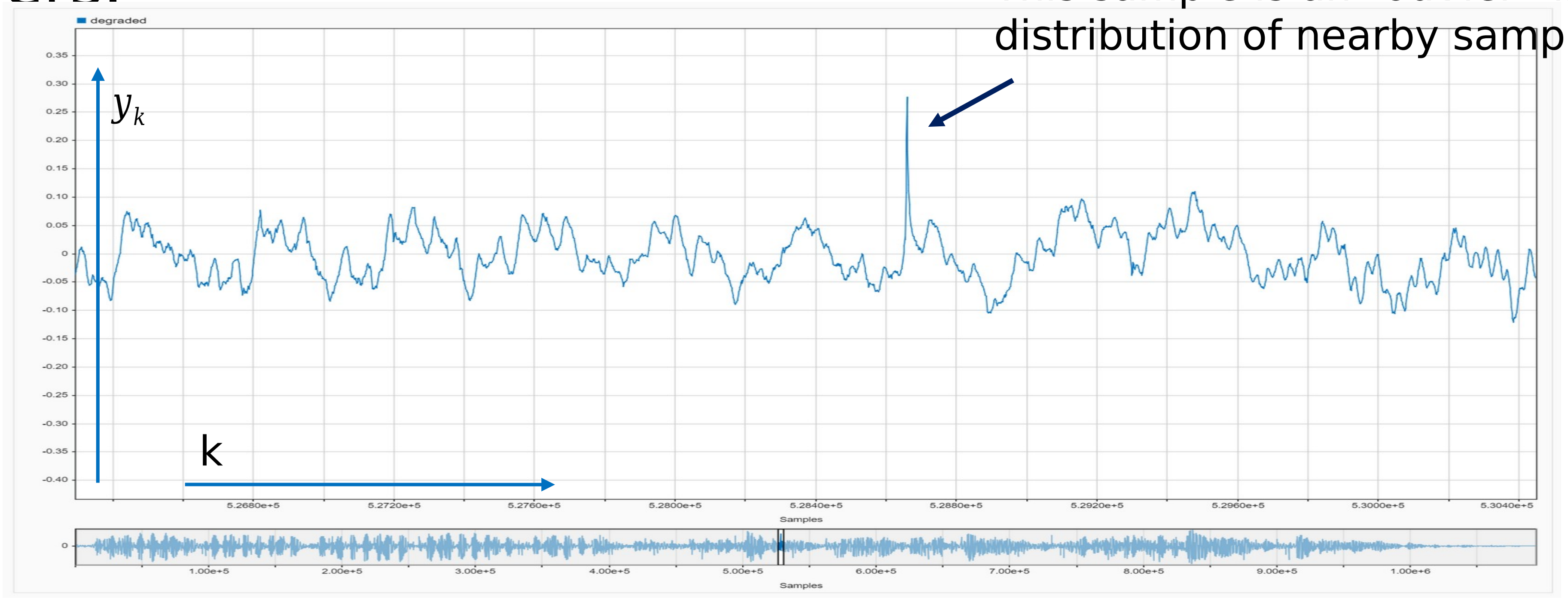

Degraded Signal Amplitude
0.4
0.2
0
-0.2
-0.4
5900
6000
6100
6200
6300
6400
6500
6600
6700
6800
6900
Time (ms)
Prediction error
0.05
0
-0.05
|e| > 0.02
1
0.8
0.6
0.4
0.2
0

Remind
er

Median - An alterative interpolator

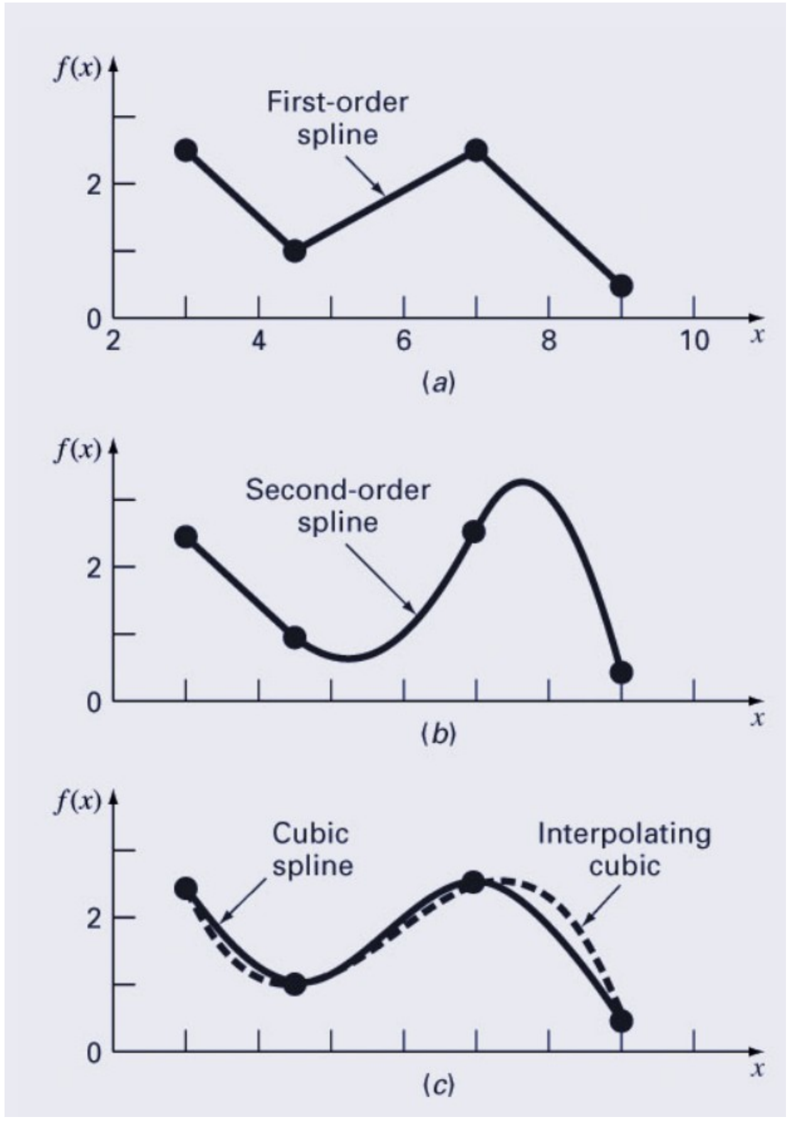
- We didn’t have to propose that the model for the signal was AR. We could have instead tried to use a simpler idea : filter the signal in such a way that we reject outliers.

