Advanced Python Guide

Master concepts, build applications, and prepare for interviews

Kriti Kumari Sinha

www.bpbonline.com

First Edition 2024

ISBN: 978-93-55516-756

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Dedicated to

My Growing Daughters
Aradhya Sinha *and* ***Anvi Sinha***

About the Author

Kriti Kumari Sinha, a seasoned professional with two decades of IT expertise, is a true visionary in the fields of Data Science, Machine Learning, AI, and Gen AI. Her passion for knowledge extends beyond her own career, as she actively mentors aspiring professionals in these domains.

Kriti, currently Director at Capgemini India Ltd., epitomizes unwavering excellence across multiple multinational companies, especially in Data Science, AI, and Gen AI, all powered by Python.

Kriti's influence extends globally. Her acclaimed book, "Structured Query Language for all RDBMS and PL/SQL," has reached readers across 44 countries. As a global panel member at McGraw Hill Pvt. Ltd., she actively fuels innovation and knowledge sharing.

Kriti Kumari Sinha's journey exemplifies dedication, expertise, and a relentless pursuit of excellence. Her impact on the world of technology and education continues to inspire professionals worldwide.

About the Reviewer

Sahajdeep Oberoi is a Data Engineer with 3 years of experience. During his college, he was a Python instructor for multiple startups and had organised various Python Workshops including facilitation at Google supported program - Explore ML. He also has a YouTube channel to help and guide college students. His primary skills consist of Python, Pyspark, AWS, SQL.

Acknowledgement

Writing a book is a journey that often involves the collective effort and support of many individuals, and this book on Python is no exception. I would like to extend my heartfelt thanks to the following people and organizations for their invaluable contributions and support throughout this endeavor:

To my pillar of support, my husband, and our two wonderful growing daughters, Aradhya and Anvi.

This book is as much yours as it is mine. Your unwavering encouragement and boundless patience during the countless late nights and weekends spent writing and editing have been the driving force behind this endeavor.

You have witnessed the sacrifices and the long hours, yet you have remained steadfast in your support. Your belief in my work has been a constant source of inspiration, and your understanding of the passion that drives me is a priceless gift.

As Aradhya and Anvi continue to grow, I hope this book serves as a testament to the importance of dedication and hard work in achieving one's goals. Your presence and unwavering love is what makes it all worthwhile.

With all my love and gratitude,

Kriti Sinha

Preface

Welcome to a book all about *Advanced Python Guide*! Whether you are just starting out or dreaming of becoming a data analyst/ data scientist/python programmer, this book is your guide to the awesome world of Python. Let us take a journey together where we will learn all about Python, step-by-step.

Python is on high demand by programmers worldwide because it is easy to read and can do lots of things in very less time. It is used for making websites, analyzing data, automating tasks, and much more. However, what makes Python special is the community of people who love to work together and solve easy to tough problems.

In this book, we will cover everything you need to know, starting from the basics and going all the way to advanced stuff. Get ready to learn about:

- **Introduction to Python:** Find where Python comes from and why it is great for beginners.
- **Data structures:** Learn about lists, dictionaries, tuples, and sets, which are the building blocks of Python programs.
- **Object-Oriented Programming (OOP):** Discover how to write fancy, reusable code using classes and objects.
- **File handling:** Master the skill of reading and writing files with different types, which every programmer needs.
- **Modules and packages:** Explore all the extra tools and libraries that make Python even more powerful to create Module and Packages.
- **Best practices and coding standards:** Learn how to write code that is easy to understand, manage and works well.
- **Real-world applications:** See how Python is used in practical projects like making websites or automating tasks.
- **Machine Learning:** Try your hand at using Machine Learning with tools like scikit-learn (also known as sklearn), TensorFlow and Keras (two popular tools used in the world of artificial intelligence and machine learning).
- **Python interview preparation for freshers and experienced developers:** This part of the book will prepare you for your exam and interviews.

I hope you will find this book informative and helpful.

