People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

ⵓⵖⵉⵣⵀ ⵏ ⵓⵙⵏⵎⵙⵍ ⵙⵏⵏⵙ ⵏ ⵙⵍⵓⵙ ⵙⵓⵓⵍⵓⵍ

Reference Guide for the Early-Stage Training of Doctoral Students

« Programming Fundamentals and Techniques »

2024 - 2025

National Steering and Monitoring Committee:

- M. BENYAMINA Saïd — President of the National Commission
- Mme. BOUALLOUCHE Rachida — Director of Doctoral Training
- M. ZEBOUCHI Mohamed Abderraouf — Deputy Director of Doctoral Training
- M. BOUAROURI Djaafar — President of the Regional Conference of Central Universities
- M.LATRECHE Mohamed El Hadi — President of the Regional Conference of Eastern Universities
- M. CHAALAL Ahmed — President of the Regional Conference of Western Universities

National Pedagogical Committee for Programming Fundamentals and Techniques

- Dr. Said GADRI — President
- Pr. Abdelkader LAOUID — Expert Member
- Dr. Abdelkader GHAZLI — Expert Member
- Pr. Karim SEHABA — Expert Member
- Dr. Abdelhakim HANNOUSSE — Expert Member
- Dr. Samia ZOUAOUI — Expert Member
- Dr. Mohamed Cherif BOUKALA — Expert Member
- Dr. Belkacem KHALDI — Expert Member
- Pr. Safia NAIT-BAHLOUL — Expert Member

Table of Contents

1. Preamble

In today's rapidly evolving world—shaped by programming, artificial intelligence, and cutting-edge technologies—researchers must remain in a state of constant adaptation. Programming has become the cornerstone of modern innovation and scientific advancement, transforming research methodologies across every discipline.

Acquiring programming skills has become an indispensable competency for doctoral students across all disciplines. It enables them to efficiently utilize digital tools and optimize their research work. In this context, a programming training program has been developed to provide them with the fundamental knowledge and essential tools in this crucial domain.

As part of this approach, the integration of advanced concepts such as computational thinking, computational modeling, and logical reasoning aims to empower doctoral students with a structured and methodical approach to solve scientific problems. This training not only allows them to understand the theoretical foundations of programming languages and data structures but also to apply them concretely in their research, leveraging tools and platforms suited to their respective disciplines.

Furthermore, particular emphasis is placed on developing essential transversal skills, such as autonomy in learning new technologies, interdisciplinary collaboration, and the optimization of research processes. By mastering these skills, doctoral students will be better prepared to tackle complex challenges related to their research topics, innovate, and propose relevant solutions to emerging issues in their respective fields.

2. Training Objectives

- Enable doctoral students to acquire essential programming competencies for human-computer interaction in research contexts.
- Strengthen computational thinking and logical reasoning to analyze, structure, and solve complex problems.
- Provide doctoral students with the necessary skills to understand, design, and develop reliable and efficient algorithms and applications to address research-related problems.
- Explore programming languages and leverage them for solving research problems.

- Master database manipulation and analysis tools using programming techniques.
- Acquire skills in collaborative programming.
- Gain expertise in object-oriented programming and apply it to develop reliable and efficient programs and applications.
- Foster innovation and technological creativity through hands-on programming practice.
- Promote the use of open-source programming languages such as Python.

3. Training Organization

The training program is structured into five complementary parts, covering both fundamental principles and advanced programming concepts, with a particular focus on the Python language:

- **Part 1:** Introduction to Computer Programming
- **Part 2:** Basic Programming Concepts in Python
- **Part 3:** File and Database Handling
- **Part 4:** Structured, Modular, and Object-Oriented Programming
- **Part 5:** Collaboration and Best Practices in Programming

Each part is accompanied by a collection of interactive educational resources, including:

- *Lectures and seminars* to deepen understanding of key concepts
- *Practical workshops* to apply knowledge to real-world scenarios
- *Case studies and mini-projects* to promote experiential learning

The objective is to enhance doctoral students' programming proficiency and empower them with the necessary skills to design and develop efficient software solutions tailored to their research fields.

Given that the training program is designed for doctoral students from diverse disciplines and specialties—each with varying levels of programming expertise and different academic backgrounds in software development—the program adopts a modular and flexible structure. This approach ensures a targeted distribution of the program’s parts, adapting the learning experience to the specific needs of each field while maintaining a common foundation in

programming. The program is therefore structured based on two major disciplinary fields:

Doctoral Students in Humanities and Social Sciences (HSS):

- Parts 1, 2, and 3 are mandatory, providing a comprehensive introduction to programming fundamentals and development tools that enable students to design basic programs to address their research challenges.
- Parts 4 and 5 are optional and can be pursued based on individual research needs and the integration of advanced programming concepts.

