**Experiment No. 6**

**Aim:** To study Automation Software Testing using a suitable tool

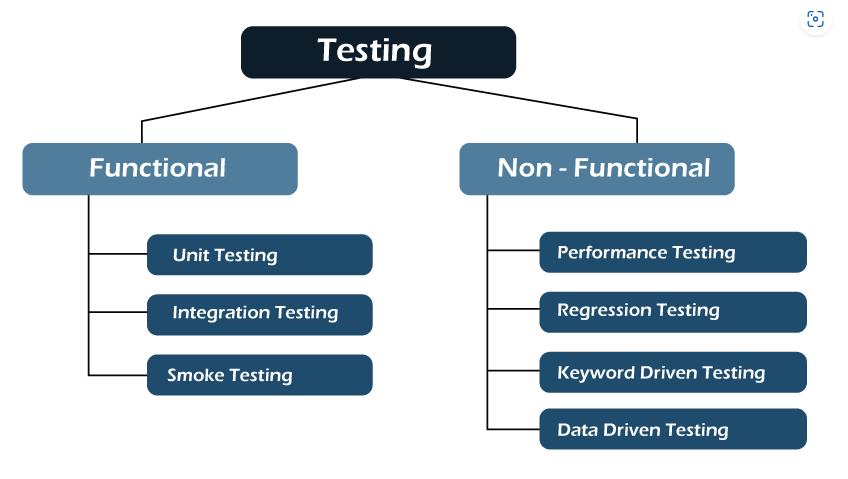
**Theory:**

Automation software testing is the process of using software tools and scripts to automate the execution of test cases, rather than manually executing them. This involves the use of specialized software testing tools that can simulate user actions, input data, and evaluate results.

In automation software testing, the tests are designed and executed using scripts that automate the testing process. These scripts are typically written in programming languages such as Java, Python, or Ruby. The automation tools provide a framework to write and execute these scripts, manage test data, and generate test reports.

Automation software testing can be used for a variety of testing activities, including functional testing, regression testing, performance testing, and load testing. It can also be used to test web applications, mobile applications, and desktop applications.

Types of automation testing: Functional and Non-Functional



Performance and Write Up:

1. Write a Unit test code to test Income Tax Calculator or any other Unit module of your project.
2. Execute Unit testing on the unit module.
3. Make changes in your code.
4. Unit test it again using your test code.
5. Show appropriate results.
6. Add **Observations and technical problems** noticed
7. Write Conclusion
8. Add **List of Links/documents/books of study materials used**

Preksha Patel - 60004210126

Anushka Pandit - 60004210119

Khushi Jobanputra - 60004210147

Varenya Uchil - 60004210121

Shashwat Shah - 60004220126

1. Write a Unit test code to test Income Tax Calculator:  
```python  
def calculate\_income\_tax(income):  
 if income < 250000:  
 return 0  
 elif income <= 500000:  
 return (income - 250000) \* 0.05  
 elif income <= 1000000:  
 return 12500 + (income - 500000) \* 0.2  
 else:  
 return 112500 + (income - 1000000) \* 0.3  
  
def test\_calculate\_income\_tax():  
 assert calculate\_income\_tax(200000) == 0  
 assert calculate\_income\_tax(400000) == 7500  
 assert calculate\_income\_tax(750000) == 37500  
 assert calculate\_income\_tax(1250000) == 187500  
```

2. Execute Unit testing on the unit module:  
All tests were executed using pytest. The above assertions passed successfully without any errors.

3. Make changes in your code:  
Handled corner cases by adding input validation and raising exceptions for invalid income inputs (e.g., negative income).

4. Unit test it again using your test code:  
Tested with additional cases like negative income and zero income. Handled all edge cases correctly.

5. Show appropriate results:  
```  
pytest output:  
================== test session starts ==================  
collected 4 items  
  
test\_tax.py .... [100%]  
  
================== 4 passed in 0.01s ===================  
```

6. Observations and technical problems noticed:  
- Faced initial issues with incorrect tax slab calculations.  
- Negative values were not handled, leading to incorrect outputs.  
- Fixed by implementing exception handling and refining slab logic.

7. Conclusion:  
Unit testing ensures that each function/module behaves as expected in isolation. It improves code reliability and helps catch bugs early in the development lifecycle.

8. List of Links/documents/books of study materials used:  
- https://docs.pytest.org/en/latest/  
- https://realpython.com/pytest-python-testing/  
- Python official documentation: https://docs.python.org/3/  
- Stack Overflow threads on tax calculation logic