This sample is an “outlier” in the distribution of nearby samples



Splines – Another Interpolator

- Concept similar to piecewise polynomial interpolation, but can be controllable by using "knots"
- In polynomial regression, we had to choose the degree
- For cubic splines, we need to choose the knots



Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

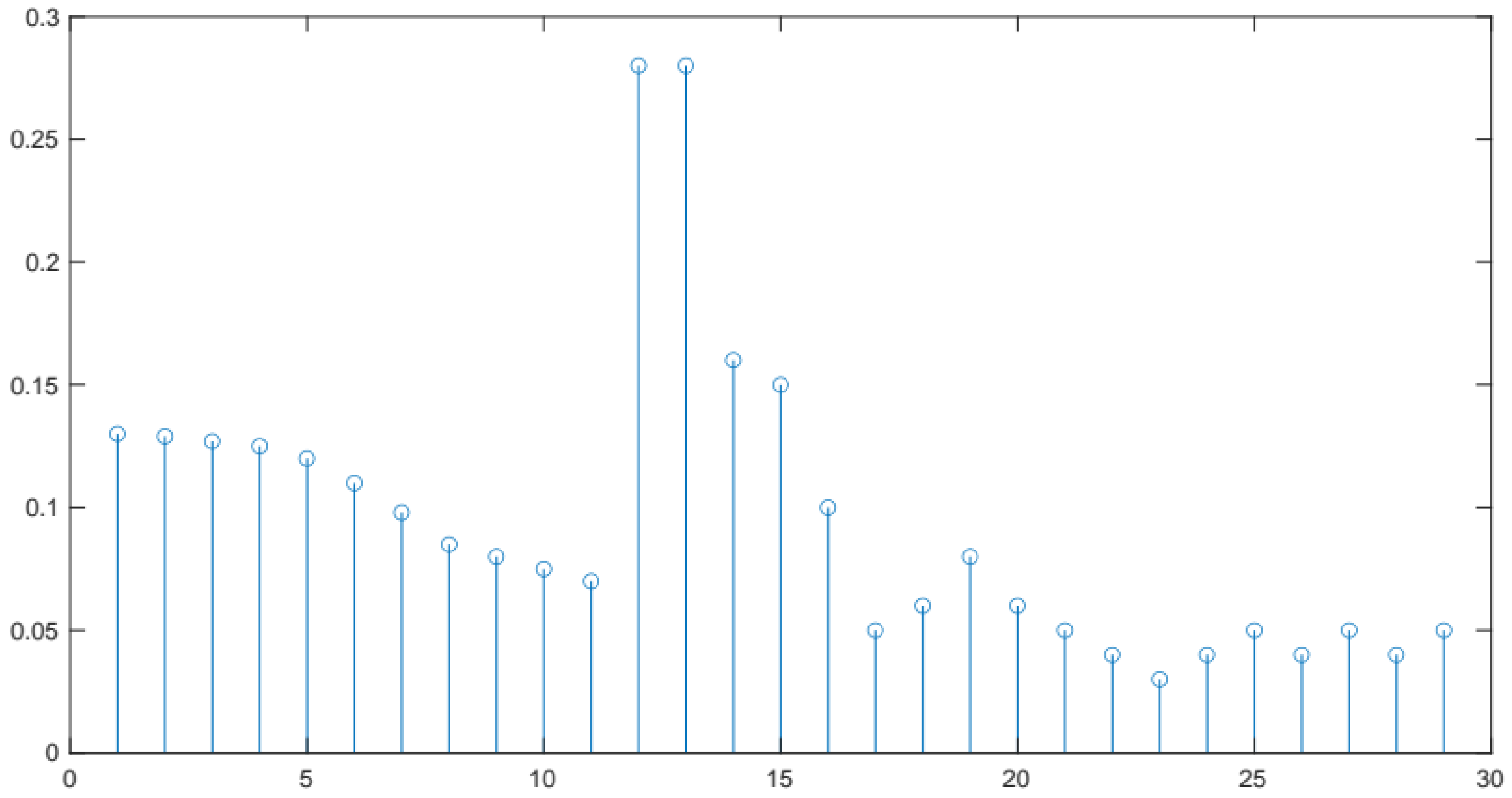
Splines – Another Interpolator

- Concept similar to piecewise polynomial interpolation, but can be controllable by using “knots”
- In polynomial regression, we had to choose the degree
- For cubic splines, we need to choose the knots
- How complex is a cubic spline with K knots?

