Программа учебной дисциплины «Основы и методология программирования»

Утверждена
Академическим советом ООП
Протокол № от «____»_____ 20____ г.

Авторы	Т. В. Вознесенская, С. А. Шершаков
Число кредитов	8
Контактная работа (час.)	120
Самостоятельная работа (час.)	184
Курс	1
Формат изучения дисциплины	без использования онлайн-курса

1 Course Description

1.1 Title of a Course

Introduction to Programming.

1.2 Pre-requisites

1.2.1 Part I

As a matter of fact, good English and Math are usually enough to enter the program, so no preliminary knowledge in data science or specific programming skills are required, because some of the students may have zero experience in these.

1.2.2 Part II

Successful completion of the first part of the course is the sole prerequisite for being enrolled for the second part.

1.3 Course Type

Compulsory.

1.4 Abstract

The training course “Introduction to Programming” is offered to students of Bachelor Program “HSE and University of London Double Degree Programme in Data Science and Business Analytics” (area code *01.03.02*) at the Faculty of Computer Science of the National Research University — Higher School of Economics (HSE). The course is classified as an *compulsory subject* (*Б.Пр.Б* unit / base module, *Б.Пр* – Major disciplines of 2018–2019 academic year working curriculum); it is a two-module course (semester A quartiles 1 and 2).

The course is divided into two logical parts, which do not basically depend on each other. *The first part* is given during semester A quartile 1 under responsibility of Dr. Tamara Voznesenskaya. *The second part* is given during semester A quartile 2 under responsibility of Lect. Sergey Shershakov.

The syllabus is prepared for teachers responsible for the course (closely related disciplines), teaching assistants, students enrolled in the course as well as experts and statutory bodies carrying out assigned or regular accreditations.

1.4.1 Part I

The first part of the course is intended to be taught during the first module (quartile) of the program, so it starts from the very beginning and takes into account, that some of the students may have zero experience in programming.

The lectures and practical classes are closely inter-related. The lectures are primarily intended to introduce new topics, whereas the practical classes are intended for solving specific problems by coding programs in Python.

1.4.2 Part II

The *second part* of the course is intended to be taught during the third module (quartile) of the program. It is dedicated to the base features of the C++ programming language and C++ Standard Library (STL). The part covers all necessary topics that are needed to start developing a modern CLI[1]-applications in C++14.

The lectures and practical classes are closely inter-related. The lectures are primarily intended to introduce new topics, whereas the practical classes are intended for solving specific problems by coding programs in C++.

2 Learning Objectives

During the course "Introduction to Programming" the participants will:

- study methodology of programming;
- develop algorithmic thinking;
- study approaches and toolkits for the development of Python applications;
- study approaches and toolchains for the development of C++-applications;
- practice application debugging and deployment with respect to various platforms and toolchains.

3 Learning Outcomes

Students who complete this course successfully will learn or acquire:

- basic concepts and methods of software development;
- skills in Python programming to formalize and solve simple development tasks;
- analyze a problem to be implemented in the form of an C++-application;
- design app architecture with respect to problem decomposition and known limitations;
- select the most appropriate toolset for app development;
- create a testbed environment for essential testing of the application.

[1] CLI — Command-line interface.

4 Course Plan

4.1 Part I: semester A quartile 1

#	Topic Name
1	Introduction
2	Dynamic typing. Operations of sequence's, selection, and iteration
3	Float in the computer memory, rounding and other aspects. Strings
4	Strings. Tkinter GUI
5	Functions and recursion. Lambda-functions. Named parameters
6	Tuples. Lists. Function map. Methods Split and Join
7	Zen of python. PEP-8. Documenting code (PEP-257). Files. Exceptions and errors. Program debugging
8	Sets. Dictionaries
9	Sort and found. Introduction into the complexity theory
10	Elements of functional programming. Function Enumerate. Module Functools
11	Object-oriented programming. Main ideas: encapsulation, inheritance and polymorphism. Operator Overriding. Function `Isinstance`. Examples
12	Python for data analysis. Overview

4.2 Part II: semester A quartile 2

#	Topic Name
1	Introduction to C++ (Sec. 4.2.1)
2	Data types and Objects (Sec. 4.2.2)
3	Control flow statements (Sec. 4.2.3)
4	Expressions and Operators (Sec. 4.2.4)
5	Functions and procedures (Sec. 4.2.5)
6	C++ Memory model(Sec. 4.2.6)
7	Classes (Sec. 4.2.7)
8	Strings (Sec. 4.2.8)
9	Standard library (STL) (Secs. 4.2.9, 4.2.10)
10	Operator overloading (Sec. 4.2.11)
11	Templates (Sec. 4.2.12)
12	Exceptions and Move semantics (Secs. 4.2.13, 4.2.14)
13	Inheritance (Sec. 4.2.15)
14	Object-Oriented Analysis and Design (Sec. 4.2.16)
15	*Optional*: C++ plus Python (Sec. 4.2.17)

Notes:

1. Each sequential number above corresponds to a separate theme, whereas a theme can span over one or more lectures and/or practical classes.