Doctoral Students in Science and Technology (ST):

- Parts 1 and 2 are optional, as these students generally have prior programming knowledge. However, they can choose to take these modules to strengthen their foundational skills.
- Parts 3, 4, and 5 are mandatory, as they cover advanced concepts, methodologies, and essential tools for developing complex applications aligned with the demands of scientific and technical research.

This structured approach ensures a well-balanced and pedagogically coherent training program, tailored to the diverse profiles of doctoral students while maintaining alignment between disciplinary requirements and the cross-disciplinary skills necessary for scientific research.

4. Training Team

The training targets doctoral students and is overseen by faculty researchers and permanent researchers with expertise in programming at the university, regional, and national levels.

5. Training Program for Instructors

To ensure a smooth and effective implementation of doctoral training while fostering active engagement and increased contribution from various universities, two types of instructor training programs are proposed:

5.1. Seminar:

A 2 to 3-hour online seminar will be organized for instructors. Its objective is to present the pedagogical goals of the training, the topics covered, and the

monitoring and evaluation methods. This seminar will serve as an interactive space where instructors can familiarize themselves with the available tools and resources, discuss the teaching approaches to be adopted, and address any questions they may have. This session will be essential to ensure a consistent and standardized implementation of the training across different institutions.

5.2. Short and Intensive Training:

Some institutions that lack specialists in programming will have the opportunity to benefit from a short, intensive training program lasting 3 to 5 days. This training aims to provide instructors with fundamental programming skills, enabling them to effectively teach doctoral students. By the end of the program, participants will have acquired the necessary foundational knowledge to mentor and support doctoral students in their learning journey. This will ensure gradual skill development and foster the autonomy of institutions in teaching programming skills.

6. Language of Instruction and Teaching Mode

6.1. Language of Instruction:

In alignment with the guidelines set by the supervising authority, the use of English in this training is strongly encouraged. However, other languages may be used at the discretion of the instructor.

6.2. Teaching Mode:

The course follows a hybrid approach. The theoretical part, consisting of various course materials (books, PowerPoint presentations, videos, code examples, etc.), will be accessible to doctoral students online through a dedicated platform.

The in-person part will be supervised by domain-specific instructors appointed by the university institution. It will take the form of workshops where practical exercises and mini-projects will be conducted, considering the group's specialties and skill levels. At the beginning of these workshops, the instructors may review fundamental concepts using the provided reference materials. The table below summarizes the doctoral training requirements for the subject "*Programming Fundamentals and Techniques*."

Supervisory structure	Equipment and platforms	Instructors' profiles
• University institutions • Regional conferences • Distance learning	• Equipped training rooms • Available training platforms (Padoc, Telum, Moodle, etc.) and others to be developed if necessary • Internet connection	Faculty and permanent researchers meeting the following criteria: • Senior Lecturer A (i.e., MCA) or higher for faculty researchers. • Senior Research Fellow A (i.e., MRA) or higher for permanent researchers. • Possessing programming skills.

7. Programming with Open-Source Tools and Free Software:

Open source and free software are more than just an economic or technical choice—they represent a way of thinking that emphasizes accessibility and transparency of information. Open-source software refers to a category of programs whose code is fully or partially accessible, allowing users and developers to modify and adapt it to meet their specific needs.

Today, open-source and free software have become the standard in various fields, particularly in education, for several reasons, including:

- Free and open access
- Independence and long-term sustainability
- Effective tools for self-learning
- Rapid skill development in programming
- Privacy and ethical considerations
- Compatibility and interoperability
- Fostering an internal ecosystem within educational institutions

Due to these advantages, we have chosen to use open-source programming tools, including the Python programming language, its various associated libraries, and open-source code editors. This approach aims to create a free, independent, efficient, collaborative, and innovative learning environment for programming.

8. Evaluation Criteria

In alignment with the aforementioned hybrid modalities, the evaluation will cover both the online theoretical part and the in-person practical part. Exercises

in various formats (multiple-choice questions, coding tasks, etc.) will be made available on the platform, and doctoral students will be required to complete them. The in-person practical part will be assessed by the instructor at the respective institution (i.e., local training) and will focus on the completion of mini-projects. The final grade distribution is as follows:

- 20% for the online theoretical part.
- 80% for the in-person practical part, which includes:
  - 50% for continuous assessment of workshops.
  - 30% for a final mini-project.

It is important to note that only the mandatory parts are subject to evaluation. The final grade will be calculated using a percentage-based system (e.g., 80%, 75%, etc.), with a predefined minimum passing threshold.

