



BK30730 · 29 SEPTEMBER 2026

Dive Into Coding

A Fun Introduction to Computer Science

Algorithms · Programming · Data Structures · Data Analysis

Dr Keith L Mannoock · Birkbeck, University of London



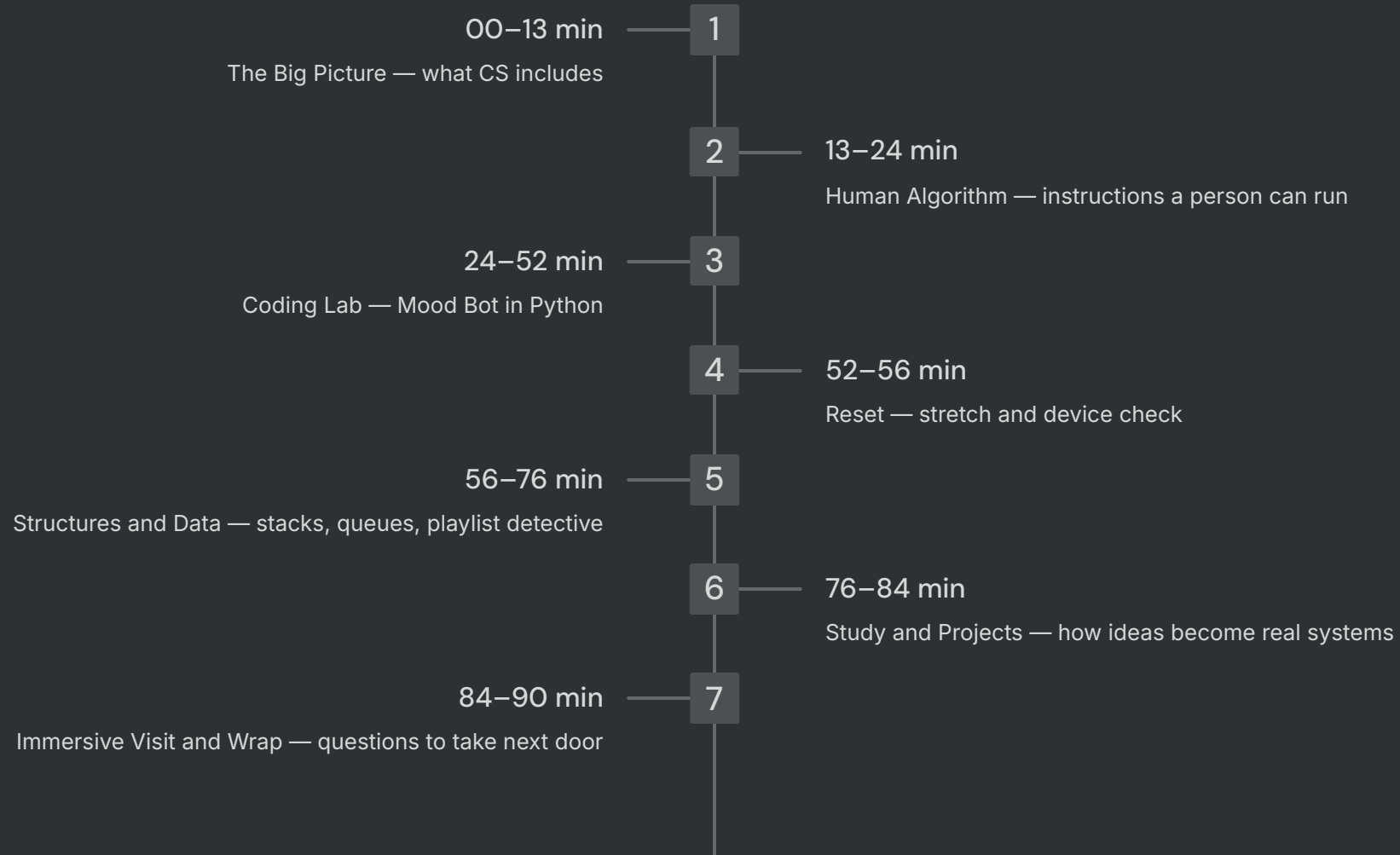
Computers Are Brilliantly Literal

The interesting part is deciding **exactly** what should happen. A computer will follow a bad instruction just as faithfully as a good one — without hesitation, complaint, or common sense. It does precisely what you say, not what you mean.

