AlmightyPythonBook

Guy Tordjman

2025-02-07

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

Markdown allows you to write using an easy-to-read, easy-to-write plain text format.

1 Introduction

Welcome to **Almighty Python: Master Python by Building Real Applications**, your ultimate guide to becoming a professional Python developer! Whether you're taking your first steps into the world of coding or you're an experienced developer looking to level up your Python skills, this book is designed for you.

In today's world, Python is everywhere. From powering the biggest tech giants to enabling groundbreaking innovations in artificial intelligence (AI), web development, data science, and automation—it's no surprise that Python has become one of the most popular and sought-after programming languages. This book is your companion on a journey to master Python while building practical, real-world applications along the way.

1.1 Why This Book?

Learning to code can feel overwhelming at first. There are countless tutorials, courses, and articles, each offering bits and pieces of knowledge. What makes this book different? It's simple: **you'll learn by building**. We won't just throw theory at you—we'll guide you through creating a range of real applications, starting from basic command-line tools to full-fledged web apps and machine learning projects.

By the time you reach the final chapters, you won't just "know Python"; you'll have the skills and confidence to build your own applications and solve real-world problems with code.

1.2 Who Is This Book For?

- **Absolute Beginners**: If you've never written a line of code, don't worry. This book will ease you into programming concepts, explaining everything in plain, simple language.

- **Intermediate Developers**: If you've dabbled in Python before, you'll find plenty of new concepts, advanced techniques, and hands-on projects that will push your skills to the next level.

- **Future Python Professionals**: This book will help you build the foundation for a successful career as a Python developer by covering everything from Python basics to advanced topics like object-oriented programming, web development, databases, and machine learning.

1.3 What You'll Learn

Here's a glimpse of what you'll build and learn in this book:

- **Python Fundamentals**: Learn the core concepts of Python by building real applications.

- **APIs and Automation**: Build applications that connect to external services and automate repetitive tasks.

- **Object-Oriented Programming (OOP)**: Create more structured and maintainable code by mastering OOP.

- **Web Development**: Build web applications using Python frameworks.

- **Data Science and Machine Learning**: Explore how Python is used in data analysis and AI with hands-on examples.

Each chapter introduces a new set of skills, guiding you through writing actual code, fixing common problems, and understanding how everything works under the hood. Along the way, you'll gain insight into how Python is used in the real world.

1.4 How to Use This Book

Each chapter is designed to build on the previous one. While you're welcome to jump around, we recommend following the chapters in order, especially if you're a beginner. Don't rush—take your time to understand the concepts and code, and don't hesitate to experiment with your own ideas. The best way to learn is by doing.

This book also pairs with a series of online courses offered on **Udemy**, giving you the opportunity to watch code in action and practice what you learn in an interactive environment.

1.5 A Quick Word About Python in the AI Era

You're learning Python at the perfect time. With the rise of artificial intelligence and automation, Python is more relevant than ever. It's the language of choice for AI developers, data scientists, and automation engineers. This book will give you the tools to start building smart, AI-driven applications while also preparing you for the future of programming.

1.6 Ready to Get Started?

Let's dive right in! The first chapter will introduce you to Python, why it's the perfect language for beginners, and how to set up your Python environment. From there, you'll start writing your first lines of Python code and building your first application. Remember, the goal is to **learn by doing**—so get ready to write lots of code and have fun along the way!

See Knuth (1984) for additional discussion of literate programming.

Part I

Part 1 - Getting Started

2 Introduction to Python

2.1 What is Python? And Why Even Bother Learning It Now (In the AI Era)?

When you go and look for a formal definition, you'll probably see something like:

"Python is a general-purpose, high-level, interpreted programming language created by Guido van Rossum and first released in 1991. It is known for its simplicity and readability, making it a great choice for beginners as well as experienced developers."

Well, if you're new to programming, this definition probably didn't tell you much. Since this book is intended for readers at all programming levels, we will break down each new term in a simple and clear way. Sometimes, if necessary, we'll explain things more than once—because understanding the basics well is key to mastering Python. Let's go over some of the important keywords in this definition in a way that actually makes sense:

Language: Python is a programming language because it gives you a way to communicate with computers. You write Python code to tell the computer what to do, just like you'd tell a friend what to do using words.

