# Assignment One: Haskell

Due: 11:59pm, Tuesday 16 January 2024. Submit solution using Canvas

You may do this question either alone or with a partner (i.e., in a group of 1 or 2); you both need to understand and explain your solution. This is only for participation marks, but you won't pass the midterm exams unless you have done the assignments and understand the solutions.

Make sure you name(s) and student number(s) are at the top of your file. **All students in a group should submit it to Canvas.**

## Question One

(a) The sum of the first n terms of the harmonic sequence is defined as

$$1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n$$

Using Haskell, define the function `harmonic` which takes one argument, a natural number n, and returns the sum of the first n terms of the harmonic sequence. You can use Haskell's built-in arithmetic operations.

(b) Does your program work with the query:

```
harmonic (length [1,2,3,4])
```

Explain why. (Hint: use `:type`). Give a reason that Haskell works like this. If it does not work, fix it so that it does work.

## Question Two

Hockey, baseball and other sports have playoff series, where the first team to win 3 or 4 games wins the series. Apparently, in the world series of baseball, the team that is up 2-1 in the best-of-seven series has won the series 70% of the time. A simple model is that each team has a probability of winning each game that is independent of the other games. Suppose that the team of interest has a chance of $p$ (where $0 \leq p \leq 1$) of winning each game. We want to compute the probability of it winning the series.

You can choose how to represent the problem, and what function(s) to use, but you need to be able to answer the following queries (suitably posed in your representation):

- If a team has a 0.6 chance of winning each game, what is the probability of it winning a best-of-5 series (where the first team to win 3 games wins the series)?
- If a team has a 0.6 chance of winning each game, what is the probability of it winning a best-of-7 series (where the first team to win 4 games wins the series)?
- If each team has a 0.5 chance of winning each game, what is the probability of a team that is up 2 games to 1 in best-of-7 series will win the series?

For each function you use, you must give its type, a comment that specifies (unambiguously) what the function computes, and the definition of the function.

Hint: the state of a series is the numbers of games the team has to win to win the series, and the number of games the team has to lose before losing the series. A win or a loss changes the state of the series. Given the probability of winning a game, you can determine the probability of winning a series recursively. The only thing you need to know about probability is reasoning by cases:

$$P(s) = P(e) * P(s \mid e) + (1 - P(e)) * P(s \mid \neg e)$$

where $P(s \mid e)$ is the probability of the state that results from event $e$ occurring in state $s$, and $\neg e$ is the negation of $e$ (the event not happening).

## Question Three

Given the following Haskell Program:

```
foo :: Int -> Int
foo x = x+3
dfoo x = foo (foo x)
nfoo x = x+3+3
fooeach [] = []
fooeach (h:t) = foo h : t
nfooeach [] = []
nfooeach (h:t) = nfoo h : t
iffoo [] = []
iffoo (h:t)
    | h < 4.3 = h:t
    | otherwise = t
dd x y = 2*x + 3*y + 7
```

What is the inferred type of the following? (The inferred type is the most general type that is consistent with the definitions.)

(a) `dfoo`

(b) `nfoo`

(c) `fooeach`

(d) `nfooeach`

(e) `iffoo`

(f) `dd`

(g) `dd 4.3`

Try to work it out yourself, and then check it using the `:type` command.

## Question Four

Give the type and definition each of the following functions. You may use Haskell's built-in functions ==, :, elem, length, Boolean functions and arithmetic functions.

A team of 2 should do all parts, but a team of 1 only needs to do 3 of the 6 parts (but is encouraged to try all). Everyone is encouraged to think about the challenge part.

(a) *myreplace x y lst* replaces all occurrences of $x$ in list *lst* with $y$. For example (where `=>` means "evaluates to"):

```
myreplace 7 3 [7,0,7,1,7,2,7,3] => [3,0,3,1,3,2,3,3]
myreplace 'a' 'x' "" => ""
myreplace 'a' 'x' "xabacadx" => "xxbxcxdx"
```

(b) *myapply lst sub* where sub is a list of $(x, y)$ pairs, replaces each occurrence of $x$ by $y$ in *lst*.

```
myapply "abcdec" [('a','f'), ('c','3'), ('g','7')] =>  "fb3de3"
myapply "baab" [('a','b'), ('b','a')] => "abba"
```

(c) *myordered lst* is True if list *lst* is ordered (by $\leq$, which in Haskell is the function `<=`).

```
myordered [] => True
myordered [2] => True
myordered [1,2] => True
myordered [1,1] => True
myordered [2,1] => False
myordered "abcdefg" => True
myordered "abba" => False
```

(d) *myremoveduplicates* which removes duplicates from a list. For example:

```
myremoveduplicates "abacad" => "bcad"
myremoveduplicates [7,3,2,1,3,2,2,1,1,3,7,8] => [2,1,3,7,8]
```

Challenge part: implement two versions, one of which keeps the last occurrence and one of which keeps the first occurrence. (The above examples keep the last occurrence; to keep first occurrences, the answers are "abcd", and [7,3,2,1,8]). [Hint: keep track the set of elements found.]

(e) *deln n e lst* returns the list that results from deleting the first $n$ occurrences of $e$ from *lst* (or all of them if there are less than $n$).

```
deln 2 'a' "avatar" => "vtar"
deln 4 'a' "avatar" => "vtr"
```

(f) *delna n e lst* returns a list of all of the lists that result from deleting exactly $n$ occurrences of $e$ from *lst*

```
delna 2 'a' "avatar" => ["vtar","vatr","avtr"]
delna 4 'a' "avatar" => []
```

## Question Five

For each question, specify how long you spend on it, and what you learned. Was the question reasonable? (This question is part of the assignment, so please do it!)