1- Testing*

Testing is a process in software development where you systematically verify that your code behaves as expected. It involves running your code with specific inputs and checking whether the outputs match the expected results. This helps catch bugs and ensures the correctness and reliability of your software.

Unit-testing

Unit testing is a type of testing where individual units or components of your code are tested in isolation to verify their correctness. In Python, the unittest library provides a framework for writing and running unit tests. It helps you design and execute tests for your functions, methods, and classes to ensure that they work as intended.

1.1- TestCase Class

This is the base class for all test cases. You can create your own test cases by subclassing this class.

In [1]:
```
import unittest

class MyTestCase(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(1 + 2, 3)  # Assertion for equality
```

1.2- Assertions

These are methods provided by the TestCase class to check conditions and report failures.

In [2]:
```
class MyTestCase(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(1 + 2, 3)  # Passes
        self.assertNotEqual(1 + 1, 3)  # Passes
        self.assertTrue(True)  # Passes
        self.assertFalse(False)  # Passes
```

1.3- Setup and Teardown Methods

These methods are called before and after each test method to set up and clean up resources.

In [3]:
```
class MyTestCase(unittest.TestCase):
    def setUp(self):
        self.data = [1, 2, 3]

    def tearDown(self):
        self.data = None

    def test_list_length(self):
        self.assertEqual(len(self.data), 3)  # Passes
```

1.4- Test Discovery

The unittest library can discover and run test cases in a specified directory.

```
In [4]: # test_module.py
import unittest

class MyTestCase(unittest.TestCase):
    def test_multiply(self):
        self.assertEqual(2 * 3, 6)  # Passes

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

# Run the tests
# python test_module.py
```

```
E
======================================================================
ERROR: C:\Users\EURO TEC COMPUTERS\AppData\Roaming\jupyter\runtime\kernel-0944d924-93c3-49c6-8c1f-0ae1282a4c
ac (unittest.loader._FailedTest)
----------------------------------------------------------------------
AttributeError: module '__main__' has no attribute 'C:\Users\EURO TEC COMPUTERS\AppData\Roaming\jupyter\runt
ime\kernel-0944d924-93c3-49c6-8c1f-0ae1282a4cac'

----------------------------------------------------------------------
Ran 1 test in 0.003s

FAILED (errors=1)

An exception has occurred, use %tb to see the full traceback.

SystemExit: True



C:\Users\EURO TEC COMPUTERS\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3452: UserWarning:
To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

1.5- Test Suites

You can group multiple test cases into a test suite.

In [ ]:
```python
import unittest

class MyTestCase(unittest.TestCase):
    def test_case1(self):
        self.assertEqual(1, 1)  # Passes

    def test_case2(self):
        self.assertNotEqual(1, 2)  # Passes

if __name__ == '__main__':
    suite = unittest.TestLoader().loadTestsFromTestCase(MyTestCase)
    unittest.TextTestRunner().run(suite)
```

2- Debugger

A debugger in Python is a tool that allows you to interactively inspect and analyze the execution of your code, helping you identify and fix errors (bugs) more efficiently. It provides features like setting breakpoints, stepping through code, inspecting variables, and evaluating expressions during runtime. The built-in debugger in Python is called pdb (Python Debugger), and there are also third-party alternatives like pdb++, ipdb, and integrated debugger support in many IDEs (Integrated Development Environments).

In [ ]:
```python
def divide(a, b):
    result = a / b
    return result

def main():
    x = 10
    y = 2
    result = divide(x, y)
    print(f"Result: {result}")

if __name__ == "__main__":
    import pdb
    pdb.set_trace()  # Start the debugger here
    main()
```

When you run this script, it will start the debugger at the pdb.set_trace() line. You can interact with the debugger using commands like:

n or next: Execute the current line and move to the next line. s or step: Step into a function (if there is a function call on the current

3- Decorators

Decorators in Python are functions that modify the behavior of another function or method. They allow you to wrap or enhance the functionality of a function without modifying its code directly. Decorators are widely used for tasks like logging, authorization, caching, and more.

A decorator is a function that takes another function as an argument, adds some functionality, and returns the modified function. It allows you to "decorate" functions, hence the name "decorator."

In [ ]:
```
import time

def timing_decorator(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Time taken by {func.__name__}: {end_time - start_time:.4f} seconds")
        return result
    return wrapper

@timing_decorator
def slow_function():
    time.sleep(2)
    print("Function executed")

slow_function()
```

4-Lambda

Lambda functions in Python are small, anonymous functions that can have any number of arguments, but can only have one expression. They are often used for short, simple operations where a full function definition would be overkill.

In [5]:
```
# Lambda function to square a number
square = lambda x: x ** 2

# Using the lambda function
result = square(5)
print(result)
```

```
25
```

In [6]:
```
# Lambda function to calculate cube
cube = lambda x: x ** 3

# Using the lambda function for cube
result_cube = cube(4)
print("Cube:", result_cube)

# Lambda function to raise to the power of 4
power_four = lambda x: x ** 4

# Using the lambda function for power of 4
result_power_four = power_four(3)
print("Power of 4:", result_power_four)
```

```
Cube: 64
Power of 4: 81
```

In [ ]:
```

```