Week 2 — Strings, Variables, Input and Output

BIG IDEA
Week 1: Python could evaluate values, numeric operators and expressions.
Week 2: programs can represent text, remember data using names, receive user data, convert it into useful types, process it and communicate results.

INPUT → STORE → PROCESS → OUTPUT

1. Learning Objectives

- Recognise and create str values using single, double and triple quotes.
- Combine strings using concatenation.
- Convert compatible values with int(), float() and str().
- Create valid, meaningful variable names using Python conventions.
- Use assignment, reassignment, augmented assignment and multiple assignment.
- Use print(), including multiple arguments, sep and end.
- Use input() and explain why it returns a string.
- Combine input, conversion, variables, expressions and output into a small program.
- Use comments appropriately and preserve indentation.

2. From Calculator to Program

```
print(28000 * 36)

rocket_speed = 28000
mission_hours = 36
distance = rocket_speed * mission_hours
print("Distance:", distance, "km")
```

The second version gives the values meaning. Good programming is not only about getting the correct answer; code should also communicate what the data represents.

3. Strings — Text Is Data

```
print("Mission Control")
print('Rocket ready')
print("123")
print(type("123"))
```

Quotation marks matter: "123" is text (str), while 123 is a number (int).

Single, Double and Triple Quotes

```
print("I'm ready for launch")
print('The spacecraft is "Aurora"')

print("""
=======================
    MISSION BRIEFING
=======================
Destination: Moon
Status: Preparing
=======================
""")
```

String Concatenation

```
print("Mission" + " " + "Control")
print(12 + 3)
print("12" + "3")
```

Try it: Predict the last two outputs and explain why they differ.

4. Type Conversion

```
print(int("42"))
print(float("42"))
print(float("3.14"))
print(str(42))

print(type(int("42")))
print(type(str(42)))
```

REMEMBER
Conversion must make sense. int("42") works; int("rocket") does not.

5. Variables — Give Data Meaningful Names

```
rocket_speed = 28000
mission_hours = 36
distance = rocket_speed * mission_hours
print(distance)
```

Read rocket_speed = 28000 as “assign 28000 to the name rocket_speed.” In Python, = is assignment.

Reassignment

```
x = 12.2
print(x)
x = 100
```

```
print(x)

x = x + 2
print(x)
```

KEY IDEA
For x = x + 2, Python evaluates the right side using the current x, then stores the new result back under x.

6. Variable Naming

- Start with a letter or underscore.
- Use letters, digits and underscores after that.
- No spaces or symbols such as #, . or *.
- Names are case-sensitive.
- Reserved words such as class cannot be variable names.

```
# Good
rocket_speed = 28000
mission2_speed = 31000
_private_value = 5

# Invalid
# 2mission = "Moon"
# rocket-speed = 28000
# class = "A"
```

Meaningful Names and Style

```
# Hard to understand
x1q3z9ocd = 35.0
x1q3z9afd = 12.50

# Better
hours = 35.0
rate = 12.50
pay = hours * rate

first_name = "Mia"
SPEED_OF_LIGHT = 300000
```

Prefer meaningful snake_case names. ALL_CAPITALS is conventionally used for values intended to behave like constants.

7. Assignment and Updating Values

```
x = 10
x = x + 2
print(x)

x += 2
```

```
print(x)

x /= 2
print(x)
```

Long form	Short form
x = x + 2	x += 2
x = x - 2	x -= 2
x = x * 2	x *= 2
x = x / 2	x /= 2

Multiple Assignment — Python Trick

```
x = y = z = "Griffith"
x, y, z = 1, "ab", 3

left = "Moon"
right = "Earth"
left, right = right, left
print(left, right)
```

8. print() — More Powerful Than It Looks

```
speed = 28000
print("Rocket speed:", speed, "km/h")

print("Earth", "Moon", "Mars", sep=" -> ")
print(3, 8, 2026, sep="/")

print("T-", end="")
print(10)
```

sep controls what appears between printed values. end controls what appears after a print call.