General-purpose: Python is a general-purpose language, meaning you can use it for pretty much anything. Whether you want to build a website, analyze data, make games, automate tasks, or even explore artificial intelligence, Python (and this book) has got you covered.

High-level: Python is called a high-level language because it's designed to be easy for humans to read and write. Unlike low-level languages that deal directly with how the computer works, Python allows you to focus on solving problems without worrying about technical details like memory management. This makes coding much simpler and more beginner-friendly.

This is actually a good place to have our first glance at some code. Let's say you would like to write a program that interacts with a user by asking for its name.

In python as it is a High-level language this can be a simple one liner code:

name = input("Enter your name:")

Now lets see what the same program looks like when it is implemented in **C** which is a Low-level language :

> **Note how long and much more complicated it is!**

The main reason for this extra lines of code is that a low-level language gives us more control over hardware and system resources, hence it requires you - the programmer, to also think about memory management, data types, explicit error handling, and more.

Python as a High-Level language is designed to abstract away most of the complexities of dealing with the machine or hardware. You can focus on writing logic, rather than worrying about memory management, compilation, or system-specific details. The language is forgiving and lets you focus on what you want to achieve without dealing with low-level operations.

Interpreted: Python is an interpreted language, meaning the computer reads and runs your code one step at a time instead of processing everything at once before running it. This makes testing and debugging easier but can sometimes make Python a bit slower than languages that compile everything first.

2.2 Why Learn Python Now? (The AI Connection)

We're living in the AI era, where artificial intelligence is transforming industries at an unprecedented pace. From self-driving cars to chat bots, AI is everywhere—and Python is at the heart of it all.

Python is the go-to language for AI and machine learning because of its vast ecosystem of tools (libraries), such as TensorFlow, PyTorch, and Scikit-learn (no worries we will learn about libraries in the next lessons). These tools make it easier to build intelligent systems without having to write complex code from scratch.

But AI isn't the only reason to learn Python. Python's simplicity, versatility, and beginner-friendly nature make it a perfect starting point for anyone who wants to step into the world of programming, whether your goal is to get into AI, web development, automation, or just solve everyday problems with code.

In the next section, we'll explore exactly why Python is considered one of the most beginner-friendly programming languages—and why it's often recommended as the first language to learn.

2.3 Why is Python Beginner-Friendly?

Simple and Clear Syntax Python's syntax is often described as "elegant" because of how clean and straightforward it is. If you're just starting out with programming, Python lets you focus on learning core programming concepts, such as variables, loops, and functions, without getting bogged down by complex syntax rules. For example, here's how you would write a program that print "Hello, World!" to the screen in Python:

```
print("Hello, World!")
```

```
Hello, World!
```

Compared to other High-level languages like C++ or Java, where you'd have to write more boilerplate code, Python's version is minimal, allowing you to start writing meaningful code right away.

Readable Code Python places a strong emphasis on code readability. Its use of indentation (rather than curly braces in many other languages) to define code blocks makes Python programs visually cleaner and easier to follow. This structure makes Python ideal for beginners because it encourages the creation of clean and well-organized code, which is key to developing good programming habits early on.

Dynamic Typing Python is dynamically typed, which means you don't need to specify the type of variable (like an integer or string) when you declare it. You can directly assign a value to a variable, and Python will figure out the type for you. For example:

```
x = 5           # x is automatically an integer
name = "Alice"  # name is automatically a string
```

This is different from languages like Java or C++, where you have to explicitly define the type of each variable. Dynamic typing makes Python more flexible and allows you to focus on solving problems rather than worrying about types.

Large, Supportive Community Python has one of the largest and most active programming communities in the world. Whether you need help troubleshooting an issue or want to learn about the latest Python libraries, there are countless forums, tutorials, and documentation available to help you. The Python community is known for being friendly and welcoming to newcomers, and you'll find plenty of resources to guide you every step of the way as you start learning.

Extensive Libraries and Frameworks Another reason Python is beginner-friendly is because of its vast selection of libraries and frameworks. These pre-built tools allow you to avoid reinventing the wheel and instead focus on building your applications faster. For example, if

you want to build a website, you can use frameworks like Django or Flask. If you're interested in data science, there are powerful libraries like Pandas, NumPy, and Matplotlib to help you manipulate and visualize data. These libraries are designed to be easy to use and can dramatically speed up your development process.

