GHENT UNIVERSITY

Course Specifications

Valid as from the academic year 2020-2021

Informatics (O000096)

Due to Covid 19, the education and evaluation methods may vary from the information displayed in the schedules and course details. Any changes will be communicated on Ufora.

Course size (nominal values; actual values may depend on programme)

Credits	Study time	Contact hrs
10.0	300 h	120.0 h

Course offerings and teaching methods in academic year 2020-2021

A (year)	English	null	lecture	44.0 h
			seminar: practical PC room classes	72.0 h
			demonstration	4.0 h

Lecturers in academic year 2020-2021

De Neve, Wesley	KR01	lecturer-in-charge

Offered in the following programmes in 2020-2021	crdts	offering
Bachelor of Science in Environmental Technology	10	A
Bachelor of Science in Food Technology	10	A
Bachelor of Science in Molecular Biotechnology	10	A
Joint Section Bachelor of Science in Environmental Technology, Food Technology and Molecular Biotechnology	10	A

Teaching languages

English

Keywords

Command line, Computational thinking, Creative problem solving, Programming, Python, SQL, UNIX

Position of the course

Scientists and engineers are often confronted with time-consuming and repetitive tasks when having to process and analyse data, namely collecting information from websites, converting files from one format to another, and analysing, summarizing, and visualizing the information obtained. In addition, the exponential flow of newly incoming information requires present-day scientists and engineers to be able to automate these tasks, in order to speed up their daily job routines.

This course teaches students how to describe time-consuming and repetitive tasks in such a way that they can be performed automatically by a (network-based) computer system. To that end, students will acquire the necessary skills for computer-based creative problem solving through learning to work and think in (1) Python, a popular programming language, and (2) UNIX, the workhorse operating system of science and engineering. The computer problems that need to be solved are taken from different scientific disciplines, including mathematics, biology, chemistry, physics, and computer science.

To take part in this course, students do not need to have any prior programming experience. However, to be successful for this course, students need to have an aptitude for mathematics and logic. In addition, given that this course follows a '*learning by doing*' and a '*learning from mistakes*' approach, students need to have a willingness to solve computer problems on a regular basis.

Contents

Programming is the process of designing, writing, testing, debugging, and maintaining the source code of computer programs. This requires knowledge of the syntax and the semantics of a programming language and the ability to write programs in that language. Additionally, and maybe most importantly, when writing computer programs,

one must learn how to think as a programmer. This process of computational thinking (that is, learning how to solve problems by programming) is a common theme throughout the whole course.

In this course, students learn how to make use of the Python programming language to solve a plethora of problems. To that end, attention is paid to:

- basic components: instructions, variables, data types, and operators;
- control structures: conditional statements, repetitive statements, and functions;
- data structures: strings, lists, tuples, dictionaries, sets, modules, and files;
- text files: reading, processing, and writing data; and
- object-oriented programming: objects, classes, attributes, methods, encapsulation, polymorphism, and inheritance.

Furthermore, in this course, students learn how to make use of UNIX-based tools to automate repetitive or complex tasks. To that end, attention is paid to:

- the principles of UNIX-based operating systems;
- consulting technical documentation;
- running a remote interactive session (VPN, SSH);
- file systems;
- interactive command line usage;
- text file formats (HTML, XML, CSV, FASTA);
- filters, redirection, and pipes;
- regular expressions;
- interactive text editing;
- automated text editing using edit commands; and
- the basics of shell scripting.

Finally, in this course, students learn how to make use of the Structured Query Language (SQL) to communicate with a relational database.

Initial competences

An aptitude for mathematics and logic.

An interest in solving (scientific) problems.

Some basic computer knowledge is advantageous (prior programming skills are not required).

Final competences

1 Translate a task described in natural language into a program written in Python.
2 Execute a program written in Python by means of a computer, generating a correct result.
3 Test and debug a program written in Python.
4 Make the right choices between different alternatives when writing a program in Python, taking into account performance (efficiency), coding style, and correctness.
5 Demonstrate a working knowledge about the basic principles of object-oriented programming.
6 Automate repetitive and complex tasks by means of UNIX-based tools.
7 Work interactively and non-interactively with operating systems, computer networks, file systems, and text editors.
8 Understand the structure of various text file formats, including HTML, XML, CSV, and FASTA.
9 Apply SQL to communicate with a relational database.
10 Resolve error messages through critical usage of technical documentation and online resources.

Conditions for credit contract

Access to this course unit via a credit contract is determined after successful competences assessment

Conditions for exam contract

This course unit cannot be taken via an exam contract

Teaching methods

Demonstration, lecture, seminar: practical PC room classes

Learning materials and price

Handbook: William F. Punch, Richard Enbody (2017). The Practice of Computing using Python. Third Edition. Addison Wesley, ISBN-13: 978-0134379760. About $130.

