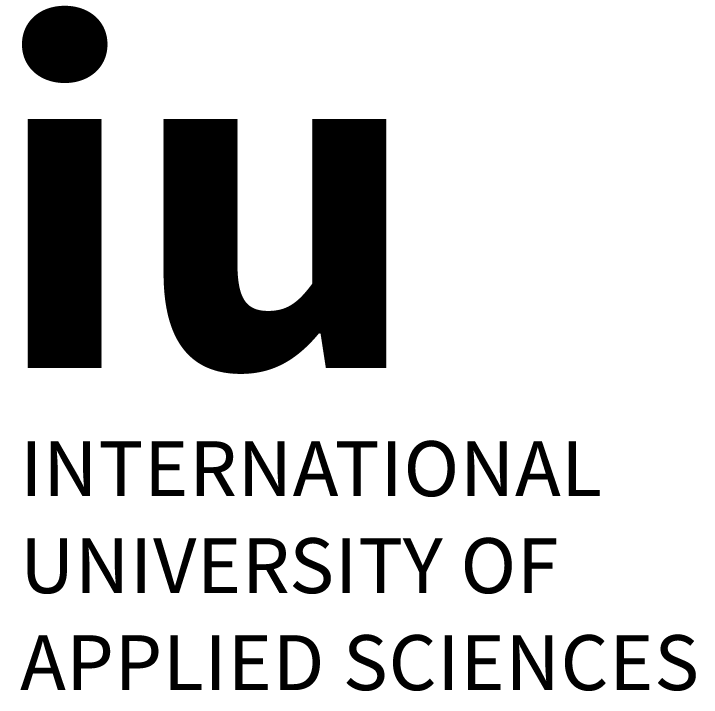
**Programming With Python (DLMDSPWP01)**

**

*A Research Essay on*

**Python Programming**

* Author’s Name: **Fahimeh Bahman**
* Matriculation Number: **4243280**
* Course of Study: **MSDS**
* Tutor’s Name: **Amna Khan**
* Date: **30/04/2024**

**Contents**

**1.Introduction………………………………………………………………………………….………...1**

**1.1.History Of Python…………………………………………………………………………….…….1**

**2.Python Programming………………………………………………………………………….……..2**

**2.1.Data Analytics……………………………………………………………………………….………2**

**2.2.Machine Learning and AI………………………………………………………………………….3**

**2.3.Database Work………………………………………………………………………………………3**

**2.4.Python for Animation………………………………………………………………………………3**

**2.5.Games Programming………………………………………………………………………………4**

**2.6.Integration Testing Framework…………………………………………………………………..4**

**2.7.Academic Research……………….………………………………………………………….....…5**

**2.8.Web Services ………………………………………………………………………………………..5**

**2.9.DevOps………………………………………………………………………………………….……5**

**3.Introducing object-oriented…………………………………………………………………….…..5**

**3.1.Objects and classes…………………………………………………………………………….…6**

**4.The 4 Pillars concept of OOP in Python……………………………………………………….…6**

**4.1.Encapsulation………………………………………………………………………………………7**

**4.2.Inheritance…………………………………………………………………………………………..7**

**4.3.Abstraction………………………………………………………………………………………….8**

**4.4.Polymorphism…………………………..…….………………………………………………....…8**

**5.Version Control……………………………………………………………………………………….9**

**5.1.What is Version Control and Why Use It?..........................................................................9**

**5.2.Why version control matters in software development…………………………………..…9**

**5.3.Types of version control………………………………………………………….……………...10**

**5.3.1.Local version control………………………………………………………………………..…10**

**5.3.2.Version management centralization…………………………………………………….......10**

**5.3.3.Distributed version control………………………………………………………………....…11**

**5.4.Advantages of Distributed Version Control Systems………………………………………12**

**5.5.Disadvantages of Distributed Version Control Systems………………………...………...12**

**6.Version Control with Git………………………………………….……………………….………..13**

**6.1.Main VC system concepts…………………………………………………………………...…..13**

**7.Conclusion……………………………………………………………………………………………15**

**8.Bibliography…………………………………………………………………………………….....…15**

**9.Task Assignment…………………………………………………………………………………….16**

**10.List of Appendices………………………...………………………………………...………….…16**

**11.Appendices……………………………………………………………………………..……...……16**

**List of Figures:**

|  |  |  |  |
| --- | --- | --- | --- |
| SL No | Name of the figure | Figure No | **Page No** |
| 1 | Data Analytics | 1 | 2 |
| 2 | Machine Learning and AI | 2 | 3 |
| 3 | Objects and classes | 3 | 6 |
| 4 | Abstraction | 4 | 8 |
| 5 | Centralized Version Control | 5 | 11 |
| 6 | Distributed version control | 6 | 11 |

**Table of Abbreviations**

|  |  |  |
| --- | --- | --- |
| SL No | Short Forms | Full Forms |
| 1 | OOP | Object-oriented programming |
| 2 | GUI | Graphical User Interfaces |
| 3 | AI | Artificial Intelligence |
| 4 | DB | Database |

