Testing Lab

Kaia Lindberg (pkx2ec)

In this Lab you will be providede a module that has 3 layers, combining 3 different behaviours into one file.

The purpose of the lab is to explore a 3 layer script (module/tests/execute), by:

- using the module as an import to execute a function
- execute the script directly as an executable to view a tiny sqlite3 database
- write proper documentation for the tests
- refactor the tests to use a pytest fixture
- set up a makefile to create short cuts for all three uses of the file
- there are sereval possiblities for extra credit for a little extra challenge

Overall the main point is to make sure

1. The file will work as a module that can be imported into a project
2. The file will have tests that only activate when run with pytest and verify the code in the file
3. The file will run when called directly
4. Along the way, we'll also get a simple intro and use to makefiles.

Setup for this assignment

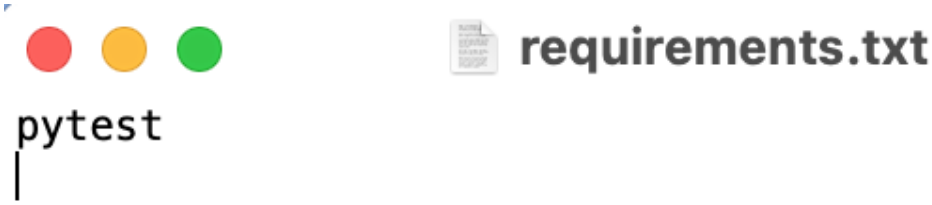
- **1 pt**: Create a new github public repo (you'll send in the link) and call it '3-layer-single-script'

Repo link: https://github.com/kaiarl/3-layer-single-script

- Clone the repo to your local machine

```
(base) Kaias-MacBook-Pro:04_sw_testing Kaia$ git clone https://github.com/kaiarl/3-layer-single-script.git
Cloning into '3-layer-single-script'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

- **1 pt**: Create a requirements.txt file with 'pytest' in it



- Copy the file provided HERE "db_viewer.py" into the repo (so at this point, there should be only two files, except maybe a readme if you added one)

db_viewer.py - /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engineering/...

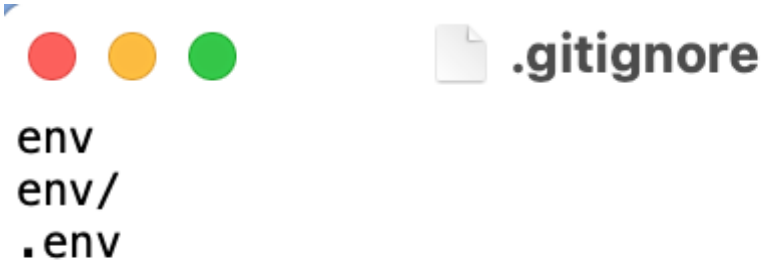
```
import sqlite3
import os


class Singleton:
    count = 0
    cursor = None
    db_name = 'aquarium.db'
    def __new__(cls):
        if not hasattr(cls, 'instance'):
            cls.instance = super(Singleton, cls).__new__(cls)
            cls.instance.get_cursor()
        return cls.instance

    def __init__(self):
        self.count += 1

    def get_cursor(self):
        if os.path.exists("aquarium.db"):
            print("DB found, getting cursor")
            self.connection = sqlite3.connect("aquarium.db")
            self.cursor = self.connection.cursor()
        else:
            print("DB NOT found!  run initialize_database first")
            self.cursor = None
```

- **1 pt**: Create a `.gitignore` file in your repo and add `env` in the file (this will keep our environment from getting checked into github.



- After the setup above, you should run `python3 -m venv env` to create the virtual environment, activate it and then run `pip install -r requiremts.txt` to install

pytest.