Handbook: Mark G. Sobell (2012). A Practical Guide to Linux: Commands, Editors, and Shell Programming. Fourth Edition. Prentice Hall, ISBN-13: 978-0134774602. About $40.

Slides shown during the lectures will be made available on Ufora (in English), together with additional learning materials (e.g., background information and links to relevant websites).

Free digital tools like Eclipse and PyCharm for writing and debugging Python source code, the Online Python Tutor for visualizing code execution, the Dodona online platform for automated verification of the correctness of solutions written in Python, and a remote UNIX (Ubuntu) environment (helios).

Students are required to have a personal laptop for use in this course.

References

Mark Lutz (2009). Learning Python: Powerful Object-Oriented Programming. Fourth Edition. O'Reilly Media, ISBN-13: 978-0596158064.

Mark Pilgrim (2009). Dive into Python. CreateSpace, ISBN-13: 978-1441413024. Free download @ http://diveintopython.org.

Hans Peter Langtangen (2009). A Primer on Scientific Programming with Python. Springer, ISBN-13: 978-3642024740.

Tony Gaddis (2009). Starting Out with Python. Pearson Education - Addison Wesley, ISBN-13: 978-0321549419.

Michael H. Goldwasser (2007). Object-Oriented Programming in Python. Prentice Hall, ISBN-13: 978-0136150312.

Jason Kinser (2008). Python for Bioinformatics. Jones & Bartlett Publishers, ISBN-13: 978-0763751869.

Sebastian Bassi (2009). Python for Bioinformatics. Chapman & Hall, ISBN-13: 978-1584889298.

Mark G. Sobell (2012). A Practical Guide to Linux: Commands, Editors, and Shell Programming. Fourth Edition. Prentice Hall, ISBN-13: 978-0134774602.

William F. Punch, Richard Enbody (2017). The Practice of Computing using Python. Third Edition. Addison Wesley, ISBN-13: ISBN-13: 978-0133085044.

Steven Haddock and Casey Dunn (2010). Practical Computing for Biologists. First Edition. Sinauer Associates, Inc, ISBN-13: 978-0878933914.

Ashley Shade, Tracy K. Teal (2015). Computing Workflows for Biologists: A Roadmap. PLOS Biology.

Pavel A. Pevzner (2004). Educating Biologists in the 21st Century: Bioinformatics Scientists versus Bioinformatics Technicians. Bioinformatics, Vol. 20, No. 14, pages 2159–2161.

Alejandra J. Magana, Manaz Taleyarkhan, Daniela Rivera Alvarado, Michael Kane, John Springer, and Kari Clase (2014). A Survey of Scholarly Literature Describing the Field of Bioinformatics Education and Bioinformatics Educational Research. CBE—Life Sciences Education, Vol. 13, pages 607–623.

Course content-related study coaching

The syntax and the semantics of the programming language Python, the database management language SQL, and selected UNIX tools are presented in the course handbooks and in the course slides, and need to be acquired largely through (guided) self-study.

Solutions for selected computer exercises are discussed during the theory lectures so that students learn how computational skills can be applied in practice.

During the weekly supervised hands-on sessions, students themselves learn how to tackle computational challenges by working on a series of mandatory computer exercises that need to be solved independently. These computer exercises aim at bringing the theory into practice.

Dodona, a digital learning environment, gives students instant feedback on their solutions submitted for the Python programming challenges, containing additional exercises for further practicing.

After each deadline, example solutions for all exercises are made available on Ufora.

Information about the calculation of the different evaluation marks is communicated during the theory lectures at the beginning and near the end of the first-term and the second-term teaching activities.

Example examinations of previous years are made available on Ufora near the end of the first-term and the second-term teaching activities.

Announcements on Ufora are used for counselling, giving feedback, and providing background information.

Through individual appointments scheduled via email, the lecturer and the teaching assistants are available for answering questions about the course in general (grading, examination), the theory, and the exercises.

Evaluation methods

end-of-term evaluation and continuous assessment

Examination methods in case of periodic evaluation during the first examination period

Open book examination, skills test

Examination methods in case of periodic evaluation during the second examination period

Open book examination, skills test

Examination methods in case of permanent evaluation

Assignment

Possibilities of retake in case of permanent evaluation

examination during the second examination period is not possible

Calculation of the examination mark

For **the first-term examination period**, the mark of the periodic evaluation (end-of-term assessment) accounts for 75% of the partial examination mark for Term 1 and the mark of the non-periodic evaluation (hands-on sessions; continuous assessment) accounts for 25% of the partial examination mark for Term 1. To qualify for passing, both the mark of the periodic and the non-periodic evaluation should be at least equal to 8/20 (40%). If that is not the case, then the partial examination mark for Term 1, as obtained during the first-term examination period, will be subject to an upper limit of 7/20. No second partial examination opportunity is administered during the first-term resit examination period.