4.2.1 Introduction to C++

Introduction to C++. C++ program structure. Statements. Programs and modules. Toolchains and building C++ Programs.

4.2.2 Data types and Objects

Datatypes and Objects. Primitive, composite and user datatypes. Scope of variables (objects). Constants. Basic input/output. Introduction to streams.

4.2.3 Control flow statements

Control flow statements: conditional (`if`, `switch`); loops (`while`, `do..while`, `for`).

4.2.4 Expressions and operations

Expressions. Operations and operators. Operator precedence. Logical and Bitwise operations.

4.2.5 Functions and procedures

Procedural decomposition. Functions and procedures. Formal and Actual Parameters of a Function. Function return value. Function signature. Passing parameters to a function *by value* and *by reference*. Function overloading. Operator overloading.

4.2.6 C++ memory model

Memory model of a C++ program. Addresses and pointers. Difference between pointer and reference. Stack- and heap-memory. Object life-cycle.

4.2.7 Classes

Classes overview. Class members: methods and fields. Access modifiers: `public`, `protected`, `private`. Classes vs. structures. The (`.`) "dot" and (`->`) "arrow" operators. Constructors. Destructor. Safe array. Copy constructor. Copy operation. *Copy-and-swap* idiom. Rule of three.

4.2.8 Strings

Strings in C++. Null-terminated string. `std:string` class. Unicode supporting. String streams. Approaches to deal with strings.

4.2.9 Standard library overview

Standard library (STL) overview. Containers and adapters. Algorithms. Iterators. "`for`-each" loop for iterating collections/containers. `stl::vector<T>` and C-style arrays. Dynamic resizing of a `vector`.

4.2.10 Standard library components

Sequence containers: `vector`, `list`, `dequeue`*. Associative containers: `map`, `set`, `unordered_map`, `unordered_set`. Adapters: `stack`, `priority_queue`. Main `std` algorithms.

4.2.11 Operator overloading

More on operator overloading. Bitwise operations. `std::bitset<N>` class.

4.2.12 Templates

Templates: classes and functions. Header-only approach. `typename` and inner types. Templates and duck-typing. Concepts. `std` template framework. Lambda functions.

4.2.13 Exceptions

Exceptions. *RAII* idiom. Smart pointers.

4.2.14 Move semantics

Move semantics. *Rvalue* reference. Rule of five.

4.2.15 Inheritance

The Three Pillars of Object-Oriented Programming: *Encapsulation*, *Inheritance*, *Polymorphism*. Single inheritance. Virtual and pure-virtual methods. Abstract classes and interface classes.

4.2.16 Object-Oriented Analysis and Design

Introduction to Object-Oriented Analysis and Design. Class relationships.

4.2.17 C++ plus Python

Optional: binding c++ and python together.

5 Reading List

5.1 Part I

Required

1. Mark Pilgrim. *Dive Into Python*. 2004
2. *The Python Tutorial*. URL: `https://docs.python.org/3/tutorial/index.html`.

5.2 Part II

Required

1. Marc Gregoire. *Professional C++*. 3rd ed. 2014
2. Mikael Olsson. *C++ 14 Quick Syntax Reference*. 2nd ed. 2015
3. Stephen R. Davis. *C++ For Dummies*. 7th ed. 2014

6 Grading System

The course grade is based on both ongoing assessment and final examination. Every module ends up with an final exam. The grade for the exam together with a cumulative grade represent a final grade for the module. The ultimate grade G for the whole course is calculated as:

$$G = 0.7 \cdot \min(P_1, P_2) + 0.3 \cdot \max(P_1, P_2), \tag{1}$$

where P_1 is a *first part* final grade, and P_2 is a *second part* final grade.

Grade G is rounded (up or down) to an integer number of points before entering them into records. P_1 and P_2 are also rounded in (1). The conversion of rounded 10-point scaled results to 5-point scaled ones is performed according to Table 1.