Optional Formatting Tricks

```
print("=" * 50)
print("MISSION CONTROL".center(50))
print("=" * 50)
print("Rocket Speed".ljust(25), "28000 km/h")
```

9. input() — Make Programs Interactive

```
name = input("Who are you? ")
print("Welcome", name)
```

input() waits for keyboard input and returns the entered data as a string.

The Important Input Trap

```
age = input("Enter your age: ")
print(type(age))
print(age * 2)
```

Try it: If you enter 20, what is printed by age * 2?

```
age = int(input("Enter your age: "))
print(type(age))
print(age * 2)
```

CORE IDEA
input() returns str. If the value will be used numerically, convert it first.

```
crew_size = int(input("Crew size: "))
rocket_speed = float(input("Rocket speed: "))
```

Use int for whole-number quantities and float when decimal values are meaningful.

10. A Complete Program: Input → Process → Output

```
# INPUT
hours = float(input("Type in the hours: "))
rate = float(input("Type in the pay rate: "))

# PROCESS
pay = hours * rate

# OUTPUT
print("The amount of payment is $", pay)
```

MENTAL MODEL
USER → input() → str → int()/float() if needed → VARIABLE → EXPRESSION → RESULT → print()

11. Comments and Indentation

```
# This is a comment
rocket_speed = 28000  # speed in km/h

# Calculate distance travelled
distance = rocket_speed * 36
```

Comments should explain purpose or reasoning. Indentation will become structurally important in conditionals and loops, so preserve it carefully.

12. Good Habits

Prefer	Avoid
rocket_speed	x1 when meaning matters
first_name	inconsistent naming
pay = hours * rate	pay=hours*rate
one purpose per variable	reusing a variable for unrelated data
convert numeric input clearly	forgetting input() returns str
comments explaining why	comments repeating obvious code

13. Coding Challenges

Challenge 1 — Type Detective

```
42
42.0
"42"
"4" + "2"
int("42")
str(42)
10 / 2
10 // 2
```

Try it: Predict the type of each result before running Python.

Challenge 2 — Human Python

```
x = 5
x = x + 3
x *= 2
x -= 4
x /= 2
print(x)
```

Try it: Track x after every line. What value and type are printed?

Challenge 3 — Legal or Illegal?

```
rocket_speed
rocket2
2rocket
rocket-speed
_speed
class
RocketSpeed
rocket_speed
```

Try it: Which are legal names? Which legal names best follow Python conventions?

Challenge 4 — Predict the Output

```
name = "Mars"
number = 3
print(name * number)
print("12" + "3")
print(12 + 3)
print("A", "B", "C", sep="-")
print("Launch", end=" ")
print("Ready")
```

Try it: Write the exact output before running the code.

Challenge 5 — Debug Mission Control

```
2mission = "Artemis"
speed = input("Rocket speed: ")
hours = input("Mission hours: ")
distance = speed * hours
print("Distance: " + distance + " km")
```

Try it: Find the problems and rewrite the program so it correctly calculates distance.

Challenge 6 — Build a Mission Banner

```
print("=" * 50)
print("YOUR MISSION NAME".center(50))
print("=" * 50)
```

Create your own terminal banner using strings, print(), *, +, sep/end and optional center()/ljust().

14. Optional Python Magic

```
a, b = 10, 20
a, b = b, a

print("🚀" * 10)
print("ha" * 5)
```

These are enrichment examples: useful glimpses of Python's expressive syntax.

15. End-of-Lecture Self-Check

1. What is the difference between 25 and "25"?

2. What does input() return?

3. Why might float(input(...)) be necessary?

4. What does assignment do?

Mission Control Dashboard v2.0

Week 2 Student Activity & Live-Coding Notes

Strings • Variables • Assignment • Type Conversion • Input • Output

MISSION UPGRADE
Week 1 gave Mission Control a display and calculations.
Week 2 gives it a memory and a voice.

