

MaleeshaWeerasekara_217285 58_ISEReport.pdf

by Turnitin LLC

Submission date: 29-May-2024 12:48AM (UTC+0700)

Submission ID: 2350214767

File name: 8_2024_05_28_MaleeshaWeerasekara_21728558_I_de008127195c5228.pdf (776.2K)

Word count: 2879

Character count: 20977



Curtin University

ISAD5004 – Introduction to Software Engineering

Final Assignment

Student Id – 21728558

Name – Maleesha Lasith Weerasekara



Contents

Table of Figures	4
List of Tables	4
Introduction	5
Module Description	5
Tech Stack	5
Modules	6
Main Module	7
Numerology Module (master_number.py)	8
Generation Module (generation.py)	8
Inputs Validation Module (validation.py)	9
Life Path class (life_path.py)	9
Test Module	10
Design Decisions	10
Applications of OOP	11
Black Box Testing	12
White-Box Test Cases	14
Test Implementation and Execution	15
Summary	16
Version Controlling	17
Discussion	20
Challenges	20
Limitations	20
Future Improvements	21

Table of Figures

Figure 1: Tech Stack	6
Figure 2: Project Structure	7
Figure 3: Main Class	8
Figure 4: Master Number class	8
Figure 5: Generation Class	9
Figure 6: Validation class	9
Figure 7: Life Path class	10
Figure 8: Class Description.....	11
Figure 9: Test Class	15
Figure 10: Work Flow Table.....	16
Figure 11: Dev Branch.....	17
Figure 12: Merging Main branch	18
Figure 13: Git pushing changes	19

List of Tables

Table 1: Design Choices and Benefits	12
Table 2: Black-Box Testing	14
Table 3: White-Box Testing.....	15

Introduction

The project created a numerology analysis tool that uses a person's birthdate to determine their Life Path Number, Lucky Colour, master numbers, and generational cohorts. As the example situation explains, it must abide by a number of crucial regulations. A software corporation has asked that tools for numerological analysis be developed. Two cases are examined: figuring out the Life Path Number and determining the Lucky Color when a birthday is entered. Finding out if two birthdays have the same Life Path Number is another goal of the research. The computation technique is predicated on lucky colors and master numbers. When a person's birthday is given, the program further attempts to locate the generate. The program takes different birth dates and is modular, easy to maintain and extend. To guarantee accuracy and resilience, it has both white-box and black-box test scenarios. Invalid dates and years are handled using input validation procedures. Throughout the development process, version control procedures were used to monitor modifications and facilitate productive teamwork. This methodical process has produced a dependable and expandable numerology analysis tool that is prepared for further improvements.

This project is a utilization of both black box and white box methodologies in the testing process to measure the modules' accuracy and latency. By using boundary value analysis and equal partitioning, tests are made to handle a wide range of scenarios. Aside from this, the project's last step involves managing code versions and tracking changes using integration with git version control, the source control method. This document will provide insight into th development of requested project scenario and testing it allowing to get clarity of professional development and version controlling process.

Module Description

Tech Stack

High-level, interpreted Python 3 is a well-liked language ³ for its ease of reading and simplicity. It is compatible with several programming paradigms, such as functional, object-oriented, and procedural programming. Python's easily comprehensible syntax facilitates the writing of clear, maintainable code. It employs dynamic typing for flexible development, supports object-oriented programming (OOP), and has a large standard library. Python boasts a sizable and vibrant community as well as an extensive ecosystem of third-party frameworks and tools. For the numerology tool, Python's flexibility, simplicity of testing, and built-in string and number handling skills were essential. Git is used for version control on the web-based platform GitHub, which also offers developers a collaborative workspace. Version control, branch management, documentation, collaboration, and CI/CD tool integration are all made possible by it.



Figure 1: Tech Stack

Modules

As details were provided by underlined company, main details and including of the module can be provided as below. When a birthday was entered to the model it needs to preform following tasks including;

- Life path number need to be calculated.
- Lucky color needs to be identified with the life path number,
- It needs to be determined if the life path number is a master number.

