SYLLABUS

Programming Fundamentals

University year 2025 - 2026

1. Information regarding the programme

1.1. Higher education institution	Babeş-Bolyai University of Cluj-Napoca
1.2. Faculty	Faculty of Mathematics and Computer Science
1.3. Department	Department of Computer Science
1.4. Field of study	Computer Science
1.5. Study cycle	Bachelor
1.6. Study programme/Qualification	Computer Science
1.7. Form of education	Full time

2. Information regarding the discipline

2.1. Name of the discipline	Programming Fundamentals					Discipline code	**MLE5005**
2.2. Course coordinator					Assoc. Prof. Molnar Arthur-Jozsef		
2.3. Seminar coordinator					Assoc. Prof. Molnar Arthur-Jozsef		
2.4. Year of study	**1**	2.5. Semester	**1**	2.6. Type of evaluation	**E**	2.7. Discipline regime	**Compulsory**

3. Total estimated time (hours/semester of didactic activities)

3.1. Hours per week	**6**	of which: 3.2 course	**2**	3.3 seminar/laboratory/project	**4**
3.4. Total hours in the curriculum	**84**	of which: 3.5 course	**28**	3.6 seminar/laboratory/project	**56**

Time allotment for individual study (ID) and self-study activities (SA)	**hours**
Learning using manual, course support, bibliography, course notes (SA)	15
Additional documentation (in libraries, on electronic platforms, field documentation)	20
Preparation for seminars/labs, homework, papers, portfolios and essays	20
Tutorship	5
Evaluations	0
Other activities:	6
3.7. Total individual study hours	**66**
3.8. Total hours per semester	**150**
3.9. Number of ECTS credits	**6**

4. Prerequisites (if necessary)

4.1. curriculum	-
4.2. competencies	-

5. Conditions (if necessary)

5.1. for the course	Classroom with video-projector and Internet access
5.2. for the seminar /lab activities	Classroom with video-projector, Internet access and workstations configured for Python software development.

6.1. Specific competencies acquired [1]

Professional/essential competencies	• Advanced programming skills in high-level programming languages • Use of theoretical foundations of computer science as well as of formal models
Transversal competencies	• Use of efficient methods and techniques to learn, inform, research and develop the abilities to bring value to knowledge, to adapt at the requirements of a dynamical society and to communicate efficiently in the Romanian language and in an international language. • Application of organized and efficient work rules, of responsible attitudes towards the didactic-scientific field, to bring creative value to own potential, with respect for professional ethics principles and norms.

6.2. Learning outcomes

Knowledge	• The graduate has the necessary knowledge for using computers, developing software programs and applications, information processing. • The graduate has the ability to apply general rules to specific problems and produce relevant solutions.
Skills	• The graduate has the necessary skills to understand and use object-oriented programming concepts to develop software applications of medium complexity. • The graduate has the necessary skills for computer program design and software systems analysis.
Responsibility and autonomy:	• The graduate has the ability to observe and obtain information from various sources. • The graduate knows the basic aspects of software management.

7. Objectives of the discipline (outcome of the acquired competencies)

7.1 General objective of the discipline	• To know the basic concepts of software engineering (design, implementation and maintenance) and be able to apply them to create small and medium-sized console-driven applications using the Python programming language in the structured or object-oriented paradigm.
7.2 Specific objective of the discipline	• To know the key concepts of programming. • To know the basic concepts of software engineering related with the design, implementation and maintenance of software systems. • To gain understanding of basic software tools used in development and testing. • To learn the Python programming language and gain proficiency in running, testing, and debugging Python programs. • To acquire and improve their individual programming style.

[1] One can choose either competences or learning outcomes, or both. If only one option is chosen, the row related to the other option will be deleted, and the kept one will be numbered 6.

8. Content

8.1 Course	Teaching methods
Introduction to the course • What is programming: algorithm, program, basic elements of the Python language, the Python interpreter.	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration
Recursion. Algorithm complexity • Notion of recursion, direct and indirect recursion, examples. • Empiric analysis and asymptotic analysis. • Asymptotic notation: big-o, little-o, big-omega, little-omega, theta. • Comparison of algorithms from an efficiency point of view, analysing both time and extra-space complexity.	
Searching. Sorting • Specification of the searching and sorting problems. • Search methods: sequential, binary, exponential. • Sort methods: bubble sort, selection sort, insertion sort and optimizations, quick sort, merge sort and optimizations. • Algorithm and extra-space complexity of searching and sorting algorithms.	
Problem solving methods • General presentation of the greedy, divide & conquer, backtracking and dynamic programming methods. • Typical problem statements. • Analysis of time and extra-space complexity.	
Procedural programming • Compound types in Python: list, tuple, dictionary • Functions: definition, creating test cases, variable scope, calling, parameter passing • Test-driven development steps and refactoring.	
Modular programming • What is a module, what is a Python module, variable scope in a module, packages, standard module libraries, installing additional modules. • How to organize source code: responsibilities, single responsibility principle, separation of concerns, dependency, coupling, cohesion. • Introduction to layered architecture.	
Classes and objects • How to define additional data types in Python using classes • Class methods, special methods, object creation, Python scope and packages	
Object based programming • Information hiding, encapsulation, inheritance, dynamic typing • How to apply the principles of layered architecture using classes and objects • Using the exceptions mechanism • Examples focused on layered architecture solutions that demonstrate class hierarchies, the use of text and binary files for storage as well as the use of Python exceptions for signalling the occurrence of exceptional situations.	
Program design guidelines • Design principles: information expert, low coupling, high cohesion, single responsibility principle, dependency injection. • The Unified Modelling Language, UML class diagrams.	
Program testing and refactoring • Introduction to program testing, program testing levels. • Manual and automated testing using PyUnit. • Introduction to the Test-driven development paradigm. • Program inspection and refactoring	
Preparing for the exam • Revision of the most important topics covered by the course. • Exam guide presentation.	