- Today we will design instructions, write real code, handle data, and debug the surprises along the way.

The 90-Minute Route

Here's how our session is structured. We'll move between explanation, hands-on activity, and reflection — so keep your browser open and your curiosity switched on.



Who Is in the Room?

A quick show of hands — no right or wrong answers! This helps us pitch the session at the right level for everyone here.

Written any code?

Anything counts — Python, Scratch, JavaScript, HTML...

Changed a spreadsheet formula?

SUM, IF, VLOOKUP — that's already algorithmic thinking.

Made a game, webpage or animation?

Even a Scratch project or a Minecraft redstone circuit.

Wondered how an app decides what to show you?

Recommendations, filters, feeds — that curiosity is exactly what CS answers.

Think you need to be good at maths?

You don't. The core skill is logical thinking and problem-solving — breaking big messy problems into small manageable steps. If you enjoy puzzles or strategy games, you already think like a computer scientist.

Think coding is a solo activity?

Apps like Spotify, TikTok and Snapchat were built by teams — designers, security experts, data analysts, project managers. CS is one of the most collaborative fields out there.



Computer Science Below the Surface

When people are asked what Computer Science is, 9 times out of 10 the answer is "coding". And that's not wrong — but it's not the whole picture. Coding is the visible tip of the iceberg. The real creative power lies beneath the surface.

The Tip of the Iceberg — What People See

Apps · Games · Websites · AI Tools · The code we write

Below the Surface

- Algorithms — designing the best recipe to solve any problem
- Data Structures — choosing how to organise information
- Artificial Intelligence & Machine Learning — teaching computers to learn
- Cybersecurity — protecting systems and data from attack
- Human-Computer Interaction — making technology work for people

❏ A computer scientist is a digital architect, a puzzle solver, a security expert, and a creative designer — all rolled into one. The coding is just the paintbrush.



Five Pillars of Computer Science

Most real projects combine several of these at once. As you work today, notice how they keep appearing together.



Algorithms

The recipe for solving a problem — clear, efficient, step-by-step. Like Google Maps finding your fastest route.



Data Structures

How we organise information. A stack, a queue, a list — the choice changes everything about how fast a program runs.



Artificial Intelligence

Systems that learn from data. Feed a computer thousands of cat photos and it learns what a cat looks like — without being told.



Cybersecurity

A constant cat-and-mouse game between defenders and attackers. Part detective work, part engineering — absolutely vital.



Human-Computer Interaction

The design of technology for real people. Why your phone is easy to navigate and a bad app feels frustrating.

How to Work Today

This is a BYOD session — bring your device, bring your curiosity. No one needs to finish every challenge. The goal is to think, predict, and explore.

Pair Up 🤝

Share a device if that makes an activity easier. Two heads catch bugs faster than one.

Say Your Prediction 🔮

Before you press Run, explain out loud what you expect to happen. This is the key habit of a good programmer.

Ask Early 🙋

A neighbour or the presenter can help you get unstuck. Asking for help is a professional skill, not a weakness.



ACTIVITY 1 · PAIRS · 6 MINUTES

Human Algorithm

One person writes instructions. The other acts as the robot. Use only the allowed commands below — no pointing, no gesturing, and no fixing the route while the robot is moving. Your goal is to guide your partner from START to GOAL.

