BUILD WITH IT

Python, from Zero, for AI

From your first line of code to your first API call.

First edition, September 2026

Addaly

Series editor: Dr Mustafa Mun

Draft, open for academic review

BUILD WITH IT

Python, from Zero, for AI

From your first line of code to your first API call.

Series editor: Dr Mustafa Mun

Addaly

First edition, September 2026 · Draft, open for academic review

Python, from Zero, for AI

First edition, September 2026, build 62a26075.

Written by AI models to a written standard, with Dr Mustafa Mun as series editor. Exam keys checked blind by a second model: 3,665 of 3,666 agreed. Academic review invited.

The manuscript was drafted with the assistance of a large language model and was edited and published under his name. That is stated plainly here rather than left to be discovered, because a reader is entitled to know how a book they are trusting was made.

How to cite: *Mun, M. (2026). Python, from Zero, for AI. Addaly. addaly.com/learn/python-for-ai.*

This book is the printed form of the course of the same name at addaly.com/learn/python-for-ai. The website is the living version and is corrected first. Where the two differ, the website is right.

Free to read, free to download, free to print, and free to teach from. There is no paid edition and there never will be.

Every diagram in it is drawn from data by code, not generated as a picture, so the numbers on the page are the numbers in the argument. The answers to every check, test and examination question are at the back.

9 modules · 89 lessons · 29 figures · 9 case studies · 69,907 words · about 12 hours

Brief contents

Contents 6

1 Getting Python running, and your first working code 11

2 Collections, loops, and the shape of your data 63

3 Functions, modules, and code you can read again in March 121

4 When it goes wrong: exceptions, debugging and tests 173

5 Data in and out: files, formats and the environment 225

6 Talking to a service: HTTP from Python, done properly 281

7 Objects, generators and the shape a larger program takes 349

8 Numbers in bulk: NumPy, pandas and the vector under every model 413

9 Building the thing: a small AI program, end to end 473

Examination 537

Answers 557

1

MODULE 1

Getting Python running, and your first working code

Install Python on a laptop, a phone or a free browser machine, run the same code both interactively and from a saved file, and predict what a short program prints by naming the type of every value in it and the branch a condition takes.

1	Getting Python onto the machine you actually have	12
2	What a program is, and running your first line	15
3	The interactive shell and the saved file, and when to use each	19
4	Variables, and the fact that everything has a type	23
5	Numbers: whole, decimal, and the one that surprises everybody	26
6	Strings, and why input() always hands you text	31
7	Building the output a person will read	35
8	True, False, and what Python counts as nothing	40
9	Making a decision: if, elif, else	45
10	What happens between your file and the answer	49
	Case study: Fourteen machines, no admin password, and a term that has already started	54
	Module 1 test	60

Lesson 1 · 9 min

Getting Python onto the machine you actually have

THE ONE THING TO KEEP

Install the stable release minus one, tick Add to PATH on Windows, and remember that `python3` on macOS and Linux is not the same command as `python`.

There are three starting points, and all of them are free

Most tutorials assume a laptop with an administrator password. Plenty of people reading this have a phone, or a shared computer they cannot install software on. All three cases work.

You already have a laptop or desktop. Install from `python.org/downloads`. On Windows, the installer shows a checkbox at the bottom of the first screen: **Add python.exe to PATH**. Tick it. If you do not, the installation succeeds and then every command in every tutorial fails with `'python' is not recognized`, and the fix is to run the installer again and choose Modify. This one checkbox is the single most common reason a beginner gives up in the first hour.

On macOS, the installer gives you a command called `python3`. On Linux, Python is already there; `python3 --version` will tell you.

You have a phone. On Android, install **Pydroid 3** from the Play Store — it ships a real Python with numpy and pandas already compiled, which matters, because compiling them yourself on a phone takes an hour. **Termux** (from F-Droid, not the Play Store version, which is abandoned) gives you a proper Linux shell and `pkg install python`. On iPhone, **a-Shell** is free and includes Python 3.