Cross-Platform Compatibility Python is a cross-platform language, meaning you can run your Python code on any major operating system, such as Windows, macOS, and Linux, without having to make any changes. This makes Python a versatile choice for developers who want to build applications that can work across multiple platforms.

3 Setting Up Python on Your Machine

Before we can start writing Python code, we need to make sure that Python is installed and properly set up on your computer. Whether you're using **Windows**, **macOS**, or **Linux**, the process is straightforward, but there are some important details to take note of.

3.1 Step 1: Check if Python is Already Installed

First, let's check if Python is already installed on your machine. This can be done through your terminal or command prompt.

1. **Windows** - open the **Command Prompt** by pressing `Win-R`, typing `cmd`, and hitting `Enter`. **macOS** - open the **Terminal** by pressing `Cmd+Alt+T`. **Linux** - open the **Terminal** by pressing `Ctrl+Alt+T`.
2. Type the following command and press Enter:

   If Python is installed, you should see something like:

   If not, you'll see an error message telling you that `python` is not recognized.

> Tip
>
> - **Don't worry if the version number isn't exactly what you see in the examples.** It's perfectly fine if you see **Python 3.11** or **Python 3.12**—those will work great for this book!
> - **What's important is that you're using Python 3.10 and higher.** Avoid using Python versions **lower than 3.10**, as some newer features or improvements might not work properly in older versions.

3.2 Step 2: Installing Python

If Python is not already installed or if you need to install a different version, follow the steps below:

3.2.1 Windows

1. Visit the official Python website.
2. Click on the "Download Python" button (make sure to select version 3.x).
3. Run the installer and **check the box that says "Add Python to PATH"** before clicking "Install Now".
4. Once installed, verify by running `python --version` in the Command Prompt.

3.2.2 macOS

macOS often comes with Python pre-installed. However, it's typically an older version, so it's recommended to install the latest version of Python 3.

1. You can install Python using Homebrew (a popular package manager for macOS):

   ```
   brew install python
   ```

2. Alternatively, you can download the latest Python installer from the Python website.
3. Verify installation by running:

   ```
   python3 --version
   ```

3.2.3 Linux

Most Linux distributions come with Python pre-installed, but you can install or update to the latest version using the package manager.

1. Open your terminal.
2. For **Debian-based** distributions (like Ubuntu), run:

   ```
   sudo apt update
   sudo apt install python3
   ```

3. For **Red Hat-based** distributions (like Fedora), run:

```
sudo dnf install python3
```

4. Verify installation by running:

```
python3 --version
```

3.3 Step 3: Installing the Python Extension in VSCode

Now that you have VSCode installed, it's time to make it Python-ready! VSCode is a powerful editor, but it can't run or understand Python by itself. This is where **extensions** come in. Extensions are small add-ons that extend the functionality of VSCode to support specific tasks or languages. For Python, the extension we need is called the **Python extension**.

3.3.1 Installing the Python Extension:

1. Open VSCode.
2. On the left sidebar, click on the **Extensions** icon, or press `Ctrl+Shift+X` (`Cmd+Shift+X` on macOS) to open the Extensions view.
3. In the search bar, type **Python**.
4. The **Python extension by Microsoft** should be the first result. Click **Install**.

The Python extension provides key features like: - Syntax highlighting (color-coding your Python code) - Code completion (helping you type faster and avoid errors) - Integrated debugging (for fixing bugs in your code) - Running Python code directly in the editor

After installing the Python extension, VSCode will be ready to run your Python code.

3.4 Step 4: Running Your First Python Script in VSCode

Now that we have the Python extension installed, let's test everything to make sure it's working!

1. Open VSCode and create a new file called `hello.py`.
2. Type the following code:

```
print("Hello, Almighty Python")
```

3. Save the file by hitting `Ctrl+S` or `Cmd+S`.
4. To run your script, follow these steps:
    - Open the integrated terminal in VSCode (go to **View** > **Terminal** or press `Ctrl+`).
    - In the terminal, type the following:

```
python hello.py
```

You should see the following output in the terminal:

Hello, Almighty Python

Congratulations! You've just written and executed your first Python program.

