+1/1/60+

INF6120 **QUIZ 1 (5%)** **Mardi 2019-10-08**
Programmation fonctionnelle et logique **Alexandre Blondin Massé**

☐0 ☐0 ☐0 ☐0 ☐0 ☐0 ☐0 ☐0
☐1 ☐1 ☐1 ☐1 ☐1 ☐1 ☐1 ☐1
☐2 ☐2 ☐2 ☐2 ☐2 ☐2 ☐2 ☐2
☐3 ☐3 ☐3 ☐3 ☐3 ☐3 ☐3 ☐3
☐4 ☐4 ☐4 ☐4 ☐4 ☐4 ☐4 ☐4
☐5 ☐5 ☐5 ☐5 ☐5 ☐5 ☐5 ☐5
☐6 ☐6 ☐6 ☐6 ☐6 ☐6 ☐6 ☐6
☐7 ☐7 ☐7 ☐7 ☐7 ☐7 ☐7 ☐7
☐8 ☐8 ☐8 ☐8 ☐8 ☐8 ☐8 ☐8
☐9 ☐9 ☐9 ☐9 ☐9 ☐9 ☐9 ☐9

⟵ Codez les 8 chiffres de votre code permanent ci-contre, et inscrivez-le à nouveau ci-dessous avec vos nom et prénom.

Code permanent :

.....

Prénom :

.....

Nom :

.....

Une unique bonne réponse par question. Vous pouvez noircir les cases ou faire des croix (×) (pas de ✓ni de cercle).

Q1 Sachant qu'on a les types (signatures) suivants

```
iterate :: (a -> a) -> a -> [a]
length :: [a] -> Int
flip :: (a -> b -> c) -> b -> a -> c
```

quel est le type de l'expression

```
length . ((flip iterate) 'a')
```

?

- ☐ `Char -> Char -> Int`
- ☐ `(Char -> Char) -> Int`
- ☐ `Int`
- ☐ `Char -> Int`

Q2 Considérez la déclaration suivante :

```
data Saison = Automne | Ete
            | Hiver   | Printemps
  deriving (Eq, Ord, Show, Enum, Bounded)
```

Quelle expression parmi les suivantes retourne `True` ?

- ☐ `pred Printemps > succ Automne`
- ☐ `Ete /= Ete`
- ☐ `Printemps <= Automne`
- ☐ `minBound :: Saison`

Q3 Considérez la fonction `f` définie par

```
f :: (a -> Bool) -> [a] -> [a]
f _ [] = []
f p (x:xs)
    | p x       = x:(f p xs)
    | otherwise = f p xs
```

Quelle fonction Haskell parmi les suivantes est équivalente à la fonction `f` ?

- ☐ `sort`
- ☐ `map`
- ☐ `filter`
- ☐ `reduce`
- ☐ `zip`
- ☐ `remove`

Q4 Quel est le résultat de l'expression suivante ?

```
foldr (/=) True [True,False,False]
```

- ☐ `True`
- ☐ `False`
- ☐ `[False]`

Q5 Quel est le résultat de l'expression

```
zipWith (:) "ab" (repeat "ab")
```

?

- ☐ `"abab"`
- ☐ `["aa","bb"]`
- ☐ `["aab","bab"]`
- ☐ une boucle infinie

Q6 Quel est le résultat de l'expression suivante ?

```
head $ map (+1) $ take 5 $ drop 3 [100..110]
```

- ☐ 103
- ☐ 101
- ☐ 104
- ☐ 105

Q7 Considérez les déclarations suivantes

```
data Arbre a = ArbreVide | Noeud a [Arbre a]
    deriving (Show)

arbre = Noeud 'a' [ArbreVide,Noeud 'b' [],ArbreVide]
```

Proposez une instance de `Functor` pour le constructeur de type `Arbre` qui permet d'appliquer une fonction à tous les noeuds. On s'attend au comportement suivant :

```
>>> fmap (:"c") arbre
Noeud "ac" [ArbreVide,Noeud "bc" [],ArbreVide]
```

☐e ☐pr ☐c ☐m ☐n

Répondre au verso ⟶

Pour votre examen, imprimez de préférence les documents compilés à l'aide de auto-multiple-choice.