Haskell: Bird’s eye view

Goals

- Overview of Haskell (Language and working environment)
- Using haskell code (In the REPL, from source file and IDEs)
- Overview of expressions, functions and values
- Overview of syntax and conventions
- Introduction to some basic functions

Mbote, Haskell

Haskell Feature

Purely functional

- Every function in Haskell is a function in the mathematical sense (i.e., "pure").
- Functions are first class values
- There are no statements or instructions, only expressions and values.
- Expressions cannot mutate variables, in fact variables are immutables in Haskell

—| *Examples:*

Statically typed

- Every expression in Haskell has a type which is determined at compile time.
- All the types composed together by function application have to match up.
- Types are not only a form of guarantee, but also a language for expressing the construction of programs

—| *Examples:*

Type inference

- You don't have to explicitly write out every type in a Haskell program
- We do not recommend this especially at a beginner level: Write your types.
- Types will be inferred by unifying every type bidirectionally.

—| *Examples:*

Lazy

- Haskell delays evaluation of any calculation as long as possible
- Most expression are reduced or evaluated only when necessary
- Functions don't evaluate their arguments. This means that programs can compose together very well,

—| *Examples:*

Concurrent & Parallel

- Haskell lends itself well to concurrent programming due to its explicit handling of effects.
- A high-performance parallel garbage collector
- A light-weight concurrency library containing a number of useful concurrency primitives and abstractions.cpu id limit max

Installing the Glasgow Haskell Compiler

- GHC is a state-of-the-art, open source, compiler and interactive environment for the functional programming language Haskell.
- You can find the installation instructions for operating systems online at: *https://www.haskell.org/downloads/*
- A word on other compilers

—| *Practice:*

What is the REPL

- REPL stands for Read-Eval-Print_Loop.
- A REPL is an interactive programming environment
- You can input your code, have it evaluated and see the result.

—| *Practice:*

What is the Prelude

- The **Prelude** is a module that contains a small set of standard definitions (Functions / values)
- Preludes' functions are automatically loaded into the REPL.
- This configuration can be switched off.

—| *Practice:*

Interacting with Haskell Code

- Starting the REPL
- Playing with the Prelude
- Evaluating expressions

—| *Practice:*

Using the REPL

- Opening your terminal and launching the REPL
- Issuing command and visualizing its effect
- Exiting the REPL

—| *Practice:*

GHCi core commands

- The :quit command (q)
- The :info command (:i)
- The :doc command (:doc)
- The :type command (:t)

- The :load command (:l)
- The :reload command (:r)
- The :main command (:main)
- The :module command (:m)
- The :set prompt command (:set prompt [*new prompt name*])

—| *Practice:*

Working from source files

- Naming conventions and extension
- The anatomy of the code in the file
- Compiling and executing the code in the file

—| *Practice:*

Using IDEs

- Definition
- Why consider them
- Which IDE for haskell?

—| *Practice:*

Understanding Expressions

- In Haskell, everything is either an expression, a declaration or a value
- Expressions are combinations of values and functions, returning a value when evaluated
- The Purpose of an expression is to create a value (with some possible side-effects),
- The term declaration is a process to naming expressions
- A Haskell program can be seen as one big expression made out of smaller ones

—| *Examples:*

Understanding Statements and Instructions

- Most programming languages have statements and instructions.
- “statements or instructions” are standalone units of execution and don't return anything.
- The sole purpose of a statement is to have side-effects.

—| *Examples:*

Functions (definition, example, characteristics)

- Expressions are the most basic unit of a Haskell program
- Functions are a specific type of expression.
- Functions in Haskell are related to functions in mathematics
- A function is an expression that is applied to an argument and always returns a result
- Functions will always evaluate to the same result when given the same values **overtime**
- *All functions in Haskell take one argument and return one result*

—| *Examples:*

Evaluation

- When we talk about evaluating an expression, we're talking about reducing the terms until the expression reaches its simplest form
- Once a term has reached its simplest form, we say that it is irreducible or finished evaluating. Usually, we call this a value

—| *Examples:*

Operators (infix, associativity, precedence, parenthesization)

- Operators are functions in Haskell.
- Function are used in a prefix style by default
- Operator can be used with an infix style

—| *Examples:*

Let and where constructs

Let and where are used in Haskell to introduce values and expressions.

- let introduces an expression
- where is simply a declaration

—| *Examples:*

Declaration of Values and Functions

In a source file

- In Haskell, the order of declarations does not matter when working from a source file
- Haskell Compiler loads the entire file at once and is aware of all values and functions present in it
- Values or functions can appears at the button of the file meanwhile they are used or referenced in upper part of the code

In the REPL

—| *Examples:*

Arithmetic functions

Below is a list of common operators and functions for arithmetic.

- The operators below are part of the prelude
- Their source code or definition can be found in corresponding modules / libraries
- Operators are just functions

Operator	Name	Purpose/application
+	plus	addition
-	minus	subtraction
*	asterisk	multiplication
/	slash	fractional division
div	divide	integral division, round down
mod	modulo	like ‘rem’, but after modular division
quot	quotient	integral division, round towards zero
rem	remainder	remainder after division

wada

Definitions

Argument: A value or and input a function is applied to

Parameter: A placeholder for argument. They exit at the definition level of a function

Expression: A combination of symbols that conforms to syntactic rules and can be evaluated to some result

Value: A value is an expression that cannot be reduced or evaluated any further

Function: A relationship between two sets with the only condition that each element from the starting set must most be in relation to at most one element in the arrival set.

Infix notation: The operator is placed in between the operands

Syntactic sugar: syntax within a programming language designed to make expressions easier to write or read.

Homework & More Resources

https://github.com/WADAlliance/Haskell_Plutus_Course/tree/main/Getting_Started/005_Practice_Exercises

References:

- Christopher Alan & Julie book: Learn Haskell from first principal
- Scott Wlaschin: Fun For Profit: https://fsharpforfunandprofit.com/
- Haskell packages reference: https://hackage.haskell.org/
- Haskell website: https://www.haskell.org/
- Haskell platform tool kits: https://www.haskell.org/downloads/
- List of GHCI commands: https://typeclasses.com/ghci/commands