Today you will upgrade the dashboard so it can store meaningful mission data, ask an operator for mission parameters, convert keyboard input into useful types, process the data and produce a personalised mission report.

1. What You Will Practise

- Distinguish string and numeric values.
- Create valid, meaningful variable names.
- Use assignment to store values and calculated results.
- Use input() to receive mission data from a user.
- Explain why input() returns a string.
- Convert numeric input using int() or float().
- Use variables in arithmetic expressions.
- Use print() to create a clear mission report.
- Recognise the overall program flow: INPUT → STORE → PROCESS → OUTPUT.

2. Recall Mission Control v1.0

Last week, Mission Control could calculate—but the values were hard-coded:

```
print("Distance:", 28000 * 36, "km")
```

Imagine the rocket speed changes to 31,500 km/h or the mission lasts 42 hours. Should we edit the source code every time?

Your prediction / answer: What is the main limitation of hard-coded values?

3. Stage 1 — Give Mission Data Meaningful Names

```
rocket_speed = 28000
mission_hours = 36
distance = rocket_speed * mission_hours

print("Rocket speed:", rocket_speed, "km/h")
print("Mission duration:", mission_hours, "hours")
print("Distance:", distance, "km")
```

KEY IDEA
= means assignment. Read rocket_speed = 28000 as “store the value 28000 under the name rocket_speed.” Meaningful names make code easier to understand and debug.

Your prediction / answer: Why are rocket_speed and mission_hours more useful names than x and y?

4. Stage 2 — Strings, Values and Types

```
mission_name = "Artemis-AU"
commander = "Mia Chen"
rocket_speed = 28000
fuel_capacity = 125000.0

print(type(mission_name))
print(type(rocket_speed))
print(type(fuel_capacity))
```

Before running the code, predict the type of each variable.

Variable	Your prediction	Python result
mission_name		
commander		
rocket_speed		
fuel_capacity		
CHECK YOUR THINKING Are "28000" and 28000 the same? No. The first is str; the second is int. This difference becomes crucial when we use input().		

5. Stage 3 — Give Mission Control a Voice

```
commander = input("Commander name: ")
mission_name = input("Mission name: ")
destination = input("Destination: ")

print("Welcome,", commander)
print("Preparing mission:", mission_name)
print("Destination:", destination)
```

Run the program twice using different mission information. Notice that the source code stays the same while the data changes.

```
print("Welcome, " + commander + "!")
```

The + operator joins strings here. This is string concatenation.

Your prediction / answer: Why is user input more flexible than hard-coding the commander and mission name?

6. Stage 4 — Discover the input() Trap

Do not fix this immediately. Run it first:

```
mission_hours = input("Mission duration (hours): ")

print(type(mission_hours))
print(mission_hours * 2)
```

Enter 36. You may expect 72. Python instead produces 3636.

Your prediction / answer: Why does Python produce 3636?

CORE IDEA
input() returns str—even when the user types digits. String repetition is therefore happening, not numeric multiplication.

6.1 Fix the Problem

```
mission_hours = float(input("Mission duration (hours): "))

print(type(mission_hours))
print(mission_hours * 2)
```

Now the data journey is:

```
KEYBOARD → STRING → CONVERSION → NUMERIC VALUE → CALCULATION
```

Your prediction / answer: Why might float() be more suitable than int() for mission duration?

7. Main Build — Mission Control v2.0

Build this program section by section. Run after each section rather than pasting the entire program at once.

Part A — Dashboard Header

```
# ============================================================
# AUSTRALIAN SPACE MISSION CONTROL — VERSION 2.0
# ============================================================

print("=" * 60)
print(" AUSTRALIAN SPACE MISSION CONTROL ".center(60, "="))
print(" Interactive Mission Planner — v2.0 ".center(60))
print("=" * 60)
```

Your prediction / answer: Which Week 1 ideas can you recognise in this section?

