SYLLABUS

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 Departament	**Departament of Computer Science**
1.4 Field of study	**Computer Science**
1.5 Study Cycle	**Bachelor**
1.6 Study Cycle / Qualification	**Computer Science**

2. Information regarding the discipline

2.1 Name of the discipline	**Fundamentals of Programming**						
2.2 Course coordinator	**Assoc. Prof. PhD. Molnar Arthur**						
2.3 Seminar coordinator	**Assoc. Prof. PhD. Molnar Arthur**						
2.4 Year of study	**1**	2.5 Semester	**1**	2.6. Type of evaluation	**E**	2.7. Type of discipline	**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	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	Class room with projector
5.2 For the seminar/lab activities	• Laboratory with computers; • Python programming language and environment

6. Specific competencies acquired

Professional competencies	• C1.1 Description of programming paradigms and of language specific mechanisms, as well as identification of syntactic and semantic differences. • C1.2 Explanation 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 given programming language, based on given specifications. • C1.4 Testing applications based on testing plans. • C1.5 Developing units of programs and corresponding documentation.
Transversal competencies	• CT1 Application of efficient and rigorous working rules, manifest responsible attitudes towards the scientific and didactic fields, respecting professional and ethical principles. • CT2 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)
7.2 Specific objectives of the discipline	• To know the key concepts of programming • To know the basic concepts of software engineering (design, implementation and maintenance of software systems). • To gain understanding of basic software tools used in development and testing. • To learn Python programming language, and to get used to Python programming, running, testing, and debugging programs. • To acquire and improve their individual programming style.

8. Content

8.1 Lecture	Teaching methods	Remarks
1. Introduction • Introduction to the Python 3 programming language (basic data types, basic compound data types, dynamic typing, creating functions, repetitive and conditional statements) • Introduction to modern IDEs (creating a project, cloning a project from source control, debugging) • Introduction to git (git clone, commit and push operations)	• Explanation • Conversation • Examples • Didactical demonstration	
2. Recursion. Computational complexity • Empirical and asymptotic analysis • Asymptotic notation: big-O, omega, theta • Examples of orders of magnitude • Comparison of algorithms from an efficiency point of view • Memory-space complexity		
3. Searching. Sorting.		

• Specification of the searching and sorting problems • Search methods: sequential, binary, exponential • Sort methods: bubble sort, selection sort, insertion sort, quick sort, merge sort, Tim sort (discussion) • The computational and memory-space complexity of searching/sorting algorithms		
4. Problem solving methods (I) • General presentation of the greedy and backtracking methods • Algorithms and computational/memory-space complexity		
5. Problem solving methods (II) • General presentation of the divide & conquer and dynamic programming methods • Algorithms and computational/memory-space complexity		
6. Procedural programming • Functions: definition, specification, variable scope, calling, parameter passing, unit tests • Test-driven development (TDD) steps, introduction to refactoring		
7. Modular programming • Python module definition, variable scope in a module, packages, standard module libraries • How to organize source code: responsibilities, single responsibility principle, separation of concerns, dependency, coupling, cohesion • Common layers in an information system - logical architecture		
8. Classes and objects (I) • Defining new classes and instantiating objects • Special class methods and operators in Python 3 • Reasons for defining and using new data types		
9. Classes and objects (II) • Inheritance: generalization, code reuse, overriding • Working with Python exceptions and defining new exception types in the context of inheritance • Working with text and binary files using Python in the context of layered architecture and inheritance • UML class diagrams (elements, relationships, associations)		
10. Layered architecture • Implementing layered architecture solutions: the UI layer, application layer, domain layer • Principles in object-oriented design: the information expert, low coupling, high cohesion, protected variation, single responsibility, dependency injection		
11. Program testing, refactoring • Testing levels: unit testing, integration testing, system level testing • Testing methods: exhaustive, random, black and white box testing • Unit testing using support libraries • Establishing a coding style, refactoring		
12. Revision • 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	Teaching Methods	Remarks
1. Introduction to Python 3, modern IDEs and git	• Interactive exposure • Explanation • Conversation • Didactical demonstration	The seminar is structured as a weekly 2 hour class.
2. Recursion. Algorithm Complexity		
3. Searching. Sorting.		
4. Problem Solving Methods: greedy, backtracking		
5. Problem Solving Methods: divide & conquer, dynamic programming		
6. Procedural Programming		
7. Modular Programming (I)		
8. Modular Programming (II)		
9. Classes and objects (I)		
10. Classes and objects (II)		
11. Layered Architecture (I)		
12. Layered Architecture (II)		
13. Layered Architecture (III)		
14. Recap for the final 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) 15. ***The Python tutorial***. (https://docs.python.org/3/tutorial/index.html)		

8.3 Laboratory	Teaching Methods	Remarks
1. Introductory programs in Python 3	• Explanation • Conversation	• The lab is structured as
2. Algorithm complexity		
3. Sorting		

4. Problem solving methods		weekly 2 hour classes. • Laboratory assignments are due 1 week after assignment. The maximum delay for handing in laboratory work is 2 weeks.
5. Procedural programming		
6. Modular programming		
7. Laboratory test (I)		
8. Classes and objects		
9. Layered architecture (I)		
10. Layered architecture (II)		
11. Layered architecture (III)		
12. Assignment delivery time		
13. Laboratory test (II)		
14. Assignment delivery time		

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 studying program of all major universities in Romania and abroad.
The content of the course is considered 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 Lecture	The correctness and completeness of the accumulated knowledge and the capacity to design and implement correct Python programs	Written exam (during the regular session)	**30%**
10.5 Seminar/ Laboratory	Be able to design, test and debug a Python program	Practical evaluation (in the regular session)	**30%**
	Correctness and comprehension of delivered laboratory assignments and documentation	Program and documentation portfolio	**40%**

10.6 Minimum performance standards

- Students must observe the standards of academic integrity. Assignments must be completed individually, and to be graded, students must prove a good understanding of the source code and programming concepts used in their implementations.
- 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 this knowledge in a coherent form, that they can establish connections and use them when solving different programming problems.
- 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).

- Successfully passing the exam is conditioned by a minimum grade of 5 at the lab activity, practical test and written examination.

Date	Signature of course coordinator	Signature of seminar coordinator
	Assoc. Prof. PhD. Molnar Arthur	Assoc. Prof. PhD. Molnar Arthur

Date of approval	Signature of the head of department
	Assoc. Prof. PhD. Sterca Adrian