**1. Introduction**

Python is a general-purpose programming language in a similar vein to other programming languages that you might have heard of such as C++, JavaScript, or Microsoft’s C# and Java. It has been around for some considerable time having been originally conceived back in the late 1980s by Guido van Rossum at Centrum Wiskunde and Informatica (CWI) in the Netherlands. The language is named after one of Guido’s favorite programs “Monty Pythons Flying Circus”, a classic and somewhat anarchic British comedy sketch show originally running from 1969 to 1974 (but which has been rerun on various stations ever since) and with several film spin-offs. This increased interest in Python is driven by several different factors:

1. Flexibility and simplicity, hence easy to learn.
2. By the Data Science community, this provides a more standard programming language than some rivals like R.
3. As a scripting language by those working in the DevOps field where this provides a greater abstraction level than the alternative traditionally used language.
4. It can be run on, almost, any operating system but, in particular on the big three operating systems Windows, MacOS, and Linux.
5. It offers a large number of libraries, modules, that can be used to extend the basic features of the language.
6. It's free! (Hunt,2023)

**2. Python Programming**

There are several different programming paradigms that a programming language may allow developers to code in, these are:

● Procedural Programming in which a program is represented as a sequence of instructions that tell the computer what it should do explicitly. Procedures and / or functions are used to provide structure to the program with control structures such as if statements and loop constructs to manage which steps are executed and how many times. Languages typifying this approach include C and Pascal.

● Declarative Programming languages, such as Prolog, that allow developers to describe how a problem should be solved with the language/environment determining how the solution should be implemented. SQL (a database query language) is one of the most common declarative languages that you are likely to encounter and is introduced at the end of this book.

● Object-oriented programming approaches that represent a system in terms of the objects that form that system. Each object can hold its data (also known as a state) as well as define behavior that the object can do. A computer program is formed from a set of these objects cooperating. Languages such as Java and C# typify the object-oriented approach.

● Functional Programming languages decompose a problem into a set of functions. Each function is independent of any external state operating only on the inputs they received to generate their outputs. The programming language Haskell is an example of a functional programming language. (Hunt,2023)

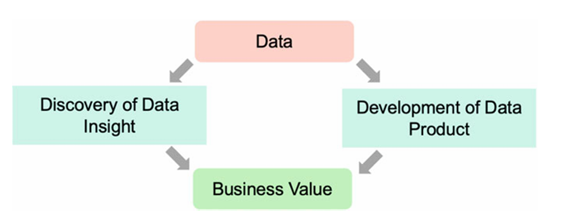
**2.1. Where is Python Used?**

we will look at eight uses of Python, some of which might surprise you.

**2.1.1. Data Analytics**

There are several widely used Data Analytics libraries available for Python including Pandas. Data Analytics (or analysis) is the practice of taking raw data and analyzing it to either understand that data and/or look for patterns in the data or to create new systems that can use the knowledge gained from previous data to predict new trends or patterns in future data. In both cases, the aim is to provide value in some way to an organization or business as illustrated below.

**Figure1: Data Analytics**



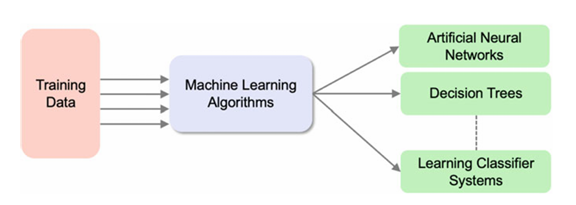
Source: (Hunt,2023)

The approaches to Data Analytics are not completely independent but illustrate the different purposes. For example, the “Discovery of Data Insight” side typically involves understanding and trends, patterns or relationships in the data to help an organization or business to improve their performance. For example, an online sales shop might use such insights to understand how and where to target specific promotions, a governmental health organization might use its data insights to determine where and when to target vaccinations, etc.

**2.1.2. Machine Learning and AI**

Following on from the previous section on data analysts it is not uncommon to apply some machine learning techniques when analyzing data, for example, to perform some classification of new data using previously seen data. That is, machine learning is a method of data analysis that automates analytical model building. Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed. The general model is shown below.

**Figure2: Machine Learning and AI**



Source: (Hunt,2023)

**2.1.3. Database Work**

Most data is held within some form of database, and Python provides interfaces that allow a programmer to access that data. These interfaces make it easy to query a database for information and pull that information back into the Python program for further progress. For example, a database of client information may be queried 2.1 Introduction 19 to retrieve details held about clients. Python could then display this information in a tabular form via some form of a table, graphically in some form of a chart, or on a map, etc.

