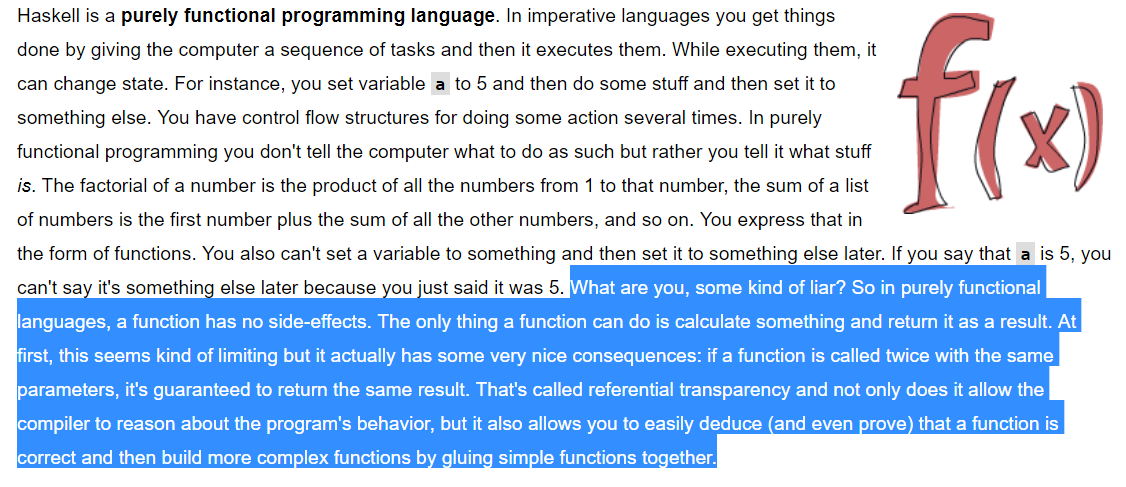
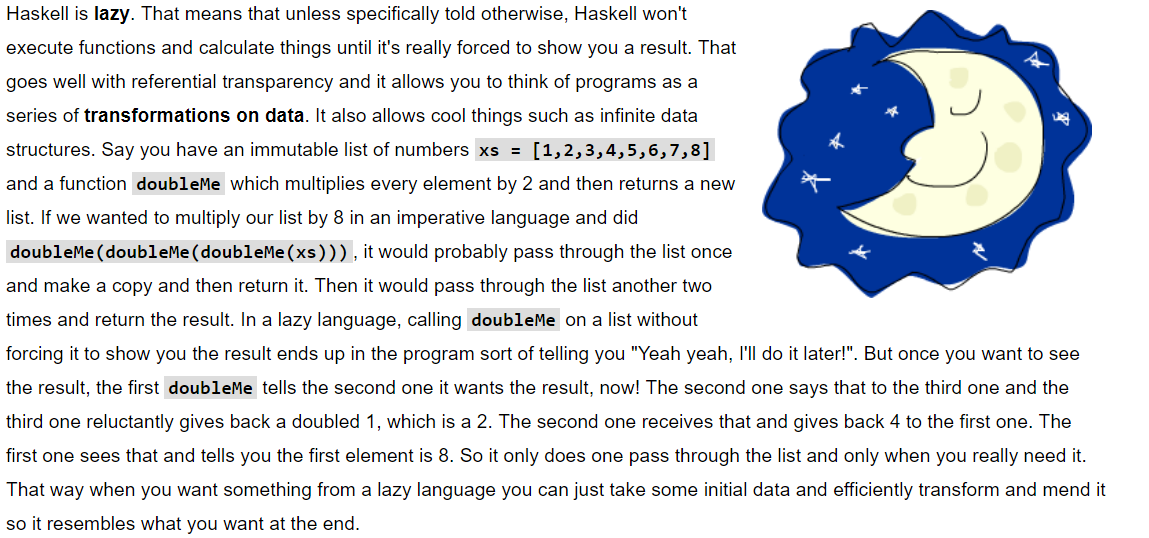
**Introduction:**

