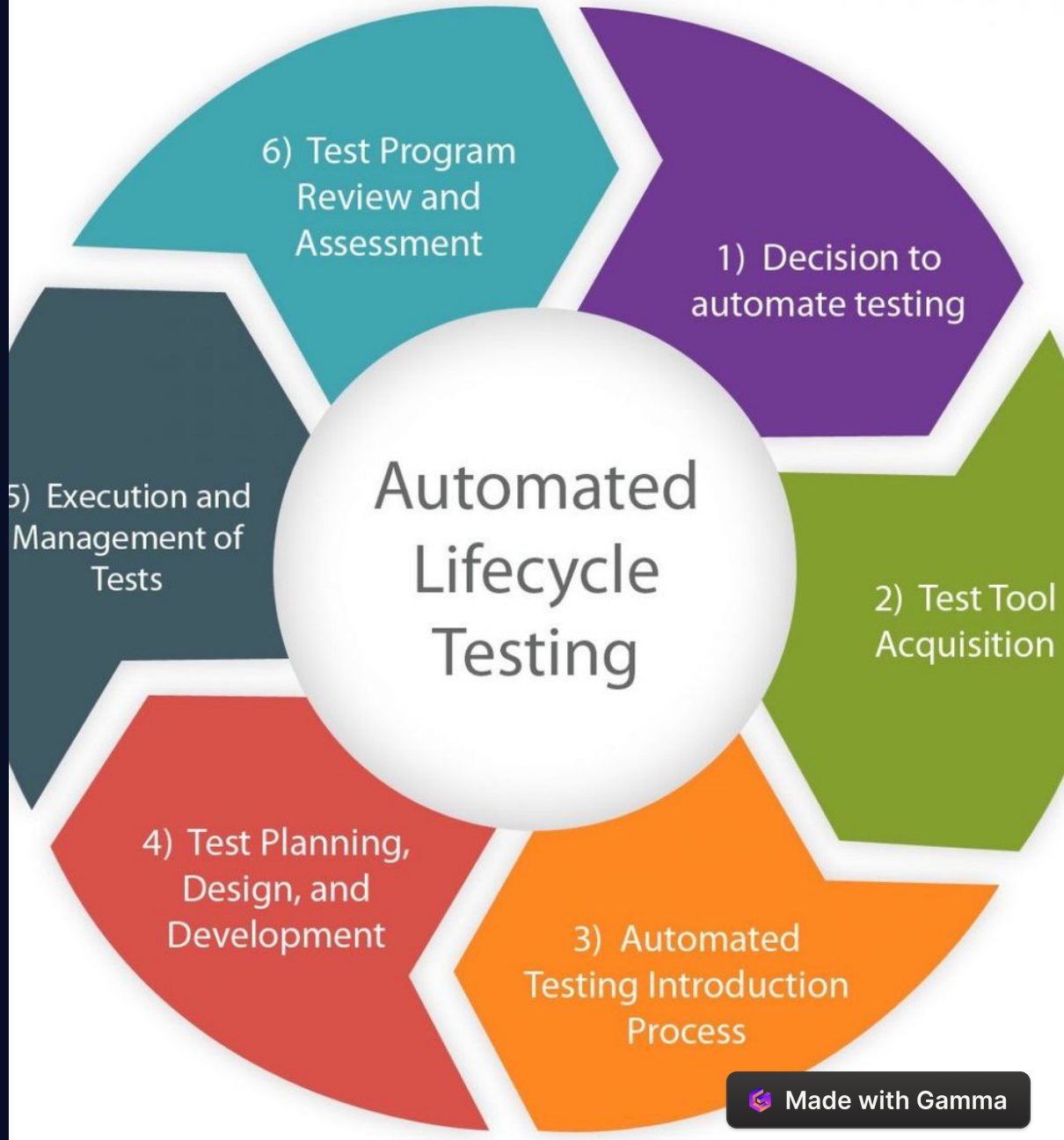
Automate Software Testing with Python

Software testing is crucial to ensure the quality of your applications. In this presentation, we will explore how Python can be used to automate software testing, saving time and improving efficiency.



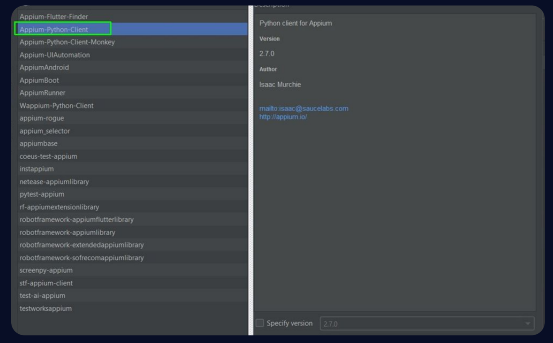
Common Tools and Frameworks for Automated Testing in Python

Selenium

Selenium is a popular framework for web application testing, providing powerful capabilities for automating browser interactions.