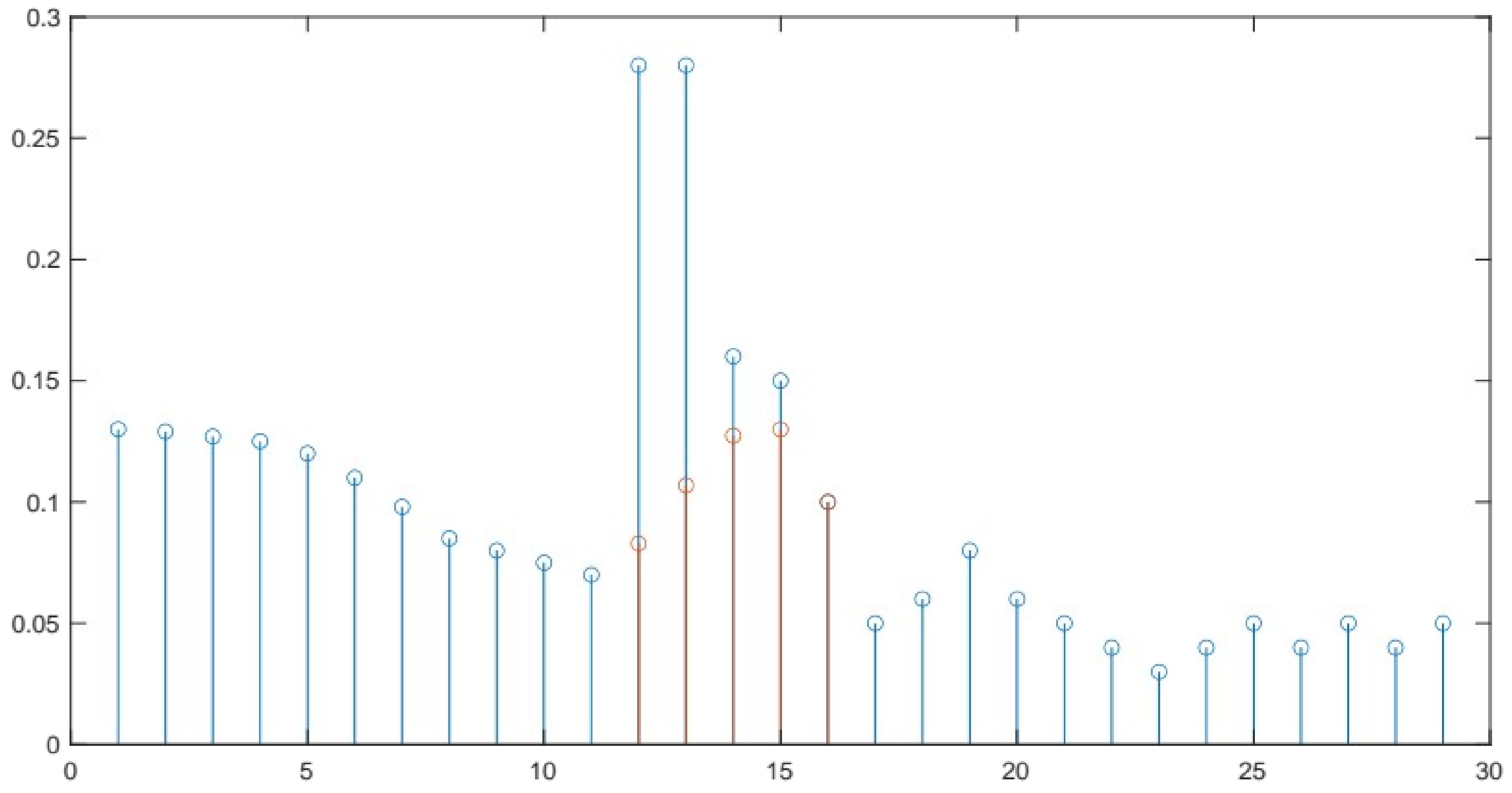
$$y = \beta_0 + \beta_1 b_1(x) + \beta_2 b_2(x) + \cdots \beta_{K+3} b_{K+3}(x) + \epsilon$$

A cubic spline with K knots is as complex as a polynomial of degree K + 3.

