1. Teachers

Author, associate professor: Stepan Kuznetsov, National Research University Higher School of Economics, Department of Computer Science.

2. Scope of Use

The present program establishes minimum demands of students' knowledge and skills, and determines contents of the course.
The present syllabus is aimed at department teaching the course, their teaching assistants, and students of the Master of Science program 010402.68 «Applied Mathematics and Infomatics».
This syllabus meets the standards required by:

- educational standards of National Research University Higher School of Economics;
- educational program «Data Sciences» of Federal Master's Degree Program 010402.68, 2016;
- University curriculum of Master's program in «Data Science» (010402.68) for 2016.

Summary

This course includes the basics of mathematical logic, graph theory, combinatorics, and formal language theory. The emphasis is put upon the algorithmic side: mathematical results act as a support for effecient algorithms operating on graphs, strings, and, finally, parsing algorithms for regular expressions and context-free grammars. The course is actually twofold: besides usual «chalk-and-blackboard» mathematical part, it also includes a practical one, i.e., implementing the algorithms discussed in the course. The students are supposed and encouraged to (but not restricted to) use the Python language. For the last part, parsing algorithms, PyBison is also welcome.

3. Learning Objectives

The learning objective of this course is to make the students firmly understand and use the following notions:

- classical propositional logic (in the Hilbert-style and/or resolution calculus)
- classical predicate logic
- graphs and algorithms on them (DFS, BFS, Dijkstra's algorithm etc)
- algorithms on strings: pattern-matching, maximal common substring, etc
- regular expressions, effective lexical analysis
- context-free grammars and parsing algorithms for them (CYK, LR)

4. Learning Outcomes

After completing this course the student should:

- know classical propositional and predicate calculi;
- know the notion of graph and basic algorithms on graphs;
- be able to implement algorithms on graphs and strings;
- be able to formalize the structure of a given formal language using regular expressions and context-free grammars;
- be able to use automated parser generators to utilize the aforemented grammars.

5. Place of the Discipline in the Master's Program Structure

The course «Discrete Mathematics for Algorithm and Software Design» is an adaptation course taught in the first year of the Master's program «Data Science». It is recommended for all students of the program who didn't take in-depth courses in mathematical logic, discrete mathematics and formal language theory in their previous education and/or didn't have enough practice in implementing algorithms mentioned above and constructing parsers for formal languages.

Prerequisites

It is supposed that the students enrolled in this course are capable of programming in Python or another programming language and are fluent in English. No other special knowledge is required.

After completing this course, the student should have the following competences:

Competence	Code	Code (UC)	Descriptors (indicators of achievement of the result)	Educative forms and methods aimed at generation and development of the competence
The ability to reflect developed methods of activity	C-1	SC-M1	The student is able to reflect developed methods in discrete mathematics and mathematical logic, and also the programming tradition.	Lectures and tutorials.
The ability to propose a model to invent and test methods and tools of professional activity	C-2	SC-M2	The student is able to model real-life objects using notions of mathematical logic and discrete mathematics.	Examples discussed during the lectures and tutorials. Assignments
Capability of development of new research methods, change of scientific and industrial profile of self-activities	C-3	SC-M3	Students are capable of implementing theoretical results (algorithms) as programs in an appropriate programming environment, that can be used in practice.	Tutorials, practical assignments, extra reading suggested.

6. Schedule

Each week this course occupies 2 academic hours for lecture followed by 2 academic hour for practical tutorial (in a computer class).

№	Topic	Total hours	Contact hours		Self-study
			Lectures	Practicum	
1	Classical propositional logic	8	2	2	4
2	Classical predicate logic	8	2	2	4
3	Basic notions of graph theory	12	2	2	8
4	Algorithms of graphs	20	2	2	16
5	Algorithms on strings	16	2	2	12
6	Regular expressions	16	2	2	12
7	Context-free grammars and parsing algorithms	28	4	4	20
Total:		**108**	**16**	**16**	**76**

7. Requirements and Grading

- **Test:** an optional test to withdraw from the subject. Students with sufficient prior knowledge in discrete mathematics and mathematical logic and good programming skills are not required to attend the course.
- **Homework:** 3 assignments during the semester: one written, two programming.
- **Exam:** written exam. Preparation time — 180 min.

8. Assessment

The *assessment* consists of three homeworks, handed to the students during the course. The students are supposed to do exersises based on lecture topics, and also implement the discussed algorithms. The *final assessment* is the final exam. It will consist of 7 problems graded equally.

