PROGRAMMING FUNDAMENTALS

CHAPTER 1

Introduction to Programming and Python

Dr Lei Wang (Griffith University)

How computers follow instructions and how you begin to think like a programmer

CHAPTER THEME

Programming is not primarily about memorising syntax.

It is about turning a problem into precise instructions that a computer can execute, test and improve.

Student Edition • Week 1

Chapter roadmap

This chapter builds the foundation for the entire course.

You will first understand what programming is and why programming languages exist. You will then meet Python, learn how to run it, and start working with numbers, expressions and basic calculations.

The chapter finishes by introducing an important professional skill: using AI tools such as ChatGPT responsibly while still developing your own programming ability.

Section	What you will learn
1	What programming is and how computers execute instructions
2	Why programming languages exist
3	How programming languages evolved
4	Why there are many languages and why Python is a strong first language
5	Where Python is used and why programming skills matter
6	Python tools: REPL, scripts and the development environment
7	Values, types and numeric literals
8	Operators and expressions
9	Division, floor division, remainder and exponentiation
10	Operator precedence and execution order
11	The type() function and reading results
12	AI-assisted programming: use AI to learn, not to outsource learning
13	Week 1 checklist and self-test

Learning outcomes

By the end of this chapter, you should be able to:

- Explain what programming is and why computers need precise instructions.
- Explain why programming languages provide a human-readable way to describe computations.

- Describe, at a high level, how programming languages evolved from machine code to high-level languages.
- Explain several reasons Python is widely used.
- Identify common areas where Python is used, including AI, data science, web development, automation and robotics.
- Run Python interactively in the REPL and run a Python program from a script.
- Recognise integers and floating-point numbers as different numeric types.
- Distinguish values, literals, operators and expressions.
- Use +, -, *, /, //, %, and ** for arithmetic.
- Predict expression results using operator precedence.
- Use type() to inspect the type of a value or expression.
- Explain why programming knowledge remains essential when using AI coding assistants.

1. What is programming?

Imagine giving a very fast assistant a task but never explaining what to do. Speed does not solve ambiguity. Computers are similar: they can execute instructions extremely quickly, but they do not automatically know what you intended.

A useful definition
Programming is the process of designing, writing, testing and maintaining instructions (programs) that tell a computer how to perform specific tasks.

Programming is behind everyday activities such as:

- sending messages
- browsing websites
- playing games
- booking flights
- artificial intelligence
- robotics
- scientific computing
- automation

Programming is a problem-solving activity

A useful way to think about programming is not “What Python syntax should I type?” but “What problem am I solving, and what precise steps would solve it?”

Human-level problem	Programming interpretation
Calculate distance travelled	Identify quantities and the mathematical operation
Check whether a value is safe	Define the rule precisely
Process many sensor readings	Define repetition

Reuse a calculation	Package the calculation for reuse

KEY IDEA
Programming combines problem analysis, algorithm design, coding, testing, debugging and maintenance. Syntax is only one part of the job.

2. Why do we need programming languages?

Computers ultimately execute low-level machine instructions. Writing large software directly as binary values would be impractical for humans. Programming languages give us a structured, readable way to express instructions.

Stage	Role
Human	Describes the problem and solution
Python program	Expresses the solution using readable Python syntax
Python implementation	Processes the Python program so it can execute
Machine-level execution	The computer carries out the resulting instructions

The exact implementation details depend on the language and Python implementation. For this course, the important idea is that Python lets humans work at a much more useful level than raw machine instructions.

3. From machine code to high-level languages

Programming languages have evolved because programmers need to express increasingly complex ideas while keeping software manageable.

Generation	Approx. period	Examples	Main idea
1st	1940s–1950s	Machine language	Binary instructions; difficult for humans to write and maintain
2nd	1950s	Assembly	Symbolic instructions; closely tied to hardware
3rd	1960s–present	FORTRAN, C, C++, Java, Python	Higher-level, readable, portable and productive

The important lesson is not to memorise the timeline. It is to understand why abstraction matters: higher-level languages allow programmers to describe solutions in ways that are closer to human reasoning.

4. Why are there so many programming languages?

There is no single best programming language for every task. Languages have different strengths, ecosystems, performance characteristics and programming models.

