CS403
PRINCIPLES OF PROGRAMMING LANGUAGES

Course Outline

Instructor:	Prof. G. Mierzwinski
Office:	Online Meetings
Phone:	Email
E-mail:	gmierzwi@ubishops.ca / gregory.mierzwinski@ubishops.ca

Lectures: **Tuesday-Thursday 11:30 - 13:00 in N8 (Nichols Building)**

Office-hrs: **None. Only available online by appointment.**

Website: **https://gmierzwinski.github.io/bishops/cs403/index.html**

A. Course description

Programming Languages are regarded by many as the crowning achievement of the discipline of Computer Science. No wonder, since almost all of the subfields of Computer Science are involved in language design and implementation. In addition, a large part of computer science can be viewed as an engineering discipline that tries to create software artifacts. These “software artifacts” are built using computer languages, much in the way that literature is “built” using natural languages. The language thus becomes the primary tool of the computer scientist's trade. Knowing what your tools are capable of is of primary importance in software construction.

This course aim at providing an overview of the principles on which programming languages are built. You have undoubtedly seen at least one (and probably two or even more) programming languages. Chances are, they are all similar, coming from a particular programming paradigm called imperative programming. Special emphasis will therefore be given to the other two mainstream programming paradigms (namely, functional and logic programming).

The main objectives of the course thus are to introduce the paradigms, features and methods of definition of modern programming languages, to provide the student with the necessary tools for the critical evaluation of existing and future programming languages and programming language constructs, and to provide an understanding of the formal methods used in the definition of languages.

We base the course on the following however tentative schedule:

(1 week) Introduction; history and evolution of major programming languages
(2 weeks) An introduction to functional programming using Haskell
(2 weeks) An introduction to logic programming using Prolog

- **Midterm on Haskell, and Prolog**

(2 weeks) Formal description of programming languages
Syntax (BNF, EBNF, diagrams)
Semantics (abstract machine, lambda calculus)
(1 week) The compilation process (recursive descent)

(3 weeks) A more in-depth look at the procedural paradigm
Primitive data types
Variables (names, binding, type checking, strong typing, scope, lifetime)
Modelling objects (storage, modification, pointers, storage management)
Subprograms and parameters (arguments, parameter passing, higher order functions)

- **Final Project on Imperative Language Compilation, and/or Interpretation**

B. Organization of the course

The class will meet for 1 1/2 hrs on:

Tuesday – Thursday 11:30 - 13:00

C. Evaluation.

The course components are weighted as follows

Component	**Weight**
Final Project	40%
Midterm	40%
Assignments	20%

ASSIGNMENTS: There will be 4 programming assignments during the term, each of approximately 2 weeks duration.

Policy on late assignments: *All late submissions will receive 0.*

MIDTERM: A midterm on Haskell, and Prolog.

FINAL PROJECT: The final examination in this course will consist of a final project that will have you build a cross-process communication system for an autonomous object (e.g. a rover, drone, etc.) using the knowledge you gained from the second-half of the semester. **Python is strongly recommended for this project to simplify your work.**

GROUP WORK: The **assignments** may be done in groups consisting of one, two, or three students. All the collaborators must be currently enrolled in the course. A single joint solution must be submitted for each group. All submissions must include the names and student numbers of all the collaborators.

For the final project, group work might be allowed (more information will be given during the semester), but there will be extra work to do.

E. Textbooks and other readings:

See the references section on the course website. Aside from those, the following book is recommended for code quality.

Code Craft: The Practice of Writing Excellent Code
by Pete Goodliffe
Recommended supplemental reading.