# 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 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

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.

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.

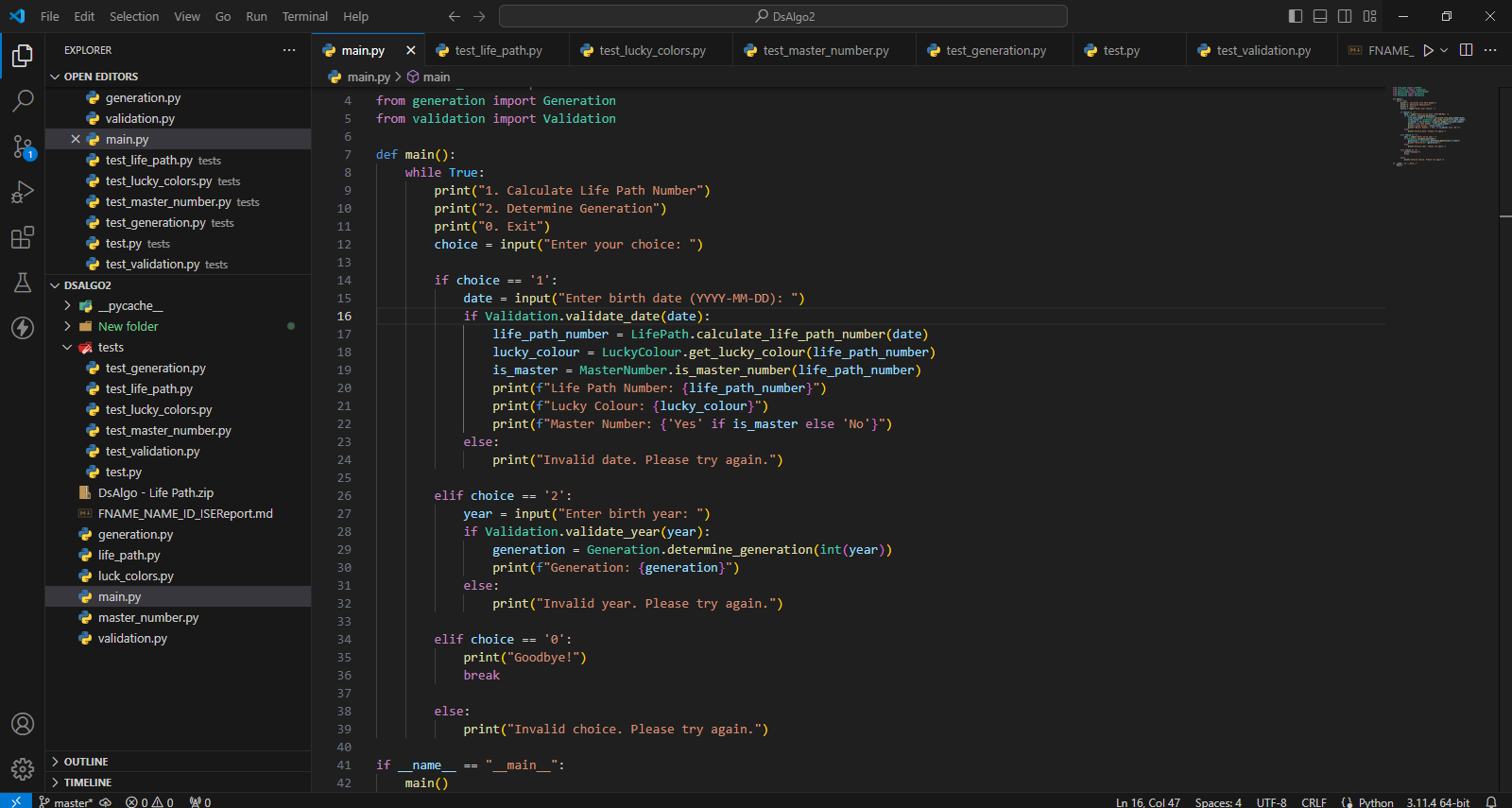


Figure : Main Class

## 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.

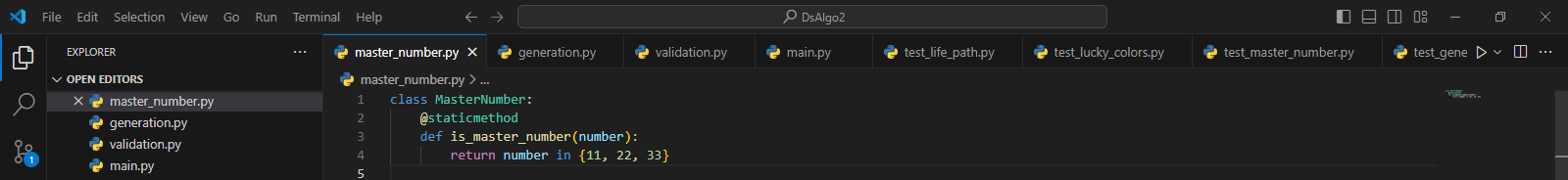


Figure : Master Number class

## 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.

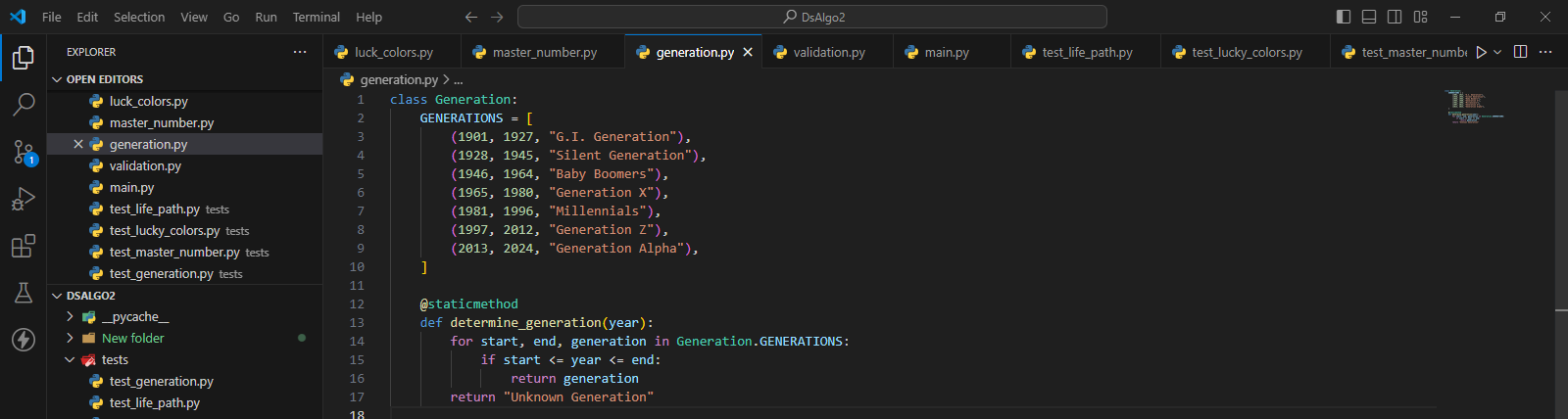


Figure : Generation Class

## 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.

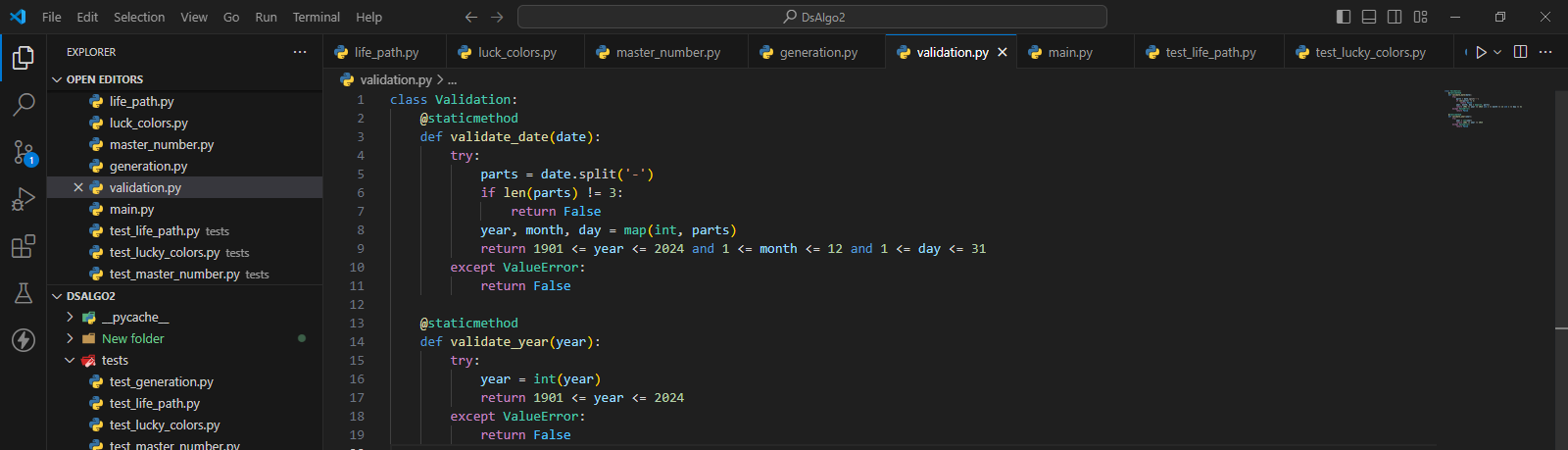