**2.1.4. Python for Animation**

Python is one of the two programming languages that can be used in the Autodesk Maya toolkit. Maya is a 3D animation, modeling, simulation, and rendering software system used by games, film, and TV companies. Within Maya, Python can be used for anything from creating scripts handling common tasks, through to develop complete plug-ins providing extensions to the core functionality of the tool. Maya currently supports Python 2 but will be moving toward Python 3 in the near future. 2.1.5 Python for Film Making Industrial Light and Magic was formed in 1975 by George Lucas to help create special effects for the original Star Wars film. Since then it has gone on to provide special effects for a huge range of films and filmmakers. Back in 1996, it used Unix shell scripts to help it automate and control its production pipeline. However, they needed to develop their increasingly complex and computationally expensive production process and felt that their existing approach was not flexible enough. Although they looked at several alternatives (including TCL and Perl) they decided to adopt Python. This allowed for faster development times and greater flexibility. Over the years they have re-evaluated their decision and considered different alternatives, but Python remains key to their development processes. 2.1.6 Cross Platform UIs Python is particularly well served in terms of Graphical User Interface frameworks (or GUI frameworks). Most of these are cross platform, although a few allow developers to exploit particular features of an underlying windowing system and are thus tied to a specific platform. Widely used GUI libraries include the cross platform TKinter, wxPython and pyQT and the platform specific Python Win (for Windows) and PyObjc (for MacOS). As an example, the following ‘Hello World’ GUI application uses wxPython and is shown running on a Mac OS and Windows system.

**2.1.5. Games Programming**

Python has also been used for games development with several well-known games titles depending on it in one way or another. For example, Battlefield 2 by Digital Illusions CE is a military simulator first-person shooter game. Within this game, Python is used to handle portions of the game logic for Battlefield Heroes involving game modes and scoring. Other games that use Python include Civilization IV, Pirates of the Caribbean Online, and Overwatch. Pygame is probably the best library utilized within the world of Python for game development. There are many extensions available to pygame, too, which help in making a large number of different types of game genres.. Pygame is built on top of the SDL library (or Simple Directmedia Layer). SDL is a cross-platform development library designed to provide access to audio, keyboards, mouse, joystick, and graphics hardware via OpenGL and Direct3D. To promote portability, pygame also supports a variety of additional backends including WinDIB, X11, Linux Frame Buffer, etc.

**2.1.6. Integration Testing Framework**

Morgan Stanley have developed a Python integration testing framework called Test plan which they have open sourced and made available through GitHub. Testplan is designed to simplify the process of configuring and driving integration tests for a range of programming languages and technologies. It supports driving integration tests that require messaging services, RESTful services, databases, files for code written in Python, C/C++, Java, etc.

**2.1.7.** **Academic Research**

Python is widely used within the academic community to support research work, not only within Computer Science departments but also across a range of different disciplines including Mechanical Engineering, Aerospace Engineering, Architecture, Pharmacology, Medicine, etc. In these settings it has been used to help develop distributed analysis systems, identify patterns in experiments of laboratory data, 2.2 Useful Resources 21 evolve designs toward optimal solutions, provide natural language frontends to research applications (e.g., using the Python NLTK library), etc.

**2.1.8. Web Services**

Python is also widely used as a server-side language for the creation of web services, whether these are RESTful services or the newer GraphQL-based services. There are a range of frameworks available to help develop such services including Flask, Django, and CherryPy. Flask and CherryPy are lightweight frameworks that can be used to create RESTful services, whereas Django is a full-stack web framework aimed at developing not just web services but full blown websites. A wide range of organizations used Python in this way for example Reddit, Spotify, and Instagram.

**2.1.9.DevOps**

DevOps is another hot trend at the moment; it represents the combination of software developers and operations working together usually to automate operations processes that were previously either handled manually or were executed as individual steps. Python is one of the key programming languages used within the DevOps world. It can be used as a scripting language to help automate operations activities as well as a tool to analyze production data and for data visualizations. (Hunt, 2023)

**3. Introducing object-oriented**

Everyone knows what an object is: a tangible thing that we can sense, feel, and manipulate. The earliest objects we interact with are typically baby toys. Wooden blocks, plastic shapes, and over-sized puzzle pieces are common first objects. Babies learn quickly that certain objects do certain things: bells ring, buttons are pressed, and levers are pulled. The definition of an object in software development is not terribly different. Software objects may not be tangible things that you can pick up, sense, or feel, but they are models of something that can do certain things and have certain things done to them. Formally, an object is a collection of data and associated behaviors. Considering what an object is, what does it mean to be object-oriented? In the dictionary, oriented means directed toward. Object-oriented programming means writing code directed toward modeling objects. This is one of many techniques used for describing the actions of complex systems. (Lott, Phillips,2021)

**3.1. Objects and classes**

It's important to understand the difference between an object and a class. Classes describe objects. They are like blueprints for creating an object. You might have three oranges sitting on the table in front of you. Each orange is a distinct object, but all three have the attributes and behaviors associated with one class: the general class of oranges. Objects are instances of classes that can be associated with each other’s. An object instance is a specific object with its own set of data and behaviors; a specific orange on the table in front of us is said to be an instance of the general class of oranges. That's simple enough, but let's dive into the meaning of those two words, data and behaviors.

**Figure3: Objects and classes**

A diagram of different types of fruit

Description automatically generated

Source: (Lott, Phillips,2021)

**3.2. The 4 Pillars concept of OOP in Python**

Abstraction, Encapsulation, Inheritance, and Polymorphism are the "four pillars" of Object-Oriented Program design, and through this document, we will take a brief look at each using Python to explain how they relate to my wishful house.