Chapter 1: Introduction to Python - This chapter lays the foundation by introducing the basics of Python. You will be acquainted with various data types such as integers,

floats, strings, lists, dictionaries, and tuples. Additionally, the chapter discusses arithmetic operations, elucidating the methods for performing mathematical calculations within the Python environment. This chapter also covers string manipulation, highlighting the diverse range of string methods available for text manipulation tasks. Moreover, you will be guided through conditional statements, mastering the usage of if-else constructs for effective decision-making processes. Lastly, the concept of mutability is discussed, clarifying the differences between mutable, and immutable objects.

Chapter 2: Python Basics - This chapter discusses the fundamental Python concepts. You will gain insight into ordered collections of elements through an examination of lists and tuples. You will learn about string manipulation functions, and uncover the power of dictionaries in storing data as key-value pairs. Additionally, you will learn how to master control flow by using if, elif, and else statements.

Chapter 3: Data Structures - In this chapter, you will discover various data structures in Python. This chapter will include topics like Lists, and Tuples. Lists are dynamic arrays ideal for storing multiple values. Tuples, on the other hand, provide immutable sequences of elements. Moreover, in this chapter, dictionaries are introduced as key-value mappings, enabling efficient data retrieval

Chapter 4: Functions - In this chapter, you will gain a comprehensive understanding of functions through exploration of key concepts. Modular programming principles emphasize breaking down code into reusable functions. By utilizing named arguments, programmers can enhance code clarity. Additionally, in this chapter, grasp the versatility afforded by default values and variable-length arguments, enabling customization of function behavior.

Chapter 5: Object-oriented Programming - In this chapter, embark on a journey into Object-oriented Programming (OOP) principles. The chapter begins by making you understand the foundational concepts of classes and objects, which creates blueprints for objects. Through inheritance, you will be able to build hierarchies of related classes. Furthermore, embrace the concept of encapsulation, which entails hiding implementation details.

Chapter 6: File Handling - This chapter will teach you how to work with files. You will learn how to efficiently read data from files, gaining access to their content. Additionally, the chapter covers writing data to files, and error handling in file operations, mastering techniques to handle exceptions that may arise during file operations.

Chapter 7: Modules and Packages - This chapter explores Python modules and packages. Modules organizes code into reusable files, while packages group related modules together.

Chapter 8: Python's Standard Library and Third-party Libraries - This chapter will help you understand how to get acquainted with built-in modules and functions. The Math module you equip you to perform various mathematical operations. The os module facilitates interaction with the operating system, and the datetime module handles dates and times.

Chapter 9: Pythonic Programming – This chapter will help you embrace Pythonic coding style. By adopting Pythonic idioms, you can write clean, concise, and idiomatic code. Using list comprehensions, you can learn to create lists elegantly. Additionally, context managers covered in this chapter will help you in efficient resource management.

Chapter 10: Advanced Topics in Python - In this chapter, learn about more complex Python concepts. While Generators will help you create memory-efficient iterators, decorators will help you modify or enhance functions. The Context managers in this chapter will help you manage resources using the with statement.

Chapter 11: Testing and Debugging - In this chapter, learn various effective testing techniques like Unit testing, and Debugging strategies. While Unit testing helps validate individual components, debugging strategies identify and fixes issues.

Chapter 12: Best Practices and Coding Standards - In this chapter, you will explore the industry best practices for writing maintainable code. The chapter will cover coding conventions that will help you understand the importance of consistent naming conventions, indentation, and other style guidelines. Additionally, you will understand about writing clear and informative comments to enhance code readability. Further, this chapter will teach you strategies for structuring your code logically, including modularization and separation of concerns, and implementing robust error-handling mechanisms to handle exceptions gracefully. At the end of this chapter, you will discover the significance of unit testing, test-driven development (TDD), and writing testable code.

Chapter 13: Building Real-world Applications - This chapter takes your Python skills beyond theory and into practical territory. You will learn how to apply Python concepts to create real-world applications. You will also be able to explore use cases, and learn about scenarios like web development, data analysis, automation, and more. Finally, this chapter has some projects that will help you work on hands-on projects that simulate real-world challenges. Examples include building a web scraper, creating a simple web app, or automating repetitive tasks.

