

PYTHON PROGRAMMING FOR CIVIL ENGINEERS

UNIT - I : FUNDAMENTALS OF PYTHON PROGRAMMING

Detailed classroom notes with civil engineering examples, sample programs and flowcharts

Prepared as teaching material

UNIT-I SYLLABUS COVERED

Introduction to Python; installation and setup; interactive and script modes; Python IDEs; keywords and identifiers; variables and data types; input and output functions; operators and expressions; operator precedence; writing and executing simple Python programs.

1. Learning Outcomes for Unit I

After completing this unit, a student should be able to:

- Explain what Python is and why it is useful for engineering computation and automation.
- Install Python and verify a working Python interpreter.
- Distinguish interactive mode from script mode and choose the appropriate mode for a task.
- Use a Python IDE to create, save, run and debug a basic program.
- Identify Python keywords and construct valid identifiers for engineering variables.
- Create variables and work with int, float, bool, str and list values at an introductory level.
- Accept keyboard input using input() and display formatted output using print().
- Use arithmetic, comparison, logical and assignment operators in expressions.
- Predict expression results using operator precedence and parentheses.
- Develop small civil engineering programs using the input-process-output pattern.

2. Introduction to Python

2.1 What is programming?

A computer does not directly understand a civil engineering problem stated in natural language such as "find the volume of concrete required for a slab". The problem must be converted into a sequence of precise instructions. A program is a sequence of instructions that specifies how a computation is to be carried out.

General problem-solving flow

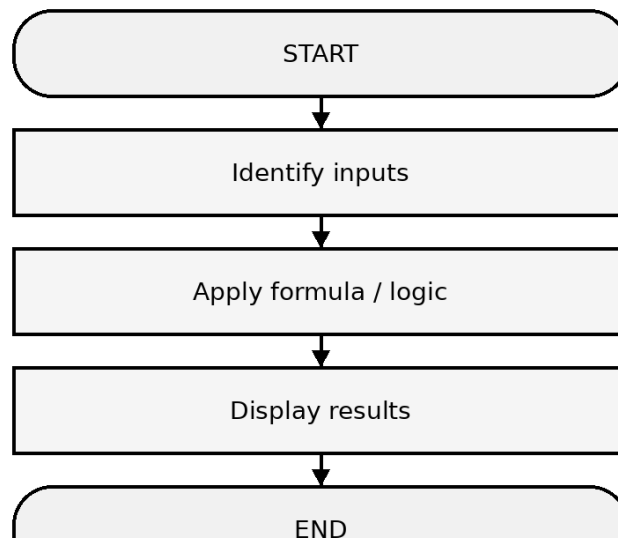


Figure 1.1 General problem-solving flow used when developing a small engineering program.

Programming is therefore the process of translating a problem-solving procedure into a programming language. Python provides a high-level syntax that lets engineering students express calculations using readable statements.

2.2 What is Python?

- Python is a high-level, general-purpose programming language with readable syntax.
- It is commonly used interactively, which makes it convenient for testing formulas, units and small calculations.

- Python supports procedural, object-oriented and other programming paradigms; Unit I focuses on fundamental procedural computation.
- Python programs normally use the .py file extension in script form.
- Python is dynamically typed: a variable name can refer to values of different types at different points in a program.

Important teaching point

For civil engineers, Python should be viewed first as a computational tool: enter engineering data -> apply a formula or algorithm -> check units and assumptions -> display or store the result.

2.3 Why Python for Civil Engineers?

Civil engineering task	How Python can help
Basic numerical calculations	Area, volume, density, stress, strain, flow and unit conversions.
Repeated calculations	Apply the same calculation to many projects, sections or test observations.
Data processing	Read measurements and perform calculations after later units introduce lists, files and data libraries.
Survey / geomatics	Coordinate calculations, distance, slope and geospatial workflows in later units/courses.
Structural engineering	Section properties, loads, reactions and simple design checks.
Hydraulics / environmental engineering	Flow calculations, hydraulic quantities and data analysis.
Automation	Reduce repetitive manual spreadsheet/calculator work and create reproducible calculations.

