Python Notes

Classes

Python OOP 1 - Classes and Instances - [https://youtu.be/ZDa-Z5JzLYM](https://www.youtube.com/watch?v=ZDa-Z5JzLYM&t=0s)

Python OOP 2 - Class Variables - [https://youtu.be/BJ-VvGyQxho](https://www.youtube.com/watch?v=BJ-VvGyQxho&t=0s)

Python OOP 3 - Classmethods and Staticmethods - [https://youtu.be/rq8cL2XMM5M](https://www.youtube.com/watch?v=rq8cL2XMM5M&t=0s)

Python OOP 4 - Inheritance - [https://youtu.be/RSl87lqOXDE](https://www.youtube.com/watch?v=RSl87lqOXDE&t=0s)

Python OOP 5 - Special (Magic/Dunder) Methods - [https://youtu.be/3ohzBxoFHAY](https://www.youtube.com/watch?v=3ohzBxoFHAY&t=0s)

Python OOP 6 - Property Decorators - [https://youtu.be/jCzT9XFZ5bw](https://www.youtube.com/watch?v=jCzT9XFZ5bw&t=0s)

***Python Virtua environment creation***

python -m venv .env

source .env/bin/activate

source .env/bin/activate

pip install pytest

pipenv – three

pipenv shell to activate virtual environment

pipenv install pytest

to execute all tests python -m pytest

to execute specific test pytest

python -m pytest –help(learn about pytest's available command line options by using the --help or -h option:)

python -m pytest –verbose or -v(pytest prints more data.)

python -m pytest –quiet or -q(pytest doesn't print the top banner or even the test modules)

python -m pytest --exitfirst or -x option. ( Pytest also provides a mechanism for stopping test execution after failures happen.By default, pytest will run all tests it finds, no matter how many failures happen.)

python -m pytest -- maxfail =1 (to stop running tests after a certain number of failures, use the --maxfail argument and set it to however many failures you want to stop after. For example python -m pytest -- maxfail =1

**unit-xml**

The pytest console output is helpful when manually triggering tests, but sometimes you might need to save test results to a file.

Many tools like Continuous Integration servers need test result files in order to report test results.

One of the most common formats is JUnit's XML report form.

Pytest includes an option that will generate a JUnit XML file for test results.

Simply use --junit-xml and add a report file path like so:

python -m pytest --junit-xml report.xml

**FILTERS**

to run a specific test folder command is

python -m pytest tests

to run a specific test module command is

python -m pytest --verbose tests/test\_accum.py

to run a specific testcase in a module command is (use double colon)

python -m pytest --verbose tests/test\_accum.py::test\_accumulator\_add\_one

to run all test cases containing a string “one”

python -m pytest -k one

to run all test cases containing a string “one and not accum”

python -m pytest -v -k 'one and not accum'

**Markers**

@pytest. mark.accumulator

@pytest. mark.math

Add markers ini file

python -m pytest -m math

When the tests run, you'll see that only test containing the given marker are executed. Nice.

You can also use "and", "or", and "not" boolean expressions with -m, just like you could with -k' expressions. Here are some of pytest's standard markers:

* "skip" will skip the test case.
* "skipif" will skip the test case based on a given condition. For example, tests may not be applicable for certain operating systems or Python versions.
* "xfail" will report an expected failure if the test case fails. This helps avoid report pollution for known problems.
* "parameterize" we've already encountered in a previous chapter.

### [#](https://testautomationu.applitools.com/pytest-tutorial/chapter8.html#testpaths)testpaths

There's one more thing I'd like to cover regarding test case filtering.

By default, pytest will search for test cases either from the current directory or from the paths and options given at the command line.

However, setting "testpaths" in the configuration file will explicitly set test case search paths.

For example, we can set "testpaths = tests" in our pytest.ini file to make sure that pytest searches only the "test" folder.

Setting paths is a great way to enforce structure and also speed up discovery time for large projects.

As you can see, pytest provides great support for filtering and marking test cases. Use it to your advantage!

### configuration Files

Now, let's talk about configuration. Command line options are great for running tests in specific ways on the fly.

However, there might be some options you want to set more permanently, especially when running tests in a Continuous Integration environment.

Some options are not available directly through the command line either.

Thankfully, you can create a configuration file for pytest to specify options in a file rather than on the command line.

Pytest supports a few different file formats for configuration files.

The most common one I've seen is "pytest.ini".