Week 2:
Introduction to Python

Dr Christian Engels
ce50@st-andrews.ac.uk

FI5699 Dissertation Module

Department of Finance
University of St Andrews Business School

Introduction

What We Will Cover Today

This is a hands-on session. Have your laptop open and follow along.

1. **Positron** — getting a Python session running
2. **Console & Files** — two ways to write Python
3. **uv** — installing packages into your project
4. **Data Structures** — organising financial data
5. **Loops** — automating repetitive calculations

No programming experience required! We will go step by step.

Table of Contents

Why Python?

- Used virtually everywhere

Why Python?

- Used virtually everywhere
- Free and open source (unlike Stata, EViews or Bloomberg Terminal)

Why Python?

- Used virtually everywhere
- Free and open source (unlike Stata, EViews or Bloomberg Terminal)
- Huge ecosystem of tools for data, statistics, and visualisation

Why Python?

- Used virtually everywhere
- Free and open source (unlike Stata, EViews or Bloomberg Terminal)
- Huge ecosystem of tools for data, statistics, and visualisation
- The most-requested programming skill in finance job postings

Why Python?

- Used virtually everywhere
- Free and open source (unlike Stata, EViews or Bloomberg Terminal)
- Huge ecosystem of tools for data, statistics, and visualisation
- The most-requested programming skill in finance job postings
- You do **not** need to be a “programmer” — think of it as a very powerful calculator

Why Python?

- Used virtually everywhere
- Free and open source (unlike Stata, EViews or Bloomberg Terminal)
- Huge ecosystem of tools for data, statistics, and visualisation
- The most-requested programming skill in finance job postings
- You do **not** need to be a “programmer” — think of it as a very powerful calculator
- With AI tools like GitHub Copilot, writing code is easier than ever

Recap: What You Already Set Up

In the Python Setup Guide you installed:

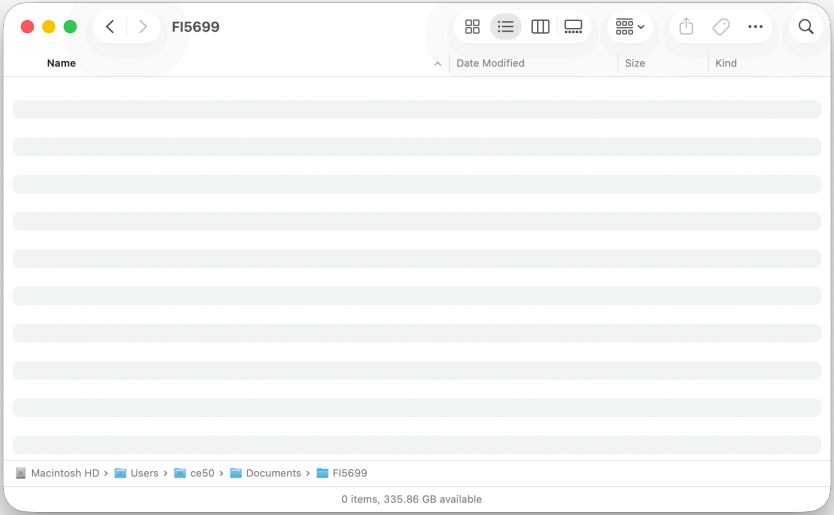
Tool	What it does
uv	Manages Python and packages
Positron	Your code editor (like Word, but for code)
GitHub Copilot	AI assistant that helps you write code

Today we will learn **what** these tools actually do and **how** to use them for finance.

Getting Started with Positron

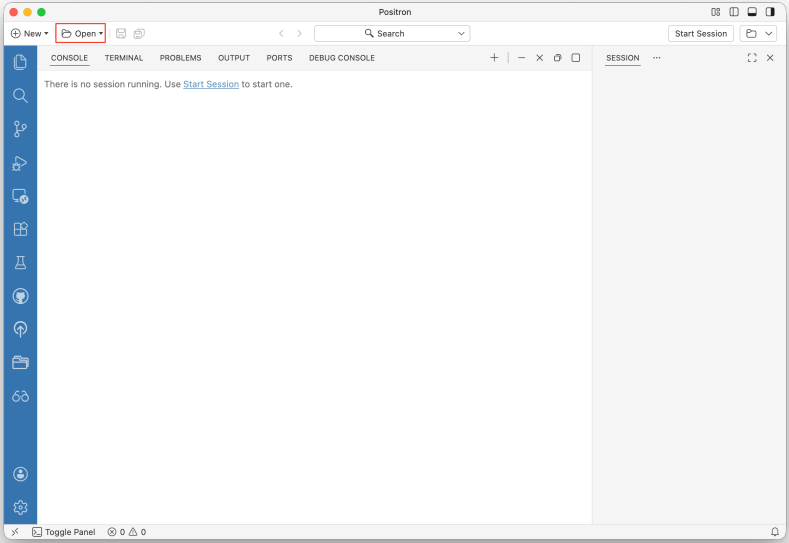
Getting Started in Positron

Step 1: Create an empty folder for your course (e.g. FI5699 in Documents)



