COMP105: Programming Paradigms
Lab Sheet 1

This lab is intended to get you started with using the Haskell interpreter ghci, loading your own code, and writing simple functions using the techniques from Lecture 3.

1. **Launching ghci.** In the start menu, look for "Haskell Platform" and run ghci (not WinGhci). The system may take a minute to configure Haskell, and then a familiar ghci prompt should appear.

2. **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.

3. **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.

4. **Loading functions into ghci.** To write our own functions, we will need to put them into a text file and then load them into ghci. Open a new file in your favorite 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 as "Lab1.hs" in your personal file store (M:\).

There are now a number of ways of loading this into Haskell. From Windows file explorer, double clicking Lab1.hs should create a new ghci instance with the code loaded. Alternatively, from an existing ghci instance you can type:

```
:l M:\Lab1.hs
```

Assuming that you saved your file in "M:\Lab1.hs". Note that `:l` here is short for `:load`.

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

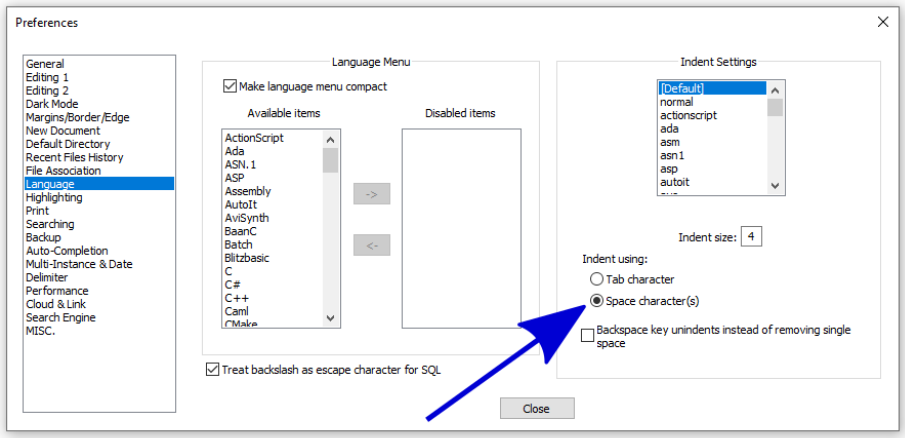
```
:cd M:\
:l Lab1.hs
```

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
```

5. **Make notepad++ use spaces instead of tabs.** Ignore this if you are not using notepad++. 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 "Space character(s)" under the "Indent using:" setting.



6. **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 Lab1.hs:

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

   Re-load Lab1.hs into the interpreter as before:

   ```
   :l M:\Lab1.hs
   ```

   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)
   ```

7. **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 Lab1.hs

   ```
   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 Lab1.hs into ghci. It will print out an error message:

   ```
   Lab1.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 Lab1.hs before continuing.

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 Lab1.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.

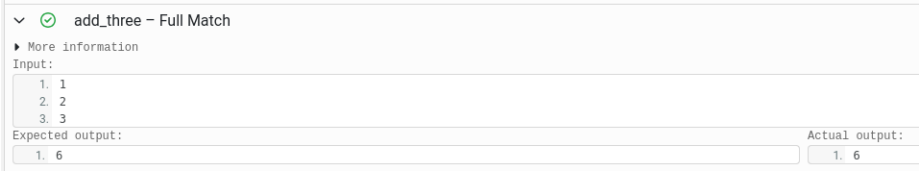
(c) Write a function `add_three` that takes three arguments `x`, `y`, and `z` and returns the sum of all three arguments.

(d) The area of a circle is, of course, πr^2. The constant π is called `pi` in Prelude. Write a function `area` that takes one argument `r` and returns πr^2.

9. **Make an initial Codegrade submission.** You must submit your solutions to each week's lab sheet through Codegrade. Look for "Lab Sheet 1" under the "Assignments" tab on Canvas. This will then take you to the Codegrade upload page. Make sure that your file loads correctly into ghci, and that it is named *exactly* Lab1.hs including capitalization, then upload this file to Codegrade. You can resubmit as many times as you want before the deadline, so this upload won't be your final submission.

   Once you've submitted, Codegrade will show you the code that you uploaded. Navigate to the "AutoTest" tab, which will show you the results of the automated tests that ran against your code. It often takes a few seconds for Codegrade's servers to run the tests, so if the results are still pending, give it some time.

   Each test calls one of your functions with a particular set of inputs. For example, the following test calls `add_three` with arguments `1`, `2`, and `3`, and it expects your code to output `6`.



   More generally, the inputs to your functions will each be shown on a separate line in the "Input" section, and the expected output will be shown below that. Your code will pass the test if it gives the expected output, and it will fail otherwise.

   Hopefully all of the tests for Question 8 passed. If not, see if you can fix the code so that the tests do pass. If any of your tests failed with a compiler error, then have a look at the "Codegrade FAQ" page on Canvas: it gives some hints on how to debug those issues.

10. **Using Prelude functions.**

    (a) 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.

    (b) Write a function `mod3or5` that takes one argument `x` and returns `True` if `x` is divisible by 3 or 5. Remember that `==` can be used to test equality, and that `||` gives the or of two Booleans.

    (c) 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 $\min(a, b) + \max(c, d)$.

(d) The `sqrt` function gives the square root of a floating point number. Use this function to implement `quadratic` that takes three arguments `a`, `b`, and `c` and returns

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

You can test your function with inputs $a = 1/2$, $b = -5/2$, and $c = 2$, which should give the answer 4. Remember that negative arguments need to be put in brackets.

11. **Pure functions.** Each of the questions below ask you to determine whether a given operation can be implemented as a pure function. There is no need to write anything down for these questions. You can check your answers against those given in the solution file, which will be released after the deadline for Lab 1.

    (a) 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?

    (b) 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?

    (c) 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?

12. **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.

13. **Ifs.** Try evaluating the following if queries in ghci:

```
if 1 > 2 then "Bigger" else "Smaller"

if (1 > 2) || (2 > 1) then 1 else 0

if (succ 1 == 2) then (max 1 0) else (min 1 0)

(if True then 1 else 0) + (if False then 0 else 1)
```

Are the answers what you expect them to be? Make sure that you understand why each of these queries give the answers that it does.

Implement the following functions:

(a) Implement a function `gt_100` with one argument `x`, which returns 1 if $x > 100$, and 0 otherwise.

(b) Implement a function `switch` with three arguments `x`, `y`, and `c`. If c is equal to 1 then the function should return x, otherwise it should return y.

(c) Implement `my_max` which takes two arguments `x` and `y` and returns $\max(x, y)$. Your solution should not use the `max` or `min` library functions.

(d) Implement a function `fizzbuzz` that takes one argument `x` and returns "Fizzbuzz!" if x modulo 3 is 0 and x modulo 5 is 0. Otherwise it should return "Nope".

14. **Let expressions.** Use the `let` syntax from Lecture 4 to implement the following functions. Try out both the single-line and multi-line versions of `let`, and make sure to remember Haskell's layout rule.

(a) Write a function `question1 x` that sets $a = x * x$ and returns $2 * a$.

(b) Write a function `question2 x` that sets $a = x + 1$, $b = a * a$, and $c = 2^b$, and then returns $a + b - c$.

(c) Write a function `bounded_square x` that returns x*x if x*x is less than 100, and 100 otherwise. You should use a `let` and an `if`.

Lab complete. Remember to upload your completed Lab1.hs to Codegrade.