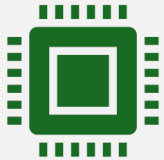
Advanced Python

Surendra Panpaliya

Week1 (Mon, Tue, Thurs)

Day 1: Python Recap + Environment & Tooling



Day 2: Functional Programming & Object-Oriented Design

Day 3: Advanced Python Concepts

Week2 (Mon, Tue, Wed, Thurs)

Day 4: Concurrency and Async Programming

Day 5: Web Services with FastAPI

Day 6: Azure Functions & Cloud Deployment

Day 7: Testing, Linting & Final Project

Day 1: Python Recap + Environment & Tooling

- Quick Python vs C# syntax mapping
- Variables, data types, control flow in Python
- Comprehensions (List, Dict, Set)
- Functions: *args, **kwargs, lambda
- Tooling: pip, venv, poetry, dependency locking

Day 1: Python Recap + Environment & Tooling

Hands-On Lab:

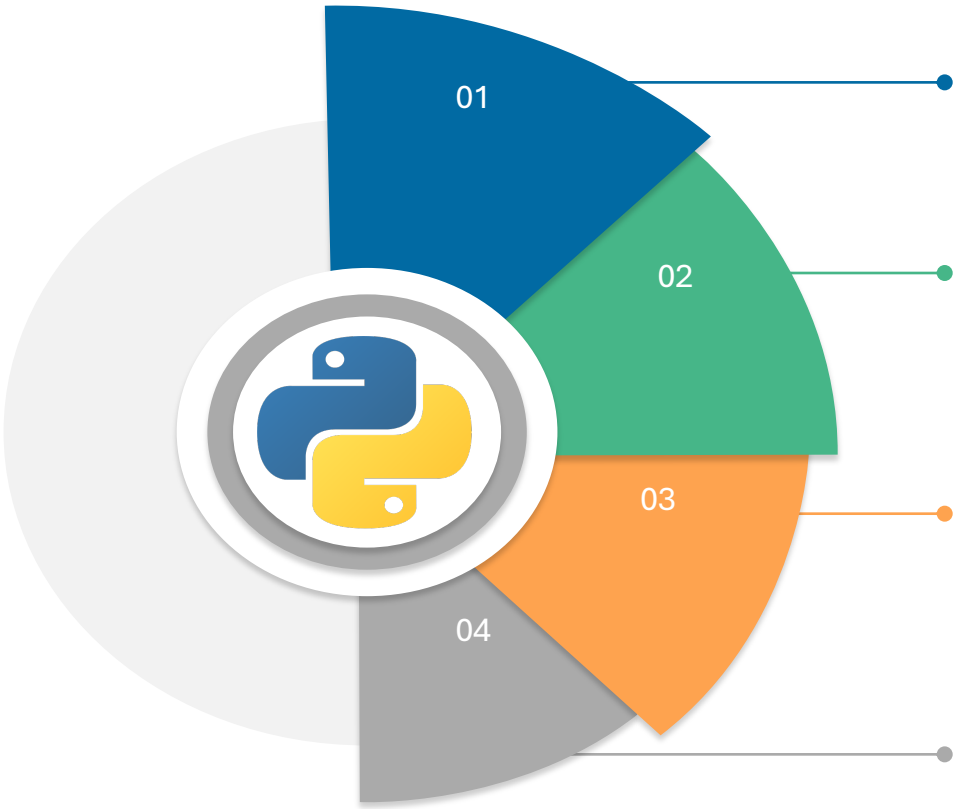
Set up Python project & create utility functions

Python vs C#: Static typing vs dynamic,

Main method vs scripts,

No semicolons/curly braces

What is Python ?



General Purpose

Interpreted

Object Oriented Scripting Language

High Level Programming Language

Language Overview

Feature	Python	C#
Type	Interpreted, Dynamic	Compiled, Static
Platform	Cross-platform	Originally Windows (.NET), now cross-platform (.NET Core/.NET 5+)
Use Cases	Data Science, Scripting, AI/ML, Automation, Web Dev	Enterprise Apps, Game Dev (Unity), Web APIs, Desktop Apps
Learning Curve	Easy	Moderate
Paradigm	Multi-paradigm: OOP + Functional + Imperative	Multi-paradigm: Strong OOP, Functional with LINQ

Syntax Comparison

Variable Declaration

```
x = 10      # No type declaration needed
```

```
int x = 10; // Type must be declared or inferred (via var)
```

