What is Computer Science?
Study of
1. Algorithms: series of sequential steps that transform inputs into desired outputs.
2. Data: information organized for processing efficiency.
3. Automata: Device that can follow instructions of algorithm. For us, the automata will be a cybernetic device – a computer.

Problem Solving and Abstraction
Focus of this course.
– Problem Solving: Recognizing common patterns in data and using standard design techniques to 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 sequence of valid Racket type contained within matching parentheses "("open and then close")"

Functional Programming
• Functional programs are composed of functions similar to math functions in terms of input and output.
• The output value of a function depends only on the arguments (input values) to the function.
• Any repeated application of a function on the same input(s) will produce the same output.
• Every entry returns a value.

Programming Languages
• Every programming language comes with 2 sub-languages: A language of primitive data and a language of primitive operations*.
• Functions you write in a computer language combine data and operations.

* data types can be added to by a programmer.
Syntax and Semantics

• The syntax rules of a language specify the legal words, expressions, statements, or sentences of that language.
• The semantic rules of a language specify what the legal words, expressions, statements, or sentences mean.
• Together, the syntax and semantics make up the computational model of a computer language.

Parentheses are an integral part of a Racket program. An unquoted left parenthesis indicates the start of a function call or the start of a call to a special form.

A quoted left parenthesis is preceded by an apostrophe: '

An unquoted left parenthesises is just: (  

To convert an infix to a prefix expression,  
1. Break down any procedure that involves multiple primitive operations into those primitive operations. E.g., \( x^2 = (x \times x) \), \( 3x = (3 \times x) \)  
2. Fully parenthesize each (binary) expression using PEMDAS to order non-parenthesized expressions.  
3. Move each operator to the immediate right of its closest enclosing left parenthesis.

The World of Racket Data

Atomic data includes numbers, booleans, characters, quoted symbols, and the empty list.  

Compound data includes strings, vectors, functions, and quoted non-empty lists.

Note: We will consider the word "symbol" to be synonymous with "quoted symbol".  
For the lab today, we will use numbers, strings, and functions.

Programs are collections of >= 1 function. Examples (in infix notation):  
\[ f(x) = ((x^2) - 7) \] and  \[ g(y) = (8 / (y + 3)) \]

Racket uses "prefix" instead of "infix" notation:  
\[ f(x) = (- (* x 7)) \]  \[ g(y) = (/ 8 (+ y 3)) \]
These are examples of the unquoted "non-empty list" data type (expressions). The process is called "default evaluation".

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:

\[
\begin{align*}
(+ (* 3 (+ (* 2 4) (+ 3 5)))) & (+ (- 10 7) 6) \\
(+ (* 3 (+ (* 2 4) (+ 3 5)))) & (+ (- 10 7) 6) \\
(+ (* 3 (+ 8 (+ 3 5)))) & (+ (- 10 7) 6) \\
(+ (* 3 (+ 8 8)) & (+ (- 10 7) 6)) \\
(+ (* 3 16) & (+ (- 10 7) 6)) \\
(+ 48 & (+ (- 10 7) 6)) \\
(+ 48 & (+ 3 6)) \\
(+ 48 9) & 57
\end{align*}
\]
Programs consist of Expressions

Expression:
Either an atomic or compound data type, e.g., number, string or something that starts with a left parenthesis and ends with a matching right parenthesis.

A literal is an expression that evaluates to itself, i.e., the output displayed is exactly the same as the input typed in.

Example literals

A literal is an expression that evaluates to itself, i.e., the output displayed is exactly the same as the input typed in.

1. numeric literals: 1, 34, 42, #i2.3344223210, ¾
2. string literals: Almost any characters inside " "'s:
   Ex: "Hello, world", "elephant", "tigre", "$#&**";

Default Evaluation Rule for Functions

The default evaluation rule for non-quouted, non-empty lists is used every time a non-quouted, non-empty list expression is evaluated. By this rule, for an expression such as (d1 d2 d3 ... dn), where every di is a valid scheme entity, each of the expressions d1...dn are evaluated. If d1 is a defined function, then d2...dn are evaluated as arguments to d1. Each expression is evaluated under the default rule if d1 is a function. If d1 is a special form, some parts of d2...dn are not evaluated.

DrRacket Developers' Application

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 you can use and combine to write functions of your own.

The Global Environment (GE)

A table with columns for name and value which is populated with all built-in names and values when you open a language in DrRacket.

You do not have access to look at this table, but you can look up built-in functions in the Help Desk under the Help menu.

When we talk about the define special form, you will learn how to add more name/value pairs to the GE.