**Haskell**

* Haskell is a **Functional Programming Language** that has been specially designed to handle **symbolic computation and list processing** applications.
* Functional programming is based on **mathematical functions.**
* In conventional programming, instructions are taken as **a set of declarations in a specific syntax or format**, but in the case of functional programming, all the **computation is considered as a combination of separate mathematical functions.**
  + **Functional Language** - in Haskell we will tell our computer **"what it is?"**
  + **Laziness** − Haskell is a **lazy language**.When the evaluation engine finds that an expression needs to be evaluated, then it creates **a thunk data structure** to collect all the required information for that specific evaluation and **a pointer to that thunk data structure.**
  + **Modularity −** A Haskell application is nothing but a series of functions.
  + **Statically Typed −** Haskell is a type interference language, Haskell compiler is intelligent enough to figure out the type of the variable declared, hence we need not explicitly mention the type of the variable used.
  + **Maintainability −** Haskell applications are modular and hence, it is very **easy and cost-effective** to maintain them.
* Functional programs are more **concurrent and they follow parallelism in execution** to provide more **accurate and better performance.**
* **Haskell is no exception; it has been developed in a way to handle multithreading effectively.**
* Haskell is intelligent enough to **decode some number as a number**.
* Therefore, you need not **mention its type externally** as we usually do in case of other programming languages.
* Haskell can intelligently identify a **character** given in as an input to it.
* Haskell follows conventional **ASCII encoding style.**
* A **string** is nothing but a **collection of characters.** There is **no specific syntax** for using string, but Haskell follows the conventional style of representing a **string with double quotation.**
* **List** is a collection of **same data type separated by comma**.
* **List comprehension** is the process of generating **a list using mathematical expression.**
* The method of creating one List using mathematical expression is called as **List Comprehension.**
* Haskell provides another way to **declare multiple values** in a single data type. It is known as **Tuple**.
* A **Tuple** can be considered as a **List**, however there are some technical differences between a Tuple and a List.
  + A **Tuple** is an **immutable data type**, as we **c**annot modify the number of elements at runtime, whereas a **List is a mutable data type.**
  + On the other hand, **List is a homogeneous data type**, but **Tuple is heterogeneous in nature**, because a Tuple may contain different type of data inside it.
  + Tuples are represented by **single parenthesis.**
* **Sequence or Range** is a special operator in Haskell. It is denoted by **"(..)".** You can use this operator while **declaring a list with a sequence of values.**
* In Haskell, every statement is considered as a **mathematical expression** and the category of this expression is called as a **Type**.
* **Type** can be considered as a **value,** whereas **Type Class** can be the considered as a **set of similar kind of Types.**
* **EQ type class** is an interface which provides the functionality to test the equality of an expression. Any Type class that wants to **check the equality of an expression** should be a part of this EQ Type Class.
* **Ord** is another interface class which gives us the **functionality of ordering.**
* **Ord interface** can be called using **">", "<", "<=", ">=", "compare".**
* **Show** has a functionality to print its argument as a **String**. Whatever may be its argument, it always **prints the result as a String.**
* **Read interface** does the same thing as Show, but it **won’t print the result in String format.**
* **Enum** is another type of Type class which enables the **sequential or ordered functionality in Haskell.** This Type class can be accessed by commands such as **Succ, Pred, Bool, Char, etc.**
* **Bounded** - All the types having **upper and lower bounds** come under this Type Class.
* **Num** - This type class is used for **numeric operations.** Types such as **Int, Integer, Float, and Double** come under this Type class.
* **Integral** can be considered as a **sub-class of the Num Type Class**. Num Type class holds all types of numbers, whereas Integral type class is used only for integral numbers. Int and Integer are the types under this Type class.
* Floating is also a part of the Num Type class, but it only holds floating point numbers. Hence, Float and Double come under this type class.
* Haskell allows developers to define user-defined types, all user-defined types in Haskell always start with a capital letter.
* Pattern Matching is process of matching specific type of expressions. It is nothing but a technique to simplify your code. This technique can be implemented into any type of Type class. If-Else can be used as an alternate option of pattern matching.
* Pattern Matching can be considered as a variant of dynamic polymorphism where at runtime, different methods can be executed depending on their argument list.
* Guards is a concept that is very similar to pattern matching. In pattern matching, we usually match one or more expressions, but we use guards to test some property of an expression.