Print / Output

```
print("Hello")
```

```
Console.WriteLine("Hello");
```

Data Types

Concept	Python	C#
Integer	num = 5	int num = 5;
Float	pi = 3.14	float pi = 3.14f;
String	name = "John"	string name = "John";
Boolean	is_valid = True	bool isValid = true;

Control Structures

If-Else:

```
if x > 0:
    print("Positive")
else:
    print("Negative")
```

Control Structures

If-Else:

```
if (x > 0)
    Console.WriteLine("Positive");
else
    Console.WriteLine("Negative");
```

For Loop

```
for i in range(5):
    print(i)
```

```
for (int i = 0; i < 5; i++)
    Console.WriteLine(i);
```

While Loop

```
while x < 10:
    x += 1
```

```
while (x < 10)
{
    x++;
}
```

Functions / Methods

Python

```
def add(a, b):
    return a + b
```

C#

```
int Add(int a, int b)
{
    return a + b;
}
```

Object-Oriented Programming

Class and Object: Python

```
class Person:
    def __init__(self, name):
        self.name = name

    def greet(self):
        print("Hello", self.name)
```

Class and Object C#

```
class Person
{
    public string Name;
    public Person(string name)
    {
        Name = name;
    }
    public void Greet()
    {
        Console.WriteLine("Hello " + Name);
    }
}
```

Data Structures

List / Array : Python

```
fruits = ["apple", "banana"]
print(fruits[0])
```

C#

```
string[] fruits = { "apple", "banana" };
Console.WriteLine(fruits[0]);
```

List Comprehension / LINQ

```
squares = [x * x for x in range(5)]
```

```
var squares = Enumerable.Range(0, 5).Select(x => x * x).ToList();
```

Exception Handling

Python

```
try:
    1 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
```

Exception Handling

C#

```
try
{
    int x = 1 / 0;
}
catch (DivideByZeroException)
{
    Console.WriteLine("Cannot divide by zero");
}
```

Advanced Features

Feature	Python	C#
Decorators	✅	❌ (Attributes used instead)
LINQ (Query syntax)	❌	✅ Powerful querying on collections
Async/Await	✅	✅
Generics	Limited (via duck typing)	Fully supported
Memory Management	Automatic (GC)	Automatic (GC)
Lambda Functions	✅	✅

Tooling and Ecosystem

Area	Python	C#
IDEs	VS Code, PyCharm, Jupyter	Visual Studio, Rider, VS Code
Web Frameworks	Django, Flask, FastAPI	ASP.NET Core
Game Dev	Not common	Unity (C# exclusive)
Data Science	Strong (NumPy, pandas, scikit-learn)	Weak
Mobile Dev	Kivy, BeeWare	Xamarin, MAUI
Enterprise Dev	Moderate	Strong

Performance

Area	Python	C#
Execution Speed	Slower (interpreted)	Faster (compiled to IL, JIT optimized)
Startup Time	Faster	Slower than scripts, but faster runtime
Memory Usage	Higher	More optimized

When to Use

Use Case	Recommended Language
Rapid Prototyping	Python
AI/ML & Data Science	Python
Web API (Enterprise)	C# (ASP.NET Core)
Game Development	C# (Unity)
Automation/Scripting	Python
Cross-platform Desktop Apps	C# (with .NET MAUI), Python (basic GUIs)

Summary

Category	Winner
Simplicity	Python
Performance	C#
AI & Data Science	Python
Enterprise Software	C#
Rapid Development	Python
Developer Tools & IDEs	C#
Versatility	Tie

Variables, Data Types, Control flow

Surendra Panpaliya

Variables in Python

A named location

used to store data in memory.

Don’t need to declare

the type of a variable explicitly.

It’s inferred at runtime.