You have a browser and nothing else. Go to `colab.research.google.com` and start a new notebook. You get a free machine with Python, numpy, pandas and most of what this course needs,

already installed. Kaggle Notebooks are the same idea. The honest limitation: the machine is temporary. It disconnects after roughly 90 minutes of inactivity and at most 12 hours in a session, and when it goes, every file you created on it goes too. Download anything you want to keep, or connect it to Google Drive.

Check that it worked

Open a terminal — Command Prompt or PowerShell on Windows, Terminal on macOS and Linux, the app itself on a phone — and type:

```
python3 --version
```

You want something like `Python 3.12.4`. On Windows the command is usually just `python`.

The `python` versus `python3` split

This confuses everybody once. Python 2 and Python 3 are different languages with the same name. Python 2 was retired on 1 January 2020, but for twenty years `python` meant Python 2 on Unix systems, so macOS and most Linux distributions still refuse to let `python` mean Python 3 by default. Some now leave `python` pointing at nothing at all.

The rule that always works: type `python3` on macOS and Linux, `python` on Windows. If a tutorial says `python` and you get an error or a version starting with 2, add the 3.

The same split hits the installer: `pip` and `pip3`. Safer than both is `python3 -m pip`, which installs into *the same* Python you are running, and cannot get out of step. Lesson nine goes into why that matters.

Which version to install

Take the current stable release minus one. As of writing that means 3.12 rather than the newest. The reason is concrete: large libraries publish precompiled wheels for each Python version, and for a few months after a new

release those wheels do not exist yet. `pip install pandas` then tries to build from source, which needs a C compiler you probably do not have, and fails with 300 lines of red text. Being one version behind avoids a whole class of pain.

Anything from 3.9 upwards runs every example in this course.

An editor, and why not a word processor

You need something that writes plain text. Never Word, Pages or Google Docs — they insert curly quotation marks, and Python does not accept them.

- **IDLE** ships with Python. It is plain, and it is already installed.
- **Thonny** (`thonny.org`) is built for beginners: it shows variables changing as the program runs, which is genuinely useful for your first fortnight.
- **VS Code** is free, is what most jobs use, and is heavier. Install the Python extension after it.
- On a phone, Pydroid has its own editor.

Any of these is fine. Do not spend an evening choosing.

Make a folder now

Create one folder for this course — `Documents/python` will do. Put every file you write inside it. When lesson nine arrives with virtual environments, and lesson thirty-something with imports failing, half of those problems are really "which folder was I in". Starting tidy costs nothing.

> *The setup is not the skill. If installation fights you for more than half an hour, open Colab in a browser and start learning. You can come back and install later, with more context and less frustration.*

BEFORE YOU MOVE ON

Somebody on macOS follows a tutorial that says `python myfile.py` and gets `command not found: python`, though they installed Python yesterday and the installer reported success. What is actually going on?

A. The installation silently failed, so it needs to be uninstalled and reinstalled before anything will run.

B. macOS blocks Python for security reasons unless it is opened once through System Settings.

C. The installer provides the command as `python3`; the bare name `python` is not guaranteed to point at anything on macOS.

D. The PATH checkbox was not ticked during installation, so the command cannot be found.

The answer and why: p. 559

Lesson 2 · 6 min

What a program is, and running your first line

THE ONE THING TO KEEP

A program does exactly what is written, in order, and shows you only what you ask it to show.

A program is a list of instructions with no gaps

A program is a list of instructions written so a machine can follow them without asking you anything. That last part is the whole difficulty.

If you tell a person "add up the prices and tell me the total", they fill in what you left out. They know prices are numbers. They know to skip the blank line. They know what to do if one price is missing. A computer fills in nothing. It

does exactly what is written, in the order it is written, and stops the moment something does not make sense.

Python is one way of writing those instructions. It was designed to look close to English, which helps you read it, and occasionally fools you into thinking it will guess what you meant. It will not.

Two places to type Python

The interactive prompt. Open a terminal and type `python3` (on Windows, usually `python`). You get a prompt like this:

```
Python 3.12.3
>>>
```

Everything you type after `>>>` runs the instant you press Enter. This is for trying things.

A file. Put your instructions in a file called `hello.py`, then run it:

```
python3 hello.py
```

This is for programs you want to keep. Both run the same Python.