9. Training Content

This section introduces the five parts of the training program, each structured into four subsections: the first provides an overview of the part, the second outlines the training part objectives, the third specifies the prerequisites in relation to other parts, and the fourth subsection provides the detailed content.

Part 1 – Introduction to Computer Programming

Overview:

This first part of the training program is dedicated to an introduction to computer programming using the visual language *Blockly*. It aims to present the fundamental concepts of computer programming by gradually introducing key topics such as programs and programming languages, variables and elementary data types, input/output operations, conditional structures, and loops. To facilitate understanding, each section will be enriched with illustrative examples.

Objectives:

- Understand the concepts of computer programs and programming languages.
- Explore the basics of computer programming using a visual language (*Blockly*).

- Learn fundamental concepts: variables, basic data types, input/output, conditional structures, and loops with *Blockly*.
- Step-by-step shift toward text-based coding languages.

Prerequisites:

- None

Detailed content:

- Program Concept
  - Understand the sequential logic of a program through the ordered execution of instructions.
  - *First Illustrative Program:* Reach a target using *Blockly-Maze*.
- Programming Language Concept
  - Define the notion of a programming language.
  - Compare visual and text-based programming languages.
- Variables and Data Types
  - Understand the necessity of data types for memory representation.
  - Introduction to variables and data types.
  - Elementary data types: *integer*, *float*, *string*, *boolean*.
  - Composite data types: *array*, *dictionary*, *list*, *object*.
  - *Second Illustrative Program:* Addition of two numbers.
- Input and Output
  - Learn how to interact with the user.
  - *Third Illustrative Program:* Display the message "*Hello [First Name] [Last Name]*", where the user inputs their first and last name.
- Conditional Structures
  - Understand conditional structures: *if* and *if-else*.
  - *Fourth Illustrative Program:* Conditional display: "*Hello Mr. [Last Name] [First Name]*" or "*Hello Ms. [Last Name] [First Name]*" based on gender input.
- Loops
  - Introduction to loops: *for, while, do-while*.
  - *Fifth Illustrative Program:* Execute an instruction multiple times to calculate the sum 1 + 2 + 3 … + 100.
- Transition to Text-Based Languages

- Convert visual code to text-based code (from *Blockly* to *Python*, *JavaScript*, etc.).

Part 2 – Basic Programming Concepts in Python

Overview:

Python is an interpreted, versatile, and easy-to-learn programming language, renowned for its clear and concise syntax. It is widely used in web development, artificial intelligence, data science, and automation. With its extensive libraries and powerful tools, Python is a preferred choice for both novices and experienced developers. In this part, we bridge visual and text-based programming by translating the basic programming concepts introduced in *Blockly* (Part I) into Python, demonstrating how core logic adapts to a real-world coding environment.

Objectives:

- Promote programming logic by learning Python's fundamental structures.
- Facilitate programming learning through Python's simple and readable syntax.
- Gradually enhance autonomy through progressively complex exercises and mini-project development.
- Prepare for professional skills by adopting a widely used language across various fields: *education, healthcare, industry, scientific research*.
- Encourage the use of open-source programming languages like Python.

Prerequisites:

- **Part 1:** Introduction to Computer Programming

Detailed content:

- Introduction to Python Programming Language
  - Why Choose Python?
  - Brief History of Python.
  - Python's Positioning.
  - Essential Tools for Python Programming.
  - Installing and Running Python.
  - First Python Program.

- Fundamental Elements of Python Programming
  - `print()` Output Function.
  - Comments in Python.
  - Variables and Expressions.
  - Basic Data Types: `int, float, str, bool, complex.`
  - Identifying Data Types.
  - Data Formatting.
  - Assignment.
  - Data Type Conversion.
  - Arithmetic Operators and Operator Precedence.
  - Logical Operators.
  - Common Mathematical Functions.
  - `input()` Function.
- Flow Control: Conditional Structures and Loops
  - `if` Statement.
  - `if-else` Statement.
  - Multiple Condition Statements.
  - `for` Loop: Simple and Using the range Function.
  - `while` Loop.
  - Loop Control Statements: `break`, `continue`, `pass.`
- Complex Data Structures in Python
  - Lists: Concept and Manipulation Functions.
  - Using Lists as Stacks.
  - 1D Arrays: Concept and Manipulation.
  - 2D Arrays: Concept and Manipulation.
  - Sets: Concept and Manipulation.
  - Tuples: Concept and Manipulation.
  - Dictionaries: Concept and Manipulation.
- String Processing
  - Determining the Length of a String.
  - String Concatenation.
  - Substrings' Extraction.
  - String Formatting.
  - Numbers to Strings Conversion and vice-versa.