Splines – Another Interpolator

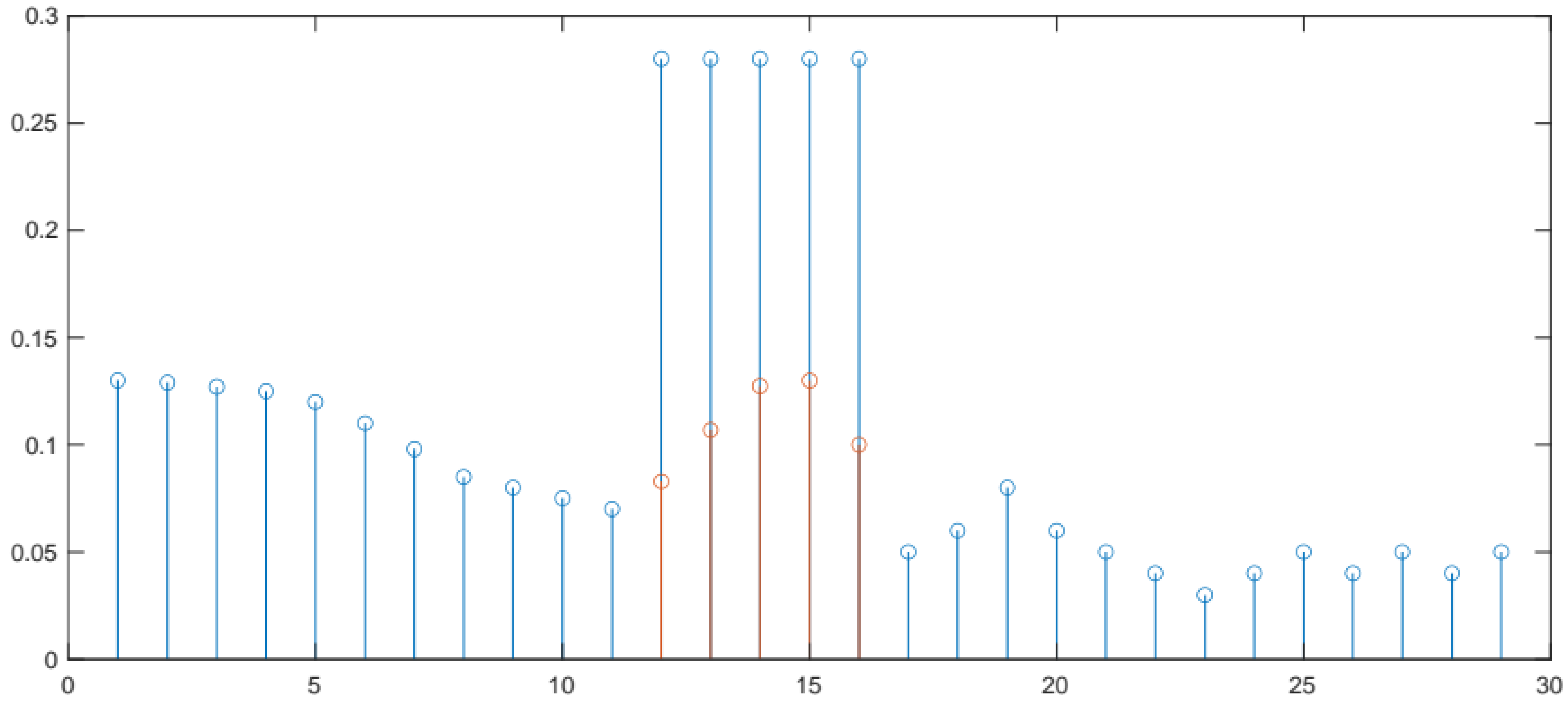


Splines – Another Interpolator



Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Another Example



First Task – Median Filter

Deadline:
30/11/202
Assessme
01/12/202

- Create a Github project – private at first.
- Create in Python an audio restoration project based on the median filter:
- Your median python routine should take as input arguments:
    - the <degraded.wav>: this is the audio file you corrupted from a clean audio <clean.wav> (you can use the same audio file from Assignment I)
    - the <detectionfile.wav>: you can use the same from Assignment I. This file gives parameter values for the median filter
    - the <filterlength>: should check that the input length is ODD and report an error if not.
    - the <outputfile.wav>: the restored audio file
- It should display a progress update (X % complete) while it is running
- It should print "Done" when it ends
- The code should generate i.e. replace every corrupted sample with the median filtered signal at that location.
- Your code should contain tests (unittests) that verify that your median filter is doing the right thing
- Compute the MSE between the clean.wav and the output.wav

Second Task – Cubic Splines Filter

**Deadline:
30/11/202
Assessme
01/12/202**

- Follow the exact same process as with the median filter, but this time the cubic splines.
- Compare MSE and execution time for the two different interpolators.
- FOR THIS EXERCISE YOU ARE NOT ALLOWED TO USE THE MEDIAN() FUNCTION IN NUMPY. But you can use it to test your code if you want.
- You are allowed to use a numpy function for the cubic splines interpolation.
- You MUST apply autopep8 formatting
- *Optional : Your programme can autoplay the degraded and restored audio files after execution completion.*
- Create a Readme file that follows the structure from the template.
- Apply version control during the development and push the code to your github project.
- **Submit your finalised files and make the project Public**

Unit Testing in Python

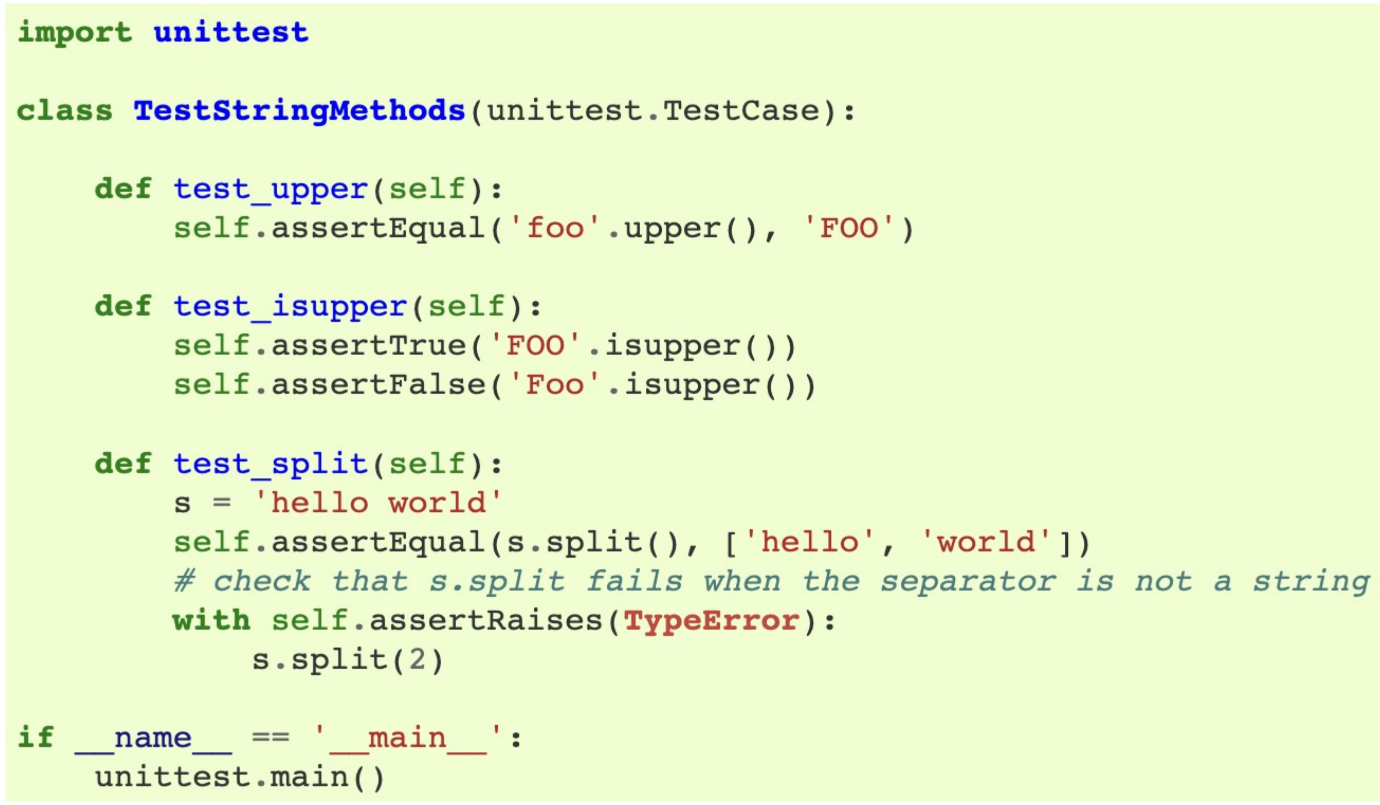
Unit Testing

- supports test automation;
- sharing of setup and shutdown code for tests
- aggregation of tests into collections
- independence of the tests from the reporting framework.

- The developers are expected to write automated test scripts, which ensures that each and every section or a unit meets its design and behaves as expected.

