Report

Assignment 3

Done By: Victor Olofsson

Student Email: victor.olofsson0034@stud.hkr.se

Class: DA115B V22

Date: 2022-03-20

Table Of contents

Introduction

In this report we will present, analyze and discuss the fundamental principles of functional programming and what problems they solve, more specifically I will present my findings and results from working on the Assignment 2 Test Driven Development and what purpose functional programming had when working on the project. Which included Test driven development (TDD), Unittesting, Software documentation, clean code, development environments and sustainable software development. While there are many great ways to develop software sufficiently, keeping the development process organized and optimized by implementing functional programming principles into your work environment, is the best way to work efficiently. It helps you and others for present and future purposes when handling the project to better understand it, share it with other developers and to continuously write good and clean code throughout the whole process of development from start to a finished product. While it is easy to jump straight in, there are core fundamental concepts that need to be understood before we can start development. This report aims to help further grasp these concepts and the understanding of how we can achieve them while also learning how to retrieve necessary data from our own project and what results we receive from that data and how we can interpret it using the functional programming philosophies.

Method

The study for Assignment 2 - Test Driven Development consisted of three participants, Me, Victor Olofsson, Walid Alzohili and Ahmad El-Jachi.
During the project which was conducted both virtually and in person we found that while using the principles of functional programming we could improve the overall quality and scalability of the project while also keeping the development environment clean. This increased our overall development process speed and made us able to develop the project sufficiently with clean code. It was done using python, pycharm IDE and github.

Test driven development (TDD)

Using TDD refers to the system development method which focuses on automated unit testing for each code block. It advocates that no code shall be written or changed before a test is initially written. *(Hamilton, 2022)*

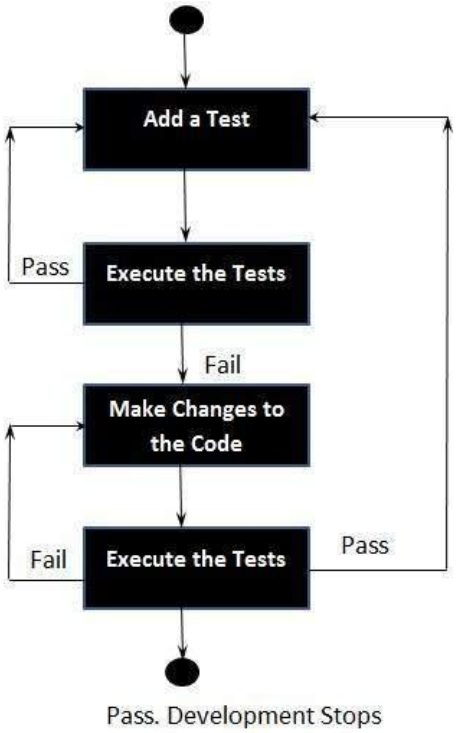


Figure 1 - Test driven development process

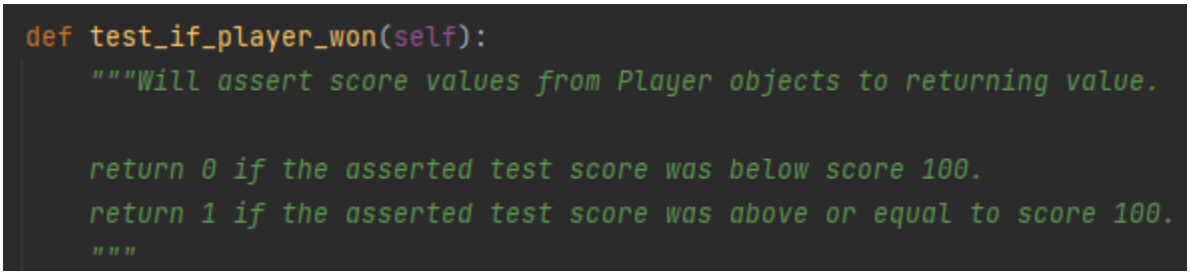
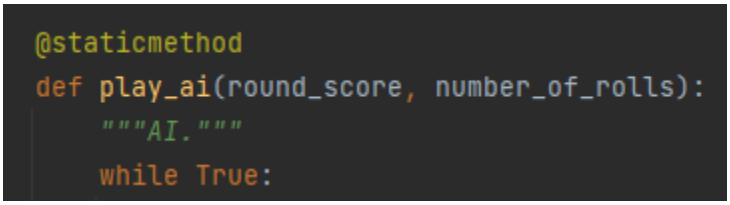
During assignment 2 it was important to add extensive tests before writing code to assess and perform code changes to our code base. Some observations done during the project concluded that when working in a team, keeping track and differentiating between good and poor changes in the code base would prove difficult unless a test was initially written for every specific unit of source code and it had to be extensively tested before merging it into the main code base. We wrote unit tests to check for return

values of methods and functions making sure that each method and function returned the expected value. From this we can collect statistics such as a code coverage rate to further enhance our understanding of our codebase.

