Programming with Patterns (31050 and 32050)
Laboratory exercises: Week 2

Question 1 Reduce the following λ-terms by hand.

$$
\begin{array}{l}
(\lambda x.x)\ 3 \\
(\lambda x.x)\ (\lambda x.x) \\
(\lambda x.x\ x)\ (\lambda x.x\ x) \\
(\lambda x.(\lambda y.y))\ 3 \\
(\lambda x.(\lambda x.x))\ 3 \\
(\lambda x.(\lambda y.x))\ (z+1) \\
(\lambda x.(\lambda y.x))\ (y+1)
\end{array}
$$

taking care to manage α-conversion explicitly.

Question 2 Natural numbers can be defined using `zero` and `successor` so that

$$\overline{3} = \texttt{successor}\ (\texttt{successor}\ (\texttt{successor}\ \texttt{zero}))$$

These can be encoded as λ-terms as follows:

$$
\begin{array}{rcl}
\texttt{zero} & = & \lambda f.\lambda x.x \\
\texttt{successor} & = & \lambda n.\lambda f.\lambda x.n\ (f\ x).
\end{array}
$$

The idea is that numbers are encoded as iterators. `zero` applies f zero times, while `successor` n applies f once, and then n times more. Code these up in (untyped) bondi. Use this to evaluate $\overline{3}\ g\ 1.0$ where g is $\lambda x.x * 2.0$.

Question 3 Following on from Question 2, define addition of natural numbers. Hint: to iterate f all of $m+n$ times, first iterate it m times and then n times more. Check this by iterating g on some addition.

Question 4 Following on from Question 3, define multiplication of natural numbers. Hint: modify the hint from Question 3. Check this by iterating g.

Question 5 It is useful to be able to define alternatives. Define

$$
\begin{array}{rcl}
\texttt{inleft} & = & \lambda x.\lambda f.\lambda g.f\ x \\
\texttt{inright} & = & \lambda y.\lambda f.\lambda g.g\ y \\
\texttt{case} & = & \lambda f.\lambda g.\lambda z.z\ f\ g.
\end{array}
$$

Encode this in **bondi** and check out some examples. For example, let f be some integer function and g be as above, and compute

$$\texttt{case}\ f\ g\ (\texttt{inleft}\ 3)$$
$$\texttt{case}\ f\ g\ (\texttt{inright}\ 3.3)$$

Question 6 **bondi** supports sequences of terms (commands). For example,

$$\texttt{print}\ \text{"}\texttt{abc}\text{"}; 4 + 1$$

prints the string "abc" and then evaluates 4+1. Also () is the command that does nothing. Use these to define while-loops by recursion.

$$\texttt{let rec while}\ b\ c = \texttt{if}\ b \ldots$$

Question 7 find out how **bondi** handles assignment to references, and use this to define the factorial function by a while-loop.