Part 3 – File and Database Handling

Overview:

This part consists of two subparts. The first subpart introduces various file handling and processing techniques for commonly used formats (TXT,

CSV, JSON, XML). Doctoral students will learn the principles of reading, writing, and manipulating these files using the Python programming language. The second subpart focuses on database systems and their management. It covers the fundamental concepts of both relational and non-relational databases, along with their practical implementation using specialized Python libraries and tools.

Objectives:

- Learn the fundamental principles of file management.
- Get introduced to the main methods of reading, writing, modifying, and deleting files.
- Develop the ability to efficiently utilize tools and libraries dedicated to file management.
- Learn to create, manipulate, and query databases with *SQL* and *NoSQL*.
- Master the use of *SQLite*, *MySQL*, and *PostgreSQL* with Python.
- Use libraries such as *sqlite3*, *SQLAlchemy*, *pymysql*, and *psycopg2* for database management.
- Work with *NoSQL* databases like *MongoDB* using *pymongo*.
- Develop interactive applications leveraging databases.

Prerequisites:

- **Part 1:** Introduction to Computer Programming
- **Part 2:** Basic Programming Concepts in Python

Detailed content:

- **File Handling in Python**
  - Introduction to File Handling and Data Formats
    - Differences Between Text Files and Binary Files.
    - Overview and Uses of Common Formats: `TXT`, `CSV`, `YAML`, `JSON`, `XLSX`, `XML`.
  - File Creation and Management in Python (`TXT`, `CSV`, `JSON`, `XML`)
    - Creating and Opening Files.
    - Reading, Exploring, and Extracting Content.
    - Writing and Modifying Files.
    - Deleting Files.
  - Advanced File Operations
    - Merging and Concatenating Multiple Files.

  - Splitting and Segmenting a File into Multiple Parts.
  - File format conversion and transformation.

- **Database Handling in Python**
  - Introduction to Databases
    - Definition and Types of Databases (Relational vs. Non-Relational).
    - Key Concepts: *Tables*, *Relationships*, *Primary* and *Foreign Keys.*
    - *SQL* vs. *NoSQL* Comparison.
  - *SQL* Database Manipulation with Python
    - Introduction to SQLite: `sqlite3` in Python.
    - Creating and Managing SQLite Databases.
    - Introduction to MySQL and PostgreSQL.
    - Using `pymysql` and `psycopg2` to Interact with *MySQL/PostgreSQL.*
    - Executing SQL Queries: `SELECT`, `INSERT`, `UPDATE`, `DELETE`.
    - Transaction Management and Query Security with *SQLAlchemy.*
  - NoSQL Databases with Python
    - Introduction to NoSQL Databases.
    - MongoDB and Its Integration with Python via `pymongo`.
    - Creating, Updating, and Deleting Documents in *MongoDB.*
    - Advanced Queries and Indexing in *MongoDB.*
  - Database Connection and Integration in Python Applications
    - Connecting a Python Application to a Database.
    - Error Handling and Query Optimization.
    - Using ORM (Object Relational Mapping) with *SQLAlchemy.*
    - Storing and Retrieving Data via *REST API*s.

Part 4 – Structured, Modular, and Object-Oriented Programming

Overview:

In a world where software complexity continues to grow, writing merely functional programs is no longer enough—they must also be well-structured, modular, reusable, and scalable. Through this part, doctoral students will learn to structure and organize their programs effectively using functions, modules, packages, and libraries ensuring better readability and

maintainability. They will then explore the core concepts of Object-Oriented Programming (OOP) to design flexible and scalable programs. Special attention will be given to best development practices, particularly the application of design patterns to solve recurring design challenges in an elegant and efficient manner.

Objectives:

- Master structuring Python programs through the use of *functions*, *modules*, and *packages*.
- Develop expertise in OOP by handling key concepts of this paradigm to create robust and scalable programs.
- Design maintainable and scalable code by applying SOLID principles and best design patterns, ensuring a clean and efficient program architecture.

Prerequisites:

- **Part 1:** Introduction to Computer Programming
- **Part 2:** Basic Programming Concepts in Python

Detailed content:

- Introduction to Structured and Modular Programming
  - Principles of Structured and Modular Programming.
  - Definition and Usage of Functions in Python.
  - Code Organization with Modules, Packages, and Libraries.
  - Utilizing Standard and External Libraries (e.g., *matplotlib*, *Tkinter*, *scipy*).
- Object-Oriented Programming with Python
  - Principles of OOP.
  - Comparison with Traditional Programming.
  - Fundamental Concepts: *Classes*, *Objects*, and *Constructors.*
  - Instance, Class Attributes and (Static) Methods.
  - Encapsulation and Access Modifiers: *Public*, *Protected*, and *Private.*
  - Relationships and Interactions between Classes: *Inheritance*, *Association*, *Aggregation*, and *Composition.*
  - Polymorphism and Method Overloading: *Abstract Methods*, *Abstract Classes*, and Method *Overriding.*
  - *Class Diagrams*: Modeling and Visualization with *PlantUML.*

  - Practical Applications of OOP.
- Software Architecture and Design Patterns
  - Introduction to SOLID Principles.
  - Design Patterns: Definition, Importance, and Key Design Patterns (*Singleton*, *Strategy*, *Chain of Responsibility*, *Adapter*, *Composite*, and *Decorator*).
  - Case Studies and Practical Implementation.

