Lab 01 — Finding Your Way Around

Files, Folders, Zips & the Command Prompt

DURATION ~3 hours **COVERS** Lecture 1 **TASKS** 9 checks **CHECKER** `check.py`

Read this first — it is the most important box in the lab

There is **no Python to write today**. Not one line. Today is about the room the programming happens in.
Every semester, some very capable students lose their first month not to loops or variables, but to *"where did my file go?"* They write good code into a window that is secretly inside a zip file, press Run, and watch it vanish. That is not a failure of intelligence. Nobody ever showed them how the machine is arranged.
So: today you will make folders, break nothing, extract a zip properly, and type a few words into a black window. **Nothing in this lab can damage your computer or delete your work.** Every command you are asked to type either *looks* at something or *creates* something. If a step goes wrong, the worst outcome is a sentence of red text — and reading those sentences calmly is itself one of today's skills.

How this lab works

- Work top to bottom — each section builds on the one before it.
- **Warm-ups** (W1, W2, ...) are quick things to look at or try. Nobody collects them; they set up the task that follows.
- **Tasks** (TASK 1–9) are the deliverable. Today the deliverable is not a program — it is a **folder in the right shape**, plus a filled-in `answers.txt`.
- **Predict before you click.** Several warm-ups ask you to write down what you think will happen *before* you do it. Write it on this sheet. Being wrong on paper is free.
- Run `check.py` whenever you like. Your goal before leaving: **all 9 tasks PASS**. Show the final report to your instructor.
- **Done early?** Part B on the last pages is optional and unmarked — and it is where the command prompt starts to feel like a superpower.

Before you start: get the files, and a note on slashes

Download the `lab01` folder from the course LMS page. It contains exactly three things: `check.py`, `answers.txt`, and `supplies.zip`. Do not rename them.
Throughout this sheet a slash separates folders: `sorted/notes` means *the `notes` folder inside the `sorted` folder*. Windows writes that with a backslash, `sorted\notes`; macOS and Linux use the forward slash. Same idea, different punctuation. Nobody is wrong.

1 Where Everything Lives (~20 min)

Ref: Lecture 1 — a computer does exactly what you tell it, including where to look

Your computer stores files the way a building stores rooms: everything is inside something else, all the way up. A **folder** (also called a *directory*) holds files and other folders. That is the whole system. There is no second idea.

```
C:\                          <- the drive: the outermost box
   Users\
      ayesha\                <- your account: your own corner
         Desktop\
         Downloads\          <- where the browser drops things
         Documents\
            COMP102\         <- the folder you are about to make
               lab01\
                  check.py
```

Reading that top to bottom gives you a file's **path** — its full address:

`C:\Users\ayesha\Documents\COMP102\lab01\check.py`

A path is not jargon. It is the answer to "which one do you mean?" There may be forty files called `check.py` on this machine; there is exactly one at that address.

Two kinds of address

An **absolute** path starts at the very top (`C:\Users\...`) and works everywhere, like writing out a full postal address. A **relative** path starts from wherever you are standing right now (`sorted\notes`), like saying "second door on the left". Both are useful. Confusing them is the single most common cause of *"but the file is right there!"*

W1. EASY Open your file manager — **File Explorer** on Windows (Win + E), **Finder** on macOS. Click into `Downloads`. You are now standing somewhere. Where? Write the folder's full path here:

. . . . . .

W2. EASY On Windows, click the empty part of the bar at the top of the File Explorer window (the one showing `Documents` › `COMP102`). It turns into text: the real path. On macOS, press Option + Cmd + P to show the path bar at the bottom. **That bar is going to be your best friend today.**

W3. EASY Predict first, then check: if you make a folder called `lab01` inside `COMP102`, and `COMP102` is inside `Documents`, how many folder names appear in `lab01`'s path after the drive letter? . . . . . .

TASK 1 Your COMP 102 Home EASY `COMP102/lab01` ~10 min

Make yourself a permanent home for this course. Pick a place you will *find again* — `Documents` is ideal, the Desktop is fine, a random folder inside `Downloads` is not.

