An advanced, purely functional programming language

November 2016

In The Name Of God

An introduction to Haskell programing language

Islamic Azad University Central Tehran Branch

November 2016

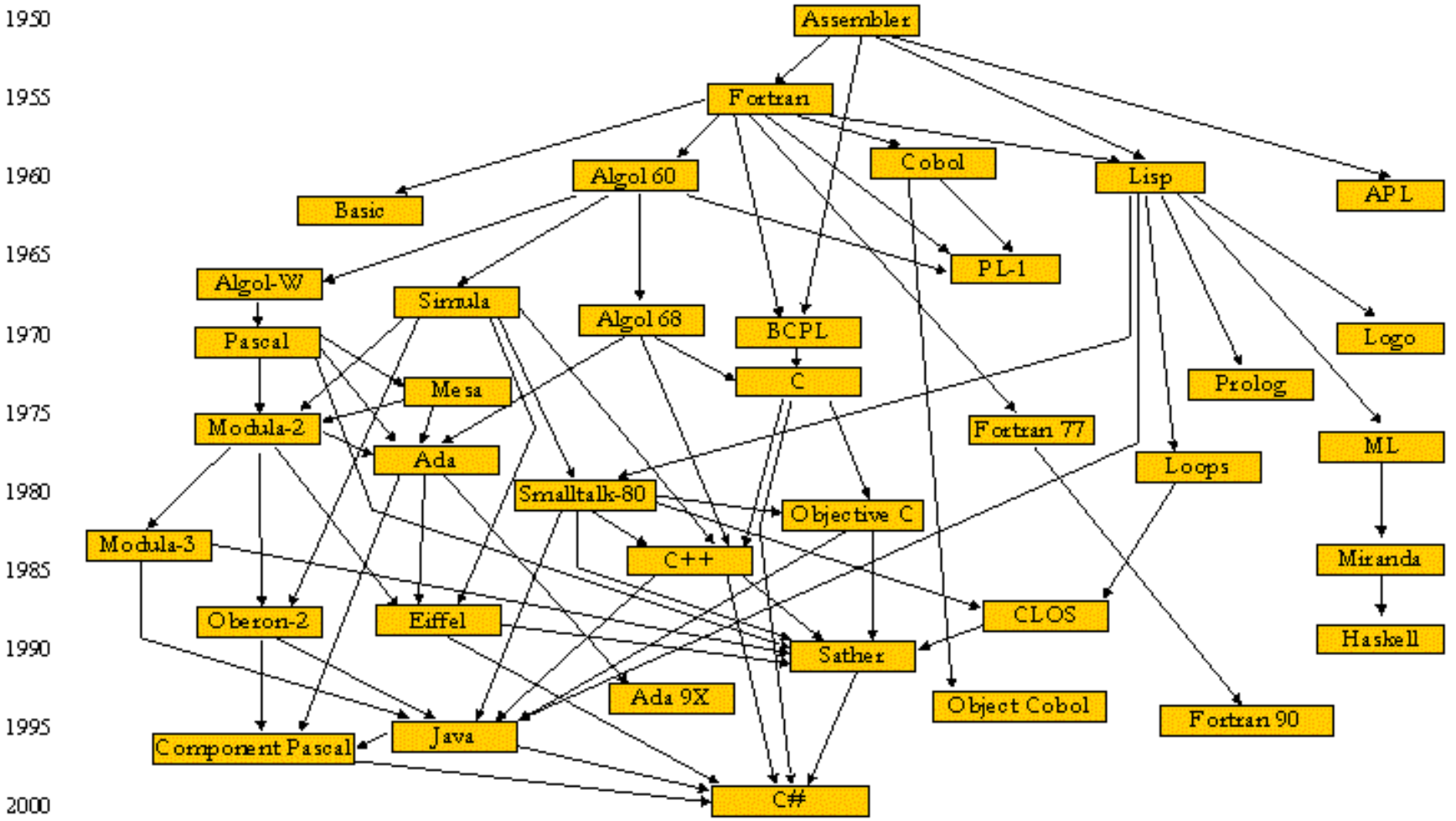
V.1.0

History

History

- 1987 : Following the release of **Miranda** , interest in **lazy functional languages** grew. At the conference on Functional Programming Languages and Computer Architecture (FPCA '87) , a meeting was held during which participants formed a strong consensus that a committee should be formed to define an **open standard** for such languages.
- 1990 : The first version of Haskell ("Haskell 1.0") was defined in 1990.
- 1998-1999 : The Haskell 98 language standard was originally published as *The Haskell 98 Report*.
- 2010 : In early 2006, the process of defining a successor to the Haskell 98 standard, informally named *Haskell Prime*, began. Haskell 2010 adds the **foreign function interface** (FFI) to Haskell, allowing for bindings to other programming languages, fixes some **syntax** issues.

Programming Language Family Tree



Haskell is influenced by:

- ISWIM
- Lisp
- ML
- Scheme
- Miranda

And Influenced:

- Python
- Rust
- Scala
- Swift
- Visual Basic 9.0
- F#
- Java/Generics
- Perl 6
- C++11/Concepts
- C#/LINQ
- Clean
- CoffeeScript
- Hack
- Idris

Introduction

Introduce

Haskell /ˈhæskəl/ is a standardized, general-purpose purely functional programming language, with non-strict semantics and strong static typing. It is named after logician Haskell Curry.

The latest standard of Haskell is **Haskell 2010**. As of May 2016, a group is working on the next version, **Haskell 2020**.

Haskell features a type system with type inference and lazy evaluation. Type classes first appeared in the Haskell programming language. Its main implementation is the Glasgow Haskell Compiler.

Haskell is based on the semantics, but not the syntax, of the language Miranda, which served to focus the efforts of the initial Haskell working group. Haskell is used widely in **academia** and also used in **industry**.

