Chapter 2:

Getting Started with Python

Introduction to the Python language

1. General presentation

Python is an **interpreted, object-oriented, and cross-platform programming language** designed to be **easy to read and use** . It allows you to write clear and efficient programs, regardless of the application domain: science, engineering, web development, artificial intelligence, or data analysis.

Its great strength lies in its **intuitive syntax** and **philosophy of simplicity** , which make it a very popular language among beginners as well as researchers and engineers.

2. History of the language

Python was created in the late 1980s by **Guido van Rossum** , a Dutch computer scientist working at **CWI (Centrum Wiskunde & Informatica)** in Amsterdam.

- **1989** : **Guido van Rossum** begins the development of Python, inspired by the **ABC language** .
- **1991** : First public version of **Python 0.9.0** , already including advanced features such as functions, exceptions and modules.
- **2000** : Launch of **Python 2.0** , more complete, but incompatible with the future version 3.
- **2008** : Release of **Python 3.0** , designed to correct inconsistencies in version 2 and improve readability and performance.
- **Today** , Python is one of the most widely used languages in the world, supported by a huge scientific, academic and industrial community.

3. Fundamental characteristics

- **Interpreted language** : Python executes line by line, without a prior compilation step.
- **Dynamically typed language** : the type of variables is defined automatically.
- **Object-oriented language** : everything is considered an object.
- **Cross-platform language** : works on Windows, macOS, Linux, etc.
- **Clear and readable syntax** : close to natural language.

4. Comparison with other languages

Features	Python	Fortran	C/C++	MATLAB
Year of creation	1991	1957	1972 / 1983	1984
Kind	Interpreted, object-oriented	Compiled, procedural	Compiled, object-oriented	Interpreter
Syntax	Very simple and readable	Heavy and rigid	Complex, error-prone	Simple, but specific to calculation
Main use	General programming, AI, web, science	Scientific computing, numerical simulation	Systems, performance, embedded	Numerical computing, signal processing
Performance	Slower than C, but optimizable with libraries (NumPy, Numba)	Very fast for matrix calculations	Very high performance (close to the original equipment)	Fast for matrix calculations, but proprietary
License	Open source	Open source	Open source	Commercial
Community	Very large	Scientist	Technique / system	Scientist and academic

5. Specific comparison with Fortran and MATLAB

Python vs Fortran

- Fortran remains an excellent language for heavy numerical computations and scientific modeling.
- Python, thanks to libraries like **NumPy** , **SciPy** and **Matplotlib** , allows you to do the same calculations **more simply** and **with better readability** .
- Python is more modern, object-oriented, and better integrated with artificial intelligence tools.

Python vs C/C++

- **compiled** languages , therefore faster.
- Python is **much simpler** to write and maintain.
- Python is preferred for **rapid prototyping** , while C/C++ is chosen for **performance-critical applications** .

Python vs MATLAB

- MATLAB is widely used in the scientific field and digital engineering.

- Python, along with **NumPy** , **Pandas** and **Matplotlib** , offers the same computing and visualization **capabilities .**
- The advantage of Python is that it is **free** , **open source** , and used in **many other fields** (web, AI, automation…).

6. Advantages of Python

Simplicity and readability

Python's clear syntax allows students and researchers to focus on logic rather than code structure.

Large standard library

Python contains many built-in modules: math, random, datetime, os, etc.

Active community

A huge community offers scientific libraries:

- **NumPy** , **Pandas** : data processing
- **Matplotlib** , **Seaborn** : visualization
- **SciPy** : scientific computing
- **TensorFlow** , **PyTorch** : artificial intelligence
- **Sympy** : symbolic computation (like MATLAB)

Portability

The same code works on all systems without modification.

Integration

Python interfaces easily with other languages such as **C** , **C++** , **Fortran** or **Java** , making it very versatile.

7. Conclusion

Python has established itself as a **universal language** in the scientific, industrial, and academic worlds. Its **rapid learning curve** , **clear syntax** , **high flexibility** , and **scientific power** make it an indispensable tool for students, teachers, and researchers in **chemistry, physics, engineering,** and **scientific computing** .

Today, Python has become the “common language” among scientists, programmers, and engineers.