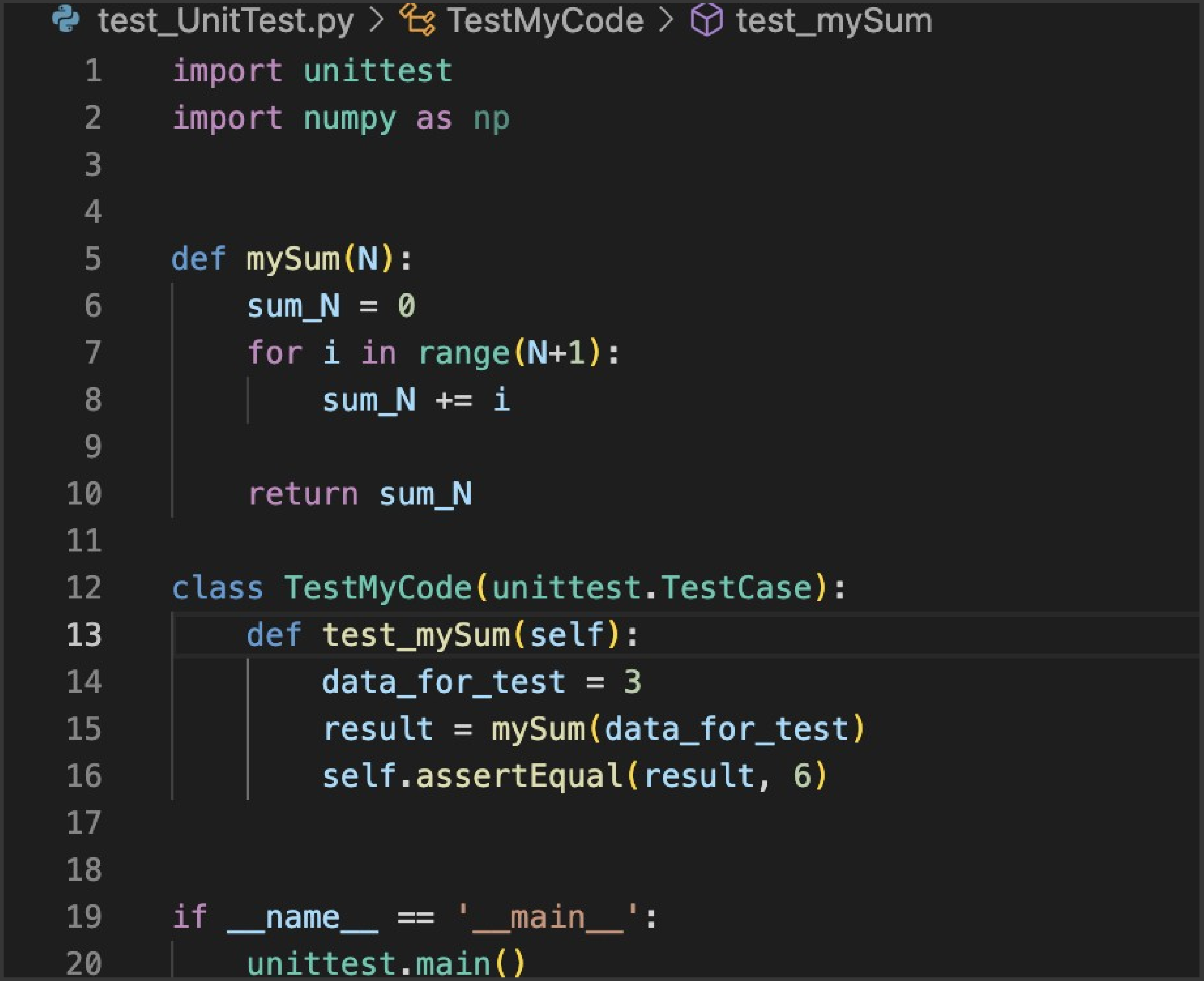
Unit Testing

- Usually, a separate file is used for unit testing
- The three individual tests are defined with methods whose names start with the letters **test**.
- This naming convention informs the test runner about which methods represent tests.



```python
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

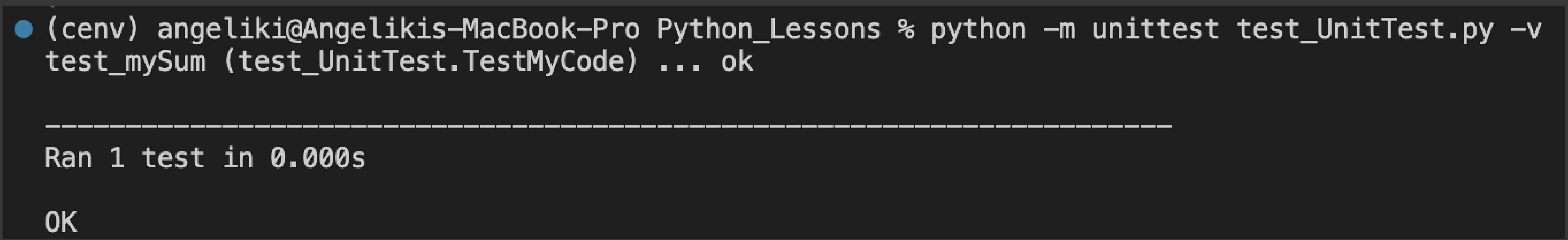
Example of Unit Testing



test_UnitTest.py > TestMyCode > test_mySum

```python
import unittest
import numpy as np


def mySum(N):
    sum_N = 0
    for i in range(N+1):
        sum_N += i

    return sum_N

class TestMyCode(unittest.TestCase):
    def test_mySum(self):
        data_for_test = 3
        result = mySum(data_for_test)
        self.assertEqual(result, 6)


