1 Introduction

Since the mid 1950s programming in a high-level language has been dominated by “imperative” languages, of which the most early example is Fortran. In such languages, the basic linguistic unit is an instruction for modifying the state of a machine, hence the name “imperative” language. Around the start of the present century object-oriented programming became popular as the basic teaching language (a big mistake, in my opinion) and as a result Java or C++ are the common first languages that a student learns today. This builds on the imperative paradigm by offering higher-level abstractions. Unfortunately, these are complicated and many students learn the Object Oriented Paradigm in a half-baked way.

From the earliest days of high-level programming languages, however, the language Lisp has offered an alternative view of programming as a prescription for evaluating expressions. Still more recently, new programming languages have emerged that embody new “programming paradigms”. A truly new language offers a new viewpoint on what computing is.

In this course we will study some new (and not so new) programming paradigms; functional programming, imperative programming and object-oriented programming. The main ideas I wish to expose you to are

1. Procedural abstraction, functions as “first-class” citizens.
2. Data abstraction.
3. The notion of state.
4. Object-oriented programming.
5. Type systems, polymorphism and genericity.
6. Streams and lazy evaluation.
7. Interpreters for functional languages.

There is no suitable text-book that meets the needs of this class so you will have to rely on class notes and material that we will provide. In the old days I taught this course using Scheme and I used “Structure and Interpretation of Computer Programs” by Abelson and Sussman. This is a
fantastic book. It is no mere text churned out by hacks who have nothing better to do. However, Scheme is an untyped language and it has become clear that understanding a typed language is crucial.

The programming languages that we will use are:

1. F#
2. Java and
3. Scheme.

These languages are running on the department computers. If you are an emacs user the fsharp mode is an excellent programming environment; but in these dark times emacs has been largely forgotten. If you have your own computer you will want to download F# and some kind of environment and run it on your own machine. If you are running Windows you can use the Visual Studio environment. If you are running a Mac OS you can use Atom with the Ionide environment, or Xamarin (which is not free) or emacs (which is what I use). Similarly, you can install Scheme and a scheme environment. Details will be posted on the web page later.

Some of the ideas that we discuss are illustrated by other languages, for example, OCaml, Haskell or Sml. You will be exposed to concepts embodied in these languages but we will not require you to program in them. The point of this class is not to master the syntax of several different languages but, rather, to learn concepts of programming languages. Until this term I used Sml which I liked but which many students hated. Accordingly, I have switched to F# which I am only learning now. However, conceptually the functional programming core is exactly the same as that of Sml. F# is supported by Microsoft and is integrated into the .net world so your knowledge can be useful later.

It is important to emphasize what this course is not. It is not a course on a dozen different languages. There is a myth that knowing more languages is better. The fallacy in this argument is that most people who know several languages actually know only one type of language but in several different forms.

More than any other course I have taught, people come to this class with their own expectations about what they will learn. Let me say at the outset that the purpose is not to cram you with the syntax of the latest popular language. Many people in the past have made remarks like “none of my friends have ever heard of Sml, why are we learning it?” One answer to this question is that I am trying to teach you programming language ideas. I have been told that students do not find Sml resources on the web. I think you will find lots of F# resources on the web. Underlying this subject is a very interesting mathematical theory, “the mathematical semantics of programming languages.” I will not talk about that subject in this class.

2 Grading

The homework accounts for 40% of the total course marks. we will not give extensions unless you have a valid medical reason. Being busy with other classes is not a valid medical reason.
The midterm examination accounts for 10% and the final examination will be 50% of the total. The midterm will be in class. The final examination will be in the final examination period. We will permit a one-page cheat sheet for the midterm and 3 pages for the final.

3 Staying in Touch

There will be two class web pages. Please ensure that you have access to them. One of them is an open web page that we maintain. The URL is:


Homeworks, announcements, notes and other information will be posted here. The second web page will be set up through the course management system (myCourses) at McGill. This will be used for the discussion forum which you should be able to see when you click on comp302 after logging in to myCourses. Students often set up a closed Facebook group. I hope that I will be allowed access to it so that I can respond to queries. I have seen students confidently answer questions incorrectly on other groups. By all means set up another closed group to criticize the course (and me) anonymously.

We will hold office hours. My office hours are Monday and Friday from 1:30pm to 3:00pm in Room 105 North Wing McConnell.

Please do come and see me during office hours. Do not feel that you are infringing on their time – office hours represent time has been reserved specifically to see you. If you would like to discuss a program, please bring a copy of the code or a laptop with the code on it. We cannot figure out what is wrong with a program just from a description of its behaviour. You should walk into my office during office hours even if there is someone there talking to me. There are times when students come and “hang around” in my office throughout the period (which I welcome) so if you wait for them to leave you will never get a chance to talk to me.

The teaching assistants will also hold office hours to be announced later on the web site.

4 Academic Integrity

McGill University values academic integrity and treats it seriously. Please see the website

http://www.mcgill.ca/integrity

for more details. In the present class we expect that the work that you turn in is your own. You can discuss ideas and approaches with your friends but you cannot blindly copy their solutions. If we find that someone has handed in a correct solution that they cannot satisfactorily explain to me, we will treat it as a case of cheating and prosecute accordingly.