3. Installation and Setup

3.1 Python interpreter and installation

To execute Python programs locally, a Python interpreter must be installed. The supplied course notes describe Python as being run through an interpreter and distinguish interactive mode from script mode. A typical setup also includes an editor or IDE for writing and saving .py files.

Python script execution - conceptual view

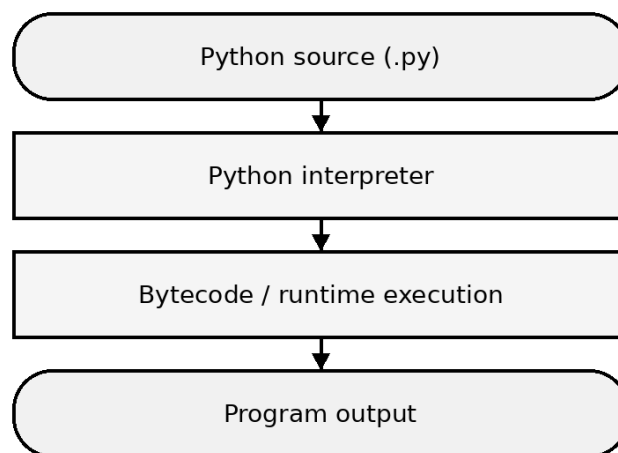


Figure 3.1 Conceptual execution path of a Python script.

3.2 Windows installation - recommended classroom procedure

1. Download a current Python 3 release from the official Python website.
2. Run the installer and select the option to add Python to PATH when the installer presents that choice. This makes the python command easier to use from Command Prompt / Terminal.
3. Complete the installation with the standard options unless your laboratory has a specific configuration.
4. Open Command Prompt and verify the installation with `python --version`. On systems where python maps to another executable, `py --version` may also be used.
5. Start IDLE, VS Code or another approved IDE and run a simple `print()` statement.

```
python --version
```

```
# On some Windows installations
```

```
py --version
```

Installation check

A successful version command should report a Python 3.x version. The exact minor version can change over time, so teaching should focus on the Python 3 workflow rather than memorising one version number.

3.3 Linux / macOS

- Open Terminal and check whether Python 3 is available using `python3 --version` (or the command configured by your institution).
- Use the operating system package manager or an approved Python installer/environment when Python is not already available.
- Run a quick test using `python3` in a terminal or through an IDE.

```
python3 --version
```

```
python3
```

3.4 Laboratory checklist

Check	Expected result
Python installed	A Python 3.x version is displayed.
Interpreter starts	The Python prompt appears, usually <code>>>></code> in interactive mode.
Simple expression	Typing <code>2 + 3</code> produces 5.
<code>print()</code> works	<code>print("Hello Civil Engineers")</code> displays the text.
Script runs	A saved <code>.py</code> file executes without an installation error.

4. Interactive Mode and Script Mode

4.1 Interactive mode

Interactive mode means that Python statements are entered at the interpreter prompt and executed immediately. This mode is especially useful for learning syntax, checking an engineering formula, testing a function and exploring the behaviour of operators.

```
>>> 2 + 3
```

```
5
```

```
>>> length = 12.5
```

```
>>> length
```

```
12.5
```

```
>>> print("Test calculation")
```

```
Test calculation
```

The >>> prompt

In the Python shell, >>> indicates that the interpreter is ready to receive a new command. The prompt itself is not part of the Python code and should not be typed into a saved .py file.

4.2 Script mode

In script mode, Python statements are written in a file, normally with a .py extension. The file can be saved, rerun and shared. Script mode is preferable for engineering calculations that must be reproduced, reviewed or extended.

```
# area_rectangle.py
length = 8.0
width = 5.0
area = length * width
print("Area =", area, "m^2")
```

4.3 Interactive mode vs script mode

Feature	Interactive mode	Script mode
Input	One statement / small block at a time	Program saved in a .py file
Feedback	Immediate	After running the file
Best use	Learning, testing and quick calculations	Reusable engineering programs
Storage	Not automatically stored as a program	Saved and versionable
Sharing	Less convenient	Easy to distribute as a file

4.4 Typical script workflow

- Open an IDE/editor.
- Create a new file and save it with a meaningful name such as slab_volume.py.
- Write the program using correct indentation and syntax.
- Save the file.
- Run the program.
- Check output, units and engineering assumptions.
- Correct any errors and rerun.

