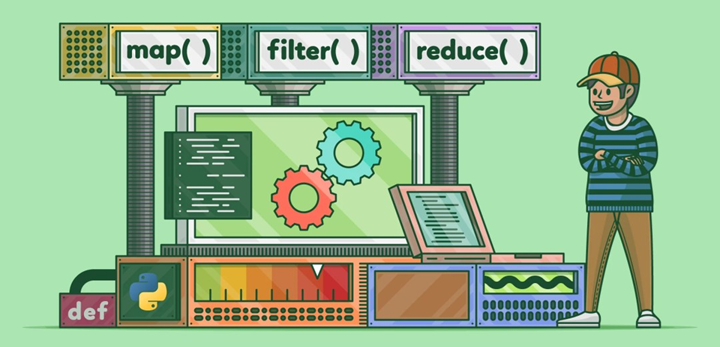
Functional Programming

5CM524

****

Getting Started with Azure lab and GHCi

Lab Instructions for week 1

Dr Youbing Zhao

Prof Stephan Reiff-Marganiec

# Aims

The attached instructions form content for 2 hours of labs. This lab contains no assessed element. Note that there are additional instructions for connecting to the Azure Lab.

1. Connect to the Azure lab and starting GHCi
2. Simple arithmetic and types

# Overview

In this lab you will be learning how to get started in Haskell. We are using an Azure lab with the Haskell compiler preinstalled. The Azure lab has already been setup for you and there are instructions in the lab folder of how to connect. Please make sure that you stop the VM when you are done as otherwise time will keep ticking down!

Note: If you work on your own machine then you can remote connect to the Azure lab or install Haskell from <https://www.haskell.org/ghc/download.html>. Install takes a little while!

# Connecting to Your Virtual Machine on Azure Labs

## Introduction

This document takes you through using the virtual machine that has been allocated to you in the cloud. Once you connected, you will see that it is just like using any other PC that is running Windows.

## Registering for your Virtual Machine

If you were enrolled on this module by the end of induction week, you have already been allocated a virtual machine for this module. All you need to do now is register for it. To do that, go to the following URL:

https://labs.azure.com/register/1hixiwoo3

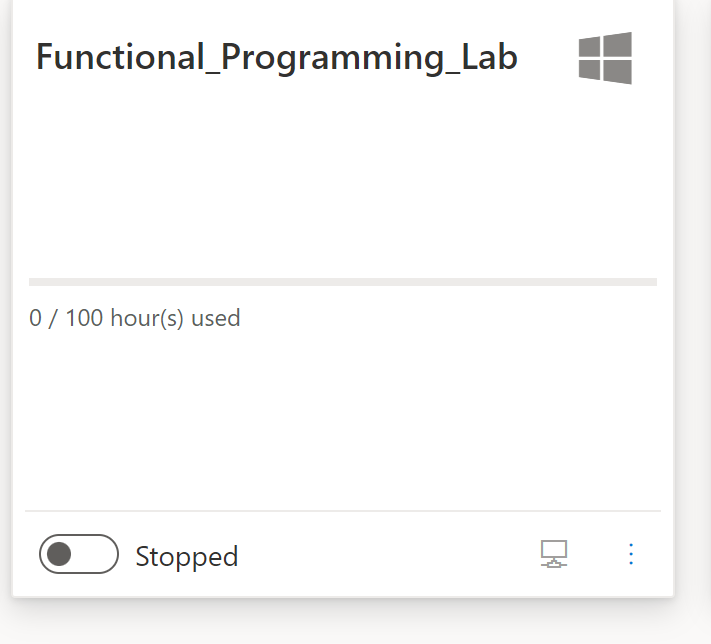
You may be asked to login. If so, use your standard University username and password.

If this fails, the most likely reason is that you enrolled later than the end of induction week and so have not yet been allocated a virtual machine. Ask your module leader to add you to the list of students in the lab.

If successful, a virtual machine called Functional\_Programming\_Lab will have been added to your list of virtual machines. You can see your list of virtual machines at:

<https://labs.azure.com/virtualmachines>

If you have used other virtual machines on Azure previously, you may have more than one virtual machine listed, but the one we are interested can be seen below:



Click on the button in the lower left corner to start your virtual machine. You will now see that the text changes from “Stopped” to “Starting…”. Wait for you virtual machine to start – it might take a while, particularly the first time you use it. Once the virtual machine has started, the button text will change to “Running”.