Allowed Commands

UP · DOWN · LEFT ·
RIGHT

Rules

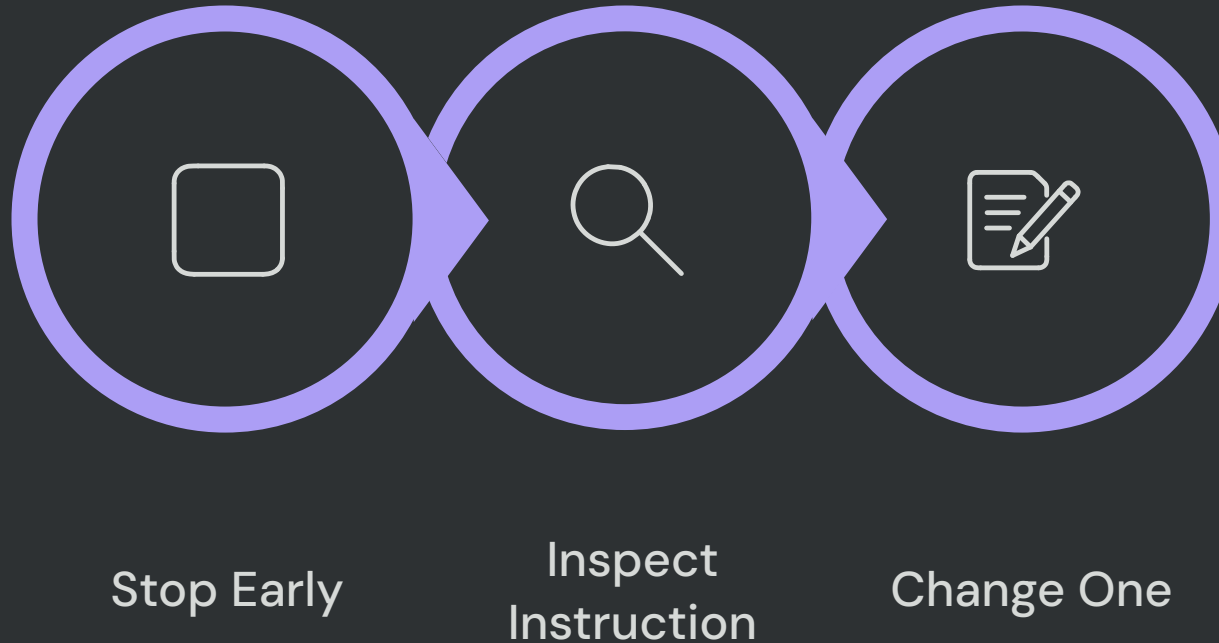
No pointing. No
correcting mid-run.

Challenge

Find a shorter
correct route once
you succeed.

Human Algorithm Checkpoint

If your route fails, don't start over from scratch. Use this three-step debugging process — it's the same technique professional programmers use every day.



□ That process — isolate, inspect, fix one thing, retry — is already debugging. You just did computer science.

What Makes an Algorithm Useful?

Not every set of instructions counts as a good algorithm. A useful algorithm has three essential properties — and you can test your human algorithm against all three right now.

Correct

It reliably reaches the goal every time, not just sometimes.

Clear

Someone else can read and run it without asking for clarification.

Efficient

It avoids unnecessary steps. Fewer moves = faster, cheaper computation.

A program is an algorithm expressed in a language a computer can execute. Everything else we do today builds on this idea.

From Commands to Code

The same logical idea can be written in plain English or in a programming language. The meaning is identical — only the audience changes. English is for humans; Python is for computers.