If `python3` is not found, install Python from python.org, or with your system's package manager.

Your first line

Type this into a file and run it:

```
print("Hello from Lagos")
```

Output:

```
Hello from Lagos
```

`print` is an instruction that puts something on the screen. The round brackets hold what you want printed. The quote marks say "this is text, not a command". Without the quotes, Python would look for something named `Hello` and fail.

Lines run top to bottom, one at a time

```
print("first")
print("second")
print(2 + 2)
```

Output:

```
first
second
4
```

Python read line one, did it, then line two, then line three. It does not look ahead. It does not reorder. This sounds obvious now and will save you an hour later.

Notice the third line: `2 + 2` was worked out to `4`, and `print` put that on the screen. Python does arithmetic with `+ - * /`. Try `print(7 / 2)` and you get `3.5`, not `3`.

The difference that catches everyone once

At the interactive prompt, typing an expression shows you its answer:

```
>>> 2 + 2
4
```

In a file, it does not. Put a file containing only this line:

```
2 + 2
```

Run it. Nothing appears. The file is not broken. Python did the addition, produced `4`, and threw it away, because nothing asked for it to be shown. The interactive prompt prints results as a convenience to you while you experiment. A running program has nobody to be convenient for.

So in files: if you want to see it, `print` it.

Notes to yourself

A `#` means the rest of the line is for humans and Python ignores it:

```
# prices are in naira
print(2500 * 3)
```

Try this now

Make a file, put four `print` lines in it, and deliberately break one: remove a closing bracket. Run it. Read what Python says. You are going to see a lot of those messages, and lesson seven is entirely about reading them.

BEFORE YOU MOVE ON

A student writes a file containing exactly one line, `2 + 2`, and runs it with `python3 sums.py`. The terminal prints nothing at all and gives no error. What happened?

A. Python worked out the answer 4 and discarded it, because nothing in the file asked for it to be displayed.

B. Python skipped the line, because a value that is not stored in a variable is never actually calculated.

C. The file failed silently, because arithmetic has to be inside a function before Python will run it.

D. Nothing printed because 4 is a number, and print-style output only ever shows text in quotes.

The answer and why: p. 559

CASE STUDY · MODULE 1

Fourteen machines, no admin password, and a term that has already started

An illustrative case, built from documented patterns.

A government higher secondary school in Bhopal, first week of the computer science term

Kavita Salve teaches computer science to thirty-two students of class eleven in a government school in Bhopal. The lab has fourteen desktops that switch on, all running Windows 10, all locked down by the district's IT contractor. The administrator password is with the contractor. He visits once a month and his next visit is in nineteen days.

She had two options and both cost something.

The first was to teach the whole term in a browser. A free notebook service starts in about thirty seconds, needs nothing installed, works on the four students' phones as well as on the desktops, and cannot be broken by anything a student does. The cost is that the lab shares a four megabit connection with the office, and every morning at about twenty past eleven the block's attendance upload saturates it for fifteen minutes. It is also, in her words, a place where code goes to be forgotten: the notebook hides where the file is, students never save anything they can find again, and a program becomes something that only exists while a tab is open.

The second was to get Python on the machines. The installer from python.org wants administrator rights. The store version does not, but the store requires a signed-in account and the machines have none. She could wait nineteen days for the contractor, or she could write to the district office and wait longer.

A colleague suggested a third way that she had not considered: a portable build of Python, copied from a USB stick into each machine's own user folder, which needs no administrator and no store. It runs. It also cannot tick the Add Python to PATH box, because there is no installer doing the ticking, so the command is not python. It is a path eleven characters longer, typed at the start of every line, or a batch file she writes for them.

The decision was not really about Python. It was about what the first three lessons of the term were going to be. Three lessons on installation is three lessons of a fifteen-week term spent on something no exam asks about, with a real chance that on lesson three four machines still do not work and those students have done nothing. Three lessons on code means the browser, and the browser means that in February, when a student sits at a cousin's laptop and wants to run what she wrote in September, none of it is there and none of the commands she knows apply.

The head teacher, who signs the internet bill, had a view of his own. He wanted the browser, because a lab of machines with software installed by a teacher is a lab he gets asked questions about when something stops working.