Area	Examples
Operating systems / low-level software	C and related systems languages
Web front-end	JavaScript and related technologies
Mobile applications	Swift / Kotlin
Artificial intelligence and data	Python is especially prominent
High-performance software	C++ and other systems-oriented languages

ANALOGY

Choosing a programming language is like choosing a tool. A carpenter does not use a hammer for every job; a software engineer does not automatically use one language for every problem.

5. Why Python?

Python was created by Guido van Rossum and first released in 1991. It emphasises readability, simplicity and programmer productivity. Today it is used by beginners, researchers, engineers and software teams.

- Readable syntax
- Relatively gentle learning curve
- Versatility across application areas
- Large global community and extensive learning resources
- Rich ecosystem of libraries and frameworks
- Strong use in AI, data science and scientific computing

Where you may encounter Python

Application area	Examples of tasks
AI / machine learning	Model development, experimentation, data processing
Data science	Analysis, visualisation, statistical workflows
Computer vision	Image and video processing
Web development	Servers, APIs and web applications

Automation	Scripts that automate repetitive work
Cybersecurity	Analysis, automation and security tooling
Scientific computing	Numerical and research workflows
Robotics / IoT	Control, sensing and data processing

6. Your first Python tools

6.1 The Python REPL

REPL stands for Read-Eval-Print Loop. It is an interactive environment where Python reads an expression, evaluates it, prints the result and waits for the next instruction.

```
>>> 52
52
>>> 5 + 7
12
>>> type(52)
<class 'int'>
>>> type(5.2)
<class 'float'>
```

On macOS and many Linux environments, Python 3 is commonly started with:

```
python3
```

On Windows, the command may be “python”, depending on the installation.

6.2 Python scripts

A script is a saved Python program, normally with a “.py” extension.

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

For example, from a macOS/Linux terminal:

```
python3 hello.py
```

REPL	Script
Interactive	Saved program
Excellent for quick experiments	Excellent for reusable programs

Results appear immediately	Run the whole program as a unit
Useful for testing ideas	Useful for assignments and projects

GOOD HABIT

Use the REPL to investigate small ideas.

Use scripts to build programs that you want to save, rerun and improve.

7. Values, types and literals

A value is data that Python can work with. A type describes what kind of value it is. A literal is a value written directly in Python source code.

Example	Description	Type
52	Integer literal	int
5.2	Floating-point literal	float
-7	Negative integer value	int
0.0	Floating-point value	float

```
>>> type(52)
<class 'int'>
>>> type(5.2)
<class 'float'>
>>> type(2 + 3)
<class 'int'>
>>> type(2 + 3.0)
<class 'float'>
```

The type of an expression depends on the values and operation involved. Different types support different operations and may produce different kinds of results.

COMMON CONFUSION

25 and "25" are not the same kind of value. The first is an integer; the second is text. We will explore text and user input in more detail in the next chapter.

8. Operators and expressions

An operator tells Python to perform an operation. An expression is something Python can evaluate to produce a value.

Operator	Meaning	Example	Result
+	Addition	7 + 3	10
-	Subtraction	7 - 3	4
*	Multiplication	7 * 3	21
/	Division	7 / 3	2.333...
//	Floor division	7 // 3	2
%	Remainder	7 % 3	1
**	Exponentiation	2 ** 3	8

Expression thinking

```
distance = speed * time
fuel_left = fuel % tank_capacity
power = 2 ** 10
```

Each right-hand side is an expression. Python evaluates it and produces a result that can be displayed, stored, or used in a later expression.

9. The operators students most often confuse

9.1 “/” versus “//”

```
>>> 125000 / 7000
17.857142857142858
>>> 125000 // 7000
17
```

“/” performs ordinary division. “//” performs floor division. For positive numbers, this looks like taking the whole-number part.

For negative values, floor division is not simply “truncate toward zero”; Python floors toward negative infinity.

9.2 “%” means remainder, not percentage

```
>>> 17 % 5
2
>>> 20 % 5
0
```

The remainder operator is useful for questions such as “Is this value divisible by 5?” A number is divisible by 5 when its remainder after division by 5 is zero.

9.3 “**” means exponentiation

```
>>> 2 ** 10
1024
>>> 3 ** 2
9
```

MEMORY TRICK
“*” means multiply; “**” means raise to a power. Two stars are not “extra multiplication”.

