Carleton UNIVERSITY | Department of Systems and Computer Engineering

ECOR 1041
Computation and Programming

Calendar Description

Software development as an engineering discipline, using a modern programming language. Language syntax and semantics. Tracing and visualizing program execution. Program style and documentation. Testing and debugging tools and techniques.

Lectures three hours per week, laboratories three hours per week.

Prerequisites

This course may not be taken concurrently with ESLA 1300 or ESLA 1500.

Precludes additional credit for COMP 1005, COMP 1405, ECOR 1051, ECOR 1606, SYSC 1005.

Prior Knowledge

No prior experience in computer programming is required. The only background we assume is:

- *language*: reasonable proficiency in reading and writing English;
- *math*: understanding of integers and operations on integers; understanding of functions as mappings from one set (the domain) to another set (the codomain);
- *logic*: familiarity with logical *and*, *or* and *not*;
- *computer literacy*: ability to use email, browse the World Wide Web, and edit text files.

Course Objectives

The course introduces students to concepts that are central to understanding computation and teaches students how to design, code, test and debug small-scale programs written in the Python programming language. By the end of this course, students should:

- know how integers and floating-point numbers are represented on a computer.
- know the fundamental concepts of procedural programming, using Python as the programming language.
- have developed a "mental model" of computation; in other words, learned how to reason about and visualize the execution of program code.
- understand the use of software experiments as an aid to learning.

List of Topics

1. Introduction to Python: using the Python shell as a calculator. Numeric types (`int` and `float`); operations supported by these types. Construction and evaluation of expressions.
2. The binary number system. Representation of integers and floating-point numbers in binary. Limitations of real-number representation in binary.
3. The assignment operator: binding values to variables. Using variables in expressions. Using Python Tutor to visualize assignment operations.
4. Calling built-in functions from the shell. Importing functions from modules. Evaluating call expressions.
5. Defining functions. Passing arguments to and returning values from functions. Using the shell to interactively test functions. Using Python Tutor to trace and visualize the execution of function definitions (creation of function objects), function calls and function execution. Using local variables in function definitions.
6. Simple programs (scripts). The `print` function. Functions that do not have return statements.
7. A step-by-step recipe for designing functions.
8. Using a simple unit-testing framework to automate testing.
9. Character strings (type `str`). Interactive programs (scripts): the input function.
10. Boolean values (type `bool`). Boolean operators. Relational operators. Evaluating expressions that have numbers, and boolean and relational operators. Comparing strings. Making decisions: (`if`, `if-else`, `if-elif-else` and `if-elif` statements).
11. Lists: Python's `list` type; creating list objects; list operators, built-in functions and methods. Using a `for` loop to iterate over a list. Using the `range` function to generate a sequence of list indices.
12. Repeating actions until a condition is satisfied (`while` statements).

Learning Outcomes

By the end of this course, students should be able to:

1. Represent signed and unsigned integers and decimal numbers using binary representation; explain the limitations of real-number computation using floating-point arithmetic.
2. Evaluate expressions consisting of literal values, variables, and operators, using the same evaluation rules as Python. In other words, predict the values that Python calculates when it evaluates expressions.
3. Design and implement Python programs that includes Python's lists and fundamental constructs for controlling program execution to solve a given problem. The constructs include: (1) sequential execution as determined by the ordering of executable statements; (2) selection (`if`, `if-else`, and `if-elif-else` statements); (3) repetition (`for` and `while` statements); (4) function calls.

4. Trace short Python programs. In other words, explain what happens, step-by-step, as the computer executes each statement
5. Visualize Phyton code execution; in other words, draw diagrams that depict the variables in the program's global frame and function activation frames (function arguments and local variables), and the objects that are bound to the variables.
6. Design, implement, execute, trace and test simple functions using Python. Testing should be done using the Python shell, and simple unit-testing framework.
7. Debug a program using simple debugging techniques (e.g., inserting print statements to instrument code or by tracing the code using a program visualization tool)
8. Explain the "client-side" view of one Python container for organizing data, namely: lists

Graduate Attributes (GAs)

The Canadian Engineering Accreditation Board requires graduates of engineering programs to possess 12 attributes at the time of graduation. Activities related to the learning outcomes listed above are measured throughout the course and are part of the department's continual improvement process. Graduate attribute measurements will not be taken into consideration in determining a student's grade in the course. For more information, please visit: https://engineerscanada.ca/

Graduate Attribute	**Learning Outcome(s)**
1.3 Knowledge base for engineering: Fundamental engineering concepts	all

Accreditation Units (AUs)

For more information about Accreditation Units, please visit: https://engineerscanada.ca/

The course has 27 AUs divided into:

- Engineering Science: 100%
- Engineering Design: 0%

Instructor and TA contact

Specific to course offering (tbd)

Textbook and Software

Specific to course offering (tbd).

Evaluation and Grading Scheme

Specific to course offering (tbd).

Breakdown of Course Requirements

Specific to course offering (tbd).

Tentative week-by-week breakdown

Specific to course offering (tbd).

General regulations

Specific to course offering (tbd).