PYTHON COURSE PRESENTATION

ABHINAV T K

Flax & Teal 2024

Agenda

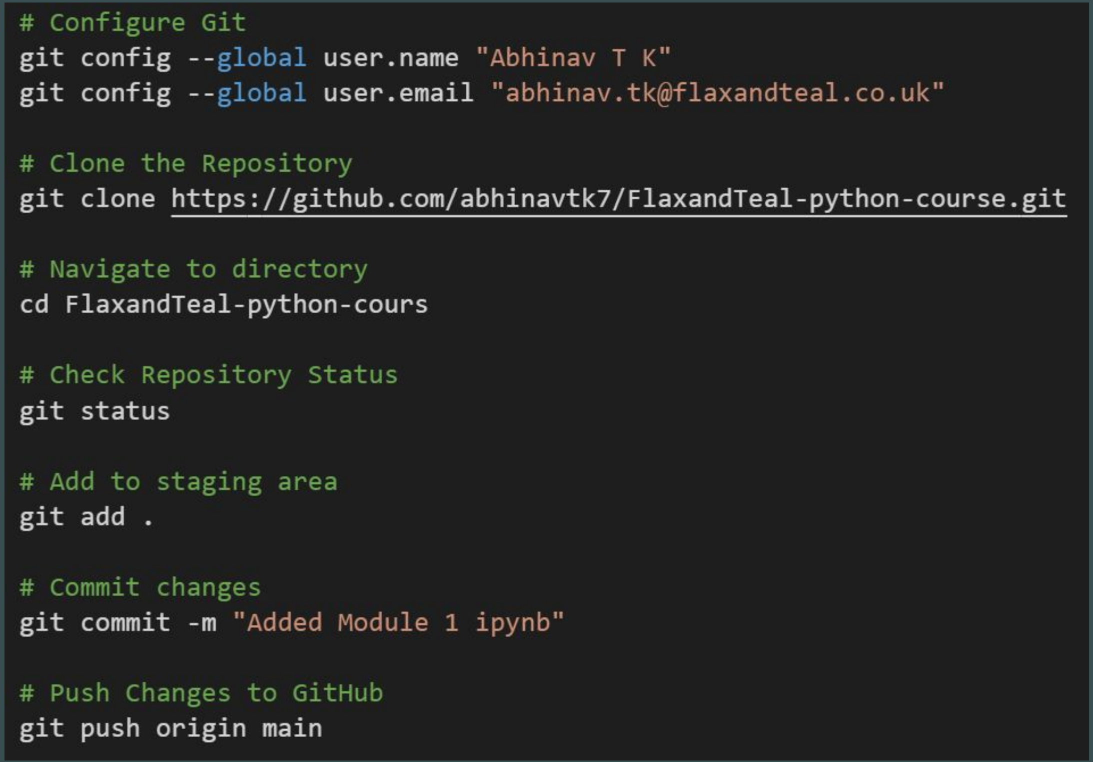
1. Introduction to Git
2. Setting up a GitHub Repository
3. Basic Git commands
4. Learnings
5. Bioinformatics
6. Python tools
7. Methodology
8. Learnings

Introduction to Git

- Git is a distributed version control system used for tracking changes in source code during software development.
- Distributed System
- Version Control
- Collaboration
- Branching and Merging
- Backup and Restore