3.5 Step 5: Common Problems and Troubleshooting

3.5.1 Problem 1: `python` Command Not Found

Solution: Python might not be added to your system's PATH.
- **Windows:** Reinstall Python and check the option to "Add to PATH" during installation.
- **macOS/Linux:** Try using `python3` instead. If that works, create an alias (`alias python=python3`).

3.5.2 Problem 2: VSCode Doesn't Detect Python

Solution:
1. Open the Command Palette by pressing `Ctrl+Shift+P` or `Cmd+Shift+P`. 2. Type `Python: Select Interpreter` and choose the Python version you installed.

3.5.3 Problem 3: Code Runner Doesn't Work

Solution: Ensure you've installed the **Python extension** by searching for and installing it from the Extensions view. Save your file with the `.py` extension.

3.5.4 Problem 4: Terminal Shows Permission Denied (macOS/Linux)

Solution: Check file permissions. Use `chmod +x filename.py` to make the file executable.

> **Note**
>
> Now that your environment is ready, you're all set to begin your Python journey! In the next section, we'll explore the basics of Python syntax and build your first real-world application. Let's get coding!

4 Setting Up the Coding Environment

In this chapter, we'll set up everything you need to start coding in Python. To make things simple and fun, we'll use **Visual Studio Code (VSCode)**—a free, lightweight, and powerful editor. If you're new to coding, don't worry. We'll guide you step by step. By the end of this chapter, you'll be all set to write and run your first Python program!

Although we'll focus on using VSCode, feel free to use another editor if you prefer. The steps might differ slightly, but the core concepts remain the same.

4.1 Step 1: Installing VSCode

1. Go to the Visual Studio Code website.
2. You'll see download options for Windows, macOS, and Linux. Click the one for your operating system.
3. Once the download is complete, open the installer and follow the instructions.

> Tip
>
> - **Windows:** Make sure you check the option to "Add to PATH" during installation. This will make it easier to use VSCode from the command line.
> - **macOS:** Drag the VSCode icon into your Applications folder.
> - **Linux:** Depending on your distribution, you might need to install it using a package manager (`apt`, `yum`, or `dnf`).

Once installed, launch VSCode. You should see the welcome screen!

> Tip
>
> Pin VSCode to your taskbar or dock for easy access.

4.2 Step 2: Verify your Python installation

> **Note**
>
> You may skip this step if you have previously verified python on your machine.

Python is the language we'll be using throughout this book. Let's make sure it's installed.

1. Open a terminal (Command Prompt on Windows, Terminal on macOS/Linux).
2. Type the following command:

   If Python is installed, you'll see something like:

   If not, you'll see an error message telling you that `python` is not recognized. Please refer to the python installation section of this book and return here when done.

4.3 Step 3: Installing the Python Extension in VSCode

Now that you have VSCode installed, it's time to make it Python-ready! VSCode is a powerful editor, but it can't run or understand Python by itself. This is where **extensions** come in. Extensions are small add-ons that extend the functionality of VSCode to support specific tasks or languages. For Python, the extension we need is called the **Python extension**.

4.3.1 Installing the Python Extension:

1. Open VSCode.
2. On the left sidebar, click on the **Extensions** icon, or press `Ctrl+Shift+X` (`Cmd+Shift+X` on macOS) to open the Extensions view.
3. In the search bar, type **Python**.
4. The **Python extension by Microsoft** should be the first result. Click **Install**.

The Python extension provides key features like: - Syntax highlighting (color-coding your Python code) - Code completion (helping you type faster and avoid errors) - Integrated debugging (for fixing bugs in your code) - Running Python code directly in the editor

After installing the Python extension, VSCode will be ready to run your Python code.

4.4 Step 4: Running Your First Python Script in VSCode

Now that we have the Python extension installed, let's test everything to make sure it's working!