Human Command

Move right three squares

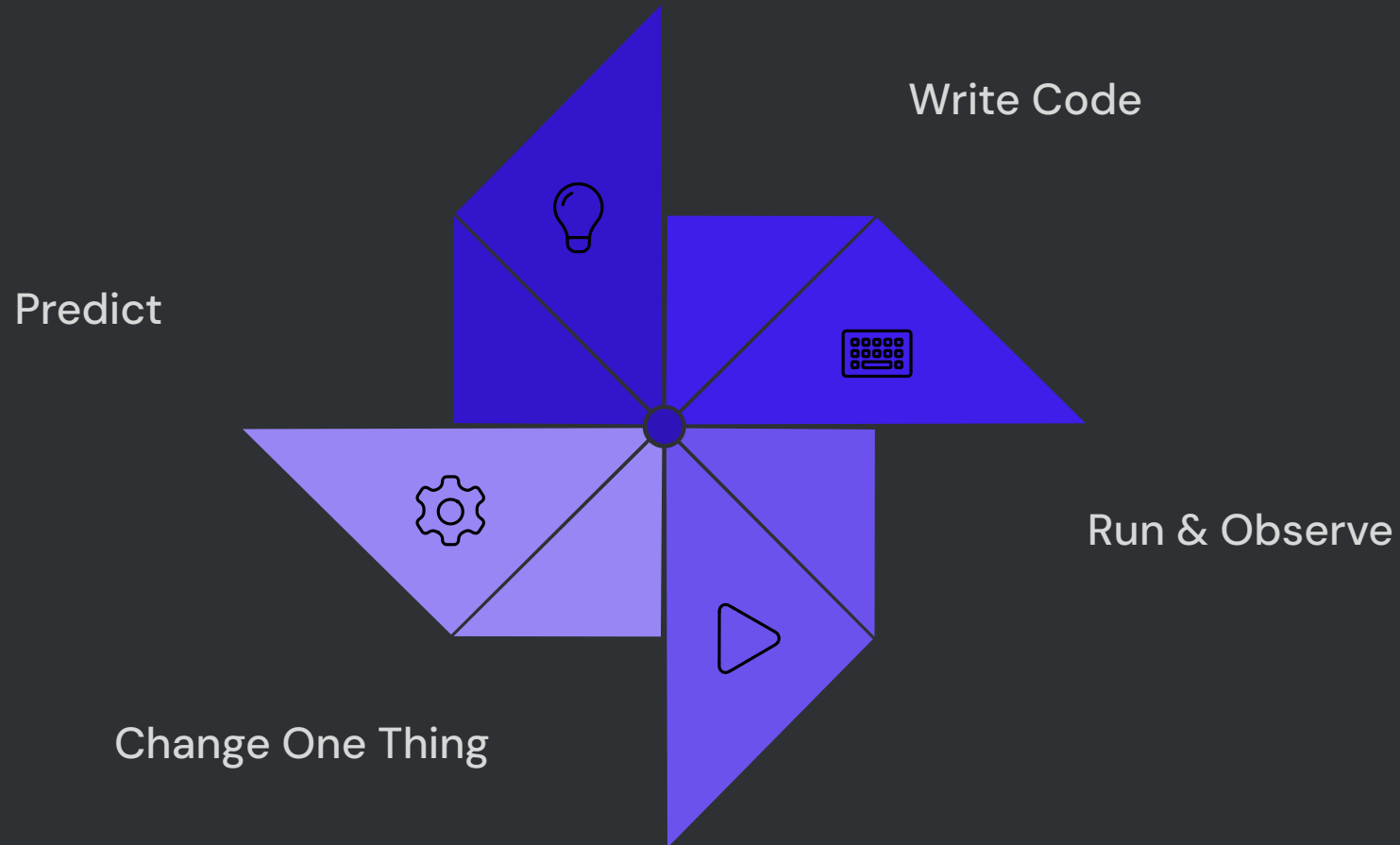
Python Instruction

```
for step in range(3):  
    move_right()
```

Both versions need **a clear action**, **a number of repetitions**, and **an agreed meaning for "right."** The translation from human thinking to code is the core skill of programming.

Programming as a Conversation

Good programmers don't just type and hope. They treat every run as an experiment — making a prediction, gathering evidence from the result, and refining their understanding. This loop is how software actually gets built.



Pair Programming Roles

Pair programming is a professional technique used in industry. Two people, one device — and both must understand every change. Swap roles after each successful run so both partners get time at the keyboard.

Driver

Types the code and runs the program. Focuses on syntax and the current line.

Navigator

Reads the instructions, predicts the result, and spots differences between expected and actual output.

📌 Both people should be able to explain the latest change at any moment. If you can't explain it, you haven't understood it yet.

Coding Lab: Open EduBlocks

app.edublocks.org

EduBlocks is a browser-based Python environment — no installation required. Open it on your device right now. It works on phones, tablets, and laptops.

01

Choose Python

Select the Python mode from the opening screen.

02

Start a Blank Project

Click New Project and choose the text editor view.

03

Find the Run Button

Locate the Run button and the output panel below the editor.

If EduBlocks Doesn't Open

No device? Slow connection? That's fine — this is a BYOD session and we have options for everyone. You'll still learn just as much.

1

Option 1

Share a device. Use one device between two people — this is actually how professional pair programming works.

2

Option 2

Predict on paper. Write the output you expect in Notes or on paper. Prediction is the most important skill here.

