Functional Programming with Haskell

A guided tour through the requirements of the AQA specification.

This document is a cosy workbook for those who like everything in one place! All the information here is presented in other ways on the Computer Science GoL site, usually with more detail and links.

Contents

[Specification Requirements 2](#_Toc467710370)

[Specification Summary 4](#_Toc467710371)

[Learning Resources 5](#_Toc467710372)

[Getting started with Haskell (!IMPORTANT!) 6](#_Toc467710373)

[Installation 6](#_Toc467710374)

[Installing at Home 6](#_Toc467710375)

[Using Haskell at College 6](#_Toc467710376)

[Using Notepad++ 6](#_Toc467710377)

[First Steps REPL 6](#_Toc467710378)

[Hello World 6](#_Toc467710379)

[Simple Operations 6](#_Toc467710380)

[Simple Functions 7](#_Toc467710381)

[First Steps Program Files 8](#_Toc467710382)

[Notepad++ with NPPexec 8](#_Toc467710383)

[Selection 9](#_Toc467710384)

[~~Data~~ Types 10](#_Toc467710385)

[Lists 11](#_Toc467710386)

[Adding to a list 11](#_Toc467710387)

[Generating number lists 12](#_Toc467710388)

[The map function. 12](#_Toc467710389)

[Recursion 13](#_Toc467710390)

[Writing Programs 13](#_Toc467710391)

[I/O 13](#_Toc467710392)

[Solutions to Tasks (DON’T CHEAT!) 14](#_Toc467710393)

[Bibliography 15](#_Toc467710394)

1. Background Reading

Read the first 2 main topics in the Wikipedia article on “Functional Programming”

Using what you have just read and the “Reaves” textbook write down the definitions for.

|  |  |
| --- | --- |
| **Declarative Programming** |  |
| **Functional Programming** |  |
| **Function** |  |
| **Argument** |  |

# Specification Requirements









If you look at the spec other languages support the functional paradigm… If you need to use it in your NEA: C#, Python and Java would all be a small step from Haskell. F# is a .net purely functional language that could be used alongside VB.net… there is even some clunky functional clauses in VB but I don’t recommend it!





## Specification Summary

I’m sure that as you scrolled through the clips of the specification had the frisson of excitement that comes when approaching something completely new to learn. In case you are feeling slightly overwhelmed… the above actually boils down to:

**Understand functional syntax and terminology***(maybe 20 new terms)*

**Be able to write functions** *(it’s no harder than normal functions.. just sneakier and more ~~concise~~ ‘terser’)*

**Know and understand the use of about 10 built-in functions***(they are similar to functions you know)*

**Be able to work with lists***(same as arrays and lists in other languages, but the methodologies are more like SQL than vb or python)*

We will learn a little more than is absolutely required for the exam, so that we can use Haskell for the Big Data topic and ensure you all can use it for your NEA if it will be useful.

# Learning Resources

If you JUST work through this workbook AND do all the exercises properly, you should have enough to get you through the exam comfortably. For a more refined understanding I suggest you use one or more of the resources listed below:

1. [Learn You a Haskell for Great Good!
A](http://learnyouahaskell.com/) funny (yes funny) online book. Uses the terminal like we do in class... LYaH uses Linux(ish) terminology so you need to do a little thinking about some tasks
2. [FutureLearn MOOC](https://www.futurelearn.com/courses/functional-programming-haskell/) Functional Programming in Haskell
Lots of Videos, examples and tasks  (I have used various parts of this course.)
3. [EdX MooC Introduction to Functional Programming](https://www.edx.org/course/introduction-functional-programming-delftx-fp101x-0)
More general than just Haskell (I have also used tasks and [resources](https://online.godalming.ac.uk/mod/folder/view.php?id=34177) from this course)
4. [Learn Haskell The Hard Way
T](http://yannesposito.com/Scratch/en/blog/Haskell-the-Hard-Way/)here are several Tutorial sites that do more or less the same job.. this one is very well written and covers everything we need and a little more… Don’t expect hand-holding here!
5. <https://www.haskell.org/documentation>)..
The main location for all info Haskell.

Regardless of what resources you use …this isn't going to be easy! It might feel like the first time you started programming... you might find yourself shouting at your monitor... things like "Why would you do it like this???"  Maybe with additional words! Just keep going DON'T SKIP BITS!!!!  Code **every** example!! Code every part of the examples….