```
(base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ python3 -m venv env
(base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ source env/bin/activate
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ pip install -r requirements.txt
Collecting pytest
  Downloading pytest-7.2.1-py3-none-any.whl (317 kB)
     |████████████████████████████████| 317 kB 3.3 MB/s
Collecting pluggy<2.0,>=0.12
  Downloading pluggy-1.0.0-py2.py3-none-any.whl (13 kB)
Collecting attrs>=19.2.0
  Downloading attrs-22.2.0-py3-none-any.whl (60 kB)
     |████████████████████████████████| 60 kB 16.9 MB/s
Collecting packaging
  Downloading packaging-23.0-py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 3.7 MB/s
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting exceptiongroup>=1.0.0rc8
  Downloading exceptiongroup-1.1.0-py3-none-any.whl (14 kB)
Collecting tomli>=1.0.0
  Downloading tomli-2.0.1-py3-none-any.whl (12 kB)
Installing collected packages: tomli, pluggy, packaging, iniconfig, exceptiongroup, attrs, pytest
Successfully installed attrs-22.2.0 exceptiongroup-1.1.0 iniconfig-2.0.0 packaging-23.0 pluggy-1.0.0 pytest-7.2
.1 tomli-2.0.1
WARNING: You are using pip version 21.2.4; however, version 23.0.1 is available.
You should consider upgrading via the '/Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/0
4_sw_testing/3-layer-single-script/env/bin/python3 -m pip install --upgrade pip' command.
```

Checkpoint:

- You should be able to run `python db_viewer.py` and see `DB NOT found!  run intialize_datase first` , this is normal. Just type `quit` and you should be back to the command line.

```
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ python db_viewer.py
DB NOT found!  run initialize_database first
=> quit
```

- **1 pt** At this point you should also be able to run `pytest -vvx db_viewer.py` and see green come through the console for the passing tests.

```
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ pytest -vvx db_viewer.py
============================================= test session starts ==============================================
platform darwin -- Python 3.10.1, pytest-7.2.1, pluggy-1.0.0 -- /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engin
eering/data_engineering/04_sw_testing/3-layer-single-script/env/bin/python3
cachedir: .pytest_cache
rootdir: /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/04_sw_testing/3-layer-single-sc
ript
collected 4 items

db_viewer.py::test_is_singleton PASSED                                                                  [ 25%]
db_viewer.py::test_not_initialized PASSED                                                               [ 50%]
db_viewer.py::test_database_connect PASSED                                                              [ 75%]
db_viewer.py::test_resetting_after_db_creation PASSED                                                   [100%]

============================================== 4 passed in 0.03s ===============================================
```

- If the last two steps worked, then youre ready for hte next steps

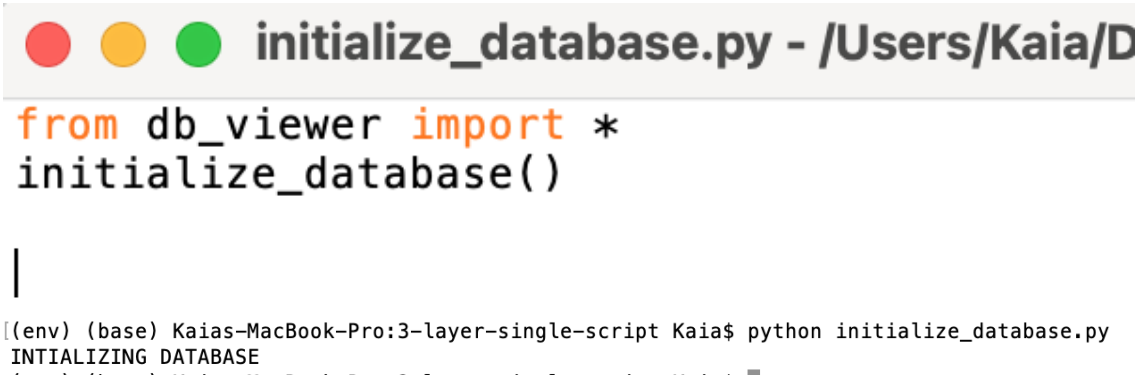
NB: If you ran the steps backwards, that is to say, ran the tests first before executing the script you'll notice the database message above doesn't show up. That is because the tests create the database, this is actually a bug we'd like to fix, but that is later in this lab. At this point, do an `ls` and if you see `aquarium.db` go ahead and delete/remove it before going onto the next step.

Using the script as a module

We just saw that the script acts as a test module since we can run pytest with it and run the tests. The message you got earlier, about the database not existing is because we have not

created one. Let's create another script that uses our main script as a module to initiaze the database.

- **1 pt** - Create a small script `initialize_database.py` , and it should literally contain 2 lines
  - The first line should import the file db_viewer
  - the second line should call the modules `initialize_database` function, and that should do it.



- **1 pt** - Verify that `python initialize_database.py` generates the `aquarium.py` file mentioned before. If it does, now try running the main script `python db_viewer.py` and you should see a new message, `DB found, getting cursor` . If you now type `select * from fish;` at the cursor and enter, you should see two lines show up with names and two types of fish.