Part 5 – Collaboration and Best Practices in Programming

Overview:

This part focuses on using debugging tools in Python to efficiently identify and fix errors in the source code. It also aims to familiarize doctoral students with collaborative development environments, facilitating optimal code management and teamwork.

Objectives:

- Develop coding expertise by applying best practices to improve programming skills and design high-quality programs.
- Master the use of error detection tools to optimize program development.
- Discover and apply various types of software testing, including unit testing, integration testing, end-to-end testing, and functional testing.
- Explore DevOps tools for source code management and version control.
- Use cloud platforms to store, share code, and promote collaborative work.

Prerequisites:

- **Part 1:** Introduction to Computer Programming
- **Part 2:** Basic Programming Concepts in Python
- **Part 3:** File and Database Handling
- **Part 4:** Structured, Modular, and Object-Oriented Programming

Detailed content:

- Debugging, Verification, and Optimization of Python Code
  - Principles and Definitions.
  - Debugging Code with `debugpy`.

  - Improving Code Quality with `flake8`.
  - Developing and Running Tests with `pytest`.
- Cloud Programming and Collaborative Development
  - Definition and Fundamental Concepts.
  - Cloud Development and Interactive Programming in Python with *Google Colab.*
  - Collaborative Coding Tools: *Codeshare*, *JupyterHub*, and *Visual Studio Live Share.*
- Collaboration and Progression with *GitHub*
  - Introduction to *Git* and *GitHub*: Principles, Features, and Usage.
  - GitHub Services : Collaboration, Code Review, etc.
  - Installing *Git* and Creating a *GitHub* Account.
  - Creating and Using a Repository.
  - Branch Management in a Version Control System.
  - Essential Git Commands: `clone`, `status`, `branch`, `merge`, `pull`, `push`, `commit`.
  - Discovering and Exploring Open-Source Projects.
  - Building a Professional Developer Portfolio.

10. Lecture Table

Objectives	Covered Topics	Modality	Competencies	WH
Part 1: Introduction to Computer Programming				
- Understand the concepts of computer programs and programming languages using a visual language (*Blockly*). - Learn fundamental concepts of computer programming using *Blockly*.	**1.** Declaration and manipulation of variables and basic data types **2.** Data input/output (I/O) operations: reading and writing functionalities **3.** Simple and nested conditional structures **4.** Loops in their various forms **5.** Translating visual code into textual code	Online	C18, C19, C20	2h
Part 2: Basic Programming Concepts in Python				
- Introduce the basic concepts of the Python language - Master the control structures of the language (conditional structures and loops) - Understand Python's data structures (simple and complex)	**1.** Introduction to the Python programming language: basic concepts and notions **2.** Simple data structures in Python **3.** Conditional structures and loops **4.** Complex data structures	Online	C17, C18, C19, C20	3h
Part 3: File and Database Handling				
- Learn the basics of file manipulation in different formats using Python. - Apply various operations on files in Python. - Master the use of *SQLite*, *MySQL*, and	**1.** Introduction to Files and Data Formats **2.** File Management and Advanced Operations **3.** Introduction to relational and NoSQL databases: Key Concepts, *SQL* vs.	Online	C17, C18, C19, C20	

PostgreSQL with Python. - Use Python libraries such as *sqlite3*, *SQLAlchemy*, *pymysql*, and *psycopg2* for managing relational databases. - Work with *NoSQL* databases, including *MongoDB* with *pymongo*.	*NoSQL* **4.** Database manipulation with Python: creation, querying, and management using *SQLite*, *MySQL*, *PostgreSQL*, and *MongoDB* **5.** Using Python libraries: *sqlite3*, *SQLAlchemy*, *pymysql*, *psycopg2*, *pymongo*			5h
Part 4: Structured, Modular, and Object-Oriented Programming				
- Understand and apply the principles of structural, modular, and object-oriented programming to improve code quality, maintainability, and reusability. - Enhance code maintainability using design patterns.	**1.** Decomposing programs into functions, modules, and packages **2.** Modeling real-world systems using classes and objects **3.** Identifying and implementing design patterns.	Online	C17, C18, C19, C20	3h
Part 5: Collaboration and Best Practices in Programming				
- Debug, verify, and improve the quality of the code - Explore tools for code management and version control - Master collaborative development via specialized platforms	**1.** Error detection and correction in code **2.** Improving code quality **3.** Creating and managing projects in a cloud environment **4.** Using collaborative development platforms	Online	C17, C18, C19, C20, C21	3h