Part B — Mission Identity

```
commander = input("\nCommander name: ")
mission_name = input("Mission name: ")
destination = input("Destination: ")

print("\nWelcome, " + commander + "!")
print("Configuring mission", mission_name, "to", destination + "...")
```

Your prediction / answer: Which three variables contain strings?

Part C — Mission Parameters

```
rocket_speed = float(input("\nRocket speed (km/h): "))
mission_hours = float(input("Mission duration (hours): "))
fuel_required = float(input("Fuel required (litres): "))
tank_capacity = float(input("Fuel tank capacity (litres): "))
```

Keyboard input begins as text. These values will be used in arithmetic, so they are converted to float.

Your prediction / answer: What would happen later if rocket_speed were not converted?

Part D — Process the Mission Data

```
distance = rocket_speed * mission_hours

full_tanks = fuel_required // tank_capacity
fuel_remainder = fuel_required % tank_capacity

communication_delay = distance / 300000
```

Each line takes existing values, evaluates an expression and stores a new result.

Expression	Question it answers	Operator focus
rocket_speed * mission_hours	How far does the rocket travel?	*
fuel_required // tank_capacity	How many complete tank-capacity units fit?	//
fuel_required % tank_capacity	How much fuel remains after complete units?	%
distance / 300000	Approximate light-travel time in seconds	/
MODELLING NOTE The communication-delay calculation is a simplified teaching calculation using approximately 300,000 km/s for the speed of light.		

Part E — Produce the Mission Report

```
print("\n" + "=" * 60)
print(" MISSION PLAN ".center(60, "="))
print("=" * 60)

print("Commander".ljust(28), commander)
```

```
print("Mission".ljust(28), mission_name)
print("Destination".ljust(28), destination)

print("-" * 60)

print("Rocket speed".ljust(28), rocket_speed, "km/h")
print("Mission duration".ljust(28), mission_hours, "hours")
print("Distance travelled".ljust(28), distance, "km")
print("Communication delay".ljust(28), communication_delay, "seconds")
print("Full fuel tanks".ljust(28), full_tanks)
print("Fuel remainder".ljust(28), fuel_remainder, "L")

print("=" * 60)
print(("MISSION " + mission_name + " CONFIGURED").center(60))
print("=" * 60)
```

The report turns calculated data into information that is readable by a person.

8. Complete Mission Control v2.0

Use this complete version after you have built and tested the stages above.

```
# ============================================================
# AUSTRALIAN SPACE MISSION CONTROL — VERSION 2.0
# ============================================================

print("=" * 60)
print(" AUSTRALIAN SPACE MISSION CONTROL ".center(60, "="))
print(" Interactive Mission Planner — v2.0 ".center(60))
print("=" * 60)

# Mission identity
commander = input("\nCommander name: ")
mission_name = input("Mission name: ")
destination = input("Destination: ")

print("\nWelcome, " + commander + "!")
print("Configuring mission", mission_name, "to", destination + "...")

# Mission parameters
rocket_speed = float(input("\nRocket speed (km/h): "))
mission_hours = float(input("Mission duration (hours): "))
fuel_required = float(input("Fuel required (litres): "))
tank_capacity = float(input("Fuel tank capacity (litres): "))

# Process mission data
distance = rocket_speed * mission_hours
full_tanks = fuel_required // tank_capacity
fuel_remainder = fuel_required % tank_capacity
communication_delay = distance / 300000

# Mission report
print("\n" + "=" * 60)
print(" MISSION PLAN ".center(60, "="))
print("=" * 60)
print("Commander".ljust(28), commander)
```

```
print("Mission".ljust(28), mission_name)
print("Destination".ljust(28), destination)
print("-" * 60)
print("Rocket speed".ljust(28), rocket_speed, "km/h")
print("Mission duration".ljust(28), mission_hours, "hours")
print("Distance travelled".ljust(28), distance, "km")
print("Communication delay".ljust(28), communication_delay, "seconds")
print("Full fuel tanks".ljust(28), full_tanks)
print("Fuel remainder".ljust(28), fuel_remainder, "L")
print("=" * 60)
print(("MISSION " + mission_name + " CONFIGURED").center(60))
print("=" * 60)
```