if __name__ == '__main__':
    unittest.main()
```

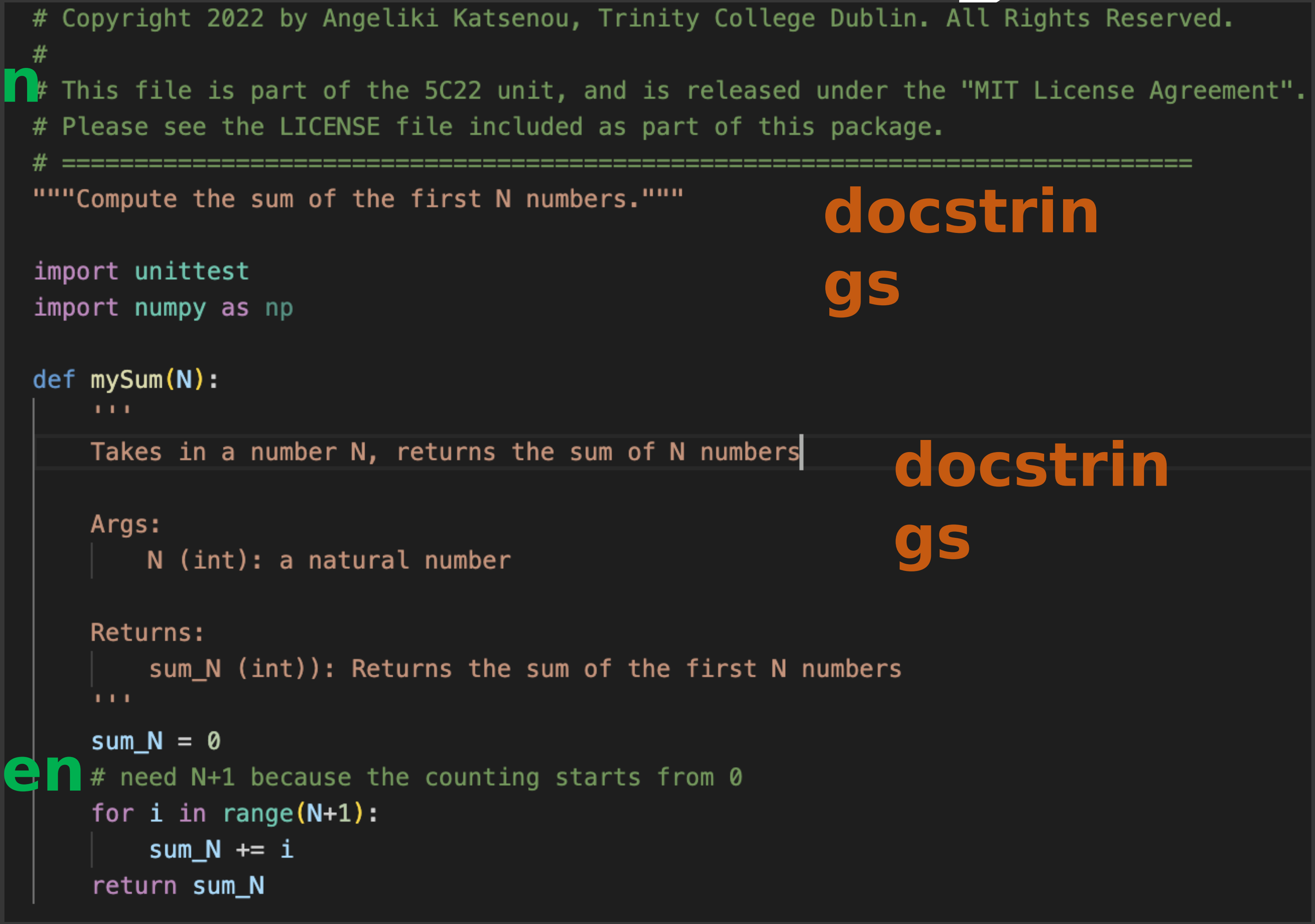
Example of Unit Test



```
● (cenv) angeliki@Angelikis-MacBook-Pro Python_Lessons % python -m unittest test_UnitTest.py -v
  test_mySum (test_UnitTest.TestMyCode) ... ok

  ----------------------------------------------------------------------
  Ran 1 test in 0.000s

  OK
```

Comments and Docstrings in Python

Comments or Docstrings?



```
# Copyright 2022 by Angeliki Katsenou, Trinity College Dublin. All Rights Reserved.
#
# This file is part of the 5C22 unit, and is released under the "MIT License Agreement".
# Please see the LICENSE file included as part of this package.
# ================================================================================
"""Compute the sum of the first N numbers."""