Applications of Python in the fields of Chemistry and Physics

1. Introduction

Python has become an indispensable tool in experimental and theoretical sciences. Its **simplicity** , **computing power** , and extensive **scientific libraries** make it an ideal language for **researchers, teachers, and students in chemistry and physics** . It allows for the rapid and interactive **modeling, simulation, visualization, and analysis of physical and chemical phenomena.**

2. Applications in Chemistry

a) Theoretical and computational chemistry

Python is widely used for quantum computing, molecular modeling, and ab initio simulations. It can interact with specialized software such as:

- **Gaussian** , **ORCA** , **NWChem** , **CP2K** (via Python scripts to automate calculations),
- **ASE (Atomic Simulation Environment)** for building, simulating and visualizing atomic systems,
- **PySCF** , **Psi4** , **RDKit** for quantum chemistry and computational chemistry.

Example: automated calculation of the energies of several molecules, optimization of structures or scanning of potential energy surfaces.

b) Analytical chemistry and data processing

Modern instruments (UV spectroscopy, IR spectroscopy, NMR spectroscopy, chromatography, etc.) produce **large volumes of data** . Python allows you to:

- read and process spectrum files (CSV, TXT, JCAMP-DX, etc.),
- filter the signals,
- plotting curves (absorbance, intensity, time, etc.),
- make curve adjustments (fit) and calculate concentrations or physico-chemical constants.

Useful libraries:

- NumPy , SciPy → numerical computation
- Matplotlib , Seaborn → plotting and visualization
- Pandas → experimental data management
- lmfit → spectral fitting (e.g., IR lines or NMR peaks)

c) Organic chemistry and biochemistry: Python can:

- to simulate chemical reactions,
- calculate molecular properties (molar mass, polarity, logP, etc.),
- modeling molecular dynamics (with **MDAnalysis** , **OpenMM** , **PyMOL API**),
- visualize 3D structures of molecules from .pdb or .mol files .

Example application: studying the flexibility of a protein or the interaction between a ligand and an enzyme (*molecular docking*).

d) Materials chemistry and nanotechnology : Python is used for:

- analyze the crystalline structure of materials (with pymatgen , ase , matminer),
- study the electronic properties (band gap, density of states, etc.),
- modeling nanofluids and composite materials.

Example: Simulating the diffusion of a non-Newtonian fluid in a porous medium or calculating the thermal conductivity of a nanomaterial.

3. Applications in Physics

a) Classical physics and mechanics: Python allows modeling:

- the movements of projectiles, harmonic oscillators, pendulums, etc.
- fluid dynamics (via matplotlib , numpy , or solvers like FiPy),
- heat transfer phenomena (conduction, convection, radiation).

Example: Numerical solution of the heat equation or simulation of laminar flow in a cavity.

b) Quantum and atomic physics: Python is used for:

- solve the Schrödinger equations,
- simulate potential wells,
- visualize atomic orbitals,
- perform quantum matrix and state calculations with NumPy and Sympy .

Example: Studying the behavior of an electron in a quantum dot using eigenfunctions.

c) Materials physics and simulation : Python allows:

- to model the crystal structure,
- to simulate heat diffusion or transport,
- to calculate electronic properties (conductivity, density of states, Fermi energy).

Libraries:

- LAMMPS (via Python interface),
- pymatgen , ase , phonopy for atomistic modeling.

d) Plasma physics, fluids and electromagnetism

Thanks to its matrix and graphics computing capabilities, Python is used to:

- solve the Navier-Stokes equations,
- to study the behavior of electric and magnetic fields,
- simulate the phenomena of diffusion, viscosity or inclined magnetic field.

Example: Simulation of a magnetic nanofluid flow in a partially porous cavity (as in your research work).

4. Advantages of Python for students and researchers

- **Easy to learn** : simple syntax, close to natural language.
- **Versatile** : a single language for modeling, analysis, and visualization.
- **Powerful** : handles complex calculations and large volumes of data.
- **Free and open source** : accessible to all students.
- **Active scientific community** : numerous libraries and tutorials.
- **Interoperable** : can communicate with Fortran, C, MATLAB, etc.

5. Conclusion

Python is currently **the most comprehensive and accessible digital tool for chemical and physical sciences** . It connects theory to practice: from quantum mechanics calculations to the visualization of experimental data, including the modeling of complex systems.

