COURSE OUTLINE

(1) GENERAL

SCHOOL	Engineering		
ACADEMIC UNIT	Department of Computer Engineering and Informatics		
LEVEL OF STUDIES	Undergraduate		
COURSE CODE	**CEID_NY132**	SEMESTER	**Spring**
COURSE TITLE	Principles of Programming Languages and Compilers		

INDEPENDENT TEACHING ACTIVITIES *if credits are awarded for separate components of the course, e.g. lectures, laboratory exercises, etc. If the credits are awarded for the whole of the course, give the weekly teaching hours and the total credits*	WEEKLY TEACHING HOURS	CREDITS
Lectures, Tutorials, Laboratory	3 (L), 1 (T), 2 (L)	6
Add rows if necessary. The organisation of teaching and the teaching methods used are described in detail at (d).		6

COURSE TYPE *general background, special background, specialised general knowledge, skills development*	Specialised general knowledge
PREREQUISITE COURSES:	Recommended prerequisite knowledge: Good familiarity with the courses “Introduction to Computers and Programming” (NY131), “Object Oriented Programming” (NY134), “Introduction to Algorithms” (NY205), “Data Structures” (NY233), “Theory of Computing” (NY301) or equivalents.
LANGUAGE OF INSTRUCTION and EXAMINATIONS:	Greek. The course may be offered in English for Erasmus students.
IS THE COURSE OFFERED TO ERASMUS STUDENTS	Yes
COURSE WEBSITE (URL)	https://eclass.upatras.gr/courses/CEID1091/

(2) LEARNING OUTCOMES

Learning outcomes

The course learning outcomes, specific knowledge, skills and competences of an appropriate level, which the students will acquire with the successful completion of the course are described.

Consult Appendix A

- *Description of the level of learning outcomes for each qualifications cycle, according to the Qualifications Framework of the European Higher Education Area*
- *Descriptors for Levels 6, 7 & 8 of the European Qualifications Framework for Lifelong Learning and Appendix B*
- *Guidelines for writing Learning Outcomes*

Upon conclusion of the course, the students ought to be able to:

1. Understand the overall context and history of the development and use of different language programming paradigms.
2. Know the basic criteria for the efficient design and success of a programming language.
3. Know the functionalities of compilers and interpreters, their differences and the environments of their use.
4. Distinguish the different levels of the syntactic elements of programming languages.
5. Construct formal grammars that are part of formal languages, a theoretical model for describing the syntax of programming languages.
6. Implement lexical analyzers, both their theoretical description by deterministic finite automata

and regular expressions, as well as programming related tools such as “flex”.
7. Implement syntactic parsers, both their theoretical description by context-free grammars using the BNF notation, as well as programming related tools such as “bison”.
8. Know how the different programming languages implement and use the higher level syntactic elements such as variables, expressions, statements.
9. Define appropriate data types in their programs, exploiting the possibilities offered by the different programming languages.
10. Know the two basic scope rules (static and dynamic) and how they are implemented by the different programming languages.
11. Know the memory management mechanisms of an executing program.
12. Know the different types, their components, features and design challenges for using subprograms.
13. Use subprograms, utilizing the different ways of parameter passing supported by programming languages.
14. Know the basic features of diverse programming paradigms, namely functional and logical programming.

Upon conclusion of the course, the students are expected to have the following skills:

1. Have acquired an overall view on the structure, operation context and capabilities of modern programming languages.
2. Constructing formal grammars.
3. Implementing lexical analyzers, both their theoretical description by deterministic finite automata and regular expressions, as well as programming related tools such as “flex”.
4. Implementing syntactic parsers, both their theoretical description by context-free grammars using the BNF notation, as well as programming related tools such as “bison”.
5. Define appropriate data types in their programs, exploiting the possibilities offered by different programming languages.
6. Using subprograms, utilizing the different ways of parameter passing supported by programming languages.

General Competences

Taking into consideration the general competences that the degree-holder must acquire (as these appear in the Diploma Supplement and appear below), at which of the following does the course aim?

Search for, analysis and synthesis of data and information, with the use of the necessary technology
Adapting to new situations
Decision-making
Working independently
Team work
Working in an international environment
Working in an interdisciplinary environment
Production of new research ideas
Project planning and management
Respect for difference and multiculturalism
Respect for the natural environment
Showing social, professional and ethical responsibility and sensitivity to gender issues
Criticism and self-criticism
Production of free, creative and inductive thinking
……
Others…
…….

- Search for, analysis and synthesis of data and information, with the use of the necessary technology
- Adapting to new situations
- Decision-making
- Production of free, creative and inductive thinking

(3) SYLLABUS

1. Introduction to programming languages
2. Evolution of programming languages
3. Compilers and interpreters
4. Programming languages syntax
5. Lexical analysis

6. Syntactic analysis (parsing)
7. Variables, expressions, statements
8. Data types
9. Scope and memory management
10. Subprograms
11. Functional programming
12. Logic programming

(4) TEACHING and LEARNING METHODS - EVALUATION

DELIVERY *Face-to-face, Distance learning, etc.*	Ex cathedra.
USE OF INFORMATION AND COMMUNICATIONS TECHNOLOGY *Use of ICT in teaching, laboratory education, communication with students*	The slides of the course and additional supplementary material are freely available from the course's website. Communication with students is done through a dedicated e-forum.
TEACHING METHODS *The manner and methods of teaching are described in detail.* *Lectures, seminars, laboratory practice, fieldwork, study and analysis of bibliography, tutorials, placements, clinical practice, art workshop, interactive teaching, educational visits, project, essay writing, artistic creativity, etc.* *The student's study hours for each learning activity are given as well as the hours of non-directed study according to the principles of the ECTS*	

Activity	*Semester workload*
Lectures	3x13 = 39
Tutorials (exercises)	1x13 = 13
Laboratory exercises	2x13 = 26
Individual study, preparation and problem solving	3x13 = 39
Weekend study	2x13 = 26
Study during the 3 “empty weeks” (2 weeks of vacation and 1 week of exam preparation)	4x3 = 12
Course total	***155***

STUDENT PERFORMANCE EVALUATION *Description of the evaluation procedure* *Language of evaluation, methods of evaluation, summative or conclusive, multiple choice questionnaires, short-answer questions, open-ended questions, problem solving, written work, essay/report, oral examination, public presentation, laboratory work, clinical examination of patient, art interpretation, other* *Specifically-defined evaluation criteria are given, and if and where they are accessible to students.*	• The language of instruction and examination is Greek. Special provisions (lecture notes and examinations in English) can be made for foreign students. • The final grade is based 100% on performance on the final examination. The evaluation criteria are posted on the course website. • The final examination is written, of graded difficulty, and may include: developing and solving complex problems, questions for short answers, judgment questions. • During the course, students are also assigned a project of theoretical and (mostly) programming nature, aiming at familiarizing them with the theoretical models for the description of formal grammars, as well as with the “flex” and “bison” programming environments used for the development of lexical and syntactic analysers (parsers) respectively.

(5) ATTACHED BIBLIOGRAPHY

- Suggested bibliography:

- Michael L. Scott, "*Programming Language Pragmatics*", Second Edition, 2006, Elsevier
- Robert W. Sebesta, "*Concepts of Programming Languages*", Eleventh Edition, 2016, Pearson

- Related academic journals:

This is an introductory course, hence there is no systematic use of articles from the scientific literature.