import unittest
import numpy as np

def mySum(N):
    '''
    Takes in a number N, returns the sum of N numbers

    Args:
        N (int): a natural number

    Returns:
        sum_N (int)): Returns the sum of the first N numbers
    '''
    sum_N = 0
    # need N+1 because the counting starts from 0
    for i in range(N+1):
        sum_N += i
    return sum_N
```

comments

docstrings

docstrings

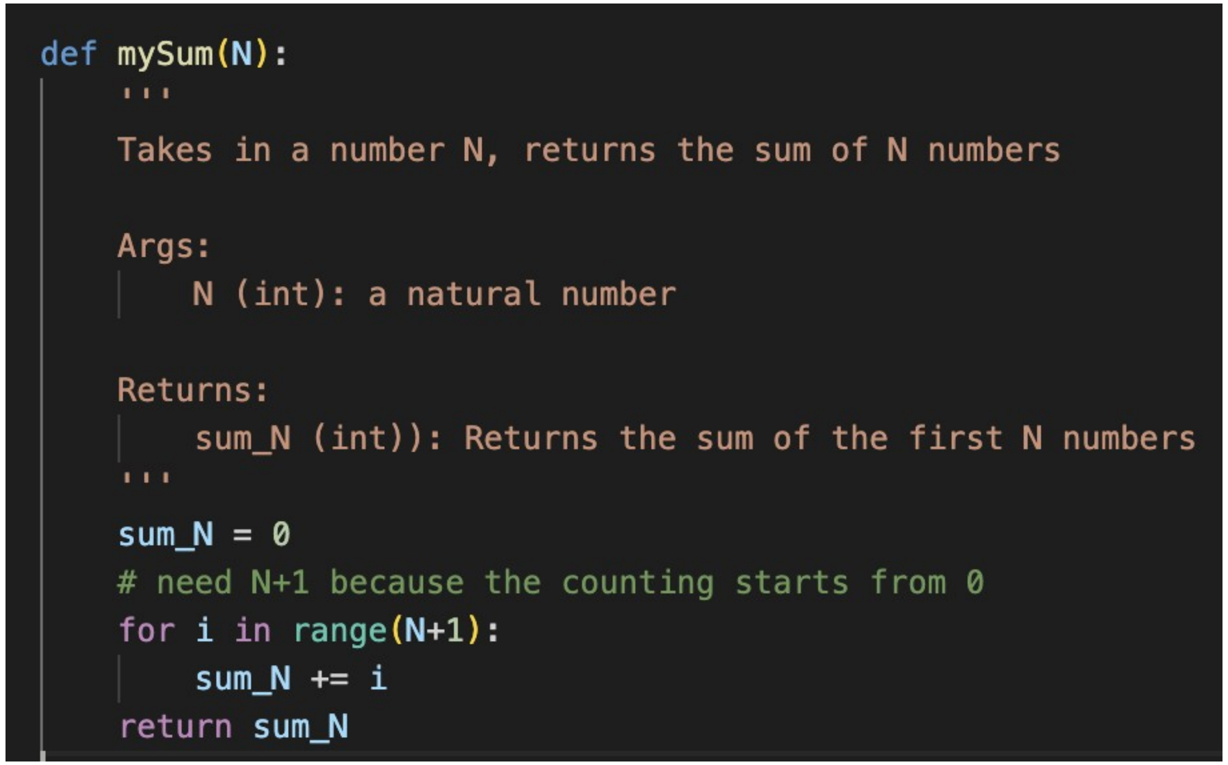
comments

Comments or Docstrings?

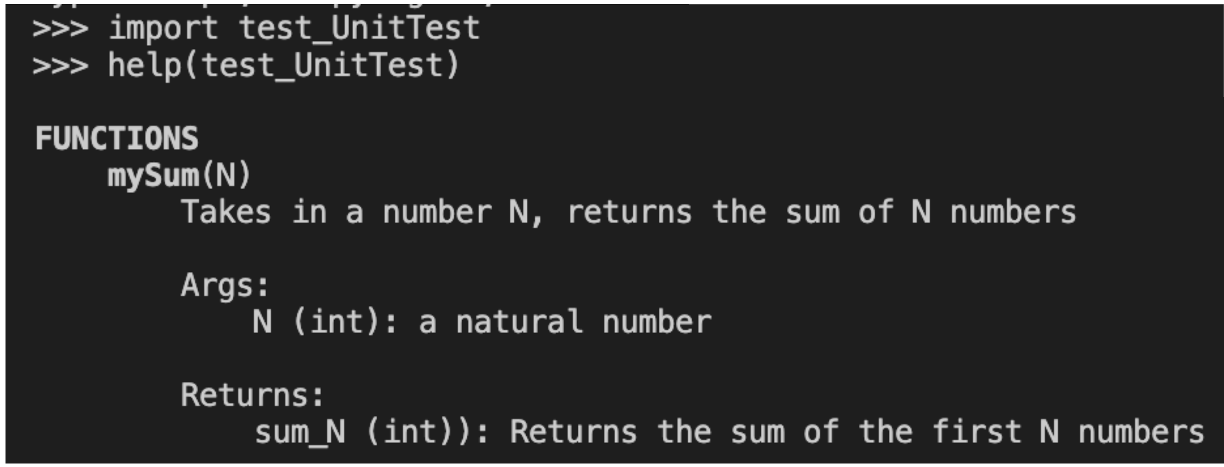
- Comments for preamble of the document
- Comments on specific lines of code - help programmers better understand the intent and functionality of the program.
- Comments are completely ignored by the Python interpreter.

- Docstrings for general description of scripts and functions
- Typically used right after the definition of a function, method, class, or module.
- Docstrings are used to document our code.
- We can access these docstrings usin:
  - the print(myFunction.__doc__) attribute or
  - through import myFunction and then help(myFunction)

Example of using docstrings



```python
def mySum(N):
    '''
    Takes in a number N, returns the sum of N numbers

    Args:
        N (int): a natural number

    Returns:
        sum_N (int)): Returns the sum of the first N numbers
    '''
    sum_N = 0
    # need N+1 because the counting starts from 0
    for i in range(N+1):
        sum_N += i
    return sum_N
```



```
>>> import test_UnitTest
>>> help(test_UnitTest)

FUNCTIONS
    mySum(N)
        Takes in a number N, returns the sum of N numbers

        Args:
            N (int): a natural number

        Returns:
            sum_N (int)): Returns the sum of the first N numbers
```

Readme files

Readme Files – Basics to Include

- Use Markdown Language to write - it is a text-to-HTML conversion tool for web writers.

- Can be written in VS Code editor

- You can render them through VS Code before pushing on Github (right click on the file and select Preview)

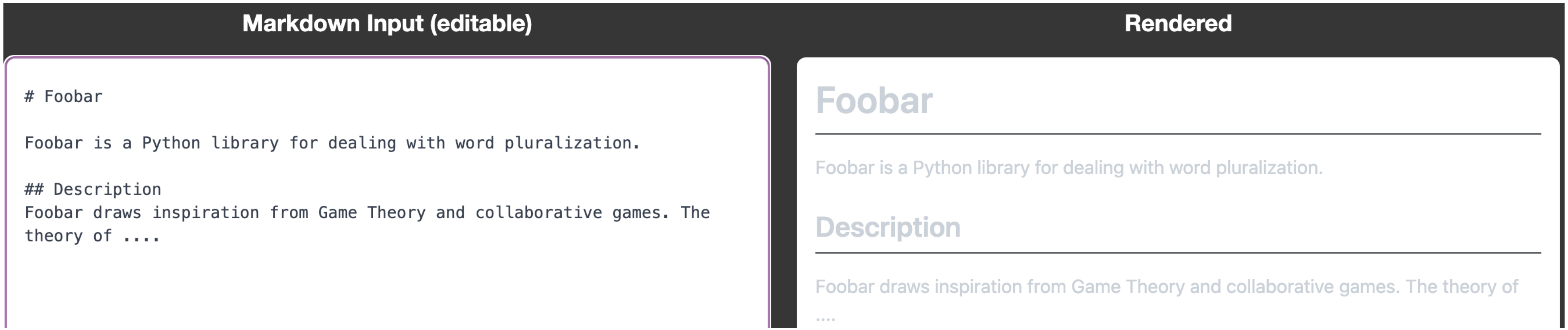
Readme Files – Basics to Include

1. Project's Title
It describes the whole project in one sentence, and helps people understand **what** the main goal and aim of the project is.

2. Project Description
Your description is an extremely important aspect of your project. It gives more information on the algorithms, design, and methodology

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Example



```
# Foobar

Foobar is a Python library for dealing with word pluralization.