As an alternative, the program determines the fortunate hue based on the Life Path number and determines whether two birthdays have the same Life Path number. For simplicity, it makes use of master numbers, fortunate colors, and a variety of techniques. A person's generational categories are established when their birthday is disclosed. File structure of project can be explained in following branch structure.

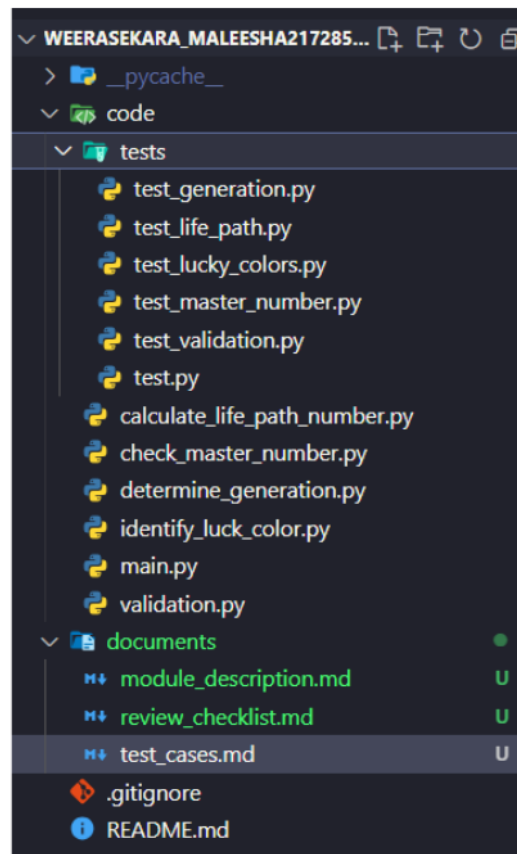


Figure 2: Project Structure

Following models were designed and implemented to preform discussed tasks above.

Main Module

This is the application's entry point, managing user interaction and coordinating calls to other modules in response to input from the user. As working as hub for multiple classes and functions were created separately, were imported in this script to utilize with expected user features. It offers functions to find master numbers, calculate life path numbers, find lucky colors, compare life path numbers between two dates, and categorize generations. A number menu was included to access each feature and `main()` was called implement executable app which runs with python command.


```

code > main.py > main
You, 2 days ago | 1 author (You)
1 from calculate_life_path_number import LifePath
2 from identify_luck_color import LuckyColour
3 from check_master_number import MasterNumber
4 from determine_generation import Generation
5 from validation import Validation
6
7 def main():
8     while True:
9         print("1. Calculate Life Path Number")
10        print("2. Determine Generation")
11        print("3. Compare Life Path Numbers of Two Birthdays")
12        print("0. Exit")
13        choice = input("Enter your choice: ")
14
15        if choice == '1':
16            date = input("Enter birth date (YYYY-MM-DD): ")
17            suit = Validation.validate_date(date)
18            print(suit)
19            if suit: ...
20        elif choice == '2': ...
21        elif choice == '3': ...
22        elif choice == '0': ...
23        else:
24            print("Invalid choice. Please try again.")
25
26 if __name__ == "__main__":
27     main()

```

Figure 3: Main Class

Numerology Module (master_number.py)

This module includes routines that take a given birth date and use it to determine the Life Path Number. It determines if the Life Path Number is a master number by applying a certain computation technique. Additionally, it uses the predetermined procedure to map the Life Path Number to a matching Lucky Color.

```

1 class MasterNumber:
2     @staticmethod
3     def is_master_number(number):
4         return number in (11, 22, 33)
5

```

Figure 4: Master Number class

Generation Module (generation.py)

Based on their dates of birth, this module groups people into generational cohorts. To identify the generation a person belongs to, it employs pre-established generational categories.

```

1 class Generation:
2     GENERATIONS = [
3         (1901, 1927, "G.I. Generation"),
4         (1928, 1945, "Silent Generation"),
5         (1946, 1964, "Baby Boomers"),
6         (1965, 1980, "Generation X"),
7         (1981, 1996, "Millennials"),
8         (1997, 2012, "Generation Z"),
9         (2013, 2024, "Generation Alpha"),
10    ]
11
12    @staticmethod
13    def determine_generation(year):
14        for start, end, generation in Generation.GENERATIONS:
15            if start <= year <= end:
16                return generation
17        return "Unknown Generation"
18