**DON'T GIVE UP! (*If you find it easy... I’d keep it to yourself!)***

***Disclaimer! If you do use a variety of resources you may get a ‘de ja vue’ feeling you may even feel some of them bare a resemblance to exercises in this pack. I have listed the sources I have used at the back of the workbook… I have not used inline/Harvard style referencing.***

# Getting started with Haskell (!IMPORTANT!)

All the tutorials and courses have slightly different versions of Haskell different IDEs /editors. This can lead to frustration when files won’t load or commands don’t work. I’m going to suggest you set-up your home installation the same as college and that we start by using the terminal/command line and a separate editor. Then we’ll do a nice trick with notepad++ to make our lives easier as the tasks get trickier. Feel free to use your editor of preference if you have one!

## Installation

### Installing at Home

Go to <https://www.haskell.org/platform/> **READ THE INSTRUCTIONS!!!!**  The minimal version will be fine… there are versions for Windows, MacOS and Linux.

### Using Haskell at College

Figure 1

All College PCs have access to Haskell through “Departmental Applications” in the start menu.( See Figure 1)

### Using Notepad++

At college you MUST use the portable version downloaded from GoL…  **ON NO ACCOUNT MUST YOU DOWNLOAD/ INSTALL OR RUN PORTABLE YOUR OWN VERSION WITHOUT PERMISSION.**  (There may be VIM, Sublime etc. available in the GoL resources folder).. If you want to use something else ask the ILT technicians.

## First Steps REPL

As almost all the tutorials do we will start by trying a few expressions in the REPL (same has python shell).

Loading GHCi should give you:

Prelude> is just the main set of libraries you can type any line of code at the prompt and it executes and prints the output…all other commands start with a colon :

### Hello World

Prelude> putStrLn "Hello World!"

Although we don’t need to do this in interactive mode it will be useful later! There is also print

### Simple Operations

Prelude> 3+3

Should output **6**… Try a few other mathematical operations. + - \* / ^ BODMAS works (ALMOST!!!!) the same. Try some Boolean operations ||, &&, /=, >=, <=, ==

Prelude> 3>3

Should output **False**

### Simple Functions

Let’s create a function that doubles a number f(x) =2x

Prelude>let double x = x\*2

Hitting return should do nothing… now you can call that function

Prelude> double 3

Should output **6 (NB notice there are NO brackets) …now try….**

Prelude> double (double 3)

Should output **12 (Notice I needed to parenthesise the second function)**functions can also return Boolean values

Prelude> let isbigger x y = x>y

Test it with 3 2 then 3 3 then 3 4. It should return True, False, False.

1. Order of operations in Haskell

Look up the order of operation in Haskell.. Make sure you know what the “whitespace” operator does. Why are there brackets around the last example? Try it without… Make sure you have a reasonable grasp of the extended BODMA rules…

Read <http://learnyouahaskell.com/starting-out#babys-first-functions> for more easy examples

1. Simple Functions

Create the following functions at the Prelude> prompt and test them with simple data.. You may need to google if you want to use a built in function or operator!

|  |  |  |
| --- | --- | --- |
| 1. **sqr f(x) = x2**
 | 1. **add3 f(x,y,z) = x+y+z**
 | 1. **pow f(x,y) = xy**
 |
| 1. **hyp** $f(x,y)=√(x^{2}+y^{2})$
 | 1. **vel f(u,a,t) = u+at**
 | 1. **circarea f(r)= πr2**
 |
| 1. **issmaller f(x,y) 🡸 x<y**
 | 1. **iseven f(x) 🡸 x mod 2 =0**
 | 1. **isTriple (a,b,c) 🡸 a2** =b2+c2
 |

1. **Palindrome checker. Write a function that returns true if a string is a palindrome.**
2. **Arithmetic series. Write a function that returns the nth term of a series with common difference c.**

**Hints: π is pi, to reverse a string use reverse “string”, mod is the mod function.**

**To use a function “infix” surround it with single quotes… i.e. mod 3 10 = 3 `mod` 10
the single quote can be used with any 2 attribute function. Try it with your is smaller function**

You used the wrong single quote…. It’s not an apostrophe it’s a left leaning single quote: **`**

## First Steps Program Files

Restart the GCHi program so we are starting with a fresh session

Open notepad (not np++)

Write the double function as above… SaveAs prog1.hs (**all files** NOT txt)
Save it to the root of your H:/ drive

Prelude> :cd H:/
Prelude> :load prog1

[1 of 1] blah blah blah
OK. Modules loaded: Main.

Prelude>

You should now be able to test double as you did earlier…
add the functions you created in Practice Task 1 to Prog1.. after adding each function test it

You will need to use :reload command (:r also works) so the compiler reads the new functions you are adding

Try reordering the functions in the file. Does the order they are written matter?