Figure : Validation class

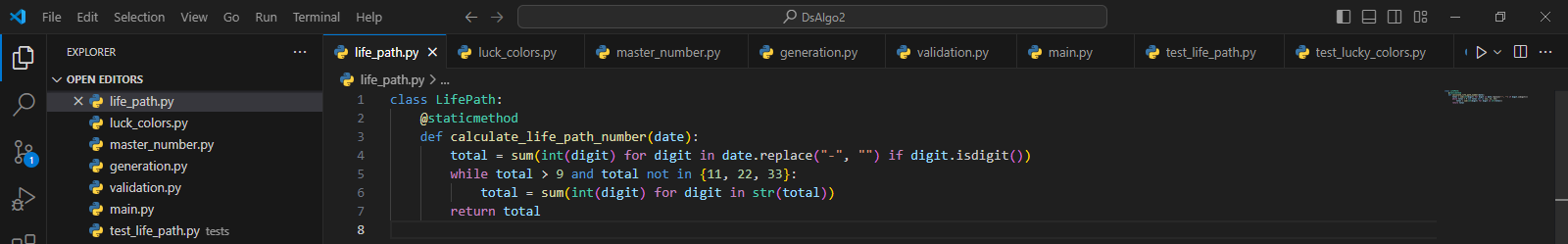


Figure : Life Path class

## 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., luckycolour, 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.

## 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 : 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) |
|  |  |  |  |

Table : 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 |

Table : White-Box Testing

# Test Implementation and Execution

In order to guarantee the accuracy and dependability of the Numerology and Generation classes, this project makes use of the Python package units. Both black-box and white-box techniques are covered in the tests, which concentrate on input and output without taking the internal code structure into account. The tests include Generation Classification, Input Validation, Master Number Identification, Lucky Color Identification, and Life Path Number Calculation. While white-box tests cover a variety of code pathways and internal logic, black-box tests concentrate on input and output. The tests guarantee accurate generation categorization, master number identification, and computation accuracy. Edge cases and boundary circumstances are covered by the tests.