```
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ python db_viewer.py
DB found, getting cursor
=> select * from fish
Executing: select * from fish
('Sammy', 'shark', 1)
('Jamie', 'cuttlefish', 7)
=>
```

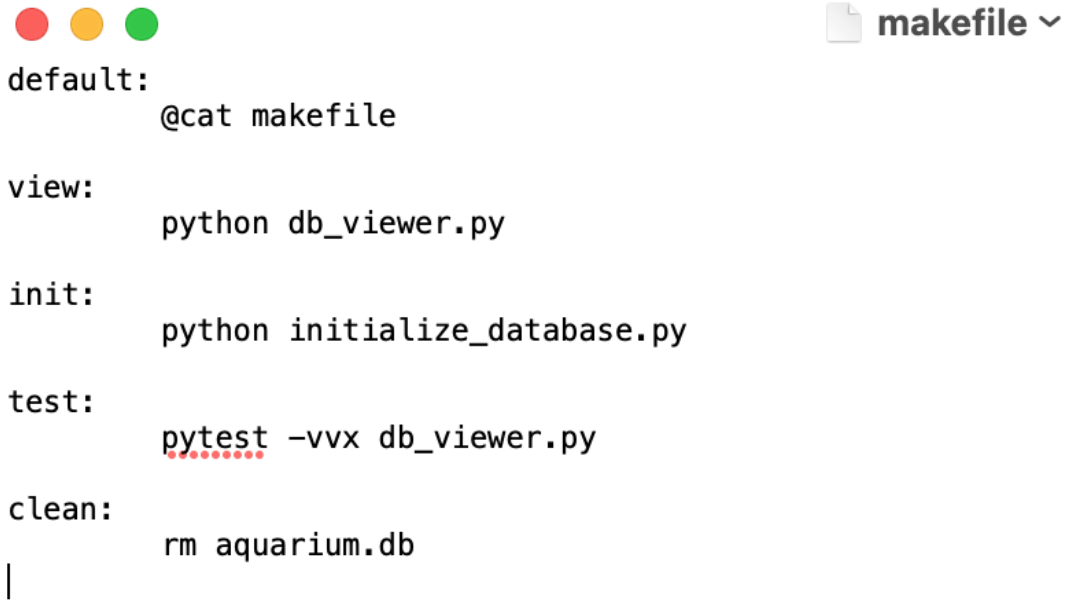
If this section worked, good, you've setup the script as a module and we're ready for a quick tangent on makefiles

Side trip to makefiles

Makefiles are some of the original automation provided by linux. It was intended for assisting in compiling tedouls multi file interactions for large programs written in c and c++. It is a bit archaic and can have some loopy syntax. However, if you keep things simple this functionality can serve as a shortcut and as documentation to how to use your script.

Setting up the Makefile

- **1 pt:** Add a file named `makefile` to your project.



- I'll give you the contents here and your task for this section will be to verify these work. These will be run when we test your repo contents in the end.

```
default:
    @cat makefile

view:
    python db_viewer.py

init:
    python initialize_database.py

test:
    pytest -vvx db_viewer.py

clean:
    rm aquarium.py
```

Now let's test it out

- type `make` and you should see the contents of the file on the console. The command make executes a section of the makefile, which by default it will be the first instruction. I got into the habit of always addint the `default:` at the top so I would not accientaly trigger anything, and this way it just shows me the contents and thus the commands available. By the way, ...
  - **IMPORTANT** words starting on the left with the `:` are the commands. No spaces on the left of those words like `default`, `view`, `init`, and `clean`. **AND** DOUBLY important, the space in the executing lines, that first space are before `@cat makefile` for example HAS TO BE A TAB character. Those are some idiosicrasies of the the makefile.

- The `@` symbol in `@cat makefile` means execute `cat makefile` but do not echo the command to the console, otherwise you'd see `cat makefile` every time you execute. Just keeps things neater on the command output.

Note: I edited the "clean" line to be "remove auqarium.db" instead of ".py".

```
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ make
default:
	@cat makefile

view:
	python db_viewer.py

init:
	python initialize_database.py

test:
	pytest -vvx db_viewer.py

clean:
	rm aquarium.db
```

- Now let's try `make view`, `make init`, `make test`, and `make clean`. You can probably tell from the simple commands what they do.