1. Make a new folder named exactly `COMP102`.

   Windows: right-click the empty space → New → Folder. macOS: right-click → New Folder. Or Ctrl + Shift + N.

2. Go *inside* it and make a folder named exactly `lab01`.

3. Move `check.py`, `answers.txt` and `supplies.zip` out of `Downloads` and into `lab01`.

   Select all three (click the first, **Shift**+click the last), **Ctrl** + **X** to cut, then **Ctrl** + **V** inside `lab01`. On macOS the paste-and-move shortcut is **Option** + **Cmd** + **V**.

Spelling counts: `COMP102` and `lab01`, no spaces, no "Comp 102". The checker is looking for those two names.
Write the full path of your new `lab01` folder here — you will need it again in Section 6, and again every week for the rest of the semester:

. . . . . .

2 A File's Name Has Two Parts (~20 min)

Ref: Lecture 1 — syntax is the machine reading your intent literally

Look at `check.py`. That name is really two pieces:

`check` · `py`

the name — yours to choose the extension — not yours

The **extension** is everything after the *last* dot. It is how your computer decides which program should open the file. `.py` means "give this to Python". Change it to `.txt` and the same bytes are suddenly a text document; the machine does not argue, it just believes you. This is Lecture 1's lesson in miniature: *the computer does exactly what you say.*

Windows hides extensions by default — turn that off now

Out of the box, Windows shows you `check` and not `check.py`. So `notes.txt` and `notes.py` both look like `notes`, and you cannot tell your program from your shopping list. Fix it once, today, on every machine you use:

Windows 11: open File Explorer → **View** → **Show** → tick **File name extensions**.
Windows 10: File Explorer → **View** tab → tick **File name extensions**.
macOS: Finder → Settings → Advanced → tick **Show all filename extensions**.

Do it now. The rest of this lab assumes you can see them, and so does every lab after it.

W4. EASY With extensions now visible, look at `check.py`, `answers.txt` and `supplies.zip` in your `lab01` folder. Three files, three extensions, three different icons. Which program does each icon belong to?

W5. MEDIUM Predict the extension, then say what would open it:

File name	Extension	Opened by...
`grades.csv`	. . . . .	. . . . .
`syllabus.md`	. . . . .	. . . . .
`crest.png`	. . . . .	. . . . .
`LICENSE`	. . . . .	. . . . .
`notes.txt.py`	. . . . .	. . . . .

The last two are the interesting ones. One has *no* extension. One has two dots — and only the last one counts.

W6. MEDIUM Somebody emails you `invoice.pdf.exe` and Windows is hiding extensions, so you see `invoice.pdf`. What is it really, and what happens when you double-click it?

This is not a hypothetical. It is how a good fraction of the world's malware gets in, and you have just immunised yourself against it.

3 The Suitcase: Zip Files (~25 min)

Ref: Lecture 1 — follow the recipe exactly; a skipped step is still a skipped step

A **zip file** is one file that has a whole folder squashed up inside it. It is a suitcase. You can peek through the lid, but you cannot live out of it.

The single most expensive mistake in first-year labs

On Windows, double-clicking a `.zip` opens a window that *looks exactly like a normal folder*. It is not. It is a preview.
If you open your code from that window, edit it, and save — Windows writes your work into a temporary copy that is deleted when the window closes. Your program runs once, then disappears forever, and nobody can get it back. Students lose whole afternoons this way.
The rule: always *extract* first. Never work inside the suitcase.

How to extract, properly

Windows. Right-click `supplies.zip` → **Extract All...** → a box appears showing where it will go. *Read that box.* It usually offers `...\lab01\supplies` — that is exactly what you want. Click **Extract**.

macOS. Double-click `supplies.zip`. macOS extracts properly, right next to the zip, and does not do the fake-folder trick.

Either way, when it finishes you have **two** things side by side: the suitcase `supplies.zip`, and the real folder `supplies`. Keep both.

W7. EASY *Before* extracting: double-click `supplies.zip` and look inside the preview window. Count the items. Now close it. Did that create a `supplies` folder in `lab01`? .....
That is the whole trap in one question: looking is not extracting.

W8. EASY Now extract it properly. Your `lab01` folder should now hold four things. List them:
....