Pytest

Pytest is a Python testing framework that simplifies the process of writing and running tests by providing a concise and intuitive syntax.



Appium

Appium is widely used for mobile application testing, supporting both Android and iOS platforms using a single API.

Writing Automated Tests using Python

1

Test Planning

Define test requirements, scenarios, and objectives before designing and implementing the test scripts.

2

Test Script Creation

Write test scripts using Python, incorporating relevant libraries and frameworks to interact with the application under test.

3

Test Data Preparation

Generate or gather test data required to simulate real-world scenarios and cover different test cases.

Running and Analyzing Automated Tests

1

Execution

Run automated tests across various platforms, configurations, and test environments to validate application functionality.

2

Result Analysis

Analyze test results and generate reports, identifying failures, defects, and areas for improvement.

3

Bug Reporting

Report identified bugs and issues following standard procedures, ensuring efficient communication and collaboration with the development team.

Program:

```
#A sample python code to automate the testing of the sample class having basic functionalities of to perform the mathematical operations on the specified operands of that particular
function.

# All the required modules which are required across the program are included at the beginning of the code.
import unittest
import sys

A class is written which has different functions to perform the different mathematical operations depending upon the input provided the mathematical operation is performed on the two
operands which are taken as input for each function,  the class which is written below is the child class of the test case class from the unit test module,  with the help of this parenting
test case class we were able to use various inbuilt functions of the test case class which help us to write various test case function to test the mathematical results which are calculated
by the various function written inside this class
class BasicMathOperationsTesting(the unit test.TestCase):
#Constructor is written which can be used to initialize the various class variable that is required to use throughout this class a variable that is declared inside the constructor of a
class can be used across the scope of that particular class
    def __init__(self):
        pass


    def perform_addition(self):
        operation_name = "addition"
        print("Enter the first number for {} operation".format(operation_name))
        num_1 = int(input())
        print("Enter the second number for {} operation".format(operation_name))
        num_2 = int(input())

        resultant = num_1 + num_2
        return resultant


    def perform_subtraction(self):
        operation_name = "subtraction"
        print("Enter the first number for {} operation".format(operation_name))
        num_1 = int(input())
        print("Enter the second number for {} operation".format(operation_name))
        num_2 = int(input())

        resultant = num_1 - num_2
        return resultant
```

```
    def perform_multiplication(self):
        operation_name = "multiplication"
        print("Enter the first number for {} operation".format(operation_name))
        num_1 = int(input())
        print("Enter the second number for {} operation".format(operation_name))
        num_2 = int(input())

        resultant = num_1 + num_2
        return resultant

    def perform_division(self):
        operation_name = "division"
        print("Enter the first number for {} operation".format(operation_name))
        num_1 = int(input())
        print("Enter the second number for {} operation".format(operation_name))
        num_2 = int(input())

        resultant = num_1 / num_2
        return resultant


    def the unit test_for_assert_equals(self,actual_value,expected_value):

        actual_value_for_equals = actual_value
        expected_value_for_equals = expected_value
        self.assertEqual(actual_value_for_equals,expected_value_for_equals)


    def the unit test_for_assert_true(self,operand1,operand2):

        boolean_resultant = operand1 == operand2
        self.assertTrue(boolean_resultant)

    def the unit test_for_assert_false(self,operand1,operand2):

        boolean_resultant = operand1 != operand2
        self.assertFalse(boolean_resultant)
```

```
def main():

    run_the unit test =  BasicMathOperationsTesting()

    while(True):

        # from the listed below the list of operations select any one of the operations
        print("Select any of the mathematical operations which are listed below:")
        print("1. To perform the addition operation and then perform the unit test on the result obtained.")
        print("2. To perform the subtraction operation and then perform the unit test on the result obtained.")
        print("3. To perform the multiplication operation and then perform the unit test on the result obtained.")
        print("4. To perform the division operation and then perform the unit test on the result obtained.")
        print("5. To exit from the code execution.")

        menu_choice = input()
        menu_choice = int(menu_choice)

        if menu_choice == 1:
            result = run_the unit test.perform_addition()
        elif menu_choice == 2:
            result = run_the unit test.perform_subtraction()
        elif menu_choice == 3:
            result = run_the unit test.perform_multiplication()
        elif menu_choice == 4:
            result = run_the unit test.perform_division()
        elif menu_choice == 5:
            sys.exit()

        print("Select any of the unit tests to perform which are listed below:")
        print("1. To perform the assertEqual the unit test on the above done mathematical operation.")
        print("2. To perform the assertTrue the unit test on the above done mathematical operation.")
        print("3. To perform the assertFalse the unit test on the above done mathematical operation.")

        menu_choice_for_unitttest = input()
        menu_choice_for_unitttest = int(menu_choice_for_unitttest)

        if menu_choice_for_unitttest == 1:
            print("Expected value for test to pass:")
            expected_value = int(input())
            run_the unit test.the unit test_for_assert_equals(result,expected_value)
```

```
        elif menu_choice_for_unitttest == 2:
            print("Expected value for test to pass:")
            expected_value = int(input())
            run_the unit test.the unit test_for_assert_true(result,expected_value)
        elif menu_choice_for_unitttest == 3:
            print("Expected value for test to pass:")
            expected_value = int(input())
            run_the unit test.the unit test_for_assert_false(result,expected_value)


        print("To go on with the code getting executed, enter input [y] or [n]")
        continue_or_exit = input()

        if continue_or_exit == 'y' or continue_or_exit == 'Y':
            pass
        elif continue_or_exit == 'n' or continue_or_exit == 'N':
            sys.exit()

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

output

