A Complete Introduction to Python

From Fundamentals to Execution

Part 1: Python Fundamentals

A beginner's guide to the Python programming language.

What is Python?

- A high-level, interpreted programming language.
- Created by Guido van Rossum and first released in 1991.
- Known for its simple, readable syntax.
- Emphasizes code readability and a design that allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java.

Why Python?

- **Easy to Learn:** Simple syntax makes it a great first language.
- **Versatile:** Used in web development, data science, artificial intelligence, automation, and more.
- **Large Community:** Extensive libraries and a supportive community.
- **Cross-Platform:** Runs on Windows, macOS, and Linux.

Basic Syntax: Variables

Variables are containers for storing data values.

```
# A variable named 'greeting' that holds a string
greeting = "Hello, World!"

# A variable named 'year' that holds an integer
year = 2025

print(greeting)
print(year)
```

Basic Syntax: Data Types

Python has various built-in data types:

- **Text Type:** `str` (e.g., "Hello")
- **Numeric Types:** `int`, `float`, `complex` (e.g., 10, 3.14)
- **Sequence Types:** `list`, `tuple`, `range`
- **Mapping Type:** `dict`
- **Set Types:** `set`, `frozenset`
- **Boolean Type:** `bool` (True or False)

Data Structures: Lists

A list is a collection which is ordered and changeable. Allows duplicate members.

```
# A list of fruits
fruits = ["apple", "banana", "cherry"]

# Accessing an item
print(fruits[1])  # Output: banana

# Adding an item
fruits.append("orange")

print(fruits)
```

Data Structures: Dictionaries

A dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

```
# A dictionary representing a person
person = {
  "first_name": "John",
  "last_name": "Doe",
  "year": 1990
}

# Accessing a value
print(person["first_name"]) # Output: John
```

Control Flow: `if` Statements

Used for conditional execution.

```
age = 20

if age >= 18:
  print("You are an adult.")
else:
  print("You are a minor.")
```

Control Flow: `for` Loops

Used for iterating over a sequence (like a list).

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

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

Functions

A block of code which only runs when it is called.

```
# Defining a function
def greet(name):
  """This function greets the person passed in as a parameter."""
  print("Hello, " + name + ". Good morning!")

# Calling a function
greet('Paul')
```

Modules and Libraries

- **Modules:** A file containing Python code.
- **Libraries:** A collection of modules.
- Python's extensive standard library and third-party packages (like NumPy, Pandas, and Requests) are a major strength.

```
# Import the 'math' module
import math

# Use a function from the module
print(math.sqrt(16))  # Output: 4.0
```

Part 2: The History of Python

Its origins and relationship to other languages.

The Creator: Guido van Rossum


Guido van Rossum

- Python was created in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands.
- He was the "Benevolent Dictator For Life" (BDFL) of the Python community until he stepped down in 2018.

The Name: Not a Snake!


Monty Python's Flying Circus

- The name "Python" comes from the British comedy group **Monty Python's Flying Circus**.
- Guido van Rossum was a fan of the show and wanted a name that was short, unique, and slightly mysterious.

Influences: A Mix of Languages

Python was inspired by several other programming languages:

- **ABC:** A teaching language that emphasized readability and simplicity. Python inherited its clean syntax and indentation-based structure from ABC.
- **C:** Python is written in C, and many of its standard library modules are C extensions for performance.
- **Modula-3:** Features like packages and exception handling were influenced by Modula-3.

Key Milestones

- **1991:** Python 0.9.0 is released.
- **2000:** Python 2.0 is released, introducing list comprehensions and a garbage collection system.
- **2008:** Python 3.0 is released. This was a major revision of the language that is not backward-compatible with Python 2.
- **2020:** Python 2 reaches its end-of-life. Python 3 is the present and future of the language.

Python's Philosophy: The Zen of Python

A collection of 19 "guiding principles" for writing computer programs that influence the design of the Python language.

```
>>> import this
```

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- ...and more.

Relationship to Other Languages

- **C/C++:** Python is often called a "glue language." It can be used to connect different components written in C or C++.
- **Java:** Python is generally considered easier to learn and write than Java. Both are object-oriented, but Python's dynamic typing offers more flexibility.
- **Perl:** Python was seen as a successor to Perl for scripting and text processing, with a cleaner and more readable syntax.
- **JavaScript:** Both are popular, dynamically-typed languages, but they dominate different domains (Python for backend/data science, JavaScript for frontend web development).

Part 3: From Code to Execution

REPLs, Virtual Environments, and How It All Works

The REPL: Your Interactive Playground

REPL stands for **R**ead-**E**val-**P**rint **L**oop. It's an interactive environment where you can type Python code and see the results immediately.

To start the REPL, just type `python` or `python3` in your shell:

```
$ python3
Python 3.12.4 (main, Jun 14 2025, 12:34:56) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

How Python Code is Executed

It's not magic! Python is an *interpreted* language, which involves a few steps to run your code.

1. **Parsing:** Your human-readable code (`.py` file) is broken down into smaller pieces (tokens).
2. **Compilation:** The code is compiled into a lower-level, platform-independent format called **bytecode**.
3. **Execution:** The **Python Virtual Machine (PVM)**, which is the runtime engine of Python, executes this bytecode.


Python Execution Flow

The Problem: Dependency Conflicts

Imagine you have two projects:

- **Project A** needs version `1.0` of a library.
- **Project B** needs version `2.0` of the *same* library.

If you install these system-wide, you'll have a conflict! One project will likely break.

The Solution: Virtual Environments

A virtual environment is an isolated Python environment that has its own installation directories, at a particular location on your system, which doesn’t share libraries with other virtual environments.

This allows each project to have its own set of dependencies, independent of other projects.

Creating a Virtual Environment

Python comes with a built-in module called `venv` to create virtual environments.

1. **Navigate to your project directory:**

```
mkdir my-project
cd my-project
```

2. **Create the environment:**

```
python3 -m venv venv
```

This creates a `venv` directory containing a private version of

Activating the Environment

To use the virtual environment, you need to "activate" it.

On macOS and Linux:

```
source venv/bin/activate
```

On Windows:

```
.\venv\Scripts\activate
```

Your shell prompt will change to show the name of the active environment, like `(venv) $`.

Managing Dependencies

Now that your environment is active, you can install packages using `pip`. They will be installed *only* in this environment.

```
(venv) $ pip install requests
...
Successfully installed requests-2.32.5 ...
```

You can save your project's dependencies to a file:

```
(venv) $ pip freeze > requirements.txt
```

And another developer can install them from that file:

Deactivating the Environment

When you're done working on the project, you can deactivate the environment.

```
(venv) $ deactivate
$
```

Your shell prompt will return to normal.

Next Steps & Thank You!

- **Practice:** The best way to learn is by doing.
- **Build Projects:** Create small applications to apply what you've learned.
- **Explore Libraries:** Discover the vast ecosystem of Python libraries.
- **Join a Community:** Engage with other Python developers.

Happy Coding!