COURSE DESCRIPTION CARD - SYLLABUS

Course name
Introduction to programming

Course

Field of study	Year/Semester
Artificial intelligence	1/1
Area of study (specialization)	Profile of study
-	general academic
Level of study	Course offered in
First-cycle studies	english
Form of study	Requirements
full-time	compulsory

Number of hours

Lecture	Laboratory classes	Other (e.g. online)
30	30	0
Tutorials	Projects/seminars	
0	0	

Number of credit points

5

Lecturers

Responsible for the course/lecturer:	Responsible for the course/lecturer:
Maciej Antczak, Ph.D.,D.Sc.	Tomasz Zok, Ph.D.
email: maciej.antczak@put.poznan.pl	email: tomasz.zok@put.poznan.pl
tel. +48 61 665 3056	tel. +48 61 665 3040
Faculty of Computing and Telecommunications	Faculty of Computing and Telecommunications
ul. Piotrowo 2, 60-965 Poznan	ul. Piotrowo 2, 60-965 Poznan

Prerequisites

The student starting this module should have basic knowledge and skills in mathematics and computer science, but programming experience is welcome, but not required. He/She should be able to edit a text file with your favorite Linux / Windows editor and run the script file from the command line. He/She should also show attitudes such as honesty, persistence, creativity, and respect for other people. After all, he/she should be able to obtain information from the indicated sources, often in English.

Course objective

The course aims to introduce students with basic programming concepts in Python. In particular, this includes:

1. Provide students the principal knowledge about the Python language mechanisms allowing them to develop programs in both structural and object-oriented manner.

2. Providing students the basic knowledge about specialized data structures available in Python and developing the skills of their practical use.

3. Developing students' skills in algorithmization of relatively simple problems and their implementation.

4. Developing students' ability to create their scripts in Python based on the given specification.

5. Acquiring by students the ability to predict possible execution errors and to secure programs in such a way to avoid them.

6. Provide students with basic knowledge about unit testing of Python programs.

Course-related learning outcomes

Knowledge

As a result of the conducted course, the student:

1. Has a practical knowledge of structured and object-oriented programming in Python.

2. Knows and is able to use in practice specialized data structures available in Python.

3. Knows and understands the basic techniques, methods, algorithms, and tools used in the process of solving IT tasks.

4. Has an organized, theoretically founded basic knowledge of computer science key areas such as algorithmics, programming languages and paradigms, operating systems, and software engineering.

Skills

As a result of the conducted course, the student:

1. Has principal IT skills in the field of computational complexity analysis of algorithms, programming in Python, and the use of operating systems.

2. Can formulate and solve relatively simple problems in the field of computer science, with particular emphasis on artificial intelligence, using appropriately selected methods.

3. Can easily adapt the existing ones and formulate and implement new algorithms using at least one of the available tools.

4. Can plan and organize work in the implementation of engineering tasks, both individually and in a team.

Social competences

As a result of the conducted course, the student:

1. Understands that knowledge and skills quickly become outdated in computer science, with particular emphasis on artificial intelligence, and perceives the need for constant additional training and raising one's qualifications.

2. Is aware of the importance of scientific knowledge and research related to computer science and artificial intelligence in solving practical problems that are essential for the functioning of individuals, companies, organizations, and the entire society.

3. Can function and cooperate in a group, playing various roles in it, and is able to properly define priorities for the implementation of a task set by himself or others.

Methods for verifying learning outcomes and assessment criteria

Learning outcomes presented above are verified as follows:

The knowledge acquired during the lecture is verified by a 60-minute test carried out during the 15th lecture, which the students solve on their own. The test consists of about 20 questions (closed, multiple-choice, equally marked). Passing threshold: 50% of points.

The skills acquired during the laboratory classes are verified using two 60-minute tests at the computer that are solved individually. During each test, exactly one Python program is implemented, transforming the given input into the expected output, both written in text format. In addition, students improve their programming skills by (a) solving individually a diverse set of predefined tasks provided on the HackerRank platform, and (b) developing on their own the project on the CodinGame platform.

All programs submitted by students are automatically compared with each other to identify plagiarism, which is forbidden. Identified plagiarism causes discarding of each task affected by it for all engaged students.