W9. MEDIUM Look at the icons of `supplies.zip` and `supplies`. One has a zip on it, one is a plain folder. From now on, that difference is the first thing you check when something "does not work".

TASK 2 Unpack the Suitcase EASY `supplies/` ~10 min

Extract `supplies.zip` so that a real folder named `supplies` sits inside `lab01`, beside the zip. Inside `supplies` you should find ten files and one folder called `archive`. The checker confirms the extraction by looking for `supplies/archive/letter.txt` — a file that is one level deeper than the preview window ever shows you. If that file is missing, you dragged something out of the suitcase instead of unpacking it.
Do not tidy anything yet. The mess is the point of Section 5.

4 Looking Inside, and Going Deeper (~30 min)

Ref: Lecture 1 — reading precisely is half of programming

Ten files, and you have opinions about none of them yet. Time to look.

W10. EASY Open `readme_first.txt` — double-click it, or right-click → Open with → Notepad. Read it. It is talking to you.

W11. EASY Switch your file manager to a view that shows file **sizes**.
Windows: View → **Details**. macOS: View → **as List** (Cmd + 2).
Now click the word **Size** at the top of the column. The list sorts. Click it again — it sorts the other way. One file is dramatically larger than the rest. Which? .....

W12. MEDIUM Count the **files** sitting directly inside `supplies`. Do not count the `archive` folder, and do not count anything inside it. .....
Select everything with Ctrl + A; the status bar at the bottom of the window tells you how many items are selected. Remember to subtract the folder.

W13. MEDIUM Try to open `LICENSE`. Windows will stop and ask you which program to use, because the file has no extension and the machine genuinely does not know. Choose **Notepad** — and when it offers to *always* use Notepad for this kind of file, say **no**. Why do you think "no" is the right answer here? .....

W14. EASY Now go one level deeper: double-click the `archive` folder. Read what is in there. There is something in it you need.

Getting back out

Every file manager has a **back** arrow (Alt + ← on Windows, Cmd + [on macOS) and an **up** arrow that moves you to the containing folder. If you ever feel lost, look at the path bar at the top. It tells you exactly which nested boxes you are standing inside, and clicking any name in it jumps you straight back there.

TASK 3 Fill In the Answer Sheet

EASY `answers.txt` ~10 min

Open `answers.txt` (in `lab01`, not in `supplies`) with Notepad or TextEdit and fill in every line except `LABPATH` — that one waits for Section 6.
You need: your `NAME`, your 9-digit `ROLL` number, the `CODEWORD` from `readme_first.txt`, your `FILECOUNT` and `BIGGEST` from the warm-ups above, and the `ARCHIVEWORD` you found one folder deeper.
Type your answer *after* the `=` sign. Do not delete lines, do not add lines, do not rename the file. Then **save it** (Ctrl + S) — a file you have typed into but not saved is still, as far as the computer is concerned, empty.

TASK 4 Get Them Right

MEDIUM `answers.txt` ~5 min

Task 3 checks that the lines are *filled*. This one checks that they are *correct* — the two codewords, the file count, and the name of the largest file.
The checker stores only a fingerprint of each answer, never the answer itself, so reading `check.py` will tell you nothing. Reading `supplies` will tell you everything.
If `FILECOUNT` is rejected, you almost certainly counted the `archive` folder as a file, or you have extensions hidden and are counting something twice. **If `BIGGEST` is rejected**, write the

file's *full* name, extension included.

5 Filing the Mess (~25 min)

Ref: Lecture 1 — a rule you can state precisely is a rule a machine could follow

Now sort the pile — by extension, which is exactly the rule a program would use.

Goes into	Everything ending in...	How many
`sorted/notes`	`.txt` or `.md`	3
`sorted/data`	`.csv`	2
`sorted/code`	`.py`	2
`sorted/images`	`.png` — in *any* capitalisation	2
`sorted/misc`	nothing at all: no dot, no extension	1

Three plus two plus two plus two plus one is ten, which is every file at the top level of `supplies`. The `archive` folder does **not** move — leave it exactly where it is.