1. Setting up and preparing a development environment

Before starting to program in Python, it is essential to install a suitable environment.

a) Installing Python

Python is available for free on the official website: https://www.python.org/downloads/

Steps:

1. Download the latest stable version (Python 3.x).
2. During installation, check the **"Add Python to PATH" box** to be able to run Python from the terminal.
3. Confirm and complete the installation.

To verify that Python is properly installed:

```
python --version
```

Or

```
python3 --version
```

b) Development Environments (DEs)

Several environments are available:

- **IDLE** (provided with Python)
- **Visual Studio Code** (free and very popular)
- **PyCharm** (free version: Community Edition)
- **Jupyter Notebook** (ideal for teaching and interactive testing)

c) First test

Once installed, open the IDE or a terminal, then type:

```
Print ("Hello, Python!")
```

If the message appears, the installation was successful.

Setting up and preparing a Python development environment

1. Standard installation from the official website

The standard installation of Python is done via the official website https://www.python.org . After installation, the Python interpreter and the **IDLE editor** are available for writing and running simple programs.

manually installing additional libraries (such as NumPy, Matplotlib, etc.) using the command:

```
pip install numpy
```

2. Installation via Anaconda (recommended method for scientists)

What is an Anaconda?

Anaconda is a free distribution of Python (and R) specifically designed for scientific computing, data science, and machine learning. It already contains most of the necessary scientific libraries:

- **NumPy** (numerical computation)
- **Pandas** (data analysis)
- **Matplotlib** (visualization)
- **SciPy** (scientific computing)
- **SymPy** (symbolic computation)
- **Jupyter Notebook** (interactive experimentation interface)
- **Spyder** (a scientific development environment similar to MATLAB)

Advantages of Anaconda

- **Quick and easy** installation of Python and its libraries.
- User-friendly interface thanks to **Anaconda Navigator** .
- Easy management of **virtual environments** (multiple versions of Python side by side).
- No need to manually install the libraries.
- Recommended for **students in chemistry, physics, biology, AI** , etc.

Anaconda Installation Steps

1. Visit the official website: https://www.anaconda.com
2. Download the version that is compatible with your system (Windows, macOS or Linux).
3. Follow the installation instructions.
4. Once installed, open:
    - **Anaconda Navigator** (graphical interface)
    - or **Spyder / Jupyter Notebook** to run your Python programs.

Example of use with Spyder

Spyder is an integrated environment that comes with Anaconda and is very similar to **MATLAB** . It allows you to:

- to write Python code in a tabbed editor,
- to execute the code line by line or in blocks,
- to display the graphs directly in the console,
- and to view the variables in a **"workspace" type window** .

3. Other methods of using Python

Method	Description	Use Cases
Jupyter Notebook	Interactive interface (cell-by-cell execution).	Ideal for practical work and scientific demonstrations.
Google Colab	Online version of Jupyter hosted by Google, without installation.	Very useful for students: accessible from a browser.
VS Code	Versatile editor with Python extension.	For advanced projects and software development.

Conclusion

chemistry and physics students , **Anaconda is the most suitable** method : it facilitates scientific programming without worrying about installation or compatibility issues. Thanks to tools like **Spyder** and **Jupyter Notebook** , Python becomes a true digital laboratory for computation, simulation, and analysis of experimental data.

Other Python distributions and environments

Although **Anaconda** is the most popular for scientists, **there are other solutions** for installing and managing Python efficiently.

Distribution / Environment	Description	Main advantages	Target audience
Anaconda	Complete distribution for data science and scientific computing.	Very rich in libraries (NumPy, SciPy, Matplotlib, Pandas, etc.), graphical interface (Navigator).	Students, researchers, engineers in science and AI.
Miniconda	A lighter version of Anaconda.	Faster, lighter installation; allows you to install only the necessary packages.	Advanced users wanting total control.
WinPython	Portable distribution for Windows.	No installation required; includes Spyder, Jupyter, NumPy, SciPy.	Students and teachers using Windows.
Canopy (Enthought)	Distribution geared towards scientific research and teaching.	Professional interface, scientific analysis tools.	Physics/chemistry laboratories and teachers.
ActivePython	Commercial distribution for businesses.	Stability, professional technical support, enhanced security.	Companies and industrial projects.
Standard	Official basic	Simple, lightweight,	Developers and

Python (python.org)	version.	flexible; customizable with `pip` .	beginners.