Chapter 14: Python's Future and Trends - This chapter will help you stay up-to-date with Python's evolution. You will learn about the features planned for future Python releases. This chapter will also keep you informed about popular libraries, frameworks, and tools along with teaching how Python is being used in various domains (e.g., machine learning, web development, scientific computing).

Chapter 15: Hands-on Python Programming - This chapter emphasizes practical learning through exercises and coding challenges. There are hands-on practice exercises, and problem-solving exercises that will help you tackle real-world problems using Python. This chapter will also teach you about Scikit-learn integration and how you can apply machine learning algorithms using scikit-learn (sklearn) to solve specific tasks.

Chapter 16: Python Interview Preparation: Beginners - This chapter will help you prepare for Python-related interviews. We will brush up on fundamental Python topics, and practice solving coding problems commonly asked in interviews.

Chapter 17: Python Interview Preparation for Experienced Developers - This last chapter is tailored for seasoned Python developers. It covers challenging interview questions, preparing you for in-depth technical discussions.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/rytc5ti

The code bundle for the book is also hosted on GitHub at
https://github.com/bpbpublications/Advanced-Python-Guide.
In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

Chapter 1
Introduction to Python

Introduction

Welcome to the world of Python! It is a special computer language that is famous for being easy to read and use. A smart person named *Guido van Rossum* made it in 1991, and since then, Python has become super important in the world of programming. People use Python for many things like making websites, working with data, creating smart computer programs, and more. In this first chapter, we are going to explore the basics of Python. Whether you are just starting your coding journey or you are a pro looking for a reliable tool, this chapter will help you get to know Python and use it effectively. Let us dive into the foundations of Python and discover how it can make your coding adventures exciting and powerful!

Structure

The chapter discusses the following topics:

- History and evolution of Python
- Python's popularity and applications
- Installing Python
- Setting up a development environment

History and evolution of Python

Python, a versatile and powerful programming language, has a rich history and has evolved significantly since its inception. In this section, we will explore the key milestones and developments that have shaped Python into the language we know today.

- **Late 1980s**: Python's story begins in the late 1980s when *Guido van Rossum*, a Dutch programmer, started working on a new programming language during his Christmas holidays in December 1989. He was inspired by the ABC language and aimed to create a language that emphasized code readability and had a clear and simple syntax.
- **1991:** The first public release of Python, version 0.9.0, was made available in February 1991. This marked the official birth of Python. This release already included many fundamental features, including exception handling, functions, and modules.
- **Python 1.0 (January 1994):** Python 1.0 was a significant milestone for the language. It introduced several key features, such as lambda functions, map, filter, and reduce functions, which made Python more expressive and powerful.
- **Python 2.0 (October 2000):** Python 2.0 continued to refine the language. It introduced list comprehensions, garbage collection, and Unicode support, making Python more versatile and capable.
- **Python 3.0 (December 2008):** Python 3.0 marked a major overhaul of the language. *Guido van Rossum* and the development team took the opportunity to clean up and simplify Python syntax, removing redundancy and inconsistencies. Some significant changes included the introduction of the **`print()`** function (replacing the print statement), the addition of the bytes data type, and the removal of older features that had been deprecated.
- **Python 3.x Series:** The Python 3 series continued to evolve with regular releases, bringing new features, optimizations, and improvements to the language. These included features like type hints (PEP 484), asynchronous programming with the async and await keywords (PEP 492), and f-strings for easier string formatting (PEP 498).
- **Python Software Foundation (PSF):** In 2001, PSF was established as a non-profit organization to promote and protect Python. The PSF plays a crucial role in Python's development and community support.
- **Python in web development:** Python gained popularity in web development, with frameworks like Django and Flask becoming widely used for building web applications. These frameworks streamlined web development and encouraged best practices.

