1. **Apply Python tools to examine the quality of the code.**

The health and growth of large projects depend on maintaining consistent code quality. To accomplish it manually, though, can take time and effort. Fortunately, many Python code tools can simplify and speed up this procedure. The tool decision is ultimately a matter of personal preference, as with many other facets of software development (Eriksen, 2019).

VanTol (2018) stated that evaluating a particular code, programme, or software is called "coding quality." Code is typically seen as excellent quality if it is adequately documented and its lines are simple to understand.

The challenge of evaluating the quality of a piece of code is subjective and varies by team, organisation, and sector (Bellairs, 2019). For instance, the definition of excellent code in a financial company can differ from that in a cybersecurity company. An organisation's user experience, application security, maintainability, testability, readability, and software performance are all significantly impacted by the quality of the code. It is significant for several reasons, such as:

- Reliability: Poorly written code has a lower probability of having flaws, which can cause crashes, data loss, and other issues.
- Maintainability: High-quality code is simpler to comprehend and adapt, saving time and money.
- Scalability: Excellent code can be expanded to accommodate heavier demands easily.
- Security: Vulnerabilities in high-quality code are less likely to exist.

According to Bellairs (2019), several factors, such as the following, might influence the quality of a piece of software:

- Coding style: The code ought to be cleanly organised and simple to understand.
- The code must be thoroughly documented for other developers to comprehend it.
- Testing: To ensure the code functions properly, it should be thoroughly tested.
- Security: Vulnerabilities in the code should be examined.

Regulation is essential to our work as developers. It must be highly calibrated before distributing or using the code in a live environment (Codersid, 2023). For instance, here are some important factors to take into account while evaluating the quality of the code:

- Readability: Good code should be simple to read and comprehend. When appropriate, comments should be used and well-organized with meaningful variable and function names.
- Coding etiquette: The code should adhere to the etiquette prescribed by the programming language and framework it was created in.
- Maintainability: Well-written code ought to be simple to update and maintain. It should be modular, have distinct divisions of labour, and reduce dependencies between various code sections.
- Efficiency: Effective resource use and efficiency are key components of well-written code. It should be performance optimised and not conduct extraneous calculations or processes.
- Testing: A complete set of tests should be included in well-written code to guarantee its accuracy and maintainability.
- Error handling: Well-written code should be able to gracefully and predictably respond to errors and unforeseen circumstances.

For any software developer, evaluating the quality of the code is essential. Fortunately, Python provides a wide range of tools and modules that can be quite helpful in examining and improving code quality (Dwyer, 2023). For instance, the following Python tools are used to assess the quality of code:

1. Code linters can examine the code to find potential errors, assure compliance with coding standards, and find stylistic violations.
2. code formatters can automatically format the code by predetermined criteria to create a uniform code style.
3. Tools for measuring code coverage calculate the proportion of the code that tests run.
4. Static analysis tools may examine the code without actually running it, warning about potential problems and giving information about the complexity and maintainability of the code.

Shaw (2018) mentioned that Python's built-in tools and libraries simplify testing. Our ability to verify that our applications are operating as planned. Python makes testing easy, allowing you to check your code quickly and sustainably using unit tests.

2. Implement data structures to store data.

Effective storage, handling, and access to data are essential when working with information. This is where data structures come into play, serving to meet our needs in an organised manner (Raj, 2022). Data structures are the fundamental building blocks of programming languages, providing a systematic approach to fulfilling the demands above. Python, for example, has four types of data structures: List, Tuple, Dictionary, and Set, referred to as implicit or built-in data structures. These structures enable us to manage, connect, manipulate, and utilise our data in various ways. Additionally, user-defined custom data structures like Stack, Queue, Tree, Linked List, and Graph offer customers complete control over their functionality and are suitable for more complex programming tasks.

Data structures are specialised formats to efficiently organise, analyse, retrieve, and store data for certain purposes. Various basic and advanced data structure types allow consumers to access and utilise data in the best possible ways (Loshin, 2021). Also, Akash (2019) states that data structures simplify information organisation so that machines and people can understand it. Since data structures provide an effective method for organising and manipulating data, their proper use is essential for efficient data storage in programming.

Data structures act as the foundation for building programmes. They provide original approaches to data organisation for the best accessibility depending on the intended use (Loshin, 2021).