Software Documentation

One of the most important parts of creating good and clean code when developing the project was to create sufficient documentation in the form of docstring to provide an overview for the project when generating python documentation and for other developers to understand what a method achieved or tried to achieve.

Table 1 - Screenshots of our code base from the project

Good

Bad


```
def test_if_player_won(self):
    """Will assert score values from Player objects to returning value.

    return 0 if the asserted test score was below score 100.
    return 1 if the asserted test score was above or equal to score 100.
    """
```

```
@staticmethod
def play_ai(round_score, number_of_rolls):
    """AI."""
    while True:
```

The standard method of writing docstring is “Do this, return that”. We show here in Table 1 our code base where we wrote 2 different docstrings, a good one and a bad one. The good one is more descriptive and tells us what to expect from the corresponding method. While the bad one, even though it gives us information, is unstructured and it is difficult to understand what the corresponding method will specifically contain and return. Documentation also includes markdown language files which should be included in your project that involve a license, a readme and a release file for the project. We used a license description which referred to how the project is allowed to be distributed, if it was free to use and who it was owned by. We wrote a readme that documented how a user could download the project, where they should start when opening it and how to

use the application overall, and we wrote a description of the applications intended use containing rules and functionality. In the release file we included all necessary version updates that the development team had implemented throughout the project's lifetime giving short descriptions for each version update.

Development Environments

We created a virtual development environment for our project, implementing a virtualenv, or venv, which is a module used to isolate dependencies, it allows us to divide package installations for different projects. (*Installing Packages Using Pip and Virtual Environments — Python Packaging User Guide*, n.d.)

It was easy to use a virtual environment when working with projects, for its ability to work with different versions of libraries, packages and installations. This could be because you need to use a specific version that's free from a specific bug or perhaps the initial program was written in an outdated version of a library. This can be handled by using a virtual environment, because instead of using only one version you can now use the suitable version for your specific needs. *(Reitz, n.d.)*

Keeping your code base consistent with specific versions of libraries isolated in your current development environment instead of downloading and sharing many different versions globally used between different projects will deliver increased good and clean code.

Sustainable Software Development

While working on the project we implemented sustainable software development, which refers to, in my analyzed opinion, a concept when developing software. It includes code sustainability, testing, documenting and maintaining an equal level of quality of code throughout the whole development process. It is meant to optimize the scalability of the project as a whole and ease the current and future development for already existing and new developers.

Unittesting

The main purpose of TTD is to test our code, using unit testing, it allowed us to test our units of source code. It is easy to use and we used it to understand and examine the output of our modules to see if it was expected or not. Unittesting relates to good and clean code because it aids us by not only testing for expected outputs but also for checking the applications performance for statistical use. It gives us more insight into how “good” our code actually is because we could see if our code was good or not

depending on how easy it was to test. When we developed our project we needed to write our own unit tests to adhere to the very concept of TDD and functional programming. *(Python Unit Testing, n.d.)*

Clean Code

Clean code is
“code that is easy to understand and easy to change.” (Woodfine, 2018)

We needed to ask ourselves when we developed our project, what is easy to understand and easy to change code? What criteria do you have to fulfill for your code to be called easy?

We would ask ourselves when developing:

- Was it easy to understand the flow of the application?
- Was it easy to understand how classes and objects talked to each other?
- Was it easy to understand what each method did?
- Was it easy to understand the purpose of our methods?

And when we needed to change our code we would ask:

- Is our code concise and divided into as small parts as possible?
- Are our classes and functions easy to understand and do they work as expected?
- Is our code easily testable?
- Are the tests easy to change and expand on?

While having these bullet points in mind when developing the software it was easy to understand if what we had written could be considered good and clean code and it would increase the quality of our application.

Results

I found that when developing the project for Assignment 2 Test Driven Development, using the concept of functional programming I could collect a lot of data about my own software using tools such as:

Coverage

Using the code coverage tool I could see how many methods and how many modules were accounted for when running my unit tests. As seen in table 2, I would get a coverage report, in both a documented html file but also directly into the console. With a 91% coverage rate. In the html version i can inspect any selected module and see which line/lines was missing in the coverage report as seen in Table 3.