Kavita made her choice on the strength of one thing she had seen the previous year. In the board practical, students are given a printed program and asked what it prints. Her class had done badly on it — not because they could not code, but because they had only ever seen code in a cell that prints the value of the last line automatically. Asked what a saved file prints, half of them included values the file never printed. That is a gap the browser had created and she could see it in the marks.

YOUR ANSWERS

1. The browser option was faster to start and worked on phones. What did it cost, in a way that showed up in marks rather than in the lesson?

2. Kavita could not tick Add Python to PATH with the portable build. Why does that box matter, and what does its absence actually change?

3. Two students learned at home that python and python3 are not the same command. Why is that difference worth teaching early rather than treating as a detail?

STOP HERE

Write your answers before you turn the page. How it played out starts on p. 58.

This page is blank so that how it played out stays out of view until you turn the page.

CASE STUDY · MODULE 1

How it played out

She split the term. Weeks one and two ran in the browser, so that every student wrote and ran code on day one, including the six who had only a phone. In week three she copied the portable Python onto all fourteen machines from a USB stick — one lesson, forty minutes, twelve machines working by the end — and wrote a one-line batch file called py.bat so the command was short enough to type.

From week three, one rule: everything is a saved file, run from the terminal, and the browser is only for trying a single line. She kept the notebook for exactly that, because the shell genuinely is the better place to ask what type() says about a value.

Two machines refused to run anything from a user folder; the policy blocked executables there. Those four students worked in pairs for the rest of the term, which she disliked and could not fix.

In the February practical, twenty-six of thirty-two correctly predicted the output of a printed program, against fourteen of thirty the previous year. The commonest remaining error was the one the module warns about: reporting a value that the file computed but never printed.

The contractor came on day nineteen and installed Python properly on all fourteen machines in about ten minutes, with the PATH box ticked. Kavita kept the batch file anyway. Two students had by then installed Python at home themselves, and both had hit the same thing: on a Mac borrowed from an uncle, python was not a command and python3 was.

The questions, discussed

1. The browser option was faster to start and worked on phones. What did it cost, in a way that showed up in marks rather than in the lesson?

A notebook prints the value of the last expression in a cell automatically. A saved file prints only what you tell it to print. Students who had only ever worked in a notebook formed a wrong model of what a program shows you,

and it appeared in the practical, where they listed values the file computed but never printed. The gap was invisible while they were coding, because in the notebook their mental model and the tool agreed.

2. Kavita could not tick Add Python to PATH with the portable build. Why does that box matter, and what does its absence actually change?

Ticking it adds the Python folder to the list of places the shell searches for a command, so that typing python or py works from any folder. Without it, nothing is broken — Python runs perfectly well — but every command must name the full path to the interpreter, which is long, error-prone and different on every machine. Her batch file recreated the effect for one command in one place, which is the practical fix when you cannot change the system.

3. Two students learned at home that python and python3 are not the same command. Why is that difference worth teaching early rather than treating as a detail?

On macOS and most Linux systems python3 is the interpreter and python is either missing or, historically, a very old version, while on Windows the installer usually provides python and py. A learner who has memorised one command believes their code has stopped working when they move machines, and the error — command not found — says nothing about versions. Knowing that the command names the interpreter, and that a machine can have several, turns a mysterious failure into a one-line check with python3 --version.

MODULE 1 TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record. The answers, and the reason behind each, are in the key on p. 558. If two go wrong, re-read the module before the exam.

1 At the interactive prompt a student types total = 5 + 3 and then total, and sees 8. She puts exactly those two lines in a file and runs it. Nothing appears. What is going on?