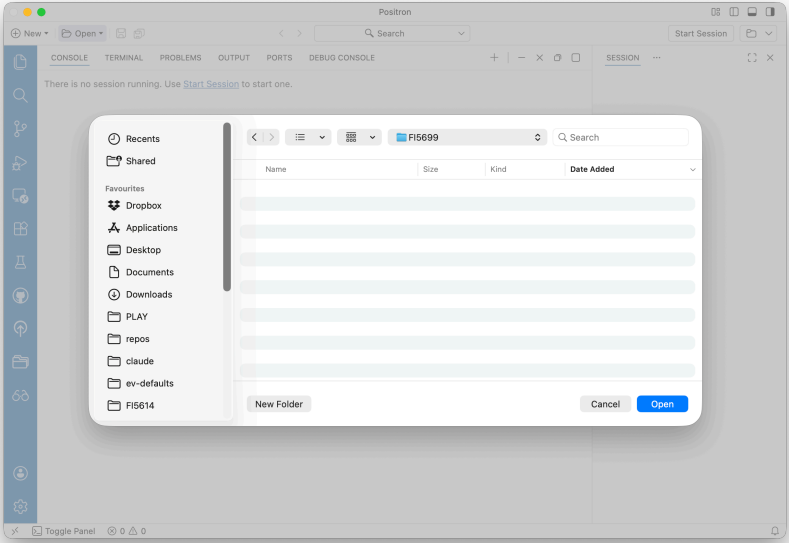
Getting Started in Positron

Step 2: Open Positron and click the Open button



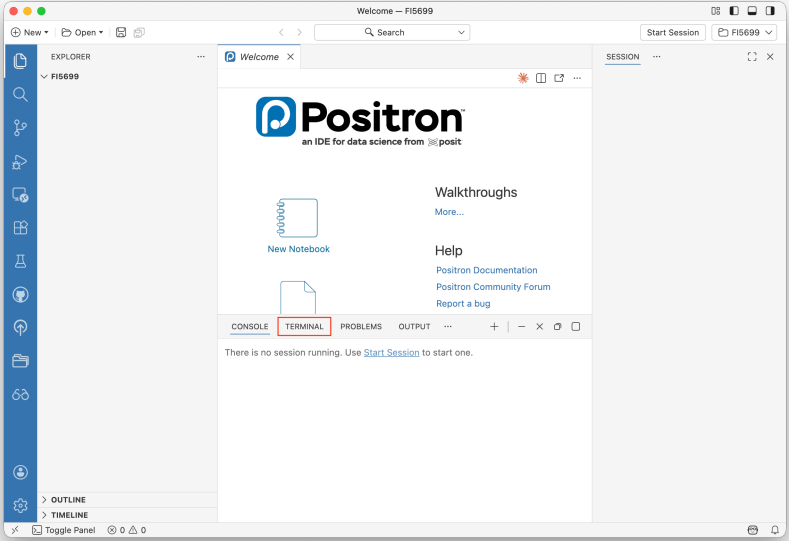
Getting Started in Positron

Step 3: Navigate to your folder and click Open



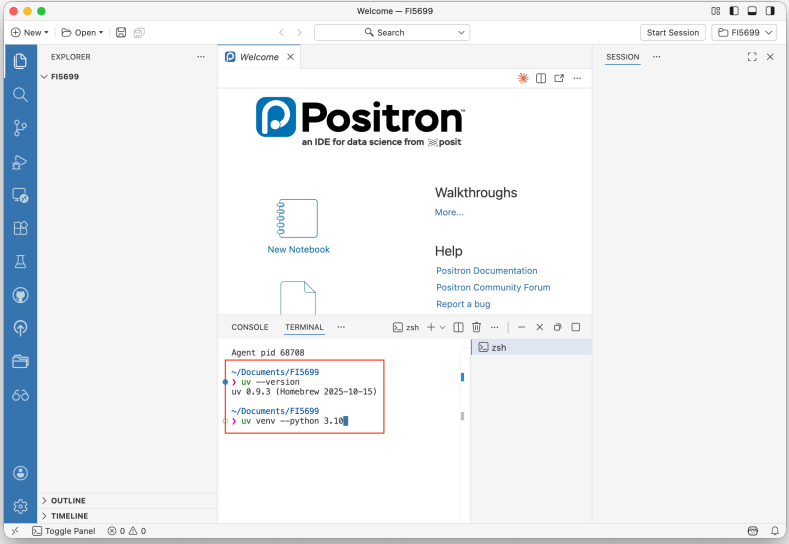
Getting Started in Positron

Step 4: Your folder is now open — you'll see the Welcome tab and sidebar



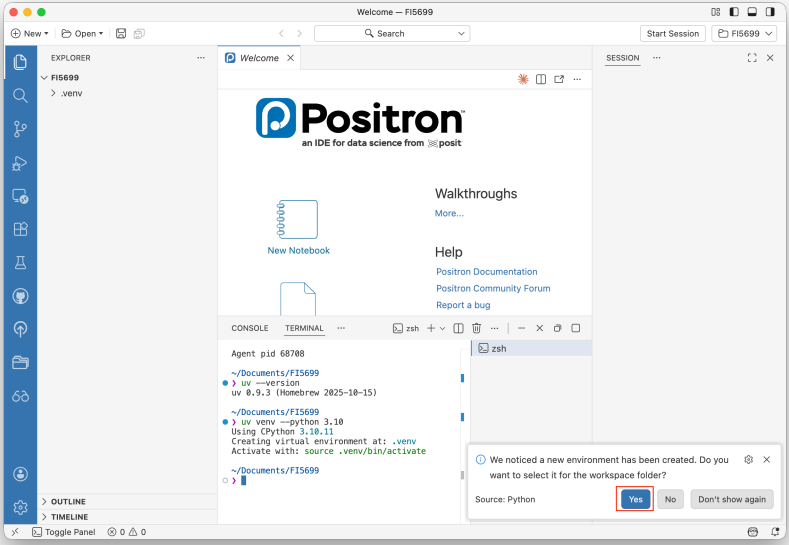
Getting Started in Positron

Step 5: In the terminal, type `uv venv --python 3.10` and press Enter



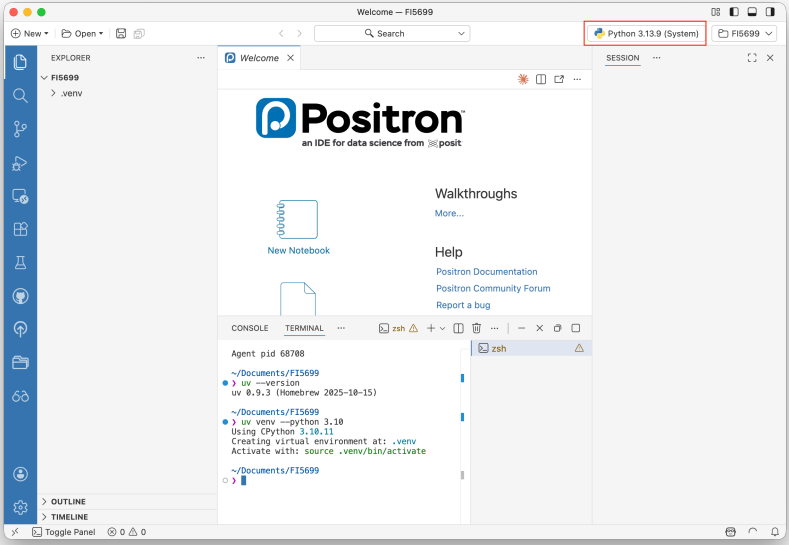
Getting Started in Positron

Step 6: Positron detects the new environment — click Yes



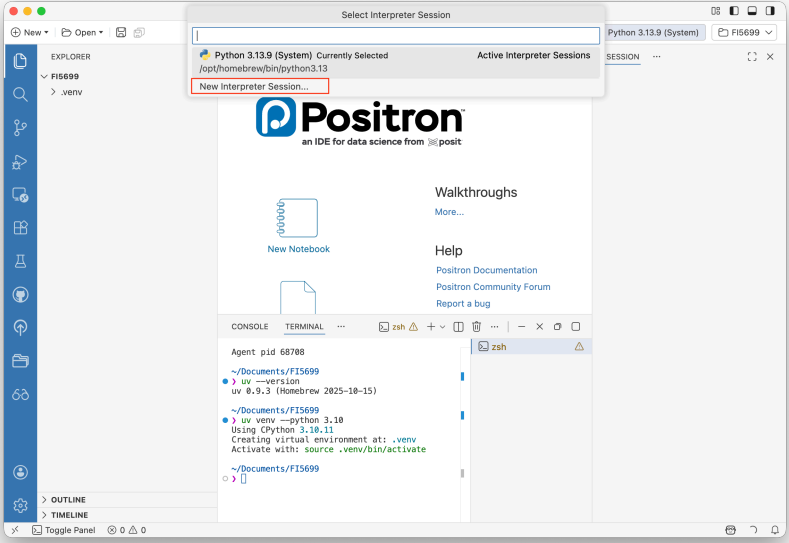
Getting Started in Positron

Step 7: A Python interpreter appears in the top-right toolbar — click on it



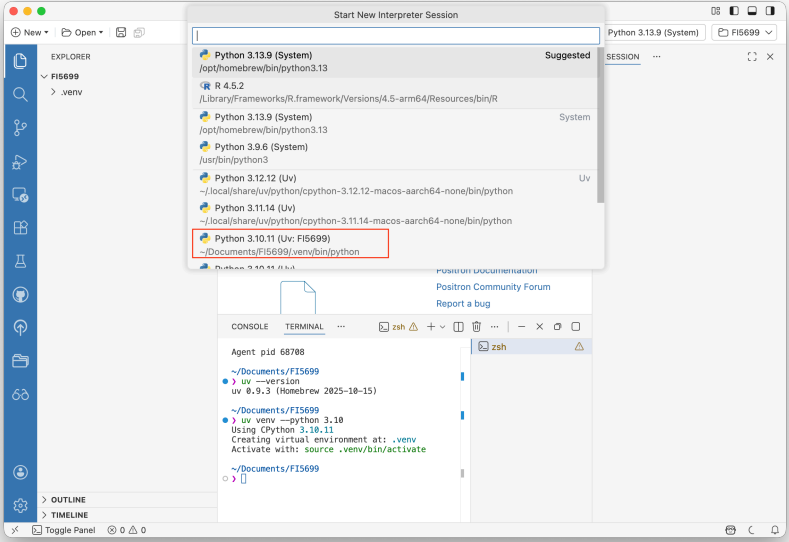
Getting Started in Positron

Step 8: Choose New Interpreter Session



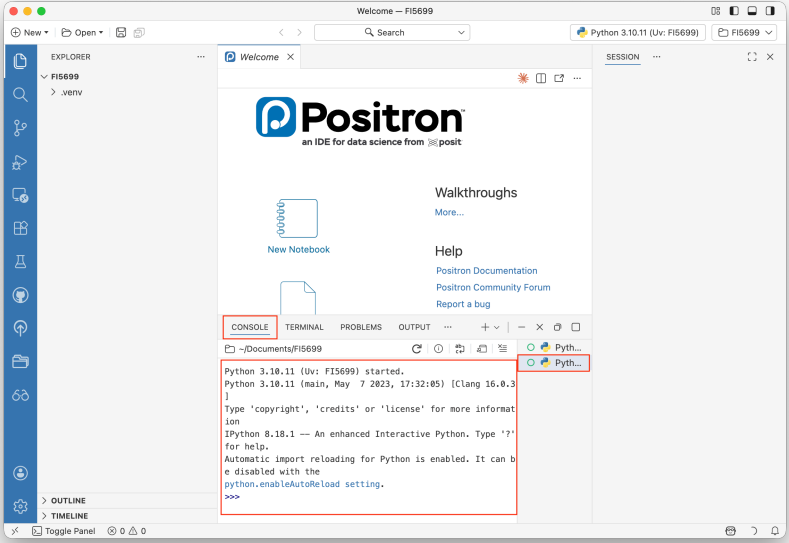
Getting Started in Positron

Step 9: Select the one for your project — e.g. Python 3.10.11 (Uv: FI5699)



Getting Started in Positron

Step 10: You should see the Python Console with the >>> prompt — you're ready!

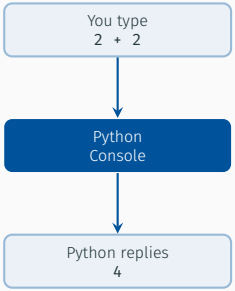


The Python Console

The Python Console

You just opened the Python Console in Positron (Step 10). Let's understand what it is.

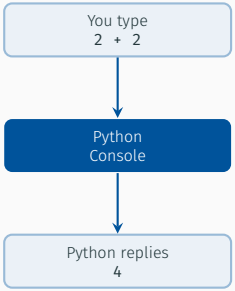
- The Console is where you type Python and get instant results



The Python Console

You just opened the Python Console in Positron (Step 10). Let's understand what it is.

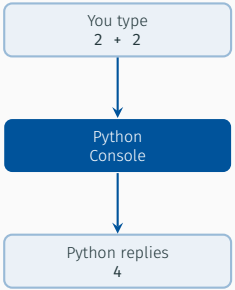
- The Console is where you type Python and get instant results
- The >>> prompt means Python is waiting for your input



The Python Console

You just opened the Python Console in Positron (Step 10). Let's understand what it is.

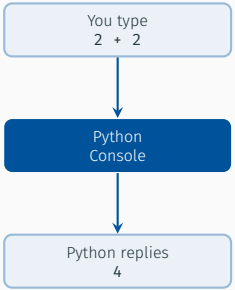
- The Console is where you type Python and get instant results
- The >>> prompt means Python is waiting for your input
- Type something, press Enter, see the answer



The Python Console

You just opened the Python Console in Positron (Step 10). Let's understand what it is.

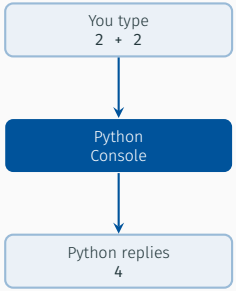
- The Console is where you type Python and get instant results
- The >>> prompt means Python is waiting for your input
- Type something, press Enter, see the answer
- Think of it as a very powerful calculator



The Python Console

You just opened the Python Console in Positron (Step 10). Let's understand what it is.

- The Console is where you type Python and get instant results