5. Python IDEs and Development Environments

An Integrated Development Environment (IDE) combines code editing with features such as running, debugging, file management and sometimes virtual-environment or package support. The supplied notes mention IDLE, Visual Studio Code and PyCharm as common choices.

Tool	Useful for beginners	Strengths for teaching
IDLE	Very suitable	Comes with standard Python installation in many distributions; simple editor plus shell.
Visual Studio Code	Suitable after basic setup	Lightweight editor, syntax highlighting, extensions, terminal and debugging.
PyCharm	Suitable, especially for larger projects	Strong project management, debugging and code navigation.
Jupyter Notebook / JupyterLab	Excellent for calculation demonstrations	Mixes explanatory text, code and output; useful for engineering data exploration in later units.

Teaching recommendation

For a first laboratory, use one standard environment for the entire class. IDLE is simple for first exposure; VS Code is a good transition to professional software development. The mathematical concepts remain the

same regardless of the IDE.

6. Python Syntax Essentials

6.1 Python statements

A statement is an instruction that Python can execute. Introductory statements include assignment, function calls such as `print()`, and later control-flow statements such as `if`, `while` and `for`. Unit I concentrates on simple sequential statements.

```
length = 10
width = 4
area = length * width
print(area)
```

6.2 Comments

Comments are notes written for human readers. A single-line comment begins with the `#` character; Python ignores the comment text when executing the program.

```
# Calculate the gross floor area
length = 25
width = 18
area = length * width
print(area)
```

- Good comments explain purpose, assumptions or non-obvious decisions.
- Do not use comments as a substitute for clear variable names.
- Comments are especially valuable in civil engineering programs to record units and assumptions.

6.3 Indentation

Python uses indentation to represent code blocks. Unit I programs may not yet contain many blocks, but students should form the habit of consistent indentation. Four spaces is the standard convention for each indentation level.

```
if area > 100:
    print("Large area")
```

Common beginner error

Mixing tabs and spaces or using inconsistent indentation can produce an `IndentationError` or unexpectedly change the structure of a program.

7. Keywords and Identifiers

7.1 Keywords

Keywords are reserved words with predefined meaning in Python. They cannot be used as ordinary user-defined variable names. Examples include `if`, `else`, `for`, `while`, `def`, `class`, `import`, `True`, `False` and `None`.

Category	Examples
Decision	<code>if</code> , <code>elif</code> , <code>else</code>
Loop	<code>for</code> , <code>while</code> , <code>break</code> , <code>continue</code>
Function / class	<code>def</code> , <code>return</code> , <code>class</code> , <code>lambda</code>

Modules / exceptions	import, from, as, try, except, finally
Boolean / absence	True, False, None
Other language keywords	and, or, not, in, is, pass, yield, with, assert, del, global, nonlocal

Checking keywords in Python

The keyword module can be used to inspect the exact keyword list for the Python version installed in the laboratory. This is preferable to memorising a list from a textbook because the language evolves.

```
import keyword
print(keyword.kwlist)
```

7.2 Identifiers

An identifier is a programmer-defined name used for variables, functions, classes and other named objects. Good identifiers communicate the engineering quantity being represented.

Rule / convention	Valid examples	Invalid examples
Start with a letter or underscore	length, beam_depth, _temp	2length
May contain letters, digits and underscores	span2, slab_width	slab-width
Cannot contain spaces	water_depth	water depth
Case-sensitive	area, Area, AREA are different names	-
Do not use keywords	total_load, design_moment	class, for, if

7.3 Naming conventions for civil engineering variables

Engineering quantity	Recommended variable name	Unit example
Beam span	beam_span	m
Width	width	m
Depth	depth	m
Area	area	m ²
Volume	volume	m ³
Density	density	kg/m ³
Stress	stress	MPa
Discharge	discharge	m ³ /s

Best practice

Prefer descriptive names such as concrete_volume over x when teaching engineering programming. A short name can be acceptable in a formula demonstration, but clear names reduce calculation and review errors.

