ENGINEER 1D04

Engineering Computation
Winter 2018

McMaster University

Course Outline[1]

Note: This course outline contains important information that may affect your grade. You should retain it and refer to it throughout the semester, as you will be assumed to be familiar with the rules specified in this document.

Instructor

Dr Asghar A Bokhari
Office: ITB 212
E-mail: bokhari@mcmaster.ca

Instructional Coordinator

Ms. Jessica Anderson
Office: ETB 110
Extension: 21163
E-mail: engic@mcmaster.ca

Instructional Assistant Interns (IAIs)

Cody Cooper cooperce@mcmaster.ca
Josh Mitchell mitchjp3@mcmaster.ca

EPIC[2] Instructional Assistant Intern (IAI)

Liam Flannigan flannilg@mcmaster.ca

Drop-In Centre Instructional Assistants

John Nakamura nakamura@mcmaster.ca

Teaching Assistants (TAs)

See course's Avenue to Learn page.

[1] Revised: Jan 2018
[2] Experiential Playground and Innovation Classroom

Schedule

Lecture C01:	Wed	08:30–9:20	HSC 1A1
Lecture C02:	Fri	16:30–17:20	ITB AB102
Weekly 2-hour tutorial:	See course's McMaster Course Timetable		
Weekly 3-hour lab:	See course's McMaster Course Timetable		
Office Hours:	Fri	13:30–14:20	ITB 212

Note: Lecture C01 is for one half of the class, and Lecture C02 is a repeat of Lecture C01 for the other half of the class.

1 Course Overview

1.1 Calendar Description

"Development and analysis of simple algorithms. Implementation of algorithms in computer programming language. Design and testing of computer programs."

1.2 Mission

Computing is, among other things, a powerful and flexible problem solving tool that every engineer needs to understand and be able to use. The *mission* of the course is to introduce students to the field of computing and teach them how to solve engineering problems by designing algorithms and then implementing them in software using a modern programming language—Python in this course.

1.3 Required Resources

1. Textbook: J. Zelle, *Python Programming: An Introduction to Computer Science, 3rd Ed.* Franklin, Beedle & Associates, 2016. ISBN 13: 978-1-59028-2755.

2. Equipment: An i>clicker remote.

1.4 Syllabus

01 Introduction to the Course

02 Fundamentals of Programming Languages

03 Numbers

04 Sequences

05 Objects and Graphics

06 Functions

07 Conditionals

08 Loops and Booleans

09 Software Development

10 Classes

11 Algorithms

12 Recursion

1.5 Work Plan

The work plan for each week will consist of

- 6-hours of preparation,
- 1-hour lecture,
- 2-hour tutorial, and
- 3-hour lab session.

The 6 hours of preparation will by done by the student outside of class.
The lectures will be given by the Instructor. The tutorials will be given by the IAIs, and the labs will be run by the IAIs with the assistance of the Teaching Assistants.

The first lab will be an introduction to the lab facility and using Python (version 3.6) in the lab; it will also include an introductory EPIC Lab exercise.

There will be an assignment in each of the remaining 11 labs. Six will be *minor assignments* marked by computer, and five will be *major assignments* marked part by computer and part by the Teaching Assistants.

The minor assignments are considered to be *practice assignments* and receive a *minor* mark, while the major assignments are considered to be *test assignments* and receive a *major* mark (see Marking Scheme below).

You may use any notes and books during the minor assignments, but you may not use your textbook, any notes, or any other books during the major assignments. Designed to be done within three hours, minor and major assignments must be completed and electronically submitted before the end of the lab session.

To receive credit for a minor assignment submission, you must successfully pass (before you leave the lab) a mini oral quiz given by one of the IAIs or TAs.

The labs are completed using a software package called JupyterHub that provides individual Jupyter Notebook for each user and allows the user to submit the assignments for grading. In order to be familiar with Jupyter Notebook each student is required to install it on his/her personal computer. Installation instructions will be provided during the introduction lab.

There will be two midterm tests:

- Friday, February 9, 2018, 19:00–21:00
- Friday, March 16, 2018, 19:00–21:00

The final exam will be 2.5 hours long. It will test accumulative knowledge and will take place on the date scheduled by the University.

The midterms and final exam are *closed book* (which means you may not refer to any notes or books during the exam period).

1.6 Marking Scheme

The course grade will be based on the student's performance on the assignments, midterm tests, and final exam as follows:

Minor assignments (6)	6%
Major assignments (5)	30%
Midterm test 1	12%
Midterm test 2	12%
Final exam	40%
Total	**100%**
Discussion session bonus	1%
EPIC extra credit exercises (3)	6%
Lecture/Tutorial Participation Negative marks (See explanation below)	

Notes:

1. A student's final percentage will be reduced by one half point for each missed lecture and tutorial. However, there is no penalty for the first two (2) missed lectures and the first two (2) missed tutorials for which participation is recorded. A student who answers less than half of the questions asked during a lecture or tutorial via i>clicker will be considered as having "missed" the lecture or tutorial.

2. The Instructor reserves the right to adjust the marks for a programming assignment, quiz, midterm test, or final exam by increasing or decreasing every score by a fixed number of points.

Discussion sessions with instructor. Each week the Instructor will invite about 50 students from 17:30 to 18:20 on Fridays to discuss how the course is going. These discussion sessions will enable students to ask questions about the course and to give feedback directly to the Instructor. Attendance is optional, but each student who participates in one of the discussion sessions will received a 1.0 percentage point bonus.

EPIC Lab. The Experiential Playground and Innovation Classroom, or EPIC, is used to increase the learning opportunities in 1D04. The main goal of this lab is to expose first year engineering students to hands-on or experiential learning. Experiential learning involves gaining skills and insights through doing an activity and then reflecting on what was done.

For 1D04, the EPIC Lab involves three bonus lab exercises that will be carried out in ETB 125. The lab exercises involve programming Python Blackjack, ASCII Art Generator, and Raspberry Pi Robot. Each EPIC Lab exercise is worth a 2.0 percentage point bonus. Details on EPIC can be found at http://epiclab.mcmaster.ca.

1.7 Logistics

Avenue to Learn. This course will be administered via Avenue to Learn. Go to http://avenue.mcmaster.ca/ to access the course's Avenue to Learn page.

Questions. Please address your questions as follows:

- questions concerning the content of the course to the Instructor
- administrative questions to the Instructional Coordinator
- questions concerning the tutorials, labs, and assignments to the IAIs

Please send e-mail to the instructional staff; do not send mail via Avenue.

Students should be aware that, when they access the electronic components of this course, private information such as first and last names, user names for the McMaster e-mail accounts, and program affiliation may become apparent to all other students in the same course. The available information is dependent on the technology used. Continuation in this course will be deemed consent to this disclosure. If you have any questions or concerns about such disclosure please discuss this with the Instructor.

It is the student's responsibility to be aware of the information on the course's Avenue to Learn page and to check regularly for announcements.

Drop-in centre. To help you with any questions you may have on the material in 1D04, a computing Drop-In Centre is offered by the Department of Computing and Software in ITB 242 on Monday through Friday at 9:30–16:30. If you need any help with the 1D04 material or have questions about Python, please take advantage of this valuable resource.

Clickers

A clicker system is for real-time, in-class feedback. This course will be using the i>clicker system, and all students taking this course are required to use i>clicker remotes. Students are expected to do the following:

1. Purchase an i>clicker remote. You only need one i>clicker for all your classes. You must have your own i>clicker; i>clickers may not be shared among students.
2. Register the i>clicker at

   `http://www.iclicker.com/registration/`

   prior to the second lecture. You must use your MacID where the "Student ID" is requested. *Do not use your student number.* We shall use iclicker classic software and will not integrate with LMS.
3. Bring the i>clicker to all lectures.
4. Maintain the i>clicker in working order throughout the course.

5. Attach a label to your i>clicker with your name so that you will not confuse your i>clicker with someone else's.

6. Record the serial number of your i>clicker in case it is rubbed off.

7. Failure to follow the policies related to i>clickers may result in confiscation of the device(s).

Students are required to attend and participate in the lectures, tutorials, and labs. The i>clicker system will be used by the Instructor during lectures and IAIs during tutorials to assess understanding and also to measure participation. (See Marking Scheme below for how the lecture and tutorial participation is factored into your final grade.) *Using another student's i>clicker or lending an i>clicker to someone to whom the i>clicker is not registered will be considered as academic dishonesty.*

2 Course Policies

2.1 Safety

Safety is a crucial concern for all engineers. Safety in the ENGINEER 1D04 labs will be discussed in the first lab session.

2.2 Academic Dishonesty

You are expected to exhibit honesty and use ethical behavior in all aspects of the learning process. Academic credentials you earn are rooted in principles of honesty and academic integrity.

Academic dishonesty is to knowingly act or fail to act in a way that results or could result in unearned academic credit or advantage. This behavior can result in serious consequences, e.g., the grade of zero on an assignment, loss of credit with a notation on the transcript (notation reads: "Grade of F assigned for academic dishonesty"), and/or suspension or expulsion from the university.

It is your responsibility to understand what constitutes academic dishonesty. For information on the various types of academic dishonesty please refer to the Academic Integrity Policy, located at http://www.mcmaster.ca/academicintegrity/.

The following illustrates only three forms of academic dishonesty:

- Plagiarism, e.g., the submission of work that is not one's own or for which other credit has been obtained.

- Improper collaboration in group work.

- Copying or using unauthorized aids in tests and examinations.

Your work must be your own. Plagiarism and copying will not be tolerated! If it is discovered that you plagiarized or copied, it will be considered as academic dishonesty.

Students may be asked to defend their written work orally.

2.3 Discrimination

The Faculty of Engineering is concerned with ensuring an environment that is free of all adverse discrimination. If there is a problem, that cannot be resolved by discussion among the persons concerned, individuals are reminded that they should contact their Department Chair and the Human Rights and Equity Services (HRES) office as soon as possible.

2.4 Academic Accommodation

Students who require academic accommodation must contact Student Accessibility Services (SAS) to make arrangements with a Program Coordinator. Academic accommodations must be arranged for each term of study. Student Accessibility Services can be contacted by phone 905-525-9140 ext. 28652 or e-mail `sas@mcmaster.ca` . For further information, consult McMaster University's Policy for Academic Accommodation of Students with Disabilities.

2.5 Missed Work

A student who would like to receive accommodation for missed academic work due to an absence needs to complete a McMaster Student Absence Form (MSAF) on-line at http://www.mcmaster.ca/msaf/. When the MSAF tool asks you for the party who should receive your request for accommodation, enter engic@mcmaster.ca. MSAFs sent to any other e-mail address will be ignored.

2.6 Course Modifications

The Instructor and University reserve the right to modify elements of the course during the term. The university may change the dates and deadlines for any or all courses in extreme circumstances. If either type of modification becomes necessary, reasonable notice and communication with the students will be given with explanation and the opportunity to comment on changes. It is the responsibility of the student to check their McMaster e-mail and course web sites weekly during the term and to note any changes. Your McMaster e-mail is the one with the address ending in `@mcmaster.ca`. This is a separate e-mail address from your Avenue address.

2.7 Other Policy Statements

- Significant study and reading outside of class are required.
- Student are encouraged to ask questions during lectures, tutorials, and labs during minor assignments.
- If there is a problem with the marking of an assignment, the student should first discuss the problem with the TA who marked it. Assignment marks will only be changed if the problem is reported within two weeks of the date that the assignment was returned.
- A student may not use his or her notes and books during major assignments in the lab, the midterm tests, and the final exam.
- No electronic devices (calculators, cell phones, etc.) may be used on major lab assignments, midterm tests, or the final exam. They will be confiscated if they are within reach of a student, whether or not they have been actually used.

- E-mail with a source address outside of McMaster University will not be read by the instructional staff.
- Suggestions on how to improve the course and the Instructor's teaching methods are always welcomed.

3 Learning Objectives

3.1 Postcondition

A *learning objective* for a course is something the student is expected to know and understand or to be able to do by the end of the course. The learning objectives for this course are given below. Taken together, this set of learning objectives constitute the *postcondition* of the course.

1. Students should know and understand:
    a. Why an engineer needs to understand computing.
    b. What an algorithm is.
    c. Programming language concepts.
    d. Basic data types.
    e. The software development process.
    f. Software design principles.
    g. Software testing principles.
    h. Mathematical and logical concepts.
2. Students should be able to:
    a. Analyze a simple program.
    b. Construct a simple program that satisfies a simple specification.
    c. Formulate a test plan for a simple program.
    d. Use a modern programming language implementation as a problem-solving tool.

By the end of the course, each student should be comfortable programming in the Python programming language and able to use it for everyday programming purposes.

3.2 Precondition

The *precondition* of the course is the set of university-level learning objectives that the student is expected to have achieved before the start of the course. Since this course has no university-level prerequisites, its precondition is empty.

3.3 Mapping to Graduate Attributes and Indicators

The table below shows how the course objectives of the course map to the graduate attributes and indicators relevant for ENGINEER 1D04. These graduate attributes and indicators are a subset of the full list provided by the Office of the Associate Dean (Academic) that are required by the CEAB (Canadian Engineering Accreditation Board.) The numbering used for the graduate attributes and indicators matches that given in the document produced by the Office of the Associate Dean.

A01 Knowledge	
1.1 Competence in Mathematics	1h
1.3 Competence in Engineering Fundamentals	1e
1.4 Competence in Specialized Engineering Knowledge	1b–1g
A02 Analysis	
2.2 Demonstrates an ability to identify a range of suitable engineering fundamentals (including mathematical techniques) that would be potentially useful for analyzing a technical problem.	2a, 2b
A03 Investigation	
3.2 Selects appropriate model and methods and identify assumptions and constraints.	2a
A04 Design	
4.1 Recognizes and follows an engineering design process.	1e, 2b, 2c
4.2 Recognizes and follows engineering design principles	1f, 2b
A05 Tools	
5.2 Demonstrates an ability to use modern/state of the art tools	2d
A08 Professionalism	
8.1 Demonstrates and understands the role of the engineer in society, especially in protection of the public and public interest.	1a, 1e

3.4 Learning Outcomes

A *learning outcome* is a measure of how well the students in a course have met one or more of the learning objectives of the course. For this course, there is a learning outcome corresponding to each learning objective (of the postcondition). The learning outcomes are expressed as rubrics; they are found at the end of this document. The learning outcomes are measured at the end of the course. The resulting measurements are used to improve the course; they have no effect on a student's course grade.

Table 1: **Students should know and understand (1/4)**

Topic	**Below**	**Marginal**	**Meets**	**Exceeds**
Understanding Computing 1a	The student is not marginal or better.	The student can recall that computing is now, and has been in the past, a crucial component in engineering work. This includes being aware of case studies in which software failure resulted in death or extreme financial loss. This also includes remembering major details about important historical figures in computing and knowing the definition of software engineering.	In addition, the student can contrast the different programming objects that can be used to represent the same mathematical object. This includes understanding the difference between integers, `int`, `long`, real numbers, rational numbers, floating point numbers, decimal numbers and binary numbers. The student should understand the inherent danger involved in approximating real arithmetic by floating point arithmetic.	

Table 2: **Students should know and understand (1/4)**

Topic	**Below**	**Marginal**	**Meets**	**Exceeds**
Algorithms 1b	The student is not marginal or better.	The student can recognize and compare simple algorithms (limited nesting of control structures) with respect to equivalence of behavior. The students should also be able to compare control structures, like for loops versus while loops. Algorithms that students should be able to compare and recognize include selection sort, linear search, binary search, etc.	In addition, the student can compare simple algorithms with respect to some standard metric such as speed, storage and understandability. For instance a linear search that stops when the element is found versus one that always transverses the full list.	In addition, the student can recognize and reason about the implementation and performance of nontrivial algorithm (such as a recursive algorithm or one with significant nesting of the control structures). Examples are the algorithms for merge sort and quicksort.

Table 3: **Students should know and understand (2/4)**

Topic	**Below**	**Marginal**	**Meets**	**Exceeds**
Programming Language Concepts 1c	The student is not marginal or better.	The student can distinguish the fundamental programming language concepts of variable, constant, literal, scope, control structure, type, function, file, expression and statement.	In addition, the student can distinguish the programming language concepts centered around objects, including state, method, field, accessor, mutator, constructor, and encapsulation. The student can recall the meaning of exception handling and aliasing.	In addition, the student can contrast imperative, object oriented and functional programming.
Basic Data Types 1d	The student is not marginal or better.	The student can differentiate, and evaluate expressions (without implicit type conversion or short-circuiting) using, the following data types in Python: integer, long, float, boolean, string and list. The student can convert between bases (binary, decimal and hexadecimal).	The student can differentiate and evaluate expressions (including those with implicit type conversion and short-circuiting) using nested data structures, including lists of lists.	In addition, the student can differentiate abstract data structures, such as stacks, queues, or trees.

Table 4: **Students should know and understand (3/4)**

Topic	**Below**	**Marginal**	**Meets**	**Exceeds**
Software Development Process 1e	The student is not marginal or better.	The student can recognize the need for a rational development process and can differentiate the different phases of a process.	In addition, the student can contrast the most common software development models, including waterfall, spiral, top down refinement, prototyping and incremental development.	
Software Design Principles 1f	The student is not marginal or better.	The student can explain several software design principles.	In addition, the student can recognize when software design principles are properly applied or when they have been violated or ignored.	In addition, the student can illustrate how software design principles can be applied to engineering in general.

Table 5: **Students should know and understand (4/4)**

Topic	Below	Marginal	Meets	Exceeds
Software Testing Principles 1g	The student is not marginal or better.	The student can describe and contrast the main approaches for selecting test cases, including unit tests, blackbox tests, whitebox tests, statistical random test and wild random tests.	In addition, the student can identify the consequences of the ideas behind the test case selection approaches. For instance, they can explain how testing can and cannot be used to check for correctness, reliability, and robustness and they can reason about the number of test cases to achieve statement coverage.	In addition, the student understands the theoretical and practical limits of what testing can achieve, such as the infeasibility of exhaustive testing and the challenges with graphics and floating point numbers.
Mathematical and Logical Concepts 1h	The student is not marginal or better.	The student can interpret the meaning of simple mathematical concepts, such as summation notation, product notation and vector notation.	In addition, the student can manipulate simple mathematical objects to show some property, or to simplify an equation. For instance, the student could use the rules for manipulating summations to simplify an expression.	In addition, the student can manipulate boolean expressions using the laws of boolean algebra, including DeMorgan's Law.

Table 6: **What students should be able to do (1/2)**

Topic	**Below**	**Marginal**	**Meets**	**Exceeds**
Analysis of Programs 2a	The student is not marginal or better.	The student can analyze a simple concrete program.	In addition, the student can analyze a program that abstracts over variables. Questions might be something like - for any value of the input integer n what has to be true of the program output.	In addition, the student can analyze a simple program that abstracts over statements, expressions, functions or uses recursion. Questions might be something like - for any statement S will the following loop terminate.
Constructing Programs 2b	The student cannot construct a program utilizing conditionals, loops and functions that satisfies a simple specification.	The student can construct a program utilizing conditionals, loops and functions that satisfies a simple specification.	In addition, the student can construct a program utilizing classes that satisfies a specification.	In addition, the student can construct a program utilizing recursion that satisfies a simple specification.

Table 7: **What students should be able to do (2/2)**

Topic	**Below**	**Marginal**	**Meets**	**Exceeds**
Formulating Test Plans 2c	The student does not know what a test plan is.	The student can propose a test plan that will achieve 100% statement coverage (for a program where this is possible).	In addition, the student can propose test plans that consider testing for abnormal cases, such as division by zero, missing files, insufficient inputs, etc.	In addition, the student understands the theoretical and practical limits of what testing can achieve, such as the infeasibility of exhaustive testing and the challenges with graphics and floating point numbers.
Using a Programming Language Implementation 2d	The student cannot use a programming language implementation.	The student can execute code with an interpreter.	In addition, the student can compile code and execute it.	In addition, the student can write a software system system consisting of code in several files.