- The >>> prompt means Python is waiting for your input
- Type something, press Enter, see the answer
- Think of it as a very powerful calculator
- Positron also has a Terminal tab — that's for system commands (installing packages, etc.)



Console vs Terminal

Positron has **two** tabs at the bottom. Make sure you're in the right one:

	Console	Terminal
Prompt	`>>>`	`$` (Mac/Linux) or `PS>` (Windows)
Language	Python	Shell (system commands)
Use for	Running Python code	Installing packages, file management
Example	`2 + 2` → `4`	`uv add polars`

For this session, we will mostly use the **Console** tab.

Python as a Calculator

In the Console, try typing these expressions:

```
>>> 2 + 2
4
>>> 100 * 1.05    # 5% return on 100
105.0
>>> 175000 / 12   # monthly salary
14583.333333333334
```

Operator	Meaning	Example
`+`	Add	`100 + 50` → `150`
`-`	Subtract	`100 - 30` → `70`
`*`	Multiply	`50 * 3` → `150`
`/`	Divide	`100 / 3` → `33.33…`
`**`	Power	`1.07 ** 10` → `1.967…`

Storing Values with Variables

Instead of retyping numbers, give them names:

```
>>> price = 142.50
>>> shares = 100
>>> position = price * shares
>>> position
14250.0
```

- A variable is a name that stores a value

Storing Values with Variables

Instead of retyping numbers, give them names:

```
>>> price = 142.50
>>> shares = 100
>>> position = price * shares
>>> position
14250.0
```

- A variable is a name that stores a value
- Use = to assign: the name goes on the left, the value on the right

Storing Values with Variables

Instead of retyping numbers, give them names:

```
>>> price = 142.50
>>> shares = 100
>>> position = price * shares
>>> position
14250.0
```

- A variable is a name that stores a value
- Use = to assign: the name goes on the left, the value on the right
- Python remembers it for the rest of your session

Storing Values with Variables

Instead of retyping numbers, give them names:

```
>>> price = 142.50
>>> shares = 100
>>> position = price * shares
>>> position
14250.0
```

- A variable is a name that stores a value
- Use = to assign: the name goes on the left, the value on the right
- Python remembers it for the rest of your session
- You can use variables in calculations, just like in a spreadsheet formula

Displaying Results with `print()`

The console shows the result of the last expression automatically. But to display something explicitly, use `print()`:

```
>>> ticker = "AAPL"
>>> price = 142.50
>>> shares = 100
>>> print(ticker)
AAPL
>>> print(f"Position value: ${price * shares:,.2f}")
Position value: $14,250.00
```