A. The file must be saved before running, so Python ran an older copy of it

B. A script shows only what you explicitly print

C. An assignment throws the value away, so total holds nothing in a file

D. Output from a file is held in a buffer that is only flushed when the program ends

2 A program reads two marks with input() and prints the larger. Given 9 and 10 it prints 9. Nothing crashes. Why?

A. input() strips leading digits, so 10 arrived as 0

B. Python compares the lengths of the two values first

C. Both values are text, compared character by character

D. The comparison worked and print displayed the wrong variable

3 Why is 0.1 + 0.2 == 0.3 False?

A. 0.1 and 0.2 have no exact binary form, so their stored sum is above 0.3

B. == compares floats by identity rather than by value

C. Python rounds every float to fifteen decimal places before comparing it

D. The addition is performed at lower precision than the values are stored at

4 What does -7 // 2 give, and why?

A. -3, because // truncates the decimal part towards zero

B. -3.5, because // and / do the same thing on negative numbers

C. -4, because Python rounds halves to the nearest even number

D. -4, because // rounds down, towards minus infinity

5 A grading script uses four separate if statements — if marks >= 75: grade = "distinction", if marks >= 40: grade = "pass", and so on — instead of elif. A mark of 90 comes out as "pass". Why?

A. Every condition is tested, and the last true assignment wins

B. Python evaluates if statements in reverse order of their thresholds

C. The first if consumed the value, so later comparisons saw nothing

D. Assigning to the same variable in two branches raises a silent error that resets it

6 A script does retries = typed_value or 3, where typed_value came from a form and is the integer 0 because the user genuinely wants no retries. What is retries?

A. 0, because or only substitutes the default when the left side is None

B. 3, because 0 is falsy and or hands back the right-hand operand

C. True, because or always produces a boolean

D. 0, because or returns its left operand whenever the types match

Choosing between them

Use a **list** when the items are the same kind of thing and their order or count matters. Four prices. Twelve months of rainfall. Every message in a conversation.

Use a **dict** when each piece has a different job and you will fetch it by name. One student's name, city and marks. One product's title, price and stock.

Figure 2.1

Positions, or named slots

list — prices = [120, 90, 200]

- Fetched by position: prices[2]
- Order is part of the meaning
- Duplicates are normal and kept
- A position that does not exist raises IndexError
- Right for: twelve months of rainfall, every turn in a conversation

dict — student = {"name": "Asha", …}

- Fetched by key: student["city"]
- You never look things up by position
- One key names one slot, so writing twice replaces
- A key that does not exist raises KeyError, or .get gives a default
- Right for: one student's name, city and marks

The honest answer to which one is usually both, nested: a list of turns, each turn a dict with a role and a content. That is the exact shape you will send to a model API, and you can already build it.

The honest answer to "which one" is usually: both, nested. This is the shape you will meet everywhere in AI work:

```
messages = [
    {"role": "user", "content": "Explain gravity in one
sentence."},
    {"role": "assistant", "content": "Things with mass pull on
each other."},
    {"role": "user", "content": "Now in Hindi."},
]

print(len(messages))                # 3
print(messages[0]["content"])       # Explain gravity in one
sentence.
print(messages[-1]["role"])         # user
```

A list, because the order of a conversation is the whole point and there can be any number of turns. Dicts inside it, because a turn has two named parts. Read `messages[0]["content"]` left to right: take the list, take item zero, that is a dict, take its `"content"` slot.

`messages[-1]` is the last item. Negative positions count from the end, which saves you writing `messages[len(messages) - 1]`.

Adding a turn

```
messages.append({"role": "assistant", "content":
"□□□□□□□□□□□□□□...□"})
print(len(messages))   # 4
```

When you call an AI API in lesson ten, you will send almost exactly this structure. You already know how to build it.

EXAMINATION

Python, from Zero, for AI — course exam

This paper covers the whole course:

- running Python and predicting what a short program prints
- lists, dicts, sets and loops
- functions, modules and project layout
- tracebacks, exceptions, logging and tests
- files, encodings, CSV, JSON and virtual environments
- HTTP, timeouts, retries, streaming and cost
- classes, generators, decorators and packaging
- NumPy and pandas
- the small AI program you assembled at the end

63 questions, the whole pool. The site draws 25 for a sitting, pass mark 70%. Work any 25 under your own conditions. Answers from page 617.

1 Module 1 · Getting Python running, and your first working code

A machine has several Pythons installed. Which command is safest for installing a package into the one you are about to run?