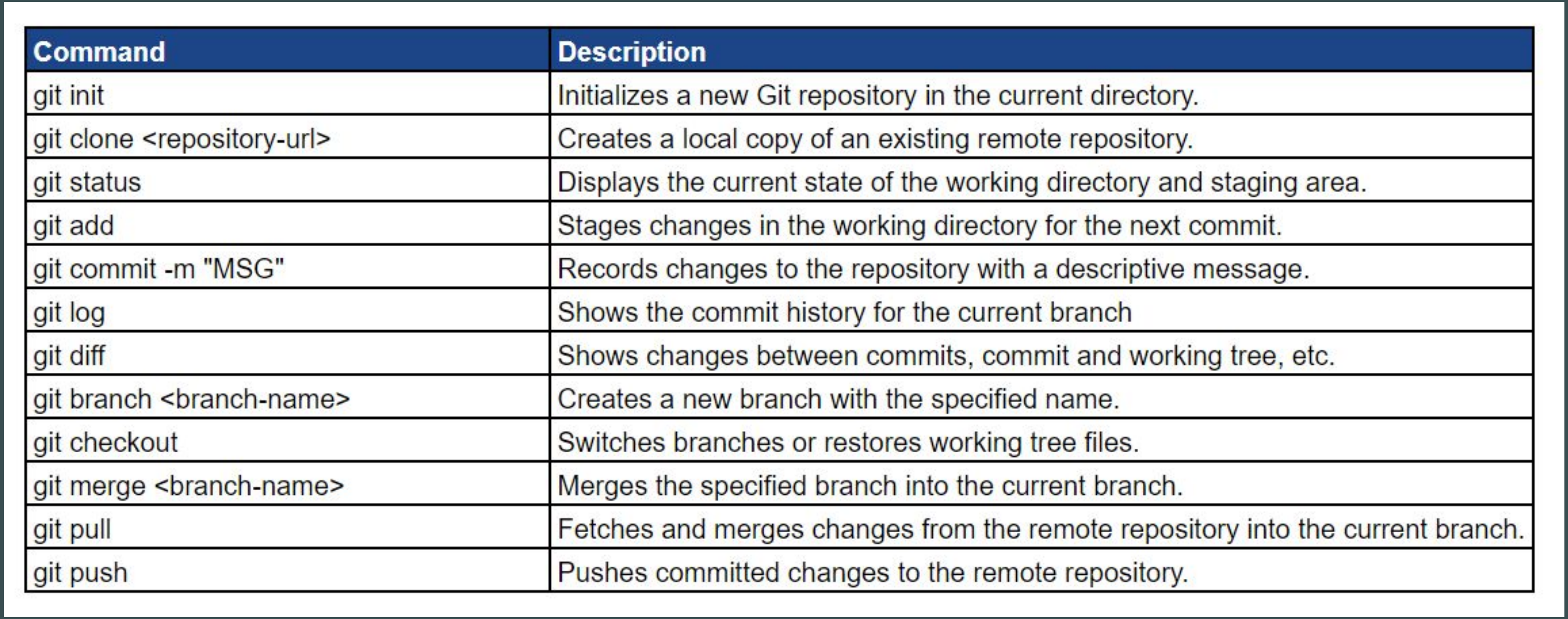
Setting up a Github Repo to host the Jupyter notebooks

1. Create a GitHub Repository
2. Set Up Git Locally
3. Clone the Repository to Local Machine
4. Add Python Notebooks to Local folder
5. Add, Commit, and Push Files to GitHub
6. Setup README file



```
# Configure Git
git config --global user.name "Abhinav T K"
git config --global user.email "abhinav.tk@flaxandteal.co.uk"

# Clone the Repository
git clone https://github.com/abhinavtk7/FlaxandTeal-python-course.git

# Navigate to directory
cd FlaxandTeal-python-cours

# Check Repository Status
git status

# Add to staging area
git add .

# Commit changes
git commit -m "Added Module 1 ipynb"

# Push Changes to GitHub
git push origin main
```

Git commands



Command	Description
git init	Initializes a new Git repository in the current directory.
git clone <repository-url>	Creates a local copy of an existing remote repository.
git status	Displays the current state of the working directory and staging area.
git add	Stages changes in the working directory for the next commit.
git commit -m "MSG"	Records changes to the repository with a descriptive message.
git log	Shows the commit history for the current branch
git diff	Shows changes between commits, commit and working tree, etc.
git branch <branch-name>	Creates a new branch with the specified name.
git checkout	Switches branches or restores working tree files.
git merge <branch-name>	Merges the specified branch into the current branch.
git pull	Fetches and merges changes from the remote repository into the current branch.
git push	Pushes committed changes to the remote repository.

