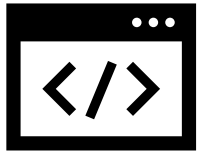
IMPERIAL



BIOE50010 – Programming 2

Computer Lab 9: Unit Tests

Binghuan Li, Maria Portela, Gauthier Boeshertz, Samuel George-White, Yilin Sun, Kamrul Hasan, Wenhao Ding, Siyu Mu, Lito Chatzidavari

1 December, 2025

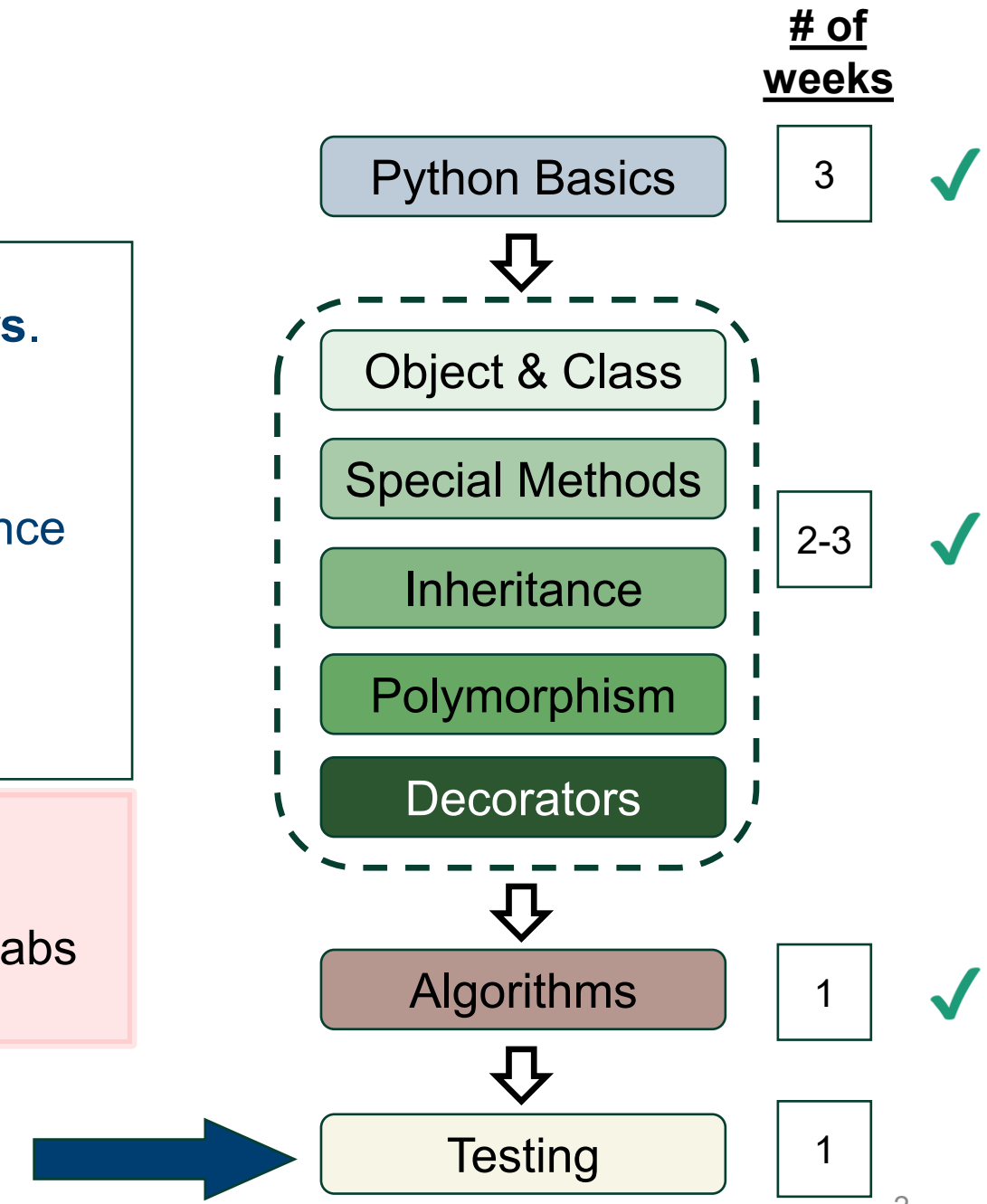
Progress Check

Revision Points (from weeks 8)