Variables in Python

```
x = 10          # Integer
name = ”Dev"   # String
price = 99.99    # Float
is_valid = True  # Boolean
```

Basic Data Types

Type	Description	Example
int	Integer numbers	x = 5
float	Decimal numbers	pi = 3.14
str	Text	name = "John"
bool	Boolean value	is_valid = True
list	Ordered, mutable collection	fruits = ["apple", "banana"]
tuple	Ordered, immutable collection	coords = (4, 5)
dict	Key-value pairs	student = {"name": "Amit", "age": 20}
set	Unordered, unique values	colors = {"red", "blue"}

Basic Data Types

```
age = 25                    # int
height = 5.9                # float
is_student = False          # bool
name = "Surendra"           # str
marks = [85, 90, 95]        # list
point = (10, 20)            # tuple
info = {"name": "Amit"}     # dict
unique_ids = {101, 102, 103} # set
```

Control Flow in Python

Control flow determines

the order in which

statements are executed

A. Conditional Statements

🔷 if...elif...else

```
marks = 85

if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
else:
    print("Grade C")
```

B. Loops

🔷 **For Loop (iterates over a sequence)**

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print(fruit)
```

B. Loops

🔷 For Loop with range()

```
for i in range(3):
    print(i)

# Output: 0, 1, 2
```

B. Loops

🔷 **While Loop (runs while condition is true)**

```
count = 1
while count <= 5:
    print("Count:", count)
    count += 1
```

C. Loop Control Statements

Statement	Description	Example
break	Exit the loop	if i == 3: break
continue	Skip current iteration	if i == 3: continue
pass	Do nothing (placeholder)	if i == 3: pass

C. Loop Control Statements

🔷 **Example:**

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

Summary Table

Concept	Python Example
Variable	name = "Amit"
Data Type - int	age = 25
Data Type - float	pi = 3.14
Data Type - str	msg = "Hello"
If Statement	if x > 0:
For Loop	for i in range(5):
While Loop	while x < 10:

Comprehensions (List, Dict, Set)

Surendra Panpaliya

What is Comprehension in Python?

Concise way

to create new sequences

(like lists, sets, or dictionaries)

using a **single line of code**,

often with **conditions**.

1. List Comprehension

Basic Syntax:

[expression for item in iterable]

Example 1: Square numbers

```
squares = [x**2 for x in range(5)]
```

```
print(squares)  # Output: [0, 1, 4, 9, 16]
```

1. List Comprehension

Example 2: Even numbers only

```
evens = [x for x in range(10) if x % 2 == 0]
```

```
print(evens)  # Output: [0, 2, 4, 6, 8]
```

2. Dictionary Comprehension

Basic Syntax:

{key_expr: value_expr for item in iterable}

Example 1: Number and its square

```
square_dict = {x: x**2 for x in range(5)}
print(square_dict)  # Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

2. Dictionary Comprehension

Example 2: Filter values

```
marks = {'Amit': 90, 'Raj': 45, 'Ravi': 78}

passed = {k: v for k, v in marks.items() if v >= 50}

print(passed)  # Output: {'Amit': 90, 'Ravi': 78}
```

3. Set Comprehension

Basic Syntax:

{expression for item in iterable}

Example 1: Square of numbers

```
squares_set = {x**2 for x in range(5)}

print(squares_set)  # Output: {0, 1, 4, 9, 16}
```

3. Set Comprehension

Example 2: Unique characters in a string

```
unique_chars = {ch for ch in "banana"}

print(unique_chars)  # Output: {'b', 'a', 'n'}
```

Summary Table

Type	Syntax Example	Output Example
List Comprehension	[x**2 for x in range(5)]	[0, 1, 4, 9, 16]
With Condition	[x for x in range(10) if x % 2 == 0]	[0, 2, 4, 6, 8]
Dict Comprehension	{x: x**2 for x in range(3)}	{0: 0, 1: 1, 2: 4}
Set Comprehension	{x % 3 for x in range(7)}	{0, 1, 2}

Bonus: Conditional Comprehension (Ternary Expression)

```
result = [x if x % 2 == 0 else "odd" for x in range(5)]

print(result)  # Output: [0, 'odd', 2, 'odd', 4]
```

Functions

*args, **kwargs, lambda

Surendra Panpaliya

1. *args → Variable-Length

Positional Arguments

*args allows a function to accept

any number of positional arguments.

Internally, it is treated as a **tuple**.

1. *args → Variable-Length

```
def add_all(*args):
    total = 0
    for num in args:
        total += num
    return total

print(add_all(1, 2, 3))       # Output: 6
print(add_all(10, 20, 30, 40)) # Output: 100
```

2. **kwargs → Variable-Length

Keyword Arguments

**kwargs allows a function

to accept **any number of keyword arguments**.

Internally, it is treated as a **dictionary**.