Learnings

- Version Control with Git: Setting Up Git, Cloning Repositories, Staging and Committing Changes, Pushing Changes to GitHub
- Collaborative Development: Understand the basics of collaborative development, including branching, merging and pull request.
- Writing README Files: Gain experience in writing comprehensive README files using Markdown to provide clear documentation and instructions for repository users.

Bioinformatics

- The genetic similarity can be understood by comparing long sequences of DNA which consist of only four bases: G, A, T and C.
- Levenshtein Distance: minimum number of single-character edits to get from the first string to the second.
- Two ways to calculate Levenshtein Distance (python_course_levenshtein_py.py)
  1. recursive approach
  2. iterative approach

Python tools

- **pip**: pip is a package management system used to install and manage software packages written in Python. pip install package_name
- **pytest**: Popular testing framework for Python, used to write simple and scalable test cases. It automatically discovers test files and functions.
- **pylint**: static code analysis tool for Python. It checks your Python code for errors and enforces a coding standard based on PEP 8. It provides suggestions for refactoring your code to improve its readability and maintainability.

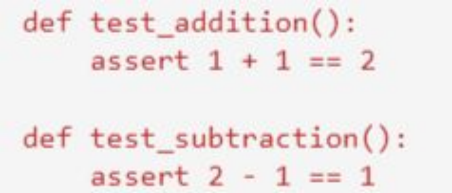
Methodology

- Ran pytest for a custom Levenshtein distance implementation, with two fixtures for generating test strings.

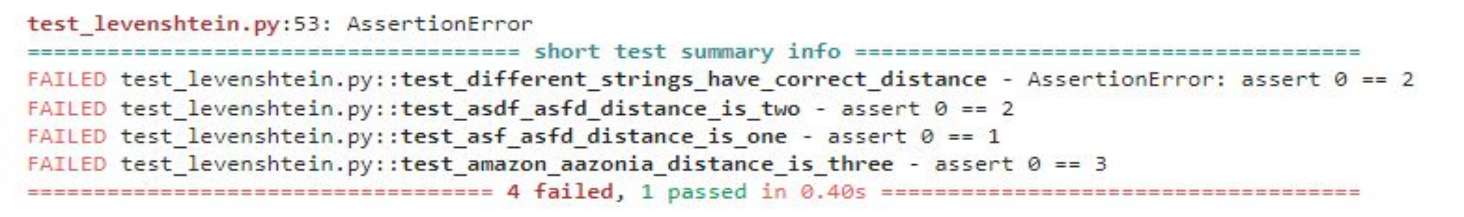
❑ **Test the function with identical strings.**

