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 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 • Example: calculator, iteration modelling	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
2. Procedural programming • Compound types: list, tuple, dictionary • Functions: test cases, definition, variable scope, calling, parameter passing • Test-driven development (TDD) steps, refactoring	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	

3. Modular programming • What is a module: Python module definition, variable scope in a module, packages, standard module libraries, deployment • How to organize source code: responsibilities, single responsibility principle, separation of concerns, dependency, coupling, cohesion • Common layers in an information system - logical architecture • Eclipse + PyDev	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
4. User defined types • How to define new data types: encapsulation, information hiding (data hiding in Python), guidelines, abstract data types	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
5. Design guidelines • Problem statement: a program for managing information (CRUD operations) • Layered architecture: UI layer, application layer, domain layer, infrastructure layer • GRASP patterns • Example of application development: entity, validator, repository, controller • Principles: information expert, low coupling, high cohesion, protected variation, single responsibility, dependency injection	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
6. Object based programming • Objects and classes: classes, objects, fields, methods, special class methods (operator overloading), Python scope and namespace • UML Diagrams: class diagrams, relationships, associations, invariants • Inheritance: UML generalization, code reuse, overriding, inheritance in Python • Exceptions • Example: working with files in Python, repository implementations using files	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
7. Program design • Top down and bottom up strategies: top down design, bottom up design, bottom up programming style, mixed approach • Organizing the UI • Class invariants	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
8. Program testing and inspection • Testing methods: exhaustive testing, black box testing, white box testing • Testing levels: unit testing, integration testing • Automated testing, TDD • Program inspection: coding style, refactoring	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
9. Recursion • Notion of recursion • Direct and indirect recursion	• Interactive exposure • Explanation	

• Examples	• Conversation • Examples • Didactical demonstration	
10. Algorithm complexity • Empiric analysis and asymptotic analysis • Asymptotic notation: big-o, little-o, big-omega, little-omega, theta; properties • Examples of magnitude orders • Comparison of algorithms from an efficiency point of view • Structural complexity	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
11. Searching. Sorting. • Specification of the searching/sorting problem • Search methods: sequential, binary. • Sort methods: BubbleSort, SelectionSort, InsertionSort, QuickSort, MergeSort • Complexity of searching/sorting algorithms	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
12. Problem solving methods (I) • General presentation of the Greedy and Backtracking methods • Algorithms and complexity Examples	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
13. Problem solving methods (II) • General presentation of the Divide & Conquer and Dynamic Programming methods • Algorithms and complexity • Examples	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	
14. Revision • Revision of most important topics covered by the course • Exam guide	• Interactive exposure • Explanation • Conversation • Examples • Didactical demonstration	

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. Frentiu, M., H.F. Pop, Serban G. - ***Programming Fundamentals***, Cluj University Press, 2006
5. ***The Python language reference***. (https://docs.python.org/3/reference/index.html)
6. ***The Python standard library***. (https://docs.python.org/3/library/index.html)
7. ***The Python tutorial***. (https://docs.python.org/3/tutorial/index.html)

8.2 Seminar	Teaching Methods	Remarks

1. Introduction to Python. Simple problems 2. Procedural Programming 3. Modular Programming (I) 4. Modular Programming (II) 5. Object Based Programming 6. User Defined Types 7. Program Design (I). Layered Architecture 8. Program Design (II). Layered Architecture 9. Program Design (III). Inspection and Testing 10. Recursion. Algorithm Complexity 11. Searching. Sorting. 12. Program Design Recap 13. Problem Solving Methods: Greedy, Divide & Conquer 14. Problem Solving Methods: Backtracking, Dynamic Programming	• Interactive exposure • Explanation • Conversation • Didactical demonstration	The seminar is structured as a weekly 2 hour class.

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. Frentiu, M., H.F. Pop, Serban G. - ***Programming Fundamentals***, Cluj University Press, 2006
5. ***The Python language reference***. (https://docs.python.org/3/reference/index.html)
6. ***The Python standard library***. (https://docs.python.org/3/library/index.html)
7. ***The Python tutorial***. (https://docs.python.org/3/tutorial/index.html)

8.3 Laboratory	Teaching Methods	Remarks
1. Simple Python program 2. Feature-driven software development process (I) 3. Feature-driven software development process (II) 4. Feature-driven software development process (III) 5. Laboratory test 6. Layered architecture (I) 7. Layered architecture (II) 8. Layered architecture (III) 9. Text Files 10. Program Testing 11. Algorithm Complexity 12. Problem Solving Methods 13. Laboratory test – practical exam simulation 14. Assignment delivery time	• Explanation • Conversation	• The lab is structured as weekly 2 hour classes. • Laboratory assignments are due 1 week after assignment.

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. Frentiu, M., H.F. Pop, Serban G. - ***Programming Fundamentals***, Cluj University Press, 2006

5. ***The Python language reference***. (https://docs.python.org/3/reference/index.html)
6. ***The Python standard library***. (https://docs.python.org/3/library/index.html)
7. ***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 of delivered laboratory assignments and documentation	Program and documentation portfolio	**40%**

10.6 Minimum performance standards

- 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 have the ability to 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).
- 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