Table 2 - Coverage report

Console Report

```
Name                    Stmts   Miss  Cover   Missing
-----------------------------------------------------
get_winner.py              36      1    97%   44
get_winner_test.py         23      1    96%   55
menu.py                    62     22    65%   28-31,
menu_test.py               57      1    98%   87
player_class.py            22      0   100%
player_class_test.py       27      1    96%   46
roll_dice.py               58     10    83%   47-52,
roll_dice_test.py          40      1    98%   81
statistics.py              45      0   100%
statistics_test.py         39      1    97%   67
-----------------------------------------------------
TOTAL                     409     38    91%
```

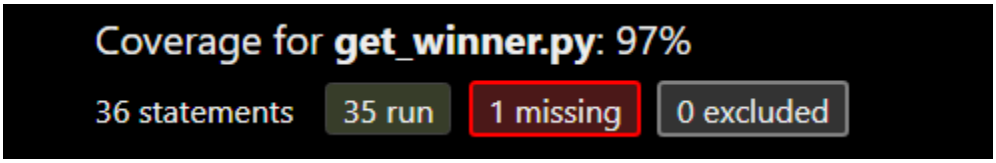
HTML report

Coverage report: 91%

Module	statements	missing	excluded	coverage
get_winner.py	36	1	0	97%
get_winner_test.py	23	1	0	96%
menu.py	62	22	0	65%
menu_test.py	57	1	0	98%
player_class.py	22	0	0	100%
player_class_test.py	27	1	0	96%
roll_dice.py	58	10	0	83%
roll_dice_test.py	40	1	0	98%
statistics.py	45	0	0	100%
statistics_test.py	39	1	0	97%
Total	**409**	**38**	**0**	**91%**

coverage.py v6.3.1, created at 2022-03-20 23:55 +0100

Table 3 - Coverage Report HTML insight



Pdoc/Pyreverse

To generate the python html documentation and to enable a view of the classes in the project in the form of UML diagrams i would use the pdoc and pyreverse commands.

Table 4 - Python documentation

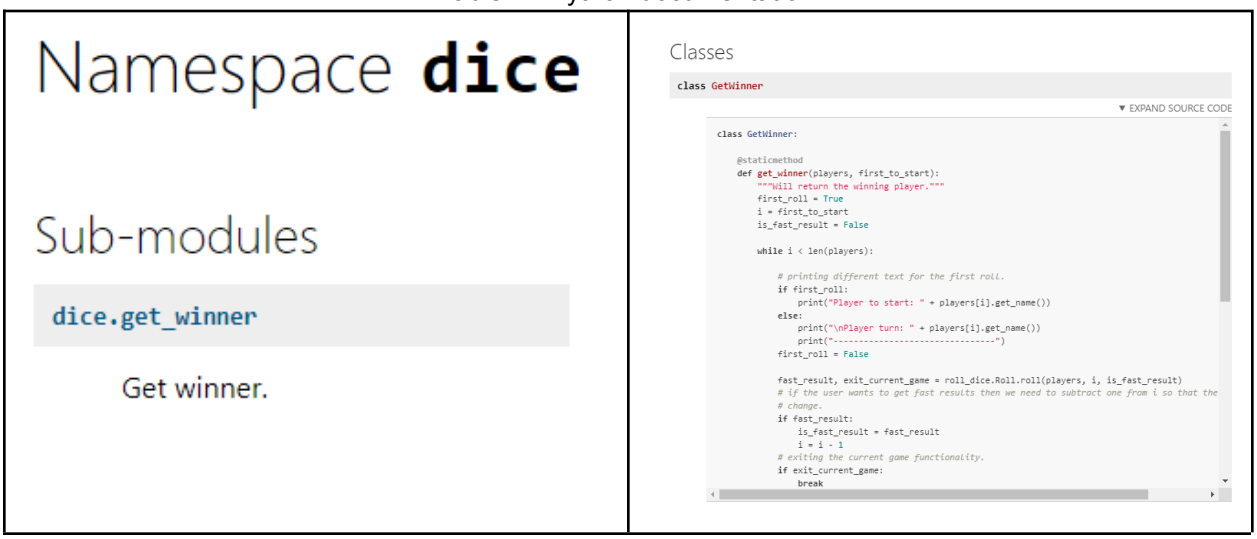
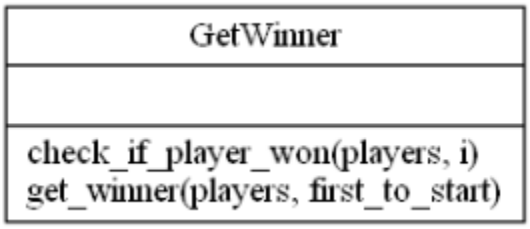


Table 5 - UML Diagram example



PyLint/Flake8

To see if the code base follows the standard coding style in python PEP8, we use flake8 and to improve the code and check for any errors we use pylint. This would give us information about where and what was wrong with our code and it would also rate our code on a scale between 0-10 as seen in Figure 2.

