TOP-UP BACHELOR OF SCIENCE (HONOURS) INFORMATICS

PORTFOLIO REPORT

COURSE NAME : TOP-UP BACHELOR OF SCIENCE (HONOURS) INFORMATICS

MODULE CODE: KOL389COM

MODULE NAME: OPEN SOURCE DEVELOPMENT

MODULE LEADER: NIELS MüLLER LARSEN

PREPARED BY: BIPLAB TWATI

WORD COUNT: 2232

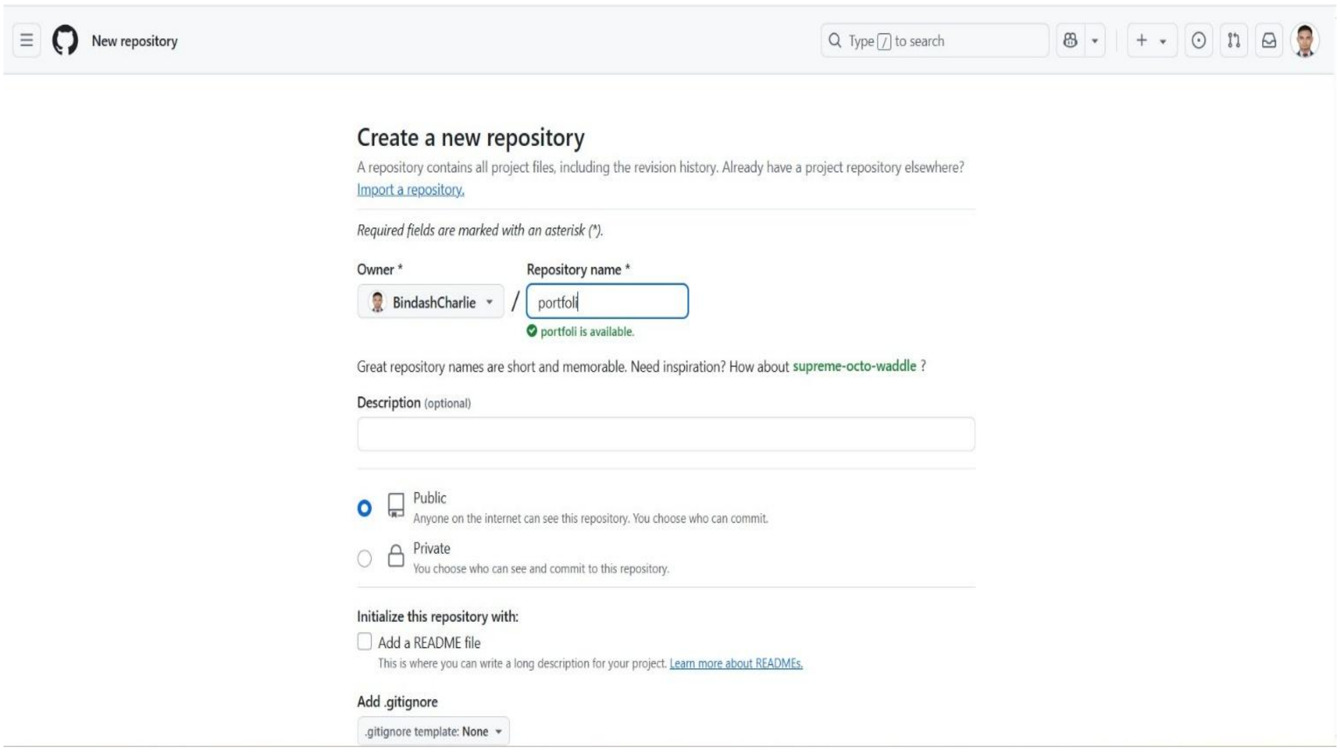
SUBMISSION DATE: 16/04/2025

Github link : https://github.com/BindashCharlie/portfolio

Table of Contents

Git and GitHub Workflow

Create a Repository



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner *: BindashCharlie / Repository name *: portfoli

portfoli is available.

Great repository names are short and memorable. Need inspiration? How about supreme-octo-waddle ?

Description (optional)

(●) Public — Anyone on the internet can see this repository. You choose who can commit.

( ) Private — You choose who can see and commit to this repository.