**So, what's Haskell?**

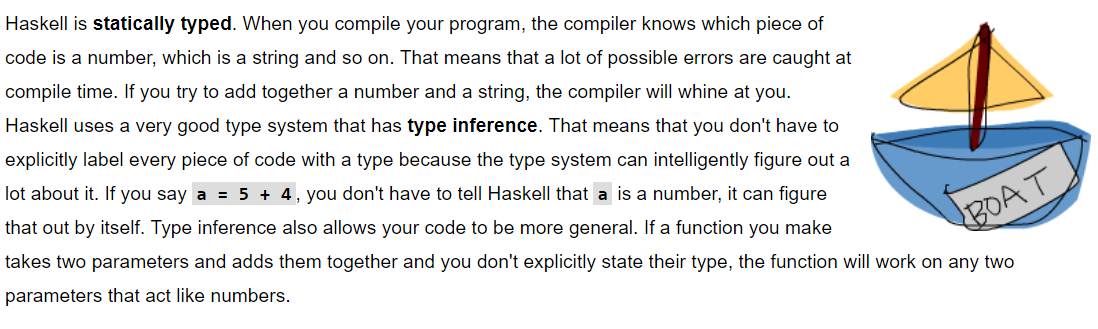
**Purely functional programming language:**



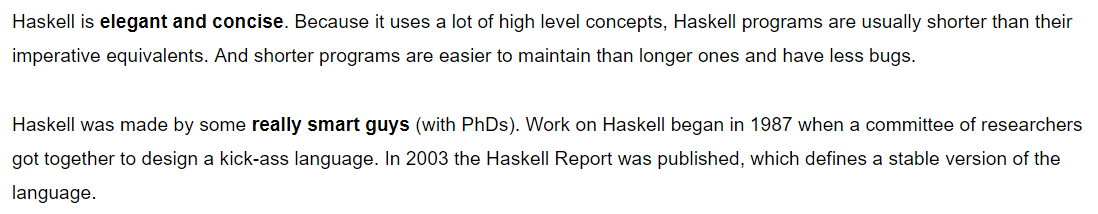
**Lazy language:**



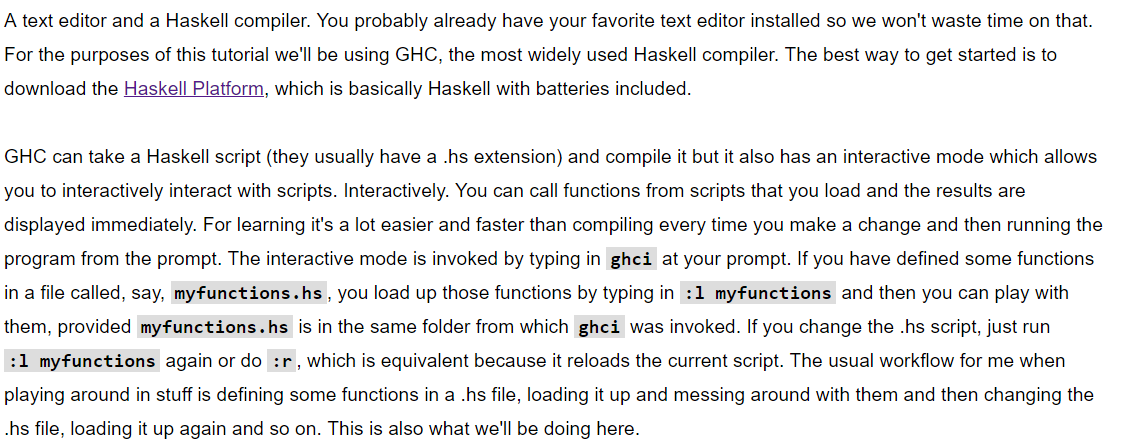
**Statically typed:**



**Special language:**



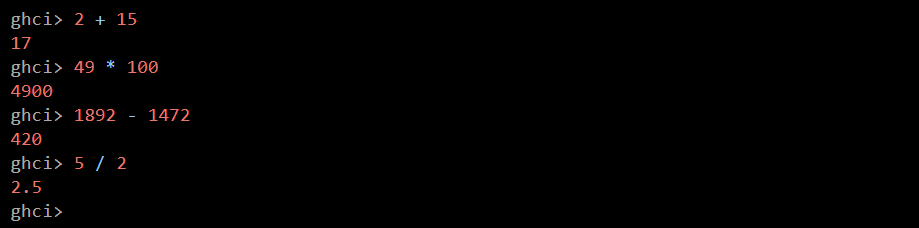
**Install Platform:**



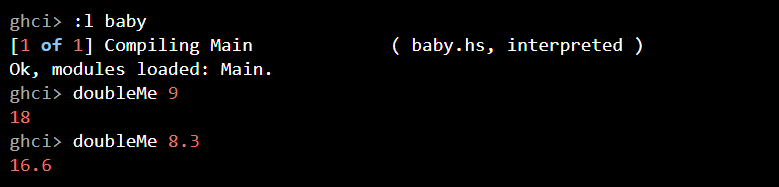
Install Haskell platform from here:<https://www.haskell.org/platform/>