Haskell is the **fifth-generation** languages, or **5GL.** In these languages you don’t have to know **how** implement an algorithm to solve problems. Just you should know **what** problems need to be solved and what conditions need to be met.

Introduce

The Foreign Function Interface (FFI)

The Foreign Function Interface (FFI) allows Haskell programs to **cooperate** with programs written with other languages. Haskell programs can **call** foreign functions and **foreign functions** can call **Haskell code**.
Compared to many other languages, Haskell FFI is very **easy** to use

Features

Statically typed

Having a *static* type system means that the compiler knows the type of every value and expression **at compile time**, before any code is **executed**. (not at **run time**)
A Haskell compiler or interpreter will detect when we try to use expressions whose types **don't match**, and reject our code with an **error** message before we run it:

```
ghci> True && "false"
<interactive>:1:8:
Couldn't match expected type `Bool' against inferred type
`[Char]'
In the second argument of `(&&)', namely `"false"'
In the expression: True && "false"
In the definition of `it': it = True && "false"
```

Some Common Basic Types

A **Char** *value*
Represents a Unicode character.

A **Bool** *value*
Represents a value in Boolean logic. The possible values of type Bool are True and False.

The **Int** *type*
Used for signed, fixed-width integer values. The exact range of values represented as Int depends on the system's longest "native" integer: on a 32-bit machine, an Int is usually 32 bits wide, while on a 64-bit machine, it is usually 64 bits wide.

An **Integer** *value*
A signed integer of unbounded size. Integers are not used as often as Ints, because they are more expensive both in performance and space consumption. On the other hand, Integer computations do not silently overflow, so they give more reliably correct answers.

Values of type **Double**
Used for floating-point numbers. A Double value is typically 64 bits wide and uses the system's native floating-point representation. (A narrower type, **Float**, also exists, but its use is discouraged; Haskell compiler writers concentrate more on making Double efficient, so **Float** is much slower.)