Initialize this repository with:

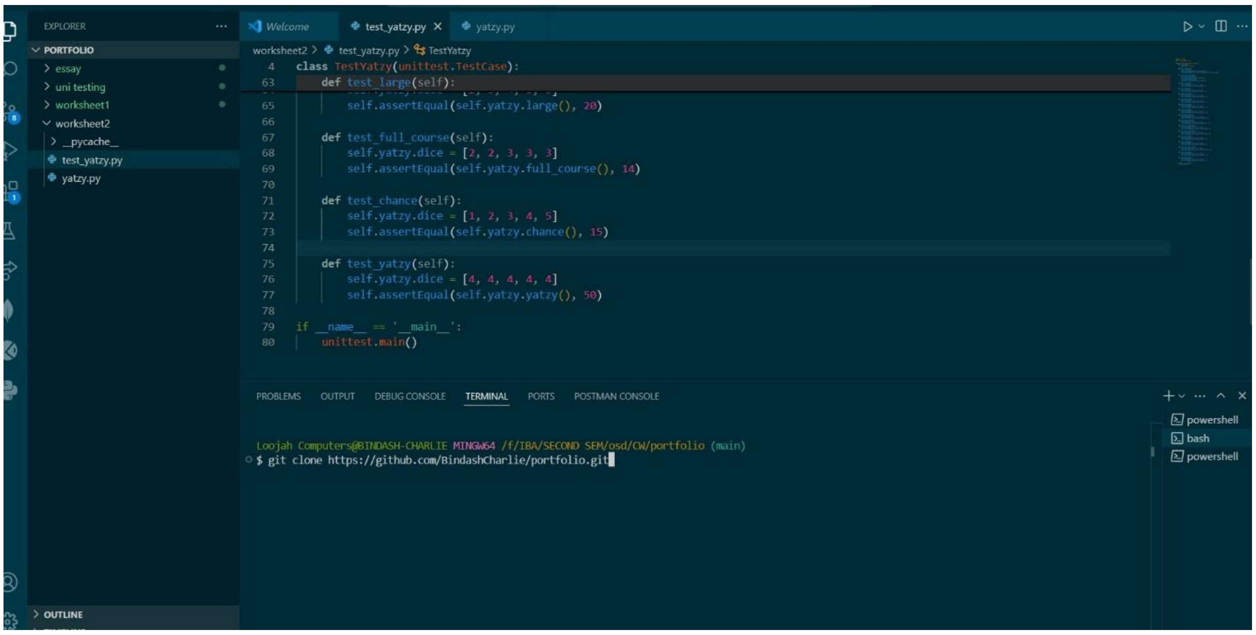
☐ Add a README file — This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None

Clone the Repository

Git Clone is the command that we employ to copy a remote repository (for example, one on GitHub or GitLab) onto our local system. It lets us copy the whole project along with its history so that we can work on it locally. When we run the command git clone, it downloads the project files from the remote repository and creates a new local repository on our machine. In this manner, we have a complete copy of the project now to manage, and we can push changes, make them, and push them again to the remote repository.



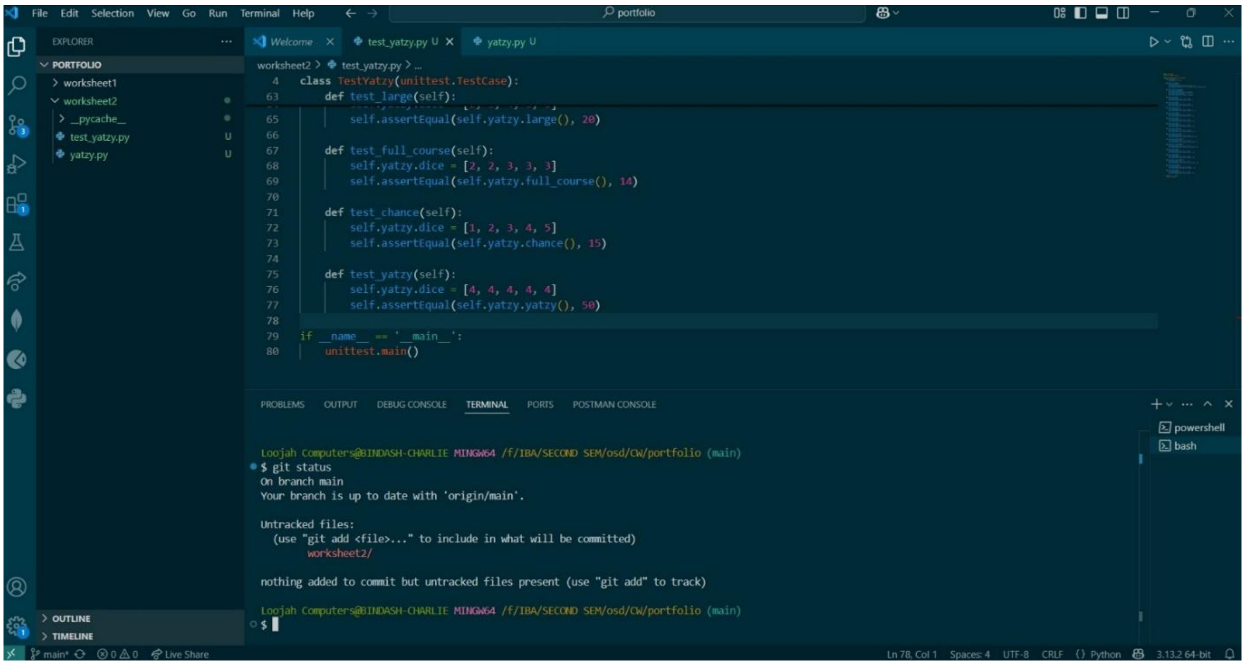
```python
class TestYatzy(unittest.TestCase):
    def test_large(self):
        self.assertEqual(self.yatzy.large(), 20)

    def test_full_course(self):
        self.yatzy.dice = [2, 2, 3, 3, 3]
        self.assertEqual(self.yatzy.full_course(), 14)

    def test_chance(self):
        self.yatzy.dice = [1, 2, 3, 4, 5]
        self.assertEqual(self.yatzy.chance(), 15)

    def test_yatzy(self):
        self.yatzy.dice = [4, 4, 4, 4, 4]
        self.assertEqual(self.yatzy.yatzy(), 50)

if __name__ == '__main__':
    unittest.main()
```

```
Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$ git clone https://github.com/BindashCharlie/portfolio.git
```