Based on the aforementioned activities, each student achieves one weighted final grade, normalized using a percentage scale. Passing threshold: 50% of points.

Students who achieved very good results (> 90%) from the laboratory classes are exempted from the final test conducted during the last lecture.

Programme content

The lecture program covers the following topics:

Lecture 1: Presentation of crucial information about the course, discussion of basic terms & concepts associated with programming, characteristics of the Python language, how to start the adventure with programming in Python?

Lecture 2: Declaring variables, keywords, basic instructions (e.g., print, assignment statements), defining simple expressions (using operators, operands, literals, values) and evaluating them, verifying value types, converting selected types, loading user input, calling functions, identifying and removing errors in the developed software.

Lecture 3: Modules (import), namespaces, random numbers generation, introduction into object-oriented programming, the introduction of for loops, basic sequential data structures (e.g., strings, lists, tuples), index and slice operators, basic functions operating on the analyzed data structures (e.g., len, count, index, split, join), concatenation and repetition of lists.

Lecture 4: Overview of the accumulator pattern usually used when visiting successive elements of the sequential data structure, iterating overvalues or using element indices, simple nesting of for loops and tracking changes in the values of variables used in them, iterable vs. iterable variable, logical values and expressions, arithmetic and logical operators, in, not in, simple, nested and chained conditional statements, mutable vs. immutable objects, delete list items.

Lecture 5: Transforming Sequential Data Structures: objects vs. references, object aliasing, list cloning, an overview of basic built-in methods for lists and strings with examples, append vs. concatenate, examples of string formatting, modifying the content of the list while iterating through it. Processing text files: basic file manipulation methods, identification of files in the filesystem, reading a file line by line, writing new data into the file, using the with statement while processing files, CSV files processing.

Lecture 6: Dictionaries: basic properties and manipulation methods on them, copying vs. aliasing, various iteration scenarios operating on dictionaries. Functions: what are functions and why they are useful in practice?, Discussing the key aspects while declaring the function, passing parameters into and returning the result(s) from functions.

Lecture 7: Understanding locality of parameters and variables declared in a function, various scenarios of using global variables in a function, rules governing functional decomposition, when to use print vs. return statement in the function body, mutable objects as parameters passed to the function, overview of the basic flow of running a set of instructions / function calls. Tuples packing / unpacking: various variants of assigning, enumerating elements in a sequential data structure, returning tuples, and passing them as arguments to a function call.

Lecture 8: Discussion of the while statement with examples (e.g., infinite loop), presentation of a listener loop example (waiting for user input) and a sentinel value, emphasis on the need to validate user input, examples of using break and continue statements in a loop , an overview of optional parameters usage when calling a function along with the potential pitfals, the use of a variable name with its assigned value when calling a function, an overview of anonymous functions using lambda expressions, sort method vs. sorted function, example scenarios for sorting the dictionary keys, tuples, the use of several conditionals within a single sorting task.

Lecture 9: Storing complex objects (e.g., lists, tuples, dictionaries, functions) in lists, nesting dictionaries, processing JSON files, examples of using nested loops, shallow vs. deep copying of lists, guidelines for appropriate processing of nested data, discussion of sophisticated mechanisms for transforming sequential data structures (e.g., map, filter, zip), introducing the concept of an exception and discussing a set of standard exceptions, extending the code execution process with exceptional situations, discussing examples of usage the basic try/except statement.

Lecture 10: A paradigm of object-oriented programming: reviewing the previously discussed terms, how to define non-standard classes, considering parameters within constructors, defining custom methods within classes, passing objects as parameters into methods and functions, converting the instance into a string, returning newly created class instances by its methods, sorting a list of custom class objects, object variables vs. class variables, characteristics of simple, multiple and multi-level inheritance and polymorphism.

Lecture 11: Strategies for developing successful programs, assertions: verification of (a) the expected result type of an expression, (b) execution of a specific path within a complex conditional statement, (c) the occurrence of edge values during iterating in a loop, (d) all possible scenarios of the function execution, (e) the correctness of modifying a mutable object within the called function, (f) the values of optional parameters were determined during the function call, (g) the expected states of the objects. Unit testing: introducing the concept of a test case and a test suite, why is it worth writing unit tests & how it should be done, introducing tools used for this purpose, e.g., unittest and pytest, presenting a commonly used assertions, testing exceptional situations. Discussion of the concept of incremental, test-driven development (TDD), manual vs. automatic & unit vs. integration testing, __import__ instruction, static code analysis.