## Description
Foobar draws inspiration from Game Theory and collaborative games. The
theory of ....
```

Foobar

Foobar is a Python library for dealing with word pluralization.

Description

Foobar draws inspiration from Game Theory and collaborative games. The theory of ....

Readme Files – Basics to Include

3. How to Install and Run the Project

You should include a file with the requirements/dependencies or include the steps required to install your project.
Provide a step-by-step description of how to get the development environment set and running. Give also a demo file or examples to run.

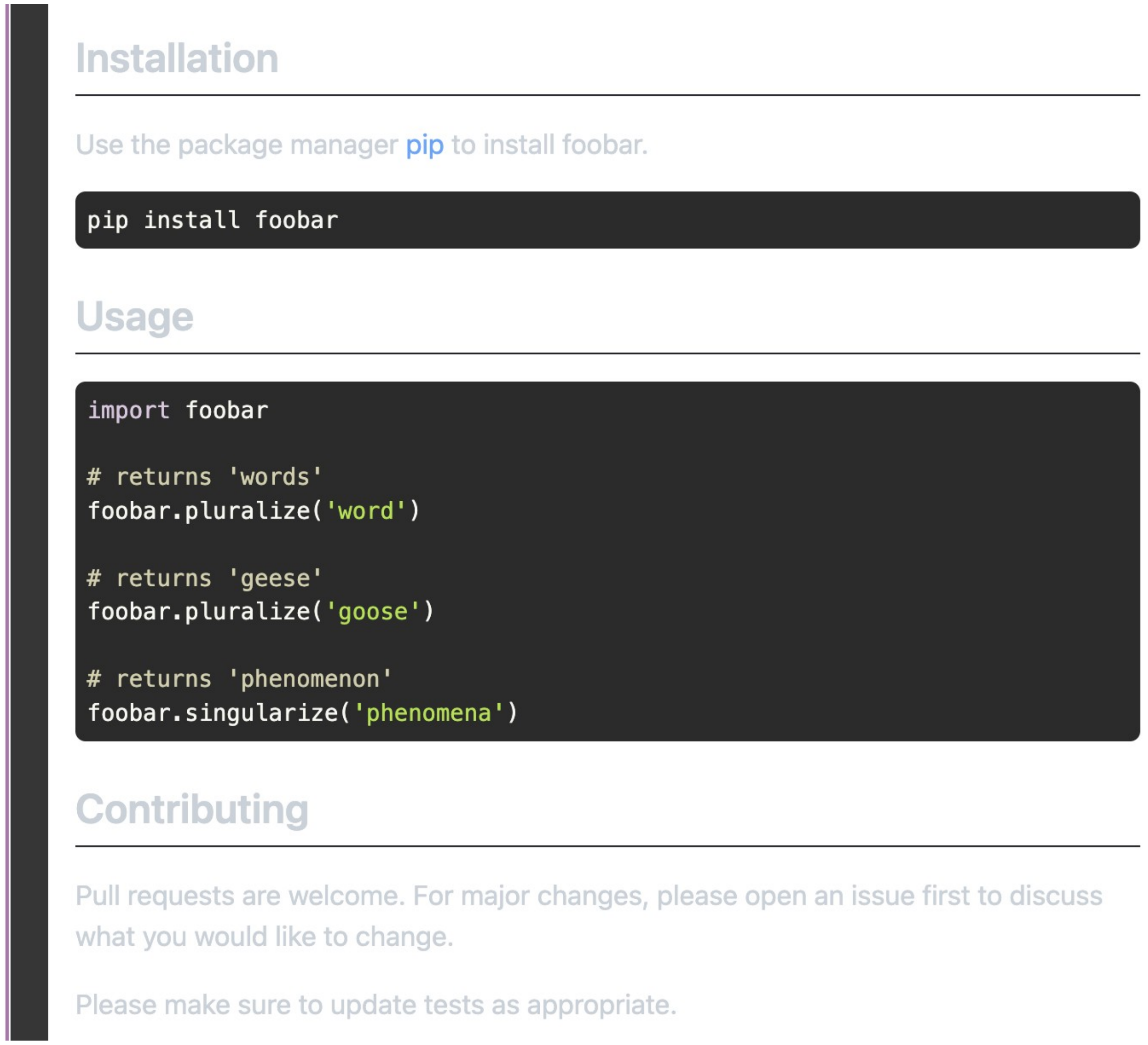
4. Include Credits

If you worked on the project as a team or an organization, list your collaborators/team members.
Also include references you used that might help others to build that particular project.

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Example

~~~
## Installation

Use the package manager [pip](https://pip.pypa.io/en/stable/) to install
foobar.

```bash
pip install foobar
```

## Usage

```python
import foobar

# returns 'words'
foobar.pluralize('word')

# returns 'geese'
foobar.pluralize('goose')

# returns 'phenomenon'
foobar.singularize('phenomena')
```

## Contributing

Pull requests are welcome. For major changes, please open an issue first
to discuss what you would like to change.

Please make sure to update tests as appropriate.
~~~



Installation

Use the package manager pip to install foobar.

```
pip install foobar
```

Usage

```python
import foobar

# returns 'words'
foobar.pluralize('word')

# returns 'geese'
foobar.pluralize('goose')

# returns 'phenomenon'
foobar.singularize('phenomena')
```

Contributing

Pull requests are welcome. For major changes, please open an issue first to discuss what you would like to change.

Please make sure to update tests as appropriate.

Resources

- https://en.wikipedia.org/wiki/Markdown
- https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax
- Online Readme Editors: https://dillinger.io/ , https://www.makeareadme.com
- https://www.freecodecamp.org/news/how-to-write-a-good-readme-file/
- Docstrings: https://www.programiz.com/python-programming/docstrings
- https://numpy.org/doc/stable/user/quickstart.html
- Numpy for matlab users https://numpy.org/doc/stable/user/numpy-for-matlab-users.html
- Progress bars: https://pypi.org/project/progress/
- **Audio Basics in Python:** https://www.it-jim.com/blog/audio-processing-basics-in-python/
- **Unittesting**:
  - https://docs.python.org/3/library/unittest.html
  - https://colab.research.google.com/github/damorimRG/practical_testing_book/blob/master/testgranularity/unittesting.ipynb