Assignment 1

Solve the following exercises. Any non-code part of the assignment must be handed in as comments in the source code.

You write multiline comments between `(*` and `*)` and single-line comments using `//`.

Some of the exercises (which are marked) are copied more or less verbatim from the course book, for your convenience, possibly with minor reformulations.

Miscelaneous exercises

Exercise 1.1

Write a function `sqr : int -> int` that given an integer `x` returns `x` squared

Exercise 1.2

Write a function `pow : float -> float -> float` that given two floating point numbers `x` and `n` returns `x` to the power of `n`

Hint: Use the library function: `System.Math.Pow`

Exercise 1.3 (based on 1.4 from HR)

Write a recursive function `sum : int -> int` such that given an integer n such that $n \geq 0$ returns the sum of all integers from 0 to n inclusive.

$$\texttt{sum}\ n = 0 + 1 + \ldots + (n - 1) + n$$

Hint: Use two clauses with 0 and n as patterns.

Exercise 1.4 (based on 1.5 from HR)

The sequence $F_0, F_1, F_2, \ldots$ of Fibonacci numbers is defined by:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

Thus, the first members of the sequence are $0, 1, 1, 2, 3, 5, 8, 13, \ldots$.

Write a function `fib : int -> int` that given an integer n such that $n \geq 0$ computes F_n. Use a declaration with three clauses, where the patterns correspond to the three cases of the above definition.

Give an evaluations for F_4.

Exercise 1.5 (based on 1.6 from HR)

Declare a recursive function `sum2 : int * int -> int`, that given a pair (m, n) where for $m \geq 0$ and $n \geq 0$ produces the following result.

$$\texttt{sum}\ (m, n) = m + (m + 1) + (m + 2) + \cdots + (m + (n - 1)) + (m + n)$$

Hint: Use two clauses with $(m, 0)$ and (m, n) as patterns.

Give the recursion formula corresponding to the declaration, similar to how we did the Fibonacci numbers in Exercise 1.4

Exercise 1.6 (based on 1.7 from HR)

Given the functions `fact` and `power` from HR

```
let rec fact =
    function
    | 0 -> 1
    | n -> n * fact(n-1)
```

```
let rec power =
    function
    | (x,0) -> 1.0
    | (x,n) -> x * power(x,n-1)
```

Determine a type for each of the expressions:

- `(System.Math.PI, fact -1)`
- `fact(fact 4)`
- `power(System.Math.PI, fact 2)`
- `(power, fact)`

Exercise 1.7 (based on 1.8 from HR)

Consider the declarations:

```
let a = 5
let f a = a + 1
let g b = (f b) + a
```

Find the environment obtained from these declarations and write the evaluations of the expressions `f 3` and `g 3`.

Exercise 1.8

Write a function `dup : string -> string` that given a string `s` concatenates `s` with itself. You can either use + or ˆ to concatenate strings.

Example: `dup "Hi " = "Hi Hi "`

Exercise 1.9

Write a function `dupn : string -> int -> string` that given a string `s` and an integer `n` concatenates `s` with itself `n` times.

Example: `dupn "Hi " 3 = "Hi Hi Hi "`

Exercise 1.10

Assume the time of day is represented as a pair `(hh, mm) : int * int`.
Write a function `timediff : int * int -> int * int -> int` so that `timediff t1 t2` computes the difference in minutes between t1 and t2, i.e., t2-t1.

Examples:

- `timediff (12,34) (11, 35) = -59`
- `timediff (12,34) (13, 35) = 61`

Exercise 1.11

Write a function `minutes : int * int -> int` that given a time `t`, represented in the same way as in Exercise 1.10, computes the number of minutes to `t` since midnight.

Hint: Use the function `timediff` from Exercise 1.10.