3

Option 3

Follow the live demo. Watch the screen and call out what the next line should say.

First Successful Run

One line is enough to start. Type this exactly — every character matters — then make your prediction before you press Run.

```
print("Hello, computer scientists!")
```

Before Run

Predict the **exact text** that will appear in the output panel. Write it down if you need to.

After Run

Change one word inside the quotes and run it again. Notice how the output changes. This is the predict-test loop in action.

Inputs and Variables

A variable gives a value a name so you can use it later. The `input()` function pauses the program and waits for the user to type something. That typed value is stored in the variable.

```
name = input("What is your name? ")  
print("Hello", name)
```

`input()`

Asks the user a question and waits for a response

`name`

A variable — it remembers whatever the user typed

`print()`

Displays a value in the output panel

Prediction: Where exactly will your own name appear when you run this?

Decisions with if

An `if` statement lets your program choose a different path depending on a value. Only the branch that matches actually runs — the rest is skipped entirely.

```
mood = input("curious or excited? ").lower()
if mood == "curious":
    print("Try changing one thing.")
elif mood == "excited":
    print("Add your own response.")
else:
    print("Any other answer arrives here.")
```

Watch the colon :

It signals that an indented block follows.

Watch the indentation

Four spaces tell Python which lines belong inside the `if`.

Watch `==`

Double equals means compare, not assign.

Starter Program: Mood Bot

Type this program carefully, predict the output for each mood, then press Run. Notice how variables, input, and decisions all work together.

```
name = input("What is your name? ")
mood = input("Pick a mood: curious, excited, sleepy ").lower()

if mood == "curious":
    reply = "Change one thing and predict what happens."
elif mood == "excited":
    reply = "Add your own response to level up."
else:
    reply = "Debug gently. Computers are very literal."

print("Hello", name)
print(reply)
```

📌 Before pressing Run — predict: what will appear if you type "sleepy"? What about "EXCITED" (in capitals)?

Mood Bot Checkpoint

Run three tests before you extend the program. If any test produces unexpected output, compare your code against the previous slide — look at spelling, punctuation, and indentation first.

Input you type	Expected reply
curious	Change one thing and predict what happens.
excited	Add your own response to level up.
banana (anything unexpected)	Debug gently. Computers are very literal.

If a test fails: compare spelling and punctuation first, then check your indentation — four spaces, not a tab.

CHALLENGE · 7 MINUTES

Make It Yours

Choose **one** extension and try it. Before you run: predict exactly which line will execute for your new input.

1

A — Add a Mood

Add a fourth `elif` branch with your own mood word and a custom reply.

2

B — Ask More

Add a second `input()` question and use the answer somewhere in the output.

3

C — Go Funny

Make the bot deliberately absurd or funny. Then explain the exact rule that causes it.

Show, Explain, Swap

Two minutes with the pair beside you. This is a mini code review — the same practice used in professional software teams to catch bugs and share ideas.



Show

Run one input that demonstrates your change. Let them see the output live.



Explain

Point to the exact line that controls the new behaviour. Can you say it in one sentence?



Ask

What other input should we test? What might break it? Challenge each other to find an edge case.

Debugging Is Part of the Work

Every programmer — beginner and expert alike — writes buggy code. The skill is learning to read error messages and fix one thing at a time. Can you spot the two bugs below?

```
if mood = "curious":  
    print("What happens here?")
```

Bug 1: = vs ==

= *assigns* a value. == *compares* two values. Inside an if condition you always need ==.

Bug 2: Missing indentation

The print line should be indented four spaces. Without it, Python doesn't know it belongs inside the if block.

Fast finishers: Repair both bugs and explain each fix to your partner before running.

Python Punctuation Map

Small marks change the meaning completely. Most beginner errors become manageable the moment you slow down and inspect one line at a time. Bookmark this reference.

=

Store a value into a variable. Example: `name = "Alice"`

==

Compare two values. Returns True or False. Example: `mood == "curious"`

:

Begin an indented block after `if`, `elif`, `else`, or `for`.

"text"

A string of characters. The quotes tell Python this is text, not a variable name.



52 minutes done 

Five-Minute Reset

Stand up, stretch, refill your water. Keep your browser open — we'll come back to it shortly.