6.1 Part I Grading Details

The *final grade* P_1 for the *first part* is calculated as follows:

$$P_1 = 0.4 \cdot E_1 + 0.6 \cdot OA_1, \tag{2}$$

Table 1: Correspondence of ten-point to five-point marks

Ten-point scale [10]	**Five-point scale [5]**
1 — unsatisfactory 2 — very bad 3 — bad	Unsatisfactory — 2
4 — satisfactory 5 — quite satisfactory	Satisfactory — 3
6 — good 7 — very good	Good — 4
8 — nearly excellent 9 — excellent 10 — brilliant	Excellent — 5

where E_1 is a grade of the *first part exam*, which takes place at the end of the quartile 1 (semester A), OA_1 is an *ongoing assessment* grade of the *first part* (both 10-point scale). The ongoing assessment OA_1 measures participant's performance throughout all classes and involves various types of activities (see Sect. 6.3).

6.2 Part II Grading Details

The *final grade* P_2 for the *second part* is calculated as follows:

$$P_2 = 0.4 \cdot E_2 + 0.6 \cdot OA_2, \tag{3}$$

where E_2 is a grade of the *second part exam*, which takes place at the end of the quartile 3 (semester B), OA_2 is an *ongoing assessment* grade of the *second part* (both 10-point scale). The ongoing assessment *OA* measures participant's performance throughout all classes and involves various types of activities (see Sect. 6.3).

6.3 Ongoing Assessment

The ongoing assessment grade is accumulated throughout all the classes and is related to a participant's activity. An ongoing control structure is individual for every class.

During the classes, there are some activities available for students to be involved in. They include (but are not limited by) writing code and developing applications, evaluating practical problems, solving tests, answering questions and so on. Every activity is evaluated and grants some points (*RP*) to participants. We consider two sorts of points: 1) regular points (*RP*) and 2) bonus points (*BP*). *BP*s are given for additional efforts and for excellent jobs.

*RP*s and *BP*s are accumulated during a module. At the end of the module *OA* is calculated according to the following formula for calculating OA_1 and OA_2 of formulas (2) and (3) correspondly:

$$OA = \max\left(\left[10 \cdot \frac{RP + BP}{RP_{max}}\right], 10\right), \tag{4}$$

where, RP_{max} denotes the maximum possible number of points that can be taken during the module.

Finally, some kinds of out-of-class activities can be accounted for as a part of ongoing assessment. *Peer review work*, *preparing and reporting one of a course-related topics* are examples of such activities.

6.3.1 Regular tests

Students' skills in programming are tested using automated testing. This way, a student is assigned an individual task, prepares it by using a personal computer and, then, submits it by using a special

service, such as Yandex.Contest or a repository-based tool. The specific solution is subject to further clarification.

The individual home-based task submissions are to be further reassessed through in-class tests or examinations.

For any two corresponding submissions, one for home work and one for class work, graded as H and C respectively, the resulting grade R is calculated as follows:

$$R = 0.8 \cdot \min(H, C) + 0.2 \cdot \max(H, C). \tag{5}$$

6.4 Other conditions

The *final exams* as well as the intermediate tests are given in the form of a written test (paper- *or* computer-based). One (10-point scale) grade is given for the exam.

The second re-sit of each of the final exams is provided by a board of professors. There is allowed to omit any ongoing assessment when calculating the ultimate grade.

7 Guidelines for Knowledge Assessment

The knowledge gained by the students is systematically and consistently assessed throughout the course, which includes understanding and attendance rate of the lectures, as well as tests taken during practical classes. A quantitative basis is provided elsewhere. The instructors are supposed to check with the students highly probable coding mistakes and drawbacks in advance.

8 Methods of Instruction

Learning happens through traditional face-to-face lectures (classroom presentations being distributed through a repositiry) and consolidation of the delivered knowledge and getting hands-on experience at practical classes.

9 Special Equipment and Software Support (if required)

Students are highly reccommended to use their own laptops with pre-installed and configured software, if possible. The computers in computer classes are also suitable for performing programming tasks. The exact set of software needed for the courses will be listed in an associated educational service, such as LMS or wiki.

References

[1] Stephen R. Davis. *C++ For Dummies*. 7th ed. 2014.

[2] Marc Gregoire. *Professional C++*. 3rd ed. 2014.

[3] Mikael Olsson. *C++ 14 Quick Syntax Reference*. 2nd ed. 2015.

[4] Mark Pilgrim. *Dive Into Python*. 2004.

[5] *The Python Tutorial*. URL: `https://docs.python.org/3/tutorial/index.html`.

Author of the program: _____ Tamara Voznesenskaya

Author of the program: _____ Sergey Shershakov