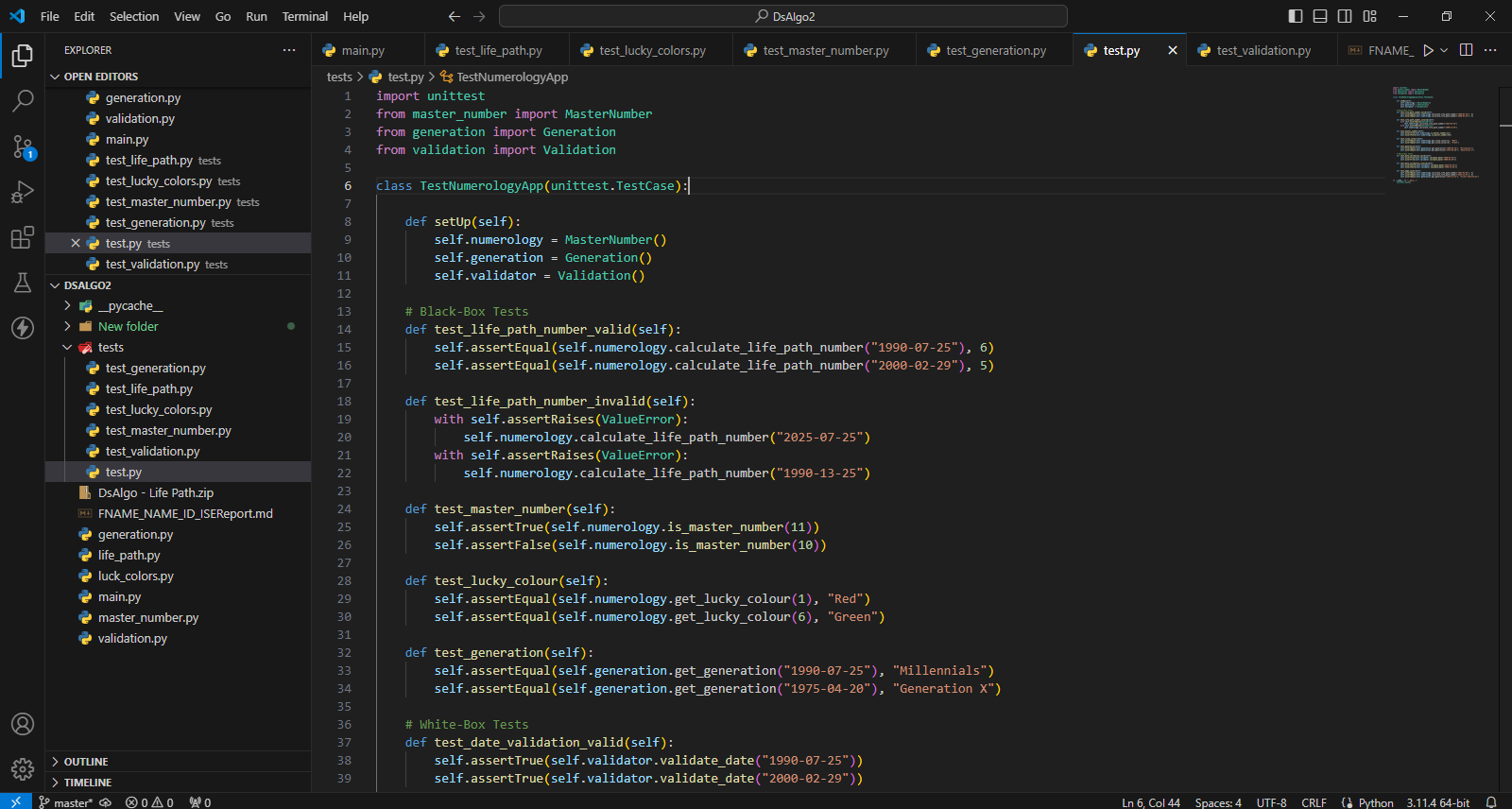


Figure : Test Class

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.

# 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.

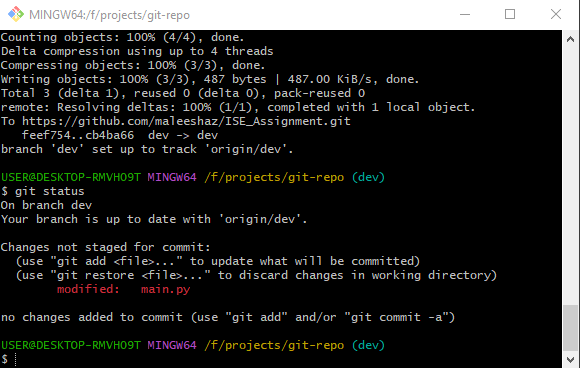


Figure : Dev Branch

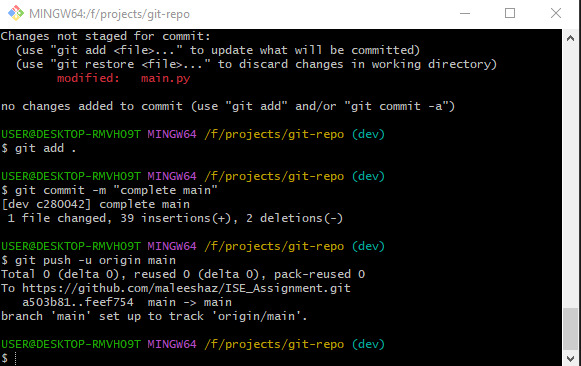


Figure : Merging Main branch

# Discussion

Using Python 3, the underlaying 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.

Incorporating machine learning algorithms and integrating a database to store user data for customized readings can yield more precise and customized forecasts by utilizing user data which will improve the application's capabilities further.