- `print()` displays whatever you put inside the brackets

Displaying Results with `print()`

The console shows the result of the last expression automatically. But to display something explicitly, use `print()`:

```
>>> ticker = "AAPL"
>>> price = 142.50
>>> shares = 100
>>> print(ticker)
AAPL
>>> print(f"Position value: ${price * shares:,.2f}")
Position value: $14,250.00
```

- `print()` displays whatever you put inside the brackets
- An f-string (note the `f` before the quotes) lets you embed variables inside text

Displaying Results with `print()`

The console shows the result of the last expression automatically. But to display something explicitly, use `print()`:

```
>>> ticker = "AAPL"
>>> price = 142.50
>>> shares = 100
>>> print(ticker)
AAPL
>>> print(f"Position value: ${price * shares:,.2f}")
Position value: $14,250.00
```

- `print()` displays whatever you put inside the brackets
- An f-string (note the `f` before the quotes) lets you embed variables inside text
- `:,.2f` is optional formatting: commas and 2 decimal places

Helpful Console Tips

Shortcut	What it does
Up arrow (↑)	Recall previous commands — saves retyping
Tab	Autocomplete a variable or function name
Ctrl + C	Cancel a running command if something goes wrong
`type(x)`	Find out what kind of value `x` is (number, text, etc.)
`help(x)`	Get documentation for any function

You don't need to memorise these. Just remember: **up arrow** to get back what you just typed.

Exercise 1: Explore the Console

Try It Yourself (3 minutes)

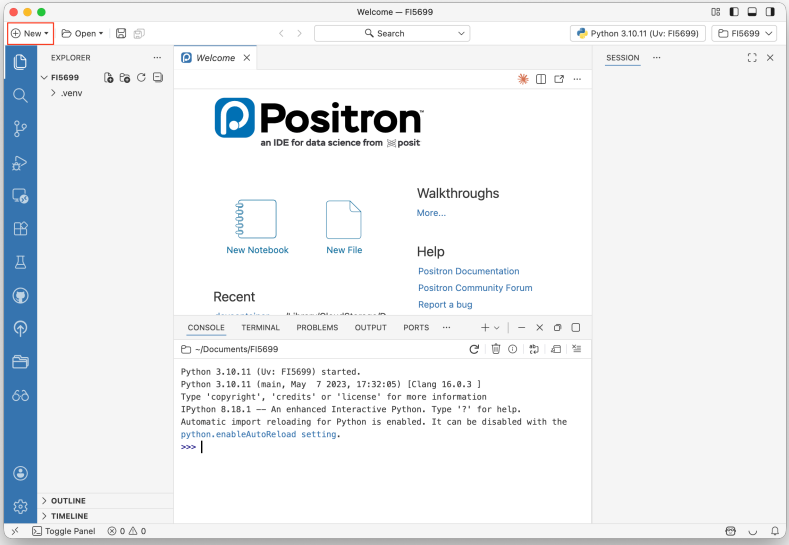
1. In the Positron **Console**, compute `1.07 ** 10` — what is 7% compounded over 10 years?
2. Create a variable `salary = 175000` and compute the monthly amount (`salary / 12`)
3. Try `print(f"Monthly: $salary / 12:,.2f")`
4. Use the **up arrow** to recall and re-run a previous command

Stuck? Raise your hand. There are no silly questions here.

Python Files (`.py`)

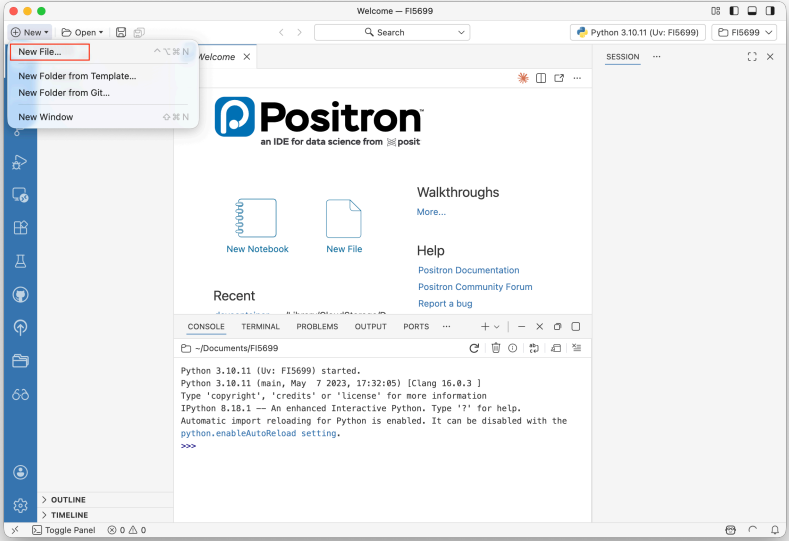
Python Files (`.py`)

So far we've typed in the Console. Let's save code in a file instead.



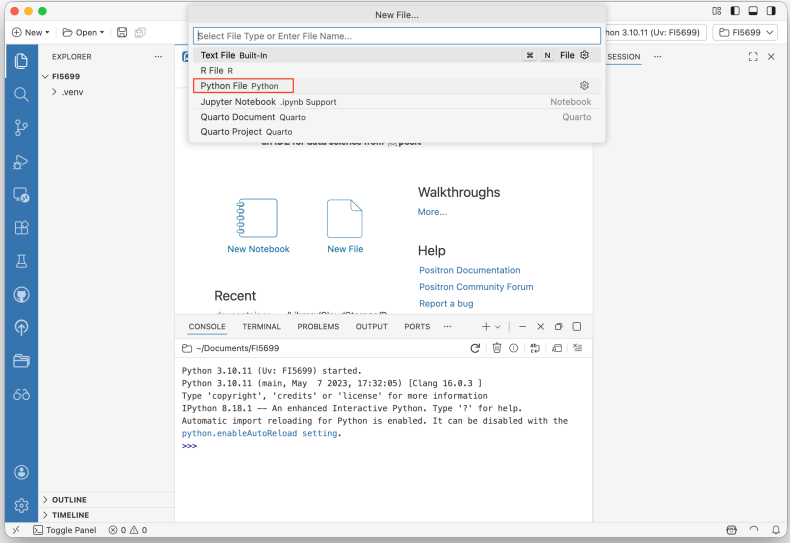
Python Files (`.py`)

Step 1: Click the New button (top-left) and choose New File...