11. Seminar Table

Related Parts	Objectives	Main Topic	Modality	Target Audience	Competencies	WH
All	- Present the learning objectives, topics, and assessment methods of the training program. - Become familiar with the available tools and resources.	Presentation of the training program: objectives, content, and evaluation process	Online	Instructors	C17, C18, C19, C20, C21	2h
All	- Understand the different families of programming languages and their fundamental characteristics. - Identify the specific application domains of each family based on needs and constraints. - Understand the impact of free and open-source software on software development (Case of Python). - Discover open-source tools used in programming (IDEs, frameworks, version control, etc.). - Promote engagement and collaboration within the open-source community.	Families of programming languages, their application domains, and their role in the development of free and open-source software.	Online	Doctoral Students	C17, C18, C19, C20, C21, C26	2h

12. Workshop Table

Objectives	Covered Topics	Modality	Competencies	WH
Part 1: Introduction to Computer Programming				
- Grasp the concept of conditional structures and loops through completing the 10 levels of *Blockly Maze*.	**1.** Simple and nested conditional structures **2.** Loops in their various forms	In-person	C18, C19, C20	1h 30
- Develop programs capable of interacting with the user - Master basic operations and operators in programming (arithmetic, logical, etc.) - Master elementary data types and some composite types	**1.** Reading and writing data **2.** Arithmetic and logical operators **3.** Primitive types: *integer, real, string, boolean* **4.** Composite types (particularly arrays)	In-person	C18, C19, C20	1h 30
Part 2: Basic Programming Concepts in Python				
- Present the basic concepts of the Python language - Understand the simple data structures in Python	Basic concepts and simple data structures in Python	In-person	C17, C18, C19, C20	2h
- Master control structures in the Python language (Conditional structures and loops)	Conditional structures and loops	In-person	C17, C18, C19, C20	1h30

- Understand complex data structures	Complex data structures	In-person	C17, C18, C19, C20	1h30
Part 3: File and Database Handling				
- Identify the necessary Python libraries for handling different file formats. - Master fundamental file operations. - Perform advanced file operations.	Advanced file handling and management in Python	In-person	C17, C18, C19	2h
- Learn to create, query, and manipulate SQL and NoSQL databases with Python. - Use the appropriate libraries for each type of database. - Integrate a database into a Python application.	**1.** Introduction to relational and NoSQL databases. **2.** Using *sqlite3*, *SQLAlchemy*, *pymysql*, *psycopg2*, *pymongo*. **3.** Basic and advanced SQL queries. **4.** Connecting a database to a Python application.	In-person		3h
Part 4: Structured, Modular, and Object-Oriented Programming				
- Write functions and modules. - Create charts with *Matplotlib* and simple graphical interfaces with *Tkinter*. - Use *SciPy* for scientific computations.	Introduction to structured and modular programming in Python	In-person	C17, C18, C19	2h
- Design modular and scalable programs by applying OOP to model real-world scenarios.				

- Generate and visualize class diagrams with *PlantUML*. - Utilize essential design patterns: *Singleton*, *Strategy*, *Chain of Responsibility*, *Adapter*, *Composite*, and *Decorator*.	Object-oriented programming and design patterns with Python	In-person	C17, C18, C19	4h
Part 5: Collaboration and Best Practices in Programming				
- Explore various debugging tools and code quality improvement techniques.	Development of high-quality Python code using *debugpy*, *flake8*, and *pytest*.	In-person	C17, C18, C19, C20, C21	1h 30
- Learn to develop in cloud environments that enhance sharing, version control, and collaborative teamwork.	Setting up and managing a project on GitHub.	In-person	C17, C18, C19, C20, C21	1h 30

13. Workload Summary Table

Part	Lectures	Workshops	Seminars	Subtotal
Part 1: Introduction to computer programming	2h	3h	2h	5h
Part 2: Basic programming concepts in Python	3h	5h		8h
Part 3: File and database handling	5h	5h		10h
Part 4: Structured, modular, and object-oriented programming	3h	6h		9h
Part 5: Collaboration and best practices in programming	3h	3h		6h
Total	16h	22h	2h	**40h**