1. Open VSCode and create a new file called `hello.py`.
2. Type the following code:

```
print("Hello, Almighty Python!")
```

3. Save the file (`Ctrl+S` or `Cmd+S`).
4. To run your script, follow these steps:
    - Open the integrated terminal in VSCode (go to **View** > **Terminal** or press `Ctrl+`).
    - In the terminal, type the following:

```
python hello.py
```

You should see the following output in the terminal:

```
Hello, Almighty Python!
```

pgsql Copy Edit

Congratulations! You've just written and executed your first Python program.

4.5 Step 5: Common Problems and Troubleshooting

4.5.1 Problem 1: `python` Command Not Found

Solution: Python might not be added to your system's PATH.
- **Windows:** Reinstall Python and check the option to "Add to PATH" during installation.
- **macOS/Linux:** Try using `python3` instead. If that works, create an alias (`alias python=python3`).

4.5.2 Problem 2: VSCode Doesn't Detect Python

Solution:
1. Open the Command Palette (`Ctrl+Shift+P` or `Cmd+Shift+P`).
2. Type `Python: Select Interpreter` and choose the Python version you installed.

4.5.3 Problem 3: Code Runner Doesn't Work

Solution: Ensure you've installed the **Python extension** by searching for and installing it from the Extensions view. Save your file with the `.py` extension.

4.5.4 Problem 4: Terminal Shows Permission Denied (macOS/Linux)

Solution: Check file permissions. Use `chmod +x filename.py` to make the file executable.

> **Note**
>
> Now that your environment is ready, you're all set to begin your Python journey! In the next section, we'll explore the basics of Python syntax and build your first real-world application. Let's get coding!

5 Python Syntax and Variables

In this chapter, we'll cover the basics of Python's syntax, data types, and how to use variables. These concepts are essential for writing your first Python programs. Remember, the best way to learn is by coding!

> **Tip**
>
> Open your VSCode, create a new Python file (e.g., `variables.py`), and type the examples from this chapter. Modify the code, play with it, and see what happens. Experimentation is key!

5.1 Comments in Python

Comments are lines in your code that Python ignores when running the program. They are used to explain what the code does, making it easier to understand for yourself and others. Comments can also be used to temporarily disable parts of your code.

5.1.1 Single-line Comments

In Python, single-line comments start with the `#` symbol.

```
# This is a comment
print("Hello, Almighty Python!")  # This comment is ignored
```

```
Hello, Almighty Python!
```

5.1.2 Multi-line Comments

Python doesn't have a specific multi-line comment syntax, but you can use triple quotes (''') to create a block of comments.

```
'''
This is a multi-line comment.
You can use it to write longer explanations
or to temporarily disable parts of your code.
'''
print("Multi-line comments are helpful!")
```

```
Multi-line comments are helpful!
```

5.1.3 Why Use Comments?

- **Explain your code:** Help yourself or others understand what the code does.
- **Debugging:** Comment out parts of the code to isolate problems.
- **Documentation:** Make your code more readable and easier to maintain.

5.2 Variables and Assignment

In Python, variables are used to store data. You assign a value to a variable using the = symbol. The variable name should be descriptive and follow certain rules (e.g., no spaces or special characters, should not start with a number). Variables have type that depends on the type of data they store. We'll talk more about data types shortly but for now, take a look at these four variables and note the different types assigned to them.

```
name = "Almighty Python"
age = 25
price = 19.99
is_cool = True

print(name)
print(age)
print(price)
print(is_cool)
```

```
Almighty Python
25
19.99
True
```

5.3 Basic Data Types: String, Integer, Float, Boolean

5.3.1 String (`str`)

Strings represent text and are enclosed in quotes (`'single'` or `"double"`).

> **Tip**
>
> Use triple quotes (`"""` or `'''`) for multi-line text.

```
name = 'Almighty Python - Learning about strings'
question = "What is Python?"
description = """It is a powerful and beginner-friendly programming
language that supports writing long texts on multiple lines like so."""

print(name)
print(question)
print(description)
```

```
Almighty Python - Learning about strings
What is Python?
It is a powerful and beginner-friendly programming
language that supports writing long texts on multiple lines like so.
```

5.3.2 String Concatenation and Repetition

You can combine strings using the `+` operator or repeat them using `*`.

```
first_name = "Almighty"
last_name = "Python"
full_name = first_name + " " + last_name
print(full_name)

repeated = "Hi! " * 3
print(repeated)
```

```
Almighty Python
Hi! Hi! Hi!
```

> **!** Important
>
> You can do many more operations with strings. Actually string operations are covered in their own section - definitely something to look forward to!