**3.2.1. Encapsulation**

This process of hiding the implementation of an object is suitably called information hiding. It is also sometimes referred to as encapsulation, but encapsulation is actually a more all-encompassing term. Encapsulated data is not necessarily hidden. Encapsulation is, literally, creating a capsule (think of creating a time capsule). If you put a bunch of information into a time capsule and lock and bury it, it is both encapsulated and the information is hidden. A common real-world example is the television. Our interface to the television is the remote control. Each button on the remote control represents a method that can be called on the television object. When we, as the calling object, access these methods, we do not know or care if the television is getting its signal from a cable connection, a satellite dish, or an internet-enabled device. We don't care what electronic signals are being sent to adjust the volume, or whether the sound is destined for speakers or headphones. If we open the television to access the internal workings, for example, to split the output signal to both external speakers and a set of headphones, we will void the warranty.

**3.2.2. Inheritance**

This relation created through Inheritance comes probably as the most famous, best-known, and abused relationship in Object Oriented Programming. Inheritance is kinda like one's family tree. My grandfather's last name was Phillips and my father inherited that name. Then I in turn inherited it from him. In Object Oriented Programming, instead of the features and behavior of a person, one class inherits the attributes and methods of another class.

For example, in our chess set, we have 32 chess pieces, but there are only six different kinds of pieces: pawns, rooks, bishops, knights, kings, and queens, each of which moves differently. All the child classes automatically have a chess set and color attribute by inheritance from the base class. Each piece provides a different shape property to be used in drawing the piece onto the screen when the board is rendered, and a different move method to move the piece to a new position on the board at each turn. It will take only a little work to resolve this. We simply need to add a basic dummy move method to the Piece class. The subclasses then override that with a more specific implementation.

The ability to override methods in subclasses enables very powerful object-oriented systems to be developed. It means that, for example, should we ever want to implement a class for a computer player with artificial Intelligence could be achieved by providing a method calculate move which takes a Board object and decides which piece to move where.

Indeed, one can design a class that does not implement at all any methods. In this case, such a class would only bring us the information of what the class should do, but provides no advice whatsoever about how it is to be done. In object-oriented terminology, classes like those are called interfaces.

* + 1. **Abstraction**

It is another object-oriented term related to encapsulation and information hiding. Abstraction means dealing with the level of detail that is most appropriate to agiven task. It is the process of extracting a public interface from the inner details. A car's driver needs to interact with the steering, accelerator, and brakes. The workings of the motor, drive train, and brake subsystem don't matter to the driver. A mechanic, on the other hand, works at a different level of abstraction, tuning the engine and bleeding the brakes. Here's an example of two abstraction levels for a car:

**Figure4: Abstraction**

A diagram of a car

Description automatically generated

Source: (Lott, Phillips,2021)

abstraction is the process of encapsulating information with separate public and private interfaces. The private interfaces can be subject to information hiding.

* + 1. **Polymorphism**

Let's explore the longest word in object-oriented argot. Polymorphism is the ability to treat a class differently, depending on which subclass is implemented. Polymorphism is pretty cool, but it is a word that is rarely used in Python programming. Python goes an extra step past allowing a subclass of an object to be treated like a parent class. This sort of polymorphism in Python is typically referred to as duck typing: if it walks like a duck or swims like a duck, it's a duck. We don't care if it really is a duck (is a cornerstone of inheritance), only that it swims or walks. Geese and swans might easily be able to provide the duck-like behavior we are looking for. This allows future designers to create new types of birds without actually specifying an inheritance hierarchy for aquatic birds. It also allows them to create completely different drop-in behaviors that the original designers never planned for. For example, future designers might be able to make a walking, swimming penguin that works with the same interface without ever suggesting that penguins are ducks.

**4. Version Control**

Version control, versioning, or source control is any process used to track changes in the source code. It simply keeps a record of every single edit made in the code and thus enables tracking and reversibility of changes. A crucial tool in every developer’s kit, version control systems are the bedrock of this process. (Rao, 2023)

**4.1. What is Version Control and Why Use It?**

At a basic level version control involves taking ‘snapshots’ of files at different stages. Version control tries to address problems like these by implementing a systematic approach to recording and managing changes in files. At its simplest, version control involves taking ‘snapshots’ of your file at different stages. This snapshot records information about when the snapshot was made but also about what changes occurred between different snapshots. This allows you to ‘rewind’ your file to an older version. From this basic aim of version control, a range of other possibilities are made available. (Strien, 2016)

**4.2. Why version control matters in software development**

Basically, software development is intrinsically version control and project management. Right at the core of this tracking is changes in a repository for source code versioning. Let us outline three of the most important reasons why version control is very important to both the software developer and the product/project manager. These include:

1. [**Streamlined release management**](https://blog.logrocket.com/product-management/version-control-systems-definition-types/#streamlinedreleasemanagement)

Version control is important for release management. It has versions of releases of software. Releases usually include a set of changes/ enhancements/ features developed for various customers as per the release roadmap.