9. Check Your Understanding

1. What type does input() return, even if you type digits?

2. Why is float() used for rocket_speed?

3. What happens on the right side of distance = rocket_speed * mission_hours before assignment?

4. Why are meaningful variable names useful?

5. What different questions do // and % answer?

6. Which statements belong to INPUT, PROCESSING and OUTPUT?

7. Which values can change between missions without changing the source code?

10. Five-Minute Debugging Challenge

Do not run this immediately. Find as many problems as you can first.

```
2mission = "Artemis"
speed = input("Rocket speed: ")
hours = input("Mission hours: ")
distance = speed * hours
print("Distance: " + distance + " km")
```

Work with a partner. Mark each problem, explain why it is a problem, then produce a corrected version.

11. Common Misconceptions

- = assigns a value; it is not mathematical equality.
- input() always returns str.
- "12" + "3" produces "123", while 12 + 3 produces 15.
- speed and Speed are different names.
- Variable names cannot begin with a digit or use reserved words.
- int() and float() convert compatible data; they are not merely formatting commands.

12. Version 1 → Version 2

Week 1	Week 2	What you unlocked
Hard-coded values	Variables	Programs remember named data.
Fixed mission	input()	One program works with different data.
Numeric expressions	Stored results	Calculated information can be reused.
Static text	Strings + concatenation	Output becomes personalised.
Basic print()	Structured report	The program communicates clearly.

13. Extension Challenges

- Add a spacecraft_name input and include it in the mission report.
- Add crew_size using int(input(...)).
- Calculate the average fuel required per mission hour.
- Calculate the distance travelled per litre of fuel.
- Improve the report formatting while keeping the calculations unchanged.
- Run two different missions and compare how the same program processes different data.

14. Bridge to Week 3

Mission Control can now receive data, remember it, calculate a mission and produce a report. But there is still something important it cannot do.

What if the fuel is too low? What if the rocket speed is unsafe? Right now the program accepts everything.

NEXT QUESTION
How can we teach a program to make a decision?

That is where Boolean expressions and conditional statements begin.

15. Final Takeaway

MISSION CONTROL v2.0
Week 1: VALUES + OPERATORS + EXPRESSIONS
↓
Week 2: STRINGS + VARIABLES + INPUT + CONVERSION
↓
PROCESS
↓
OUTPUT

You are no longer writing only fixed calculations. You are building a program that can receive changing data, remember it, transform it and communicate the result.

Week 2 — Python Superpowers

Generative Art • Simulation • Data Visualisation

WHY ARE WE DOING THIS?
You have only learned a small part of Python, but those foundations can already be combined with powerful libraries.

In these activities, some lines are deliberately beyond Week 2. You are NOT expected to memorise or fully understand them yet.

Your job is to:
1. recognise the Week 1–2 ideas you already know;
2. observe what an imported library adds;
3. experiment safely by changing selected values;
4. connect simple Python foundations to larger applications.

THE MOST IMPORTANT RULE
Do not be intimidated by unfamiliar code.

When you see a program, separate it into:
KNOWN — concepts you have already learned.
PREVIEW — capabilities you will understand later.

Professional programmers do this all the time when reading unfamiliar code.

1. What You Already Bring to These Programs

Week 1–2 idea	What it lets a program do
Values and types	Represent information
Strings	Represent text
Variables	Give data meaningful names
Assignment	Store or update values
input()	Receive data from a user
int() / float()	Convert text input into numeric data
Expressions	Transform data and calculate results
print()	Communicate results

The imported libraries add new capabilities, but the foundations above still organise the data and control how it is used.