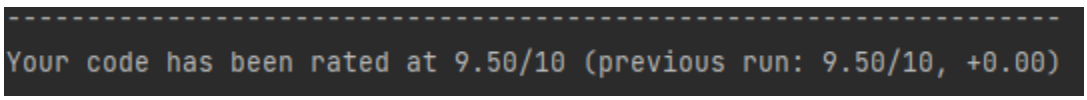


Figure 2 - Pyling and Flake 8 code evaluation

Metrics

To understand if your code was complex to understand we would use a Python tool named Radon that checks for code complexity using mathematical equations to measure it's complexity. As shown in Figure 3.

```
get_winner_test.py:
    h1: 1
    h2: 2
    N1: 1
    N2: 2
    vocabulary: 3
    length: 3
    calculated_length: 2.0
    volume: 4.754887502163469
    difficulty: 0.5
    effort: 2.3774437510817346
    time: 0.1320802083934297
    bugs: 0.0015849625007211565
```

Figure 3 - Radon metrics

Bandit

To find common security issues in the project we would use a tool called bandit. It allows us to see different metrics regarding security in our code, as can be seen in Figure 4.

```
Code scanned:
        Total lines of code: 549
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0.0
                Low: 3.0
                Medium: 0.0
                High: 0.0
        Total issues (by confidence):
                Undefined: 0.0
                Low: 0.0
                Medium: 0.0
                High: 3.0
Files skipped (0):
```

Figure 4 - Bandit results

Discussion

Let’s look at table 2 and table 3, it shows us extensive information regarding our coverage rate for the independent modules.

What data did we collect from this?
By analyzing the data we can open the source code from the documentation and see which parts are run, missing and excluded in the coverage process. This data allows insight to how we can simplify and modify the code to get a greater cover rate from our unit tests.

How does the retrieved data relate to using functional programming?
Increasing our overview and insight of the project also increases the level of good and clean code. It allows us to maintain the quality of our code through our project from start to finished product.

Table 4 and Table 5 show documentations and UML diagrams of our code base. The specific part we have taken from the project, is a module named get_winner which we use as an example to show what benefits writing doc strings has.

Is writing python documentation and UML diagrams necessary?
While it might not seem necessary from the start, it is good practise to document the code as extensively as possible when the project is nearing a finished product, this includes creating an organized document that explains each functions with docstrings and shows it’s corresponding code snippet in a functional manner, Pdoc allows this and is the reason why we also used it. Pyreverse allows us to create organized UML diagrams of our classes which is a nice way for developers to get an overview and insight of each class's inclusion of attributes and functions.

Does it comply with the concept of functional programming?
We treat documentation as a key aspect of functional programming due to its ability to explain what our code base and project as a whole is about. When entering a new project with new eyes the first thing we look for is documentation, and that explains why it is necessary to have valuable and organized documentation in different forms that explains the project.

Figure 2, Figure 3 and Figure 4 shows data collected when running static code linters, collecting software metrics and searching for security flaws within your own project while development is ongoing.

Why is it important to collect data during development?
It is a crucial part of development to know how well your current style of coding is, if it is aligned with the python code standard (PEP 8), if your code is complicated to understand and write, if a class has low cohesion or if imported statements are unused. Also finding out about security flaws early in development will prevent you and others working on the project from continuing to use flawed security.

What will this data give us in return?
In return this increases our overall productivity, insight and quality of code. It gives us a different response to our unit tests than the normal output which is an expected returned value. It forces us to adhere to a good and clean code standard throughout the whole project, our code base becomes more sustainable and it makes us able to document our code along the way and it gives us reasoning if we need to reformat a part of the code, for example any found security flaws or development issues that comes to surface using the data might make us need to reformat the code and thus keeps the cycle going of maintaining good and clean code through out the project.

Summary

In conclusion, using functional programming and adhering to the mentioned above report, we can develop software cleaner and in a more structured way. Overall the Assignment 2 - Test Driven Development enabled us to understand the core fundamental concepts of functional programming and applying it taught us to collect data, write extensive documentation and maintain good and clean code throughout the project following these concepts. The most important finding was the ability to test code before it was written, using Test Driven Development the unit tests guided us into what we *should* write rather than what we *can* write to develop our software functionally.

References

Hamilton, T. (2022, February 26). *What is Test Driven Development (TDD)? Tutorial with Example*. Guru99. Retrieved March 21, 2022, from https://www.guru99.com/test-driven-development.html

Installing packages using pip and virtual environments — Python Packaging User Guide. (n.d.). Python Packaging User Guide. Retrieved March 21, 2022, from https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/

Python Unit Testing. (n.d.). Javatpoint. Retrieved March 21, 2022, from https://www.javatpoint.com/python-unit-testing

Reitz, K. (n.d.). *12. Virtual Environments and Packages — Python 3.10.3 documentation*. Python Docs. Retrieved March 21, 2022, from https://docs.python.org/3/tutorial/venv.html

Woodfine, G. (2018, October 8). *What is Clean Code ?* Gary Woodfine. Retrieved March 21, 2022, from https://garywoodfine.com/what-is-clean-code/