Change the **circarea** function so it uses the **sqr** function 🡺 circarea r = pi\*sqr r

Change the hyp function to use **add2** and **sqr 🡺 hyp x y =sqrt(add2 (sqr x) (sqr y))**

1. Haskell Syntax

Look a little closer at Haskell Syntax… here is very ~~pithy “~~Terse” summary <https://www.fpcomplete.com/blog/2012/09/ten-things-you-should-know-about-haskell-syntax>

Try to replace the brackets in the last **hyp** example with $ so you get the same output.

### Notepad++ with NPPexec

Notepad++ can have the Haskell GHCi console built into it… This will allow us to use it a little like an IDE… although we are not going to set-up much automation as I want to get you used to Haskell commands.

Here’s how to set-up NPP to work with Haskell.

Download NPP from GoL unzip and run.

Select “Haskell” from languages

Select Plugins:NPPExec:Execute

In the dialogbox enter:
Z:\ICT\Haskell Platform\bin\GHCi.exe

Save the script as Haskell, and then hit OK

Select Plugins:NPPExec: $FollowCurrentDirectory
Your Haskell file will now be saved in the same folder as NPP

NPP is now set-up. Write a couple of functions in NPP. Test them in the embedded console as you have already done.

## Selection

As with all languages there are Boolean operators

|  |  |
| --- | --- |
| == | Is equal to |
| < > <= >=  | Greater/less than Greater/less or equal to |
| && | AND |
| || (pipes) | OR |
| not  | not |

There are also IF statements but these do no branch they sect which operations a function will perform.

Create a new file using notepad++ and create the following function that calculates monet exchange commission.. It charges at least £2 or 2% of the total.

commission p = if p < 100 then 2 else p\*0.02

You can also structure the if as a block (remember indents are important)

commission p = if p == 0 then 0

 else if p < 100 then 2

 else p\*0.02

If you have specific cases you can just state them… e.g.

commission 0 = 0

commission p = if p < 100 then 2 else p\*0.02

Try the if statement first… remember I said order doesn’t matter??? Well it does for assessing the same named function it does…

1. Sheepish Function

I have a friend who owns a sheep farm, but has forgotten to pay his sheep taxes. Can you write a function named sheepTax that will help him work out how much he has to pay?

* 0-25 sheep cost £1.50 per sheep
* 26-50 sheep cost £1.40 per sheep
* 51-75 sheep cost £1.30 per sheep
* 76-100 sheep cost £1.20 per sheep
* 101 + sheep cost £1 per sheep

The function should accept the number of sheep as ana argument and output the total tax to pay

## ~~Data~~ Types

So far we have been very pythiony about data types. Haskell will assign Types BUT it is better if we define them…

You can find out what type something is by entering the command :type (or just :t)
in GHCi try the following. Write down what you get back next to the line.

ghci> :t 'a'

ghci> :t True

ghci> :t "HELLO!"

ghci> :t (True, 'a')

ghci> :t (True, “a”)

ghci> :t 4 == 5

e.g. Float, Double, Int, String (also called [Char]) , Bool…. Types must start with a capital letter.You don’t really handle attribute types separately, but define the as part of a function.
Find out more about types here: <https://www.haskell.org/onlinereport/basic.html>

A function type looks like this:

add ∷ Int -> Int -> Int

Before the :: is the function name After the :: are the types of the parameters of the function

After the final -> sign is the type of the result of the function

minus∶: Int -> Int -> Int

minus a b = a - b

The function is called minus, it takes two integers (Int) as parameters that we then subtract and
return an Integer (Int) as the result.

1. Types

Write the function types for the following functions

1. multiply a b = a \* b
2. square a = a \* a
3. powerOf a b = a ^ b
4. areaRectangle length width = length \* width
5. volumeCuboid length width height = length\*width\*height

the above might seem boring! Here are some that are a little more interesting:
find out the type by entering the function and using :t function name

1. frac x y = x / y
2. rem x y = x Mod y
3. rms x y = sqrt ((x^2 + x^2)/2) I’ll leave you to check the brackets are correct!

## Lists

Similar to other languages **BUT REMEMBER HASKELL HAS STATIC VARIABLES** once you have defined them you can’t change the data held… even in a list… so if you create an empty list that’s how it stays… FOREVER! Instead you create a new variable (or list) each time there is a change…

Create a list using this syntax

animals = [“tiger”,”polar bear”,”lion”,”monkey”,”elephant”]

What gets returned if you use the following functions?

head animals

tail animals

last animals

init animals

reverse animals

length animals

1. lists

Give examples of the following:

1. A list with 3 strings in it, different to the example, with a suitable variable name
2. A suitably named list of numbers
3. A list of numbers as strings
4. A suitably named list of stringed floats attached to the list of stringed numbers

Answer the following questions based on this list.

subjects = [“computing”,”maths”,”art”,”science”,”PE”]

1. What is the head of this list?
2. What is the tail of the list?
3. What does an empty list look like?
4. What would be returned if we called init on this list?