Statically typed

Useful Composite Data Types

Lists

A list is surrounded by square brackets; the elements are separated by commas:

```
>       X = [1, 2, 3]
```

All elements of a list must be of the **same type**. Here, we violate this rule. Our list starts with two Bool values, but ends with a string:

```
>       X = [True, False, "testing"]          error
```

If we write a series of elements using *enumeration notation*, Haskell will fill in the contents of the list for us:

```
[1..10] ==  [1,2,3,4,5,6,7,8,9,10]
```

Operators on Lists:
There are two ubiquitous operators for working with lists.

(++) operator:

```
ghci> [3,1,3] ++ [3,7]
[3,1,3,3,7]
```

(:) operator:

```
ghci> 1 : []
[1]
```

Statically typed

Useful Composite Data Types

tuple

A tuple is a **fixed-size collection** of values, where each value can have a **different type**. This distinguishes them from a list, which can have any length, but whose elements must all have the same type.

```
>      X = (True, 1, "testing")          OK!
```

Our Own Types

Let's see how the **Bool** type is defined in the standard library.

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

make our own type to represent a shape! (rectangle or circle)

```
data Shape = Circle Float Float Float | Rectangle Float Float Float Float
```

Statically typed

Type inference

A Haskell compiler can **automatically** deduce the types of almost† all expressions in a program. This process is known as *type inference*.

Haskell allows us to explicitly declare the type of any value, but the presence of type inference means that this is almost always optional, not something we are required to do.

A. This example has a type signature for every binding:

```
main :: IO ()
main = do line :: String <- getLine
          print (parseDigit line)
  where parseDigit :: String -> Maybe Int
        parseDigit ((c :: Char) : _) =
          if isDigit c
             then Just (ord c - ord '0')
             else Nothing
```

B. You can just write:

```
main = do line <- getLine
          print (parseDigit line)
  where parseDigit (c : _) =
          if isDigit c
             then Just (ord c - ord '0')
             else Nothing
```

Purely functional

Every function in Haskell is a function in the mathematical sense (i.e., "pure"). Even side-effecting IO operations are but a description of what to do, produced by pure code. There are no statements or instructions, only expressions which cannot mutate variables (local or global) nor access state like time or random numbers.

Pure means *no* ***side effects***
A function will **never modify** a *global variable*
Order doesn't matter!

For example, if we run the following tiny **Python** script, it will print the number 11:

```
x = 10
x = 11
# value of x is now 11
print x
Result: OK
```

In contrast, trying the equivalent in **Haskell** results in an error:

```
x = 10
x = 11
Result: error
```

Lazy

In a language that uses *strict* evaluation, the arguments to a function are evaluated before the function is applied. Haskell chooses another path: ***nonstrict*** **evaluation**.

```
F = 1 + 2
```

In Haskell, the subexpression 1 + 2 is *not* reduced to the value 3. Instead, we create a "**promise**" that when the value of the expression F **is needed**, we'll be able to **compute** it.

Nonstrict evaluation is often referred to as ***lazy evaluation.****

*The terms "nonstrict" and "lazy" have slightly different technical meanings, but we won't go into the details of the distinction here.

Lets go for a test! We have an function that it have to calculate 1 divide 0 (zero)!

Commend: `> let foo = 1/0`

Commend: `> foo`
Result: `> Infinity`

We **don't** have any error or crash!

Concurrent

Haskell lends itself well to concurrent programming due to its explicit handling of effects. Its flagship compiler, GHC, comes with a **high-performance parallel garbage collector** and **light-weight concurrency library** containing a number of useful concurrency primitives and abstractions.

Packages

Open source contribution to Haskell is very active with a wide range of packages available on the public package servers.

Hackage is the Haskell community's central package archive of open source software. Package authors use it to publish their libraries and programs while other Haskell programmers use tools like cabal-install to download and install packages (or people get the packages via their distro).

There are 6,954 packages freely available. You can access they on Hackage website:

Hackage.haskell.org

Cabal is a system for building and packaging Haskell libraries and programs. It defines a common interface for package authors and distributors to easily build their applications in a portable way.

```
$ cabal install cabal cabal-install
$ cabal update
$ cabal install PACKAGE
```

Note: Most people already have Cabal because it is included in the Haskell Platform.