**Language Syntax and basics:**

* Everything is a function so \* and + etc. is a function that called in-fix and takes two parameters like 1 + 2. Functions is a first-class object.
* Like in Python we can use the interactive mode to learn Haskell quickly by open terminal and type **ghci**:



* Also, we can write a Haskell script and execute it and call its functions like show below:



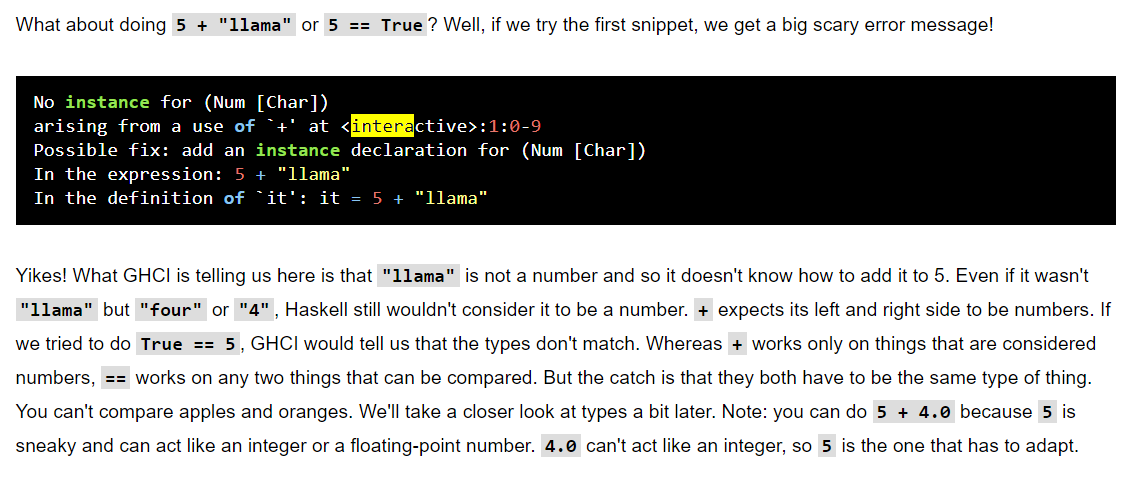
**What we do is:**

1. Create a file called **baby.hs**
2. Put this function inside the file doubleMe x = x + x
3. Save the file.
4. Now we can load our script by typing **:l baby**
5. And we can call any script’s function.

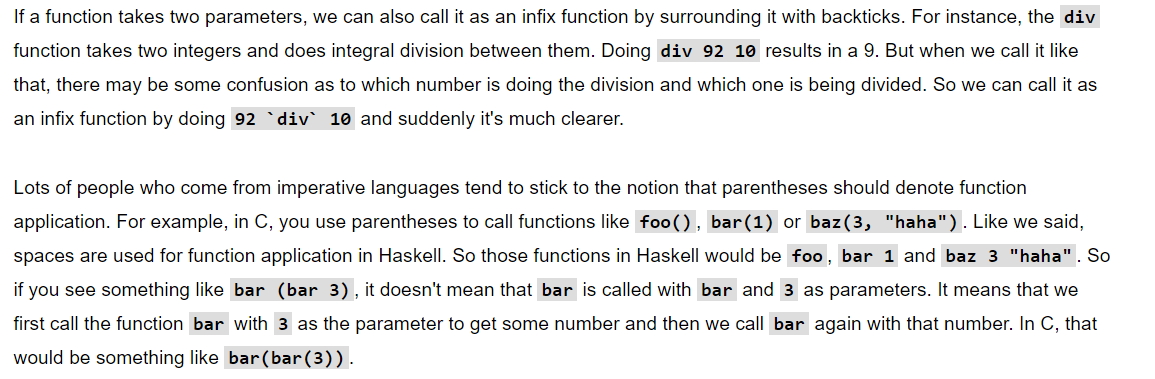
* Doing **5 \* -3** will make GHCI yell at you but doing **5 \* (-3)** will work just fine.
* Boolean algebra is also straightforward. As you probably know, **&&** means a Boolean *and*, **||** means a Boolean *or*. **Not** negates a **True** or a **False**.