1. [**Confliction prevention**](https://blog.logrocket.com/product-management/version-control-systems-definition-types/#conflictprevention)

## This helps in avoiding collisions on the source code base. Several branches are maintained for different releases and so replication of changes by overlap leads to potential conflicts.

1. [**Tracking changes to digital artifacts**](https://blog.logrocket.com/product-management/version-control-systems-definition-types/#trackingchangestodigitalartifacts)

Apart from the source code, version control does not break a sweat when tracing all the changes made to other digital artifacts in the software development process. This could include technical design specifications, requirement documents, and any kind of deliverable susceptible to being taken through a number of iterations. (Rao, 2023)

## 4.3. Types of version control

Modifications exist in three prime forms of version control systems, each with advantages and disadvantages: (Rao, 2023)

**4.3.1. Local version control**

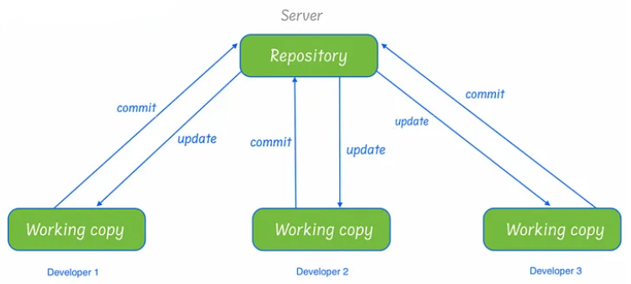
It stores changes in local files as a hotfix or patch before pushing it to a single version of code in a database. It becomes difficult to retrieve updates in case of corruption of local versions or the single code version is corrupted.

**4.3.2. Version management centralization**

In centralized version control, various versions of the code are stored in a central repository. For instance, after accessing such versions, a user can pull or push modifications to his machine.

It is, however, difficult to recover if the central store becomes corrupted.

**Figure5: centralized Version Control**



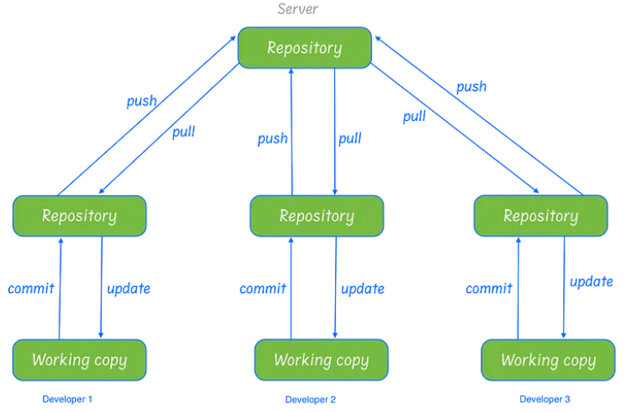
Source: (Rawat. A, 2022**)**

### **4.3.3. Distributed version control**

Distributed version control is the most sophisticated of the three. Every local repository in this case maps the history of the central repository exactly, just as the central repository.

Therefore, it shouldn't be a major concern if the central repository becomes corrupted. Any local servers can send a copy of the history and repository to other local servers or back to the central server.

**Figure6: Distributed version control**



Source: (Rawat. A, 2022**)**

**4.4. Advantages of Distributed Version Control Systems**

* Except for pushing and pulling the code, the user can work offline in DVCS, due to this the full history is always available.
* DVCS is faster than CVCS because you don’t need to communicate with the remote server for each and every command.
* Merging and branching the changes in DVCS is very easy.
* No need to access a remote server therefore the performance of DVCS is better.
* If the main server goes down or it crashes in DVCS, you can still get the backup or entire history of the code from your local repository or server where the full revision of the code is already saved.
* Because of local commits, the full history is always available.
* Ability to push your changes continuously
* Good for projects with off-shore developers

**4.5. Disadvantages of Distributed Version Control Systems:**

* Working with a lot of binary files requires a huge amount of space, and developers can’t do diffs.
* Projects with a long history, i.e., a large number of changesets may take a lot of time and occupy more disk space.
* It may not always be obvious who did the most recent change
* File locking doesn’t allow different developers to work on the same piece of code simultaneously. It helps to avoid merge conflicts but slows down development.
* DVCS enables you to clone the repository — this could mean a security issue
* Managing non-mergeable files is contrary to the DVCS concept

**5. Version Control with Git**

Git is the most popular Distributed VC system today. In Git, changes in source control don't have to be committed to a single central repository. Then, this removes the requirement for every member of your team to have access to that central repository and download the newest code just to save their changes. Instead, every member of your team may have their own localized repository containing its whole history.

It was developed to meet the following goals:

* a simple design for easy maintenance.
* strong support for non-linear development with many parallel independent lines of source code and a completely distributed system.

**5.1. Main VC system concepts** (Rother, 2017).

• *Repository*: A central directory that keeps copies of changes made through a file. In its simplest form, a repository is nothing more than an easily created, hidden folder named ".git" in the root directory of all your project files. There can be a local and a remote repository.