❑ **Test the function with known different strings.**

❑ **Test specific cases with pre-determined results.**



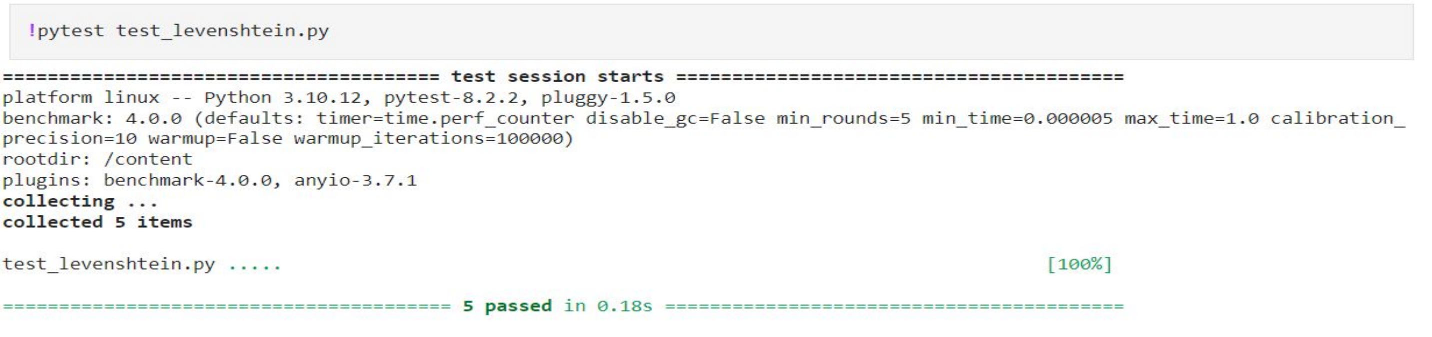
```
def test_addition():
    assert 1 + 1 == 2

def test_subtraction():
    assert 2 - 1 == 1
```



```
test_levenshtein.py:53: AssertionError
======================================= short test summary info ========================================
FAILED test_levenshtein.py::test_different_strings_have_correct_distance - AssertionError: assert 0 == 2
FAILED test_levenshtein.py::test_asdf_asfd_distance_is_two - assert 0 == 2
FAILED test_levenshtein.py::test_asf_asfd_distance_is_one - assert 0 == 1
FAILED test_levenshtein.py::test_amazon_aazonia_distance_is_three - assert 0 == 3
==================================== 4 failed, 1 passed in 0.40s =======================================
```

Learnings

- Using pytest helped to learn how to write test cases, use fixtures, and debug test failures.
- Running pytest on the test file helped to validate the correctness of implementation of my Levenshtein distance function as it compares my implementation's output with that of a known library (Levenshtein Module).
- The test cases include various input string scenarios. This helps ensure that the function handles different edge cases and inputs correctly.

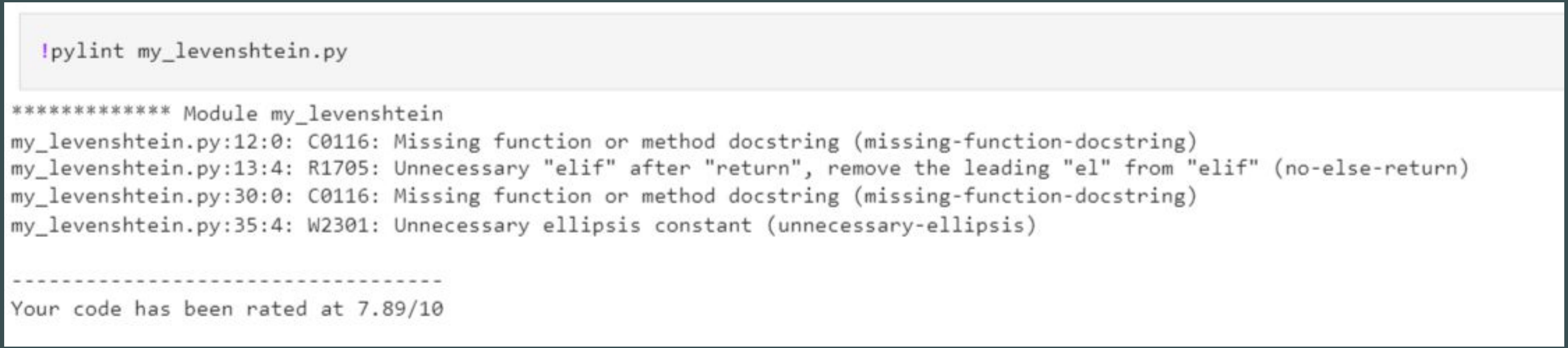


```
!pytest test_levenshtein.py
```

```
======================================= test session starts ========================================
platform linux -- Python 3.10.12, pytest-8.2.2, pluggy-1.5.0
benchmark: 4.0.0 (defaults: timer=time.perf_counter disable_gc=False min_rounds=5 min_time=0.000005 max_time=1.0 calibration_
precision=10 warmup=False warmup_iterations=100000)
rootdir: /content
plugins: benchmark-4.0.0, anyio-3.7.1
collecting ...
collected 5 items

test_levenshtein.py .....                                                                   [100%]

======================================== 5 passed in 0.18s =========================================
```

