heig-vd

Laboratoire 1

Haskell : Expressions, Fonctions, Listes et Tuples

heig-vd

Haskell en pratique

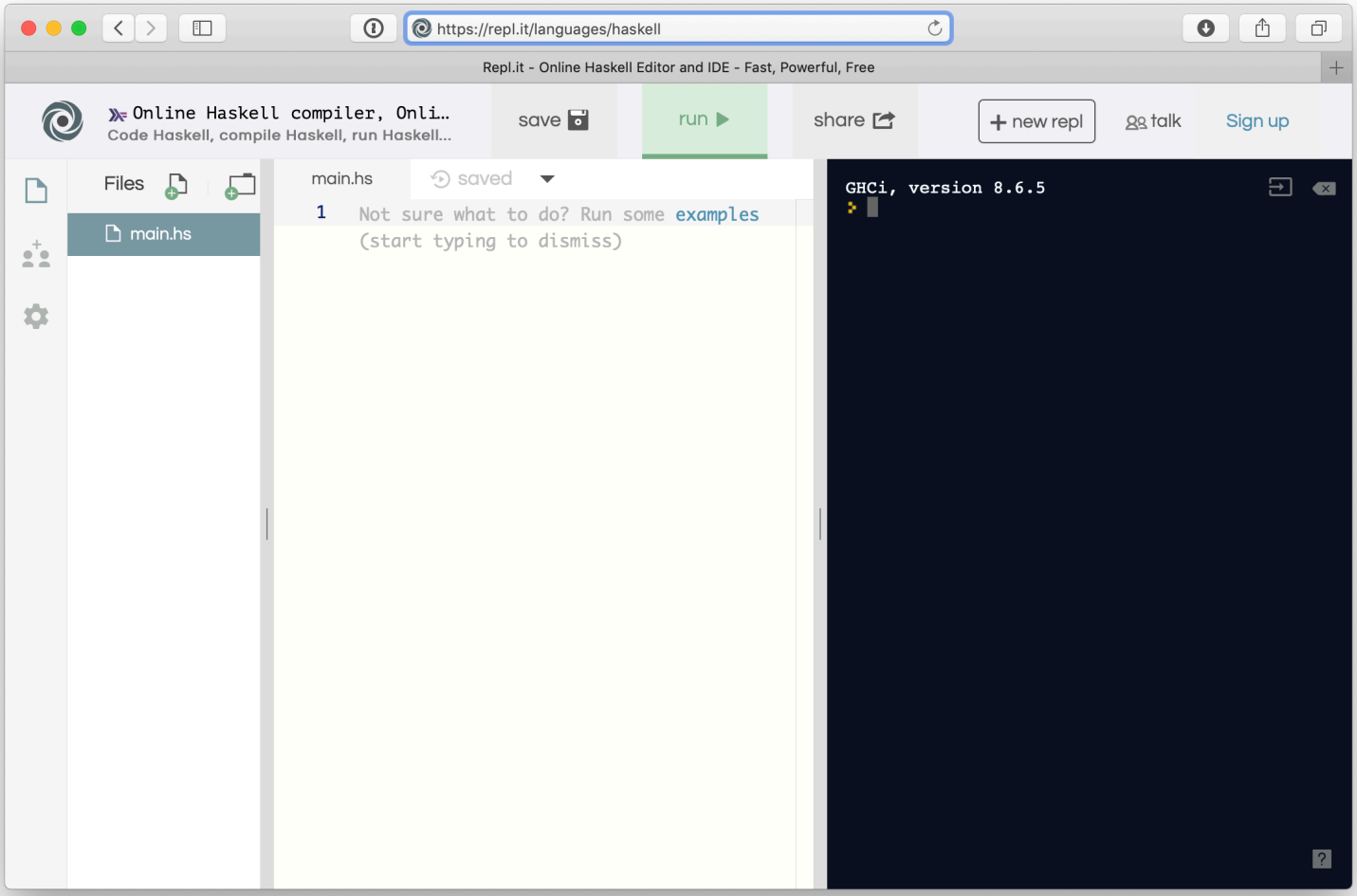
- https://www.haskell.org
- Installer GHCi

```
MacBook-de-Marc:~ marc$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Prelude> :l test
[1 of 1] Compiling Main             ( test.hs, interpreted )
Ok, one module loaded.
*Main> inc 2
3
*Main>
```

❗ Le répertoire de base est celui depuis lequel ghci est lancé,
c'est là qu'il trouve les fichiers.

heig-vd

Alternative



heig-vd

Installation GHCi

- macOS
  - installer home brew (https://brew.sh/index_fr)
  - installer xz : `brew install xz`
  - installer GHCi : (https://www.haskell.org/ghcup/)
  - ajouter export `PATH=$PATH:~/.cabal/bin:~/.ghcup/bin` à `~/.bash_profile`
- Windows
  - instructions bienvenues ,-)

heig-vd

Commandes GHCi

:l <fichier>	charger un fichier
:r	recharger le fichier
:quit	quitter
:cd <répertoire>	changer de répertoire
:?	liste des commandes

heig-vd

Exercice 1.1 - Expressions

1. Quel est le résultat des expressions ci-dessous ?

   Attention, certaines sont incorrectes, lesquelles ? Pourquoi ?

```
1. 'Z' < 'a'
2. "abc" <= "ab"
3. if 2 < 3 then 3
4. [1,3,3,6] < [1,3,4]
5. 4. + 3.5
6. 5.0 - 4.2 / 2.1
7. if 4 > 4.5 then 3.0 else 'a'
8. 3 > 4 || 5 < 6 && not (7 /= 8)
9. if 6 < 10 then 6.0 else 10.0
```

