# L-1
# Programming in the Age of AI

ENGG1330
Computer Programming I



SCHOOL OF
COMPUTING &
DATA SCIENCE
The University of Hong Kong

# If AI Can Write Python, Why Are You Here?

AI can generate code in seconds

So what is left for you to learn?

# AI Changes the Work

AI can already help us:

- draft code
- explain unfamiliar code
- translate between languages
- find and repair some errors
- automate routine work

It is getting better quickly

# Use Is High, but Trust Is Not

- **84%** use or plan to use AI tools in development
- **46%** distrust the accuracy of AI output
- **66%** are frustrated by solutions that are almost right

2025 Stack Overflow Developer Survey

# The Hard Part Starts Before the Code

- What problem are we solving?
- What assumptions are we making?
- What information matters?
- What would count as correct?
- Who is responsible for the result?

AI can generate code. You must decide what the code should mean

# A Simple Question

You travel from A to B at **60 km/h**

You return over the same distance at **40 km/h**

What is your average speed?

**40 · 48 · 50 · 60 km/h**

# The Obvious Answer Is Wrong

Arithmetic mean of the speeds: **50 km/h**

Average speed is total distance divided by total time

Actual average speed: **48 km/h**

The problem was not Python syntax. The model was wrong

# Perfect Code Can Implement a Bad Model

```
outbound_speed = 60
return_speed = 40

average_speed = (outbound_speed + return_speed) / 2
print(average_speed)
```

Runs without an error. Still gives the wrong answer

# Computational Thinking Connects the World to Code

**Frame → Represent → Design → Express → Test → Improve**

Coding is one stage of the process

# These Skills Belong to Every Engineer

- model a physical process
- analyse measurements
- automate repeated work
- simulate competing designs
- control a device
- test whether an idea behaves as expected

# Code Makes Ideas Executable

During this course, you will build a runnable project of your own design

Selected projects will be exhibited for your classmates to play and vote on

# The Course Builds One Layer at a Time

| Lectures | Focus |
|---|---|
| L1–L3 | Values, decisions, repetition, and collections |
| L4–L7 | Files, functions, modules, and data behaviour |
| L8–L10 | Search, sorting, recursion, and integrated problem solving |
| L11 | Project game show and course wrap-up |

# Train Independently, Then Practise with AI

**Programming assignments**

Build independent fluency without AI, collaboration, or pasted code

**Programming project**

Use outside tools, including AI, with full disclosure and responsibility

# How the Course Works

- Common lecture for concepts, demonstrations, and participation
- Nine subclass tutorials for guided practice and project development
- A self-formed project group of three or four students in one tutorial
- Bring a laptop to every tutorial

# Assessment Rewards Understanding and Building

| Component | Weight |
|---|---|
| Written examination | 50% |
| Three programming assignments | 18% |
| Programming project | 16% |
| Lecture participation | 8% |
| Tutorial participation | 8% |

# Know Where the Authoritative Version Lives

- **Moodle:** official course information and released materials
- **Ed:** assessed programming, activities, and course questions
- **Python on Ed:** authoritative for assessed program behaviour

# Today You Will Make an Interactive Program

- store information in variables
- receive input
- produce output
- calculate with expressions
- test whether the result makes sense

# A Program Transforms Input into Output

**Input → Instructions and data → Output**

Name → greeting

Journey details → average speed

# Your First Program Is Already Running

main.py

```
1 name = input("What is your name? ")
2 print(f"Hello, {name}!")
3
```

# Variables Give Information a Name

```
distance = 120
outbound_speed = 60
return_speed = 40
```

Names let us describe the problem in the language of the problem

# Input Arrives as Text

```
distance_text = input("One-way distance in km: ")
```

`input()` always returns text

# Convert Text Before Calculating

```
distance_text = input("One-way distance in km: ")
distance = float(distance_text)
```

`float(...)` converts suitable text to a number

# Expressions Encode Relationships

```
total_distance = 2 * distance
time_out = distance / outbound_speed
time_back = distance / return_speed
total_time = time_out + time_back
average_speed = total_distance / total_time
```

The expression should match the model, not merely look plausible

# Build the Model, Then Run It

main.py

```python
distance = float(input("One-way distance (km): "))
outbound_speed = float(input("Outbound speed (km/h): "))
return_speed = float(input("Return speed (km/h): "))

total_distance = 2 * distance
total_time = distance / outbound_speed + distance / return_speed
average_speed = total_distance / total_time

print("Average speed:", average_speed, "km/h")

```

# Test Cases Are Evidence

| Journey | Expected result |
| --- | --- |
| Same speed in both directions | That same speed |
| 60 km/h out, 40 km/h back | 48 km/h |
| Zero or negative speed | The model needs a rule |

A program is not finished because it ran once

# Different Failures Tell Us Different Things

- **Syntax error:** Python cannot read the instructions
- **Runtime error:** execution reaches an impossible operation
- **Logic error:** the program runs but gives the wrong result

# Activity 1: Repair the Calculator

main.py

```
1 width = input("Width: ")
2 height = input("Height: ")
3
4 area = width * height
5 print("Area:", area)
6
```

**Predict → Run → Repair → Test → Explain**

# Why Learn to Code in the Age of AI?

To make vague problems precise

To test whether proposed solutions are correct

To turn ideas into systems you can understand and improve

AI can produce an answer. Computational thinking helps you decide whether it is the right answer to the right problem