2. Activity One — Generative Art Studio 🎨

Goal: use familiar input, variables and numeric expressions to influence a graphical artwork.

BEFORE YOU START
The turtle library and the for loop are PREVIEW concepts. You do not need to understand their internal details today.

2.1 Run the Program

1. Create a new Python file.
2. Type or copy the program below.
3. Run the program in a Python environment that supports turtle graphics.
4. Enter your artist name, artwork size, pen width and colours.
5. Observe how your inputs affect the generated image.

```
import turtle

print("=" * 55)
print("PYTHON GENERATIVE ART STUDIO".center(55))
print("=" * 55)

artist = input("\nArtist name: ")
shape_size = int(input("Artwork size (50-200): "))
pen_width = int(input("Pen width (1-10): "))
background = input("Background colour: ")
pen_colour = input("Drawing colour: ")

print("\nArtist:", artist)
print("Generating your artwork...")

# PREVIEW: graphics capability
screen = turtle.Screen()
screen.bgcolor(background)
screen.title(artist + "'s Generative Art")

pen = turtle.Turtle()
pen.color(pen_colour)
pen.width(pen_width)
pen.speed(0)

# PREVIEW: repetition
for i in range(120):
    pen.forward(shape_size)
    pen.right(123)
    shape_size = shape_size * 0.98

pen.hideturtle()
print("Artwork complete!")
turtle.done()
```

2.2 Find the Python You Already Know

Code	What you already know
artist = input(...)	input + variable + assignment
shape_size = int(input(...))	input + conversion
artist + "'s Generative Art"	string concatenation
shape_size * 0.98	numeric expression
shape_size = shape_size * 0.98	reassignment
"=" * 55	string repetition

KEY INSIGHT
An advanced-looking program can still be built around very simple foundations. You do not need to understand every component before you can understand part of the program.

2.3 Experiment — One Number, Different World

Find this line:

```
pen.right(123)
```

Run the program, then try changing 123 to 121, 117 or another nearby value. Keep the rest of the program unchanged.

YOUR THINKING: What changed in the artwork? Why is it interesting that only one number changed?

TAKEAWAY
Complex patterns can emerge from simple rules repeated many times. This idea appears in graphics, simulations, games and many algorithms.

2.4 Your Challenges

- Choose a different background and pen colour.
- Change the starting artwork size.
- Predict what changing pen_width will do before running.
- Try several turning angles and compare the geometry.
- Highlight every line that uses Week 1–2 knowledge.
- Put a star beside every PREVIEW line.

3. Activity Two — Galactic Mission Simulator 🚀

Goal: combine user-generated data, computer-generated data and calculated data in one simulation.

PREVIEW
random is a Python library. random.randint(a, b) gives the program a generated integer in a specified range. You do not need to learn how random-number generation works yet.

3.1 Run the Program

```
import random

print("=" * 60)
print("GALACTIC MISSION SIMULATOR".center(60))
print("=" * 60)

commander = input("\nCommander name: ")
spaceship = input("Spaceship name: ")

engine_power = int(input("Engine power (1-100): "))
shield_power = int(input("Shield power (1-100): "))
weapon_power = int(input("Weapon power (1-100): "))

print("\nCommander", commander)
print(spaceship, "is entering unknown space...")
print("." * 60)
```

```
# PREVIEW: computer-generated values
enemy_power = random.randint(50, 150)
asteroids = random.randint(1, 20)
luck = random.randint(1, 10)

# WEEK 1-2: calculations
ship_power = engine_power + shield_power + weapon_power
battle_score = ship_power * luck
damage = enemy_power * asteroids
mission_score = battle_score - damage

print("\n" + "=" * 60)
print("BATTLE REPORT".center(60))
print("=" * 60)
print("Commander".ljust(25), commander)
print("Spaceship".ljust(25), spaceship)
print("-" * 60)
print("Ship Power".ljust(25), ship_power)
print("Enemy Power".ljust(25), enemy_power)
print("Asteroids".ljust(25), asteroids)
print("Luck Factor".ljust(25), luck)
print("-" * 60)
print("Battle Score".ljust(25), battle_score)
print("Damage".ljust(25), damage)
print("FINAL MISSION SCORE".ljust(25), mission_score)
print("=" * 60)
```

