**COMPSCI 1JC3**
**Introduction to Computational Thinking**
**Fall 2017**

# 02 What is Functional Programming?

William M. Farmer

Department of Computing and Software
McMaster University

September 15, 2017

McMaster University

## Admin

- Discussion sessions start this week (chapter 1 of CT).
- M&Ms start this week.
- First programming assignment will be posted this Friday, September 15, and will due on Friday, September 29.
- Office hours: To see me please send me a note with times.
- Are there any questions?

## Advice

1. Make a weekly work plan for CS 1JC3!
2. Schedule time during the week to do your your work plan!
3. Programming can only be learned by writing programs!

## Review

1. Computational thinking includes:
   - Mathematical thinking.
   - Engineering thinking.
   - Scientific thinking.
   - Artistic thinking.
2. Fundamental question of computational thinking.
3. Gottfried Leibniz, the brilliant polymath and fantastic computational thinking.

## Computers and Programs

- What do computers do?

  Store and manipulate information.

- What determines the behavior of a computer?

  A sequence of statements called a program.

- Why are computers so useful?

  1. Hardware enables massive amounts of data to be stored and manipulated with great speed and accuracy.
  2. Software provides the means to control the behavior of the hardware with extraordinary power and flexibility.

## Programs and Programming

- A programming language is a formal language for writing programs.
- An imperative statement expresses an action to be performed, while a declarative statement expresses a property to be employed.
- An imperative program is a sequence of imperative statements that expresses how the program will work.
- A declarative program is a sequence of declarative statements that expresses what the program will achieve.
- A programming paradigm is a well-developed style of programming.
- Each programming language supports one or more programming paradigms.

## Imperative Statements (iClicker)

Which of the following is an imperative statement in English?

A. Long live the King!

B. If you are so smart, why ain't you rich?

C. Start your engine.

D. Your dinner is ready.

## Major Programming Paradigms

1. Procedural. Programs are imperative; a program is a collection of procedures possibly having side effects.

   Examples: Fortran, COBOL, C, BASIC, Pascal, Ada.

2. Object Oriented. Programs are imperative; a program behaves as a collection of interacting objects.

   Examples: Smalltalk, C++, Java, C#, Python.

3. Functional. Programs are mostly declarative; a program is a collection of side-effect free function definitions.

   Examples: Lisp family, ML family, Haskell.

4. Logical. Programs are mostly declarative; a program is a collection of logical statements.

   Example: Prolog.

## Modes of Program Execution

1. The program can be interpreted directly line by line.
   - Advantage: Supports interactive development and debugging of code.
   - Disadvantage: Interpreting code is generally slower than executing compiled code.
2. The program can be compiled into native machine code.
   - Advantage: The machine code is optimized to run fast.
   - Disadvantage: Code development is more difficult.

Haskell programs can be executed by both modes!

## Values

- Values are the information (called data) stored and manipulated by computer programs.
- Examples:
  - Booleans that represent the truth values true and false.
  - Machine integers that represent small integers.
  - Floating point numbers that represent rational numbers in scientific notation.
  - Strings that represent sequences of characters.
  - Tuples that represent sequences of values of different kinds.
  - Lists that represent sequences of values of the same kind.
  - Functions that represent mathematical functions.
- There are various ways that mathematical objects and other data are represented in programming languages.

## Expressions

- An expression is a syntactic entity that denotes a value.
- Examples in Haskell:
  - `(x * 2) + 7`
  - `"abc"`
  - `True && y`
- An atomic expression is an identifier (e.g., `x` or `rotateHorse`) or a literal (e.g., `2.3`, `"cat"`, or `True`).
- A compound expression is formed by applying a function or operator to other expressions (e.g., `abs 10` or `1 + 2`).
- The value of an expression is obtained by evaluating the expression.

## Expressions Question (iClicker)

Which of the following is not a literal in Haskell?

A. `False`.

B. `1000000000000000`.

C. `"The secret is to reboot!"`.

D. $\boxed{\texttt{x1887.}}$

# Types

- A data type (or type for short) is a syntactic entity that denotes a collection of values of similar form.
- Examples in Haskell:
  - `Bool` denotes the values `True` and `False`.
  - `Int` denotes the set of machine integers.
  - `Float` denotes the set of floating point numbers.
  - `Integer` denotes the set of integers.
  - `Integer -> Integer` denotes the set of functions from `Integer` to `Integer`.
- A type error happens when a value of one type is used where a value of another type is expected.
- An expression is type checked in Haskell (before it is evaluated) to determine if it contains any type errors.
- The following evaluates to the type of the expression $e$:

  `:type` $e$
- Types are used extensively in computing and logic.

# Types (iClicker)

Which type in Haskell contains infinitely many members?

A. `Bool.`

B. `Int.`

C. `Float.`

D. `Integer.`

# Mathematical Functions

- Functions are the one of the most fundamental values in mathematics and computing.
- Definition 1: A function is a rule $f : I \to O$ that associates members of $I$ (inputs) with members of $O$ (outputs).
  - Every input is associated with at most one output.
  - Some inputs may not be associated with an output. Example: $f : \mathbb{Z} \to \mathbb{Q}$ where $x \mapsto 1/x$.
- Definition 2: A function is a value (set) $f \subseteq I \times O$ such that if $(x, y), (x, y') \in f$, then $y = y'$.
- Each function $f$ has a domain $D \subseteq I$ and a range $R \subseteq O$.
- Important properties of functions: total, surjective, injective, bijective.

# Function Definitions in Haskell

- A function definition has the form:

  $f$ `::` $t^{\mathrm{i}}_1$ `->` $\cdots$ `->` $t^{\mathrm{i}}_n$ `->` $t^{\mathrm{o}}$

  $f\ p_1 \cdots p_n$ `=` $e$

  where $n \geq 0$.
- The first line declares the type of the function $f$, while the second line defines $f$.
- $t^{\mathrm{i}}_1, \ldots, t^{\mathrm{i}}_n$ are the types of the inputs to $f$, and $t^{\mathrm{o}}$ is the type of the output from $f$.
- $p_1, \ldots, p_m$ are the formal parameters of the definition that represent the inputs to $f$.
- The expression $e$ is the result, defined in terms of the formal parameters, that describes the output from $f$.
- Example:

```
minus :: Integer -> Integer -> Integer
minus m n = m + (-n)
```

## Function Applications in Haskell

- A function application has the form

  $f\ a_1 \cdots a_n$

  where each $a_i$ is an expression of type $t_i^{\mathrm{i}}$.
- $a_1, \ldots, a_m$ are the actual parameters of the application whose values are the inputs to $f$.
- $f\ a_1 \cdots a_n$ is an expression of type $t^{\mathrm{o}}$ whose value is the output from $f$.
- $f\ a_1 \cdots a_n$ is evaluated by substituting $a_1, \ldots, a_m$ for $p_1, \ldots, p_m$ in $e$ and then simplifying the resulting expression ("plug and chug").
- Example:

  ```
  minus 4 9
  ```

  which is evaluated by plugging `4` and `9` into `m` and `n` in `m + (-n)` and then chugging to get `-5`.

## What is Functional Programming?

- Programs are declarative: A program is a sequence a side-effect free function definitions.
- Results are produced by evaluating expressions built from functions.
- Functions are defined as first-class values and used as rules.
- Recursion plays a crucial role.
- State change and data mutation are avoided as much as possible.

## Advantages of Functional Programming?

These are some of the advantages of functional programming over imperative programming:

1. The meaning of a program is simpler and more explicit.
2. Testing and reasoning about programs is much simpler.
3. Expressions can be safely moved around in the code.
4. Code is more compact.
5. Evaluation can be performed in parallel.

## Leading Functional Programming Languages

- Lisp family (e.g., Scheme, Common Lisp, Clojure).
  - Lisp is the second oldest programming language (1958).
  - Imperative + FP.
  - Weak type system with dynamic type-checking.
- ML family (e.g., Standard ML, OCaml, F#).
  - Imperative + FP
  - Strong type system with static type-checking.
- OO + FP languages
  - Python
  - Ruby
  - Scala.
- Pure FP languages
  - Erlang.
  - Haskell.

## Why Functional Programming for CS 1JC3?

- The language is simpler than an imperative programming language:
  - No imperative statements.
  - No loops.
- The focus is on what instead of how.
  - The implementation takes care of the how.
- Functional programming encourages good programming practice.
- Testing and reasoning about programs is much easier.
  - QuickCheck can be used to check whether a function satisfies a particular property.

## Why Haskell for CS 1JC3?

- Is a well-established functional programming language.
- Is highly accessible and easy to use.
- Is purely functional.
- Has a sophisticated static type system.
- Has an interpreter (i.e., GHCi).