2. **kwargs → Variable-Length

```
def print_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_info(name="Amit", age=25, city="Pune")

# Output:
# name: Amit
# age: 25
# city: Pune
```

3. Lambda → Anonymous (Inline) Function

lambda creates a **short,**

unnamed function in a single line.

Syntax:

lambda arguments: expression

3. Lambda → Anonymous (Inline) Function

Example 1: Basic usage

```
square = lambda x: x * x
print(square(5))  # Output: 25
```

3. Lambda → Anonymous (Inline) Function

Example 2: With multiple arguments

```
add = lambda a, b: a + b
print(add(3, 4))  # Output: 7
```

3. Lambda → Anonymous (Inline) Function

Example 3: In sorted() with key

```
students = [("Amit", 90), ("Raj", 75), ("Ravi", 85)]
students.sort(key=lambda x: x[1])
print(students)
# Output: [('Raj', 75), ('Ravi', 85), ('Amit', 90)]
```

Summary Table

Concept	Meaning	Treated As	Example
***args**	Variable positional args	Tuple	*args = (1, 2, 3)
****kwargs**	Variable keyword args	Dict	**kwargs = {'x': 1, 'y': 2}
lambda	Anonymous function	Function	lambda x: x**2

Bonus: Combine All Together

```
def demo(a, b, *args, **kwargs):
    print("a:", a)
    print("b:", b)
    print("args:", args)
    print("kwargs:", kwargs)

demo(1, 2, 3, 4, name="Amit", age=25)
```

Tooling
pip, venv, poetry, dependency locking

Surendra Panpaliya

🛠️ 1. pip – Python Package Installer

pip is the **default tool** for installing Python packages from PyPI.

```
pip install requests          # Install a package

pip install numpy==1.23.0     # Install specific version

pip uninstall requests        # Remove package
```

🛠️ 1. pip – Python Package Installer

```
pip list                    # Show installed packages

pip freeze > requirements.txt    # Save installed packages

pip install -r requirements.txt  # Install from requirements file
```

🧪 2. venv – Virtual Environment

- Creates a **separate Python environment**
- isolated from global packages.
- Useful to manage dependencies per project.

🧪 2. venv – Virtual Environment

```
# Create virtual environment
python -m venv env

# Activate

# On Windows

env\Scripts\activate
```

🧪 2. venv – Virtual Environment

```
# On macOS/Linux
source env/bin/activate

# Deactivate
deactivate
```

🧪 2. venv – Virtual Environment

Why use it?

- Avoid version conflicts between projects.
- Keep your global Python clean.

📦 3. poetry – Dependency Management + Packaging

poetry handles:

- Dependency management
- Virtual environments
- Publishing Python packages

Install:

pip install poetry

3. poetry – Dependency Management + Packaging

```
poetry new my_project          # Create new project


cd my_project
poetry install                 # Install dependencies
poetry add requests            # Add a new dependency
poetry update                  # Update dependencies
poetry shell                   # Activate environment
```

Project Structure:

Poetry uses a pyproject.toml file (like package.json in Node.js).

[tool.poetry]

```
name = "my_project"
version = "0.1.0"
dependencies = {
   python = "^3.10",
   requests = "^2.28"
}
```

4. Dependency Locking

- Ensuring **exact versions** of all dependencies
- (and their sub-dependencies) are recorded.
- Helps maintain consistency across
- development, testing, and
- deployment environments.

Tools

Tool	Lock File
pip	requirements.txt (via pip freeze)
pip-tools	requirements.txt + requirements.in
poetry	poetry.lock
pipenv	Pipfile.lock

✅ Example: With poetry

```
poetry lock     # Generates poetry.lock
```

This poetry.lock file captures **all pinned versions** and is used to reproduce the same environment.

Summary Table

Tool	Purpose	Example
pip	Install packages	pip install flask
venv	Isolate environments	python -m venv env
poetry	Modern dependency & project manager	poetry add pandas
Locking	Freeze versions for reproducibility	pip freeze > requirements.txt, poetry.lock

Python vs C#

Static vs Dynamic Typing

Main Method vs Script Execution

Syntax: No Semicolons or Curly Braces

1. Static Typing (C#) vs Dynamic Typing (Python)

Feature	Python	C#
Type System	Dynamic	Static
Type Declaration	Optional	Mandatory or inferred (var)
Flexibility	High	Strict

✅ 2. Main Method vs Script Execution

Feature	Python	C#
Entry Point	Script starts from top	Main() is required
Simplicity	High	Formal structure