5.3.3 Integer (`int`)

An integer is a whole number without a decimal point. Integers can be positive, negative, or zero.

```
age = 25
year = 2025
balance = -100

print(age + 5)       # Addition
print(year - 2000)   # Subtraction
print(balance * 2)   # Multiplication
```

```
30
25
-200
```

5.3.4 Float (`float`)

A float is a number with a decimal point. It is used for precise values like measurements, scientific calculations, or financial data.

```
price = 19.99
temperature = -10.5
pi = 3.14159

print(price * 2)
print(temperature + 5)
print(pi + pi)
```

```
39.98
-5.5
6.28318
```

5.3.5 Boolean (`bool`)

Booleans represent logical values: True or False. They are used for decision-making and comparisons in your programs.

```
is_python_fun = True
is_sky_green = False

print(is_python_fun)
print(is_sky_green)
```

```
True
False
```

5.3.6 Type converstion and interrogation

TODO

5.4 Summary

In this chapter, you've learned about Python's syntax, basic data types, variables, and how to use comments. These are fundamental concepts that will help you build a strong foundation for writing Python programs.

Key Takeaways:

- Python has four main basic data types: `str`, `int`, `float`, and `bool`.
- Use variables to store data and make your code more readable.
- Comments improve code clarity and help with debugging.

> **Tip**
>
> Remember: Keep experimenting! The best way to learn is by trying different things in your code. Modify the examples, create your own, and observe the results.

6 Basic Operators in Python

Before we jump into coding, let's take a moment to understand **operators**—what they are, what they do, and why they are essential in programming.

6.1 What Are Operators? Why Do We Need Them?

Operators are special symbols in Python that allow us to perform computations and manipulate data. They are the building blocks of expressions and help us execute mathematical operations, assign values to variables, compare values, and even make logical decisions in our programs. Without operators, coding would be extremely limited, as we wouldn't be able to perform calculations, check conditions, or control program flow effectively.

6.1.1 Basic Operators

These fundamental operators are widely used in almost every Python program:

- **Arithmetic Operators** – Perform mathematical operations like addition, subtraction, multiplication, and division (`+`, `-`, `*`, `/`).
- **Assignment Operators** – Assign values to variables (`=`, `+=`, `-=`).
- **Comparison Operators** – Compare two values and return `True` or `False` (`==`, `!=`, `<`, `>`).
- **Logical Operators** – Combine boolean values and control flow (`and`, `or`, `not`).

6.1.2 Advanced Operators

Some operators require a deeper understanding of programming concepts and will be introduced later in the book:

- **Bitwise Operators** – Perform operations at the binary level (`&`, `|`, `^`, `~`).
- **Membership Operators** – Check if a value exists within a sequence (`in`, `not in`).

- **Identity Operators** – Compare memory locations of two objects (`is`, `is not`).

We will begin by exploring **arithmetic operators**, followed by **assignment operators**, as they are foundational to understanding more advanced operations in Python.

6.2 Arithmetic Operators in Python

Arithmetic operators are the most basic and frequently used operators. They allow you to perform mathematical calculations such as addition, subtraction, multiplication, and division. Let's go over each one with examples to see how they work.

6.2.1 Addition (+)

The addition operator is used to add two values. It works with numbers (integers and floats) and even with strings (by concatenating them).

```
# Integer addition
a = 10
b = 5
result = a + b
print(result)
```

```
15
```

```
# Float addition
x = 10.5
y = 4.3
result = x + y
print(result)
```

```
14.8
```

```
# String addition (concatenation)
greeting = "Hello, "
name = "Almighty Python!"
message = greeting + name
print(message)
```

```
Hello, Almighty Python!
```

6.2.2 Subtraction (-)

The subtraction operator (-) is used to subtract one value from another. It can also be used with both integers and floats.

```
# Integer Subtraction
a = 10
b = 3
result = a - b
print(result)

result = b - a
print(result)
```

```
7
-7
```

```
# Float Subtraction
a = 10
b = 3
result = a - b
print(result)

result = b - a
print(result)
```

```
7
-7
```

```
# Mixed Type Subtraction
a = 2
b = 1.2
result = a - b
print(result)

result = b - a
print(result)
```

```
0.8
-0.8
```