8. Variables and Data Types

8.1 Variables

A variable is a name that refers to a value. Python does not require a separate declaration such as int x before assignment. The variable is created when a value is assigned.

```
length = 12
width = 8
area = length * width
print(area)
```

- Assignment uses the = operator.
- The expression on the right-hand side is evaluated first; the resulting value is associated with the name on the left.
- A variable can later refer to a value of another type because Python is dynamically typed.

```
x = 5
```

```
print(type(x))
x = "five"
print(type(x))
```

8.2 Core data types for Unit I

Data type	Purpose	Example	Civil engineering example
int	Whole numbers	25	Number of floors / test specimens
float	Real-valued numerical quantities	3.75	Beam depth, density, discharge
bool	True/False conditions	True	Pass/fail status, condition checks
str	Text / character data	"Concrete"	Material name, project name
list	Ordered collection of values	[10, 12, 15]	A set of measured dimensions; formally covered later

8.3 Integer (int)

An integer is a whole number, positive, negative or zero, with no fractional part.

```
n = 12
print(type(n))
print(n + 8)
```

8.4 Floating-point number (float)

A float represents a real-valued quantity with a fractional component or decimal notation. Most physical engineering quantities are handled using floats.

```
density = 2400.0
water_depth = 1.75
print(type(density))
print(water_depth)
```

Engineering caution

Floating-point values are finite-precision representations. For ordinary classroom engineering calculations they are usually adequate, but students should not assume that every decimal is represented exactly in binary floating-point.

8.5 Boolean (bool)

A Boolean has one of two values: True or False. Boolean values are commonly produced by comparison expressions.

```
strength = 28.5
required = 25.0
adequate = strength >= required
print(adequate)
```

8.6 String (str)

A string is a sequence of characters enclosed in single or double quotation marks. Strings represent text rather than numerical quantities.

```
project = "Riverside Bridge"
material = "Concrete"
print(project)
```

```
print(material)
```

- Use quotes around text.
- Do not perform numerical arithmetic on strings without appropriate conversion.

8.7 List - introductory view

A list is an ordered, changeable collection. Lists are useful for storing several values, such as observed measurements. Detailed list operations belong to later units, but students should recognise the syntax.

```
rainfall = [12.4, 8.5, 15.2, 11.0]
print(rainfall)
print(type(rainfall))
```

8.8 Checking a value with type()

```
x = 12.5
print(type(x))

y = "12.5"
print(type(y))
```

The two variables above may display similarly when printed, but one contains a numerical float and the other contains text. This distinction is critical when engineering formulas are evaluated.

9. Type Conversion (Type Casting)

User input is normally obtained as text. Before using input in a numerical calculation, the text usually needs to be converted to int or float. Common conversion functions are int(), float() and str().

Function	Purpose	Example
int()	Convert to integer where conversion is valid	int("25") -> 25
float()	Convert to floating-point number	float("25.6") -> 25.6
str()	Convert a value to text	str(25.6) -> "25.6"
bool()	Convert to Boolean according to Python truth rules	bool(0) -> False

```
diameter = float(input("Enter diameter (m): "))
radius = diameter / 2
print("Radius =", radius, "m")
```

Very common mistake

Writing `diameter = input("Enter diameter: ")` stores text, not a numerical float. Use `float(input(...))` for decimal engineering input and `int(input(...))` for whole-number input.

10. Input and Output Functions

10.1 input()

The input() function reads a line of text entered by the user. By default, the returned value is a string.

```
name = input("Enter project name: ")
print("Project:", name)
age = input("Enter age: ")
print(type(age))
```

```
# age is a string, even when the user types 20
```

10.2 Numerical input

```
length = float(input("Enter length (m): "))
width = float(input("Enter width (m): "))
area = length * width
print("Area =", area, "m^2")
```

10.3 print()

The print() function sends data to standard output, usually the screen. Multiple items can be supplied, separated by commas.

```
length = 12.0
width = 4.0
area = length * width
print("Length =", length, "m")
print("Width =", width, "m")
print("Area =", area, "m^2")
```

10.4 f-strings for readable engineering output

Formatted string literals, commonly called f-strings, provide a clear way to insert variable values into output text.

```
area = 48.0
print(f"Area = {area:.2f} m^2")
```

- `:.2f` displays two digits after the decimal point.
- Formatting is presentation; it does not change the underlying numerical value.