### Adding to a list

Add to the start of a list with this syntax

“geography” : subjects (NOTICE WE ADD AN ITEM)

Add to the end of a list with this syntax

subjects ++ [“history”] (NOTICE WE ADD A LIST)

### Generating number lists

Haskell has some very handy ways to define a list of numbers
Try the following, and note to the right what is returned:

Prelude> let x =[1..15]

Prelude> x

Prelude> let y = [2,4..100]

Prelude> y

Prelude> let z = [1,3..99]

Prelude> z

You can also create lists of infinite size! DON’T TRY TO PRINT THE WHOLE LIST!!

Prelude> let w =[3,6..]

 But you can now access any element of the list using it’s index !!

Prelude> w!!10

33

### The map function.

This is where functional programming starts to get fun!
Create a file and enter the following functions

addone x = x+1 -- simple function to add one

evens=[2,4..12] -- creates a list of even nums to 12

odds = map addone evens -- maps addone to the list

Call the odds function from the terminal.

1. List creation and simple mapping
* In a file: Create a list that is equivalent to the 5 times table up to 12x5
* Write a function that calculates the circumference of a circle of diameter D
* Use map to find the circumferences of circles of 5,10 ,15 etc. diameter
* Try mapping a few of the functions you created earlier to a number list you generate.
* Make sure you don’t try mapping an infinite list!

For more on lists check out: <https://wiki.haskell.org/How_to_work_on_lists>

## Recursion

The main way to loop is to use recursion: Try the Fibonacci sequence:

fib 0 = 1

fib 1 = 1

fib n = fib (n-1) + fib (n-1)

you simply state the base cases and then construct the function with recursive calls.

1. Recursion

You should have seen most of these before.. so just need to write them in Haskell:

Write a function that will:

1. Produce the nth **Triangle Number**
2. Produce the nth **Factorial**
3. Convert an integer into a binary word

## Built-in recursive functions

Recursion is so key to functional programming many built in functions use it without you realising!

### Fold

Try this in GHCi:

Prelude> fred = [1..5]

Prelude> barney = foldl (+) 0 fred

Prelude> barney

15

Look at Bob Reaves page 402 for the explanation..

Now try this function in a file:

nums =[5,6,2,35,1,19,65,40,34]

listmax = foldl (max) 0 nums

Apply the function listmax. What is returned?

Write down the definition for the max function (DON’T GUESS!) :

Write down the definition for the foldl function:

To find out how fold works see <https://wiki.haskell.org/Foldr_Foldl_Foldl>'

1. Fold (reduce)

Write functions that:

|  |  |
| --- | --- |
| 1. Sum a list of numbers between 1 and 15
 | 1. Fids the product of a list of numbers between 1 and 50
 |
| 1. finds the smallest number in a list
 | 1. Finds the average of a list
 |

# Writing Programs

So far we have only written individual functions and tested them. We now need to combine functions.

## I/O

There are some occasions when you **need** to sequence actions… Input and output are considered “impure” in Haskell as they interact with the user… (look it up if you are interested)
Try this:

test = do

 putStrLn ("what is your name?")

 x <- getLine

 putStrLn ("hello " ++ x)

Save as Prog1.hs and Load the module… call the function test… .. NOW comment the code above make sure you are **really** clear about what each operator is doing. ( use the [HOOGLE search engine](https://www.haskell.org/hoogle/))

1. Input/ Output

Make sure that for the following tasks you specify the Type

1. Volume of a cuboid:
Write a function that ASKS THE USER “nicely” to enter the height, width, depth of cuboid and “nicely” return the volume. **APPLY** your function and show the output here.
2. Nth Fibonacci
Write a function that ASKS THE USER “nicely” to enter which term of the Fibonacci sequence they would like and “nicely” return the that term . **APPLY** your function and show the output here.
3. Fibonacci upto N
Extend you solution for Q2. But show all the terms up to and including the Nth term. **APPLY**  your function.

# Solutions to Tasks (DON’T CHEAT!)

1.
2.
3.
4.
5.
6. Recursion

|  |  |
| --- | --- |
| 1 | tri 1 = 1tri n = n + tri (n-1) |
| 2 | fac 1 = 1fac n = n \* fac (n-1) |
| 3 | dec2bin:: Int -> [Int]dec2bin 0 = [0]dec2bin 1 = [1]dec2bin n = if n `mod` 2 == 0 then dec2bin (n `div` 2) ++ [0] else dec2bin (n `div` 2) ++ [1] |

1. Programs

### Bibliography

Introduction to Functional Programming, Katie Maclachlan,
Taken from CAS resources by permission. Exercises and example used/editied as appropriate.