A. pip install requests

B. python3 -m pip install requests

C. pip3 install requests --user

D. sudo pip install requests, so every Python on the machine can see it

2 Module 1 · Getting Python running, and your first working code

A script prints three lines before a missing bracket on line 40. Run it and nothing at all is printed, only a SyntaxError. Why did the first three lines not run?

A. Output is buffered and lost when the program exits with an error

B. The error handler cleared the screen before printing the message

C. Python runs a file backwards from the last line when checking syntax

D. The whole file is compiled before any of it executes

3 Module 1 · Getting Python running, and your first working code

print("3" * 2) shows 33 and print(3 * 2) shows 6. What decided the difference?

A. The type of the operands, which selects what * means

B. The quotation marks, which make Python treat * as concatenation

C. The order of the operands, since text always comes first

D. An automatic conversion, which turns "3" into 3 only when the other side is a string

4 Module 1 · Getting Python running, and your first working code

round(0.5) gives 0 and round(1.5) gives 2. Is this a bug?

A. Yes: round() should always round halves upwards

B. No: Python rounds exact halves to the nearest even number

C. Yes: the float representation makes 0.5 slightly less than a half

D. No: round() returns an int, and converting a float to an int truncates towards zero

5 Module 1 · Getting Python running, and your first working code

total = 19.999 and print(f"{total:.2f}") shows 20.00. What is total afterwards?

A. 20.0, because the f-string rounded it

B. 20.00, stored with two decimal places from now on

C. Still 19.999

D. Undefined until the next assignment, because formatting consumes the value

6 Module 1 · Getting Python running, and your first working code

a = 1000; b = 1000; a is b is False, while the same code with 100 gives True. What is going on?

A. Larger integers are stored as floats, which are never identical

B. is compares values below 256 and identity above it

C. Small integers are cached, so is accidentally agrees

D. Assignment copies small numbers and references large ones

7 Module 1 · Getting Python running, and your first working code

A grade chain is written as if marks >= 40: grade = "pass" / elif marks >= 75: grade = "distinction". Nobody ever gets a distinction. Why?

A. elif is only checked when the preceding if raised an exception

B. Two thresholds on the same variable are collapsed by the compiler

C. The second condition is unreachable, because 75 also satisfies 40

D. The chain must be written from the lowest threshold upwards, and this one is

8 Module 2 · Collections, loops, and the shape of your data

items has four elements. Which expression raises IndexError?

A. items[-1]

B. items[len(items)]

C. items[len(items) - 1]

D. items[1:99]

Module 1 · Getting Python running, and your first working code

The module starts on p. 11.

MODULE 1 TEST

Module 1 test, Q1, p. 60

At the interactive prompt a student types total = 5 + 3 and then total, and sees 8. She puts exactly those two lines in…

Answer B: A script shows only what you explicitly print

Why: The REPL prints the value of every expression automatically — that is what read, evaluate, print, loop means. A script does not: Python evaluated total, produced 8, and discarded it because nothing asked for it. Adding print(total) makes the two behave the same.

Module 1 test, Q2, p. 60

A program reads two marks with input() and prints the larger. Given 9 and 10 it prints 9. Nothing crashes. Why?

Answer C: Both values are text, compared character by character

Why: input() always returns a str, so the comparison is between "9" and "10". Strings compare character by character, left to right, the way a phone book orders words, and "9" comes after "1". Nothing errors because comparing two strings is perfectly legal. Convert at the edge: int(input(…)).

Module 1 test, Q3, p. 60

Why is 0.1 + 0.2 == 0.3 False?

Answer A: 0.1 and 0.2 have no exact binary form, so their stored sum is above 0.3

Why: A float is a 64-bit IEEE 754 number with 53 bits of mantissa storing a binary fraction. Neither 0.1 nor 0.2 is exactly representable, so the nearest doubles are stored, their sum rounds to 0.30000000000000004, and 0.3 stores as a different double. Every language using doubles behaves this way; compare with math.isclose.

Module 1 test, Q4, p. 61

What does -7 // 2 give, and why?

Answer D: -4, because // rounds down, towards minus infinity

Why: Floor division rounds downwards, not towards zero, so -3.5 becomes -4 rather than -3. If you expected -3 you were thinking of truncation, which is what several other languages do. The value -4 with the wrong reason is still worth spotting: banker's rounding belongs to round(), not to //.

