# Functional Programming Exercises

## 0 Getting started

We will be using GHCi, the interactive version of the Glasgow Haskell Compiler, for the exercises. To install it, I recommend GHCup:

`https://www.haskell.org/ghcup/`

Having done that, there are then clever ways of setting up an editor such as Emacs or VSCode to get syntax highlighting, autocomplete, etc; but in the interests of simplicity, I'm going to ignore all that and stick to first principles.

To run GHCi, simply open a terminal window and type '`ghci`'. One typically uses a text editor to write or edit a Haskell script, saves that to disk, and loads it into GHCi. To load a script, it is helpful if you run GHCi from the directory containing the script. You can simply give the name of the script file as a parameter to the command `ghci`. Or, within GHCi, you can type '`:l`' followed by the name of the script to load, and '`:r`' with no parameter to reload the file previously loaded.

For example, you should be able to type the following definitions into a file, called say `sample.hs`:

```
-- a sample Haskell script

square :: Integer -> Integer
square x = x * x

smaller :: (Integer, Integer) -> Integer
smaller (x, y) = if x <= y then x else y
```

Then save the file; navigate in your terminal to the directory containing that file; start GHCi and load in that file:

```
ghci sample.hs
```

then evaluate some expressions using the new definitions:

```
*Main> square (3+4)
49
*Main> smaller (3,4)
3
```

*Jeremy Gibbons*
*October 2023*

# 1 Basic definitions

1. Define the following numeric functions:

   - a function *square* that squares its argument, then a function *quad* that raises its argument to the fourth power using *square*;
   - a function *larger* that returns the larger of its two arguments;
   - a function for computing the area of a circle with a given radius (use the type *Double*). (Hint: the formula for calculating the area $A$ of a circle with a radius $r$ is $A = \pi r^2$, where $\pi$ is called *pi* in Haskell.)

2. Here is a script of function definitions:

$$
\begin{array}{l}
\mathit{add} :: \mathit{Integer} \to \mathit{Integer} \to \mathit{Integer} \\
\mathit{add}\ x\ y \quad = x + y \\
\mathit{double} :: \mathit{Integer} \to \mathit{Integer} \\
\mathit{double}\ x \quad = x + x \\
\mathit{first} :: \mathit{Integer} \to \mathit{Integer} \to \mathit{Integer} \\
\mathit{first}\ x\ y \quad = x \\
\mathit{cond} :: \mathit{Bool} \to \mathit{Integer} \to \mathit{Integer} \to \mathit{Integer} \\
\mathit{cond}\ x\ y\ z = \mathbf{if}\ x\ \mathbf{then}\ y\ \mathbf{else}\ z \\
\mathit{twice} :: (\mathit{Integer} \to \mathit{Integer}) \to \mathit{Integer} \to \mathit{Integer} \\
\mathit{twice}\ f\ x \quad = f\ (f\ x) \\
\mathit{infinity} :: \mathit{Integer} \\
\mathit{infinity} \quad = \mathit{infinity} + 1
\end{array}
$$

   (The function *twice* is an example of a higher-order function, which takes another function as one of its arguments. Although we haven't studied higher-order functions yet, just follow the reduction rules.) Give the applicative- and normal-order reduction sequences for the following expressions:

   - $\mathit{first}\ 42\ (\mathit{double}\ (\mathit{add}\ 1\ 2))$
   - $\mathit{first}\ 42\ (\mathit{double}\ (\mathit{add}\ 1\ \mathit{infinity}))$
   - $\mathit{first}\ \mathit{infinity}\ (\mathit{double}\ (\mathit{add}\ 1\ 2))$
   - $\mathit{add}\ (\mathit{cond}\ \mathit{True}\ 42\ (1 + \mathit{infinity}))\ 4$
   - $\mathit{twice}\ \mathit{double}\ (\mathit{add}\ 1\ 2)$
   - $\mathit{twice}\ (\mathit{add}\ 1)\ 0$

**Note:** There is not a mistake in the last expression; all you need to know for the time being is that a function application that doesn't have enough arguments is already in normal form. Just follow the rules when reducing the expression.

3. Give a reduction sequence for *fact* 3, where the factorial function *fact* is as defined as follows:

$$
\begin{array}{l}
\mathit{fact} :: \mathit{Integer} \rightarrow \mathit{Integer} \\
\mathit{fact}\ 0 = 1 \\
\mathit{fact}\ n = n * \mathit{fact}\ (n - 1)
\end{array}
$$