Haskell Tutorial.

(Always include the type signature of your functions!)

---

1. Write an Haskell function **count_one** to count the number of "1" in a list of numbers.
   count_one [1,2,4,5,6,6,7,1,3,1] = 3

2. Write an Haskell function **sum10** to get all the tuple of four numbers (x,y,z,w) that sum up to 10. The values of x,y,z,w goes from 0 to n , where n is an input.
   sum10 3 =
   `[(1,3,3,3),(2,2,3,3),(2,3,2,3),(2,3,3,2),(3,1,3,3),(3,2,2,3),(3,2,3,2),(3,3,1,3),(3,3,2,2),(3,3,3,1)]`

3. Write a Haskell function **mcd** to get the highest number that can divide two given numbers (=MCD, maximum common dividend of two numbers).
   (you might need to define a function max_list that gets the maximum of a list of numbers, even if it is not strictly needed)
   mcd 45 18 = 9

4. Write an Haskell function **pos_one** to get the position of the "1" of a list containing all "0" and just one "1".
   pos_one [0,0,0,0,1,0,0,0] = 5

5. Define a Haskell a function **league** that receives as input a list of football teams and compute a list of all the possible league matches (home and away). For example:

league ["chelsea", "arsenal", "man utd"] = [("chelsea","arsenal"),("chelsea","man utd"),

("arsenal","chelsea"),("arsenal","man utd"),("man utd","chelsea"),("man utd","arsenal")]

6. Define a Haskell function **power** that computes the following

   power x,y = x^y

   (y must be a positive integer, do not use the built-in operator ^ !!!). For instance:

   power 2 4 = 16 or power 3 3 = 27

7. Define a Haskell function **first_n_odds** that computes a list of the first n odd numbers. For instance:
   ```
   first_n_odds 0 = []
   first_n_odds 1 = [1]
   first_n_odds 5 = [1,3,5,7,9] (first 5 odds numbers)
   ```

8. Define an Haskell function **c_vowel** that counts the number of vowels in a string of small letters only. Example:
   c_vowel "this is a text" = [('a',1),('e',1),('i',2),('o',0),('u',0)]

9. A small database contains two tables, one is a list of employees , identified by a unique emp_id, and an emp_name, salary and job_id. The other table is a list of jobs (job_id, job_name). Table jobs is related by a 1 to many relation with table employees, job_id is the foreign key.

The following is the content of the tables:

```
employees = [(1,"Mark",70000,1), (2,"Adam",40000,2), (1,"Karl",30000,3),
(1,"Mary",50000,2), (1,"Florence",50000,3), (1,"Jim",30000,3),
(1,"Tom",30000,1), (1,"Siobhan",45000,3), (1,"Ann",60000,4),
(1,"Kevin",90000,4),]

countries = [(1,"DBA"), (2,"Developer"), (3,"Analyst"), (4,"Specialist")]
```

Write the following Haskell functions:

a. get_emp_below n – to get all the names of the employees with a salary below below n (=the input)

```
get_emp_below 45000 = ["Adam",Karl","Jim","Tom"]
```

b. get_emp job_name – to get all the employee names given a job_name (not job_ id! You need to have a function that, given the job_name gets the job_id)

```
get_emp "Analyst" = ["Florence","Karl","Jim","Siobhan"]
```

c. num_emp – to list all the country and for each country the number of cities in each country.

```
num_emp = [("DBA",2),("Analyst",4), ("Developer",2),("Specialist",2)]
```

10. Write an Haskell function **ascending** to check if a list of numbers is ordered in ascending order. Number can be repeated in the list!

```
ascending [1,4,5,7,10] = True

ascending [1,4,2,7,1] = False
```

HIGHER ORDER FUNCTIONS

11. Write an Haskell function **max_f** to compute the maximum of a function in the interval *[a,b]*, where *a* and *b* are integer (positive or negative number) with *a<b*, using a 0.1 interval accuracy. The first input of the function is a function that computes a real number, the second and third parameters are *a* and *b* (defining the interval).

```
max_f h 0 1 = 3  (for instance, h is a function computing x*x+2)
```

12. A sequence is a function accepting only positive integer numbers as inputs. Write an Haskell function **higher_f** to check if a sequence is always higher than a second one in an interval *[a,b]*, where *a* and *b* are positive integer numbers and *a<b*. A sequence A is higher than sequence B if, for all the integer in the interval [a,b], the value of A is higher than the values of B. The input of the functions are the two sequences and the two point *a* and *b*. The output is Boolean (False/True)

`higher_f h g 0 5 = False` (h is x*x and g is x+2. Since x*x is not always higher than x+2 in [0..5]. In fact when x=0 or x=1 the function x+2 is higher, they have the same value if x=2 and x*x is higher for x>2)

`higher_f h g 2 5 = True` (h is x*x and g is x. Since function h is always bigger than function g in the interval [2..5])