Examples:

```
minutes (14,24) = 864
minutes (23,1) = 1381
```

Exercise 1.12 (based on 2.8 from HR)

The following figure gives the first part of Pascal's triangle

$$
\begin{array}{c}
1 \\
1\ 1 \\
1\ 2\ 1 \\
1\ 3\ 3\ 1 \\
1\ 4\ 6\ 4\ 1
\end{array}
$$

The entries of the triangle are called binomial coefficients. The k'th binomial coefficient of the n'th row is denoted $\binom{n}{k}$, for $n \geq 0$ and $0 \leq n$. For example, $\binom{2}{1} = 2$ and $\binom{4}{2} = 6$. The first and last binomial coefficients, that is, $\binom{n}{0}$ and $\binom{n}{n}$, of row n are both 1. Binomial coefficient inside a row is the sum of the two binomial coefficients immediately above it. These properties can be expressed as follows:

$$\binom{n}{0} = \binom{n}{n} = 1$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \text{ if } n \neq 0, k \neq 0, \text{and } n > k$$

Declare a function `bin : int * int -> int` that given a pair (n, k) computes $\binom{n}{k}$.

Extra-curicular activities

There is an interesting video on fractal-like properties that are found when looking at even and odd numbers of Pascal's triangle.

The really bitten can have a crack at this Kattis exercise in your programming language of choice (non-trivial and ***absolutely not required***).

Exercise 1.13 (Based on 2.9 from HR)

Consider the declaration:

```
let rec f =
  function
  | (0,y) -> y
  | (x,y) -> f(x-1, x*y)
```

1. Determine the type of f.
2. For which arguments does the evaluation of f terminate?
3. Write the evaluation steps for `f(2, 3)`.
4. What is the mathematical meaning of `f(x, y)`?

Exercise 1.15 (Based on 2.10 from HR)

Consider the following declaration:

`let test(c,e) = if c then e else 0`

1. What is the type of test?
2. What is the result of evaluating `test(false, fact(-1))` ?
3. Compare this with the result of evaluating `if false then fact -1 else 0`

Exercise 1.16 (based on 2.13 from HR)

The functions curry and uncurry of types

`curry : ('a * 'b -> 'c) -> 'a -> 'b -> 'c`
`uncurry : ('a -> 'b -> 'c) -> 'a * 'b -> 'c`

are defined in the following way:

`curry f` is the function `g` where `g x` is the function `h` where `h y = f(x, y)` .

`uncurry g` is the function `f` where `f(x, y)` is the value `h y` for the function `h = g x` .

Write declarations of `curry` and `uncurry` .

Scrabble assignments

Before we start, a slight warmup.

Assignment 1.17

Create a function `isVowel : char -> bool` that given any of the characters `A` , `E` , `I` , `O` , `U` , returns true, and false otherwise. The function must work for both uppper and lower case characters

Hint: Use `System.Char.ToLower` before you pattern match

**Example:*

```
> isVowel `I`
- val it : bool = true

> isVowel `i`
- val it : bool = true

> isVowel `Q`
- val it : bool = false
```

Assignment 1.18

Create a function `isConsonant : char -> bool` that returns true if a character is a letter, but not a vowel, and false otherwise.

Hint: Use `System.Char.IsLetter` to determine if a character is a letter and `isVowel` to figure out if it is a consonant. Matching on all the consonants works for our purposes, but is not pretty.

```
> isConsonant `I`
- val it : bool = false

> isConsonant `i`
- val it : bool = false

> isConsonant `Q`
- val it : bool = true
```

Scrabble fundamentals

If you have not done so already, make sure that you have read and understood the [rules of Scrabble](). It is not strictly required in order to solve this assignment, but it helps to have a mental image of what we are implementing.

We are now going to get started with the Scrabble project as well as getting familiar with the *first-class* nature of functions in functional programming languages. Before we start, we will use the following terminology for the duration of the project:

- A *tile* is a piece that you place on the board. It contains a character and a point value. The pair `('H', 4)` or `('A', 1)` are, for instance, two examples of tiles where the first element denotes a character and the second value its point value. We currently do not consider blank tiles (where you may place any letter you want) but they will be covered later.
- A *square* is one element of the grid of the board that helps calculate the points of words that are placed over it. Two examples of squares on a standard Scrabble board are *double letter score* or *tripple word score* but we will have the oportunity to make our own. All empty squares on the board can be thought of as *single letter score*.