W15. EASY Fill this in *before* you move anything. Write the destination folder next to each file:

File	Goes to	File	Goes to
`grades.csv`	.....	`hello.py`	.....
`syllabus.md`	.....	`chart.PNG`	.....
`LICENSE`	.....	`crest.png`	.....
`readme_first.txt`	.....	`average.py`	.....
`week 01 notes.txt`	.....	`attendance.csv`	.....

W16. MEDIUM `chart.PNG` is shouting and `crest.png` is not. Are they the same kind of file? .....

Windows treats `.PNG` and `.png` as identical; Linux — where a great deal of the world's code eventually runs — does not. Habit worth forming now: **lower-case everything you name yourself.**

W17. EASY How would you move a file? Two ways: drag it with the mouse, or select it, Ctrl + X, open the destination, Ctrl + V. The second is far more reliable, because dragging into the wrong folder is silent and undoing it is not.

Moved something into the wrong place?

Ctrl + Z (Cmd + Z on macOS) undoes a move, a copy or a rename in File Explorer and Finder, just like it does in a document. Most people never discover this. You now have.

TASK 5 Five Empty Drawers EASY `sorted/` ~5 min

Inside `lab01`, make a folder called `sorted`. Inside `sorted`, make five folders: `notes`, `data`, `code`, `images`, `misc`.
All lower-case, spelled exactly as printed. Build the empty shelves before you start filing — carrying files around while inventing folder names is how things end up in three places at once.

TASK 6 **File the Notes** EASY `sorted/notes` ~5 min

Move the three text files — the two `.txt` files and the one `.md` file — from `supplies` into `sorted/notes`.
One of them has **spaces in its name**. It will behave perfectly well here, and cause you a small amount of trouble in Section 6. Notice which one it is.

TASK 7 **File the Data and the Code** MEDIUM `sorted/data + code` ~5 min

Move both `.csv` files into `sorted/data`, and both `.py` files into `sorted/code`.
Do not open the `.csv` files by double-clicking if Excel is installed — it will offer to "fix" them and change what is inside. If you want to look, right-click → Open with → Notepad. A `.csv` is just text with commas.

TASK 8 **File the Images and the Oddment** MEDIUM `sorted/images + misc` ~5 min

Move both image files into `sorted/images`, and the one file with no extension at all into `sorted/misc`.
When this passes, `supplies` contains nothing but the `archive` folder — and your `sorted` tree looks exactly like the diagram on the next page. Run `check.py` and you should be on 8 of 9.

```
lab01\                      <- you are here
   check.py
   answers.txt
   supplies.zip             <- the suitcase, kept
   supplies\
      archive\              <- untouched
         letter.txt
         old_syllabus.md
   sorted\
      notes\    3 files
      data\     2 files
      code\     2 files
      images\   2 files
      misc\     1 file
```

6 The Command Prompt: The Same House, Another Door (~45 min)

Ref: Lecture 1 — a programming language is an exact way of telling a machine what to do. So is this.

Everything you just did with a mouse, you can also do by typing. The window you type into is called the **command prompt** (Windows) or the **terminal** (macOS and Linux).

It has a reputation. Ignore the reputation. Here is the honest version:

What the black window actually is

It is a file manager with no pictures.
You are always *standing in* exactly one folder — the same as having one File Explorer window open. You type a short instruction, press Enter, and the machine does that one thing and waits

for the next. That is all that ever happens.
It cannot delete anything unless you type a delete command, and this lab never asks you to. If you type nonsense, it says so and waits. If you close the window, nothing is lost. There is no way to "break" it by exploring.
Why bother, when the mouse works? Three reasons, and you will meet all three this semester: it is the only way to run some tools; it is precise enough to describe in a message ("type this" beats "click the thing near the top"); and it does not care how many files you have — one or ten thousand, same command.