6.2.3 Division (/)

The division operator is used to divide one number by another. In Python, it always returns a float value, even if the result is a whole number.

```
# Integer division
a = 10
b = 5
result = a / b
print(result)  #(Note that it's a float)

# Float division
x = 10.5
y = 4.3
result = x / y
print(result)
```

```
2.0
2.441860465116279
```

6.2.4 Floor Division (//)

The floor division operator returns the largest integer less than or equal to the result of division. It essentially rounds down the result.

```
# Integer floor division
a = 10
b = 3
result = a // b
print(result)

# Float floor division
x = 10.5
y = 4.3
result = x // y
print(result)
```

```
3
2.0
```

6.2.5 Modulus (%)

The modulus operator returns the remainder of the division of two numbers. This is useful for determining if a number is divisible by another or for wrapping around values in cyclic operations (like circular buffers).

```
# Integer modulus
a = 10
b = 3
result = a % b
print(result)  # 10 divided by 3 leaves a remainder of 1

# Float modulus
x = 10.5
y = 4.3
result = x % y
print(result)  # 10.5 divided by 4.3 leaves a remainder of 1.9
```

```
1
1.9000000000000004
```

6.2.6 Exponentiation (**)

The exponentiation operator raises the first number to the power of the second number.

```
# Integer exponentiation
a = 2
b = 3
result = a ** b # 2 raised to the power of 3
print(result)

# Float exponentiation
x = 2.5
y = 5.5
result = x ** y # 2.5 raised to the power of 5.5
print(result)

# Mixed exponentiation
a = 3
b = 2.5
result = a ** b # (3 raised to the power of 2.5)
```

```
print(result)
```

```
8
154.40808887540913
15.588457268119896
```

6.2.7 Operator Precedence – The Order of Operations

In Python, like in regular mathematics, arithmetic operators have an order in which they are evaluated. This order is crucial for understanding how complex expressions are computed. While we will focus on arithmetic operators for now, keep in mind that the order of operations applies to all operators in Python, and we'll revisit it as we explore more complex operations later in the book.

Here is the order of precedence for the basic arithmetic operators:

1. **Exponentiation (`**`)**
   The exponentiation operator is evaluated first. It raises a number to the power of another.
2. **Multiplication (`*`), Division (`/`), Floor Division (`//`), Modulus (`%`)**
   These operators are evaluated next, from left to right.
3. **Addition (`+`), Subtraction (`-`)**
   Finally, addition and subtraction are evaluated, also from left to right.

6.2.7.1 Example of Operator Precedence

Let's see how this works in action:

```
result = 3 + 2 * 5 - 10 / 2
print(result)
```

6.2.7.2 Here's how the expression is evaluated:

- First, 2 * 5 is computed (multiplication).
- Then, 10 / 2 is computed (division).
- Next, the addition and subtraction are performed from left to right: 3 + (result of multiplication) and then subtracting the division result. Thus, the order is followed strictly as per the rules of precedence, and the result will be 10.0.

6.2.7.3 Parentheses Override the Precedence

You can control the order of operations in your code by using parentheses () to change the default precedence. Anything inside parentheses is computed first.

```
result = (3 + 2) * (5 - 10 / 2)
print(result)
```

```
0.0
```

Here, the addition and subtraction are computed first due to parentheses, followed by multiplication and division.

> **!** Important
>
> This section introduces the general order of arithmetic operations in Python. As we advance and introduce more operators, you'll need to remember that operator precedence determines how expressions are evaluated, and we will come back to this concept when we cover more complex topics in the book.

6.2.8 Using `e` in Arithmetic Operations

In Python, the letter `e` is often used in scientific notation to represent powers of 10. It is not an operator, but rather a part of a number written in exponential form. The `e` can be used in arithmetic operations just like any other number. The expression `2e3` means (2 $\times$10^3), which is equal to 2000.

