Imperial College
London

8 November 2025

> python3 session_1.py

Introduction to Python: Programming Basics for Mathematicians

>

Imperial College
London

Main Topics:

- What is Python and why mathematicians use it
- Variables and data types
- Basic arithmetic operations and expressions
- Lists for storing multiple values
- Functions for reusable calculations
- Practical examples: calculating option payoffs

Imperial College
London

What is Python — What & Why

What it is:

Python is a programming language that allows you to give instructions to a computer to perform calculations, process data, and solve problems automatically.

Why it exists:

- **Automate repetitive tasks:** Instead of calculating 100 option payoffs by hand, write code once and run it many times.
- **Handle large datasets:** Process thousands of stock prices instantly.
- **Visualize results:** Create graphs and charts to understand patterns.
- **Reproducibility:** Share your exact method with others.

Why mathematicians use it:

Python excels at numerical computations, has extensive mathematical libraries, and uses syntax similar to mathematical notation.

Imperial College
London

What is a Variable?

> **Definition:**
>
> A **variable** is a named container that stores a value in your program. Think of it as a labeled box where you can put data and retrieve it later.

Variables in Python work like variables in mathematics, but they can hold different types of data (numbers, text, etc.).

Basic syntax:

```
variable_name = value
```

Example:

```
stock_price = 100
```

This creates a variable called `stock_price` and stores the value 100 in it.

Imperial College
London

Types of Data

Numeric Types:

Integers (int): Whole numbers like 100, -5, 0

Floats (float): Decimal numbers like 5.50, 0.01, 120.75

Text Type:

Strings (str): Text enclosed in quotes like "call option", "Apple"

Boolean Type:

Booleans (bool): True or False values for logical operations

Example:

```
strike_price = 100 # integer
premium = 5.50 # float
option_type = "call" # string
is_profitable = True # boolean
```

Imperial College
London

Numbers — Integers

> **Integer Definition**
>
> **Integers** are whole numbers without decimal points. They are used when you need exact counts or when decimal precision is not required.

Examples in finance:

```
number_of_shares = 50
strike_price = 100
days_to_expiry = 30
```

Why use integers:

When counting discrete items (shares, contracts, days) or when the value is naturally a whole number.

Imperial College
London

Numbers — Floats

> **Float Definition**
>
> **Floats** (floating-point numbers) are numbers with decimal points. They are essential for precise calculations involving fractions, percentages, and continuous values.

Examples in finance:

```
option_premium = 5.50

interest_rate = 0.01

stock_price = 120.75

percentage_return = 15.5
```

When to use floats:

Prices, rates, percentages, and any measurement requiring decimal precision.

Imperial College
London

Basic Arithmetic Operations

Basic operators:

`+` Addition

`-` Subtraction

`*` Multiplication

`/` Division

`**` Exponentiation

Example: Calculating profit

```
initial_price = 100
final_price = 120
profit = final_price - initial_price
# profit equals 20
```

Example: Calculating percentage return

```
percentage_return = (profit / initial_price) * 100
# percentage_return equals 20.0
```

Imperial College
London

Expressions — Combining Values and Operations

> **Definition:**
>
> An **expression** is a combination of values, variables, and operators that Python evaluates to produce a result. Expressions are the building blocks of calculations.

Simple expression:

```
120 - 100
```

Result: 20

Complex expression:

```
((120 - 100) - 5) / 5 * 100
```

Result: 300.0

This calculates the percentage return on an option:
(profit - premium) / premium × 100

Key idea: Expressions follow mathematical order of operations (PEMDAS/BODMAS).

Imperial College
London

Strings — Working with Text

String Definition

Strings are sequences of characters used to represent text. They are enclosed in single quotes ('...') or double quotes ("...").

Examples:

```
asset_type = "call option"

company_name = "Apple"

ticker_symbol = 'AAPL'
```

Imperial College
London

String Operations

Common operations:

```
# Concatenation (joining strings)
message = "Stock price: " + "100"

# String length
length = len("Apple") # equals 5
```

When to use strings:

Labels, names, descriptions, or any non-numeric data.

Imperial College
London

Lists — Storing Multiple Values

> **List Definition**
>
> A **list** is an ordered collection of values stored in a single variable. Lists can hold multiple items of any type and are enclosed in square brackets.

Syntax:

```
list_name = [item1, item2, item3, ...]
```

Example: Stock prices over time

```
prices = [100, 105, 110, 115, 120]
```

Example: Mixed data

```
option_data = ["call", 100, 5.50, True]
```

Lists are like mathematical sequences: they maintain order and can be indexed.

Imperial College
London

Working with Lists — Access and Modify

Accessing elements:

Python uses **zero-based indexing** (first element is at position 0).

```
prices = [100, 105, 110, 115, 120]
first_price = prices[0] # equals 100
last_price = prices[4] # equals 120
second_price = prices[1] # equals 105
```

Useful list operations:

```
# Get list length
num_prices = len(prices) # equals 5
# Add new element
prices.append(125) # adds 125 to end
# Get last element
```

Imperial College
London

What are Functions and Why Do They Exist?

> **Definition:**
>
> A **function** is a reusable block of code that performs a specific task. Functions take inputs (called parameters), process them, and return an output.