3.2 Three Sources of Data

Data source	Example	Meaning
Human-generated	engine_power = int(input(...))	The user supplies information.
Computer-generated	enemy_power = random.randint(...)	A library generates information.
Derived	mission_score = battle_score - damage	Expressions create new information.

KEY INSIGHT
Programs often combine information from several sources. The important programming skill is understanding where data comes from, what type it has and how it is transformed.

3.3 The Same Inputs Experiment

6. Choose one set of engine, shield and weapon values.
7. Run the program and record the mission score.
8. Run it again using exactly the same human inputs.
9. Repeat once more.
10. Compare the results.

YOUR THINKING: Why can the results differ even though your input values are identical?

Look carefully at enemy_power, asteroids and luck. These values are generated again on each run.

3.4 Design Challenge

- Add pilot_skill as another integer input.

- Add solar_storm = random.randint(1, 20).
- Create your own formula that includes pilot_skill or solar_storm.
- Explain which values are input, generated and derived.
- Predict which variable has the greatest effect on your score.

3.5 A Question the Program Cannot Yet Answer

Suppose mission_score is positive. Perhaps the program should say MISSION SUCCESS. If it is negative, perhaps it should say MISSION FAILED.

YOUR THINKING: Why can our Week 2 program calculate the score but not yet choose which message to display?

LOOKING AHEAD
Week 3 introduces Boolean expressions and conditional statements. They allow programs to make decisions.

4. Activity Three — Personal Data Scientist

Goal: see how familiar variables and expressions can become a data visualisation.

PREVIEW
The square-bracket values below are lists, which you will study later. Matplotlib is a visualisation library. Focus today on how your familiar Week 1–2 values become the data used by the chart.

4.1 Run the Program

```
import matplotlib.pyplot as plt

print("=" * 55)
print("PERSONAL STUDY ANALYTICS".center(55))
print("=" * 55)

student = input("\nStudent name: ")

monday = float(input("Monday study hours: "))
tuesday = float(input("Tuesday study hours: "))
wednesday = float(input("Wednesday study hours: "))
thursday = float(input("Thursday study hours: "))
friday = float(input("Friday study hours: "))

total = monday + tuesday + wednesday + thursday + friday
average = total / 5

print("\n" + "-" * 55)
print("STUDY REPORT")
print("-" * 55)
print("Student:", student)
print("Total study time:", total, "hours")
print("Average per day:", average, "hours")

# PREVIEW: lists
```

```
days = ["Mon", "Tue", "Wed", "Thu", "Fri"]
hours = [monday, tuesday, wednesday, thursday, friday]

# PREVIEW: visualisation
plt.bar(days, hours)
plt.title(student + "'s Study Week")
plt.xlabel("Day")
plt.ylabel("Study Hours")
plt.show()
```

4.2 What Is Already Familiar?

- student = input(...) — strings, variables, input and assignment.
- float(input(...)) — input and type conversion.
- total = monday + ... — numeric expressions.
- average = total / 5 — deriving new information.
- student + "'s Study Week" — string concatenation.
- print(...) — communicating results.

KEY INSIGHT
Matplotlib does not invent the information. Your variables contain the data; your expressions process it; the library provides a powerful new way to communicate it.

4.3 Make the Data Yours

Replace study hours with another five-value dataset. Examples include exercise hours, screen time, temperatures, rainfall or sensor readings.

YOUR THINKING: What data did you choose? Which variables would you rename to make the program meaningful?