Data structures must be built through important processes to store data successfully (Novotny, 2023). For instance, these procedures include carefully testing the data structure, precisely specifying it, correctly implementing it, and choosing the best data structure.

According to Novotny (2023), utilising compound data structures such as lists, arrays, and hash tables is essential for managing large amounts of data. They recommend implementing formal programming methods, modular design, and expert coding techniques. These data structures provide standardised solutions that decrease development and testing efforts while being faster and more memory-efficient. Additionally, they are easier to comprehend and manage since programmers follow the same conventions.

We must completely comprehend and use data structures to be proficient programmers. It is necessary for effective data organising and manipulation. The implementation of frequently used data structures in Python will be covered in detail in this guide. Sannikov (2022) outlines the processes for constructing the frequently used data structures in Python:

1. Python lists are dynamic arrays that can hold different data types and objects. They are ordered and can be accessed by index. We can add, remove, and change elements within the list. Lists are useful for storing nested data structures and for data analysis. However, they can be slow for arithmetic operations and use more disk space.

```
# Creating a list
my_list = [1, 2, 3, 4, 5]
my_list1 = list()
```

2. Python dictionaries are like real-world dictionaries with keys and values. They help access specific data related to a unique key quickly. It's crucial to ensure key uniqueness to make sure everything is clear. Dictionaries are useful for mapping a special key to data and quick look-up, but they need more space and may cause compatibility issues in different Python versions.

```
# Creating a dictionary
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}

# Accessing values in a dictionary
print(my_dict['name'])

# Adding or modifying values in a dictionary
my_dict['occupation'] = 'Engineer'

# Removing a key-value pair from a dictionary
del my_dict['age']
```

3. Python sets are useful for removing duplicates and have set operations similar to mathematics. They're also efficient for checking element presence but not for maintaining insertion order or changing elements through indexing.

```
# Creating a set
my_set = {1, 2, 3}
# Adding elements to a set
my_set.add(4)
# Removing elements from a set
my_set.remove(3)

# Set operations
other_set = {3, 4, 5}
union_set = my_set.union(other_set)
intersection_set = my_set.intersection(other_set)
```

4. Lists and tuples are similar, but tuples are immutable and can be used as dictionary keys if their elements are immutable. Creating a tuple is easy, and they provide benefits such as content protection, but they are unsuitable for modifiable objects and take up more memory than lists.

```
# Creating a tuple
my_tuple = (1, 2, 3)
```

Sannikov (2022) concluded that understanding data structures is crucial to programming since it enables effective data storage and retrieval. Lists, dictionaries, sets, and tuples are the four main types of data structures in Python, while lists, sets, and dictionaries are immutable types. Lists work well for storing various related items, whereas dictionaries are needed to quickly access data and link a key to a value, much like a real-world dictionary. Sets are excellent for comparing two data sets because they can easily perform operations like intersection and difference. Although immutable, tuples are similar to lists in that they act as error-proof data containers.

3. Implement a search algorithm to explore stored data.

In practically any application, finding data that is stored in several data structures is an essential task. Several algorithms can be used, each with its implementation and using various data structures. Developers must be adept at selecting the appropriate algorithm for a given task because doing so can make or break an application. The

choice of algorithm can affect the application's performance in terms of speed, dependability, and stability, and if the incorrect algorithm is made, even a straightforward request may fail (Stack Abuse, 2019).

Any algorithm that retrieves information from a data structure or is calculated in the search space of a problem domain, either with discrete or continuous values, is considered a search algorithm, according to Wikipedia (2019).

Based on their operation, search algorithms can be divided into two main categories: sequential search and interval search (GeeksforGeeks, 2017). Like linear search, sequential search includes going through a list or array one element at a time and verifying them all. However, interval search uses algorithms that are much more effective than linear search and is specifically created for sorted data structures. These algorithms divide the search space in half and continually target the centre of the search structure. An excellent illustration of an interval search algorithm is a binary search.

An essential and basic search method is the *linear search* algorithm. It is comparable to an improved in-operator for Python that we can build ourselves (Stack Abuse, 2019). The procedure consists of exhaustively scanning an array and immediately reporting the index of the first instance of the desired item. Let's take a look at an idea *linear search* presented by Thomas (2022):

```
def LinearSearch(data, target):
    for index, value in enumerate(data):
        if value == target:
            return index
    return -1  # Target not found

my_list = [10, 5, 2, 8, 3, 9]
target = 8
result = LinearSearch(my_list, target)
if result != -1:
    print(f"Found at index {result}")
else:
```

```
    print("Target not found")
```

In this example, the LinearSearch function repeatedly iterates through the data list, comparing each member to the goal value. The index is given back if a match is discovered. -1 is returned if the search is unsuccessful in finding a match.

