COMP105: Programming Paradigms
Week 1 Homework Sheet

This is the homework sheet for **Week 1**. Complete your answers in a file named `week1.hs` and submit them to the "Week 1" assessment in SAM here

`https://sam.csc.liv.ac.uk/COMP/Submissions.pl`

Submission of the weekly homework sheets contributes 10% of the overall module mark, and each homework sheet counts equally towards this. Each homework sheet will be marked on a pass/fail basis. You will receive full marks for submitting a *reasonable attempt* at the homework. If no submission is made, or if a non-reasonable attempt is submitted, then no marks will be awarded.

The deadline for submission is

Friday Week 1 (16/10/2020) at 16:00.

Late submission is **not** possible. Individual feedback will not be given, but full solutions will be posted promptly after the deadline has passed.

If you feel that you are struggling with the homework, or if you have any other questions, then you can contact the lecturer at any point during the week via email, or you can drop in to the weekly Q&A session on MS Teams on Friday between 1PM and 4PM.

Week 1 issues. As this is week 1, and there may be students transferring onto the course during the week, late submissions will be accepted for up to five days **only if you transferred on to the course after Monday 12/10/20.** Please email me (`john.fearnley@liverpool.ac.uk`) if this applies to you and you would like to submit late.

During week 1 we do not do much Haskell, and we will only start programming for the first time in Lecture 3. Due to this, the homework this week is mostly about getting your Haskell environment set up, and ready for the rest of the course. So the file that you submit this week will not contain very much, but this will change as we progress with the course.

Lecture 1 – Install Haskell. You have the option of either installing Haskell on your own machine, or using remote access to use the Haskell installation on the lab machines in Computer Science.

- For **Windows** users, the recommend way to install Haskell is to download Haskell Platform 8.6.5 from `https://www.haskell.org/platform/download/8.6.5/HaskellPlatform-8.6.5-core-x86_64-setup.exe`, which should install all necessary components.

Windows users are **not** recommended to follow the instructions on Haskell.org, which instruct you to install via chocolatey. This will lead to a less functional install, as you will not have a stand-alone ghci application.

- For **Linux** or **OSX** users, the recommended way to install Haskell is to use ghcup, which can be found at `https://www.haskell.org/ghcup/`.

If you wish, you can skip this step and access a Computer Science lab machine remotely, which will already have Haskell installed.

Lecture 1 – Launch ghci. Once you have installed Haskell, the next step is to launch ghci. For Windows users who install the Haskell Platform, look in the start menu (the Windows search bar sometimes can't find Haskell, so look in the programs list) for "Haskell Platform" and run ghci (not WinGhci). If you installed via Chocolatey (not recommended), then you will instead have to open a command prompt and then run `ghci`.

For Linux or OSX users, you will need to open a terminal and type `ghci` to run ghci.

When you run ghci, a `Prelude>` prompt should appear. If so, then you have successfully installed Haskell, and you are set up for the rest of the course.

Lecture 2 – Pure functions. Use a text editor to create a new file (try using `notepad++` if you don't have a favorite text editor), and save it as `week1.hs`. Make sure that the file is actually `week1.hs`, and not `week1.hs.txt`, as Windows will hide file extensions by default. As a quick check, a file ending with `.hs` should automatically launch ghci when opened (if you installed Haskell Platform), whereas a file ending in `.txt` will open a text editor. This is the file that you will submit once you have finished.

To create a comment in Haskell, you just need to prefix by `--`, eg.

```
-- This is a comment

-- This is another one
```

In your file, in a Haskell comment, answer the following questions.

1. A program takes two integers, it adds them together, and then squares the result and returns that number. Can this be implemented as a pure function? Justify your answer.

2. A program takes the URL of a website, and then outputs the HTML source code of the front page of that website. Can this be implemented as a pure function? Justify your answer.

3. A program takes a list of 52 cards. It then shuffles those cards and returns them in a random order. Can this be implemented as a pure function? Justify your answer.

Lecture 3 – Getting started with Haskell.

1. **Evaluating queries using ghci.** Try entering the following queries to ghci

```
1 + 1

7 * 191

True || False

1 > 0
```

Write a *single query* that evaluates whether $7 \times 11 \times 13$ is less than 17×59. You should not multiply any of the arguments together before writing your query.

2. **Evaluating functions.** Recall from Lecture 3 that Haskell has a special syntax for calling functions. Try the following queries:

```
max 10 11

max 10 (1 + 10)

max 10 1 + 10

max 10 1 + max 1 2
```

Is the output what you expect? Write a *single query* that outputs the maximum of 5×199 and 3×331.

3. **Loading functions into ghci.** Open `week1.hs` in your favourite text editor. If you don't have a favorite editor, then try notepad++, as it has Haskell highlighting support. Enter the following code into the file:

```
plus_one x = x + 1
```

Now save the file.

There are now a number of ways of loading this into Haskell. From Windows file explorer, double clicking `week1.hs` should create a new ghci instance with the code loaded. Alternatively, on Windows, from an existing ghci instance you can type:

```
:l C:\Users\John\week1.hs
```

where you should replace the path with the directory where you saved the file. Note that `:l` here is short for `:load`. If one of the folders in the path to your file contains a space you must instead type

```
Prelude> :l "C:\\Users\\John\\COMP105 Homeworks\\week1.hs"
```

where the path has been put in double quotes, and all backslashes are replaced with double backslashes. This can be avoided by removing the spaces from the folder name.

Linux and OSX users should open ghci and then type

```
:l /home/john/week1.hs
```

where the path should be replaced by the path to your file. If the file path contains spaces then instead use

```
:l "/home/john/COMP105 homeworks/week1.hs"
```

where the filepath is now in quotes. There is no need to double the slashes on Linux or OSX.