1. True2. False3. Error missing else part4. True5. Error w/ 4. it should be 4.06. 3.07. Error

heig-vd

Exercice 1.2 - Listes

1. Donnez les résultats des expressions ci-dessous

   Attention, certaines sont incorrectes, lesquelles ? Pourquoi ?

```
1. [1,2,3] !! ([1,2,3] !! 1)
2. head []
3. tail [1,2,3]
4. "a":["b","c"]
5. "abc" ++ "d"
6. tail "abc" ++ "d"
7. [1] : [2,3,4]
8. ([1,2,3] !! 2 : []) ++ [3,4]
9. [3,2] ++ [1,2,3] !! head [1,2] : []
```

1. 32. Error head doesnt work on empty lists3. [2,3]4. ["a", "b", "c"]5. abcd6. bcd

heig-vd

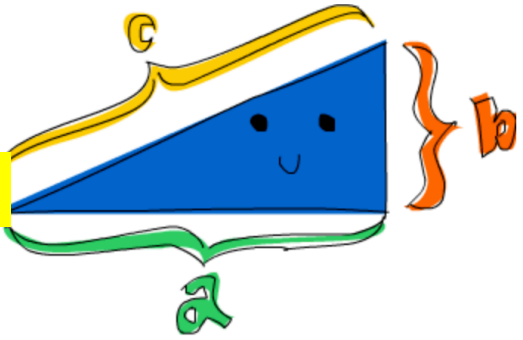
Exercice 1.3 - Listes en compréhension

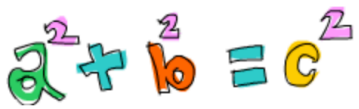
1. Définissez les listes suivantes en compréhension
    1. Les nombres impairs de 1 à 20 (extension) `[1,3..20]`
    2. Les nombres pairs de 1 à 20 (compréhension) `[2*x | x <- [1..10]]`
    3. Les carrés parfaits entre 1 et 100 `[x*x | x <- [1..100], x*x < 100]`
    4. Les paires (x,y) telles que x<7, y<7, x<y

       `[(x, y) | x < 7, y < 7, x < y, x <- [1..7], y <- [1..7]]`
    5. Améliorez votre solution de paires pour ne pas répéter [1..7]

       `let a = [1..7] in [(x,y) | x <- a, y <- a, x <7, y <7, x<y]`
    6. Les triplets (x,y,z) d'**entiers** inférieurs à 100 qui représentent la longueur des trois côtés d'un triangle rectangle.

       `let a = [1..100] in [(x,y,z) | x <- a, y <- a, z <- a, x*x + y*y == z*z]`





heig-vd

Exercice 1.4 - Fonctions

1. Définissez et testez les fonctions suivantes:
   1. `addition x y z` -- Somme de x, y et z.

```
add x y z = x + y + z
```

   2. `carre x` -- Calcul de x au carré.

```
square x = x*x
```

   3. Utilisez la fonction `carre` dans l'exercice précédent (triplets).

```
let a = [1..100] in [(x,y,z) | x <- a, y <- a, z <- a, square x
```

   4. `ou p q` -- Le *ou* logique des prédicats `p` et `q` **sans** utiliser l'opérateur `||` bien sûr !

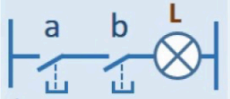
```
or p q = if p then p else q
```

   5. `et p q` -- Le *et* logique des prédicats `p` et `q` **sans** utiliser l'opérateur `&&` !

```
and p q = if p then q else p
```

   6. Utilisez les listes en compréhension pour lister la table de vérité de vos fonctions `or` et `and` (sous forme de triplets).

```
[(a, b, a `and` b)| a <- [False, True], b <- [False, True]][(a, b, a `or` b) | a <- [False, T
```

Nom	Schéma à contact	Table de vérité
ET ou AND		a b L 0 0 0 0 1 0 1 0 0 1 1 1

heig-vd

Exercice 1.5 - Chaines

1. Il existe des fonctions `sum` et `product` sur des listes,
   font-elles ce que vous soupçonnez ? oui

2. Les fonctions `drop` et `take` permettent respectivement d'éliminer ou de retenir les n premiers éléments d'une liste.
   Comment utiliser ces fonctions pour supprimer le mot *Hello* au début d'une chaine (on sait que la chaine commence par Hello) ? drop (length "Hello") "Hello world"if take (length "Hello"
   Pour tester si la chaine commence par *Hello* ? take (length "Hello") "Hello world" == "Hello"

3. Je suis allergique à l'américain, écrivez une fonction qui supprime le mot *Hello* s'il figure début d'une chaine. let a = "Hello world" in if (take (length "Hello") a == "Hello") then drop (length "Hello") a else a

4. La phrase résultante manque de majuscule, pouvez vous corriger ça ? Utilisez `toUpper` de `Data.Char` : `import Data.Char`

   let c = "bla" in toUpper (head c) : tail c

heig-vd

Exercice 1.6 - Tuples

1. Ecrivez une fonction `tuple n` qui retourne un triplet avec les carrés de n, n+1, n+2.

   ```
   let tuple n f = (f n, f (n + 1), f (n + 2))// better version :tuple n = let sqr x = x*x in (sqr n, sqr (n + 1), sqr (
   ```

   Utilisez votre propre fonction pour calculer le carré d'un nombre à l'aide d'un énoncé `let`.

2. On construit une liste avec des tuples, par exemple des noms et des ages.

   ```
   people = [("Doran", 24), ("Bryan", 23), ("Yasmine", 24)]age nom liste = head [y
   ```

   Ecrivez une fonction qui donne l'âge en fonction du nom.

   Pouvez vous simplifier avec un `let` ?

   Que se passe t'il si le nom n'est pas dans la liste ?

   ```
   une liste vide est retourné
   ```