Haskell Use Community

Commercial Use

Internet
Facebook - Haxl rule engine "fighting spam with pure functions"
Biotech
Amgen - informatics, simulation
Finance
Credit Suisse - quantitative modeling
Barclays - DSEL for exotic equity derivatives
Deutsche Bank - trading group infrastructure
Tsuru Capital - trading platform
McGraw-Hill Financial - report generation (with ermine)
Semiconductor Design
Bluespec - high-level language for chip design

Consumer Apps

Silk
"A platform for sharing collections about anything"
Chordify
"Chord transcription for the masses"
Bump (Google, Sep 2013)
"Send files, videos, everything!" Mobile + web.
MailRank (Facebook, Nov 2011)
Email inbox prioritization. Shuttered post-acquisition.
Bazqux
"RSS reader that shows comments to posts"

Commercial Services

janrain
User management platform.
Spaceport (Facebook, Aug 2013)
Mobile game framework using ActionScript 3
scrive
E-signing service (nordic market)
OpenBrain
Computing platform for scientific and business analytics
skedge.me
Enterprise appointment scheduling

Standalone Apps

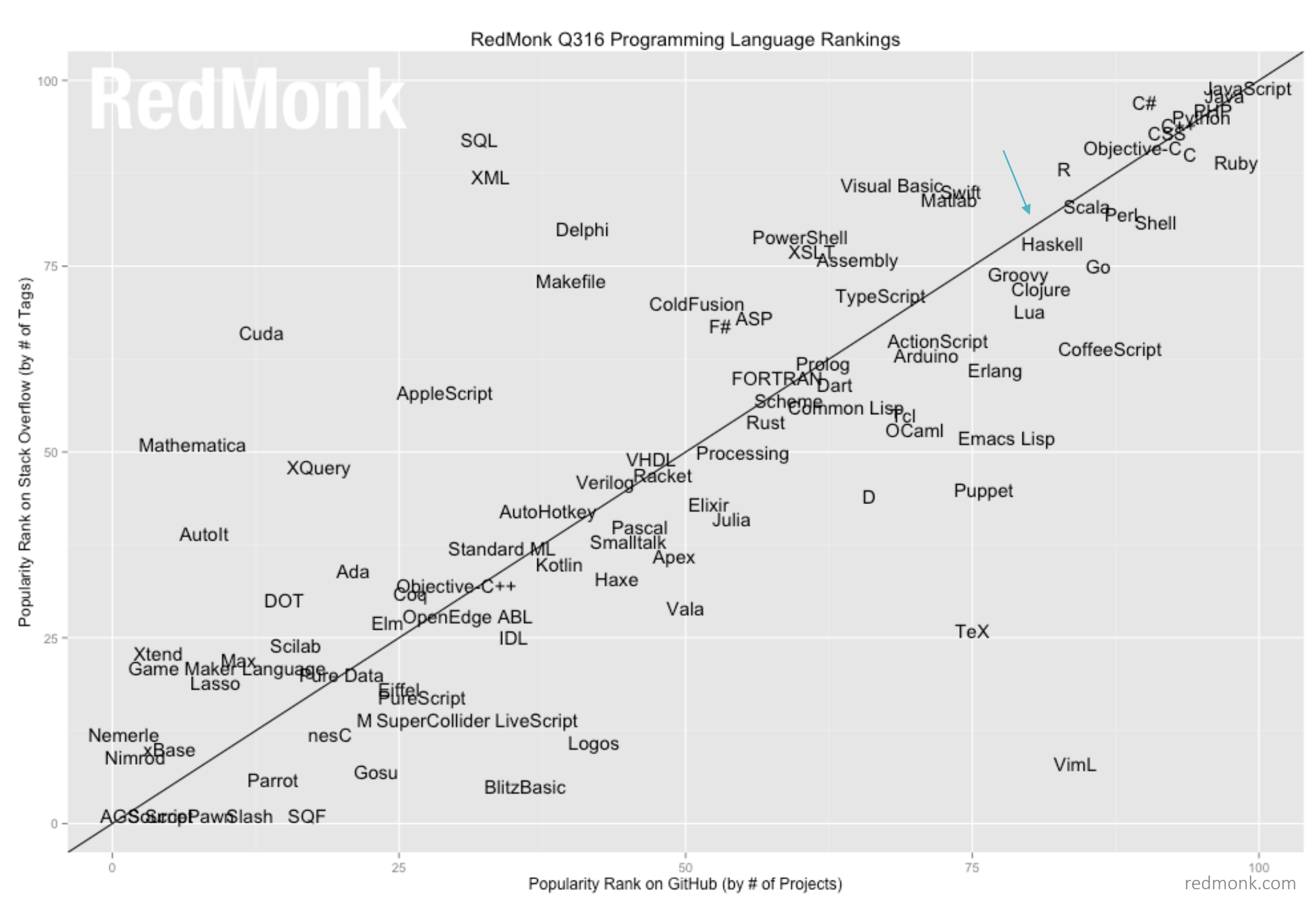
Pandoc
Markup swiss-army knife (used to make these slides!)
Darcs
Distributed revision control system (like Git or Mercurial)
xmonad
"the tiling window manager that rocks"
Gitit
Wiki backed by Git, Darcs, or Mercurial
git-annex
Manage large files with git (similar to Dropbox)
ImplicitCAD
Programmatic Solid 3D CAD modeler