Check Status

git status

The git status command checks the current state of your Git working directory and staging area. It shows which files have been modified, added, or deleted since your last commit, and whether they're staged for the next commit or untracked (not yet added to Git).



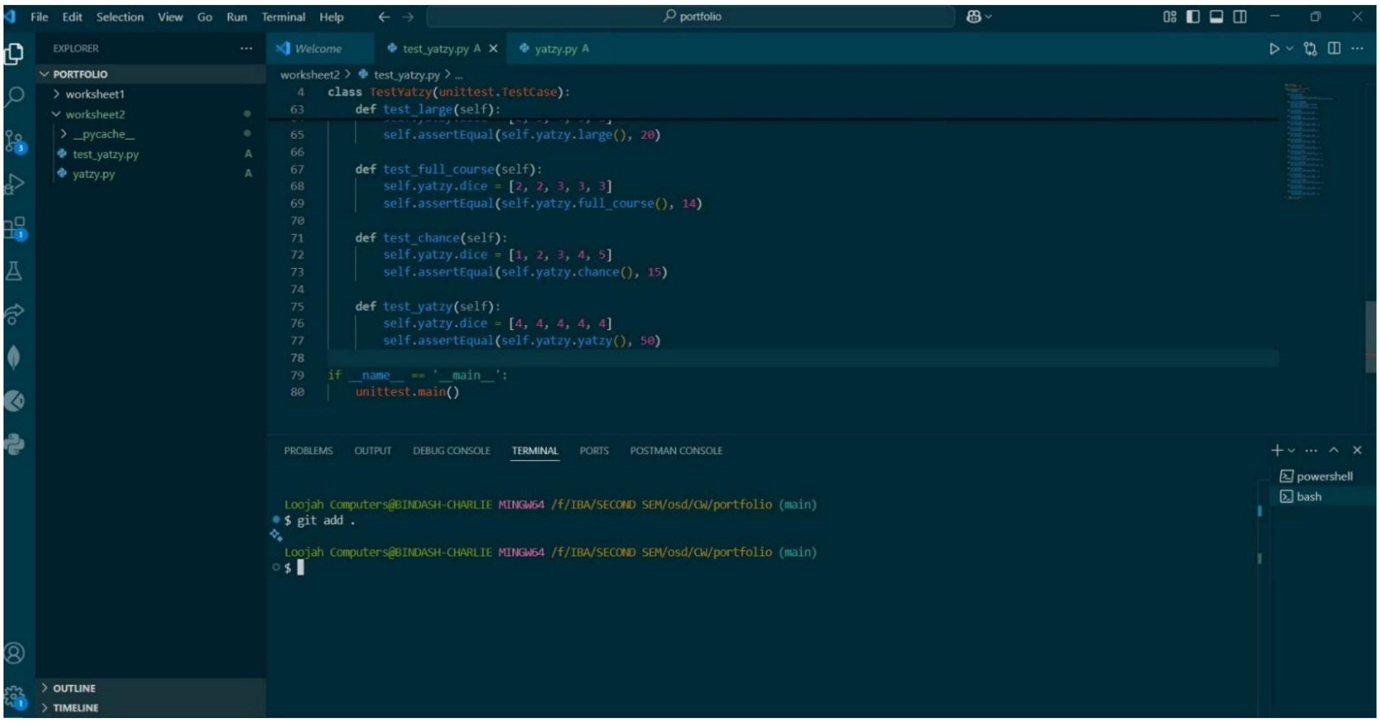
```python
class TestYatzy(unittest.TestCase):
    def test_large(self):
        self.assertEqual(self.yatzy.large(), 20)

    def test_full_course(self):
        self.yatzy.dice = [2, 2, 3, 3, 3]
        self.assertEqual(self.yatzy.full_course(), 14)

    def test_chance(self):
        self.yatzy.dice = [1, 2, 3, 4, 5]
        self.assertEqual(self.yatzy.chance(), 15)

    def test_yatzy(self):
        self.yatzy.dice = [4, 4, 4, 4, 4]
        self.assertEqual(self.yatzy.yatzy(), 50)

if __name__ == '__main__':
    unittest.main()
```

```
Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        worksheet2/

nothing added to commit but untracked files present (use "git add" to track)

Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$ 
```

Stage Changes

git add .