Next: what changes when data arrives in a different order?

Data Structures: Order Changes Behaviour

The same items, stored with different rules, produce completely different results. This is why choosing the right data structure is one of the most important decisions a programmer makes.



Stack

Add and remove at the **same end**. The last item in is the first item out.

Think: a pile of plates. You always take from the top.

Used for: Undo history, browser back button, function calls.



Queue

Join at one end, leave at the other. First in, first out.

Think: people waiting for lunch. First to arrive is first to be served.

Used for: Print queues, message buffers, task scheduling.



Same items. Different rules. Different result. The structure is the algorithm.

List Anatomy

A Python list is the most versatile built-in data structure. Square brackets hold an ordered collection of values that you can access, change, and measure. Here's a real example we'll use in the next activity.

```
plays = [12, 4, 18, 7, 18, 3, 9, 18]
```

Name

`plays` — the variable that holds the whole list

Values

Eight play counts, each separated by a comma

Order

Left to right — position 0, 1, 2 ... 7

Brackets

`[]` tell Python this is a list, not a single value



ACTIVITY 2 · GROUPS OF 5 · 6 MINUTES

Stack or Queue?

No device needed for this one — stand up and act it out. Choose four song titles or snack names, then work through these three steps as a group.

01

Choose four items

Song titles, snack names — anything memorable. Write them on paper or just remember them.

02

Act as a stack

Add items one at a time. Remove them. Who comes out first?

03

Repeat as a queue

Add in the same order. Remove from the other end. Who comes out now?

Discussion: Which rule suits a playlist? Which suits an Undo button?

A List Stores Values in Order

Type this program and predict all four answers before pressing Run. This is your detective work — use mental arithmetic or estimation first.

```
plays = [12, 4, 18, 7, 18, 3, 9, 18]

print("tracks:", len(plays))
print("total plays:", sum(plays))
print("top play count:", max(plays))
print("average:", round(sum(plays) / len(plays), 1))
```

 Prediction first! Write down your four expected answers before you press Run. Compare — did any surprise you?

Playlist Checkpoint

Here is the expected output for the supplied list. If your output differs, compare your list values first (typos are common), then compare the function names character by character.

```
tracks: 8
total plays: 89
top play count: 18
average: 11.1
```

len()

Counts the number of items
in the list

sum()

Adds all the values together

max()

Finds the largest single
value

round()

Rounds to a given number of
decimal places

Change One Value

Replace the first `12` with `30` in your list. Before you press Run, answer these questions with your partner. Only run after both of you have made predictions.

What stays the same?

`len(plays)` will not change — you replaced a value, not added or removed one. The count stays at 8.

What changes?

`sum()` increases by 18 ($30 - 12$).
`average` will also increase. Does `max()` change? Think carefully.

Which can you predict without calculating?

You can reason about `len` instantly.
`max` requires comparing 30 against 18.
The average needs arithmetic — or trust the code.

STRETCH CHALLENGE · WHEN YOUR PAIR IS READY

Playlist Stretch Challenge

Choose one extension and try it. For each new line, explain out loud what question it is answering about the data.

1

A — Count a Value

```
print(plays.count(18))
```

How many times does 18 appear in the list?

2

B — Sort It

```
print(sorted(plays))
```

Show the values in ascending order — without changing the original list.

3

C — Your Data

Replace the list with eight numbers that mean something to you. What question will you investigate?

What the Program Just Did

Without realising it, you just combined three foundational areas of computer science in a single short program. This is exactly how real software works — nothing in isolation.



Data Structure

The list kept all the play counts together and in order, so you could process them as a group.



Algorithms

`len`, `sum`, and `max` are each small algorithms — precise steps that process the values.



Data Analysis

The outputs described the collection and revealed patterns — which track is most popular, what the average is.

Change one number and predict the effect — that's the analyst's instinct.