```
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ make init
python initialize_database.py
INTIALIZING DATABASE
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ make view
python db_viewer.py
DB found, getting cursor
=> select * from fish
Executing: select * from fish
('Sammy', 'shark', 1)
('Jamie', 'cuttlefish', 7)
=> quit
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ make test
pytest -vvx db_viewer.py
============================================= test session starts ==============================================
platform darwin -- Python 3.10.1, pytest-7.2.1, pluggy-1.0.0 -- /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engin
eering/data_engineering/04_sw_testing/3-layer-single-script/env/bin/python3
cachedir: .pytest_cache
rootdir: /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/04_sw_testing/3-layer-single-sc
ript
collected 4 items

db_viewer.py::test_is_singleton PASSED                                                                  [ 25%]
db_viewer.py::test_not_initialized PASSED                                                               [ 50%]
db_viewer.py::test_database_connect PASSED                                                              [ 75%]
db_viewer.py::test_resetting_after_db_creation PASSED                                                   [100%]

============================================== 4 passed in 0.02s ===============================================
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ make clean
rm aquarium.db
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$
```

Why of makefiles

- You now have shortcuts to your most common commands you'll be using
- Another dev/Data-Scientist now using your repo has a view of how to use it, in real life there may be all sorts of flags `-x.... -file ....` etc, so you may even have a few ways of running it. In this way the file serves as documentation.

Also

- `clean` is very typical as you can use it "reset" to some initial conditions etc
- `test` is very common in makefiles
- You can 'chain' the commands. If you wanted to load the database fresh every time you may do something like this

```
clean_view: clean init view
```

- **1 pt** And that's it, a `make clean_view` should run the `clean`, `init` and then `view`

```
clean_view: clean init view
```

for you.

```
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ make clean_view
rm aquarium.db
python initialize_database.py
INTIALIZING DATABASE
python db_viewer.py
DB found, getting cursor
=> select * from fish
Executing: select * from fish
('Sammy', 'shark', 1)
('Jamie', 'cuttlefish', 7)
=> quit
```

Using a pytest fixture

- **2 pt** Use a pytest fixture to provide the data used to create the test database (The `CREATE TABLE... INSERT INTO... INSERT INTO...` lines)

Fixture that creates database with test data:

```
# Pytest fixture to provide data used to create test database
@pytest.fixture
def setup_database():
    delete_database()
    connection = sqlite3.connect('aquarium.db')
    cursor = connection.cursor()
    cursor.execute("CREATE TABLE fish (name TEXT, species TEXT, tank_number INTEGER)")
    cursor.execute("INSERT INTO fish VALUES ('Sammy', 'shark', 1)")
    cursor.execute("INSERT INTO fish VALUES ('Jamie', 'cuttlefish', 7)")
    connection.commit()
    yield connection
```

Test that uses fixture:

```
def test_resetting_after_db_creation(setup_database):
    delete_database()

    db_a = Singleton()
    db_b = Singleton()
    assert id(db_a) == id(db_b)
    db_a.get_cursor()
    assert [] == db_a.sql("SELECT * FROM FISH;")
    assert [] == db_b.sql("SELECT * FROM FISH;")

    cursor = setup_database
    assert 2 == len(list(cursor.execute("SELECT * FROM fish;")))
```

Test execution:

```
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ make test
pytest -vvx db_viewer.py
============================= test session starts ==============================
platform darwin -- Python 3.10.1, pytest-7.2.1, pluggy-1.0.0 -- /Users/Kaia/Desk
top/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/04_sw_testing/3-layer-sin
gle-script/env/bin/python3
cachedir: .pytest_cache
rootdir: /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/
04_sw_testing/3-layer-single-script
collected 4 items

db_viewer.py::test_is_singleton PASSED                                   [ 25%]
db_viewer.py::test_not_initialized PASSED                                [ 50%]
db_viewer.py::test_database_connect PASSED                               [ 75%]
db_viewer.py::test_resetting_after_db_creation PASSED                    [100%]

============================== 4 passed in 0.03s ===============================
```

At this point we've got 10 pts for the lab possible. And if everything works good, we'll be running the make file commands, and looking at the Given/When/Then, plus the implementation of a fixture. The rest of this lab is for extra credit, or to possibly cover any points missed above.

In [ ]:

In [ ]:

Going further

- **1 pt** Set up your makefile so the tests always run when you use `make view`

Note: I also added init to make sure that the database exists because otherwise there is nothing to view (e.g. so I could run the "select * from fish").