• *Local repository*: This will be a local ".git" directory in the root directory of your project in your computer. Only the team member is authorized to work with this repository, committing changes to it. It may be any of the chosen directories on your computer.

• *Remote repository*: This is a remote ".git" folder, usually located on some remote server on the internet, or in your local network. The remote repository doesn't include real working files. No work is done in a remote repository. Instead, it contains just the ".git" repository folder. Remote repositories are used by team members for sharing and exchanging data. These repositories serve as a shared foundation where everyone can publish their own updates and receive changes from their teammates.

• *Blob*: A blob is a binary large object version of a file. In other words, a blob simply stores the actual content of the file without all the accidental metadata accompanying it, like a simple binary file within the Git database.

• *Trees*: These are objects holding a directory. They contain both blobs and other subdirectories. A tree is a binary file storing references to blobs and trees.

• *Commits* represent one state of the repository. A commit object is a node in a linked list, and each Commit object refers to its parent Commit objects. The parent pointers can be used to trace back through the commit history. If a commit has more than one parent commit then the commit was produced by merging two branches.

• A *branch* means that independent developments are taken from the same source code. Several branches can be merged into any single branch within the same repository. Git usually starts by default with a 'master' branch. The usual case is making a branch to develop a new feature of a program. After finishing the feature, it is merged again into the master and then deleted.

• *Clone* itself is an operation used to create the instance of the repository. The cloning operation checks out working copy and it mirrors the complete repository. The team members can perform many operations with this local repository. Networking is only required when the repository instances are syncing with each other.

• *Pull* is an operation that makes a copy of the changes from an instance of a remote repository to an instance of a local repository. The pull operation enables synchronization between two repository instances. This operation is used quite frequently.

• *Push*: It is an operation that makes a copy of the changes from an instance of the local repository to a remote one. The reason for using this operation is to store the changes permanently in the Git repository. This operation is highly used.

• *HEAD* is a reference to the last commit created on the branch. Every time you commit, it updates HEAD with this new commit. This is unique in terms of code.

• *Merge*: This is the operation that commits one location into another. Be it for testing, debugging, or any other reason, branches are created and changes can be applied from one place to another through merging. It incorporates changes from the source branch to the target one. The former remains intact, and its history is maintained.

•*Rebase*: Another way to integrate changes from one branch to another is by rebasing. The changes will all be squished into one big “patch," and this patch will then be applied to the target of changes. In contrast with merging, rebasing creates a narrow history. Rebasing relocates done work from one branch to another. History that is not necessary is discarded in the process.

• *Diff*: It is a utility that takes different versions of your files as input to compute their differences. It is used to compare different versions of files and changes made in them. One of the most typical use cases would be examining which changes were made since states the last commit. (Chowdhury, 2019)

**6. Conclusion**

In other words, the popularity is reflected in its versatility, large collections of modules and tools, and in having an active user community. It became a necessity for developers, especially now that the potential tech industry mushrooms because of changes in AI, data science, and machine learning. The importance of Python—rooted in its flexible nature and essential features since it is considered an anchor in every developer's arsenal—is further increased by these factors. As this community continues to grow and applications continue to diversify, Python will sustain its impetus to drive several important innovations shortly founded on its robust language capability.

**7. Bibliography**

1. Moore.A.D, Python GUI Programming with Tkinter (Packt Publishing Ltd.May, 2018), 2.Ed.
2. Harwani.B.M, Qt5 Python GUI Programming Cookbook (Packt Publishing Mirmingham- Mumbai, 2018), 1.Ed.
3. Phillips.D, Creating Apps in Kivy Mobile with Python( [O'Reilly Media](https://www.google.de/search?hl=en&gbpv=1&dq=Kivy+Documentation&pg=PT178&printsec=frontcover&q=inpublisher:%22O%27Reilly+Media%22&tbm=bks&sa=X&ved=2ahUKEwiOj7-El_KGAxW5xQIHHZIfAVcQmxMoAHoECBcQAg&sxsrf=ADLYWIKDOszH8K6iByNstFoIVe6cF5mCTw:1719161333189), 2014), ISBN: 9781491947357, 1491947357
4. Summerfield.M, Rapid GUI Programming with Python and QT: The Definitive Guide to PyQt Programming (Prentice Hall, 2007), ISBN: 9780132354189
5. Rother.K, Pro python best practices: Debugging, testing and maintenance (Apress, 2017), ISBN:9781484222416, 1484222415 O
6. Chowdhury. K, Git tutorial for beginners (Git bash commands) (2019), <https://www.kunal-chowdhury.com/p/git-cheatsheet.html>
7. Hunt.J, A Beginners Guide to Python3 Programming, (AG 2023), ISBN 978-3-031-35122-8 (eBook)
8. Lott,S. , Phillips, D. , Python Object-Oriented, (2021), ISBN 978-1-80107-726-2
9. Strien.D, (2016), [An Introduction to Version Control Using GitHub Desktop](https://programminghistorian.org/en/lessons/retired/getting-started-with-github-desktop), <https://programminghistorian.org/en/lessons/retired/getting-started-with-github-desktop#what-is-version-control-and-why-use-it>
10. Rawat. A, What is a version control system?, (2022), Published in Stackademic,

<https://blog.stackademic.com/what-is-a-version-control-system-2f3509066b72>

1. Rao, A, What is version control? Definition, types, systems and tools, (2023),

<https://blog.logrocket.com/product-management/version-control-systems-definition-types>

**8.Task Assignment:**

The Programs and the Output as well as the unit tests of the task assignment is attached as Appendices.