The *binary search* algorithm is the preferred technique for locating a particular value in a sorted array. It uses the "divide and conquer" strategy, which is extremely effective, and is the fastest searching approach accessible (Krishna, 2022). Let's examine a *binary search* to see how this approach functions in action:

```
def BinarySearch(list, target):
    start = 0
    end = len(list)-1

    while start <= end:
        mid = start + (end-start)//2


        if list[mid] > target:
            end = mid-1
        elif list[mid] < target:
            start = mid+1
        else:
            return mid

    return -1

if __name__ == '__main__':
    list = [2, 12, 15, 17, 27, 29, 45]
    target = 17
    print(f"Found at index {result}")
```

A start and end index defines the search range for the BinarySearch function. It calculates the middle index and compares the value to the desired value. The comparison reduces the search range until a match is discovered or the search space is exhausted.

Implementing and using search algorithms like linear search or binary search, which operate based on certain search criteria, is essential for effectively exploring and retrieving material from stored collections.

References:

Eriksen, M. (2019). *5 Awesome Tools For Python Code Quality*. [online] Available at: https://dev.to/madelyneriksen/5-awesome-tools-for-python-code-quality-2pc [Accessed 17 Jun. 2023].

VanTol, A. (2018). *Python Code Quality: Tools & Best Practices. [online] Realpython.com. Available at: https://realpython.com/python-code-quality/.*

Bellairs, R. (2019). *What Is Code Quality? Overview + How to Improve Code Quality*. [online] Perforce Software. Available at: https://www.perforce.com/blog/sca/what-code-quality-overview-how-improve-code-quality.

CODERSID BLOGS (2023). *How to Find Examples of Well-Written & Clean Code?* [online] Codersid. Available at: https://codersid.com/how-to-find-examples-of-well-written-clean-code/ [Accessed 17 Jun. 2023].

Dwyer, T. (2023). *12 Python Scripts for Optimising Your Code Development*. [online] Medium. Available at: https://bootcamp.uxdesign.cc/12-python-scripts-for-optimising-your-code-development-310d4a0915ca [Accessed 16 Jun. 2023].

Shaw, A. (2018). *Getting Started With Testing in Python – Real Python*. [online] realpython.com. Available at: https://realpython.com/python-testing/#unit-tests-vs-integration-tests.

Raj, A. (2022). *Data Structures in Python - AskPython*. [online] Available at: https://www.askpython.com/python/data-structures-in-python#:~:text=The%20data%20structures%20in%20Python%20are%20List%2C%20Tuple%2C [Accessed 12 Jun. 2023].

Loshin, D. (2021). *What are Data Structures? - Definition from WhatIs.com*. [online] SearchDataManagement. Available at: https://www.techtarget.com/searchdatamanagement/definition/data-structure.

Akash (2019). *Data Structures in Python | List, Tuple, Dict, Sets, Stack, Queue*. [online] Available at: https://www.edureka.co/blog/data-structures-in-python/.

Novotny, J. (2023). Understanding Data Structures: Definition, Uses & Benefits. [online] Available at: https://www.linode.com/docs/guides/data-structure/.

Sannikov, A. (2022). *Python Data Structures: Lists, Dictionaries, Sets, Tuples (2022)*. [online] Dataquest. Available at: https://www.dataquest.io/blog/data-structures-in-python/.

Stack Abuse. (N.D.). *Search Algorithms in Python*. [online] Available at: https://stackabuse.com/search-algorithms-in-python/.

GeeksforGeeks (2017). *I am searching Algorithms - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/searching-algorithms/.

Wikipedia Contributors (2019). *Search algorithm*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Search_algorithm.

Thomas (2022). *Linear Search through a list*. [online] Available at: https://stackoverflow.com/questions/74026648/linear-search-through-a-list [Accessed 20 Jul. 2023].

Krishna, A. (2022). *Search Algorithms – Linear Search and Binary Search Code Implementation and Complexity Analysis*. [online] freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/search-algorithms-linear-and-binary-search-explained/