

# Lesson 01: Python Setup

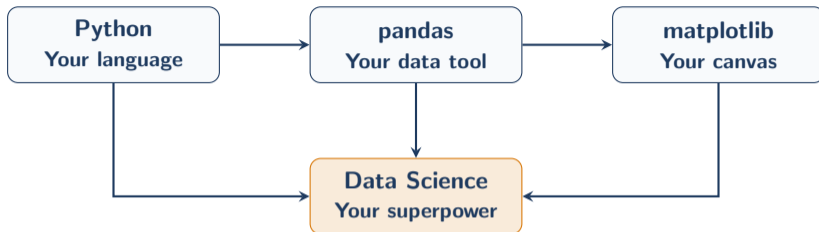
Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

# Welcome to Data Science!



**This semester:** 48 lessons from zero Python to deploying ML models.  
Today we start with the absolute basics – Python as your new calculator.

---

**No prior programming experience required**

# Learning Objectives

After this lesson, you will be able to:

- **Explain** Python's role in the data science ecosystem
- **Navigate** the Jupyter Notebook environment
- **Create** variables and assign values of different types
- **Perform** arithmetic operations on numeric data
- **Manipulate** strings using methods and f-strings

**Finance Application:** Store and compute stock prices, returns, and portfolio values.

---

Bloom's levels: Explain, Navigate, Create, Perform, Manipulate

# What You Need to Get Started

## What you bring:

- Curiosity about data
- A laptop with internet
- Willingness to make mistakes

## No experience needed.

If you can use a calculator, you can write Python.

## What you'll gain:

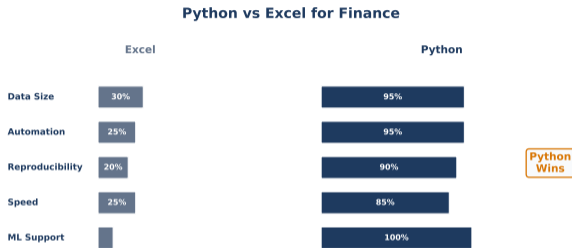
- Analyze thousands of data points
- Automate repetitive calculations
- Create publication-quality charts
- Build predictive models

**By lesson 48:** You deploy a machine learning app to the cloud.

---

Python is the #1 language for data science (Stack Overflow 2024)

# The Python Data Science Ecosystem



*Python scales to millions of rows; Excel struggles past 100K*

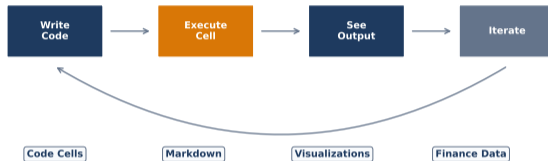
**Why Python?** Millions of rows, reproducible scripts, free and open-source.

---

**Python:** millions of rows, full automation. **Excel:** manual, limited.

# The Jupyter Notebook Environment

## Jupyter Notebook Workflow



*Interactive Development Environment*

**Key elements:** Cells (code or text) — Run button (Shift+Enter) — Kernel status

---

Jupyter = Julia + Python + R – interactive development for data science

# Working in Jupyter

## Cell Types:

- **Code cells** – run Python
- **Markdown cells** – formatted text
- **Output** – appears below code

## Essential Shortcuts:

Shift+Enter – run cell

Esc+A – insert cell above

Esc+B – insert cell below

Esc+DD – delete cell

## Best Practices:

1. One idea per cell
2. Add markdown explanations
3. Run cells top to bottom
4. Restart kernel if confused

## Common Pitfall:

Running cells out of order causes errors. When in doubt:

```
Kernel > Restart & Run All
```

---

Treat notebooks like lab journals – document as you go

# Variables: Named Containers

A variable is a name that points to a value in memory

## Variable Assignment: Names Point to Values

Python Code:

```
stock_price = 185.50  
ticker = "AAPL"  
shares = 100
```

Memory:



*Variable names are labels that point to values in memory*

**Key: Use descriptive names like `stock_price`, not `x`**

**Assignment:** `price = 185.50` creates a label "price" pointing to the value 185.50

---

Python uses dynamic typing – no need to declare types

# Variable Naming Rules

## Valid Names:

`stock_price` – descriptive

`dailyReturn` – camelCase OK

`_private` – underscore prefix

`price2` – numbers OK (not first)

## Python Convention:

Use `snake_case` for variables:

`buy_price`, `total_value`

## Invalid Names:

`2price` – starts with number

`my-var` – hyphen not allowed

`class` – reserved keyword

`for` – reserved keyword

## Good vs Bad:

`stock_price` Y descriptive

`sp` N cryptic

`x` N meaningless

---

Descriptive names make code self-documenting

# Python Data Types

## Python Basic Data Types



*Use `type(variable)` to check any type*

Finance uses: int for shares, float for prices, str for tickers, bool for signals

**Four fundamental types:** `int` (whole numbers), `float` (decimals), `str` (text), `bool` (True/False)

---

Use `type(variable)` to check any variable's type

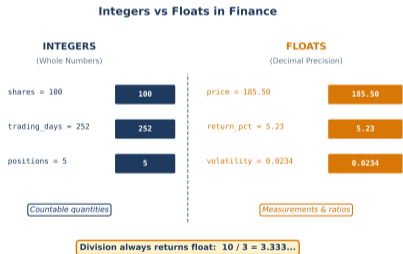
# Integers vs Floats in Finance

## Integers (`int`):

- Shares owned: `shares = 100`
- Trading days: `days = 252`
- Position count: `n = 5`

## Floats (`float`):

- Price: `price = 185.50`
- Return: `ret = 0.0523`
- Percentage: `pct = 5.23`



## Key Rule:

Division always returns float:

`10 / 3` → 3.333...

`10 // 3` → 3 (floor division)

---

Use `int` for counts, `float` for prices and returns

# Arithmetic Operations

## Basic Operators:

Op	Symbol	Example
Add	+	$100 + 50 = 150$
Subtract	-	$185 - 150 = 35$
Multiply	*	$50 * 185 = 9250$
Divide	/	$35 / 150 = 0.233$
Power	**	$2 ** 3 = 8$
Floor div	//	$7 // 2 = 3$
Modulo	%	$7 \% 2 = 1$

## Order of Operations (PEMDAS):

1. Parentheses ( )
2. Exponents \*\*
3. Multiply / Divide \* / // %
4. Add / Subtract + -

### Example:

$$2 + 3 * 4 \rightarrow 14$$

$$(2 + 3) * 4 \rightarrow 20$$

---

When in doubt, use parentheses to make order explicit

# Finance: Computing Returns

## Portfolio Calculation:

```
buy_price = 150.25
current_price = 185.50
shares = 50

investment = shares * buy_price
# 7512.50

current_value = shares * current_price
# 9275.00

profit = current_value - investment
# 1762.50

return_pct = (profit / investment) * 100
# 23.46%
```

## Compound Interest:

```
principal = 10000
rate = 0.07 # 7% annual
years = 5

future = principal * (1 + rate) ** years
# 14,025.52
```

**Formula:**  $FV = PV \times (1 + r)^n$

Python computes this instantly.  
Excel needs a formula per cell.

---

Python turns financial formulas into one-line calculations

# String Operations

## Strings: Text Data

```
ticker = "AAPL"
```



### Key Operations

<code>len(ticker)</code>	→	<b>4</b>	<i>Count characters</i>
<code>"Stock: " + ticker</code>	→	<b>"Stock: AAPL"</b>	<i>Concatenation (joining)</i>
<code>str(185.50)</code>	→	<b>"185.50"</b>	<i>Convert number to text</i>

*Finance Use: Tickers, dates, company names are all strings*

Strings are always in quotes: "AAPL" or 'AAPL'

**Strings store text:** ticker symbols, company names, dates, log messages

---

**Strings are immutable – methods return new strings, originals unchanged**

# Working with Strings

## String Methods:

```
ticker = "aapl"  
ticker.upper() → "AAPL"  
"AAPL".lower() → "aapl"  
len("AAPL") → 4
```

## Concatenation:

```
"NASDAQ:" + "AAPL"  
→ "NASDAQ:AAPL"
```

## Indexing:

```
ticker[0] → "A"  
ticker[-1] → "L"
```

## F-strings (Recommended):

```
price = 185.50  
shares = 50  
msg = f"Price:  ${price}"  
# "Price:  $185.5"  
msg = f"{shares} shares"  
# "50 shares"  
msg = f"Return:  {23.46:.1f}%"  
# "Return:  23.5%"
```

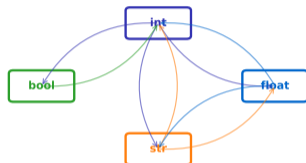
**Key:** F-strings embed variables directly inside strings with `f"..."`

---

F-strings are the modern way to format strings (Python 3.6+)

# Type Conversion

## Type Conversion (Casting)



### Conversion Examples

```
int("150")      # "150" -> 150   int(150.99)    # 150.99 -> 150 (truncates!)
float("150.50") # "150.50" -> 150.5 bool(0)      # 0 -> False
str(150)        # 150 -> "150"  bool(150)   # 150 -> True
```

## Converting Between Types:

```
int("100") → 100
```

```
float("185.5") → 185.5
```

```
str(185.5) → "185.5"
```

```
bool(1) → True
```

## Finance Use Case:

CSV files deliver prices as strings.