- **Data science and machine learning:** Python became a dominant language in data science and machine learning, thanks to libraries like NumPy, pandas, scikit-learn, and TensorFlow. Its simplicity and rich ecosystem made it a preferred choice for data analysis and machine learning projects.
- **Python's popularity:** Python's simplicity, readability, and versatility contributed to its widespread adoption across various domains, from web development and data analysis to scientific computing and Artificial Intelligence.
- **Python 2 End of Life (EOL):** Python 2 reached its **End Of Life** (**EOL**) in January 2020, meaning it no longer received official support or updates, encouraging users to migrate to Python 3.
- **Python 3.9 and beyond:** Python 3.9, released in October 2020, continued to refine and expand the language with new features and improvements. Python's development continues with a commitment to maintaining backward compatibility while adding new capabilities.

On September 7, 2022, four new releases were made due to a potential **denial-of-service attack**: 3.10.7, 3.9.14, 3.8.14, and 3.7.14.

Notable changes from 3.10 include increased program execution speed and improved error reporting.

Since 27 June 2023, Python 3.8 is the oldest supported version of Python (albeit in the *security support* phase), due to Python 3.7 reaching **end-of-life**.

The first release candidate of Python 3.12 was offered on 6 August 2023.

Python's history and success are closely tied to its vibrant community and ecosystem of libraries and frameworks, making it a powerful and versatile tool for developers in various fields.

Python's history is characterized by a commitment to simplicity, readability, and backward compatibility. Its evolution continues, with Python remaining a popular and influential programming language in various domains.

Birth of Python

Python was created by *Guido van Rossum*, a Dutch programmer, in the late 1980s. *Guido* was working at the **Centrum Wiskunde and Informatica** (**CWI**) in the Netherlands when he started developing Python. His motivation was to create a language that emphasized code readability and allowed programmers to express concepts in fewer lines of code.

The project officially began in December 1989, and the first public release, Python 0.9.0, was introduced in February 1991. *Guido*'s choice of the name *Python* was inspired by the British comedy group *Monty Python*, whose work he enjoyed. The birth of Python can be traced back to the late 1980s and early 1990s. Here is a brief account of its origin:

- **Python's name:** The name *Python* was inspired by *Guido's* fondness for the British comedy group *Monty Python's Flying Circus*. It was chosen as a name that was both unique and memorable.
- **Design philosophy:** *Guido van Rossum* designed Python with a strong emphasis on code readability, using significant whitespace (indentation) to define code blocks rather than relying on explicit braces or keywords. This design philosophy, often referred to as the *Zen of Python*, is encapsulated in the **Python Enhancement Proposal** (**PEP**) 20 and has guided the language's development.

Python's simplicity, readability, and ease of use contributed to its gradual rise in popularity over the years. It gained a reputation as a versatile and beginner-friendly programming language, making it suitable for a wide range of applications, from web development and scientific computing to data analysis and Artificial Intelligence. *Guido van Rossum*'s vision and ongoing contributions, along with the vibrant Python community, have played pivotal roles in Python's success and continued evolution.

Guido's design philosophy

Guido van Rossum, the creator of Python, had a well-defined design philosophy that guided the development of the language. This philosophy, often referred to as the *Zen of Python*, is a set of guiding principles and aphorisms that capture the essence of Python's design and philosophy. These principles are encapsulated in PEP 20 which is titled *The Zen of Python*. Here are some key elements of Guido's design philosophy:

- **Readability counts:** One of the most fundamental principles of Python's design is that code should be easy to read and understand. This is reflected in the use of indentation to define code blocks and the avoidance of excessive punctuation.
- **Beautiful is better than ugly:** Python encourages writing code that is aesthetically pleasing and elegant. Code should not only work but also be well-structured and expressive.
- **Explicit is better than implicit:** Python favors clarity over cleverness. It is better to be explicit in your code, making it clear what you are doing, rather than relying on implicit or obscure techniques.
- **Simple is better than complex:** Python strives for simplicity in both language design and code. It values straightforward and understandable solutions over overly complex ones.
- **Complex is better than complicated:** While simplicity is important, Python recognizes that some problems are inherently complex. In such cases, Python encourages providing a clear and manageable way to deal with complexity without adding unnecessary complication.