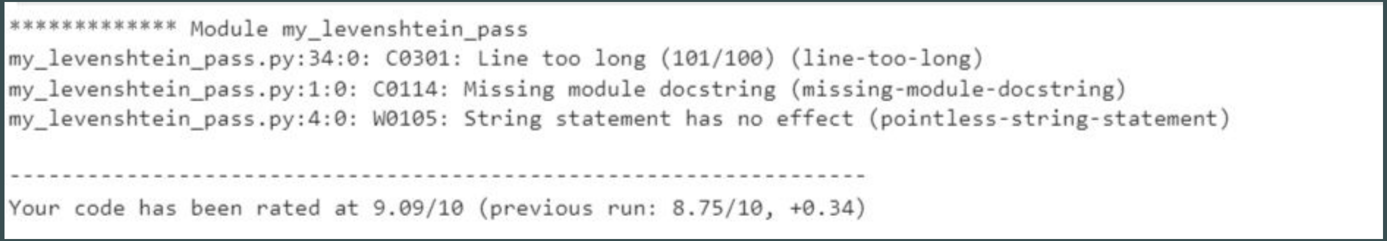
Methodology

- Running pylint provides various warnings and suggestions for improving the code.



```
!pylint my_levenshtein.py
```

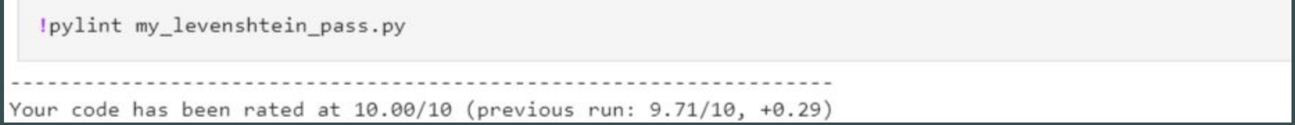
```
************* Module my_levenshtein
my_levenshtein.py:12:0: C0116: Missing function or method docstring (missing-function-docstring)
my_levenshtein.py:13:4: R1705: Unnecessary "elif" after "return", remove the leading "el" from "elif" (no-else-return)
my_levenshtein.py:30:0: C0116: Missing function or method docstring (missing-function-docstring)
my_levenshtein.py:35:4: W2301: Unnecessary ellipsis constant (unnecessary-ellipsis)

-----------------------------------
Your code has been rated at 7.89/10
```



```
************* Module my_levenshtein_pass
my_levenshtein_pass.py:34:0: C0301: Line too long (101/100) (line-too-long)
my_levenshtein_pass.py:1:0: C0114: Missing module docstring (missing-module-docstring)
my_levenshtein_pass.py:4:0: W0105: String statement has no effect (pointless-string-statement)

------------------------------------------------------------------
Your code has been rated at 9.09/10 (previous run: 8.75/10, +0.34)
```

Learnings

- Pylint checks for errors, enforces a coding standard, and suggests best practices.
- Naming Conventions: Pylint may suggest using snake_case for function names and variables to adhere to PEP 8 standards.
- Docstrings: Pylint might recommend adding docstrings to your functions to describe their purpose, parameters, and return values. This helps with code readability and maintenance.
- Code Simplification and Readability: improves the code's clarity by eliminating redundant conditions, making it easier to understand and maintain. Eg. elif after a return can be simplified to an if statement.



```
!pylint my_levenshtein_pass.py
```

```
-------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.71/10, +0.29)
```

FLAX & TEAL

Thank you

Flax & Teal 2024