14. Appendices

Appendix 1: Targeted Competencies

Competencies	Position in the Grid	Covered Parts
Develop new skills in programming and artificial intelligence.	C17	All
Strengthen computational thinking and logical reasoning to analyze and structure complex problems.	C18	All
Develop the ability to understand, design, and implement reliable and efficient algorithms and applications to solve research problems.	C19	All
Explore programming languages and leverage them for research problem-solving.	C20	All
Acquire skills in collaborative programming and effective teamwork.	C21	All
Adopt a spirit of sharing, collaboration, and ethics through the use of free and open-source software.	C26	All

Appendix 2: Distribution of Training Program Parts by Discipline (*M: Mandatory, O: Optional*)

Discipline	Part 1	Part 2	Part 3	Part 4	Part 5
Humanities and Social Sciences (HSS)	M	M	M	O	O
Science and Technology (ST)	O	O	M	M	M

Appendix 3: Mobilized Material Resources (by Institution) *

N.	Designation	Minimum Required Characteristics
1	Equipped training room	
2	PCs	Processor: I5, 6Gen, RAM: 8Go
3	Laptop (for instructors)	I5/I7, 10Gen, RAM: 16Go
4	Projector	Epson, WIFI port

5	Projection Screen	
6	WIFI Access Point	
7	Whiteboard	1.60x0.90
8	Laser Pointer	

*Appendix 3 illustrates the minimum hardware requirements per institution. Each university must determine its requirements based on the number of its doctoral students.

Appendix 4: Human Resources (by Institution)

N.	Designation	Main task(s)
1	Lead Instructors	Teaching Lectures
2	Assistant Instructors	Workshop Supervision
3	IT Engineers /Technicians	Maintenance and Installation

Appendix 5: Software Resources

Involved Parts	Software Tools	Source/Link
Part 1	*Blockly Labyrinthe*	https://blockly.games/maze?lang=fr
	Blockly demo	https://blockly-demo.appspot.com/static/demos/code/index.html
	Notepad ++	https://notepad-plus-plus.org/downloads/
	Sublime Text	https://www.sublimetext.com/
Parts 2-4	*Python & Python IDLE*	www.python.org
	Anaconda	www.anaconda.com
	PyCharm	https://www.jetbrains.com/pych arm/
	Jupyter	www.jupyter.org
	Spider	https://www.spyder-ide.org/
	PyTorch	https://pytorch.org/

Part 4	*PlantUML*	https://plugins.jetbrains.com/plugin/7017-plantuml-integration
Part 5	*Git*	https://desktop.github.com/download/
	debugpy	https://pypi.org/project/debugpy/
	flake8	https://pypi.org/project/flake8/
	pytest	https://pypi.org/project/pytest/

Appendix 6: Seminar Description Sheet

Title:

Families of programming languages, their application domains, and their role in the development of free and open-source software.

Overview:

This seminar explores the various families of programming languages and their application domains, with a particular focus on their role in the development of free and open-source software. It will introduce the main programming families and paradigms before examining their diverse application areas. A significant part of the seminar will be dedicated to introducing the fundamental concepts of free and open-source software, clarifying the distinctions between these two approaches, and analyzing the different licensing models. Emphasis will also be placed on the use of open-source tools in programming and software development. Finally, an interactive session will engage doctoral students in discussions on the perspectives and trends of open-source development, encouraging participation in this collaborative and innovative ecosystem.

Objectives:

- Explore the main families of programming languages.
- Analyze the application domains of programming languages.
- Develop doctoral students' ability to select the most suitable programming language for a project based on its specific requirements, application domain, and technical constraints.
- Understand the differences between free software and open-source software, as well as the associated types of licenses.
- Understand the impact of free and open-source software on programming and software development.

Detailed Content:

- Programming Language Families
  - Assembly Languages.
  - Functional Languages.
  - Imperative and Procedural Languages.
  - Object-Oriented Languages.
  - Logic-Based Languages.
  - Scripting Languages.
  - Declarative Languages.
  - Concurrent and Parallel Languages.
- Application Domains of Programming Languages
  - Web and Mobile Development.
  - Scientific Computing & High-Performance Computing.
  - System and Low-Level Programming.
  - Artificial Intelligence and Data Science.
  - Embedded Programming and IoT.
  - General-Purpose Programming.
- Free and Open-Source Software
  - Free Software vs. Open-Source Software.
  - License Types (GPL, MIT, Apache, etc.).
  - Benefits and Challenges of Open-Source Software and Tools.
- Programming with Open-Source Tools
  - Development Environments.
  - Version Control and Collaboration Tools.
  - Popular Open-Source Frameworks and Libraries.
  - Practical Cases (e.g., Python).