Moreover, The Code for the additional Task is mentioned as follows :

Git address : https://github.com/fahimehbahman/PythonCourseAssignment.git

**9.List of Appendices:**

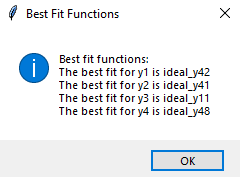
|  |  |  |
| --- | --- | --- |
| **SL No** | **Particulars** | **Page No** |
| **1** | **Program** |  |
| **2** | **Output** |  |
| **3** | **Unittest** |  |

**10.Appendix:**

* **Program:**

from tkinter import \*  
from tkinter import filedialog  
from tkinter import ttk  
from tkinter import messagebox as msg  
import pandas as pd  
import webbrowser  
from Service.DataProcessor import DataProcessor  
from Service.IdealFunctionLoader import IdealFunctionLoader  
from Service.TestDataLoader import TestDataLoader  
from Service.TrainingDataLoader import TrainingDataLoader  
from sqlalchemy import create\_engine, Column, Float, String, Table, MetaData  
import os  
  
class Main:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.setup\_gui()  
  
 def setup\_gui(self):  
 self.root.geometry("700x700")  
 self.root.resizable(0, 0)  
 self.root.title("Main")  
 self.root.configure(background="white")  
  
 # Training file  
 lblPath1 = Label(self.root, text="Training file ", bg="white")  
 lblPath1.grid(row=0, column=0, padx=(10, 10), pady=10)  
 button1 = Button(self.root, text="...", command=lambda: self.open\_file\_dialog(self.entryPath1))  
 button1.grid(row=0, column=1, pady=10)  
 self.entryPath1 = ttk.Entry(self.root, width=50)  
 self.entryPath1.grid(row=0, column=2, padx=(10, 10), pady=10)  
 link1 = Label(self.root, text="Download Sample Training File", fg="blue", cursor="hand2", bg="white")  
 link1.grid(row=0, column=3, padx=(10, 10), pady=10)  
 link1.bind("<Button-1>", lambda e: self.open\_url(os.path.abspath("SampleInputExcel//Training1.csv")))  
  
  
 # Function file  
 lblFunctionPath = Label(self.root, text="Function file", bg="white")  
 lblFunctionPath.grid(row=4, column=0, padx=(10, 10), pady=10)  
 buttonFunction = Button(self.root, text="...", command=lambda: self.open\_file\_dialog(self.entryFunction))  
 buttonFunction.grid(row=4, column=1, pady=10)  
 self.entryFunction = ttk.Entry(self.root, width=50)  
 self.entryFunction.grid(row=4, column=2, padx=(10, 10), pady=10)  
 link5 = Label(self.root, text="Download Sample Function File", fg="blue", cursor="hand2", bg="white")  
 link5.grid(row=4, column=3, padx=(10, 10), pady=10)  
 link5.bind("<Button-1>", lambda e: self.open\_url(os.path.abspath("SampleInputExcel//ideal\_functions.csv")))  
  
 # Test file  
 lblTestPath = Label(self.root, text="Test file", bg="white")  
 lblTestPath.grid(row=5, column=0, padx=(10, 10), pady=10)  
 buttonTest = Button(self.root, text="...", command=lambda: self.open\_file\_dialog(self.entryTest))  
 buttonTest.grid(row=5, column=1, pady=10)  
 self.entryTest = ttk.Entry(self.root, width=50)  
 self.entryTest.grid(row=5, column=2, padx=(10, 10), pady=10)  
 link6 = Label(self.root, text="Download Sample Test File", fg="blue", cursor="hand2", bg="white")  
 link6.grid(row=5, column=3, padx=(10, 10), pady=10)  
 link6.bind("<Button-1>", lambda e: self.open\_url(os.path.abspath("SampleInputExcel//test\_data.csv")))  
  
 btnRegister = Button(self.root, text="Register", command=self.register)  
 btnRegister.grid(row=6, column=1, columnspan=2, padx=(10, 10), pady=20)  
  
 def open\_file\_dialog(self, entry):  
 file\_path = filedialog.askopenfilename(filetypes=[("CSV files", "\*.csv"), ("All files", "\*.\*")])  
 if file\_path:  
 entry.delete(0, END)  
 entry.insert(END, file\_path)  
  
 def open\_url(self, url):  
 webbrowser.open\_new(url)  
  
  
  
 def register(self):  
 if self.entryPath1.get() == "" or self.entryFunction.get() == "" or self.entryTest.get() == "":  
 msg.showinfo("Warning", "Please select all CSV files.")  
 return  
  
 filePath1 = self.entryPath1.get()  
 fileFunction = self.entryFunction.get()  
 fileTest = self.entryTest.get()  
  
 try:  
  