```

Figure 5: Generation Class

Inputs Validation Module (validation.py)

This module verifies that the birth dates entered are correctly formatted and are within the permitted range of 1901–2024. The expected format of user input is given as YYYY-MM-DD. For the convenience of the user, it supports both named month and numerical forms.

```

1 class Validation:
2     @staticmethod
3     def validate_date(date):
4         try:
5             parts = date.split('-')
6             if len(parts) != 3:
7                 return False
8             year, month, day = map(int, parts)
9             return 1901 <= year <= 2024 and 1 <= month <= 12 and 1 <= day <= 31
10        except ValueError:
11            return False
12
13    @staticmethod
14    def validate_year(year):
15        try:
16            year = int(year)
17            return 1901 <= year <= 2024
18        except ValueError:
19            return False
20

```

Figure 6: Validation class

Life Path class (life_path.py)

A crucial numerology tool is the LifePath class, which computes the Life Path Number, ascertains whether it's a master number, and ascertains the Lucky Color connected to it. This ensures that the calculation process is modular and maintainable.

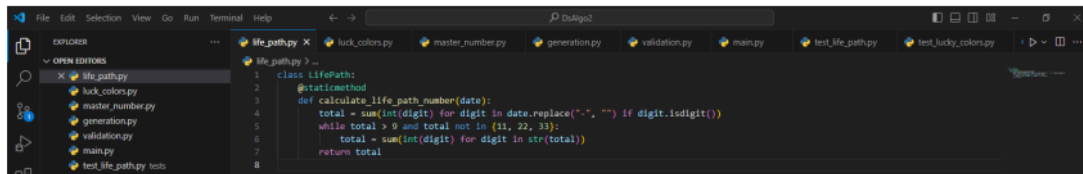


Figure 7: Life Path class

Test Module

Both **white-box** and **black-box** testing include thorough **test cases** in the testing module. It confirms that the functions for classifying generations and numbers are accurate and reliable. To guarantee dependability, the test cases are made to span a broad range of valid and incorrect inputs.

Design Decisions

To guarantee simplicity of use, scalability, and maintainability, the program adheres to modular principles. Clear responsibilities are incorporated into each module's architecture to preserve the separation of concerns. This method lowers complexity and improves code quality by enabling autonomous development, testing, and maintenance of each module. The user interface and interaction logic are managed by the main module (main.py), which makes sure that modifications don't impact the essential functionality of the underlying development. Specialized functionality is provided by modules such as generation., luckycolor, and lifepath modules, which enable their functionalities to be reused across different areas of the program without becoming redundant. Every module has a personal accountability objective that makes maintenance and troubleshooting easier. modifications made to one module won't impact other modules thanks to encapsulation, which guarantees that modifications won't negatively impact other modules.

Python Script	Covered Aspects
main.py	<p>Only Maintains what need to be shown at interfaces and initiate connection with relevant features according to user interactions.</p> <p>Single responsibility – Only handling import and user interactions.</p> <p>OOP – Supports encapsulation</p> <p>Other – Scalable, Maintainable, Testable</p>
generation.py	<p>Only Maintains which need to determine user's generation containing necessary data in tabular formats. Allows to initiate any where with data like Year.</p> <p>Output – Generation as String</p>

	Single responsibility – Only handling Generation selecting responsibility. OOP – Supports encapsulation Other – Reusable, Scalable
life_path.py	Only Maintains which need to determine user's Life Path. Allows to initiate anywhere with data like date. Output – life path as Number Single responsibility – Only handling Life path processing. OOP – Supports encapsulation Other – Reusable, Scalable
luck_colors.py	Only Maintains which need to determine user's luck color. Allows to initiate anywhere with life path number. Output – Color as string Single responsibility – Only handling Color processing. OOP – Supports encapsulation Other – Reusable, Scalable
master_number.py	Only Maintains which need to determine master number. Output – master number as number Single responsibility – Only handling Master number. OOP – Supports encapsulation Other – Reusable, Scalable

Figure 8: Class Description

Interaction with users and intermodular communication are managed by the main module. It features strong testing and error-handling capabilities and makes use of shutdown mechanisms. The sole responsibility concept is adhered to while computing and comparing Life Path Numbers in the Life Path Number Calculation Module.