5. Known or Preview?

For each line, decide whether it uses Week 1–2 knowledge or is a preview.

```
artist = input("Artist: ")
size = int(input("Size: "))
result = size * 2
print("Result:", result)

import turtle
for i in range(100):
    pen.forward(size)

enemy = random.randint(1, 100)
plt.bar(days, hours)
```

Line / idea	KNOWN or PREVIEW?	Why?
input(...)		
int(input(...))		
size * 2		
print(...)		
import turtle		

for ...		
random.randint(...)		
plt.bar(...)		

6. Three Applications — One Foundation

Application	Extra capability	Week 1–2 foundation	Where it leads
Generative Art	Graphics	values, input, int, strings, expressions, assignment	loops and graphics algorithms
Mission Simulator	Randomness	input, conversion, arithmetic, variables	conditionals and simulation
Data Scientist	Visualisation	float, variables, expressions, strings	lists and data science

BIG IDEA
The applications look very different, but the same foundational ideas keep appearing. That is why learning fundamentals matters.

7. What Does import Mean?

For now, use this simple mental model:

```
import turtle
import random
import matplotlib.pyplot as plt
```

An imported library gives your program access to capabilities written by other programmers.

KEY TAKEAWAY
Professional programming is not about building everything from zero. Programmers combine fundamental concepts, their own problem-solving logic, and reusable libraries.

8. Where These Ideas Can Eventually Lead

Python ecosystem	Example capability
turtle	graphics and algorithmic drawing
random	games, sampling and simulation
Matplotlib	data and scientific visualisation
Pandas	data analysis
OpenCV	image processing and computer vision
PyTorch	machine learning and AI
Pygame	2D games
Flask / Django	web applications

You are not expected to learn these libraries now. They are shown to help you see where the foundations you are learning can lead.

9. Key Insights to Remember

- Unfamiliar code does not mean the entire program is unfamiliar.
- Identify values, types, variables, inputs, conversions, expressions and outputs first.

- Imported libraries add capabilities; your program still needs data and logic.
- The same foundation can support graphics, simulation and data science.
- Input values and computer-generated values can be combined in expressions.
- Expressions transform existing information into new information.
- Small changes to values can produce large changes in program behaviour.
- Programming is increasingly about combining components intelligently, not writing every capability from scratch.

10. Week 1–2 Connection

```
name = input("Name: ")
value = float(input("Value: "))
result = value * 10
print(name, result)
```

This tiny program contains much of the core foundation:

Concept	Where it appears
String	prompt text and name
Input	input(...)
Conversion	float(...)
Variable	name, value, result
Assignment	=
Numeric expression	value * 10
Output	print(...)

CORE MESSAGE
A sophisticated library does not replace these foundations. It builds on them.

11. Reflection Questions

1. Which of the three programs surprised you most, and why?

2. Which lines in the Generative Art program were already familiar?

3. What are the three sources of data in the Mission Simulator?

4. Why does the Mission Simulator produce different results on different runs?

5. What role do your variables play in the Data Scientist program?

6. What does an imported library add to a program?

7. Why is it useful to separate KNOWN code from PREVIEW code?

8. How do these examples change your view of what Python can be used for?

5. Why is mission_duration better than x?

6. How are x = x + 1 and x += 1 related?

7. What do sep and end control?

8. Can you describe a program as input → process → output?

16. Mini Build — Personal Mission Card

Use only Week 1–2 concepts. Ask for a name, mission name, speed and duration; calculate distance; then produce a clear report.

```
print("=" * 50)
print("PERSONAL MISSION CARD".center(50))
print("=" * 50)

# INPUT


# PROCESS


# OUTPUT
```

EXTENSION
Make it look like a small application using meaningful names, comments, string repetition, sep/end and optional center()/ljust().

17. Week 2 Takeaway

Week 2 is not a collection of unrelated commands. It is about how data moves through a program: users provide information; Python represents it using values and types; variables give it names; expressions transform it; and output communicates the result. Next, you will learn how a program can make decisions based on that data.