- **Flat is better than nested:** Python promotes flat code structures over deeply nested ones. This principle helps maintain code readability and avoids excessive levels of indentation.
- **Sparse is better than dense:** Python prefers code that is spaced out and easy to visually parse. It encourages using whitespace judiciously to enhance code readability.
- **Errors should never pass silently:** Python emphasizes the importance of error handling. When something goes wrong, Python encourages raising explicit exceptions rather than silently ignoring errors.
- **In the face of ambiguity, refuse the temptation to guess:** Python avoids making assumptions when code is ambiguous. It prefers to raise exceptions or require explicit instructions to resolve ambiguity.
- **There should be one, and preferably only one way to do it:** Python discourages unnecessary redundancy and multiple ways to accomplish the same task. It promotes a single, clear, and canonical way of achieving a particular goal.
- **Now is better than never:** Python values progress and encourages developers to take action rather than waiting for a perfect solution.
- **If the implementation is hard to explain, it is a bad idea:** Python favors code that is easy to explain and understand. Complex, convoluted solutions should be avoided.

These principles reflect *Guido van Rossum*'s vision for Python as a language that is not only powerful and versatile but also easy to learn, read, and maintain. The Zen of Python has been a cornerstone of Python's success and continues to influence its development and community practices. You can view the complete *Zen of Python* by typing import this in a Python interpreter.

Python 1.0 and early growth

Python 1.0 was a significant milestone in the early growth and development of the Python programming language. It introduced several key features and improvements that contributed to Python's popularity. Here is an overview of Python 1.0 and its impact:

Release date: Python 1.0 was officially released in January 1994.

The key features and changes are as given as follows:

- **Lambda functions:** Python 1.0 introduced lambda functions, also known as anonymous functions. These functions allow you to define small, unnamed functions for use in expressions.

- **Map, filter, and reduce:** Python 1.0 included the map, filter, and reduce functions, which are powerful tools for working with lists and sequences. They allowed for more concise and expressive code.
- **Exception handling:** Exception handling was improved in Python 1.0, making it easier to handle and recover from errors and exceptional situations in code.
- **Simple and powerful data types:** Python 1.0 retained its simple and consistent data types, including integers, floats, strings, lists, and dictionaries. These data structures were easy to work with and provided a solid foundation for writing Python programs.

Early growth and community adoption

Python 1.0 marked a significant step in the growth of Python, and it started gaining attention and adoption for several reasons:

- **Simplicity**: Python's design philosophy, emphasizing code readability and simplicity, appealed to many developers. It was seen as an accessible and easy-to-learn language.
- **Versatility**: Python's versatility allowed it to be used for a wide range of applications, from scripting and automation to web development and scientific computing.
- **Community**: Even in its early days, Python had an active and supportive community. This community contributed to the language's growth and development, creating a sense of camaraderie among Python enthusiasts.
- **Cross-platform**: Python's cross-platform nature meant that code written in Python could run on different operating systems without major modifications.
- **Educational use**: Python's readability made it an excellent choice for teaching programming, and it started to gain popularity in educational settings.

Python's growth continued in subsequent versions, with each release adding more features and improvements. The simplicity, readability, and versatility of Python continued to attract developers from various backgrounds and industries, leading to its widespread adoption and the emergence of a strong Python ecosystem.