 # create an Instance from class  
 ideal\_loader = IdealFunctionLoader(fileFunction)  
 test\_loader = TestDataLoader(fileTest)  
 trainer\_loader = TrainingDataLoader(filePath1)  
  
 # Validate CSV columns  
 trainer\_loader.validateCSVColumns(filePath1)  
 ideal\_loader.validateCSVColumns(fileFunction)  
 test\_loader.validateCSVColumns(fileTest)  
  
 # Create a SQLite database and tables  
 processor = DataProcessor(db\_folder='Model', db\_name='PythonTaskDataBase.db')  
 processor.create\_tables()  
 # Proceed with other operations...  
  
 # Load training data into the database  
  
 trainer\_loader.load()  
 trainer\_df = trainer\_loader.get\_dataframe()  
 processor.save\_to\_db(trainer\_df, 'Trainers')  
  
  
 # Load ideal functions data into the database  
 ideal\_loader.load()  
 ideal\_df = ideal\_loader.get\_dataframe()  
 processor.save\_to\_db(ideal\_df, 'IdealFunctions')  
  
 # Load test data  
 test\_loader.load()  
 test\_df = test\_loader.get\_dataframe()  
  
 # Find the best fit functions for each training data column  
 best\_fit\_functions = processor.find\_best\_fit(trainer\_df, ideal\_df)  
  
 # Calculate deviations for test data  
 deviations = processor.calculate\_deviation(test\_df, ideal\_df, best\_fit\_functions)  
 deviation\_df = pd.DataFrame(deviations, columns=['X', 'Y', 'ideal\_function', 'deviation'])  
 processor.save\_to\_db(deviation\_df, 'TestResults')  
  
 # Print the results  
 result\_str = "Best fit functions:\n"  
 for key, value in best\_fit\_functions.items():  
 result\_str += f"The best fit for {key} is {value}\n"  
 msg.showinfo("Best Fit Functions", result\_str)  
  
 msg.showinfo("Success", "Data successfully loaded into the database and best fit functions identified.")  
 except Exception as e:  
 msg.showinfo("Error", f"Failed to process CSV files: {str(e)}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = Tk()  
 Main(root)  
 root.mainloop()

* **Output:**



* **Useful Unit Tests:**

**For DataProcessor file in service**

|  |
| --- |
| import unittest from unittest.mock import patch, MagicMock import os import pandas as pd from Service.DataProcessor import DataProcessor from sqlalchemy import create\_engine, MetaData, Table, Column, Float   class TestDataProcessor(unittest.TestCase):   def setUp(self):  self.db\_folder = 'TestDatabase'  self.db\_name = 'TestPythonTaskDataBase.db'  self.processor = DataProcessor(db\_folder=self.db\_folder, db\_name=self.db\_name)   def tearDown(self):  db\_path = os.path.join(self.db\_folder, self.db\_name)  if os.path.exists(db\_path):  os.remove(db\_path)  if os.path.exists(self.db\_folder):  os.rmdir(self.db\_folder)   def test\_create\_tables(self):  with patch.object(self.processor.metadata, 'create\_all') as mock\_create\_all:  self.processor.create\_tables()  mock\_create\_all.assert\_called\_once()   @patch('pandas.DataFrame.to\_sql')  def test\_save\_to\_db(self, mock\_to\_sql):  df = pd.DataFrame({'x': [1, 2, 3], 'y1': [4, 5, 6], 'y2': [7, 8, 9]})  self.processor.save\_to\_db(df, 'TestTable')  mock\_to\_sql.assert\_called\_once\_with('TestTable', self.processor.engine, if\_exists='replace', index=False)   def test\_find\_best\_fit(self):  training\_df = pd.DataFrame({  'x': [1, 2, 3],  'y1': [1, 2, 3],  'y2': [2, 4, 6]  })  ideal\_df = pd.DataFrame({  'x': [1, 2, 3],  'f1': [1, 2, 3],  'f2': [2, 4, 6],  'f3': [3, 6, 9]  })  best\_fit = self.processor.find\_best\_fit(training\_df, ideal\_df)  expected\_best\_fit = {'y1': 'f1', 'y2': 'f2'}  self.assertEqual(best\_fit, expected\_best\_fit)   def test\_calculate\_deviation(self):  test\_df = pd.DataFrame({  'x': [1, 2, 3],  'y': [1, 2, 3]  })  ideal\_df = pd.DataFrame({  'x': [1, 2, 3],  'f1': [1, 2, 3],  'f2': [2, 4, 6]  })  best\_fit\_functions = {'y': 'f1'}  deviations = self.processor.calculate\_deviation(test\_df, ideal\_df, best\_fit\_functions)  expected\_deviations = [  (1, 1, 'f1', 0),  (2, 2, 'f1', 0),  (3, 3, 'f1', 0)  ]  self.assertEqual(deviations, expected\_deviations)  if \_\_name\_\_ == '\_\_main\_\_':  unittest.main() |