Haskell Facts

RAM footprint per unit of concurrency (approx)

1.3KB	Haskell ThreadId + MVar (GHC 7.6.3, 64-bit)
2.6 KB	Erlang process (64-bit)
4.0 KB	Go goroutine
9.0 KB	C pthread (minimum, 64-bit Mac OS X)
64.0 KB	Java thread stack (minimum)
513 KB	C pthread (default, 64-bit Mac OS X)
1024 KB	Java thread stack (default)

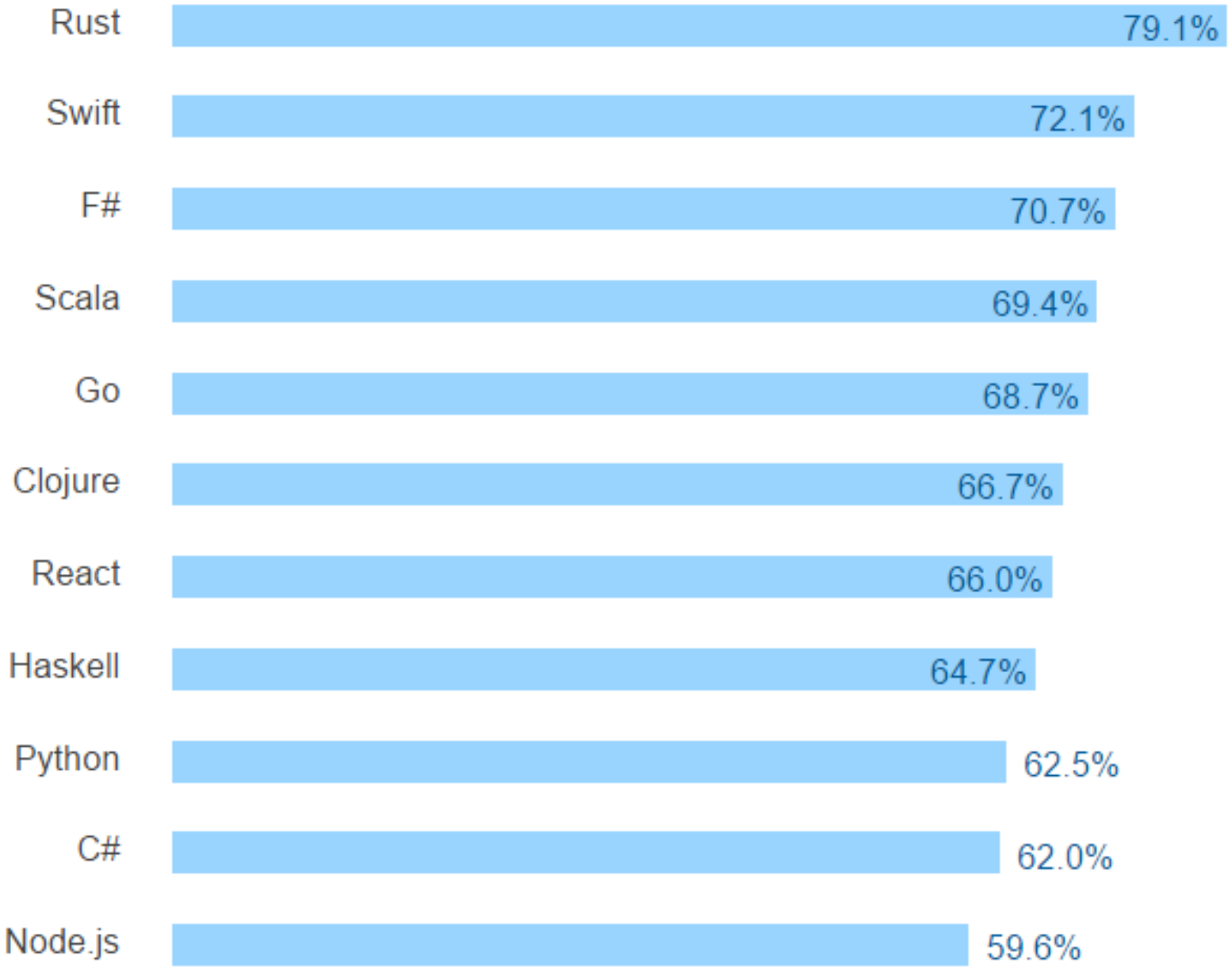
Good support for parallelism and concurrency



redmonk.com

RedMonk Ranking:

1 JavaScript
2 Java
3 PHP
4 Python
5 C#
5 C++
5 Ruby
8 CSS
9 C
10 Objective-C
11 Shell
12 R
13 Perl
14 Scala
15 Go
16 Haskell
17 Swift
18 Matlab
19 Visual Basic
20 Clojure
20 Groovy



Haskell is 8th

stackoverflow

Developer Survey Results

2016

The Most Loved Programing languages

% of developers who are developing with the language or tech and have expressed interest in continuing to develop with it

Haskell Time Performance

C, C++, JAVA < Haskell << Python

Haskell = max **40X** C, C++, JAVA

Python = max **900X** C, C++, JAVA

Implementations

Implementations

Compilers and interpreters :

- 1990 : hbc
- 1991 : Yala Haskell, Gofer
- 1992 : GHC
- 1993- 1995 : PH, nhc, Hugs
- 2001 : GHCi
- Up to Now: Heluim, jhc, yhc, UHC , LHC