Opening it, standing in the right place

The shortcut that skips all the navigation

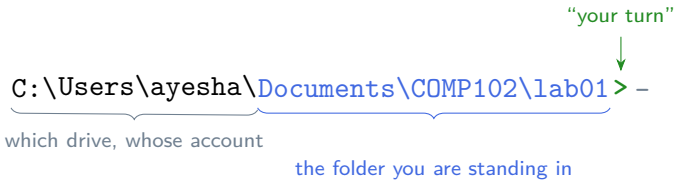
Windows. Open your `lab01` folder in File Explorer. Click the empty part of the address bar at the top, type `cmd`, and press `Enter`. A black window opens **already standing in that folder**. This trick will save you hours.

macOS. Right-click the `lab01` folder → Services → **New Terminal at Folder**. (If it is not there: System Settings → Keyboard → Keyboard Shortcuts → Services → tick it.)

The long way round works too: `Win` then type `cmd`, or `Cmd` + `Space` then type `Terminal`. You will then have to walk to your folder yourself — which is the next page.

Reading the prompt

The line waiting for you is not decoration. It is telling you where you are standing.



On macOS the same line looks like `ayesha@MacBook lab01 %` — different punctuation, identical meaning: *here is where you are, go ahead.*

The four commands that are the whole job

You want to...	Windows	macOS / Linux	In words
know where you are	`cd`	`pwd`	prints the full path you are standing in
see what is here	`dir`	`ls`	lists the files and folders
go into a folder	`cd sorted`	`cd sorted`	step through that door
come back out	`cd ..`	`cd ..`	`..` always means "the folder above"

Four commands. That is genuinely the entire toolkit for today.

```
C:\Users\ayesha\Documents\COMP102\lab01>dir

 Directory of C:\Users\ayesha\Documents\COMP102\lab01

11/09/2026  10:42    <DIR>          .
```

```
11/09/2026  10:42    <DIR>          ..
11/09/2026  10:41             8,214 answers.txt
11/09/2026  10:40             9,102 check.py
11/09/2026  10:42    <DIR>          sorted
11/09/2026  10:41    <DIR>          supplies
11/09/2026  10:38             7,007 supplies.zip
               3 File(s)         24,323 bytes
               4 Dir(s)  241.7 GB free

C:\Users\ayesha\Documents\COMP102\lab01>cd sorted

C:\Users\ayesha\Documents\COMP102\lab01\sorted>dir /b
code
data
images
misc
notes

C:\Users\ayesha\Documents\COMP102\lab01\sorted>cd ..

C:\Users\ayesha\Documents\COMP102\lab01>
```

Read that transcript slowly. Notice that the prompt *changed* when you moved, and changed back. The prompt is a live answer to "where am I?". Notice `<DIR>`: that is how `dir` marks a folder. And `dir /b` is the same command asking for the bare list, without the dates and sizes.

W18. EASY Open a command prompt standing in `lab01` (use the `cmd`-in-the-address-bar trick). Type `dir` on Windows or `ls` on macOS, and press Enter. You should see the same four things File Explorer shows. Same house, other door.

W19. EASY Predict, then run: what do you expect `cd supplies` followed by `dir` to print?

.....

Then run it. Then `cd ..` to come back.

W20. MEDIUM Navigate all the way into `sorted/notes` using two commands, then all the way back out using two more. Write the four commands you used:

....

....

W21. MEDIUM **Tab completion — learn this one properly.** Type `cd sup` and then press Tab. The machine finishes the word for you. It is faster than typing, and it cannot misspell. Professionals type perhaps half the characters you do.

W22. HARD Now try `cd sorted`, then `cd notes`, then type `type week` and press Tab (macOS: `cat week` then Tab). Look at what Tab produced — it wrapped the name in quotation marks by itself. Press Enter and the file's contents print.
Why did it add quotes? .....

Spaces need quotes

A space is how the command prompt separates one word from the next. So this fails:

```
C:\...\notes>type week 01 notes.txt
The system cannot find the file specified.
```

The machine read that as *three* separate things. Wrap the name in double quotes and it becomes one thing again:

```
C:\...\notes>type "week 01 notes.txt"
Lecture 01 -- rough notes
...
```

This is the same lesson as Lecture 1, in a different costume: the machine splits your sentence by a rule, not by your intention. Say precisely what you mean — or let **Tab** say it for you.