Real Projects Join the Ideas

A health app might feel simple to the user — but behind the screen, it draws on almost every area of computer science simultaneously. Here's what that might look like:

Code

The app itself — interface, buttons, screens

Algorithms

Reminder logic, pattern detection, calorie calculations

Data Structures

Health records stored in lists, queues, and databases

Analysis

Trend charts, averages, progress over time

Security

Protecting sensitive personal health information

Interaction

Designing for real people — including those with accessibility needs



Studying Computer Science at Birkbeck

At Birkbeck you learn the foundations — then use them to build, test, and defend your own work. Projects are not just exercises; they become your portfolio. Birkbeck's computing history reaches back to **Andrew and Kathleen Booth** and some of the earliest electronic computers ever built.

□ The most useful habit you can build is the one you practised today: predict · test · explain · improve.

How You Learn at Birkbeck

The degree is structured so that understanding and practice reinforce each other at every stage. By the time you graduate, you can talk about your work as confidently as you can write it.



Lectures Explain the Ideas

Theory, concepts, and the "why" behind each technique — giving you the mental models to think like a computer scientist.



Labs Turn Them Into Code

Supervised practical sessions where you write, debug, and test programs — exactly as you did today.



Projects Make You Combine and Defend

You design, build, and explain your own systems — combining algorithms, data, and design decisions into something real.



Where It Can Lead

Computer science is one of the most versatile degrees available. The skills transfer across every industry — and many of the most interesting roles haven't been named yet.



Software and Web Development

Building the apps and websites that billions of people use every day.



Cybersecurity and Digital Forensics

Protecting systems and investigating digital crime.



Data Analysis and Data Science

Finding patterns in large datasets to drive decisions in business, science, and policy.



AI and Machine Learning

Building systems that learn, predict, and make decisions — the fastest-growing field in tech.

NEXT STOP: IMMERSIVE LEARNING CENTRE

Take a Computer Scientist's Questions With You

As you experience the Immersive Learning Centre next door, don't just be a visitor — be an analyst. Ask these questions about what you see and feel.

- 1 What inputs does the experience detect?**
Touch, gesture, voice, position — what sensors are involved?
- 2 What data structure might represent the space?**
A grid? A graph? A 3D coordinate list?
- 3 What rule decides what happens next?**
Can you spot the if statement hiding inside the experience?
- 4 How would you test it for different people?**
Think about accessibility, different body sizes, different inputs.



Three Ideas to Take With You

The big picture in three lines.

1 — CS is broader than coding

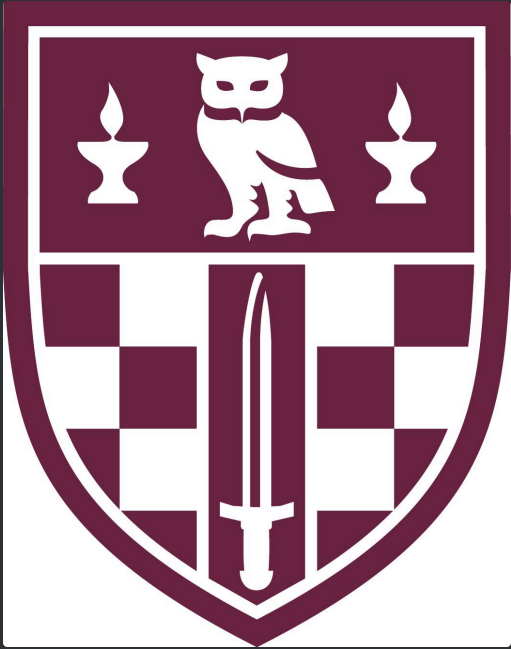
Algorithms, data, security, and interaction design are all part of the discipline. Code is just the way we express the ideas.

2 — Programs connect algorithms and data

Every program you write stores information in a structure and processes it with an algorithm. You've done both today.

3 — Debugging turns surprises into evidence

When something goes wrong, that's not failure — it's data. Read the message, isolate the cause, fix one thing, and learn.



Questions and Next Steps

Dr Keith L Mannock

School of Computing and Mathematical Sciences · Birkbeck,
University of London

 **Taster Course
Page**

[Programming in Python](#)

 **Keep Coding**

edublocks.org — free,
browser-based Python, no
sign-up needed

 **Explore More**

CS50 (Harvard), Codecademy, and Python.org all have free
beginner tracks to continue from here.

Scan to explore the support
materials



Thank you for diving in. 🚀