```
Select any of the mathematical operations which are listed below:
1. To perform the addition operation and then perform the unit test on the result obtained.
2. To perform the subtraction operation and then perform the unit test on the result obtained.
3. To perform the multiplication operation and then perform the unit test on the result obtained.
4. To perform the division operation and then perform the unit test on the result obtained.
5. To exit from the code execution.
1
Enter the first number for the addition operation
99
Enter the second number for the addition operation
578
Select any of the unit tests to perform which are listed below:
1. To perform the assertEqual the unit test on the above done mathematical operation.
2. To perform the assertTrue the unit test on the above done mathematical operation.
3. To perform the assertFalse the unit test on the above done mathematical operation.
1
The expected value for the test to pass:
677
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

```
To go on with the code getting executed, enter input [y] or [n]
y
Select any of the mathematical operations which are listed below:
1. To perform the addition operation and then perform the unit test on the result obtained.
2. To perform the subtraction operation and then perform the unit test on the result obtained.
3. To perform the multiplication operation and then perform the unit test on the result obtaine
4. To perform the division operation and then perform the unit test on the result obtained.
5. To exit from the code execution.
1
Enter the first number for the addition operation
100
Enter the second number for the addition operation
544
Select any of the unit tests to perform which are listed below:
1. To perform the assertEqual the unit test on the above done mathematical operation.
2. To perform the assertTrue the unit test on the above done mathematical operation.
3. To perform the assertFalse the unit test on the above done mathematical operation.
1
The expected value for the test to pass:
647



self.assertEqual(actual_value_for_equals,expected_value_for_equals)
AssertionError: 647 != 677
----------------------------------------------------------------------
Ran 1 test in 0.001s
```

```
FAILED (failures=1)


To go on with the code getting executed, enter input [y] or [n]
y
Select any of the mathematical operations which are listed below:
1. To perform the addition operation and then perform the unit test on the result obtained.
2. To perform the subtraction operation and then perform the unit test on the result obtained.
3. To perform the multiplication operation and then perform the unit test on the result obtained.
4. To perform the division operation and then perform the unit test on the result obtained.
5. To exit from the code execution.
2
Enter the first number for the subtraction operation
122
Enter the second number for the subtraction operation
22
Select any of the unit tests to perform which are listed below:
1. To perform the assertEqual the unit test on the above done mathematical operation.
2. To perform the assertTrue the unit test on the above mathematical operation.
3. To perform the assertFalse the unit test on the above done mathematical operation.
2
The expected value for the test to pass:
100
----------------------------------------------------------------------
Ran 1 test in 0.000s
```

```
OK
To go on with the code getting executed, enter input [y] or [n]
y
Select any of the mathematical operations which are listed below:
1. To perform the addition operation and then perform the unit test on the result obtained.
2. To perform the subtraction operation and then perform the unit test on the result obtained.
3. To perform the multiplication operation and then perform the unit test on the result obtained.
4. To perform the division operation and then perform the unit test on the result obtained.
5. To exit from the code execution.
2
Enter the first number for the subtraction operation
255
Enter the second number for the subtraction operation
50
Select any of the unit tests to perform which are listed below:
1. To perform the assertEqual the unit test on the above done mathematical operation.
2. To perform the assertTrue the unit test on the above done mathematical operation.
3. To perform the assertFalse the unit test on the above done mathematical operation.
2
The expected value for the test to pass:
200

self.assertTrue(boolean_resultant)
AssertionError: False
----------------------------------------------------------------------
Ran 1 test in 0.001s
```

```
FAILED (failures=1)


To go on with the code getting executed, enter input [y] or [n]
y
Select any of the mathematical operations which are listed below:
1. To perform the addition operation and then perform the unit test on the result obtained.
2. To perform the subtraction operation and then perform the unit test on the result obtained.
3. To perform the multiplication operation and then perform the unit test on the result obtained
4. To perform the division operation and then perform the unit test on the result obtained.
5. To exit from the code execution.
3
Enter the first number for the multiplication operation
100
Enter the second number for the multiplication operation
2
Select any of the unit tests to perform which are listed below:
1. To perform the assertEqual the unit test on the above done mathematical operation.
2. To perform the assertTrue the unit test on the above done mathematical operation.
3. To perform the assertFalse the unit test on the above done mathematical operation.
2
The expected value for the test to pass:
200
----------------------------------------------------------------------
Ran 1 test in 0.000s
```

```
To go on with the code getting executed, enter input [y] or [n]
y
Select any of the mathematical operations which are listed below:
1. To perform the addition operation and then perform the unit test on the result obtained.
2. To perform the subtraction operation and then perform the unit test on the result obtained.
3. To perform the multiplication operation and then perform the unit test on the result obtained
4. To perform the division operation and then perform the unit test on the result obtained.
5. To exit from the code execution.
3
Enter the first number for the multiplication operation
122
Enter the second number for the multiplication operation
654
Select any of the unit tests to perform which are listed below:
1. To perform the assertEqual the unit test on the above done mathematical operation.
2. To perform the assertTrue the unit test on the above done mathematical operation.
3. To perform the assertFalse the unit test on the above done mathematical operation.
2
The expected value for the test to pass:
55412

self.assertTrue(boolean_resultant)
AssertionError: False
----------------------------------------------------------------------
Ran 1 test in 0.001s
```

```
FAILED (failures=1)

To go on with the code getting executed, enter input [y] or [n]
y
Select any of the mathematical operations which are listed below:
1. To perform the addition operation and then perform the unit test on the result obtained.
2. To perform the subtraction operation and then perform the unit test on the result obtained.
3. To perform the multiplication operation and then perform the unit test on the result obtained.
4. To perform the division operation and then perform the unit test on the result obtained.
5. To exit from the code execution.
4
Enter the first number for division operation
55
Enter the second number for division operation
5
Select any of the unit tests to perform which are listed below:
1. To perform the assertEqual the unit test on the above done mathematical operation.
2. To perform the assertTrue the unit test on the above done mathematical operation.
3. To perform the assertFalse the unit test on the above done mathematical operation.
3
The expected value for a test to pass:
12

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