Applications of OOP

A software business is creating tools for examination of numerology. There are two cases that are examined: finding the Lucky Color, figuring out the Life Path Number, and figuring out whether it's a master number. In addition, the program identifies the person's generation by comparing two birthdays. The project follows the guidelines of object-oriented programming and is organized into

many classes. Based on user input, the primary class manages user interaction and coordinates calls to other classes. It offers tools to compare Life Path Numbers across dates, find master numbers, find Lucky Colors, calculate Life Path Numbers, and categorize generations. Maintainability, extensibility, and modularity are encouraged in this architecture.

Design Choice	Benefits
Include interface interactions related settings in main.py	Independent from changing codes. Scalable Allowed to maintain concise code architecture.
Includes Features in separate classed	Maintainability of code Reusability of code Easiness of feature separation and version controlling Maintain single responsibility
Following OOP	Allowed to employ characteristics like encapsulation and enhance code

Table 1: Design Choices and Benefits

Black Box Testing

Test Case	Test Input	Output	Expected Output
Life_path.py			
BB_C01	1990-12-17	3	Standard date input
BB_C02	2000-01-01	4	Start of the new millennium
BB_C03	1963-07-11	11	Birth date resulting in master number 11
BB_C04	1988-08-08	22	Birth date resulting in master number 22

BB_C05	1977-07-07	33	Birth date resulting in master number 33
BB_C06	2025-01-01	Invalid Date	Date out of valid range (year 2025)
BB_C07	1899-12-31	Invalid Date	Date out of valid range (year 1899)
luck_colour.py			
BB_C08	1	Red	Standard Life path number
BB_C09	11	Silver	Master Number 11
BB_C10	33	White	Master Number 33
BB_C11	0	Unknown	Invalid Life Path Number (zero)
BB_C12	-1	Unknown	Invalid Life Path Number (negative)
BB_C13	12	Unknown	Invalid Life Path Number (out of range)
Master_number.py			
BB_C14	11	True	Standard Master Number 11
BB_C15	22	True	Standard Master Number 22

BB_C16	10	False	Not Master number
BB_C17	0	False	Zero as input
BB_C18	-11	False	Negative number
generation.py			
BB_C19	1925	Silent Generation	Mid-range year for Silent Generation
BB_C20	1945	Silent Generation	Last year of Silent Generation
BB_C21	1965	Generation X	First year of Generation X
BB_C22	2000	Generation Z	Mid-range year for Generation Z
BB_C23	2025	Unknown Generation	Year out of valid range (too late)

4

Table 2: Black-Box Testing

White-Box Test Cases

Test Case	Test Input	Output	Expected Output
WB_C01	1963-07-11	True	Standard date input to test master number detection

WB_C02	2000/01/01	False	Incorrect Date Format due to slashes
WB_C03	2001-02-29	False	Tests invalid date (non-leap year)
WB_C04	2026-07-30	False	Date out of range
	1901-01-01	True	Earliest valid date within specified range