Python 2 versus Python 3

Python 2 and Python 3 are two major versions of the Python programming language. While Python 2 was the dominant version for many years, Python 3 was introduced to address some of its limitations and design issues. Here is a comparison of Python 2 and Python 3:

- **Python 2:** Python 2 was a major release of the Python programming language and was officially released on October 16, 2000. It was a continuation of the Python

1.x series with some significant improvements and changes. Python 2 introduced several features and enhancements, but it also faced challenges related to certain design decisions that were later addressed in Python 3. Here are some key aspects of Python 2:

- **Legacy version:** Python 2 was released in October 2000 and was the dominant version of Python for a long time. The final release in the Python 2 series was Python 2.7, released in July 2010.
- **Print statement:** In Python 2, the print statement was used for printing to the console. For example, print `Hello, World!`.
- **Division:** The division of integers in Python 2 used floor division by default, meaning that 5/2 would result in 2 instead of 2.5. To perform true division, you needed to explicitly convert one of the operands to a float, for example, 5.0/2.
- **Unicode**: Python 2 had limited support for Unicode. Handling non-ASCII characters and text encoding/decoding required extra care.
- ** **xrange():**: Python 2 introduced the `xrange()` function, which is more memory-efficient than `range()` when iterating over large ranges.
- **Iteration:** Python 2 used iterators and the `next()` function for iteration.
- **Relative imports:** Relative imports in Python 2 were more permissive, which could lead to ambiguity in larger codebases.

- **Python 3:** Python 3 is the latest major version of the Python programming language and was first released on December 3, 2008. It was designed to address and rectify certain design flaws and inconsistencies present in Python 2. It introduced several new features, optimizations, and improvements while maintaining backward compatibility. Here are some key aspects of Python 3:
  - **Modern version:** Python 3 was introduced to address and rectify some of the design flaws and inconsistencies in Python 2. It was first released in December 2008.
  - **Print function:** In Python 3, the print statement was replaced with the `print()` function, making it more consistent with other function calls. For example, print(`Hello, World!`).
  - **Division:** Python 3 introduced true division by default, meaning that 5/2 results in 2.5. To perform floor division, you use //, e.g., 5//2.
  - **Unicode:** Python 3 has robust support for Unicode by default. Text strings are Unicode by default, making it easier to work with non-ASCII characters.
  - **range():** Python 3's `range()` function behaves like Python 2's `xrange()`, providing more efficient memory usage for iteration.

- **Iteration:** Python 3 introduced the `next()` method as the `__next__()` method of iterators, making it more consistent and Pythonic.
- **Relative imports:** Python 3 tightened the rules for relative imports to improve code clarity and avoid ambiguity.

Migration from Python 2 to Python 3

Python 2 reached its EOL on January 1, 2020. This means it no longer received official support or updates. As a result, the Python community and many organizations migrated their codebases from Python 2 to Python 3. The transition required updating code to be compatible with Python 3's syntax and features. Various tools and guides are available to facilitate this migration.

So, Python 2 and Python 3 have several key differences, with Python 3 being the modern and recommended version due to its improvements, particularly in terms of Unicode support and code clarity. Developers are strongly encouraged to use Python 3 for all new projects and to migrate existing Python 2 code to Python 3 to ensure long-term compatibility and support.

Python's ongoing evolution

Python's ongoing evolution is a testament to its commitment to staying relevant and meeting the changing needs of the developer community. Python continues to grow and improve through regular releases and enhancements. As of the update in September 2021, here are some key aspects of Python's ongoing evolution:

- **Regular releases:** Python follows a predictable release schedule, with new versions typically released in October of each year. These releases introduce new features, improvements, and optimizations. Users can expect a steady stream of updates and enhancements to the language.
- **PEP Process: Python Enhancement Proposals** (**PEPs**) play a crucial role in shaping Python's future. The PEP process allows developers to propose and discuss changes to the language and its standard library. Many new features and enhancements are proposed and developed through this process.
- **Syntax enhancements:** Python's syntax continues to evolve, making the language more expressive and user-friendly. New syntax features are introduced, such as the *walrus operator* (:=) in Python 3.8, which allows for inline variable assignment within expressions.
- **Type hints:** Python's type hinting system, introduced in Python 3.5, has gained traction and is increasingly used for static analysis and improved code documentation. Type hinting has seen enhancements and increased adoption in the Python ecosystem.
- **Asyncio and concurrency:** Python's support for asynchronous programming, introduced with the async and await keywords in Python 3.5, has continued to