Grade formula: the *final course mark* is obtained from the following formula:

Final = 0.5 * (Homeworks) + 0.5 * (Exam)

All grades with the fractional part strictly greater then 0.5 are rounded up. Rounding when the fractional part less or equal to 0.5 is determined by the lecturer/examiner.

Table of Grade Accordance

Ten-point Grading Scale	Five-point Grading Scale	Binary Grading Scale
1 — very bad 2 — bad 3 — no pass	Unsatisfactory — 2	FAIL
4 — pass 5 — highly pass	Satisfactory — 3	PASS
6 — good 7 — very good	Good — 4	
8 — almost excellent 9 — excellent 10 — perfect	Excellent — 5	

9. Course Description

Topic 1. Classical Propositional Logic
Propositional formulae. Classical (Boolean) interpretation of propositional formulae. Tautologies and satisfiable formulae. Classical propositional calculus (Hilbert-style derivations). Resolution method.

Topic 2. Classical Predicate Logic
Predicate formulae; quantifiers. Interpretations (models) of predicate logic. Predicate calculus. Completeness and undecidability of classical predicate logic (without proof). Proof search and proof assistants (overview).

Topic 3. Basic Notions of Graph Theory
Definition of graph. Special types of graphs. Trees. Using graphs for modelling networks etc. Isomorphism of graphs. Euler graphs.

Topic 4. Algorithms on Graphs
BFS. DFS. Dijkstra's algorithm. Checking whether a graph is a Euler graph.

For advanced students: the notion of NP-hardness. NP-hardness of 3-SAT and Hamiltonian path problems.

Topic 5. Algorithms on Strings

Topic 6. Regular Expressions
The notion of regular expression. Regular expressions and finite automata (Kleene's theorem): sketch of proof, examples. Algorithms for matching with regular expressions. Implementation of regular expressions (e.g., in Python).

Topic 7. Context-Free Grammars and Parsing Algorithms

Context-free grammars and context-free languages. Parsing with CFG: CYK, LR (overview). The limits of CFG (pumping lemma). Practical parsing using CFG: the Bison/YACC parser generator.

10. Term Educational Technology

The following educational technologies are supposed to be used in the study process:

- lectures;
- discussion and analysis of the results during tutorials;
- regular assignments to train the students and monitor their progress;
- posting solutions of the exercises and related materials on the course webpage;
- consultations and code review (mainly online, by e-mail).

11. Recommendations for Course Lecturer

The lecturer is advised to make the course interactive. The lectures, usually based on beamer slide presentations and/or writing on board, are expected to turn into group discussions sometimes. The practical part should indeed include one-to-one discussions with all the students involved (if there are many students, teaching assistants are required). The course is declared as adaptive, however, it is normal to provide advanced learners with more complicated and interesting tasks.

12. Recommendations for Students

The course is planned to me as interactive as possible. Students are encouraged to ask questions and actively participate in the classes. The lecturer is ready to answer questions online via e-mail, in particular, questions concerning the programming part, including in-depth code review and comments. The course is taught in English, and students can ask the lecturer and teaching assistants to help them with the language.

13. Final Exam Questions

The final exam will consist of 7 questions, one per each section of the course. The questions will be weighted equally. No material is allowed for the exam.

14. Materials

Recommended Reading

- C. C. Leary, L. Kristiansen. A friendly introduction to mathematical logic (2nd edition). Milne Library, SUNY Geneseo, NY, 2015.
- J. E. Hopcroft, R. Motwani, J. D. Ullman. Introduction to automata theory, languages, and computation (3rd edition). Pearson, 2007
- Python reference manual: https://docs.python.org/2/reference/
- T. Niemann. Lex & Yacc tutorial: http://epaperpress.com/lexandyacc/download/LexAndYaccTutorial.pdf

Supplementary Reading

- I. Chiswell, W. Hodges. Mathematical logic. Oxford University Press, 2007.
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. Introduction to Algorithms (3rd edition), MIT Press, 2009.
- A. V. Aho, M. S. Lam, R. Sethi. J. D. Ullman. Compilers: principles, techniques, and tools (2nd edition). Addison-Wesley, 2006.

Course materials will be made available through the author's webpage, http://www.mi.ras.ru/~sk/
Students will be provided with lecture notes, problem sheets and examples with solutions, home assignments and additional readings.

15. Equipment

For the lectures, a laptop with a projector is required, along with a blackboard (whiteboard). The practical part requires a computer class with a working Python development environment.

The course structure and this syllabus are prepared by Stepan Kuznetsov.