```
OK
To go on with the code getting executed, enter input [y] or [n]
y
Select any of the mathematical operations which are listed below:
1. To perform the addition operation and then perform the unit test on the result obtained.
2. To perform the subtraction operation and then perform the unit test on the result obtained.
3. To perform the multiplication operation and then perform the unit test on the result obtained.
4. To perform the division operation and then perform the unit test on the result obtained.
5. To exit from the code execution.
4
Enter the first number for division operation
100
Enter the second number for division operation
10
Select any of the unit tests to perform which are listed below:
1. To perform the assertEqual the unit test on the above done mathematical operation.
2. To perform the assertTrue the unit test on the above done mathematical operation.
3. To perform the assertFalse the unit test on the above done mathematical operation.
3
The expected value for the test to pass:
10

self.assertFalse(boolean_resultant)
AssertionError: False
----------------------------------------------------------------------
Ran 1 test in 0.001s
```

```
FAILED (failures=1)


To go on with the code getting executed, enter input [y] or [n]
y
Select any of the mathematical operations which are listed below:
1. To perform the addition operation and then perform the unit test on the result obtained.
2. To perform the subtraction operation and then perform the unit test on the result obtained.
3. To perform the multiplication operation and then perform the unit test on the result obtained
4. To perform the division operation and then perform the unit test on the result obtained.
5. To exit from the code execution.
```

Best Practices for Automated Software Testing in Python

Modularity

Design test scripts that are modular and reusable, enabling easy maintenance and scalability of the test suite.

Data-Driven Testing

Utilize external data sources, such as CSV or Excel files, to validate test scenarios with different inputs and expected outputs.

Error Handling

Implement appropriate exception handling mechanisms to gracefully capture and handle errors encountered during test execution.

Continuous Integration

Integrate automated testing into the software development lifecycle, leveraging tools like Jenkins or GitLab CI/CD for seamless integration and execution.