Table 3: White-Box Testing

Test Implementation and Execution

The Python package units are used in this project to ensure the reliability and precision of the Numerology and Generation classes. The tests, which focus on input and output without considering the internal code structure, cover both black-box and white-box methodologies. Input Validation, Master Number Identification, Lucky Color Identification, Generation Classification, and Life Path Number Calculation are among the tests. White-box testing cover a wide range of internal logic and code routes, whereas black-box tests are more focused on input and output. correct master number identification, correct calculation, and proper generation classification are guaranteed by the tests. The tests include border conditions and edge situations.

```

1  import unittest
2  from master_number import MasterNumber
3  from generation import Generation
4  from validation import Validation
5
6  class TestNumerologyApp(unittest.TestCase):
7
8      def setUp(self):
9          self.numerology = MasterNumber()
10         self.generation = Generation()
11         self.validator = Validation()
12
13     # Black-Box Tests
14     def test_life_path_number_valid(self):
15         self.assertEqual(self.numerology.calculate_life_path_number("1990-07-25"), 6)
16         self.assertEqual(self.numerology.calculate_life_path_number("2000-02-29"), 5)
17
18     def test_life_path_number_invalid(self):
19         with self.assertRaises(ValueError):
20             self.numerology.calculate_life_path_number("2025-07-25")
21         with self.assertRaises(ValueError):
22             self.numerology.calculate_life_path_number("1990-13-25")
23
24     def test_master_number(self):
25         self.assertTrue(self.numerology.is_master_number(11))
26         self.assertFalse(self.numerology.is_master_number(10))
27
28     def test_lucky_colour(self):
29         self.assertEqual(self.numerology.get_lucky_colour(1), "Red")
30         self.assertEqual(self.numerology.get_lucky_colour(6), "Green")
31
32     def test_generation(self):
33         self.assertEqual(self.generation.get_generation("1990-07-25"), "Millennials")
34         self.assertEqual(self.generation.get_generation("1975-04-20"), "Generation X")
35
36     # White-Box Tests
37     def test_date_validation_valid(self):
38         self.assertTrue(self.validator.validate_date("1990-07-25"))
39         self.assertTrue(self.validator.validate_date("2000-02-29"))

```

Figure 9: Test Class

The setup method initializes instances before each test, while the unittest.main() function runs all test cases, providing detailed output on test status, including pass, fail, or error, along with descriptions.

Summary

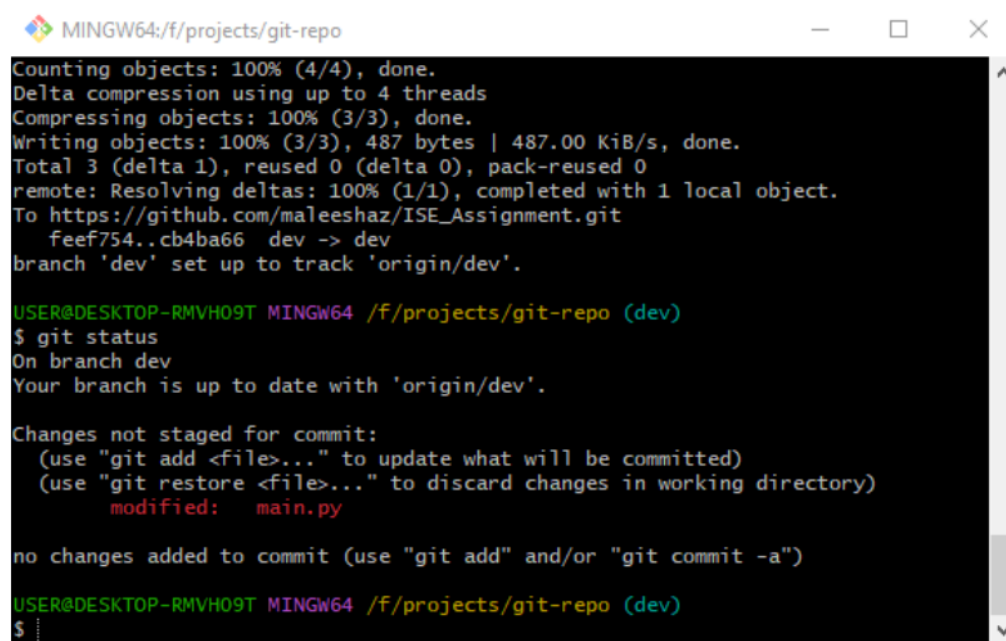
Using Python, I created a thorough numerical analysis tool for this project, paying close attention to conditions given in the scenario as client's requirement to guarantee scalability and maintainability of the tool (Luck Number Generator). The program has modules for lineage classification, fortunate color identification, numerology number computation, and date confirmation. Because each module is intended to address a particular facet of statistical analysis, issues and personal accountability are kept well apart. One of the project's problems was merging several modules into a single user interface and managing crucial quantities in computations appropriately. The application was successfully tested and implemented in spite of these difficulties, proving its resilience and functionality. Future improvements might include better numerology readings, support for other languages and date formats, and the use of machine learning techniques for tailored insights. The project has been a tremendous learning opportunity, highlighting the value of user-centered development, error management, and modular design in the creation of software solutions.