Git add is a command we use to tell Git which changes we want to save in your project. When we make changes to files in your project, we use to select the specific files or modifications we want to keep. Once we've added them, they are staged and ready to be saved.



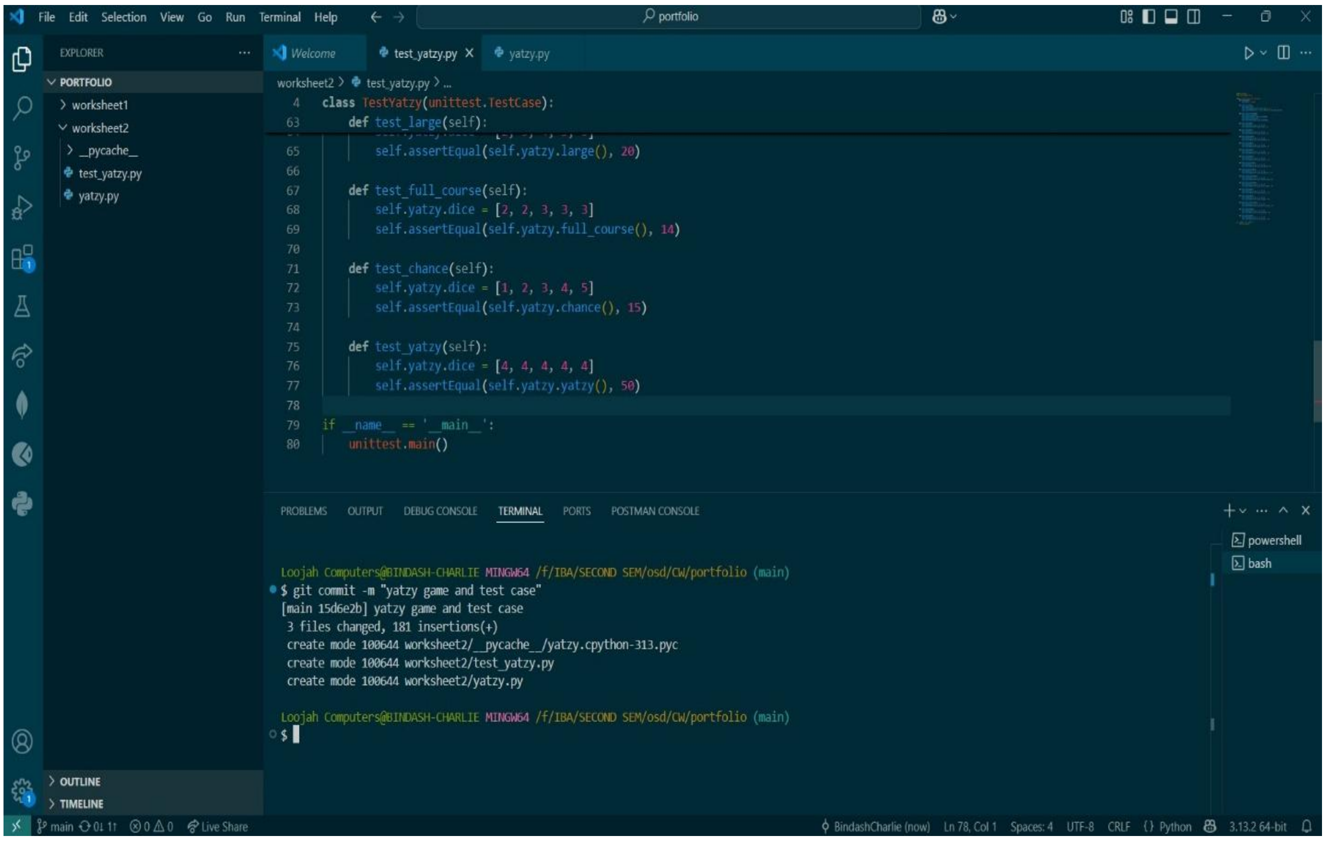
```
class TestYatzy(unittest.TestCase):
    def test_large(self):
        self.assertEqual(self.yatzy.large(), 20)

    def test_full_course(self):
        self.yatzy.dice = [2, 2, 3, 3, 3]
        self.assertEqual(self.yatzy.full_course(), 14)

    def test_chance(self):
        self.yatzy.dice = [1, 2, 3, 4, 5]
        self.assertEqual(self.yatzy.chance(), 15)

    def test_yatzy(self):
        self.yatzy.dice = [4, 4, 4, 4, 4]
        self.assertEqual(self.yatzy.yatzy(), 50)

if __name__ == '__main__':
    unittest.main()
```

```
Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$ git add .

Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$ 
```

Commit Changes

git commit -m "first commit"

When we invoke git commit, we're asking Git to commit the changes we've made and staged with git add. This will keep the state of our 4 files, along with the changes we've chosen, as a new snapshot or version of our project. Every commit includes a message explaining the change, so it's easy to trace the history of our project and understand what was done and why. The changes, once implemented, are safely preserved in project history and can be recalled or undone if necessary.



