

# PROGRAMMING IN HASKELL<sup>0</sup>



## Chapter 2 - First Steps

Based on "Programming in Haskell" by Graham Hutton

# Introduction to GHC (Glasgow Haskell Compiler)

1

- **What is GHC?**
  - The leading implementation of Haskell, comprising a compiler and interpreter.
  - Interactive interpreter (*GHCi*) suitable for teaching, prototyping, and testing.
- **Availability:**
  - Freely available at [www.haskell.org](http://www.haskell.org)

GHC

The Glasgow Haskell Compiler



# Starting with GHCi

2

## How to start

```
$ ghci
```

```
GHCi, version X: http://www.haskell.org/ghc/ :? for help
```

```
ghci>
```

GHCi prompt (***ghci>***) indicates the interpreter is ready for expressions

# Starting with GHCi

3

```
> 2 + 3 * 4      -- 14
```

```
> (2 + 3) * 4    -- 20
```

```
> sqrt (3^2 + 4^2) -- 5.0
```

```
[GHCi, version 9.4.8: https://www.haskell.org/ghc/  :? for help
ghci> 2+ 3* 4
[14
ghci> (2+3) * 4
[20
ghci> sqrt (3^2 + 4^2)
[5.0
```



# The Standard Prelude

4

- Library Overview:

- Haskell includes many built-in functions for lists and numbers.

- **Examples:**

- Select the first element:

```
> head [1,2,3,4,5]  
> 1
```

- Remove the first element:

```
> tail [1,2,3,4,5]  
[2,3,4,5]
```

- Sum of a list:.

```
> sum [1,2,3,4,5]  
15
```

# The Standard Prelude

5

- Select the nth element of a list:

```
> [1,2,3,4,5] !! 2  
3
```

- Select the first n elements of a list:

```
> take 3 [1,2,3,4,5]  
[1,2,3]
```

- Remove the first n elements from a list:

```
> drop 3 [1,2,3,4,5]  
[4,5]
```



# The Standard Prelude

6

❑ Calculate the length of a list:

```
> length [1,2,3,4,5]  
5
```

❑ Calculate the sum of a list of numbers:

```
> sum [1,2,3,4,5]  
15
```

❑ Calculate the product of a list of numbers:

```
> product [1,2,3,4,5]  
120
```

# The Standard Prelude

7

□ Append two lists:

```
> [1,2,3] ++ [4,5]  
[1,2,3,4,5]
```

□ Reverse a list:

```
> reverse [1,2,3,4,5]  
[5,4,3,2,1]
```



# Function Application in Haskell

8

- Syntax
  - Function application uses spaces instead of parentheses:

Java:

`f(a,b) + c d`



Haskell:

`f a b + c * d`

Apply the function `f` to `a` and `b`, and add the result to the product of `c` and `d`.

# Function Application in Haskell

9

Moreover, function application is assumed to have higher priority than all other operators.

`f a + b`



Means  $(f\ a) + b$ , rather than  $f\ (a + b)$ .



# Examples

10

## Mathematics

$f(x)$

$f(x, y)$

$f(g(x))$

$f(x, g(y))$

$f(x)g(y)$

## Haskell

$f\ x$

$f\ x\ y$

$f\ (g\ x)$

$f\ x\ (g\ y)$

$f\ x\ * \ g\ y$

# Haskell Scripts

11

- What is a script?
  - Text files containing Haskell code, typically with a .hs extension.
- Example Script:

```
double x = x + x
quadruple x = double (double x)
```
- Save the file as test.hs and load it into GHCi

```
$ ghci test.hs
> quadruple 10 -- 40
```



# Defining Functions

12

- Examples:

- factorial

```
factorial n = product [1..n]
```

- average

```
average ns = sum ns `div` length ns
```

- Use backticks for infix notation: `sum ns `div` length ns`
- `x `f` y` is just syntactic sugar for `f x y`.

# Useful GHCi Commands

13

Command	Description
<code>:load &lt;file&gt;</code>	Load a Haskell script
<code>:reload</code>	Reload the current script
<code>:type &lt;expr&gt;</code>	Show the type of an expression
<code>:quit</code>	Exit GHCi

Tip : use `:q` in GHCi for a list of all commands



# Naming Requirements

14

- Function and argument names must begin with a lower-case letter. For example:

myFun

fun1

arg\_2

x'

- By convention, list arguments usually have an s suffix on their name. For example:

xs

ns

nss

- Strive to write clear, self-explanatory code (see <https://wiki.haskell.org/Commenting>)

```
swap :: (a,b) -> (b,a)
```

- Avoid redundant comments



# Comments in Haskell - Syntax

16

```
-- A one-line comment looks like this
```

```
{- A multiline  
   comment can continue for many lines  
-}
```

# The Layout Rule

17

In a sequence of definitions, each definition must begin in precisely the same column:

```
a = 10
```

```
b = 20
```

```
c = 30
```



```
a = 10
```

```
  b = 20
```

```
c = 30
```



```
a = 10
```

```
b = 20
```

```
c = 30
```





# The Layout Rule

The layout rule avoids the need for explicit syntax to indicate the grouping of definitions.

```
a = b + c
  where
    b = 1
    c = 2
d = a * 2
```



```
a = b + c
  where
    {b = 1;
     c = 2}
d = a * 2
```

implicit grouping

explicit grouping

**Tip:** Align code neatly to avoid syntax errors.

# Recap

19

## •Key Takeaways:

- GHCi is an interactive and powerful environment for Haskell.
- Use the Prelude for built-in functions and utilities.
- Write clean, concise scripts and test them in GHCi.





Any Questions ?