Python Script	Black Box	White Box	Form of Input	Form of Output	Data Type
main.py	Done	Done	User Inputs (Number, String)	Results (Number, String)	-
generation.py	Done	Done	Year	Generation	String Enum
life_path.py	Done	Done	Date	Life Path number	Number
luck_colors.py	Done	Done	Life Path Number	Color	String
master_number.py	Done	Not Done	Life Path Number	Is Master number	Boolean

Figure 10: Work Flow Table

Version Controlling

By utilizing Git to establish a version control system, the development could make sure that the software project's progress was managed and that all of the documentation and code were kept under observation. Because of the folder structure included into the repository, code and documents can be stored in different folders. This will support maintaining the structure's neatness and organization.



```
MINGW64:/f/projects/git-repo
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 487 bytes | 487.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/maleeshaz/ISE_Assignment.git
   feef754..cb4ba66 dev -> dev
branch 'dev' set up to track 'origin/dev'.

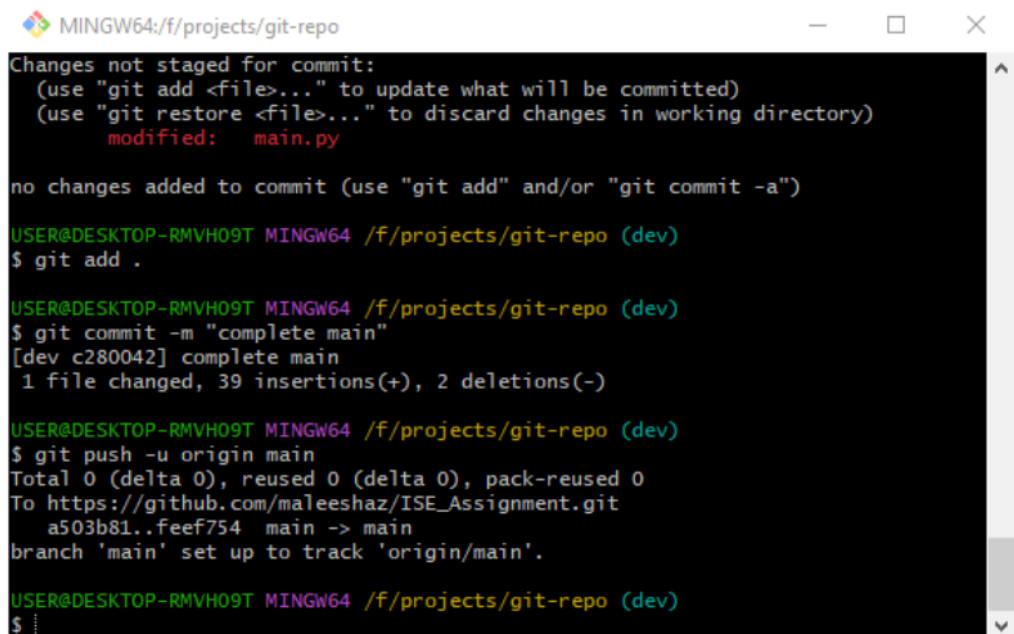
USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (dev)
$ git status
On branch dev
Your branch is up to date with 'origin/dev'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.py

no changes added to commit (use "git add" and/or "git commit -a")

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (dev)
$
```

Figure 11: Dev Branch



```
MINGW64:/f/projects/git-repo
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.py

no changes added to commit (use "git add" and/or "git commit -a")

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (dev)
$ git add .

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (dev)
$ git commit -m "complete main"
[dev c280042] complete main
1 file changed, 39 insertions(+), 2 deletions(-)

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (dev)
$ git push -u origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/maleeshaz/ISE_Assignment.git
   a503b81..feef754  main -> main
branch 'main' set up to track 'origin/main'.

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (dev)
$
```

