SYLLABUS

1. 1. Information regarding the programme

1.1 Higher education institution	**Babeş-Bolyai University**
1.2 Faculty	**Faculty of Biology and Geology**
1.3 Department	**Department of Molecular Biology and Biotechnology**
1.4 Field of study	**Biology**
1.5 Study cycle	**Master**
1.6 Study programme / Qualification	**Bioinformatics applied in life sciences**

2. Information regarding the discipline

2.1 Name of the discipline (en) (ro)			**Algorithms and Programming** **Algoritmi şi Programare**				
2.2 Course coordinator			**Prof. Camelia Chira, PhD**				
2.3 Seminar coordinator			**Prof. Camelia Chira, PhD**				
2.4. Year of stud	**1**	2.5 Semester	**1**	2.6. Type of evaluation	**C**	2.7 Type of discipline	**Optional**
2.8 Code of the discipline		**MLE5115**					

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	2 sem 2 lab
3.4 Total hours in the curriculum	84	Of which: 3.5 course	28	3.6 seminar/laboratory	56

Time allotment:	hours
Learning using manual, course support, bibliography, course notes	14
Additional documentation (in libraries, on electronic platforms, field documentation)	12
Preparation for seminars/labs, homework, papers, portfolios and essays	14
Tutorship	8
Evaluations	18
Other activities:	

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	• Projector • Online communication platform
5.2. for the seminar /lab activities	• Computers, Python programming language and environment

6. Specific competencies acquired

Professional competencies	C1.1 Definition and description of programming paradigms and of language specific mechanisms, as well as identification of syntactic and semantic differences. C1.2 Description of existing software applications, on different levels of abstraction (architecture, classes, methods) using adequate basic knowledge. C1.3 Elaboration of adequate source code and testing of components in a well-known programming language, based on given specifications. C1.4 Testing applications based on testing plans. C1.5 Development of units of programs and corresponding documentation
Transversal competencies	TC1 Application of efficient and rigorous working rules, manifest responsible attitudes towards the scientific and didactic fields, underlying the individual potential and respecting professional and ethical principles. TC2 Use of efficient methods and techniques for learning, information, research and development of abilities for knowledge exploitation, for adapting to the needs of a dynamic society and for communication in a widely used foreign language.

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 to learn Python programming language
7.2 Specific objective of the discipline	• To know the key concepts of programming • To know the basic concepts of software engineering • To gain understanding of basic software tools used in development of programs • To learn Python programming language and tools to develop, run, test and debug programs • To acquire and improve a programming style according to the best practical recommendations

8. Content

8.1 Course	Teaching methods	Remarks
1. Introduction to software development processes • What is programming: algorithm, program, basic elements of the Python language, Python interpreter, basic roles in software engineering • How to write programs: problem statement, requirements, feature driven development process	• Interactive exposure • Explanation • Presentation • Practical examples	
2. Procedural programming • Compound types: list, tuple, dictionary		

• Functions: test cases, definition, variable scope, calling, parameter passing • Test-driven development (TDD), refactoring	• Interactive exposure • Explanation • Presentation • Practical examples	
3. Modular programming • What is a module: Python module definition, variable scope in a module, packages, standard module libraries, deployment • Eclipse + PyDev		
4. User defined types • How to define new data types: encapsulation, data hiding in Python, guidelines • Introduction to object-oriented programming • Exceptions		
5. Object-oriented programming • Abstract data types • Implementation of classes in Python • Objects and classes		
6. Software design guidelines • Layered architecture: UI layer, application layer, domain layer, infrastructure layer • How to organize source code: responsibilities, single responsibility principle, separation of concerns, dependency, coupling, cohesion		
7. Program testing and inspection • Testing methods: exhaustive testing, black box testing, white box testing • Automated testing • File operations in Python		
8. Recursion • Notion of recursion • Direct and indirect recursion • Examples • Computational complexity		
9. Search algorithms • Problem definition • Search methods: sequential, binary • Complexity of algorithms		
10. Sorting algorithms • Problem definition • Sort methods: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort • Complexity of algorithms		
11. Problem solving methods (I)		

• General presentation of the Backtracking, Divide & Conquer methods • Algorithms and complexity • Examples	• Interactive exposure • Explanation • Presentation • Practical examples	
12. Problem solving methods (II) • General presentation of the Greedy and Dynamic Programming methods • Algorithms and complexity • Examples		
13. Revision • Revision of most important topics covered by the course		
14. Evaluation		

Bibliography

1. M.L. Hetland, Beginning Python: From Novice to Professional, Apress, 2005.
2. M. Frentiu, H.F. Pop, Fundamentals of Programming, Cluj University Press, 2006.
3. K. Beck, Test Driven Development: By Example. Addison-Wesley Longman, 2002. http://en.wikipedia.org/wiki/Test-driven_development
4. M. Fowler, Refactoring. Improving the Design of Existing Code, Addison-Wesley, 1999. http://refactoring.com/catalog/index.html
5. The Python Programming Language - https://www.python.org/
6. The Python Standard Library - https://docs.python.org/3/library/index.html
7. The Python Tutorial - https://docs.python.org/3/tutorial/

8.2 Seminar / laboratory	Teaching methods	Remarks
1. Simple Python programs	• Interactive exposure • Explanation • Conversation • Didactical demonstration	
2. Procedural Programming		
3. Modular Programming		
4. Feature-driven software development		
5. Abstract data types		
6. Design principles		
7. Object-oriented programming		
8. Program design. Layered architecture		
9. Inspection and testing		
10. Recursion. Complexity of algorithms		
11. Search and sorting algorithms		
12. Problem solving methods: Backtracking		
13. Problem solving methods: Greedy		
14. Practical test		

Bibliography

1. M.L. Hetland, Beginning Python: From Novice to Professional, Apress, 2005.
2. M. Frentiu, H.F. Pop, Fundamentals of Programming, Cluj University Press, 2006.
3. K. Beck, Test Driven Development: By Example. Addison-Wesley Longman, 2002. http://en.wikipedia.org/wiki/Test-driven_development
4. M. Fowler, Refactoring. Improving the Design of Existing Code, Addison-Wesley, 1999. http://refactoring.com/catalog/index.html
5. The Python Programming Language - https://www.python.org/
6. The Python Standard Library - https://docs.python.org/3/library/index.html
7. The Python Tutorial - https://docs.python.org/3/tutorial/

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 studying program of all major universities in Romania and abroad.
- The content of the course is considered by the software companies as important for average programming skills.

10. Evaluation

Type of activity	10.1 Evaluation criteria	10.2 Evaluation methods	10.3 Share in the grade (%)
10.4 Course	The correctness and completeness of the accumulated knowledge and the capacity to design and implement correct Python programs	Written exam	40%
10.5 Seminar/lab activities	Be able to design, implement and test a Python program	Practical exam	30%
	Correctness of laboratory assignments and documentation delivered during the semester	Program and documentation	30%
10.6 Minimum performance standards			
A minimum grade of 5 should be obtained for the written exam, for the practical exam and for the final grade. In order to obtain the minimum grade 5, the student must demonstrate knowing the basic concepts of algorithms and programming.			

Date	**Signature of course coordinator**	**Signature of seminar coordinator**
16.01.2023	**Prof. Camelia Chira**	**Prof. Camelia Chira**

Date of approval

20.01.2023

Signature of the head of department
Prof. dr. Laura Dioşan