11. Input - Process - Output (IPO) Model

A reliable way to design an introductory engineering program is to identify the INPUTS, PROCESS and OUTPUTS before writing code. This separates engineering reasoning from programming syntax.

Input - Process - Output model

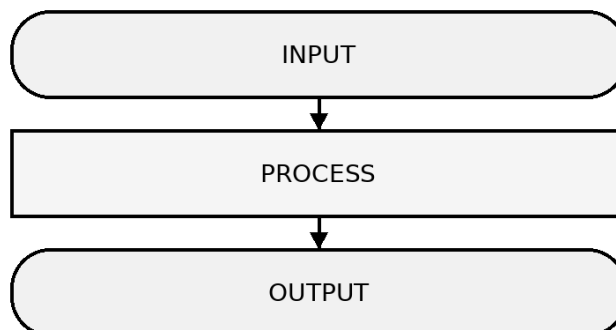


Figure 11.1 The basic Input - Process - Output model.

Stage	Questions to ask	Example: rectangular slab
Input	What data must the user provide?	Length L, width B, thickness t
Process	What formula/algorithm is required?	$V = L * B * t$
Output	What result should be reported, and in what units?	Concrete volume in m ³

12. Operators and Expressions

12.1 What is an expression?

An expression is a combination of values, variables, operators and function calls that Python can evaluate to produce a value.

```
width * depth
```

```
load / area
```

```
3.14 * radius ** 2
```

12.2 Arithmetic operators

Operator	Meaning	Example	Result
+	Addition	12 + 5	17
-	Subtraction	12 - 5	7
*	Multiplication	12 * 5	60
/	True division	12 / 5	2.4
//	Floor division	12 // 5	2
%	Remainder (modulo)	12 % 5	2
**	Exponentiation	2 ** 3	8

Civil engineering connection

Expressions are the direct translation of engineering equations. For example, $\text{area} = \text{length} * \text{width}$ and $\text{volume} = \text{length} * \text{width} * \text{thickness}$ are simple arithmetic expressions represented in Python syntax.

12.3 Comparison operators

Operator	Meaning	Example	Result
==	Equal to	25 == 25	True
!=	Not equal to	25 != 30	True
>	Greater than	30 > 25	True
<	Less than	20 < 25	True
>=	Greater than or equal to	25 >= 25	True
<=	Less than or equal to	20 <= 25	True

```
strength = 28.0
```

```
required = 25.0
```

```
print(strength >= required)
```

12.4 Logical operators

Operator	Meaning	Example
and	True only when both conditions are true	$\text{area} > 100$ and $\text{depth} > 0.15$
or	True when at least one condition is true	$\text{soil_type} == \text{"A"}$ or $\text{soil_type} == \text{"B"}$
not	Reverses a Boolean value	not safe

```
load = 450.0
```

```
capacity = 500.0
```

```
span = 6.0
limit = 8.0
acceptable = (load <= capacity) and (span <= limit)
print(acceptable)
```

12.5 Assignment operators

Operator	Equivalent idea	Example
=	Assign	area = 100
+=	Add then assign	load += 50
-=	Subtract then assign	load -= 20
*=	Multiply then assign	factor *= 2
/=	Divide then assign	value /= 10

Do not confuse = and ==

= assigns a value; == compares two values. This distinction is one of the most important beginner concepts.

12.6 Identity and membership operators - awareness level

- is / is not check object identity and are not general substitutes for ==.
- in / not in test membership in a sequence or collection, such as a string or list.

```
materials = ["concrete", "steel", "timber"]
print("steel" in materials)
```

13. Operator Precedence

When an expression contains several operators, Python follows a precedence order. Higher-precedence operations are evaluated before lower-precedence operations. Parentheses can be used to make the intended order explicit.