Figure 12: Merging Main branch

```
MINGW64:/f/projects/git-repo
[dev c280042] complete main
1 file changed, 39 insertions(+), 2 deletions(-)

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (dev)
$ git push -u origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/maleeshaz/ISE_Assignment.git
a503b81..feef754 main -> main
branch 'main' set up to track 'origin/main'.

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (dev)
$ git add .

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (dev)
$ git checkout -b test
Switched to a new branch 'test'

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (test)
$ git commit -m "Test cases"
[test f1bd8ba] Test cases
8 files changed, 237 insertions(+), 1 deletion(-)
create mode 100644 YourFirstNameYourLastName_your student ID_ISEReport.docx
create mode 100644 tests/test.py
create mode 100644 tests/test_generation.py
create mode 100644 tests/test_life_path.py
create mode 100644 tests/test_lucky_colors.py
create mode 100644 tests/test_master_number.py
create mode 100644 tests/test_validation.py

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (test)
$ git push -u origin test
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 4 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 699.39 KiB | 18.90 MiB/s, done.
Total 14 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'test' on GitHub by visiting:
remote:   https://github.com/maleeshaz/ISE_Assignment/pull/new/test
remote:
To https://github.com/maleeshaz/ISE_Assignment.git
* [new branch] test -> test
branch 'test' set up to track 'origin/test'.

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (test)
$ git status
On branch test
Your branch is up to date with 'origin/test'.

nothing to commit, working tree clean

USER@DESKTOP-RMVH09T MINGW64 /f/projects/git-repo (test)
$
```

Figure 13: Git pushing changes

Discussion

Using Python 3, the underlying development was effectively build as a numerical analysis tool for this project, resulting in a useful and intuitive interface interact. The main accomplishment was the merging of many modules into a single, cohesive system in a scalable and manageable manner, including lineage classification, fortunate color identification, numerology computations, data validation and date confirmation. Ensuring correct numerology computations, particularly when dealing prime numbers, required careful logic implementation, which was one of the major issues encountered.

The present tool has limitations, such as limited date format compatibility and no support for languages other than English. Furthermore, the program only offers a rudimentary comprehension of numerology at this time, which could not please customers who need a more in-depth examination. Future upgrades could include extending numerology readings with more in-depth insights and supporting many languages.

Challenges

- Complex input processing, modularity, precise numerology computations, generation categorization, thorough testing, interface handling, version control management, scalability, incorrect data handling, and thorough documentation are all required for this project.
- It necessitates managing a variety of date ranges and formats in order to guarantee seamless module integration and communication. Precise mathematical and logical procedures are needed to calculate Life Path Numbers, master numbers, and lucky color calculations accurately. To guarantee robustness and dependability, the project also needs extensive testing, covering scenarios with both black-box and white-box operations.
- User interaction is handled by the main module (main.py), and effective teamwork requires version control management using Git. The tool's code is clear, modular, and thoroughly documented, and it is built for further features and enhancements in the future.

Limitations

The numerology tool has several drawbacks, such as a restricted date range of 1901–2024, only support for English as a language, simplified methods for determining Lucky Colors and Life Path Numbers, a simple user interface, no data persistence, a restricted number of input formats, no real-time feedback, no graphical results representation, a lack of personalization, and no machine learning integration. Future projections, historical data analysis, and user satisfaction may all be impacted by these limitations. The tool's minimal user interface (UI) could be perplexing to non-English speakers, and frequent users might find its lack of data permanence bothersome. Furthermore, visually challenged learners may find it difficult to comprehend the results due to the tool's lack of graphical representation.

Future Improvements

The application's capabilities may be further enhanced by using machine learning algorithms and integrating a database to store user data for customized readings, which can produce more accurate and personalized forecasts by employing user data. This needs to cooperate with concepts like data mining, and data science to be effective and valued. Other features of the application include enhanced numerology analysis, personalized reports, adjustable date formats, and multilingual support. It provides thorough readings for each Life Path Number, personalized reports, and multilingual assistance. Additionally, the tool has an easy-to-use graphical user interface and a web version for internet access.

.

ORIGINALITY REPORT

6%

SIMILARITY INDEX

1%

INTERNET SOURCES

0%

PUBLICATIONS

5%

STUDENT PAPERS

PRIMARY SOURCES

1

Submitted to Curtin University of Technology

Student Paper

3%

2

Submitted to University of Northumbria at
Newcastle

Student Paper

1%

3

Submitted to University of Bedfordshire

Student Paper

1%

4

openseminar.org

Internet Source

1%

5

www.fdle.state.fl.us

Internet Source

<1%

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off