

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 Department	Department of Computer Science
1.4 Field of study	Computer Science
1.5 Study cycle	Bachelor
1.6 Study programme / Qualification	Computer Science

2. Information regarding the discipline

2.1 Name of the discipline		Fundamentals of Programming					
2.2 Course coordinator		Lect. PhD Găceanu Radu Dan					
2.3 Seminar coordinator		Lect. PhD Găceanu Radu Dan					
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 environment

6. Specific competencies acquired

Professional competencies	<p>C1.1 Description of programming paradigms and of language specific mechanisms, as well as identification of syntactic and semantic differences.</p> <p>C1.2 Explanation of existing software applications, on different levels of abstraction (architecture, packages, classes, methods) using adequate basic knowledge</p> <p>C1.3 Elaboration of adequate source codes and testing of components in a given programming language, based on some given specifications</p> <p>C1.4 Testing applications based on testing plans</p> <p>C1.5 Developing units of programs and corresponding documentations</p>
Transversal competencies	<p>CT1 Application of efficient and rigorous working rules, manifest responsible attitudes toward the scientific and didactic fields, respecting the professional and ethical principles.</p> <p>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 Romanian as well as in a widely used foreign language</p>

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

7.1 General objective of the discipline	<ul style="list-style-type: none"> To know the basic concepts of software engineering (design, implementation and maintenance of software systems) and to learn Python programming language
7.2 Specific objective of the discipline	<ul style="list-style-type: none"> To know the key concepts of programming To know the basic concepts of software engineering (design, implementation and maintenance of software systems) To understand the basic software tools To learn Python programming language, and to get used to Python programming, running, testing, and debugging programs. To acquire and improve the programming style.

8. Content

8.1 Course	Teaching methods	Remarks
<p>1. Introduction to software development processes</p> <ul style="list-style-type: none"> What is programming: Algorithm, Program, Basic Elements Of Python, Python Interpreter, Basic roles in software engineering How to write programs: Problem Statement, Requirements, Feature Driven Development Process Example: calculator, iteration modeling 	<ul style="list-style-type: none"> Interactive exposure Explanation Conversation Examples Didactical demonstration 	
<p>2. Procedural programming</p> <ul style="list-style-type: none"> Structured types: Lists, Tuples, Dictionaries What is a function: Test cases, Definition, Variable scope, Calls Passing parameters Anonymous functions development (TDD) steps, Refactorings 	<ul style="list-style-type: none"> Interactive exposure Explanation Conversation Examples Didactical demonstration 	
<p>3. Modular programming</p> <ul style="list-style-type: none"> What is a module: Python module definition, 	<ul style="list-style-type: none"> Interactive exposure Explanation 	

<p>variable scope in a module, packages, standard module libraries, deployment</p> <p>How to organize the source code: responsibilities, single responsibility principle, separation of concerns, dependency, coupling, cohesion</p> <p>Common layers in an information system logical architecture</p> <p>Eclipse+PyDev</p>	<ul style="list-style-type: none"> Conversation Examples Didactical demonstration 	
<p>4. User defined types</p> <p>How to define new data types: encapsulation, information hiding (data hiding in Python), guidelines, abstract data types</p>		
<p>5. Deployment principles</p> <ul style="list-style-type: none"> 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 	<ul style="list-style-type: none"> Interactive exposure Explanation Conversation Didactical demonstration 	
<p>6. Object based programming</p> <ul style="list-style-type: none"> 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 implementation using files 	<ul style="list-style-type: none"> Interactive exposure Explanation Conversation Didactical demonstration 	
<p>7. Program design</p> <ul style="list-style-type: none"> Top down and bottom up strategies: top down design, bottom up design, bottom up programming style, mixed approach Organizing the UI Class invariants 	<ul style="list-style-type: none"> Interactive exposure Explanation Conversation Didactical demonstration 	
<p>8. Program testing and inspection</p> <ul style="list-style-type: none"> Testing methods: exhaustive testing, black box testing, white box testing Testing levels: unit testing, integration testing Automated testing, TDD Program inspection: coding style, refactoring 	<ul style="list-style-type: none"> Interactive exposure Explanation Conversation Didactical demonstration 	
<p>9. Recursion</p> <ul style="list-style-type: none"> Notion of recursion Direct and indirect recursion Examples 	<ul style="list-style-type: none"> Interactive exposure Explanation Conversation Didactical demonstration 	

10. Algorithms complexity <ul style="list-style-type: none"> • 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 	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
11. Backtracking method <ul style="list-style-type: none"> • General presentation of the Backtracking method • Backtracking algorithm/subalgorithm and complexity • Extensions of the Backtracking method • Examples 	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
12. Division method <ul style="list-style-type: none"> • General presentation • Description of the subalgorithm • Examples Search algorithms and their complexity <ul style="list-style-type: none"> • specification of the search problem • search methods • sequential traversal • binary search • complexity of search algorithms 	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
13 Sort algorithms and their complexity <ul style="list-style-type: none"> • Specification of the sort problem • Sort methods: BubbleSort, SelectionSort, InsertionSort, QuickSort, MergeSort • Complexity of sort algorithms 	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
14. Revision	<ul style="list-style-type: none"> • Interactive exposure • Conversation 	
Bibliography		
<ol style="list-style-type: none"> 1. Kent Beck. <i>Test Driven Development: By Example</i>. Addison-Wesley Longman, 2002. See also Test-driven development. http://en.wikipedia.org/wiki/Test-driven_development 2. Martin Fowler. <i>Refactoring. Improving the Design of Existing Code</i>. Addison-Wesley, 1999. See also http://refactoring.com/catalog/index.html 3. Frentiu, M., H.F. Pop, Serban G., <i>Programming Fundamentals</i>, Cluj University Press, 2006 4. <i>The Python language reference</i>. http://docs.python.org/py3k/reference/index.html 5. <i>The Python standard library</i>. http://docs.python.org/py3k/library/index.html 6. <i>The Python tutorial</i>. http://docs.python.org/tutorial/index.html 		
8.2 Seminar	Teaching methods	Remarks
		The seminar is structured as 2 hours classes every week
1. Python programs	<ul style="list-style-type: none"> • Interactive exposure • Explanation 	

	<ul style="list-style-type: none"> • Conversation • Didactical demonstration 	
2. Procedural programming	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
3. Modular programming	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
4. User defined types	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
5. Deployment principles	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
6. Object based programming	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
7. Programs design	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
8. Program testing and inspection	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
9. Recursion. Algorithms complexity	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
10. Algorithms complexity	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
11. Backtracking	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical 	

	demonstration	
12. Division method. Search algorithms	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
13. Preparation for the practical test	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
14: Preparation for the written exam	<ul style="list-style-type: none"> • Interactive exposure • Explanation • Conversation • Didactical demonstration 	
8.3 Laboratory	Teaching methods	Remarks
		<ul style="list-style-type: none"> • The lab is structured as 2 hours classes every week. • The lab documents are due one week after the lab theme has been given and the lab programs are due two weeks later.
1. Simple Python program	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
2. Feature driven software development process	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
3. Feature driven software development process	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
4. Feature driven software development process	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
5. Layered architecture	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
6. Layered architecture	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
7. Layered architecture	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
8. Text files	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
9. Testing	<ul style="list-style-type: none"> • Lab assignment 	

	<ul style="list-style-type: none"> • Explanation • Conversation 	
10. Algorithms complexity	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
11. Backtracking method	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
12. Lab delivery time (see remark above)	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
13. Lab delivery time (see remark above)	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
14. Practical test simulation	<ul style="list-style-type: none"> • Lab assignment • Explanation • Conversation 	
Bibliography		
<ol style="list-style-type: none"> 1. Kent Beck. <i>Test Driven Development: By Example</i>. Addison-Wesley Longman, 2002. See also Test-driven development. http://en.wikipedia.org/wiki/Test-driven_development 2. Martin Fowler. <i>Refactoring. Improving the Design of Existing Code</i>. Addison-Wesley, 1999. See also http://refactoring.com/catalog/index.html 3. Frentiu, M., H.F. Pop, Serban G., <i>Programming Fundamentals</i>, Cluj University Press, 2006 4. <i>The Python language reference</i>. http://docs.python.org/py3k/reference/index.html 5. <i>The Python standard library</i>. http://docs.python.org/py3k/library/index.html 6. <i>The Python tutorial</i>. http://docs.python.org/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 Course	<ul style="list-style-type: none"> • The correctness and completeness of the accumulated knowledge and the capacity to design and implement correct Python programs 	Written exam (in the regular session)	40%
10.5 Seminar/Lab activities	<ul style="list-style-type: none"> • Be able to design, test and debug a Python program 	Practical evaluation (in the regular session)	30%
	<ul style="list-style-type: none"> • Correctness of Python programs and lab 	-documentation portfolio	30%

	documentations	-continuous observations	
10.6 Minimum performance standards			
<ul style="list-style-type: none"> Each student has to prove that (s)he acquired an acceptable level of knowledge and understanding of the, that (s)he is capable of stating these knowledge in a coherent form, that (s)he has the ability to establish certain connections and to use the knowledge in solving different problems in Python programming language. Successful passing of the exam is conditioned by a minimum grade of 5 at the lab activity, practical test and written exam. 			

Date

30.05.2015

Signature of course coordinator

Lect. PhD Radu Dan Găceanu

Signature of seminar coordinator

Lect. PhD Radu Dan Găceanu

Date of approval

Signature of the head of department

Prof. dr. Bazil Pârv