When it goes wrong — and it will

Red text is not a scolding. It is the machine telling you, quite precisely, what it could not do. There are only a handful of messages, and here they all are:

It says	It means
`The system cannot find the path specified.`	There is no folder by that name *here*. Type `dir` and read what is actually around you — usually you are one folder up or down from where you thought.
`'python' is not recognized...`	The command was fine; Python is not installed, or not installed in a way this window can see. Not your fault, and fixable — ask the lab engineer.
`The system cannot find the file specified.`	The file name is wrong, or it has spaces and no quotes. **Tab** solves both.
`Access is denied.`	You are trying to touch something that is not yours. Nothing today should ever produce this; if it does, you are outside your own folders.

Three keys that make this pleasant

↑ brings back the last command you typed — press it repeatedly to walk back through your history. No retyping, ever.
Tab completes a name you have started.
`cls` (Windows) or `clear` (macOS) wipes the screen when it gets cluttered. It deletes nothing but the mess.

7 Capstone: Ask the Machine Where You Are (~15 min)

One task left, and it is the one that ties the mouse and the keyboard together.

TASK 9 **Your Own Address** HARD `answers.txt` ~15 min

Open a command prompt **standing in your `lab01` folder**. Then:

1. Type `cd` (Windows) or `pwd` (macOS) and press **Enter**. On its own, with nothing after it, `cd` does not move you anywhere — it simply prints the full path of where you are.
2. Copy that line into the `LABPATH` line of `answers.txt`, exactly as printed, backslashes and all.

   To copy out of a Windows command prompt: drag across the text with the mouse, then press **Enter**. That is genuinely the shortcut, and yes, it is strange.
3. Save `answers.txt`.
4. Back at the prompt, run the checker:

```
C:\Users\ayesha\Documents\COMP102\lab01>python check.py
```

The prediction that matters: before you press **Enter**, write down how many of the nine tasks you expect to pass. .....
Then find out. If your guess was wrong, the report will tell you exactly where — that is what it is for.

If the checker says ***`LABPATH is not this folder`****, you are standing in a different folder from the one* ***`check.py`*** *lives in. Type* ***`dir`*** *— if you cannot see* ***`check.py`*** *in the list, you are in the wrong room.*

If Windows says ***`'python' is not recognized`****, try* ***`py check.py`*** *instead. If that also fails, open* ***`check.py`*** *in Thonny and press* ***Run*** *— it works exactly the same from there — and mention it to the lab engineer before you leave, because you will need Python on the command line later in the semester.*

Before you pack up

Run `check.py` one last time and show the report to your instructor. Then — and this is not a joke — make sure you can find your `COMP102` folder again next week. Everything you write for the rest of this course goes inside it.

Self-Assessment Checklist

Before you leave, honestly tick what you can now do:

- ☐ I can read a path and say which folder is inside which.
- ☐ I can state the full path of my `lab01` folder from memory.
- ☐ I can make, rename and move a folder without hesitating.
- ☐ I have turned on file name extensions, and I know why that matters for my safety as well as my marks.
- ☐ I can name the extension of a file and say what opens it.
- ☐ I know the difference between looking inside a zip and extracting it — and I know which one loses your work.
- ☐ I can sort files by size and by name in my file manager.
- ☐ I can navigate into a folder that is inside another folder, and get back out.
- ☐ I know **Ctrl** + **Z** undoes a move in File Explorer.
- ☐ I can open a command prompt already standing in a chosen folder.
- ☐ I can read a prompt line and say where I am standing.
- ☐ I can use `cd`, `dir`/`ls` and `cd ..` to move around and look.
- ☐ I use **Tab** to complete names, and ↑ to recall commands.
- ☐ I know why a file name with spaces needs quotation marks.
- ☐ I can read an error message without panicking, and act on it.
- ☐ I can run a Python file from the command prompt.

Part B — Going Deeper (~45 min)

Optional. Unmarked. Only for after `check.py` *says 9 of 9.*

If you arrived already knowing where your files live, the nine tasks above were housekeeping. This part is the other thing the command prompt is for: *making* things, not just looking at them. Two of these show up as bonus rows in `check.py`. The rest are for you.

B1 Predict, then run

Stand in your `lab01` folder. Write your prediction for every line **before** you touch the keyboard. Then run them in order.

