CMPU101
Problem Solving and Abstraction

Syllabus and Calendar
What is Computer Science?

Study of

1. Algorithms: series of sequential steps that transform inputs into desired outputs.

2. Data: information organized for processing by an algorithm.

3. Automata: Device, person, trained gerbil, etc. that can follow instructions of algorithm. For us, the automata will be a cybernetic device.
Problem Solving and Abstraction

Focus of this course.

– Problem Solving: Recognizing common patterns in data and using standard design techniques will help solve new problems.

– Abstraction: Technique of suppressing irrelevant detail to make implementation simpler.
Our route to learning problem solving via computer is through the use of a computer language known as Scheme, Racket, or Swindle.

These languages are all descendants of one of the oldest programming languages – LISP

LISP stands for "LIS"t "P"rocessing. A list is any characters contained within matching parentheses "("open and then close")"
Functional Programming

• Functional programs treat computation like the evaluation of mathematic functions (although not all arithmetic is done on numbers).

• The output value of a function depends only on the arguments (input values) to the function.

• For any repeated application of the function on the same inputs, a function will produce the same output.
Programming Languages

• Every programming language comes with 2 sub-languages: A language of atomic data and a language of operations.

• Arithmetic in a computer language combines data and operations.
The world of Racket data types.

There is another type called "void", which means no value.

The **symbols** here are not names for variables. They are "quoted symbols". Names are composed of symbols, but names in the GE are different from a quoted symbol.
Parentheses are an integral part of Racket. A left parenthesis indicates the start of a function call or the start of a call to a special form.

Programs are collections of >= 1 function. Ex:

\[ f(x) = x^2 - 7 \quad \text{and} \quad g(y) = \frac{8}{y + 3} \]

Racket uses "prefix" instead of "infix" notation:

\[ (- (* x x) 7) \quad (\div 8 (+ y 3)) \]

These are examples of the "non-empty list" data type.
Evaluating long prefix expressions

Start by evaluating expressions that have all primitive arguments (i.e. that can be reduced to a value), from left to right:

\[
(+ (* 3 (+ (* 2 4) (+ 3 5))))(+ (- 10 7) 6))
\]
\[
(+ (* 3 (+ (* 2 4) (+ 3 5))))(+ (- 10 7) 6))
\]
\[
(+ (* 3 (+ 8 (+ 3 5))))(+ (- 10 7) 6))
\]
\[
(+ (* 3 (+ 8 8))(+ (- 10 7) 6))
\]
\[
(+ (* 3 16)(+ (- 10 7) 6))
\]
\[
(+ 48 (+ (- 10 7) 6))
\]
\[
(+ 48 (+ 3 6))
\]
\[
(+ 48 9)
\]
\[
57
\]
Using built-in indentation

(+ (* 3
      (+ (* 2 4)
          (+ 3 5)))
   (+ (- 10 7)
      6))

This is a indented version of a non-empty list.
When you start up DrRacket, about 200 primitive operations and constant values are written into a table called the Global Environment.

The primitive operations are functions that you can use and combine to write compound functions of your own.
After you choose a language and press Run, the window is split into the upper part (the definitions window) and the bottom part (the interactions window).

The definitions window is where you type programs because the contents of that window can be saved and re-opened. The interactions window is used to show the results of running programs in the DW and also to evaluate expressions.
Programs consist of Expressions

An expression is either a plain number or something that starts with a left parenthesis "(" and ends with a matching right parenthesis ")". DrR shades the text between matching parens to help you know the parens match.

The default evaluation rule for non-empty lists is used every time a list expression is evaluated. By this rule, for an expression such as (d1  d2  d3  ...  dn), each of the expressions d1...dn are evaluated. d1 must be a primitive operation (function), and the results of evaluating d2...dn are arguments to function d1.