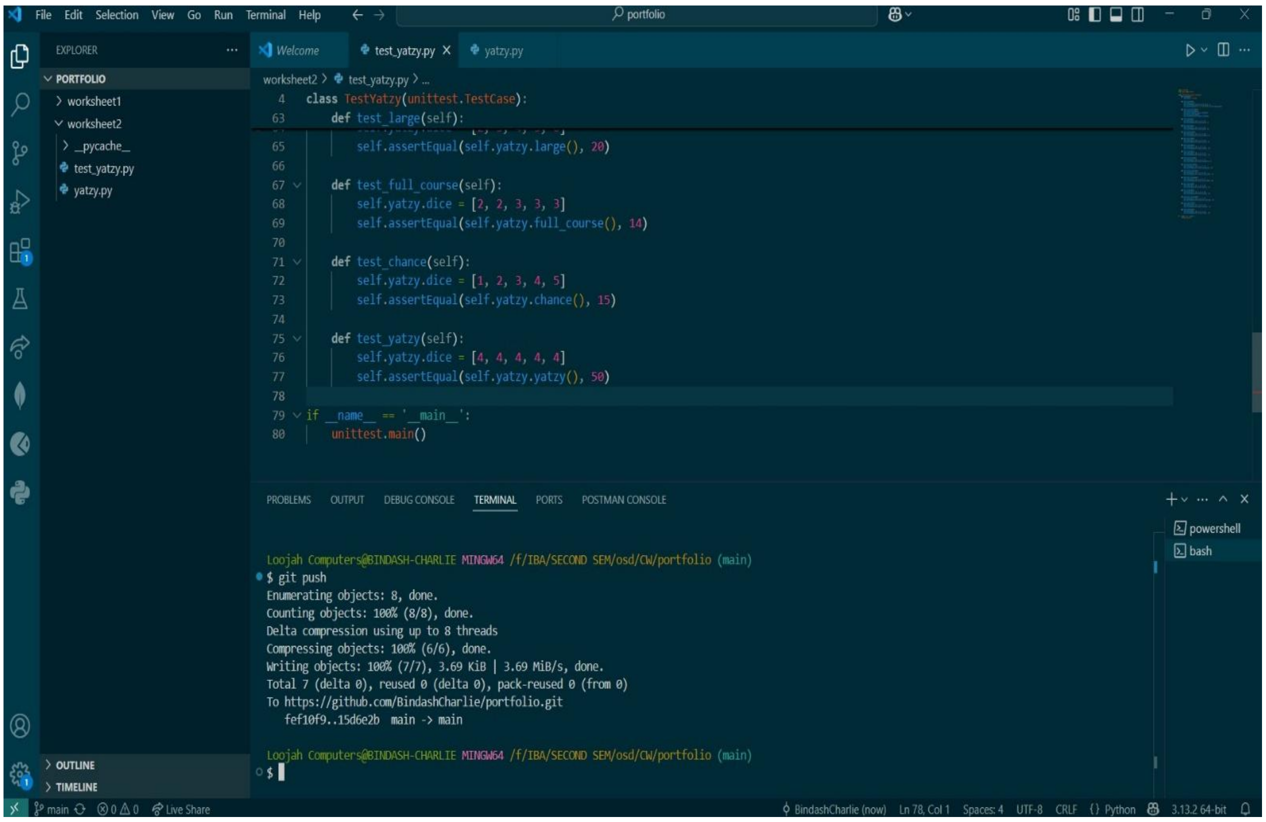
```
class TestYatzy(unittest.TestCase):
    def test_large(self):
        self.assertEqual(self.yatzy.large(), 20)

    def test_full_course(self):
        self.yatzy.dice = [2, 2, 3, 3, 3]
        self.assertEqual(self.yatzy.full_course(), 14)

    def test_chance(self):
        self.yatzy.dice = [1, 2, 3, 4, 5]
        self.assertEqual(self.yatzy.chance(), 15)

    def test_yatzy(self):
        self.yatzy.dice = [4, 4, 4, 4, 4]
        self.assertEqual(self.yatzy.yatzy(), 50)

if __name__ == '__main__':
    unittest.main()
```

```
Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$ git commit -m "yatzy game and test case"
[main 15d6e2b] yatzy game and test case
 3 files changed, 181 insertions(+)
 create mode 100644 worksheet2/__pycache__/yatzy.cpython-313.pyc
 create mode 100644 worksheet2/test_yatzy.py
 create mode 100644 worksheet2/yatzy.py

Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$ 
```

Push to GitHub

git push origin main

Git push is the command that we use to transmit our project updates on the local system to a remote repository (e.g. GitHub or GitLab). Having updated our project and staged the updates with `git add` and committed it with `git commit`, we employ `git push` so that other users can view the changes. This command pushes our changes to the remote repository so that other team members can see and utilize the updated version of the project. It informs all the team members about the latest status of the project.