Priority	Operator / construct	Example
1 (highest)	Parentheses ()	(2 + 3) * 4
2	Exponentiation **	2 ** 3
3	Unary + and -	-5
4	*, /, //, %	20 / 5 * 2
5	+, -	20 - 5 + 2
6	Comparisons	x >= y
7	not	not safe
8	and	a and b
9	or	a or b
10 (assignment is a statement)	=	x = 10

```
print(2 + 3 * 4)
print((2 + 3) * 4)

# Civil engineering style example
net_area = gross_area - 2 * opening_area
net_area2 = (gross_area - (2 * opening_area))
```

Teaching rule

When an engineering formula has several operations, use parentheses to make the intended mathematical grouping obvious, even when Python precedence already gives the same result.

14. Sample Programs for Civil Engineering Students

14.1 Program 1 - Hello, Civil Engineers

```
print("Welcome to Python Programming for Civil Engineers")
```

Expected output:

```
Welcome to Python Programming for Civil Engineers
```

14.2 Program 2 - Add two measured dimensions

```
# Add two dimensions
a = float(input("Enter first dimension (m): "))
b = float(input("Enter second dimension (m): "))
total = a + b
print(f"Total = {total:.2f} m")
```

14.3 Program 3 - Area of a rectangular plot

```
# Area of rectangular plot
length = float(input("Enter length (m): "))
width = float(input("Enter width (m): "))
area = length * width
print(f"Area = {area:.2f} m^2")
```

Civil engineering example: area of a circular footing

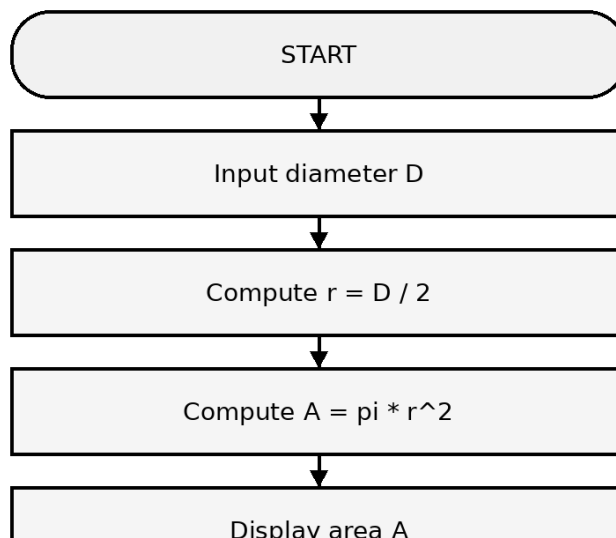


Figure 14.1 Flowchart for the same input-process-output idea using a circular footing example.

14.4 Program 4 - Volume of a rectangular concrete block

```
# Volume of rectangular concrete block
length = float(input("Length (m): "))
width = float(input("Width (m): "))
thickness = float(input("Thickness (m): "))
volume = length * width * thickness
print(f"Concrete volume = {volume:.3f} m^3")
```

14.5 Program 5 - Area of a circle / circular footing

```
import math

diameter = float(input("Enter footing diameter (m): "))
radius = diameter / 2
area = math.pi * radius ** 2
print(f"Footing area = {area:.3f} m^2")
```

Why import math?

The math module provides mathematical constants and functions. `math.pi` is a more reliable way to represent pi than typing a short decimal approximation.

14.6 Program 6 - Convert millimetres to metres

```
mm = float(input("Enter length (mm): "))
m = mm / 1000
print(f"Length = {m:.3f} m")
```

14.7 Program 7 - Calculate density

```
mass = float(input("Enter mass (kg): "))
volume = float(input("Enter volume (m^3): "))
density = mass / volume
print(f"Density = {density:.2f} kg/m^3")
```

Engineering check

Always check whether units are consistent before calculating. The formula may be correct but the result can be meaningless if mass and volume are not expressed in compatible units.