You must convert before calculating.

## Gotcha:

```
int("185.5") fails!
```

```
Use int(float("185.5")) instead.
```

---

Real-world data arrives as strings – conversion is essential

# Checkpoint: Check Your Understanding

**Q1:** What type is "42"?

- (a) int      (b) str      (c) float      (d) bool

**Q2:** What does `type(3.14)` return?

- (a) int      (b) str      (c) float      (d) pi

**Q3:** What is `10 / 3`?

- (a) 3      (b) 3.0      (c) 3.333...      (d) Error

---

Answers: Q1=b (quotes make it a string), Q2=c, Q3=c (division returns float)

# Boolean Values and Logic

## Booleans: True or False

```
price = 185.50
```

## Comparisons Return True or False

<code>price &gt; 150</code>	→	<b>True</b>	<i>185.50 is greater than 150</i>
<code>price &lt; 100</code>	→	<b>False</b>	<i>185.50 is not less than 100</i>
<code>price == 185.50</code>	→	<b>True</b>	<i>Exact match</i>
<code>price != 200</code>	→	<b>True</b>	<i>185.50 is not equal to 200</i>

## Comparison Operators

<code>&gt;</code> greater than	<code>&lt;</code> less than	<code>==</code> equal to
<code>&gt;=</code> greater or equal	<code>&lt;=</code> less or equal	<code>!=</code> not equal

*Booleans are used for decisions -- covered in Lesson 3 (Control Flow)*

**Booleans:** Only two values – True or False. The basis of all decision-making in code.

---

Booleans control if-statements, filters, and trading signals

# Comparison and Logical Operators

## Comparison Operators:

Op	Meaning	Example
>	greater than	185 > 150 → T
<	less than	100 < 50 → F
>=	greater or equal	5 >= 5 → T
<=	less or equal	3 <= 2 → F
==	equal to	5 == 5 → T
!=	not equal	5 != 3 → T

## Logical Operators:

and – both must be True

or – at least one True

not – inverts True/False

## Trading Example:

```
price = 145
```

```
volume = 1200000
```

```
buy = (price < 150) and  
      (volume > 1000000)
```

```
# buy = True
```

---

Comparisons return booleans – the foundation of trading logic

# Finance: Simple Interest vs Compound Interest

## Simple Interest:

principal = 10000

rate = 0.05

years = 3

interest = principal \* rate \* years

# 1500.00

total = principal + interest

# 11500.00

$$I = P \times r \times t$$

## Compound Interest:

principal = 10000

rate = 0.05

years = 3

total = principal \* (1 + rate) \*\* years

# 11576.25

interest = total - principal

# 1576.25

$$FV = PV \times (1 + r)^n$$

**Difference:** Compound earns \$76.25 more – interest on interest.

---

Compound interest: "the eighth wonder of the world" – attributed to Einstein

# Your Python Toolkit So Far

## Data Types

`int, float`

`str, bool`

## Operations

`+ - * / ** //`

`> < == != and or`

## Tools

Jupyter Notebook

`type(), print()`

Variables store values: `price = 185.50`

F-strings format output: `f"Price: ${price}"`

---

These building blocks combine into powerful data analysis tools

# Hands-on Exercise (25 min)

## Create a Jupyter notebook and complete:

1. Create variables for a stock portfolio:
  - `ticker = "AAPL", shares = 50`
  - `buy_price = 150.25, current_price = 185.50`
2. Calculate portfolio metrics:
  - Total investment: `shares * buy_price`
  - Current value: `shares * current_price`
  - Profit and return percentage
3. Create a boolean: `is_profitable = profit > 0`
4. Print results using f-strings:  
`f"{ticker}: {return_pct:.2f}% return"`

---

Save your notebook – we build on this next lesson

---

Everything from here builds on what you learned today

# Lesson Summary

## Key Takeaways:

- Jupyter Notebook is our interactive development environment
- Four basic types: `int`, `float`, `str`, `bool`
- Variables store values with descriptive names (`snake_case`)
- Arithmetic follows PEMDAS order of operations
- F-strings format output: `f"Price: ${price}"`
- Type conversion needed when reading external data

**Next Lesson:** Data Structures – lists and dictionaries to organize collections of data.

---

**Practice:** try different variable types in Jupyter tonight