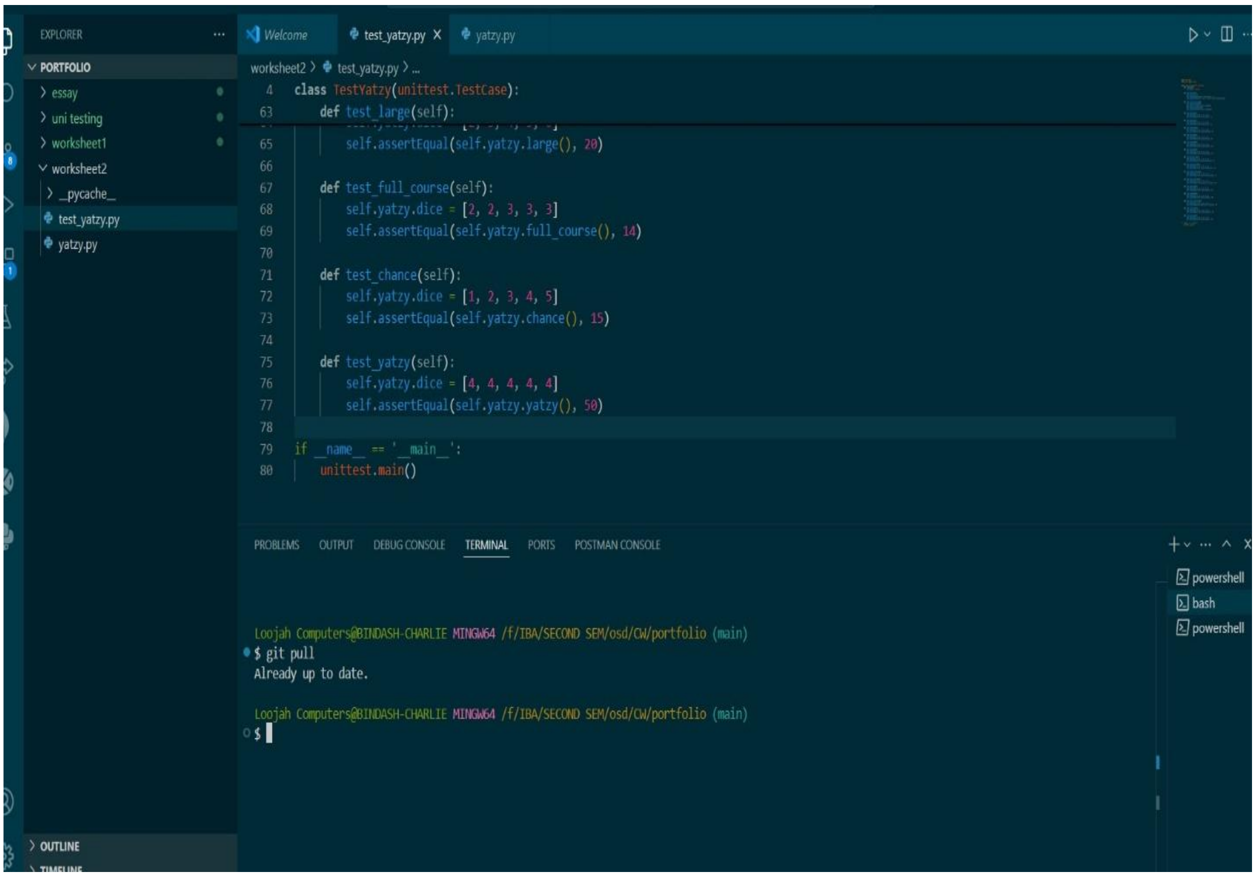

```python
class TestYatzy(unittest.TestCase):
    def test_large(self):

        self.assertEqual(self.yatzy.large(), 20)

    def test_full_course(self):
        self.yatzy.dice = [2, 2, 3, 3, 3]
        self.assertEqual(self.yatzy.full_course(), 14)

    def test_chance(self):
        self.yatzy.dice = [1, 2, 3, 4, 5]
        self.assertEqual(self.yatzy.chance(), 15)

    def test_yatzy(self):
        self.yatzy.dice = [4, 4, 4, 4, 4]
        self.assertEqual(self.yatzy.yatzy(), 50)

if __name__ == '__main__':
    unittest.main()
```

```
Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$ git push
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 3.69 KiB | 3.69 MiB/s, done.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/BindashCharlie/portfolio.git
   fef10f9..15d6e2b  main -> main

Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$
```