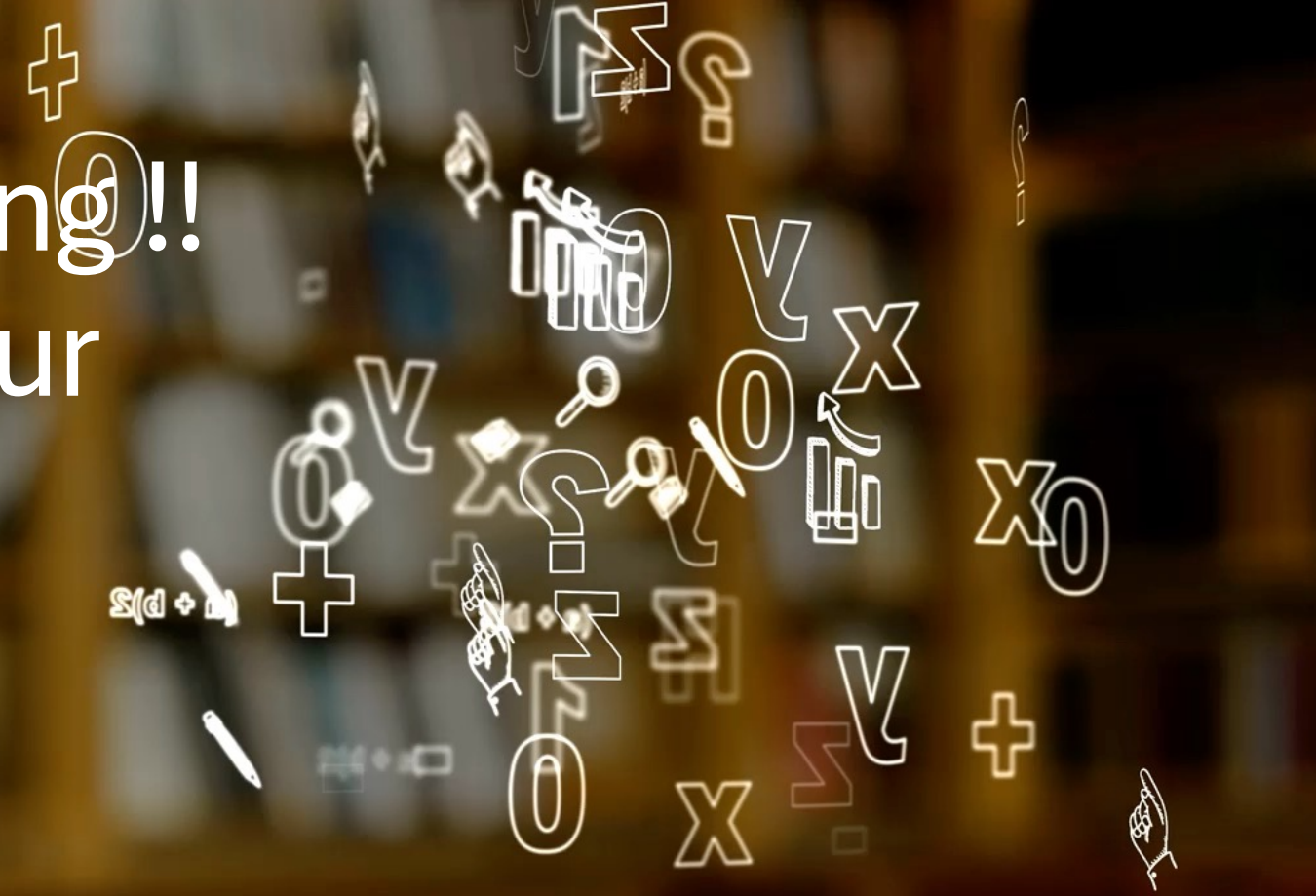
3. No Semicolons and Curly Braces in Python

Feature	Python	C#
Block Structure	Indentation-based	Curly braces {}
Statement Termination	No semicolon needed	Semicolon ; required

Conclusion Table

Feature	Python	C#
Typing	Dynamic	Static
Entry Point	Top-down script	Main() method
Syntax	Indentation-based, no semicolons	Curly braces & semicolons
Use Case	Fast prototyping, scripting, ML	Enterprise apps, games, strong OOP

Happy Learning!! Thanks for Your Patience ☺



Surendra Panpaliya

GKTCS Innovations