This week we focus on two things:

1. representing words as a sequence of tiles
2. representing squares as functions that compute point values

As we progress through the course these design will change slightly, but the principles that we cover here will remain the same.

Sequences of tiles

From your previous courses you will be familiar with several ways to handle collections of data (Linked lists, arrays, maps, etc). F# has all of these. Today we will focus on representing words as functions of type `int -> char * int` that given an integer position `p` returns a character and its point value at that position. This solution is not entirely optimal.

- All words are infinite (up to 32-bit integers anyway) and a default value needs to be used outside the intended length of the word.
- The length of the word is not made clear from the representation while structures like lists and arrays have their lengths readily available.
- In our setting, lookup times are linear

Nevertheless, getting to grips with *functions as values* is fundamental to learning funcitonal programming and we will optimise in future exercises.

As an example, The letter H is worth 4 points in scrabble, and the letter `O` is worth two points. If I have a function `f` that spells the word HELLO then `f 0 = ('H', 4)` and `f 4 = ('O', 2)`. We will come back to this example later.

Assignment 1.19

Create a function `empty : char * int -> (int -> char * int)` (note that the parentheses are strictly not necessary for the type but help with readibilty in this case) that given a default value `def` returns a function that given any integer returns `def`.

Examples:

```
// Explicit typing information will, depending on your definition of empty
// most likely be necessary here.
> let theLetterA : int -> char * int = empty ('A', 1);;
- val theLetterA : (int -> char * int)

> theLetterA 0;;
- val it : char * int = ('A', 1)

> theLetterA 42;;
- val it : char * int = ('A', 1)

> theLetterA -762;;
- val it : char * int = ('A', 1)
```

Assignment 1.20

Create a function `add : int -> (char * int) -> (int -> char * int) -> (int -> char * int)` (note that the middle set of parentheses *are* needed in this case and are not just there for readability) that given a position `pos`, a pair containing a character and a point value `cv`, and a word represented by the function `word` returns another function that behaves exactly like `word` except that it returns `cv` when its position argument is equal to `pos`.

Examples:

```
> let theLettersAB = add 1 ('B', 3) theLetterA;;
- val theLettersAB : (int -> char * int)

> theLettersAB 0;;
- val it : char * int = ('A', 1)

> theLettersAB 1;;
- val it : char * int = ('B', 3)

> theLettersAB 42;;
- val it : char * int = ('A', 1)
```

Assignment 1.21

Use the functions `empty` and `add` to create the function `hello : int -> char * int` that spells HELLO starting at index 0 and ending at index 4. Use `(char 0, 0)` as the default value. The letter H is worth four points, O is worth two points, and all other letters are worth one point.

Hint: By far the easiest way to do this is by using the piping operator `|>` as described in HR 2.11.

Creating squares

The squares on the board help calculate the number of points players get when placing words over them. Modelling them in a good way is non-trivial and we start here with a few simple cases. For this exercise we focus on squares that give points for single letters of words only (double letter score for instance) and next week we move on to squares that give points for the entire word (like double word score). We also keep our representation reasonably general in order to be able to expand to more esoteric types of squares down the line (squares that give negative points for vowels for instance).

Assignment 1.22

For this exercise, squares are functions and have the type `(int -> char * int) -> int -> int`. They take a word function `word`, the position in the word of the tile that is placed on the square `pos`, and returns the number of points you get for that square.

Create the square functions `singleLetterScore`, `doubleLetterScore` and `trippleLetterScore` that all have the type `(int -> char * int) -> int -> int` such that:

- `singleLetterScore` returns the point value of the tile placed on the square
- `doubleLetterScore` returns twice the point value of the tile placed on the square
- `trippleLetterScore` returns thrice the point value of the tile placed on the square

Examples:

```
> singleLetterScore hello 4;;
- val it : int = 2

> doubleLetterScore hello 4;;
- val it : int = 4

> trippleLetterScore hello 4;;
- val it : int = 6
```

Next week we will cover more types of squares and alternative ways to represent our words. The overall structure and the general intuition will, however, remain the same.