6.2.8.1 Example:

```
# Using e for scientific notation
result = 2e3  # 2 * 10^3 = 2000
print(result)
```

6.2.8.2 Combining `e` with other Arithmetic Operators

You can also perform arithmetic operations with numbers written in scientific notation. Let's see how this works with different operators:

```
# Adding, subtracting, multiplying and dividing with e notation
add_result = 2e3 + 5e2  # Adding 2000 + 500
sub_result = 5e3 - 1e3  # Subtracting 5000 - 1000
mul_result = 3e2 * 4e1  # Multiplying 300 * 40
div_result = 1e4 / 2e2  # Dividing 10000 / 200
small_value = 1e-4  # A small value, equivalent to 0.0001

print("Addition:", add_result)
print("Subtraction:", sub_result)
print("Multiplication:", mul_result)
print("Division:", div_result)
print("Small value:", small_value)
```

```
Addition: 2500.0
Subtraction: 4000.0
Multiplication: 12000.0
Division: 50.0
Small value: 0.0001
```

6.2.8.3 Exponentiation with `e`

When used with exponentiation (`**`), `e` represents powers of 10, and the operation behaves like normal arithmetic:

```
exp_result = 2e3 ** 2  # 2000^2
print(exp_result)
```

```
4000000.0
```

6.2.8.4 Why Use `e`?

Using `e` allows us to easily handle large or small numbers without typing out all the zeros. It's especially helpful in scientific and engineering calculations where such numbers frequently occur.

6.2.9 Summary of Arithmetic Operators

In this chapter, we explored the core arithmetic operators in Python and how they are used to perform calculations.

6.2.9.1 Key Points

- **Order of Operations**: Python follows a specific order of precedence when evaluating arithmetic expressions. Exponentiation (`**`) is evaluated first, followed by multiplication, division, floor division, and modulus, and finally, addition and subtraction. Parentheses can be used to control the order of evaluation.
- **Using `e` in Arithmetic**: We also discussed how `e` is used in Python to represent numbers in scientific notation (e.g., `2e3` represents 2000). We demonstrated how to perform arithmetic with scientific notation and how `e` can be combined with other operators like addition, subtraction, multiplication, division, and exponentiation.

6.2.9.2 Arithmetic Operators Table

Operator	Description	Example	Output
`+`	Addition: Adds two operands.	`5 + 3`	`8`
`-`	Subtraction: Subtracts the second operand from the first.	`5 - 3`	`2`
`*`	Multiplication: Multiplies two operands.	`5 * 3`	`15`
`/`	Division: Divides the first operand by the second. Returns float.	`5 / 3`	`1.6667...`
`//`	Floor Division: Divides and returns the largest integer less than or equal to the result.	`5 // 3`	`1`
`%`	Modulus: Returns the remainder of a division operation.	`5 % 3`	`2`
`**`	Exponentiation: Raises the first operand to the power of the second.	`2 ** 3`	`8`

6.2.9.3 Important Notes:

- **Integer Division (`//`)** always returns the largest whole number less than or equal to the result of division.
- **Modulus (`%`)** can be used to check divisibility and is commonly used in algorithms like checking if a number is even or odd.
- **Exponentiation (`**`)** is often used for calculating powers, square roots, and mathematical functions.

6.2.9.4 Next Steps

As you continue to experiment with these operators, remember that these foundational concepts will help you tackle more advanced topics later in the book. You will explore more operators and complex expressions as we build on this knowledge.

> Tip
>
> Feel free to modify the examples and experiment with different operations. Create your own expressions and see how Python handles them. The more you play with these operators, the more comfortable you will become using them in real-world applications.

7 Basic Operators and Input

7.1 Arithmetic operators (+, -, *, /)

7.2 Comparison and logical operators

7.3 Taking user input with input()

7.4 Project 1: Simple Temperature Converter

7.5 Convert between Celsius, Fahrenheit, and Kelvin

7.6 Use input from the user and perform arithmetic operations

7.7 Format and display the result

8 ty

9 Basic Input and Output

9.1 Taking user input with input()

9.2 Project 1: Simple Temperature Converter

9.3 Convert between Celsius, Fahrenheit, and Kelvin

9.4 Use input from the user and perform arithmetic operations

9.5 Format and display the result

10 Summary

In summary, this book has no content whatsoever.

References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.

Index