14.8 Program 8 - Simple stress calculation

```
force = float(input("Enter axial force (N): "))
area = float(input("Enter cross-sectional area (mm^2): "))
stress = force / area
print(f"Stress = {stress:.2f} N/mm^2")
print(f"Stress = {stress:.2f} MPa")
```

Unit interpretation

1 N/mm² is numerically equal to 1 MPa. The code is intentionally simple so that students can focus on input, variables, arithmetic expression and output.

14.9 Program 9 - Simple rate calculation

```
distance = float(input("Distance travelled (km): "))
time = float(input("Time taken (h): "))
speed = distance / time
print(f"Average speed = {speed:.2f} km/h")
```

14.10 Program 10 - Two-step calculation using an engineering formula

```
# Gross-to-net area example
gross_area = float(input("Gross floor area (m^2): "))
opening_area = float(input("Total opening area (m^2): "))
net_area = gross_area - opening_area
```

```
print(f"Net usable area = {net_area:.2f} m^2")
```

15. Writing and Executing a Simple Python Program: Complete Walkthrough

13. Define the engineering problem in one sentence.
14. List the known quantities and their units.
15. Identify the required output quantity and unit.
16. Write the engineering formula or algorithm on paper.
17. Construct the IPO table.
18. Choose descriptive Python variable names.
19. Write the program using `float(input(...))` or `int(input(...))` as appropriate.
20. Run the program with a hand-checked sample value.
21. Compare the Python result with an independent calculator calculation.
22. Test at least one boundary or unusual input if relevant.
23. Save the .py file with a meaningful file name.

Problem element	Example
Problem	Find concrete volume for a rectangular slab.
Inputs	Length, width, thickness.
Formula	$V = L * B * t$.
Variables	length, width, thickness, volume.
Output	Concrete volume in m^3 .
Verification	Hand-calculate one test case before accepting program output.

16. Common Beginner Errors and Debugging

Error / issue	Typical symptom	Correction
SyntaxError	Program cannot be parsed	Check quotes, parentheses, colons and spelling.
NameError	Unknown variable name	Check spelling and whether the variable was assigned before use.
TypeError	Operation not valid for the given types	Check whether a numeric value is actually a string; use <code>int()/float()</code> where appropriate.
ValueError	Conversion fails, e.g. <code>float("abc")</code>	Check input format.
ZeroDivisionError	Division by zero	Validate denominator before division when required.
IndentationError	Unexpected indentation / missing indentation	Use consistent indentation.
Wrong output but no error	Program runs but result is incorrect	Check the formula, units, variable assignment and operator precedence.

```
value = input("Enter a value: ")
print(type(value))
```

```
# Correct numerical input
value = float(input("Enter a numerical value: "))
print(type(value))
```

Three-question debugging habit

1) Is the Python syntax valid? 2) Are the data types correct? 3) Is the engineering formula and unit system correct? A program can be syntactically correct and still produce an engineering error.

17. Good Programming Practices from the First Lab

- Use meaningful names: `beam_span` is better than `x` for a classroom engineering calculation.
- Include units in prompts and outputs: "Enter thickness (mm):" is clearer than "Enter thickness:".
- Use comments for assumptions and non-obvious steps.
- Use parentheses when they improve the readability of an engineering expression.
- Avoid unnecessary hard-coded numerical constants when a named variable communicates the meaning.
- Test the program with values whose answer you can calculate independently.
- Do not copy `>>>` prompts from interactive mode into a script file.
- Save each laboratory program with a descriptive `.py` filename.

18. Short-answer / Viva Questions

1. What is a programming language?
2. What is Python?
3. Why is Python called a high-level language?
4. What is an interpreter?
5. What is the difference between interactive mode and script mode?
6. What is an IDE? Name two Python IDEs.
7. What is the purpose of the `.py` extension?
8. What is a keyword?
9. What is an identifier?
10. Why is `total_load` a better engineering variable name than `x`?
11. Why is 25 different from "25" in Python?
12. What does `type()` do?
13. What type of data does `input()` return by default?
14. Why is `float(input(...))` commonly used in engineering programs?
15. What is the difference between `=` and `==`?
16. What is operator precedence?
17. Why are parentheses useful in engineering formulas?
18. What does `**` mean?
19. What does `%` mean?
20. Why should engineering units be included in prompts and outputs?