10. Operator precedence

Python does not simply evaluate every expression from left to right. Operators have a precedence order. For the arithmetic operators introduced here, multiplication, division, floor division and remainder are evaluated before addition and subtraction.

```
>>> 10 + 3 * 2
16
>>> (10 + 3) * 2
26
```

Parentheses explicitly change the grouping. When an expression is important or potentially confusing, parentheses can make your intention clearer.

Priority	Operators
Higher	**
Next	*, /, //, %
Lower	+ , -

This is a simplified Week 1 arithmetic view. Python's complete precedence table contains additional operators that you will learn later.

A useful prediction habit

Before running code, pause and predict the result. Then run it. If your prediction is wrong, ask which assumption was wrong. This is more valuable than simply seeing the correct output.

11. Execution order: Python goes step by step

A Python script normally executes statements in order from top to bottom. This is one of the most important early mental models.

```
print("A")
print("B")
print("C")
```

Output:

```
A
B
C
```

Later weeks introduce decisions and loops that can change this simple flow. For now, remember: Python follows the rules of the language and processes the program in a defined order.

12. A complete numerical program

```
rocket_speed = 28000
mission_duration = 36

distance = rocket_speed * mission_duration

print("Mission distance:", distance, "km")
```

This small program demonstrates several important ideas:

- “28000” and “36” are numeric literals.
- “rocket_speed” and “mission_duration” are names used to refer to values.
- “rocket_speed * mission_duration” is an expression.
- “distance” stores the resulting value.
- “print()” displays information.
- The program executes from top to bottom.

DESIGN INSIGHT

Readable programs use meaningful names. “rocket_speed” tells the reader much more than a name such as “x”.

13. A first mental model: Input → Process → Output

A large proportion of programs can be understood using a simple three-stage model:

Stage	Question
INPUT	What information does the program need?
PROCESS	What calculations or transformations are needed?
OUTPUT	What result should the program show or produce?

For Week 1, the values may simply be written in the program. In the next weeks, the program will learn to receive information from the user and store it in variables.

LOOKING AHEAD

This Input → Process → Output model will reappear throughout the course and become especially powerful when combined with variables, decisions, loops and functions.

14. Programming in the age of AI

AI coding assistants can generate explanations, code, tests and debugging suggestions. This changes how programmers work, but it does not remove the need to learn programming.

What is a large language model?

A Large Language Model (LLM) is trained on very large collections of text and code and learns statistical patterns that allow it to generate likely continuations and responses. GPT stands for Generative Pre-trained Transformer.

AI can help with	But you must still...
Explaining a programming concept	Check whether the explanation fits the actual problem
Generating an example	Read and understand important lines
Finding possible syntax errors	Run and test the program
Suggesting debugging ideas	Verify the proposed fix
Summarising documentation	Check original documentation when accuracy matters
Suggesting alternatives	Compare trade-offs and choose deliberately

THE CORE PRINCIPLE

AI can accelerate programming, but it cannot take responsibility for the correctness of your program.

You remain the programmer.

Why programming knowledge still matters

- Problem analysis — understand what the problem actually asks.
- Algorithm design — decide on a sensible sequence of steps.
- Debugging — identify why a program behaves incorrectly.
- Testing — check normal and unusual cases.
- Evaluation — judge whether generated code is correct and maintainable.
- Maintenance — modify software safely as requirements change.

Use AI as a learning partner

Better approach	Avoid
Attempt the problem yourself first	Copying a complete solution without understanding it

Ask for an explanation or hint	Depending on AI for every small decision
Ask why your code is wrong	Assuming the AI's answer must be correct
Compare approaches	Accepting the first generated solution
Rewrite the solution yourself	Submitting code you cannot explain
Test with different inputs	Skipping testing because code looks plausible

A practical AI-assisted learning workflow

1. Attempt the problem independently.
2. If stuck, ask for a hint or explanation before requesting a full solution.
3. Compare different approaches.
4. Rewrite the solution yourself.
5. Explain the program in your own words.
6. Test it with different inputs.
7. Improve the program based on what you learned.

A USEFUL TEST

If an AI assistant gives you code, close the answer and ask: “Could I explain every important line, and could I modify it if the requirement changed?” If not, keep learning.