Lecture 12: Using alternative string formatters, e.g., % or f-string, creating and printing docstring comments, initializing the seed of a pseudorandom generator, discussing selected functions allowing to randomly select a single elements or sublists from the input list and shuffling it in-place, validating identifiers, e.g., keywords or variable names, discussion of reserved identifier classes, examples of the use of isinstance statement and selected math module functions, proper processing of decimal (ensuring the expected precision) and rational numbers, other built-in functions for string manipulation, bitwise and ternary operators, operator overloading, mimicking a non-existent switch statement, walrus operator usage example, more complex manipulation of namespaces, using else or pass statement after or inside a loop, advancing the parameter passing mechanism using the '/' and '*' operators, discussing try/except/finally and raise statements.

Lecture 13: Presentation of other useful built-in functions, the introduction of the following concepts: iterator, generator, decorator, and set with examples of their use, presentation of specialized data structures and iterators provided in the collections and itertools module, respectively, discussion of using regular expressions and properties, defining a set of practices ensuring high-quality and readability of the code, characteristics of Python Virtual Environment and Jupiter notebooks.

Lecture 14: Introduction mechanisms allowing for the date and time manipulation, serialization, multithreading, and multiprocessing.

Lecture 15: Final test from the lecture.

Laboratory classes are conducted in the form of fifteen two-hour sessions taking place in the computer laboratory. The first classes are intended to familiarize students with the rules of using the laboratory and completing the classes. Programming exercises are carried out by students individually. The program of laboratory classes covers the following topics:

1. Introduction into working in Linux - basic instructions, pipeline processing, redirection of input & output streams.

2. Familiarize with the Python interpreter, its help system, and documentation being a part of the programming environment.

3. Implementation of simple programs aimed at learning the data types and control structures of the language based on the interactive Python programming course provided by Runestone Academy [5].

4. Improving Python programming skills by individually solving as many tasks as possible with different levels of difficulty, available on the HackerRank platform.

5. Independent development of a program solving the problem published on the CodinGame platform.

6. Testing of implemented programs and preparation of a set of unit tests for the software produced on the CodinGame platform.

Teaching methods

1. Lecture: slide show presentations on different aspects of programming in Python, illustrated with additional examples presented on the board if needed.

2. Laboratory classes: practical exercises at the computer carried out according to a specific scenario, implementation of code fragments and scripts solving relatively simple algorithmic problems, discussion of applied solutions, and programming constructs.

Bibliography

Basic

1. Automate the boring stuff with Python. Author: Sweigart, Albert. No Starch Press; 1st Edition (April 14, 2015).

2. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. Author: Matthes, Eric. No Starch Press; Illustrated Edition (May 3, 2019).

3. Begin to Code with Python. Author: Miles, Rob S. Microsoft Press; 1st Edition (December 8, 2017).

4. The Quick Python Book. Author: Ceder, Naomi R. Manning Publications; 2nd Edition (January 15, 2010).

5. Foundations of Python Programming provided by Runestone Academy (https://runestone.academy/runestone/books/published/fopp/index.html).

Additional

6. Get Programming: Learn to code with Python. Author: Bell, Ana. Manning Publications; 1st Edition (April 19, 2018).

7. Learn Python 3 the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code. Author: Shaw, Zed. Addison-Wesley Professional; 1st Edition (June 27, 2017).

8. Python Tutorial for Beginners. How to Quickly Learn Python? provided by Data Flair (https://data-flair.training/blogs/python-tutorial/).

Breakdown of average student's workload

	Hours	ECTS
Total workload	125	5,0
Classes requiring direct contact with the teacher	60	2,4
Student's own work (literature studies, preparation for laboratory classes, solving by yourself all exercises that had not solved during the laboratories, preparation for tests/exam) [1]	65	2,6

[1] delete or add other activities as appropriate