Bibliography

1. Kent Beck - ***Test Driven Development: By Example***. Addison-Wesley Longman, 2002.
2. Kleinberg and Tardos – ***Algorithm Design***. Pearson Educational, 2014 (http://www.cs.princeton.edu/~wayne/kleinberg-tardos/)
3. Martin Fowler - ***Refactoring. Improving the Design of Existing Code***. Addison-Wesley, 1999.

(http://refactoring.com/catalog/index.html)
4. ***The Python language reference***. (https://docs.python.org/3/reference/index.html)
5. ***The Python standard library***. (https://docs.python.org/3/library/index.html)
6. ***The Python tutorial***. (https://docs.python.org/3/tutorial/index.html)

8.2 Seminar / laboratory	Teaching methods
1. Introduction to the course and the Python language	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration
2. Recursion. Algorithm complexity	
3. Searching. Sorting	
4. Problem solving methods: greedy, divide & conquer, backtracking, dynamic programming	
5. Procedural programming	
6. Modular programming	
7. Classes and objects	
8. Object based programming	
9. Program design. Layered architecture	
10. Preparing for the exam	

Bibliography

1. Kent Beck - ***Test Driven Development: By Example***. Addison-Wesley Longman, 2002.
2. Kleinberg and Tardos – ***Algorithm Design***. Pearson Educational, 2014 (http://www.cs.princeton.edu/~wayne/kleinberg-tardos/)
3. Martin Fowler - ***Refactoring. Improving the Design of Existing Code***. Addison-Wesley, 1999. (http://refactoring.com/catalog/index.html)
4. ***The Python language reference***. (https://docs.python.org/3/reference/index.html)
5. ***The Python standard library***. (https://docs.python.org/3/library/index.html)
6. ***The Python tutorial***. (https://docs.python.org/3/tutorial/index.html)

9. Corroborating the content of the discipline with the expectations of the epistemic community, professional associations and representative employers within the field of the program

- The course respects the IEEE and ACM Curricula Recommendations for Computer Science studies.
- The course exists in the study program of all major universities in Romania and abroad.
- The content of the course is considered by software companies as important for average programming skills.

10. Evaluation

Activity type	10.1 Evaluation criteria	10.2 Evaluation methods	10.3 Percentage of final grade
10.4 Course	The correctness and completeness of the accumulated knowledge and the capacity to design and implement correct Python programs.	Written exam	**30%**
10.5 Seminar/laboratory	Be able to design, implement, test and debug a Python program.	Practical evaluation	**30%**
	Correctness of delivered assignments and documentation.	Program and documentation portfolio	**40%**

10.6 Minimum standard of performance

- Students must observe the standards of academic integrity.
- Each student must prove that they acquired an acceptable level of knowledge and understanding of the core concepts taught in the class, that they can use knowledge in a coherent form, that they can establish certain connections and to use the knowledge in solving different problems in programming.
- Entering the examination during the regular or retake sessions is conditioned by having 10 attendances at the seminar (out of 14 possible) and 12 attendances at the laboratory (out of 14 possible)
- Entering the examination during the regular session is conditioned by having a minimum grade of 5 at the lab activity.
- Successfully passing the exam is conditioned by a minimum grade of 5 at the lab activity, practical test and written examination.

11. Labels ODD (Sustainable Development Goals)[2]

Not applicable.

Date:
28.04.2025

Signature of course coordinator

Assoc. Prof. PhD. Molnar Arthur-Jozsef

Signature of seminar coordinator

Assoc. Prof. PhD. Molnar Arthur-Jozsef

Date of approval:
...

Signature of the head of department

Assoc.prof.phd. Adrian STERCA

[2] Keep only the labels that, according to the *Procedure for applying ODD labels in the academic process*, suit the discipline and delete the others, including the general one for *Sustainable Development* – if not applicable. If no label describes the discipline, delete them all and write „*Not applicable.*".