If you get tired of typing the directory for your code, you can change directory in ghci like so:

```
:cd C:\Users\John
:l week1.hs
```

If you have already loaded some code, Haskell will unload it when you change directory. It will print a warning when it does this.

Once you have loaded the code try out the function `plus_one` in the interpreter:

```
plus_one 500
```

If you get an error like

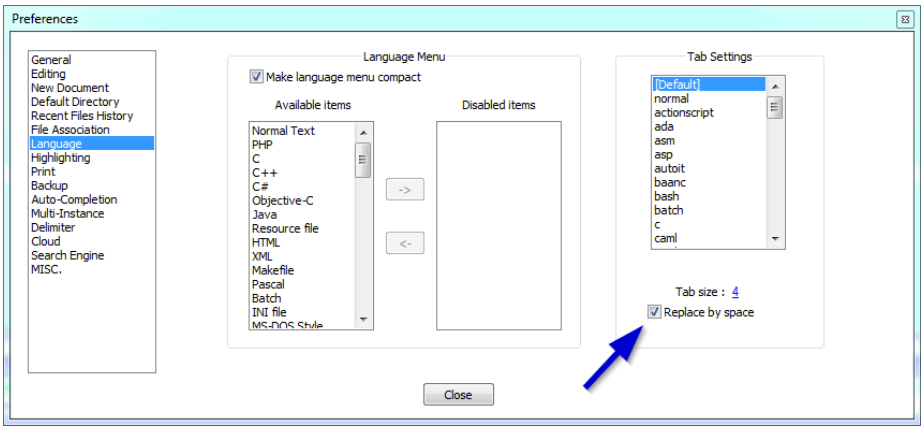
```
<interactive>:8:1: error: Variable not in scope: plus_one
```

then this means that Haskell couldn't find your function. There are two possible causes for this.

(a) You forgot to save your file, so the version that Haskell loaded in didn't have `plus_one` in it.

(b) There was an error when your file was loaded in. In this case ghci should have told you about it, so you can fix that error.

4. **Make notepad++ use spaces instead of tabs.** Or, if you are not using notepad++, then figure out how to make your editor automatically turn tab characters into spaces. The Haskell layout rule gets confused when your code uses tabs, and by default, notepad++ will put tabs at the start of new line. You should configure notepad++ to use spaces instead. You can do this by going to the language settings in the preferences menu, and selecting "Replace by space" under "Tab size".



5. **Editing code.** If you want to change your code, you can edit the source file and then reload the file into the interpreter. Try adding the following code to week1.hs:

   ```
   five_sum x y = (x + y) * 5
   ```

   Re-load week1.hs into the interpreter as before, or use the `:r` command to reload the currently loaded file. This loads `five_sum` into the interpreter. It also re-loads `plus_one`, but since we have not changed that, we won't notice the difference. Try out the functions:

   ```
   five_sum 2 (plus_one 1)
   ```

6. **Errors in Haskell.** If your code is not correct, then Haskell will print out an error message when you try to load it. Input the following broken code into your file

   ```
   broken x = x + 1 + "hi"
   ```

   This code tries to add a number and a string together, which will definitely not work since Haskell is strongly typed. Try loading your file into ghci. It will print out an error message:

   ```
   week1.hs:3:18:
       No instance for (Num [Char]) arising from a use of ‘+’
       In the expression: x + 1 + "hi"
       In an equation for ‘broken’: broken x = x + 1 + "hi"
   ```

   The first line here is important: it tells you the line and character that the error occurs on. In this example, the error is on line 3, and it 18 characters from the start of that line. Your numbers may be different if, for example, you put the code on a different line.

   The second line is Haskell saying that it does not know how to apply + to "Hi" (a bit cryptically – you will understand this better after we discuss types.) The third and fourth lines tell you where the error was.

   Remove `broken` from your file before continuing.

7. **Some ghci shortcuts.** There are a few shortcuts in ghci that can save you some time. If you have loaded a file with the `:l` command and have made some changes, then instead typing out the entire path again, you can just type `:r` to reload the file.

   Tab completion can be used to automatically fill out function names for you. Try typing the following at the ghci prompt

   ```
   ghci> len
   ```

   and then press the tab key. Note how ghci automatically fills out the rest of `length` for you, since that is the only loaded function that starts with `len`. If you instead type

   ```
   ghci> l
   ```

   and press tab, then ghci will give you a list of the functions that start with the letter `l`. Tab completion also works with folder and file names when you are using the `:l` command to load a file.

   Finally, you can use the up arrow key to bring up previous queries that you have entered to ghci. You can then edit these queries, or just re-run them as is. So there is never any reason to type out the same query twice.

8. **Writing your own functions.** Now that we have the basics down, it's time to write our own functions. For each function below, write some code that implements the function into week1.hs, load it into ghci, and test that it works.

   (a) Write a function `minus_one` that takes one argument `x` and returns $x - 1$

   (b) Write a function `quad_power` that takes one argument `x` and returns 4^x. Recall from lecture 3 that `^` is the exponentiation operator.

   (c) The library function `mod x y` returns x modulo y. Use this function to write a function `mod_three` that takes one argument `x` and returns x modulo 3.

   (d) Write a function `add_three` that takes three arguments `x`, `y`, and `z` and returns the sum of all three arguments.

   (e) Recall that `min x y` and `max x y` return the minimum and maximum of their arguments. Write a function `min_max` that takes four arguments `a` `b`, `c`, `d`, and returns $\min(a, b) + \max(c, d)$.

End of homework sheet. Don't forget to submit your `week1.hs` file.