* Casting:

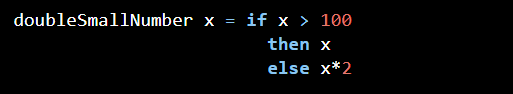


* Functions can be called pre-fix which is the default, also it can called in-fix and post-fix:



* Samples of functions declaration:





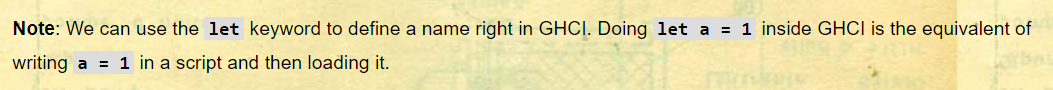




* If then else: The difference between Haskell's if statement and if statements in imperative languages is that the else part is mandatory in Haskell. In imperative languages, you can just skip a couple of steps if the condition isn't satisfied but in Haskell every expression and function must return something.
* Expressions: Another thing about the if statement in Haskell is that it is an *expression*. An expression is basically a piece of code that returns a value. **5** is an expression because it returns 5, **4 + 8** is an expression, **x + y** is an expression because it returns the sum of **x** and **y**.
* **'** at the end of the function name. That apostrophe doesn't have any special meaning in Haskell's syntax. It's a valid character to use in a function name. We usually use **'** to either denote a strict version of a function (one that isn't lazy) or a slightly modified version of a function or a variable.

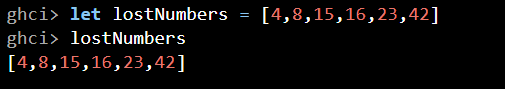
**Variables:**

* We can declare variable using let keyword or without it.