Now click on the little computer icon in the lower right corner of the panel. This will download a .rdp file containing the connection information for your virtual machine. If you open this file, Windows will open the Remote Desktop Connection application. After a short pause, you will see the following dialog:

A screenshot of a computer security

Description automatically generated

The username “functional” has already been filled in for you. The password is “Programming@UoD”. Click OK and you will be connected to your virtual machine. If you see any dialog boxes along the way that ask you to accept certificates, etc, just accept the defaults.

Eventually, you will see the desktop for your virtual machine. You can now use this just like any Windows PC. You can switch back to your local PC desktop by selecting the Minimise button on the task bar at the top of the screen. Maximising the Remote Desktop Connection will take you back to the virtual machine.

When you have finished with the virtual machine, select “Disconnect” from the Power menu. This will close the Remote Desktop Connection and take you back to your local PC. It is now important that you shutdown the virtual machine. Go back to

<https://labs.azure.com/virtualmachines>

and click the button that says “Running”. You will now see that the text changes to “Stopping…”. You can now close the browser.

Please note that if you do not stop the virtual machine, it may continue to run and use up your allocated quota. Please treat the virtual machine as a valuable resource and always shut it down when it is not being used for a while.

# Getting started with GHCi

Follow the Azure lab instructions to get started and once you are connected to the remote machine open a command prompt (Powershell is recommended )and launch GHCi by typing **ghci** and enter at the prompt. You should see a screen like this (the key bit is the last two lines):

A screenshot of a computer screen

AI-generated content may be incorrect.

Let’s quit GHCi just to learn how to do that: type **:quit** and upon pressing enter you should be back at your command prompt.

Now you know how to get in and out. At the GHCi prompt you can either make function calls or you can use a ‘system’ command. The latter are prefixed by **:** (a colon).

# Simple Arithmetic and Types

Now that we have got the basics out of the way, let’s work with some functions. Remember: play around have fun! The below are suggestions but you can try other things. The worst that can happen is that you get an error message.

Task: Write basic Haskell expressions to evaluate arithmetic expressions.

Goal: Understand basic arithmetic, types (e.g., Int, Double), and type inference in Haskell.

Try things such as mathematical expressions: 5+9, 7-5, 7.5/3 etc and see what happens.

You can try these operators: **+**, **-**, **\***, **/**, **mod**, **^**, **\*\***, **div**. All these are binary (they take two arguments) and are infix (they go between the two arguments).

Note: **mod** and **div** are functions, by default, they should be put in front of operands. If you like to put **mod** and **div** in the middle like in some programming languages, you can use back quotes, such as:

8 `div` 2

8 `mod` 3

A screenshot of a computer screen

AI-generated content may be incorrect.

Also, running through a number of examples check the resulting types: e.g. **:t 5.0-2.5**

# Conditional Expressions (if-then-else)

We can use conditions in functional programming – I am sure you are relieved to hear that some bits you learned last year in programming are reoccurring!

Task: Create some simple functions that use conditional expressions.

Goal: Learn how to use if-then-else in Haskell.

Let’s create a conditional function: **maxOfTwo a b = if a > b then a else b**. Add it to your script, load into GHCi and try **maxOfTwo 7 9** (or such) and see what it does.

Note that you have versions with or without elseif: **if then else** or **if then else if else**.

Note: For multiline programming in GHCi, you need to include your code in :{ and :}

In addition, like Python, Haskell requires indentation.

A black background with white text

AI-generated content may be incorrect.

If you are writing your code in your .hs files, then use :load filename to load the files and :reload without a filename to reload the file

# Playing with list and recursion

**Haskell’s sum** is defined with 2 equations

**sum** :: **Num a => [a] -> a**

**sum [] = 0**

**sum (n:ns) = n + sum ns**

(The list operator : separates a left element from the list)

Can you follow a similar method to **Define a function that produces the product of a list of numbers, such as show that [2, 3, 4] = 24**

**(the function name product has been used by Haskell, so try to use another name)**