15. Bibliography

Part 1

1. Amber Lovett, *Coding With Blockly*, Cherry Lake Publishing, 2017

Part 2

2. Al Sweigart, *Automate the Boring Stuff with Python: Practical Programming for Total Beginners*, No Starch Press, 2nd edition, 2019.
3. David Beazley and Brian K. Jones, *Python Cookbook*, O'Reilly Media, 3rd edition, 2013.
4. Eric Matthes, *Python Crash Course: A Hands-On – Project-Based Introduction to Programming*, No Starch Press, 2nd edition, 2019.
5. Florian Dedov, *The Python Bible 7 in 1: Volumes One to Seven (Beginner, Intermediate, Data Science, Machine Learning, Finance, Neural Networks, Computer Vision)*, Self-published, 1st edition, 2020.
6. Luciano Ramalho, *Fluent Python: Clear, Concise, and Effective Programming*, O'Reilly Media, 2nd edition, 2022.
7. Mark Lutz and David Ascher, *Learning Python*, O'Reilly Media, 5th edition, 2013.
8. Sweigart, A. *Invent Your Own Computer Games with Python,* No Starch Press, 4th edition, 2016
9. ليزا تاغليفيري، البرمجة بلغة بايثون: تعلم البرمجة وكتابة البرامج وتلقيحها بلغة بايثون. أكاديمية حاسوب. ترجمة: محمد بغات، عبد اللطيف أيمش. 2020
10. متاح على موقع PDFهديل محمد الطاهر. تعلّم البايثون للمبتدئين (بالعربي). مطبوعة . Foulabook.com

Part 3

11. Severance C., *Python for Everybody: Exploring Data in Python,* CreateSpace Independent Publishing Platform, 2016. Available on : https://www.py4e.com/book
12. Barry P., Head *First Python: A Brain-Friendly Guide*, O'Reilly Media, 2nd edition, 2016
13. Panneerslvam R., *Databases and Python Programming: MySQL, MongoDB, OOP and Tkinter*, Amazon Digital Services LLC - Kdp, 2022
14. Albert Lukaszewski, *MySQL for Python*, Packt Publishing, 2010

Part 4

15. Jeff Maynard, *Modular programming*, Auerbach Publishers, 1972
16. Erik Westra, *Modular Programming with Python*, Packt Publishing, 2016

17. Martin Abadi and Luca Cardelli, *A Theory of Objects (Monographs in Computer Science)*, Springer, Corrected edition, 1996
18. Brett McLaughlin, Gary Pollice and David West, *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D*, O'Reilly Media, 1st edition, 2007
19. Steven F. Lott and Dusty Phillips, *Python Object-Oriented Programming: Build robust and maintainable object-oriented Python applications and libraries*, Packt Publishing, 4th edition, 2021
20. Mark Lutz, *Learning Python: Powerful Object-Oriented Programming*, O'Reilly Media; 6th edition, 2025
21. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides and Grady Booch, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1st edition, 1994
22. Eric Freeman and Elisabeth Robson, *Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software*, O'Reilly Media, 2nd edition, 2021
23. Kamon Ayeva and Sakis Kasampalis, *Mastering Python Design Patterns: Craft essential Python patterns by following core design principles*, Packt Publishing, 3rd edition, 2024

Part 5

24. Downey A. B., *Think Python: How to Think Like a Computer Scientist*, O'Reilly Media, 2nd edition, 2015. Available on: https://greenteapress.com/wp/think-python-2e/
25. Slatkin B., *Effective Python: 90 Specific Ways to Write Better Python*, Addison-Wesley Professional, 2nd edition, 2019. Available on: https://effectivepython.com/
26. Mariot Tsitoara, *Beginning Git and GitHub: Version Control, Project Management and Teamwork for the New Developer*, Apress, 2nd edition, 2024

16. Webography

Programming with Python

27. https://www.freecodecamp.org/news/learn-python-free-python-courses-for-beginners/
28. https://www.udemy.com/topic/python/free/?srsltid=AfmBOorjgRsKTP-Sep3UKHhr8rafQSo6enqqdtxloqzkJWcrkM-6N7RF
29. https://www.youtube.com/watch?v=rfscVS0vtbw
30. https://www.youtube.com/watch?v=qwAFL1597eM

31. https://www.youtube.com/watch?v=eWRfhZUzrAc
32. https://www.youtube.com/watch?v=K5KVEU3aaeQ
33. https://www.youtube.com/watch?v=pdsc9SVW-S8
34. https://www.youtube.com/watch?v=6i3e-j3wSf0

Modularity, Debugging, Verification, and Optimization

35. https://code.visualstudio.com/docs/python/debugging
36. https://flake8.pycqa.org/en/latest/
37. https://docs.pytest.org/en/stable/
38. https://docs.github.com/en/get-started/git-basics
39. https://docs.github.com/en/get-started