```
view: test init
        python db_viewer.py
```

```
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ make view
pytest -vvx db_viewer.py
============================= test session starts ==============================
platform darwin -- Python 3.10.1, pytest-7.2.1, pluggy-1.0.0 -- /Users/Kaia/Desk
top/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/04_sw_testing/3-layer-sin
gle-script/env/bin/python3
cachedir: .pytest_cache
rootdir: /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/
04_sw_testing/3-layer-single-script
collected 4 items

db_viewer.py::test_is_singleton PASSED                                   [ 25%]
db_viewer.py::test_not_initialized PASSED                                [ 50%]
db_viewer.py::test_database_connect PASSED                               [ 75%]
db_viewer.py::test_resetting_after_db_creation PASSED                    [100%]

============================== 4 passed in 0.02s ===============================
python initialize_database.py
INTIALIZING DATABASE
python db_viewer.py
DB found, getting cursor
=> select * from fish
Executing: select * from fish
('Sammy', 'shark', 1)
('Jamie', 'cuttlefish', 7)
=> quit
```

- **2 pt** use a pytest decorator so only one tests for the db runs, say you consider this a smoke test. What you're simulating here is running all the fast, or most important tests quickly. Add a makefile entry so we can run `make test_smoke` and `make test`. `test_smoke` should run a subset only.

```
@pytest.mark.smoketest
def test_resetting_after_db_creation(setup_database):
    delete_database()

    db_a = Singleton()
    db_b = Singleton()
    assert id(db_a) == id(db_b)
    db_a.get_cursor()
    assert [] == db_a.sql("SELECT * FROM FISH;")
    assert [] == db_b.sql("SELECT * FROM FISH;")

    cursor = setup_database
    assert 2 == len(list(cursor.execute("SELECT * FROM fish;")))
```

```
test_smoke:
	pytest -vvx -m smoketest db_viewer.py
```

```
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$ make test_smoke
pytest -vvx -m smoketest db_viewer.py
============================= test session starts ==============================
platform darwin -- Python 3.10.1, pytest-7.2.1, pluggy-1.0.0 -- /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/04_sw_testing/3-layer-single-script/env/bin/python3
cachedir: .pytest_cache
rootdir: /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/04_sw_testing/3-layer-single-script
collected 4 items / 3 deselected / 1 selected

db_viewer.py::test_resetting_after_db_creation PASSED                    [100%]

=============================== warnings summary ===============================
db_viewer.py:101
  /Users/Kaia/Desktop/UVA_MSDS/DS_5559_Data_Engineering/data_engineering/04_sw_testing/3-layer-single-script/db_viewer.py:101: PytestUnknownMarkWarning: Unknown pytest.mark.smoketest - is this a typo?  You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-to/mark.html
    @pytest.mark.smoketest

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
================== 1 passed, 3 deselected, 1 warning in 0.02s ==================
(env) (base) Kaias-MacBook-Pro:3-layer-single-script Kaia$
```

- **3 pt** As mentioned earlier, there is a bug here. If we run the tests, they leave the test intantiation of `aquarium.db` behind. Fix the bug and make sure all your tests still pass.

Added delete_database() in fixture so the aquarium.db is deleted when the test that uses the fixture is complete (teardown code runs after yield)

```
# Pytest fixture to provide data used to create test database
@pytest.fixture
def setup_database():
    delete_database()
    connection = sqlite3.connect('aquarium.db')
    cursor = connection.cursor()
    cursor.execute("CREATE TABLE fish (name TEXT, species TEXT, tank_number INTEGER)")
    cursor.execute("INSERT INTO fish VALUES ('Sammy', 'shark', 1)")
    cursor.execute("INSERT INTO fish VALUES ('Jamie', 'cuttlefish', 7)")
    connection.commit()
    yield connection
    delete_database()
```

All of my tests still pass, but the aquarium.db is not still there after testing.

- **3 pt** Modify the script so you pass it the name of a pre-existing database, while keeping all the tests running with our test database `aquarium.db` . After all the utlity of the file is looking at any sqlite3 db, not just that one.
- **4 pt** Modify the file further and

Part 3 Adding an `if __name__=="__main__":` section to script

This will allow for both using the script standalone to inspect a sqlite database, but also can be used to explore or do some exploratory testing.

```
if __name__=="__main__":


    db = Singleton()

    while True:
        stmt = input("=> ")
        if stmt == 'quit':
            break

        rows = db.sql(stmt)
        for row in rows:
            print(row)
```

In [ ]: