wada

Basic Types

Goals

- Learn about built-in datatypes, type constructors, and data constructors;
- Work with predefined datatypes and introduce libraries and modules containing them;
- Learn more about type signatures, a bit about typeclasses and constraints.

Types: What & Why?

- Haskell has a robust and expressive type system. Types play 3 fundamental key roles in Haskell programs / code:
    - Readability
    - Safety
    - Maintainability
- Expression in haskell reduces to values when evaluated. Every value is Haskell has a type
- Types are a great tool to group, classify, organize and delimit data that share common characteristics

—| *Practice:*

Anatomy of data declaration

Data declaration is the process of defining data that is used in our program. Data declaration comprises 2 concepts:

- ***The type constructor****, which* is the name of the type being defined and is capitalized.
- ***Data constructors***, which are the values that inhabit the type they are being defined in.
- Type constructors are used at **type level**, and data constructors at **term level**.
    - Type level: Functions signatures, Values Declarations, used at compile time
    - Term level: Evaluated expressions, used at run time

—| *Example 1:*

wada

Data definition for Bool

```
-- the definition of Bool
data Bool = False | True
--    [1]    [2] [3] [4]
```

[1] *Type constructor* for the datatype Bool being defined. The type constructor is the name of the type being defined

- Type constructor (type names) show up at type level (type signatures)

[2] **False**: *Data constructor* for one of the two possible values of the datatype Bool being defined

- This is use at term level

[3] The pipe **|** is a ***sum type indicator.*** It is also called a logical disjunction **or** in mathematics. It refers to algebraic datatypes Sum. Product types exist as well.

[4] **True** *Data constructor* for one of the two possible values of the datatype Bool being defined

- This is use at term level

—| *Example 2:*

- *Data definition for the type Car*
- *Data definition for the type Person*
- *Data definition for the type LostItem*

—| *Practice:*

```
λ>  :t not
    not :: Bool -> Bool
```

```
λ> not True
   False
```

—| *Example:*

- *Data definition for the type LostItemStatus*
- *Simple function to check the Status of a LostItem (**itemStautus**)*
- *Evaluate the **itemStatus** function*

Numeric types

In Haskell numeric type are organized in 2 core groups: ***Integral*** and ***fractional*** numbers

Integral:

Integral numbers are whole numbers with no fractional component or decimal part. They can be positive or negative. Under this group exits the following types:

- **Int**: This type has a well defined range that comes with a minimum and a maximum value.
- **Integer**: It is used to define arbitrary large or small numbers. It does not have minimum and maximum values like Int

Fractional:

Fractional numbers are used to represent a part of a whole. They come with a whole and a decimal part. This group comprises:

- **Float**: The type is used to represent number with decimal place
- **Double**: The type is used to represent number with decimal place with a need to have move precision
- **Rational**: This is a fractional number that represents a ratio of two integers. The value 1 / 2
- **Scientific**: This is a space efficient and almost arbitrary precision scientific number type. Scientific numbers are represented using scientific notation

—| *Practice 1:*

- *Int*

- *Integer*

—| *Practice 2*

- *Float*
- *Double*
- *Rational*
- *Scientific*

—| *Practice 3*

- *Introducing Type class constraints*
- *Factional and Num type classes*
- *Introduction to subclasses (Num and Fractional)*

Characters

The type Char is

Bool

The type Bool is part of the standard prelude. It is a sum type as opposed to product types. It is often used to create conditions and control how the program behaves when certain things happen.

λ> :info Bool

```
data Bool = False | True
```

—| *Practice 1:*

- *Referring to a type*
- *Data constructor arguments*
- *Data constructor need to be Capitalized*

—| *Practice 2:*

- *Introducing Sum types*
- *Querying the type of a Value*
- *&& and || with the Bool type*

—| *Practice 3::*

- *The if – then – else construct*
- *Logic and Boolean Algebra*
- *Comparing values*

Tuples

Tuples are types that allow storing and passing of values with different types within a single value namely tuple. Tuple arity refers to the number of values a tuple can hold. Tuples are said to be heterogeneous.

λ> :info (,)

```
data (,) a b = (,) a b
```

—| *Practice 1:*

- *A different anatomy type constructors*
- *Two type variables **a** and **b** in the type constructor name*
- *Introducing product types*

Lists

Lists are types used to contain multiple values within one. Unlike tuples, lists are homogenous in type, meaning the values they contain have the same type

λ> :info [ ]

```
data  [ ]  a = [ ]  | a : [a]
```

—| *Practice 1:*

- *Another anatomy for type constructors*
- *One type variable **a** in the type constructor name*
- *Introducing recursive types*

—| *Practice 2:*

- *Defining list values*

- *Introducing String type as list of Characters (Char) types*
- *Introducing list of lists*

—| *Practice 2:*

- *Basic lists functions (head, last, tail, init, length, elem, (!!), sum, isPalindrome)*
- *Combining lists (concat)*
- *Average on lists with integral values*

Strings

The type String in Haskell is syntactic sugar for a list of characters.

λ> :info String

type String = [Char]

—| *Practice 1:*

- *Another keyword* ***(type)*** *for type constructors*
- *String as list of characters*
- *String as syntactic sugar for [Char]*

Printing strings

—| *Practice:*

- *Introduction to IO vs Evaluating*
- *Printing String with* ***print***
- *Printing Strings with* ***PutStr*** *and* ***PutStrLn***

Scoping

In programming, scoping is the process of defining where variables, functions or expressions can be accessed or referenced. This is done in Haskell with a few concepts:

- Modules
- Top level definition
- Local definition

—| *Practice 1:*

- *Introduction to modules*
- *Being at the top level*

- *Being at the local level*

Concatenating strings

Concatenation, in the context of programming, is the operation of joining or merging two strings together.

—| *Practice:*

- *Concatenation functions:* ***(++) , concat***
- *Generalizing concatenation*
- *A minute on the type of the function concat,* ***:type concat***

More List functions

- singleton
- take
- drop
- null
- map
- reverse
- intersperse
- intercalate
- transpose
- subsequence
- permutation
- and
- or
- any
- all
- maximum
- minimum
- ..etc

—| *Practice:*

- *Checking the types from the REPL*
- *Running some examples*
- *Introduction to* ***Foldable***

Naming entities in the Haskell Language

In haskell, there are seven core categories of entities that have names.

Names

- Functions
- Term level variables
- Data constructors
- Type constructors
- Type variables
- Typeclasses
- Modules

—| *Practice:*

wada

Definition

Tuple: A tuple is an ordered grouping of values

Typeclass: is a set of operations defined with respect to

a polymorphic type

Data constructor: They provide a means of creating

values that inhabit a given type

Type constructor: A name given to a type being defined

Data declaration: They define new datatypes in Haskell

Type alias: is a way to refer to a type constructor or type

constant by an alternate name

Arity: is the number of arguments a function accepts

Polymorphism: in Haskell means being able to write code in terms of values which may be one of several, or any, type.

Value: a value is the result of evaluating an expression

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