Module 1 test, Q5, p. 61

A grading script uses four separate if statements — if marks >= 75: grade = "distinction", if marks >= 40: grade = "pass", and so…

Answer A: Every condition is tested, and the last true assignment wins

Why: With separate ifs there is no chain, so a mark of 90 satisfies more than one condition and each matching branch runs in order. The later assignment overwrites the earlier, correct one, and nothing warns you. elif means otherwise-if: once a branch is taken the rest are skipped.

Module 1 test, Q6, p. 61

A script does retries = typed_value or 3, where typed_value came from a form and is the integer 0 because the user genuinely wants no…

Answer B: 3, because 0 is falsy and or hands back the right-hand operand

Why: or returns one of its operands, not a boolean, and it returns the left one only if it is truthy. Zero, empty strings and empty lists are all falsy, so a legitimate 0 is silently replaced by the default. When zero is a real answer, test explicitly: retries = 3 if typed_value is None else typed_value.

THE LESSON CHECKS

Lesson 1, p. 15

Somebody on macOS follows a tutorial that says `python myfile.py` and gets `command not found: python`, though they installed Python yesterday and the installer reported…

Answer C: The installer provides the command as `python3`; the bare name `python` is not guaranteed to point at anything on macOS.

Why: Install the stable release minus one, tick Add to PATH on Windows, and remember that `python3` on macOS and Linux is not the same command as `python`.

A The installation silently failed, so it…: Treats a missing command as a broken install, which is the intuitive reading when the only evidence is an error message.

B macOS blocks Python for security reasons…: Borrows the real macOS quarantine prompt for downloaded apps and applies it to a command that was never blocked.

D The PATH checkbox was not ticked…: Names a real cause of exactly this message, but that checkbox exists only in the Windows installer.

Lesson 2, p. 18

A student writes a file containing exactly one line, `2 + 2`, and runs it with `python3 sums.py`. The terminal prints nothing at all and…

Answer A: Python worked out the answer 4 and discarded it, because nothing in the file asked for it to be displayed.

Why: A program does exactly what is written, in order, and shows you only what you ask it to show.

B Python skipped the line, because a…: It feels wasteful for a computer to compute something it throws away, so it seems the calculation must have been skipped entirely.

C The file failed silently, because arithmetic…: Most code you see in examples sits inside something, so a bare line at the top of a file looks like it is missing a container.

D Nothing printed because 4 is a…: Every printing example so far had quote marks in it, which makes quotes look like the thing that causes output.

Lesson 3, p. 22

A learner tests `name.upper()` in the REPL, sees `'ASHA'`, copies the same three lines into `greet.py`, runs it, and sees no output. What has happened?

Answer A: The script ran fine and computed the value, but a file only shows what `print` is asked to show.

Why: The REPL prints the value of every expression automatically and a script prints nothing you did not ask for, so the same lines behave differently in the two places.

B The script needs `python3 -i` to…: Correctly remembers that `-i` shows more, but attributes the ordinary behaviour of scripts to a missing flag.

C `upper()` returns nothing when called outside…: Blames the method for the silence, which is plausible if you have met functions that return None.

D The file was saved with the…: Reaches for a file-level explanation when the behaviour is the same for any correctly named script.

BUILD WITH IT

Python, from Zero, for AI

From your first line of code to your first API call.

WHAT IT TEACHES	Takes someone who has never written a line of code to the point of calling an AI API from their own Python script. Every lesson has code you can paste into a terminal and run, and every lesson ends with a question about what the code actually does.
WHAT IS INSIDE	Nine modules and 89 lessons, 29 figures drawn from data, nine case studies, nine module tests, a 63-question examination, and every answer with the reason behind it.
LICENCE	CC BY-NC-SA 4.0. Copy, print, share and adapt it for any non-commercial purpose, with credit, under the same licence. There is no paid edition.
THE COURSE	Free to read, with the lessons, the films and the examination, at **addaly.com/learn/python-for-ai**

Scan to open the course

Addaly

First edition, September 2026 · build 62a26075