2. Basic Python Syntax

Python is an **interpreted language** , **readable** and **structured by indentation** .

a) Indentation

In Python, code blocks are defined by indentations (spaces), not by curly braces {}:

```
if True:
print("This is a block")
```

Number of spaces or gaps

- **The standard indentation in Python is 4 spaces** .
- These 4 spaces must be **identical** for all lines in the same block.
- Mixing spaces and tabs is **strongly discouraged .**

b) Case sensitivity

Python distinguishes between uppercase and lowercase letters:

```
Name = "Ali"
print(name) # Error, because "name" ≠ "Name"
```

c) Comments

- Single line:

  # This is a comment
- Multiline:
- """

  This is a

  comment

  across several lines
- """

3. Variables and constants

A **variable** is used to store a value in the computer's memory.

a) Declaring a variable: No type declaration is necessary:

```
x = 10
name = "Ahmed"
pi = 3.14
```

b) Name a variable

The rules:

- Do not start with a number

- Do not contain spaces or special characters
- Use underscores to separate words: my_variable

c) Constants

Python does not have true constants, but by convention, their names are written in **uppercase** :

PI = 3.14159

GRAVITY = 9.81

4. Fundamental Data Types

a) Integers (int)

These represent numbers without decimals:

a = 5

b = -3

b) Floats

These represent real numbers with decimals:

x = 3.14

y = -0.5

c) Booleans (bool)

They represent true/false logic:

true

false = False

d) Type conversion

One type can be converted into another:

x = int(3.5) # 3

y = float(4) # 4.0

z = str(10) # "10"

To find out the type of a variable: type(x)

5. The operators

a) Arithmetic operators

Operator	Meaning	Example	Result
+	Addition	3 + 2	5
-	Subtraction	5 - 1	4
*	Multiplication	4 * 2	8
/	Real division	7/2	3.5
//	Entire division	7 // 2	3
%	Modulo (remainder)	7% 2	1

**	Power	3 ** 2	9

b) Comparison operators

Operator	Meaning	Example	Result
==	Equal to	5 == 5	True
!=	Different from	4 != 5	True
<	Less than	3 < 4	True
>	Superior to	5 > 3	True
<=	Less than or equal to	3 <= 3	True
>=	Greater than or equal to	5 >= 4	True

c) Logical operators

Operator	Meaning	Example	Result
and	AND logical	True and False	False
gold	OR logic	True or False	True
not	logically nonsensical	Not True	False

6. Input and output instructions

a) Input using input()

Allows you to request a value from the user:

```
name = input("Enter your name: ")
print("Hello", name)
```

Values entered with input() are **always of type str** (string of characters).

Conversion: age = int(input("Enter your age: "))

b) Output with print()

Displays data on the screen:

```
print("Hello", name, "you are", age, "years old.")
```

We can also use the **f-string** (formatted strings):

```
print(f"Hello {name}, you are {age} years old.")
```

7. Assignment and Expressions

a) Simple assignment

```
x = 10
y = 5
```

b) Multiple assignment

```
a, b, c = 1, 2, 3
```

c) Combined allocation

```
x = 5
```

```
x += 2 # is equivalent to x = x + 2
x *= 3 # is equivalent to x = x * 3
```

d) Expressions

An **expression** combines variables, constants, and operators:

```
z = (x + y) * 2
```

Exercises:

1. Algorithm (in pseudocode)

```
Beginning
Read A, B, C
If (A > B) and (A > C) then
Max ← A
Otherwise, if (B > C) then
Max ← B
Otherwise
Max ← C
End
Display "The maximum is:", Max
END
```

Equivalent Python program: Method with if conditions

```
# Program to determine the maximum between three numbers

# Reading the values
A = float(input("Enter the value of A: "))
B = float(input("Enter the value of B: "))
C = float(input("Enter the value of C: "))
# Comparison
if A > B and A > C:
Max = A
elif B > C:
    Max = B
else:
Max = C

# Displaying the result
print("The maximum is:", Max)
```

```
# Calculating the factorial of a number using a for loop
# Request for value from the user
N = int(input("Enter an integer: "))
# Initialization
fact = 1
# Calculating the factorial
for i in range(1, N + 1):
    fact *= i
# Displaying the result
print(f"The factorial of {N} is: {fact}")
```