If the partial examination mark for Term 1, as obtained during the first-term examination period, is higher than or equal to 10/20, then one partial examination is used during the second-term examination period, only covering the course content of Term 2, with this partial examination consisting of both a periodic and non-periodic evaluation.

If the partial examination mark for Term 1, as obtained during the first-term examination period, is lower than 10/20, then two partial examinations are used during the second-term examination period: a first partial examination covering the content of Term 1, only consisting of a periodic evaluation, and a second partial examination covering the content of Term 2, consisting of both a periodic and non-periodic evaluation.

Students who obtained a partial examination mark for Term 1 higher than or equal to 10/20 during the first-term examination period, may decide to retake the partial examination covering the content of Term 1 during the second-term examination period, with this partial examination only consisting of a periodic evaluation. When doing so, the mark obtained for the latter periodic evaluation is used for calculating the partial examination mark for Term 1 (that is, the mark obtained for the periodic evaluation that took place during the first-term examination period is discarded).

For **the second-term examination period**, the mark of the periodic evaluation accounts for 75% of the partial examination mark for Term 2 and the mark of the non-periodic evaluation accounts for 25% of the partial examination mark for Term 2. To qualify for passing, both the mark of the periodic and the non-periodic evaluation should be at least equal to 8/20 (40%). If that is not the case, then the partial examination mark

for Term 2, as obtained during the second-term examination period, will be subject to an upper limit of 7/20.

During Term 2, the non-periodic evaluation for Term 1 cannot be retaken. Therefore, the partial examination mark for Term 1 during the second-term examination period is calculated twice. For the first calculation, the mark of the non-periodic evaluation, as obtained during Term 1, accounts for 25% of the partial examination mark and the mark of the periodic evaluation, as obtained for the Term 1 content during the second-term examination period, accounts for the remaining 75% of the partial examination mark. For the second calculation, the partial examination mark is equal to the mark of the periodic evaluation, as obtained for the Term 1 content during the second-term examination period (that is, the mark of the non-periodic evaluation, as obtained during Term 1, is not taken into account). The partial examination mark for Term 1 during the second-term examination period is then equal to the maximum of the above two calculations. Note that the mark of the periodic evaluation, as obtained for the Term 1 content during the second-term examination period, should be at least equal to 8/20 (40%). If that is not the case, then the partial examination mark for Term 1, as obtained during the second-term examination period, will be subject to an upper limit of 7/20.

The final examination mark is the average of the partial examination mark for Term 1 and the partial examination mark for Term 2. To qualify for passing (that is, to have a final examination mark that is higher than or equal to 10/20), both partial examination marks should be higher than or equal to 8/20 (40%).

During **the second-term resit examination period**, a third partial examination opportunity is organized that covers the Term 1 content, as well as a second partial examination opportunity that covers the Term 2 content, with both partial examination opportunities only consisting of a periodic evaluation. Given that the non-periodic evaluation for Term 1 and the non-periodic evaluation for Term 2 cannot be retaken during the second-term resit examination period, both the partial examination mark for Term 1 and the partial examination mark for Term 2 are calculated twice, following the approach and the conditions used for calculating the partial examination mark for Term 1 during the second-term examination period. Furthermore, the final examination mark is again the average of the partial examination mark for Term 1 and the partial examination mark for Term 2, with both partial examination marks again having to be higher than or equal to 8/20 (40%) in order to qualify for passing.

Marks for partial examinations can never be transferred to the second-term resit examination period.

Throughout the yearlong Informatics course, the non-periodic evaluation is organized as follows. During Term 1 of the yearlong Informatics course, students are asked to tackle 48 Python coding challenges (36 basic and 12 advanced), and during Term 2, students are asked to tackle 12 Python coding challenges (8 basic and 4 advanced), 2 SQL problem statements (1 basic and 1 advanced), and 16 UNIX problem statements (11 basic and 5 advanced). Basic exercises come with a low to intermediate complexity, whereas advanced exercises are more challenging in nature (examination-level complexity). Binary grading (correct or not correct) is used for the Python programming challenges, whereas partial grading is used for the SQL and the UNIX exercises. During grading, advanced exercises get twice the weight of basic exercises. The rounding approach for non-periodic evaluation marks is based on halves.

Students who eschew period aligned and/or non-period aligned evaluations for this course unit may be failed by the examiner.