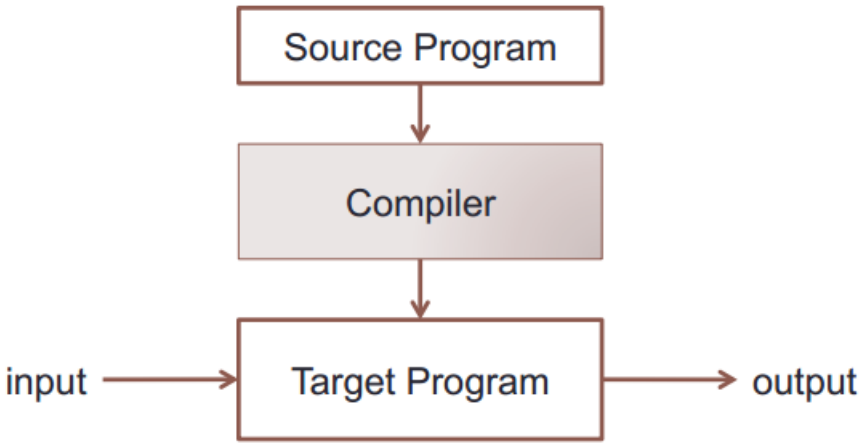
Implementations

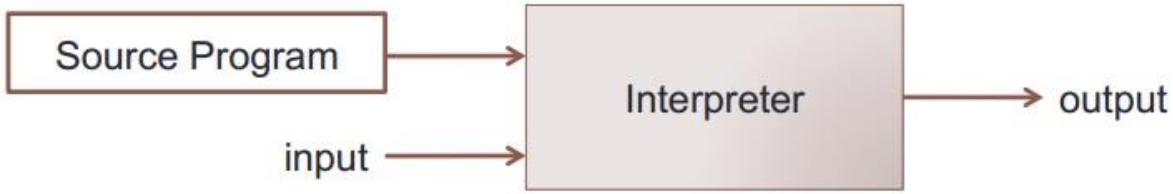
Compiler

- Translates a source program into a target program
- If the target program is an executable machine-language program, it can then be called by the user to process inputs and produce outputs

Interpreter

- Translates a source program to its equivalent machine-language program and immediately executes them.





Haskell 2010

- Glasgow Haskell Compiler (GHC)

  GHC is an optimizing compiler for Haskell, providing many language extensions.
  GHC is the de facto standard compiler if you want fast code.
  GHC is written in Haskell (plus extensions), and its size and complexity mean that it is less portable than Hugs, it runs more slowly, and it needs more memory. However, the programs it produces run *much* faster.

  There is also an interactive environment, GHCi, which is like Hugs but supports interactive loading of compiled code. It is available for most common platforms, including Windows, Mac OS X, and several Unix variants (Linux, *BSD, Solaris).

- Utrecht Haskell Compiler (UHC)

  UHC is a Haskell implementation from Utrecht University. UHC supports almost all Haskell 98 and Haskell 2010 features plus many experimental extensions. The compiler runs on Mac OS X, Windows (Cygwin), and various Unix flavors.
  On April 18, 2009 UHC was announced at the 5th Haskell Hackathon in Utrecht.

- LLVM Haskell Compiler (LHC)

Implementations

Editor Support

IntelliJ
plugin for Haskell
Emacs
ghc-mod + HLint
Vim
hdevtools + Syntastic + vim-hdevtools
Kdevelop

Sublime Text
hdevtools + SublimeHaskell
Leksah
an IDE for Haskell written in Haskell.
Eclipse
EclipseFP + HLint
Web
FP Haskell Center

Lets code!

Sample code

Using recursion:

```
fib 0 = 0
fib 1 = 1
fib n = fib(n-1) + fib(n-2)
```

With self-referencing data:

```
fib n = fibs !! n
   where fibs = 0 : scanl (+) 1 fibs
```

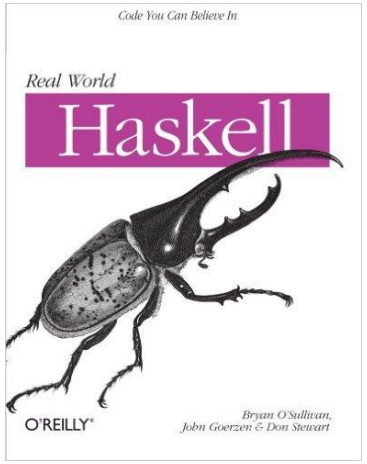
Lets run them to compare the results!

Questions ?

Resources

1. https://wiki.haskell.org
2. bob.ippoli.to
3. http://benchmarksgame.alioth.debian.org
4. http://stackoverflow.com/research/developer-survey-2016

Real World Haskell



Thanks

Mahdi Jedari
i.jedari@gmail.com

IAUCTB, 1 Nov 2016