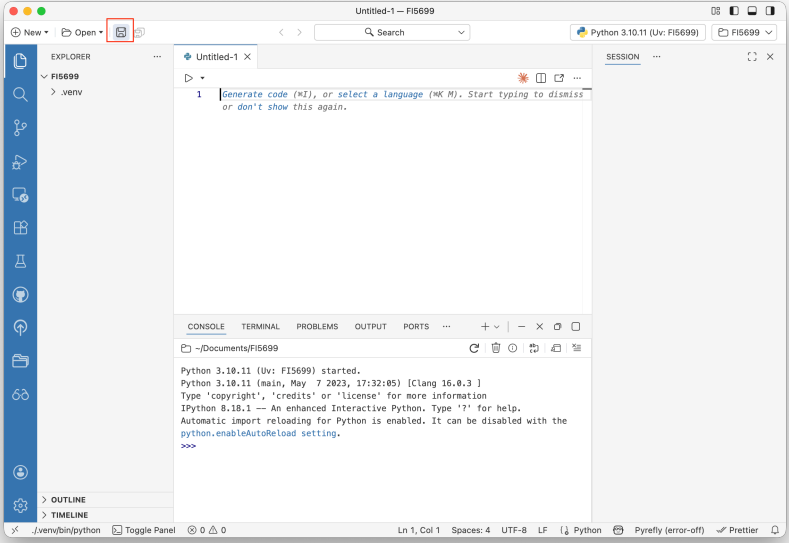
Python Files (`.py`)

Step 2: Select Python File from the list



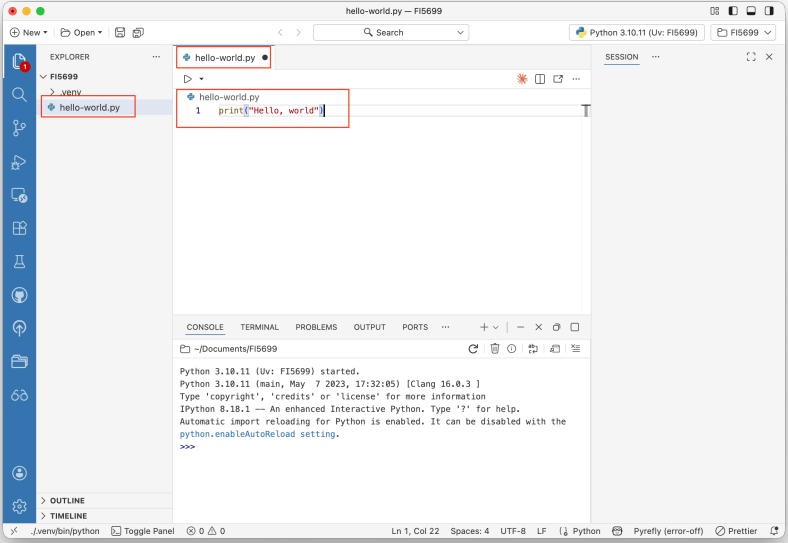
Python Files (`.py`)

Step 3: A new empty file opens in the editor — this is where you write your code



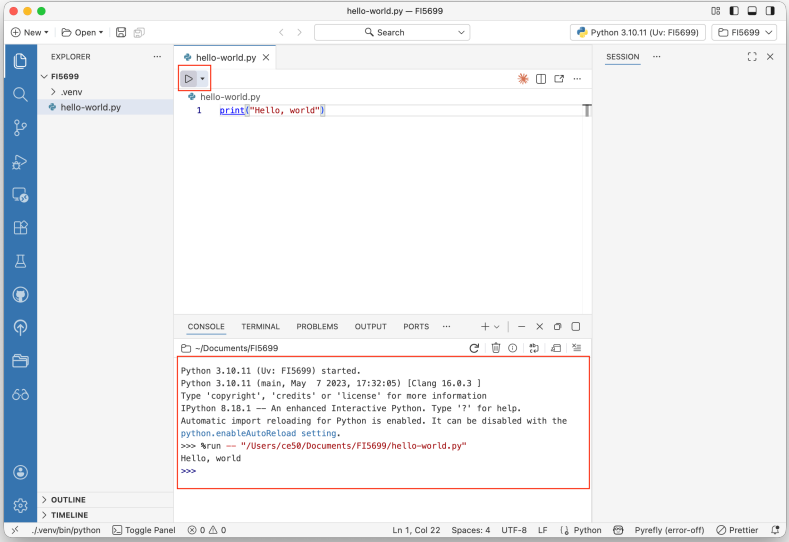
Python Files (`.py`)

Step 4: Type your code and save with Cmd+S / Ctrl+S — give it a `.py` name



Python Files (`.py`)

Step 5: Click the play button (▶) to run — output appears in the Console below



Running Code: Whole File vs Line by Line

Run the whole file

- Click the ▶ button at the top of the editor

Run a single line

Important: When running line by line, Python only knows about lines you've already run. Always run lines **in order from the top** — e.g. run `x = 5` before `print(x)`.

Running Code: Whole File vs Line by Line

Run the whole file

- Click the ▶ button at the top of the editor
- Runs every line from top to bottom

Run a single line

Important: When running line by line, Python only knows about lines you've already run. Always run lines **in order from the top** — e.g. run `x = 5` before `print(x)`.

Running Code: Whole File vs Line by Line

Run the whole file

- Click the ▶ button at the top of the editor
- Runs every line from top to bottom
- Best for finished scripts

Run a single line

Important: When running line by line, Python only knows about lines you've already run. Always run lines **in order from the top** — e.g. run `x = 5` before `print(x)`.

Running Code: Whole File vs Line by Line

Run the whole file

- Click the ▶ button at the top of the editor
- Runs every line from top to bottom
- Best for finished scripts

Run a single line

- Place your cursor on a line

Important: When running line by line, Python only knows about lines you've already run. Always run lines **in order from the top** — e.g. run `x = 5` before `print(x)`.

Running Code: Whole File vs Line by Line

Run the whole file

- Click the ▶ button at the top of the editor
- Runs every line from top to bottom
- Best for finished scripts

Run a single line

- Place your cursor on a line
- Press Cmd+Enter (Mac) or Ctrl+Enter (Windows)

Important: When running line by line, Python only knows about lines you've already run. Always run lines **in order from the top** — e.g. run `x = 5` before `print(x)`.

Running Code: Whole File vs Line by Line

Run the whole file

- Click the ▶ button at the top of the editor
- Runs every line from top to bottom
- Best for finished scripts

Run a single line

- Place your cursor on a line
- Press Cmd+Enter (Mac) or Ctrl+Enter (Windows)
- That line runs in the Console

Important: When running line by line, Python only knows about lines you've already run. Always run lines **in order from the top** — e.g. run `x = 5` before `print(x)`.

Installing Packages with uv

Setting Up Your Project

You already have a folder open in Positron with a virtual environment. Now let's turn it into a proper project so we can install packages.

In the **Terminal** tab (not the Console), type:

```
$ uv init
```

This creates two files in your folder:

File	What it does
`pyproject.toml`	The recipe — lists which packages your project needs
`uv.lock`	The exact versions — so your code works the same way every time

Think of `pyproject.toml` like a shopping list. You're about to add items to it.

Your Dissertation Toolkit

These are the packages we'll use throughout the module:

Package	What it does	Think of it as…
`polars`	Work with tables of data	A supercharged spreadsheet
`plotnine`	Create charts and plots	Excel charts, but better
`pyfixest`	Run regressions	Stata's `reghdfe`, in Python
`diff-diff`	Difference-in-differences	Causal inference toolkit

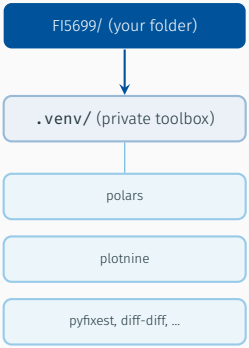
Install them all in one go (in the **Terminal** tab):

```
$ uv add polars plotnine pyfixest diff-diff
```

What Is a Virtual Environment?

When you ran `uv venv` earlier, it created a `.venv` folder inside your project.

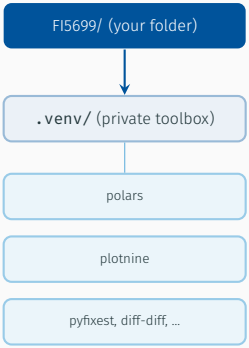
- This is your project's private toolbox



What Is a Virtual Environment?

When you ran `uv venv` earlier, it created a `.venv` folder inside your project.

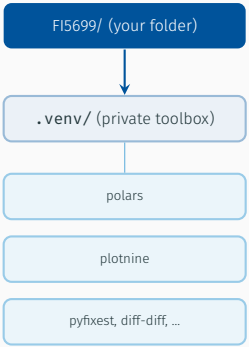
- This is your project's private toolbox
- Packages you install go here, not on your whole computer



What Is a Virtual Environment?

When you ran `uv venv` earlier, it created a `.venv` folder inside your project.

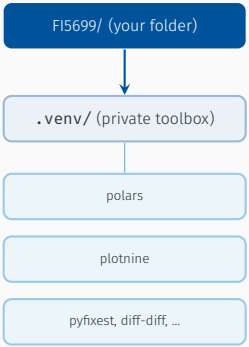
- This is your project's private toolbox
- Packages you install go here, not on your whole computer
- Different projects can use different packages (or different versions) without clashing



What Is a Virtual Environment?

When you ran `uv venv` earlier, it created a `.venv` folder inside your project.

- This is your project's private toolbox
- Packages you install go here, not on your whole computer
- Different projects can use different packages (or different versions) without clashing
- Positron already knows about it (you selected it in step 9)



What Happens If You Forget `uv add`?

If you try to use a package you haven't installed:

```
>>> import polars as pl
```

```
ModuleNotFoundError: No module named 'polars'
```

This is Python telling you: *"I don't have that tool in my toolbox."*

The fix is always the same — go to the **Terminal** tab and run:

```
$ uv add polars
```

Then try the import again in the Console. It will work.

`ModuleNotFoundError` = you need to `uv add` the package first.

Try It: Import and Use `polars`

After running `uv add polars`, switch to the **Console** and type:

```
>>> import polars as pl
>>> df = pl.DataFrame({
...     "ticker": ["AAPL", "MSFT", "TSLA"],
...     "price":  [142.50, 378.90, 248.50],
... })
>>> print(df)
```

```
shape: (3, 2)
+--------+-------+
| ticker | price |
| ---    | ---   |
| str    | f64   |
+========+=======+
| AAPL   | 142.5 |
| MSFT   | 378.9 |
| TSLA   | 248.5 |
+--------+-------+
```

You just created a table of data in Python. Don't worry about the `{ }` and `[ ]` syntax — we'll cover those after the break.

The Three uv Commands You Need

What you want to do	Command (in the Terminal tab)
Set up a new project	`uv init`
Install a package	`uv add polars`
See what's installed	`uv pip list`

That's it. Three commands. Everything else is just Python.

The pattern: `uv add …` in the Terminal, then `import …` in the Console (or your `.py` file).

Exercise 2: Install and Import

Try It Yourself (3 minutes)

1. In the **Terminal** tab, run: `uv add polars plotnine pyfixest diff-diff`
2. Watch it install — it should only take a few seconds
3. In the **Console**, type: `import polars as pl`
4. Then type: `print(pl.__version__)` — you should see a version number
5. Open `pyproject.toml` in the editor — find where `polars` appears

If you see `ModuleNotFoundError`, check: are you in the **Console** (not Terminal)? Did `uv add` finish?

Data Structures

Why Data Structures Matter

Later in the module, `polars` will handle your actual datasets (tables of stock prices, returns, etc.). So why learn lists and dictionaries?

Because they are the building blocks that show up **everywhere** in Python:

Situation	Example
Passing options to a function	`yf.download(["AAPL", "MSFT"], …)`
Storing settings or parameters	`{"start": "2024-01-01", "end": "2024-12-31"}`
Collecting results in a loop	`rows.append({"ticker": t, "return": r})`
Organising any small group of values	`rates = [0.05, 0.07, 0.10]`

Think of lists and dictionaries as Python's **plumbing** — you won't store large datasets in them, but you can't write a useful program without them.

Variables: Giving Names to Values

```python
price = 142.50          # a decimal number
shares = 100            # a whole number
ticker = "AAPL"         # text (always in quotes)
is_profitable = True    # yes or no (True / False)

print(f"Ticker: {ticker}")
print(f"Price:  {price}")
print(f"Value:  {price * shares}")
```

```
Ticker: AAPL
Price:  142.5
Value:  14250.0
```

A variable is just a **name tag** you stick on a value. Python figures out the type automatically.

Lists: Ordered Collections

```
# A portfolio of stock tickers
portfolio = ["AAPL", "MSFT", "TSLA", "AMZN"]

print(portfolio[0])     # first item  (counting starts at 0!)
print(portfolio[-1])    # last item
print(len(portfolio))   # how many items
```

```
'AAPL'
'AMZN'
4
```

- Created with square brackets [ ]

Lists: Ordered Collections

```
# A portfolio of stock tickers
portfolio = ["AAPL", "MSFT", "TSLA", "AMZN"]

print(portfolio[0])     # first item  (counting starts at 0!)
print(portfolio[-1])    # last item
print(len(portfolio))   # how many items
```

```
'AAPL'
'AMZN'
4
```

- Created with square brackets [ ]
- Items have a fixed order (first, second, third...)

Lists: Ordered Collections

```
# A portfolio of stock tickers
portfolio = ["AAPL", "MSFT", "TSLA", "AMZN"]

print(portfolio[0])     # first item  (counting starts at 0!)
print(portfolio[-1])    # last item
print(len(portfolio))   # how many items
```

```
'AAPL'
'AMZN'
4
```

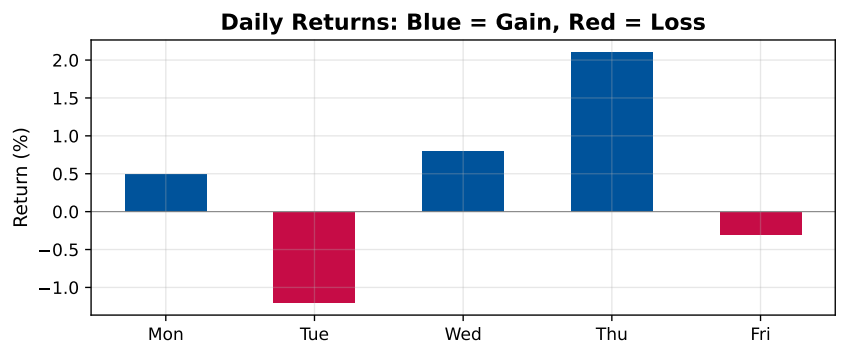
- Created with square brackets [ ]
- Items have a fixed order (first, second, third…)
- Counting starts at 0, not 1 (this trips everyone up at first!)

Useful Things You Can Do with Lists

```
portfolio = ["AAPL", "MSFT"]
portfolio.append("GOOGL")       # add to the end
portfolio.remove("MSFT")        # remove by name
print(portfolio)                # ['AAPL', 'GOOGL']

returns = [0.5, -1.2, 0.8, 2.1, -0.3]
print(sum(returns))      # total:  1.9
print(max(returns))      # best:   2.1
print(min(returns))      # worst: -1.2
print(len(returns))      # count:  5
```

Built-in functions like `sum()`, `max()`, `min()`, and `len()` work directly on lists.

Visualising Those Returns



Blue bars = positive days. Red bars = negative days. (We'll learn to make charts like this next week with `plotnine`.)

Dictionaries: Labelled Data

```python
# Stock info as a dictionary
stock = {
    "ticker": "AAPL",
    "price": 142.50,
    "shares": 100,
    "sector": "Technology",
}

print(stock["price"])          # look up by name
stock["price"] = 145.00        # update a value
print(stock["price"])
```

- Created with curly braces { }

Dictionaries: Labelled Data

```
# Stock info as a dictionary
stock = {
    "ticker": "AAPL",
    "price": 142.50,
    "shares": 100,
    "sector": "Technology",
}

print(stock["price"])          # look up by name
stock["price"] = 145.00        # update a value
print(stock["price"])
```

- Created with curly braces { }
- Each entry has a name (key) and a value

Dictionaries: Labelled Data

```python
# Stock info as a dictionary
stock = {
    "ticker": "AAPL",
    "price": 142.50,
    "shares": 100,
    "sector": "Technology",
}

print(stock["price"])          # look up by name
stock["price"] = 145.00        # update a value
print(stock["price"])
```

- Created with curly braces `{ }`
- Each entry has a name (key) and a value
- Like a row in a spreadsheet where each column has a header

Lists vs Dictionaries

	List	Dictionary
Looks like	`["AAPL", "MSFT"]`	`{"ticker": "AAPL", "price": 142}`
Access by	Position (number)	Name (key)
Best for	Sequences of similar items	A record with named fields
Analogy	A column in Excel	A row in Excel

For this course, these two are the ones you'll use most. We'll mention **tuples** `(1, 2, 3)` and **sets** `{1, 2, 3}` as we go, but don't worry about them now.

Live: Building a Portfolio

Live Demonstration

Create a dictionary of stock positions and compute values.

```python
portfolio = {
    "AAPL": {"shares": 50, "price": 142.50},
    "MSFT": {"shares": 30, "price": 378.90},
}

# Total value of the Apple position
aapl_value = (portfolio["AAPL"]["shares"]
              * portfolio["AAPL"]["price"])
print(f"AAPL position value: ${aapl_value:,.2f}")
```

```
AAPL position value: $7,125.00
```

Exercise 3: Data Structures

Try It Yourself (5 minutes)

1. Create a **list** of 5 stock tickers you know (e.g. `["AAPL", "TSLA", ...]`)
2. Print the first and last ticker
3. Create a **dictionary** for one stock with keys `"ticker"`, `"price"`, `"shares"`
4. Print the total value (price $\times$ shares)

Hint: Create a new Python file `exercise3.py` (New $\rightarrow$ New File $\rightarrow$ Python File), write your code, and click ▶ to run it.

Loops

Why Do We Need Loops?

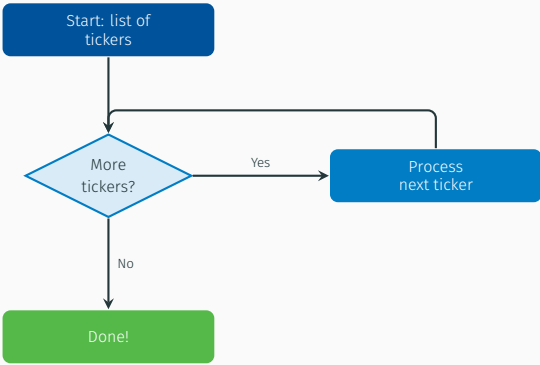
Computing the return for **1 stock** is easy:

```
value = 100 * 142.50
```

But what about **500 stocks**?

A loop says: *"do this for **each** item in a list"*

It's like dragging a formula down a column in Excel — but more powerful.



Your First `for` Loop

```python
tickers = ["AAPL", "MSFT", "TSLA", "AMZN"]

for ticker in tickers:
    print(f"Analysing {ticker}...")
```

```
Analysing AAPL...
Analysing MSFT...
Analysing TSLA...
Analysing AMZN...
```

How to read it: “For each ticker in the list, print a message.”
The indented line (4 spaces) is what repeats. Everything else runs once.

`range()`: Counting by Numbers

```python
# Compound interest: 10 years at 5%
value = 1000
for year in range(1, 11):
    value = value * 1.05
    print(f"Year {year:2d}: ${value:>10,.2f}")
```

```
Year  1: $  1,050.00
Year  2: $  1,102.50
Year  3: $  1,157.62
Year  4: $  1,215.51
Year  5: $  1,276.28
Year  6: $  1,340.10
Year  7: $  1,407.10
Year  8: $  1,477.46
Year  9: $  1,551.33
Year 10: $  1,628.89
```

- `range(5)` gives you 0, 1, 2, 3, 4

`range()`: Counting by Numbers

```python
# Compound interest: 10 years at 5%
value = 1000
for year in range(1, 11):
    value = value * 1.05
    print(f"Year {year:2d}: ${value:>10,.2f}")
```

```
Year  1: $  1,050.00
Year  2: $  1,102.50
Year  3: $  1,157.62
Year  4: $  1,215.51
Year  5: $  1,276.28
Year  6: $  1,340.10
Year  7: $  1,407.10
Year  8: $  1,477.46
Year  9: $  1,551.33
Year 10: $  1,628.89
```

- `range(5)` gives you 0, 1, 2, 3, 4
- `range(1, 11)` gives you 1, 2, 3, ..., 10

range(): Counting by Numbers

```python
# Compound interest: 10 years at 5%
value = 1000
for year in range(1, 11):
    value = value * 1.05
    print(f"Year {year:2d}: ${value:>10,.2f}")
```

```
Year  1: $  1,050.00
Year  2: $  1,102.50
Year  3: $  1,157.62
Year  4: $  1,215.51
Year  5: $  1,276.28
Year  6: $  1,340.10
Year  7: $  1,407.10
Year  8: $  1,477.46
Year  9: $  1,551.33
Year 10: $  1,628.89
```

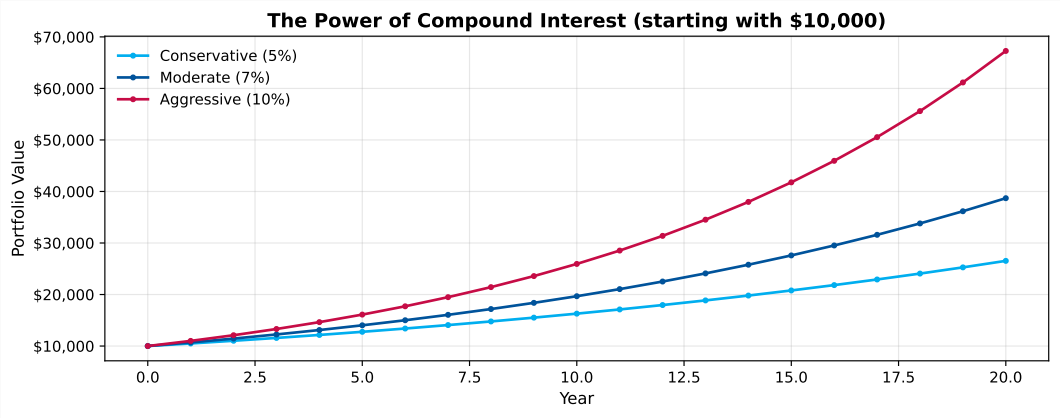
- `range(5)` gives you 0, 1, 2, 3, 4
- `range(1, 11)` gives you 1, 2, 3, ..., 10
- Useful for simulating time periods

Loops with Calculations

```python
prices = [142.50, 378.90, 248.50, 186.75]
shares = [50, 30, 20, 40]

total = 0
for i in range(len(prices)):
    position = prices[i] * shares[i]
    total += position              # shorthand for total = total + position
    print(f"Position {i+1}: ${position:,.2f}")

print(f"\nPortfolio value: ${total:,.2f}")
```

```
Position 1: $7,125.00
Position 2: $11,367.00
Position 3: $4,970.00
Position 4: $7,470.00

Portfolio value: $30,932.00
```

Compound Interest: Visualised



Same starting amount, different rates. Small differences compound into huge gaps over time.

The Code Behind That Chart

```
principal = 10000
rates = {"5%": 0.05, "7%": 0.07, "10%": 0.10}

rows = []
for label, rate in rates.items():      # .items() gives each key-value pair
    value = principal
    for year in range(0, 21):           #   for each year...
        rows.append({"rate": label,
                     "year": year,
                     "value": value})
        value = value * (1 + rate)      #   grow by the rate

# This is a nested loop: a loop inside a loop!
```

Two loops working together: the outer loop picks a rate, the inner loop simulates 20 years.

`while` Loops: Repeat Until a Condition

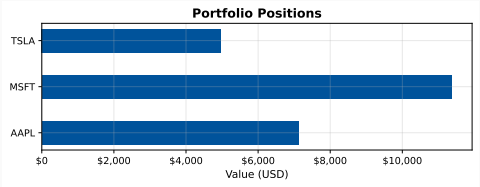
```python
# How many years to double your money at 7%?
value = 1000
target = 2000
years = 0

while value < target:
    value = value * 1.07
    years += 1                # add 1 to years

print(f"It takes {years} years to double at 7%")
```

```
It takes 11 years to double at 7%
```

Use **`for`** when you know how many times to repeat. Use **`while`** when you want to keep going **until** something happens.

Combining Loops and Data Structures

```python
portfolio = {
    "AAPL": {"shares": 50,
             "price": 142.50},
    "MSFT": {"shares": 30,
             "price": 378.90},
    "TSLA": {"shares": 20,
             "price": 248.50},
}

total = 0
for ticker, info in portfolio.items():  # each key + value
    value = info["shares"] * info["price"]
    total += value
    print(f"{ticker}: ${value:,.2f}")

print(f"\nTotal: ${total:,.2f}")
```



Loop over structured data to compute everything at once.

Exercise 4: Loops

Try It Yourself (5 minutes)

1. Write a `for` loop that prints each ticker in your list from Exercise 3
2. Compound returns exercise:
   - Start with `value = 1.0`
   - For each return in `[0.05, -0.02, 0.03, 0.01, -0.01]`, do: `value = value * (1 + r)`
   - Print the final value
3. **Bonus:** Use a `while` loop to find how many years $5,000 takes to reach $10,000 at 6%

Summary

What You Learned Today

Tools & setup

- Positron: Console vs Terminal vs Editor
- Python files (`.py`) and how to run them
- `uv init` / `uv add` for packages
- Virtual environments (your project's private toolbox)

Python fundamentals

- Variables and types (`int`, `float`, `str`, `bool`)
- Lists `[ ]` and dictionaries `{ }`
- `for` loops and `while` loops
- `range()` for counting
- Combining data structures with loops

Quick Self-Check

Can you answer these in your own words?

1. What is the difference between the Console and the Terminal in Positron?

Quick Self-Check

Can you answer these in your own words?

1. What is the difference between the Console and the Terminal in Positron?
2. What does `uv add polars` do, and where do you run it?

Quick Self-Check

Can you answer these in your own words?

1. What is the difference between the Console and the Terminal in Positron?
2. What does `uv add polars` do, and where do you run it?
3. What is the difference between a list and a dictionary?

Quick Self-Check

Can you answer these in your own words?

1. What is the difference between the Console and the Terminal in Positron?
2. What does `uv add polars` do, and where do you run it?
3. What is the difference between a list and a dictionary?
4. What does `for ticker in portfolio:` do?

Quick Self-Check

Can you answer these in your own words?

1. What is the difference between the Console and the Terminal in Positron?
2. What does `uv add polars` do, and where do you run it?
3. What is the difference between a list and a dictionary?
4. What does `for ticker in portfolio:` do?
5. When would you use `while` instead of `for`?

What's Next

- Week 4: real financial data with `polars` and `plotnine`

If something breaks: read the error message, restart your terminal, try again.
Still stuck? Email me at ce50@st-andrews.ac.uk.

What's Next

- Week 4: real financial data with `polars` and `plotnine`
  - Downloading stock prices, filtering tables, computing returns, making charts

If something breaks: read the error message, restart your terminal, try again.
Still stuck? Email me at ce50@st-andrews.ac.uk.

What's Next

- Week 4: real financial data with `polars` and `plotnine`
  - Downloading stock prices, filtering tables, computing returns, making charts
- **Homework:** complete today's exercises and experiment in the Console

If something breaks: read the error message, restart your terminal, try again.
Still stuck? Email me at ce50@st-andrews.ac.uk.

What's Next

- Week 4: real financial data with `polars` and `plotnine`
  - Downloading stock prices, filtering tables, computing returns, making charts
- **Homework:** complete today's exercises and experiment in the Console
- Resources:

If something breaks: read the error message, restart your terminal, try again.
Still stuck? Email me at ce50@st-andrews.ac.uk.

What's Next

- Week 4: real financial data with `polars` and `plotnine`
  - Downloading stock prices, filtering tables, computing returns, making charts
- **Homework:** complete today's exercises and experiment in the Console
- Resources:
  - Python tutorial: https://docs.python.org/3/tutorial/

If something breaks: read the error message, restart your terminal, try again.
Still stuck? Email me at ce50@st-andrews.ac.uk.

What's Next

- Week 4: real financial data with `polars` and `plotnine`
  - Downloading stock prices, filtering tables, computing returns, making charts
- **Homework:** complete today's exercises and experiment in the Console
- Resources:
  - Python tutorial: https://docs.python.org/3/tutorial/
  - `uv` docs: https://docs.astral.sh/uv/

If something breaks: read the error message, restart your terminal, try again.
Still stuck? Email me at ce50@st-andrews.ac.uk.

What's Next

- Week 4: real financial data with `polars` and `plotnine`
  - Downloading stock prices, filtering tables, computing returns, making charts
- **Homework:** complete today's exercises and experiment in the Console
- Resources:
  - Python tutorial: https://docs.python.org/3/tutorial/
  - `uv` docs: https://docs.astral.sh/uv/
  - polars guide: https://docs.pola.rs/

If something breaks: read the error message, restart your terminal, try again.
Still stuck? Email me at ce50@st-andrews.ac.uk.

What's Next

- Week 4: real financial data with `polars` and `plotnine`
  - Downloading stock prices, filtering tables, computing returns, making charts
- **Homework:** complete today's exercises and experiment in the Console
- Resources:
  - Python tutorial: https://docs.python.org/3/tutorial/
  - `uv` docs: https://docs.astral.sh/uv/
  - polars guide: https://docs.pola.rs/
  - plotnine docs: https://plotnine.org/

If something breaks: read the error message, restart your terminal, try again.
Still stuck? Email me at ce50@st-andrews.ac.uk.