	Command (Windows)	Your prediction	It actually...
1	`cd sorted\notes`	.....	.....
2	`cd ..\..`	.....	.....
3	`dir sorted`	.....	.....
4	`dir *.zip`	.....	.....
5	`dir sorted\*\*.csv`	.....	.....
6	`dir /s /b *.png`	.....	.....
7	`cd \`	.....	.....
8	`tree`	.....	.....

macOS equivalents, in the same order: `cd sorted/notes` `cd ../..` `ls sorted` `ls *.zip` `ls sorted/*/*.csv` `find . -name "*.png"` `cd /` `find .`

Read only after you have run all eight

- **1, 2** — a relative path can name a chain of folders in one go, in either direction. `..\..` climbs two levels.
- **3** — you can *look* into a folder without *going* into it. Most commands take a path as an argument.
- **4, 5, 6** — `*` means "any run of characters". `*.zip` is "anything ending in .zip". This is a **wildcard**, and it is the moment the command prompt starts beating the mouse: one line just searched folders you never opened.
- **7** — `cd \` jumps straight to the top of the drive. Type `dir` there and look at how little is actually at the root of your computer.
- **8** — `tree` draws the whole nested structure at once. Run it in `lab01` and compare it with the diagram on page 6.

B2 Two tasks for `check.py`'s bonus rows

TASK 10 **Send a Listing Into a File** HARD `proof.txt` ~15 min

Anything a command prints to the screen can be written into a file instead, by putting `>` and a file name after it. The technical name is **redirection**; the everyday name is "save that output". Standing in `lab01`, produce a file called `proof.txt` that contains a listing of your `supplies` folder or your `sorted/data` folder — whichever you like, as long as `grades.csv` is mentioned

somewhere in it.
Predict first: after you run the command, will anything appear on the screen?
. . . . . .
Then run it, and open `proof.txt` to see where the output went instead. On Windows start from `dir`; on macOS from `ls`. Careful: `>` overwrites the file every time, without asking. `>>` adds to the end instead.

TASK 11 **Three Folders Deep, No Mouse** HARD sandbox/ ~10 min

The command that makes a folder is `mkdir` — "make directory" — on every operating system there is.
Standing in `lab01`, and **without touching the mouse**, create this:

```
sandbox\
   alpha\
      beta\
         gamma\
```

Do it the slow way first: `mkdir sandbox`, `cd sandbox`, `mkdir alpha`, and so on — eight commands, and you will feel every one. Then delete `sandbox` with the mouse and do the whole thing again in **one** command.
Write the one-command version here before you look it up:
. . . . . .
Hint: `mkdir` accepts a path, not just a name. What did you learn from B1 line 1?

B3 Questions with no checker

B1. Run `dir` in `lab01` and look at the first two lines of the listing: `.` and `..`. One means "this folder", the other means "the folder above". Which is which, and why does `cd ..` therefore do what it does?

. . . . . .

B2. In File Explorer, tick **Hidden items** under the View menu (macOS: Cmd + Shift + . in Finder). Things appear that were always there. Why do you think an operating system hides files from its own owner — and is that a kindness or a problem?

. . . . . .

. . . . . .

B3. You typed `python check.py` and it worked. But there is no file called `python` in your `lab01` folder — so how did the machine know what `python` meant? Search for the phrase "the PATH environment variable" and write a one-sentence answer.

. . . . . .

B4. Make a copy of `sorted/notes/syllabus.md` called `syllabus.md.backup`, then rename it back. Which commands did you use, and what does your computer think the second file's type is?

. . . . . .

B5. Zip your finished `sorted` folder into `sorted.zip` using the mouse (Windows: right-click → Send to → Compressed folder; macOS: right-click → Compress). Compare the size of the zip

with the total size of the folder. Which files shrank a lot, and which barely shrank at all? What does that suggest about what was already compressed?

. . . . . .

. . . . . .

Where this goes next

Next week you open Thonny, type your first line of Python, and save it into `COMP102\lab02`. When the file is where you meant it to be, and you can find it again, and you can run it from either door — none of that will cost you a single minute of thought.
That is the entire purpose of today. Programming is hard enough on its own merits; it should never also be a scavenger hunt.