Imperial College London

BIOE50010 – Programming 2

Computer Lab 9

Binghuan Li | *Department of Chemical Engineering*

binghuan.li19@imperial.ac.uk

Maria Portela | *Department of Bioengineering*

maria.s-marques-figueiredo-portela18@imperial.ac.uk

December 04, 2023

Coursework 2

- Will be released on ***8th December*** (***Friday***), to be summitted 1 week after.
- Retrospectively, this coursework was designed in the ***object-oriented programming*** paradigm.
- But as you are aware, a solid mastery of the programming basics is *sine qua non* – re-visiting your labs and lectures over the entire term ?
- Fear not – think of how far you have gone !
  - Very impressive (unexceptional) cohort performance in your coursework 1.

Questions should be logged on the ed discussion

End of service time: 9 am on the submission day.

Why We Need Testing? One Example

- Suppose that you are implementing a *numerical* solver, using the 4th-order Runge-Kutta scheme, to solve a 2nd-order ordinary differential equation.

$$
\begin{aligned}
k_{1i} &= f(x_i, y_i), \\
k_{2i} &= f\left(x_i + h/2, y_i + h/2 k_{1i}\right), \\
k_{3i} &= f\left(x_i + h/2, y_i + h/2 k_{2i}\right), \\
k_{4i} &= f(x_i + h, y_i + h k_{3i})
\end{aligned}
$$

guess initial values of x_{11}, y_{11}

f is the first order derivative of *y* w.r.t. *x*

$$y_{i+1} = y_i + \frac{h}{6}(k_{1i} + 2k_{2i} + 2k_{3i} + k_{4i}).$$

calculate new *y*

i=*i*+1, iterate until convergence

- However, you want to check whether your implementation is correct – hence, you come up with a test case, with the known *analytical* solution, to compare with the numerical solution. This is commonly referred to as the **sanity check**.

governing eqn. boundary cond.

$$y' + 2y = x^3 e^{-2x}, \quad y(0) = 1 \Rightarrow y = \frac{e^{-2x}}{4}(x^4 + 4)$$

analytical sol. $\overset{?}{\Leftrightarrow}$ numerical sol. from RK4

Unit Test (1/)

- Lucky us, Python provides a build-in package, **`unittest`**, for the testing purpose, *e.g.*, to check the numerical example.
- **`unittest`** requires that:
  - You put your tests into classes as methods
  - You use a series of special assertion methods in the **`unittest.TestCase`** class

Example from **`test_point_pp.py`**

```
import unittest
import point_pp as point

class TestPointPP(unittest.TestCase):

    def test_add(self):
        result = point.add([10, 2],[1, 7])
        self.assertEqual(result, [11, 9])
```

Access to the testing methods defined in the **`unittest`** package

Method names should begin with a keyword **`test`**

`assertEqual` method evaluates the coherence between the input and output

Unit Test (2/)

- **`unittest`** methods at a glance:

`unittest` Method	Checks that…	**`unittest`** Method	Checks that…
`assertEqual(a,b)`	`a==b`	`assertIsNone(x)`	`x is None`
`assertNotEqual(a,b)`	`a != b`	`assertIsNotNone(x)`	`x is not None`
`assertTrue(x)`	`bool(x) is True`	`assertIn(a, b)`	`a in b`
`assertFalse(x)`	`bool(x) is False`	`assertNotIn(a, b)`	`a not in b`
`assertIs(a,b)`	`a is b`	`assertIsInstance(a, b)`	`isinstance(a, b)`
`assertIs(a,b)`	`a is b`	`assertNotIsInstance(a, b)`	`not isinstance(a, b)`
`assertIsNot(a, b)`	`a is not b`		

- also the `setUp` method and `tearDown` method allows you to config & post-processing your testing objects.

OOP resources

- W3 Schools: Python Classes (w3schools.com)
- Tips: 8 Tips For Object-Oriented Programming in Python - GeeksforGeeks
- Review: Object-Oriented Design (article) | Khan Academy (good for revising concepts – careful, examples are not Python !)
- Object-Oriented Programming (OOP) in Python 3 – Real Python

Efficiency and algorithms resources

- Concepts overview: Introduction to Computation Complex Theory - GeeksforGeeks
- Algorithm analysis (Computer Science): Analysis of Algorithms | Big-O analysis - GeeksforGeeks
- Algorithm analysis (Computer Science): Asymptotic notation (article) | Algorithms | Khan Academy