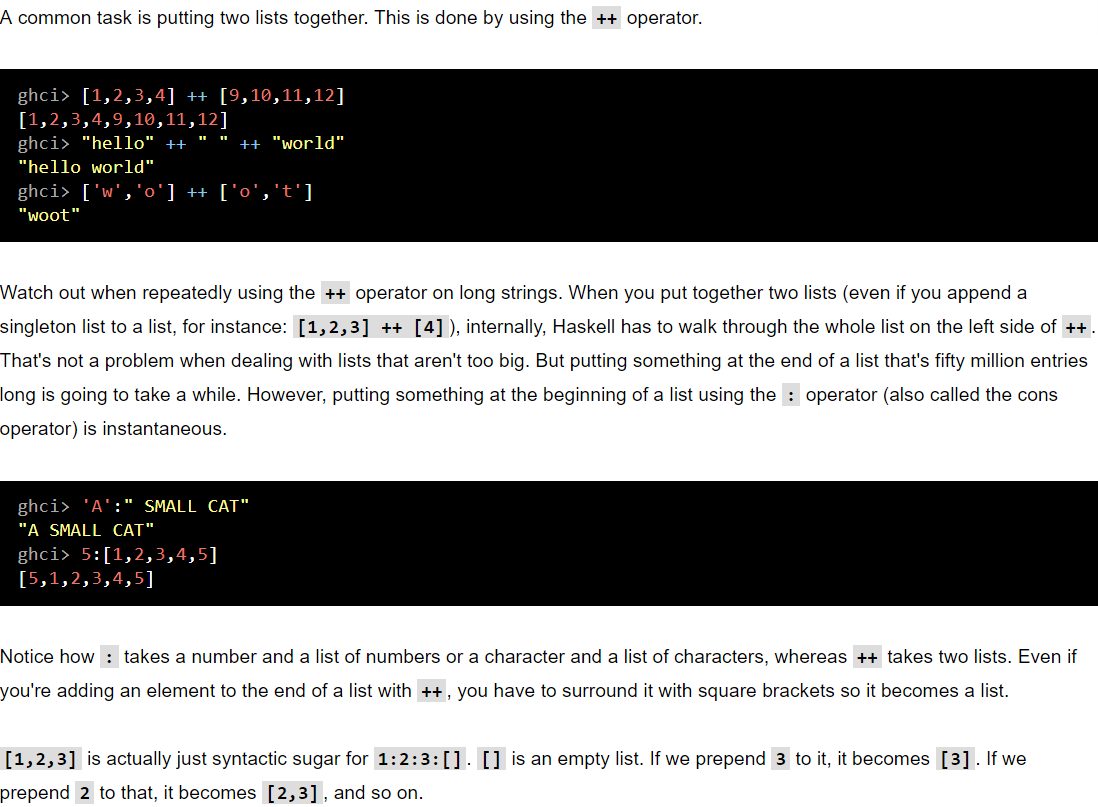


**Lists:**

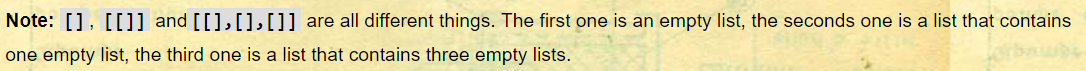
* In Haskell, lists are a **homogenous** data structure. It stores several elements of the same type. That means that we can have a list of integers or a list of characters but we can't have a list that has a few integers and then a few characters.



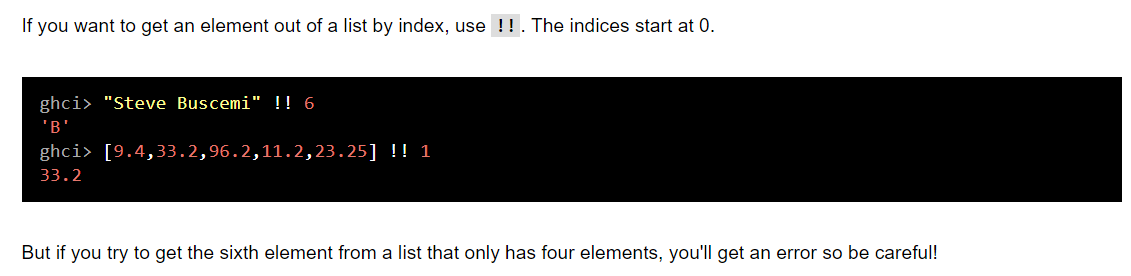
* peaking of characters, strings are just lists of characters. **"hello"** is just syntactic sugar for **['h','e','l','l','o']**. Because strings are lists, we can use list functions on them, which is really handy.
* String is a list of characters in Haskell
* ++ Operator and : Operator:



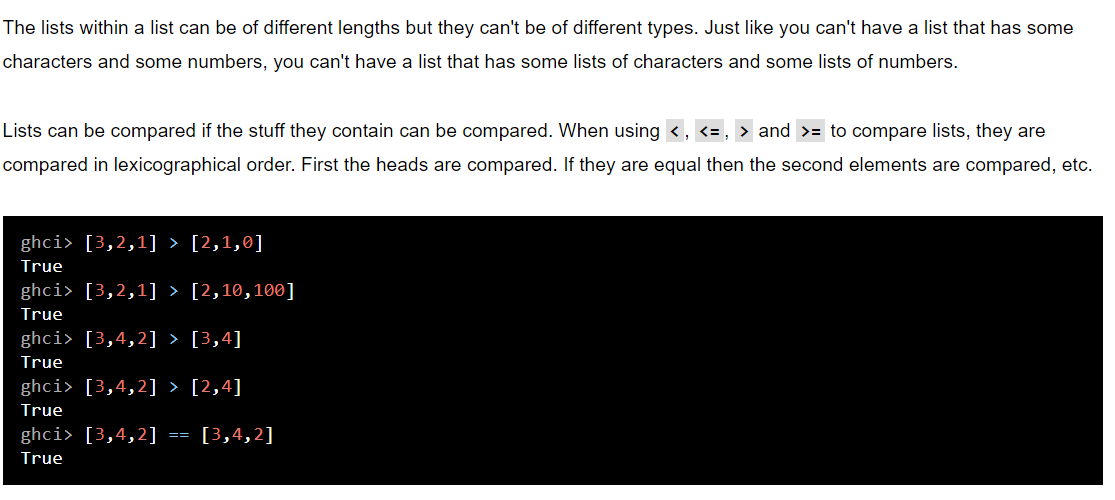
* List of list of list …



* !! operator:



Compare lists:



* Some basic operations:

