Description of Course Unit

Master in Informatics and Computing Engineering
Programming Fundamentals
Instance: 2019/2020
—

Institutional page

General information

Course Unit: Programming Fundamentals
Acronym: FPRO
Course: MIEIC
Curricular Years: 1º
Official website: Sigarra
Credits ECTS: 6
Contact hours: 70
Total Time: 162
Instance: 2019/2020
Semestre: 1S
Lectures (2x1,5h): João Correia Lopes (JCL)
Recitations (8x2h): João Correia Lopes (JCL), Rui Camacho (RCS), Ricardo Cruz (RPMC), Fernando Cassola Marques (FJCM)
Attendance (2x1h): António Cadilha, Telmo Baptista

Teaching language

Portuguese.

Objectives

1. BACKGROUND

Fluency in the process of software development is a basic prerequisite to the work of Informatics Engineers. In order to use computers to solve problems effectively, students must be competent at reading and writing programs using higher-order programming languages.

2. SPECIFIC AIMS

The global aim of this Unit is to give the student the ability to create algorithms, and to use a programming language to implement, test, and debug algorithms for solving simple problems.

The student will be able to understand and use the fundamental programming constructs, and the functional approach to programming, specifically effect-free programming where function calls have no side-effects and variables are immutable, and contrast it with the Imperative approach.

3. PERCENT DISTRIBUTION

Scientific component: 40%

Technological component: 60%

Learning outcomes and competences

At the end of the course, the student is expected to handle programming problems of medium complexity, using the imperative or functional programming approaches or paradigms.

More specifically, the student will be able to:

1. Design, implement, test and debug a program that uses the fundamental programming constructs, basic computations, standard conditional and iterative structures, simple Input/Output, persistence and exceptions.
2. Understand Data Abstraction and use simple and aggregate data types.
3. Understand Procedural Abstraction and use the definition of functions, parameter passing, recursion.
4. Write useful functions that take and return other functions.
5. Implement basic algorithms that avoid assigning to mutable state or considering reference equality.
6. Correctly reason about variables and lexical scope in a program using function closures.
7. Define and use iterators and other operations on aggregates, including operations that take functions as arguments (especially map, reduce/fold and filter).
8. Use the programming tools that help writing, testing and documenting computer programs according to the programming best practices.

Working method

Presential.

Pre-requirements (prior knowledge) and co-requirements (common knowledge)

No previous knowledge is required.

Programme

- Introduction to Computational Thinking, algorithms and programming with Python.
- Fundamental programming concepts: simple data types; variables, expressions and statements; program flow, conditionals, iteration; functions, parameter passing, recursion; persistence; functions, parameter passing and recursion
- Working with aggregate data types: strings, tuples, sets, lists, dictionaries.
- Effect-free programming by using function calls without side effects and immutable variables.
- First-class functions and closures.
- Utility higher-order functions: map, reduce/fold and filter.
- Generators and list comprehensions.
- Problem-solving strategies.
- Programming tools, debugging, exceptions.

Mandatory literature

- Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, *How to Think Like a Computer Scientist — Learning with Python 3*, Release 3rd Edition, 2019 [PDF] [HTML] [Library]

Complementary bibliography

- Steven F. Lott, Building Skills in Python - A Programmer's Introduction to Python, FreeTechBooks, 2010 HTML
- Allen Downey, Think Python — How to Think Like a Computer Scientist, 2nd Edition, Version 2.2.23, Green Tea Press, 2015 HTML PDF code
- David Mertz, Functional Programming in Python, O'Reilly Media, 2015
- Ernesto Costa, Programação em Python – Fundamentos e Resolução de Problemas. FCA – Editora de Informática, 2015. ISBN: 978-972-722-816-4 [Library]

Teaching methods and learning activities

The continuous enrolment of the student in the course is promoted, through the study and discussion of the course topics, previously distributed in Jupyter Notebooks, both in lectures and recitation classes and with in-class and away programming assignments.

The student is motivated to find the best ideas to solve specific problems, execute them and implement the programming solutions, in an elegant, legible and efficient (time and space) mode using the Python programming language.

Automatic correction tools are used to increase the efficiency of the feedback given to the students.

Lecture classes (T) are used to present and discuss the topics of the program, using a computer connected to a multimedia projector.

Recitation classes (TP) are used to help students understand the topics of the program and solve the weekly programming assignments.

In-class and away programming assignments are given on a weekly basis, to improve the regular and effective development of autonomous learning processes and are tested and graded using Moodle assignments.

In-class assignments are used for summative evaluation at the end of each lecture class using Moodle quizzes.

Students are encouraged to use a Web App (Play) containing programming exercises selected by theme: exercises (easy) to perform before the theoretical class, exercises (of average difficulty) to perform before the weekly assignments (RE) and exercises (difficult) to perform before the practical tests in computer (PE).

In the classroom, the students use the same working environment (IDE Spyder, Pylint, Web App (Test) and submissions of weekly assignments) that is used later in the individual assessments (PE).

Whenever deemed necessary in the classroom, especially during the first month of classes, students are encouraged to go to the "Clinic", supervised by student assistants, to get help.

Software

- Anaconda Distribution 2019.03 for Linux (https://www.anaconda.com)
- Spyder3 (included in Anaconda)
- Python 3.7 (included in Anaconda)

Keywords

Physical sciences > Computer science > Programming

Type of assessment

Distributed evaluation without final exam.

Assessment Components

Designation	Weight (%)
Test	80
Practical or project work	20
Total:	100

Amount of time allocated to each course unit

Designation	Time (hours)
Autonomous study	92
Class frequency	70
Total:	162

Eligibility for exams

Students are eligible for the final theory evaluation (TE), if they do not exceed the absences limit (25% of the total number of estimated recitation classes) and if they obtain a minimum of 40% at least in one of the two last practical on computer evaluations (MAX(PE04, PE05) >= 40%).

Calculation formula of final grade

The evaluation will be based on the following components:

LE = **L**ecture in-class **e**valuation: Multiple-choice questions about programming concepts, to be answered on an individual basis in the theoretical classroom (the best 20 from a total of 26 are selected) *[5 questions, 5 minutes]*
RE = **R**ecitation and away weekly **e**valuation: Weekly recitation classes and away programming assignments (the best 10 out of 13 are selected) *[5 questions, NA]*
PE = **P**ractical on computer **e**valuation: Individual programming assignments in Moodle (the best 3 from a total of 5 are selected) *[5 questions, 105 minutes]*
TE = **T**heory **e**valuation: Multiple-choice questions about programming concepts, to be answered on an individual basis in Moodle with the consultation of one book *[50 questions, 105 minutes]*

Final classification = 10% LE + 10% RE + 50% PE + 30% TE

Observations:

1. A minimum classification of 40% in the component TE is required.
2. A minimum mark of 40% is required for at least one of the last two practical on computer evaluation

(PE04 or PE05).
3. There will be another TE (TE02) for students, eligible for the assessment, that miss the evaluation with proper justification, or students that did not obtain the minimum required in the component; in the later case, the grade is limited to 50%.
4. If the final classification is greater than 17, the student may be submitted to an oral examination and the final grade is the average of both grades.
5. Students which achieve at least 90% in the in-class and weekly programming assignments (LE + RE), by the middle of the course, may opt to retain the grade and do a small project to achieve the maximum grade (100%).

Examinations

The evaluation of this course uses three types of examinations:

1. In-class and away programming assignments related with the current topics, to be submitted in Moodle and automatically graded.
2. Practical on computer evaluation of a set of small computer programs, using Moodle and automatically graded, with consultation of given languages reference texts.
3. Theory evaluation with one book consultation using a multiple-choice questionnaire in Moodle to assess students ability to use and discuss the best algorithms and data structures for specific problems.

Classification improvement

In the following academic year, the student can request improvement of classification that includes the practical evaluation in computer PE04 or PE05 and the theoretical evaluation TE01 or TE02, with consultation of only one book, the same as for the regular students.

Final classification = 60% PE + 40% TE

Observations:

1. A minimum classification of 40% of each component, PE and TE, is required.
2. If the final classification is greater than 17, the student may be submitted to an oral examination and the final grade is the average of both grades.

Special evaluation (TE, DA, ...)

In the case of students who do not attend classes and go without the weekly evaluation (LE and RE), the final classification will be the average of the classification of two evaluation components: Practical on computer evaluation (PE) and Theory evaluation (TE), as of the regular students.

Final classification = 60% PE + 40% TE

Observations:

1. A minimum classification of 40% of each component, PE and TE, is required.
2. A minimum mark of 40% is required for at least one of the last two practical on computer evaluation (PE04 or PE05)
3. There will be another TE (TE02) for students, eligible for the evaluation, that miss the examination with proper justification, or students that did not obtain the minimum required in the component. In the later case, the grade is limited to 50%.
4. If the final classification is greater than 17, the student may be submitted to an oral examination and the

final grade is the average of both grades.

Observations

No individual component grades (LE[1)], RE, PE, TE[2)]) can be reused in another enrolment in the course.

— FPRO, 2019/2020

1)
LE = Read the book (and understand) + See good programming examples
2)
REPETE = Repeat work with programs and exercises given or found elsewhere