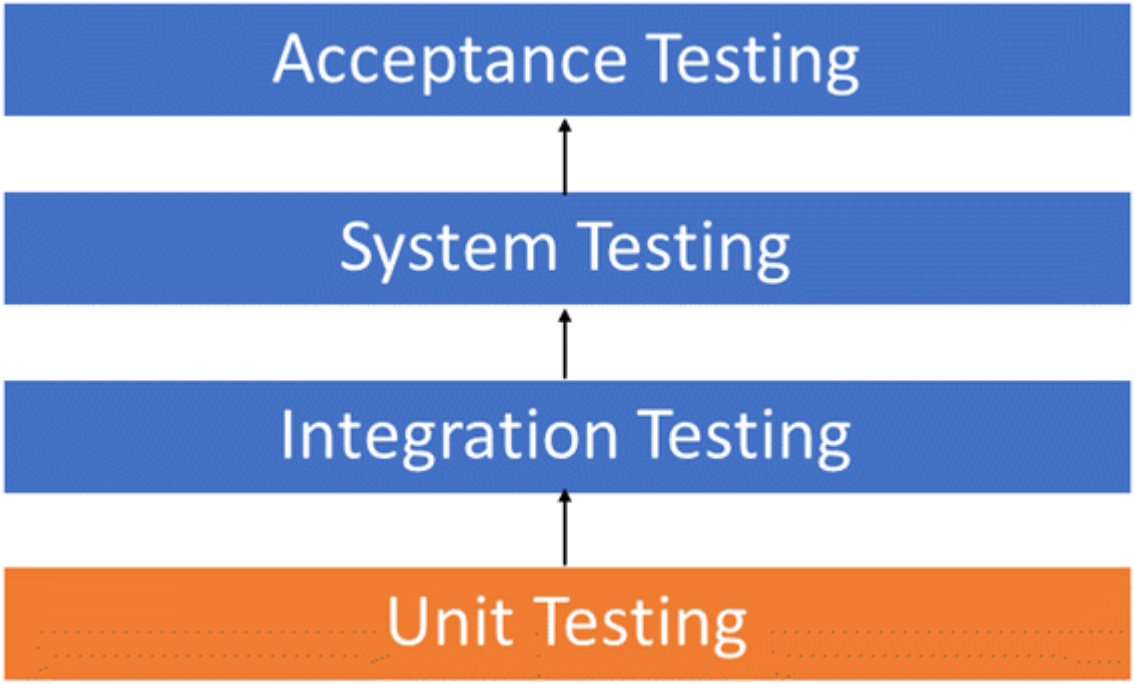
Unit Testing in Python (with pytest)

Software Testing

Software testing can be stated as the process of verifying and validating that a software or application is bug free, meets the technical requirements as guided by it's design and development and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases.

Levels of software testing:



1. **Unit Testing**: A level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
2. **Integration Testing**: A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

3. **System Testing**: A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.
4. **Acceptance Testing**: A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

Types of software testing:

1. **Manual Testing**: Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug.
2. **Automated Testing**: Automated testing is the execution of your test plan by a script instead of a human. It is used to re-run the manually performed test scenarios quickly and repeatedly.

Unit Testing in Python

The three most popular test runners in Python are:

unittest (https://docs.python.org/3/library/unittest.html)

Built into the Python standard library.

2 major requirements of `unittest` :

- Tests are written in a class which inherits from `unittest.TestCase` .
- Forced to use a series of special assertion methods in the `unittest.TestCase` class instead of the built-in assert statement

nose or nose2 (https://docs.nose2.io/en/latest/)

```
$ pip install nose2
```

Nose's tagline is " `nose` extends `unittest` to make testing easier".

`nose` is compatible with any tests written using the `unittest` framework and can be used as a drop-in replacement for the `unittest` test runner. The development of nose as an open-source application fell behind, and a fork called nose2 was created.

pytest (https://docs.pytest.org/en/latest/)

```
$ pip install pytest
```

`pytest` supports execution of `unittest` test cases. `pytest` test cases are a series of functions in a Python file either starting with `test_` or ending with `_test` . Despite the fact that it reduces boilerplate code to the minimum, it still remains readable.

Other great features of pytest:

- Support for the built-in assert statement instead of using special `self.assert*()` methods
- Auto-discovery of test modules and functions
- Support for filtering of test cases
- Ability to rerun from the last failing test
- An ecosystem of hundreds of plugins to extend the functionality

In [ ]:
```
1
```