Pull Changes

git pull origin main

The Git pull command functions to retrieve remote repository changes from GitHub or GitLab into our local project. The git pull command allows us to obtain updates from the remote repository which contains changes made by other team members. The process maintains our project's status with the newest modifications from other contributors. The process ensures our local project files match the content of the remote repository.



```
class TestYatzy(unittest.TestCase):
    def test_large(self):
        self.assertEqual(self.yatzy.large(), 20)

    def test_full_course(self):
        self.yatzy.dice = [2, 2, 3, 3, 3]
        self.assertEqual(self.yatzy.full_course(), 14)

    def test_chance(self):
        self.yatzy.dice = [1, 2, 3, 4, 5]
        self.assertEqual(self.yatzy.chance(), 15)

    def test_yatzy(self):
        self.yatzy.dice = [4, 4, 4, 4, 4]
        self.assertEqual(self.yatzy.yatzy(), 50)

if __name__ == '__main__':
    unittest.main()
```

```
Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$ git pull
Already up to date.

Loojah Computers@BINDASH-CHARLIE MINGW64 /f/IBA/SECOND SEM/osd/CW/portfolio (main)
$
```

What is Unit Testing?

Unit testing is a method of software testing where individual "units" of a program are tested to verify they behave as required. In the context of the Yatzy game, a "unit" is the smallest testable unit, i.e., a method of the Yatzy class (e.g., Ones(), TwoPairs(), Yatzy()).

- **Key Characteristics**:
    - **Isolation**: Each unit is tested individually independent of the rest of the program. For instance, testing TwoPairs () has nothing to do with the overall game logic, just the dice value.
    - **Automation**: is typically automated and thus can be executed repeatedly with minimal effort (e.g., via a testing framework like pytest).
    - **Granularity**: Focused on single functionality in contrast with the whole application.
- **Purpose**:
    - Confirm that each method returns the correct output for given inputs.
    - Detect issues early during development.

Why is Unit Testing Important?

1. **Reliability**: Ensures your Yatzy strategies compute scores accurately (e.g., Yatzy() returns 50 points only if the five dice are identical).
2. **Maintainability**: Enables code changes (e.g., enhancing Pair() logic) by preventing updates from introducing new bugs.
3. **Automation in CI/CD**: Automated tests can be integrated with tools like GitHub Actions, running on every code commit to maintain quality.
4. **Documentation**: Tests provide clear examples of what methods should accomplish (e.g., a test for LargeStraight() shows that it expects the sequence 2-3-4-5-6 to appear).
5. **Teamwork**: Unit tests allow other team members to test your code, so methods work as intended when inserted into the bigger picture.

How Unit Testing Applies

- Implementing a Yatzy class with specific methods.
- Writing unit tests for each method to verify their scoring logic (e.g., Ones (), TwoPairs() etc).
- Configuring automated testing with GitHub Actions to execute tests.

Open-Source Development and Its Relevance

Introduction

Open-source development is revolutionizing the software creation environment, providing collaboration and access that proprietary systems do just not afford. Open-source software (OSS) offers its source code to all comers. With licenses like the Apache License or GNU General Public License, therefore, a new kind of freedom arises people are allowed to take, change and share it. At this level it fuels a whole host of essential technologies from the Apache web server to Python programming language. This essay delves into the essence of open-source development, its broader impact, and its personal significance as I prepare to enter the tech industry. Through this module's portfolio- building a Yatzy class, using Github, and a collaborating with Peer- I have gained skills and insights that highlight the vital role of open source in my education and future career.

What is Open-Source Development?

Open-source development rests on openness and collective effort. It starts as programmers share their code on pages like GitLab, GitHub, or Source Forge, where anyone can submit the improvements, bugs fix or reuse the program. An instance is the Apache HTTP servers that powers much of the internet owing to code inputs from a global community. Similarly, Git which is a revolutionized version control, enabling collaborative coding at scale-tools I've used in my coursework.

The open-source model is rooted in processes like peer review and continuous improvement. Review by the community ensures solid, reliable code, as with software like Mozilla Firefox. Issues exist, including organizing the divergent contributions of discovering stable funding. These aside, the advantages—rapid innovation, cost savings, and flexibility—make OSS inexorable. Within this module, I applied these principles to develop a Yatzy game on GitLab, using version control, automated testing through GitLab CI/CD, and peer collaboration via merge requests. Through this hands-on work, I have been able to see how open-source workflows are applied in real-world contexts and make the process both accessible and inspiring.

Collaboration of Developers

Open-source development flourishes on its power to unite individuals across the world. It unites programmers, hobbyists, and professionals in a shared forum to exchange ideas and knowledge, creating a feeling of ownership and belongings. Such diversity fuels progress, as developers with diverse skills work on different parts of a project concurrently. For beginners, open source provides an environment that is friendly to learn from, with seasoned contributors frequently mentoring. This collaboration has been evident in my projects, where I worked with other to enhance my Yatzy game, learning from the experience that improved both my code and my understanding of collaboration.