Mathematical analogy: Just like f(x) = x² in mathematics, Python functions transform inputs into outputs.

$$f(x) = x^2$$

Why functions exist:

- **Reusability:** Write code once, use it many times
- **Organization:** Break complex problems into smaller pieces
- **Clarity:** Give meaningful names to operations
- **Efficiency:** Avoid repeating the same code

Example: Instead of calculating option payoff manually each time, create a function that does it automatically.

Imperial College
London

Creating Functions – Basic Structure

Function syntax:

```
def function_name(parameter1, parameter2):
# Code to execute
result = some_calculation
return result
```

Simple example: Calculate profit

```
def calculate_profit(initial_price, final_price):
profit = final_price - initial_price
return profit
```

Imperial College
London

Using Functions – Key Components

Using the function:

```
my_profit = calculate_profit(100, 120)
# my_profit equals 20
```

Key components:

- **def:** keyword to define a function
- **Function name:** descriptive identifier
- **Parameters:** inputs in parentheses
- **return:** sends result back to caller

Imperial College
London

Function Example – Call Option Payoff

A call option gives the right to buy at the strike price. The payoff depends on whether the stock price exceeds the strike price.

Function definition:

```
def call_option_payoff(stock_price, strike_price, premium):
    if stock_price > strike_price:
        intrinsic_value = stock_price - strike_price
        profit = intrinsic_value - premium
    else:
        profit = -premium # Lose the premium paid
    return profit
```

Example usage:

```
payoff = call_option_payoff(120, 100, 5)
# payoff equals 15
```

When stock price (120) > strike price (100), you gain 20 but paid 5 premium, so net profit is 15.

Imperial College
London

Function Example — Put Option Payoff

A put option gives the right to sell at the strike price. The payoff depends on whether the stock price falls below the strike price.

Function definition:

```
def put_option_payoff(stock_price, strike_price, premium):
    if stock_price < strike_price:
        intrinsic_value = strike_price - stock_price
        profit = intrinsic_value - premium
    else:
        profit = -premium # Lose the premium paid
    return profit
```

Example usage:

```
payoff = put_option_payoff(80, 100, 5)
# payoff equals 15
```

When stock price (80) < strike price (100), you gain 20 but paid 5 premium, so net profit is 15.

Imperial College
London

Practical Example — Stock Price Rises to £120

Scenario Setup:

- Current stock price: £100
- Call option strike: £100
- Option premium: £5

Case 1 — Stock Price Rises to £120

Stock investment:

```
stock_profit = 120 - 100
# stock_profit = 20
stock_return = (20 / 100) * 100
# stock_return = 20%
```

Call option investment:

```
option_profit = call_option_payoff(120, 100, 5)
# Intrinsic value = 120 - 100 = 20
# Profit = 20 - 5 = 15
option_return = (15 / 5) * 100
# option_return = 300%
```

With only £5 invested in the option, you earn £15 profit — that's three times your initial outlay → +300% return.

Imperial College
London

Practical Example — Stock Price Falls to £90

Case 2 — Stock Price Falls to £90

Stock investment:

```
stock_profit = 90 - 100
# stock_profit = -10
stock_return = (-10 / 100) * 100
# stock_return = -10%
```

Call option investment:

```
option_profit = call_option_payoff(90, 100, 5)
# Stock below strike (worthless at expiry)
# Profit = 0 - 5 = -5
option_return = (-5 / 5) * 100
# option_return = -100%
```

You lose your entire £5 premium → -100% return.

This demonstrates the leverage effect: options amplify both gains and losses.

Imperial College
London

Visualizing Payoffs — Creating Graphs

Matplotlib is Python's most popular plotting library. It allows you to create professional graphs and charts to visualize option payoffs.

```
import matplotlib.pyplot as plt
# Generate stock price range
stock_prices = range(80, 130)
strike = 100
premium = 5
# Calculate payoffs for each price
payoffs = [call_option_payoff(price, strike, premium) for price in stock_prices]
# Create the plot
plt.plot(stock_prices, payoffs, 'b-', linewidth=2)
plt.xlabel('Stock Price (£)')
plt.ylabel('Profit/Loss (£)')
plt.title('Call Option Payoff Diagram')
plt.grid(True)
plt.axhline(y=0, color='k', linestyle='-')
plt.show()
```

This creates a visual representation of how your option profit changes with stock price.

Imperial College
London

Summary and Next Steps

What We Covered Today:

- **Variables:** Storing values in named containers
- **Data types:** Integers, floats, strings, booleans
- **Arithmetic operations:** +, -, *, /, **
- **Expressions:** Combining values and operators
- **Lists:** Storing multiple values in ordered collections
- **Functions:** Creating reusable code blocks
- **Option payoff calculations:** Applying Python to finance

Practice Recommendations:

- Write your own functions for put options and other derivatives
- Experiment with different stock prices and strike prices
- Try calculating portfolio returns with multiple positions

Next Steps:

- Learn loops (for, while) to process multiple scenarios
- Explore matplotlib for visualizing payoff diagrams
- Study NumPy for efficient numerical computations

> **You now have the foundation to automate financial calculations with Python. Keep practicing!**