19. Practice Programs / Laboratory Exercises

- Read length and breadth and calculate rectangular area and perimeter.
- Read diameter and calculate circumference and area of a circular footing.
- Read length, width and thickness of a slab and calculate concrete volume.
- Read mass and volume and calculate density.
- Convert millimetres to metres, centimetres to metres and square millimetres to square metres.
- Read force in kN and area in m^2 and calculate stress in kPa.
- Read rise and run and calculate slope as a ratio and percentage.

- Read floor dimensions and calculate gross floor area and carpet area after subtracting an opening area.
- Write a program that reads two numbers and displays the results of +, -, *, /, //, % and **.
- Write one program that demonstrates the difference between 5, 5.0 and "5" using type().
- Write an IPO table and a flowchart for a civil engineering calculation of your choice before coding it.

20. Unit I Summary

Concept	Key point
Python	High-level programming language suitable for engineering computation and automation.
Interpreter	Executes Python code; interactive mode provides immediate feedback.
Interactive mode	Best for quick experiments and learning.
Script mode	Best for reusable, saved engineering programs.
IDE	Integrated environment for editing, running and debugging code.
Keyword	Reserved Python word with a predefined language role.
Identifier	User-defined name for a variable/function/etc.
Variable	Name referring to a value.
Data type	Defines the kind of value, such as int, float, bool or str.
input()	Reads user input as text.
print()	Displays output.
Type conversion	Uses functions such as int(), float(), str().
Expression	Combination of values/variables/operators evaluated to a value.
Operator precedence	Rules governing evaluation order.
IPO	Input -> Process -> Output provides a simple program-design framework.

21. References and Source Material Consulted

These notes were developed by synthesising the syllabus and the uploaded teaching references, while adding original civil-engineering-oriented examples and classroom explanations.

- OpenStax, Introduction to Python Programming, 2024. Relevant coverage: statements, input/output, variables, strings, number basics, Python shell, type conversion, expressions and operator precedence. The textbook identifies itself as ©2024 Rice University / OpenStax and is licensed under CC BY 4.0.
- Malla Reddy College of Engineering & Technology, Python Programming Lecture Notes (R17A0554), Unit I. Relevant coverage: introduction and installation, data types, variables, expressions, statements, precedence, comments, interactive/script execution and basic programming concepts.
- Martin Ganchev, Intro to Python - Course Notes. Relevant coverage: introduction to programming, why Python, variables and data types, basic Python syntax, arithmetic/comparison/logical operators, input/output and installation/IDE overview.
- Brian Jenkins, Python Programming for Beginners. Relevant coverage: what Python is, installation, interactive interpreter, script execution, IDEs, syntax, input and variables.

Source-use note

The uploaded references were used as teaching references. Examples in this document are intentionally rewritten and contextualised for civil engineering education rather than reproduced as textbook passages.

22. Instructor Teaching Plan for a First Unit

Suggested session	Content	Quick classroom activity
Session 1	What is programming? What is Python? Why Python for civil engineering?	Ask students to convert a rectangular-area calculation into IPO form.
Session 2	Installation, interpreter, interactive mode	Students execute <code>2 + 3</code> , assign variables and inspect <code>type()</code> .
Session 3	Script mode and IDEs	Students create, save and run a .py file.
Session 4	Keywords, identifiers, variables, data types	Repair invalid variable names and classify sample engineering data.
Session 5	<code>input()</code> , <code>print()</code> , type conversion	Build the rectangular plot area program.
Session 6	Operators and expressions	Predict outputs before running arithmetic expressions.
Session 7	Operator precedence and engineering formulas	Translate 5-6 standard engineering equations into Python expressions.
Session 8	Mini lab / assessment	Student creates one complete IPO + flowchart + Python program and validates the answer by hand.

Suggested assessment approach

Give marks for problem definition, IPO/flowchart, variable naming and units, correct Python syntax, correct output and independent verification. This encourages engineering thinking rather than syntax-only memorisation.