Relevance to Me as a Student

As a computer student, open-source development is core to my studies. This module provided me with hands-on skills that are most desirable to industry. Learning GitLab, for example, was life changing. In Worksheet 1, I got to know how to use commit, merge, branch, and push commands to manage my Yatzy game project. Such skills are most important in collaborative development, where tracking changes and merging contributions is the order of the day. GitLab taught me the value of disciplined workflows—using branches for new features or bug fixes keeps my work organized and recoverable, a practice I’ll apply in future projects or jobs.

Unit testing, which we covered in Worksheet 2, sharpened my focus on the reliability of code. Creating tests for my "Yatzy" class method (i.e., checking 'Two Pairs()' return 0 except when two distinct pairs exist) forced me to anticipate edge cases and create specific outcomes. Executing these tests with Gihub CI/CD introduced me to continuous integration, the defining characteristic of open-source projects that ensures code quality through automated verification. This habit has changed my software development methodology.

Teamwork, emphasized in Worksheet 3, echoed open-source community processes. Collaborating with a peer to review and fix an error in my code (e.g., enhancing 'Two Pairs()'function) illustrated the power of positive feedback. Using Github issues to annotate and fix errors resembles open-source maintainers' handling of submissions. These lessons honed communication and analytical ability, making me prepared for working environments where diversity of input leads to success.

Aside from technical competence, open source promotes a culture of openness and a desire to learn throughout life. Experience with OSS tools has provided me with the confidence to contribute to real projects. My Github repository, developed through this coursework, is a portfolio showing my skills to potential employers—a precious asset as I near graduation.

Cost-Effectiveness

Open-source software is cost-effective by its very nature, eliminating expensive licenses. Open-source software can be taken up by individuals and businesses without cost barriers, channeling savings into innovation or employee development. Contributions voluntarily rendered make many open-source projects even cheaper to create. This open access to technology democratizes it, enabling startups and makers to compete without having to incur significant initial costs.

Customizability and Adaptability

One of the key characteristics of open-source software is its flexibility. Developers can customize the source code to satisfy needs, enabling custom-made solutions for companies or individuals. This customizability sparks creativity, as authors experiment with new features or combinations that the core developers did not envision. The WordPress system, for instance, founded on open-source programming, hosts millions of sites owing to its customizability by plug-ins and themes. In my Yatzy game project, I modified game logic to include an AI opponent, a modification that furthered my understanding of OSS's versatility.

Innovation

Open-source projects are incubators of innovation. Released from the shackles of proprietary solutions, developers experiment with novel algorithms, architectures, and applications. The Linux kernel, started by Linus Torvalds, is an excellent example—its openness led to its application ranging from supercomputers to IoT devices. Similarly, projects like TensorFlow and PostgreSQL have transformed machine learning and database administration. My interaction with Python's open-source community in this module motivated me to experiment with libraries like Pygame, and subsequently, ideas about what to develop next.

Community Support

Open-source projects are rich communities offering documentation, forums, and tutorials to support users and developers. Beyond technical assistance, such projects collaborate on planning project goals, proposing features, and sharing best practices, making the projects worthwhile. Joining in my peers' conversations during this module was like being part of such a community—a microcosm unto itself—because our discussions about GitLab made my project better and established connections that extend beyond the class. That experience inspires me to go out and explore more about larger OSS communities after graduation.

Licensing and Legal Considerations

Open-source projects operate under licenses that define the use of code, changes, and sharing. Permissive licenses, like MIT, give leeway, while copyleft licenses, like GNU GPL, enforce derivative works to remain open source. Understanding the difference is critical to avoid legal complications and ensure project integrity. While pursuing my studies, I ensured my Yatzy game conformed to the MIT License, further highlighting software developer awareness of licenses.

Global Impact

Open-source development challenges national boundaries, getting people from across the globe involved. This diversity produces solutions for varied demands, from enterprise solutions to software for underprivileged populations. Open-source software-powered projects like Raspberry Pi introduced computing to remote locations, bridging digital divides. My class coursework collaboration with global peers mirrored this global ethos, showing me, the value of common goals to result in successful outcomes.

Conclusion

Open-source development is more than a methodology—it’s a philosophy of inclusion and innovation that aligns with my aspirations as a student. Through this module, I’ve gained proficiency in Git, unit testing, and CI/CD, skills that position me well for a tech career increasingly shaped by OSS. Building and refining my Yatzy game on GitLab has shown me the value of iterative collaboration, reflecting the open-source ethos of community-driven progress.

Forward-looking, open source will define my career. Whether I'm implementing OSS tools within a corporation or contributing to projects myself, transparency, quality, and collaboration will be the watchwords. This coursework has not only expanded my technical skill set but also fostered a love of continuing learning—open source evolves, and I look forward to adapting with it. My GitLab portfolio, a product of this experience, is proof that I am ready to engage the open-source world as I embark on my professional career.