TP 1 : Premiers pas

1. **(basic data type `Int`)** On considère des nombres entiers de type `Int`.
   Définir :
   - la fonction `square` qui élève un nombre au carré,
   - la fonction `sumSquares` qui calcule la somme des carrés de deux nombres,
   - la fonction `sumSquaresMax` qui, à trois nombres, associe la somme des carrés des deux plus grands. Exemples :

     ```
     > sumSquaresMax 5 6 7  ==> 85
     > sumSquaresMax 7 6 5  ==> 85
     > sumSquaresMax 6 7 5  ==> 85
     ```

2. **(basic data type `Float`)** Conversions entre températures exprimées en degrés Celsius et en degrés Fahrenheit. Le lien entre les deux est donné par l'équation : $C = 5/9(F - 32)$. D'où la fonction de conversion :

   ```
   c2f :: Float -> Float
   c2f c = 9 / 5 * c + 32
   ```

   Définir la fonction réciproque `f2c`

3. **(basic data type `Float`)** Définir une fonction qui prend comme paramètres un nombre de kilomètres et un nombre de personnes, et retourne le montant à rembourser, sachant que :
   - le tarif appliqué est de 0.52 F du km,
   - une réduction de 25 % est appliquée sur le prix au km entre 2 et 4 personnes,
   - une réduction de 50 % est appliquée sur le prix au km entre 5 et 10 personnes,
   - une réduction de 75 % est appliquée sur le prix au km à partir de 11 personnes.

   ```
   > travelExpenses 150 3   ==> 175.5
   > travelExpenses 100 7   ==> 182.0
   > travelExpenses 90 3  ==> 105.299995
   ```

   Indication. On pourra utiliser la fonction ci-dessous après l'avoir complétée :

   ```
   reductionRate :: Float -> Float
   reductionRate nbPers
      | nbPers < 2  = 0
      | nbPers < 5  = 0.25
      |    ?   < ?  = ?
      | otherwise   = ?
   ```

4. **(basic data type `Char`)** Définir la fonction `(nextUpperCase c)` qui prend une lettre majuscule et retourne la lettre majuscule suivante; on fera en sorte que `'A'` suive immédiatement `'Z'`.

```
> nextUpperCase 'V'  ==> 'W'
> nextUpperCase 'Z'  ==> 'A'
```

Indication. On pourra utiliser les deux fonctions suivantes :

```
decode :: Int -> Char          code :: Char -> Int
decode n = toEnum n            code c = fromEnum c
```

5. **(`Int` and list of `Int`)** La fonction de Collatz est définie pour n entier, $n \geq 2$ par

$$collatz(n) = \begin{cases} 1 & \text{si } n = 2 \\ \frac{n}{2} & \text{si n est pair} \\ 3n+1 & \text{si n est impair} \end{cases}$$

La conjecture de Collatz affirme que :

$$\forall n \geq 2, \exists k \in N^*, \underbrace{collatz(...(collatz(n))...)}_{k fois} = 1$$

(a) Définir la fonction `(collatz n)`.

(b) Définir la fonction `(nbCalls n)` qui calcule le nombre d'applications nécessaires de la fonction `collatz` pour atteindre la valeur `1` en partant de `n`.

```
> nbCalls 16  ==> 4
> nbCalls 3  ==> 7
```

Indication. Compléter la définition : `nbCalls n = if (n == 1) then 0 else (1 + ?)`

(c) De manière analogue, définir la fonction `(syracuse n)` qui construit, pour `n` donné, la liste des valeurs successives permettant d'atteindre `1`.

```
> syracuse 16  ==> [16, 8, 4, 2]
> syracuse 3  ==> [3, 10, 5, 16, 8, 4, 2]
```

(d) En déduire une seconde définition de la fonction `(nbCalls n)`

(e) Que se passerait-il si la conjecture de Collatz était fausse ?

6. **(list of `Bool`)** Définir récursivement la fonction `(allEven xs)` qui détermine si la liste `xs` est uniquement composée de nombres pairs.
Exemples :

```
> allEven [2,4,6,8,10]  ==> True
> allEven [2,3,6,8]  ==> False
```

7. **(list of `Char`)** Définir la fonction `(laugh n)` qui retourne la chaîne contenant `n` occurrences de l'interjection `"HA "`.
   Exemples :

   ```
   > laugh 3  ==> "HA HA HA "
   > laugh 1  ==> "HA "
   ```

8. **(list of `Char`)** Définir la fonction `(double xs)` qui "double" chaque chaîne de `xs`.
   Exemples :

   ```
   > double ["je", "be", "gaye"] ==> "je je be be gaye gaye "
   > double ["stut", "te", "ring"]  ==> "stut stut te te ring ring "
   ```

9. **(ZF-expressions)** En utilisant les listes en compréhension, définir la fonction `(sumSquareEven n)` qui calcule la somme des carrés des `n` premiers nombre pairs.
   Exemples :

   ```
   > sumSquareEven 1  ==> 4
   > sumSquareEven 2  ==> 20
   > sumSquareEven 8  ==> 816
   ```

   Indication. Evaluer, par exemple, l'expression `[x | x <- [1..6], even x]`

10. **(ZF-expressions)** Que fait la fonction suivante :

    ```
    mystery :: [Int] -> [Int]

    mystery []     = []
    mystery (x:xs) = mystery [y | y <- xs, y <= x]
                     ++  [x]
                     ++ mystery [y | y <- xs, y > x]
    ```