15. Common Week 1 misconceptions

Misconception	Correct idea
“123” is automatically an integer because it contains digits.	A literal such as “123” is an integer; text containing those characters is a different type.
“//” is just another spelling of “/”.	“/” and “//” have different meanings and can produce different results.
“%” calculates percentages.	“%” gives the remainder after division.
“**” means multiplication twice.	“**” is exponentiation.
Python guesses what I mean.	Python follows defined language rules.
Code is executed all at once.	A script normally proceeds statement by statement in order.
If the answer looks right, the program is correct.	Programs should be tested, including unusual cases.

16. Practice: predict before you run

Write down your prediction first. Then run the expressions in the REPL.

```
1.  7 + 3 * 2
2.  (7 + 3) * 2
3.  17 / 5
4.  17 // 5
5.  17 % 5
6.  2 ** 8
7.  type(7)
8.  type(7.0)
9.  type(7 + 0.0)
```

Challenge: design a small calculation

Choose a real-world scenario and create a tiny Python script that performs one useful calculation. Examples include travel distance, energy consumption, storage capacity, study time or a game score.

- Choose meaningful names.
- Use at least two arithmetic operators.
- Use parentheses where they make the calculation clearer.
- Print a sentence explaining the result.
- Run the program and check the result manually.

17. Connection to the Mission Control workshop

The separate workshop book will use these foundations to build the first version of a Mission Control Dashboard. You will not learn the concepts by copying a large application; instead, the workshop applies each idea in a meaningful context.

This chapter teaches	Workshop application
Numeric values and types	Mission measurements
Arithmetic operators	Distance, delay, fuel and power calculations
Expressions	Derived mission quantities
“print()”	A readable mission dashboard
Execution order	A sequence of dashboard calculations
Incremental development	A dashboard that grows each week

IMPORTANT

The Mission Control Dashboard is deliberately designed to evolve. Future weeks will add user input, decisions, loops, functions and other ideas. The workshop book will document that development separately.

18. Week 1 quick reference

Concept	Remember
Programming	Designing, writing, testing and maintaining instructions
REPL	Interactive Python environment for quick experiments
Script	Saved Python program, usually a “.py” file
int	Whole-number numeric type
float	Floating-point numeric type
Literal	A value written directly in source code
Expression	Code Python evaluates to produce a value
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Floor division
%	Remainder
**	Exponentiation
type(x)	Reports the type of x
Precedence	Rules controlling evaluation order
print(...)	Displays values

19. Self-test: can you explain these?

1. What is programming?

2. Why do we need programming languages?

3. What is the difference between a REPL and a script?

4. What is the difference between an integer and a floating-point value?

5. What is a literal?

6. What is an expression?

7. What is the difference between “/” and “//”?

8. What does “%” return?

9. What does “” mean?**

10. Why are “10 + 3 * 2” and “(10 + 3) * 2” different?

11. What does “type()” tell you?

12. Why are meaningful names important?

13. Why should you test AI-generated code?

20. Summary

Programming is the systematic design of instructions for solving problems. Programming languages give humans a practical way to express those instructions. Python is readable, versatile and widely used in AI, data, automation, science and many other areas.

Technically, you have met the Python REPL and scripts, numeric values and types, literals, expressions and the main arithmetic operators. You have learned that “/”, “//”, “%” and “**” have distinct meanings, and that operator precedence affects expression results. You have also started developing a professional habit: predict, run, inspect, test and explain.

Finally, you have learned how to work with AI responsibly. A strong programmer is not someone who never uses AI; it is someone who understands enough programming to ask good questions, evaluate answers, test generated code and remain responsible for the final solution.

ONE SENTENCE TO REMEMBER

Learn Python not just to write code, but to learn how to turn problems into precise, testable and maintainable solutions.

Further reading and useful resources

The Week 1 workshop resources identify the Python website and official documentation as key resources. Become comfortable using documentation as part of normal programming practice.

Resource	Use it for
Python official website	Python downloads and the Python ecosystem
Python documentation	Language reference, built-ins and standard library documentation
Development environment	Writing, running and debugging ".py" files
Course lecture notes	Conceptual explanations and examples
Course workshop book	Guided practice and the evolving Mission Control project