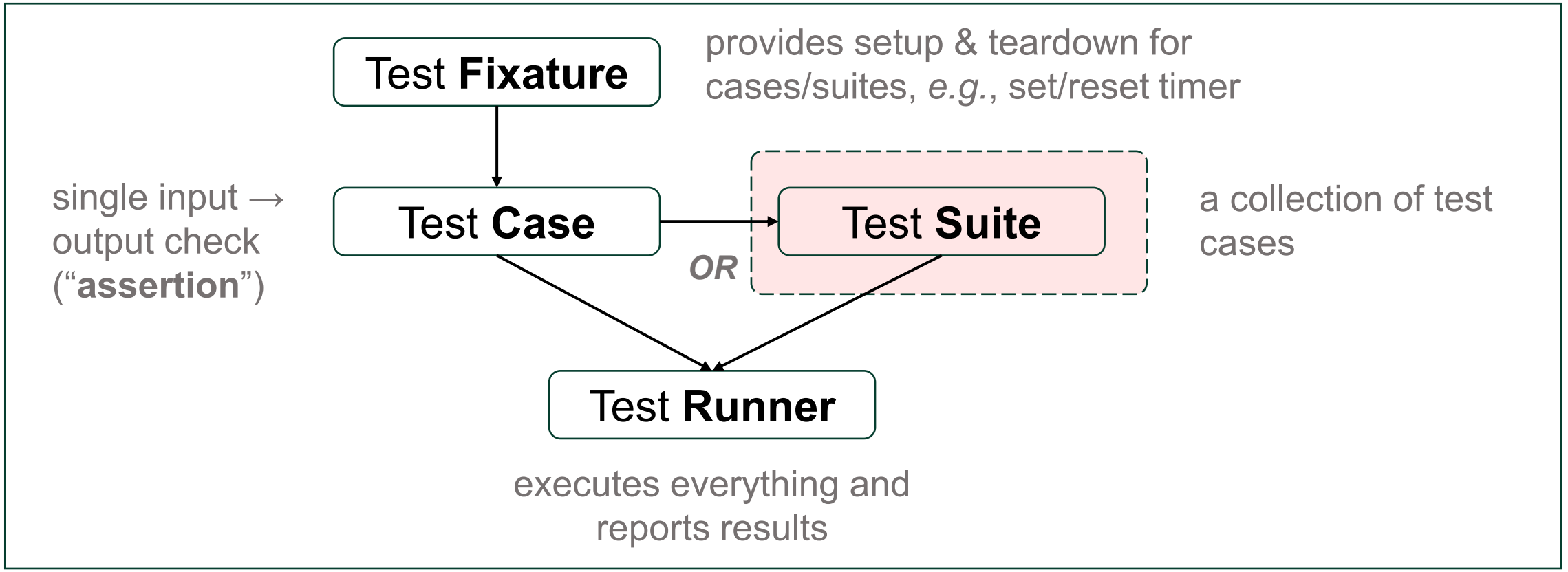
- How to use **wrapper functions** and **decorators**. Understand the flow of execution.
- How to use a **class method** in class. Clearly differentiate between class attributes and instance attributes.
- How to use a **static method** and a **property** decorator in class.

- The assignment will be released on Friday 5th December, 2025.
- There will be no additional tasks for week 10. Labs will be running in a **Q&A mode**.

Week 9:
we are here



Workflow with Unit Test



- `unittest` allows users to customise the tests: either test a single case (test case), or test a collection of cases (test suite).

Unit Test

To define the test cases using **`unittest`**

- Each test case should be defined as a method, with its name starting with the keyword '**`test`**'.
- A series of assertion methods have been defined in **`unittest.TestCase`** class – hence you need to use **inheritance** to access to these methods.

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])
```

Driver (test runner)

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

- You can define multiple test cases within one test class.
- All test cases will run automatically `unittest.main()`

A Coursework Grader (1/)

```
class TestSim(unittest.TestCase):

    @classmethod
    def setUpClass(cls):

    @classmethod
    def tearDownClass(cls):

    def setUp(self):
        self.t0 = time.time()

    def tearDown(self):
        self.dt = round(time.time() - self.t0,8)
        student_time[index].append(self.dt)
```

Test Fixature

```
    # Testing whether the students have not altered the original code
    def test_cat1_0_(self):

    def test_cat1_1_(self):

    def test_cat2_0_(self):

    @timeout_decorator.timeout(20)
    def test_cat6_0_(self):

    @timeout_decorator.timeout(20)
    def test_cat6_1_(self):
```

Test Cases

A Coursework Grader (2/)

```
def suite():
    suite = unittest.TestSuite()
    suite.addTest(TestSim('test_cat1_0_'))
    suite.addTest(TestSim('test_cat1_1_'))
    suite.addTest(TestSim('test_cat2_0_'))
    suite.addTest(TestSim('test_cat6_0_'))
    suite.addTest(TestSim('test_cat6_1_'))
    return suite
```

Test Suite

```
def main():
    runner = unittest.TextTestRunner(verbosity=2, descriptions=0)
    runner.run(suite())
```

Test Runner

Your Task Today

Generate the test examples, and create test cases using module **`unittest`**, perform tests to two functions **`eval_win()`** and **`board_full()`** in the Tic Tac Toe game.

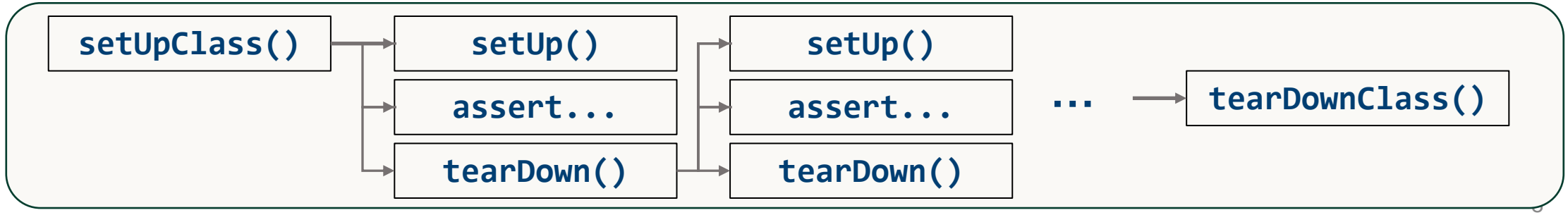
To start…

- Read and study the example Python scripts from your Friday lecture.
- The functions to be tested are given out in `TicTacToe.py` on Blackboard. To start, import them to your script.
- Refer the summaries of the unit test methods (given out the subsequent pages), when necessary.

Appendix 1: Unit Test Methods

- Test fixature methods

Method	Description
`setUp()`	The method is called automatically before running *each* test method in a test case class.
`tearDown()`	The method is called automatically after running *each* test method in a test case class.
`setUpClass()`	The method is called automatically before running the tests in a test case class.
`tearDownClass()`	The method is called automatically after running the tests in a test case class.



Appendix 1: Unit Test Methods

- Test assertion methods

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`		