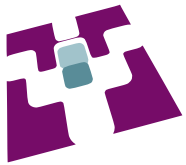
UNIVERSITÉ
CAEN
NORMANDIE

Autre Paradigme

Patrice Boizumault

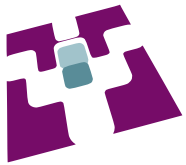
Université de Caen - Normandie

Lundi 6 janvier 2020



Partie #1 : Premiers Pas en `Haskell`

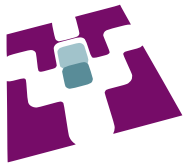
1. Evaluer une expression
2. Définir une fonction en donnant son type et ses équations
3. Un coup d'oeil aux listes
4. Exemple : nombres premiers inférieurs à un nombre k fixé



Evaluer une expression

- **Nombres Entiers (types `Int` et `Integer`)**

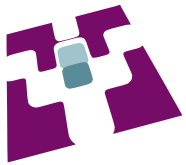
```
> 6*7   ==> 42

> 7*4 + 3    ==> 31

> div 13 4    ==> 3

> mod 13 4    ==> 1
```



Evaluer une expression

- **Nombres Flottants (types `Float` et `Double`)**

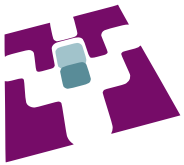
```
> sqrt 65    ==> 8.06225774829855

> pi    ==> 3.141592653589793

> sin (pi/2)    ==> 1.0

> cos (pi/4)    ==> 0.7071067811865476
```



Evaluer une expression

- **Booléens (type `Bool`)**

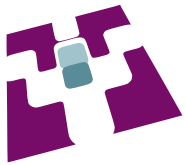
```
> True && False   ==> False

> True || False   ==> True

> 2 == 1+1   ==> True

> 5 > 3*4   ==> False
```



Evaluer une expression

- **Caractères (type `Char`)**

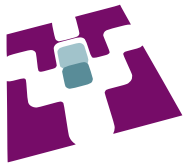
```
> 'a'   ==> a'

> 'A' < 'a'  ==> True

> succ 'b'  ==> 'c'

> pred 'z'  ==> 'y'
```



Evaluer une expression

- **tuples**

```
> (2^5, True || False, 'a')   ==> (32, True, 'a')

> (pi/2, 'A' < 'a')   ==> (1.5707963267948966, True)

> ("Lundi", 9, "Janvier", 2018)

      ==> ("Lundi", 9, "Janvier", 2018)
```



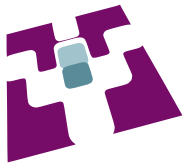
Evaluer une expression

- **Liste d'entiers : type `[Int]`**

```
> [1..5]   ==> [1,2,3,4,5]

> [2+3, 7-4, div 72 7, 23]   ==> [5,3,10,23]
```

- **Liste de booléens : type `[Bool]`**

```
> [1 < 2, True, 'a'=='b'] => [True,True,False]
```



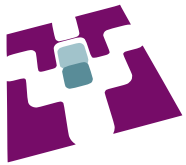
Evaluer une expression

- **Liste de caractères : type `String`**

```
> "hello " ++ "world"   ==> "hello world"

> ['h', 'e', 'l', 'l', 'o']   ==> "hello"
```

- **`String` et `[Char]` sont des types synonymes**

```
> "hello" == ['h', 'e', 'l', 'l', 'o']   ==> True

> "hello " == ['h', 'e', 'l', 'l', 'o', ' ']   ==> True
```



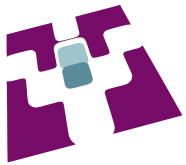
Définir une fonction

- **Carré d'un entier**

```
square :: Int -> Int

square x = x*x
```

- **Carré du carré**

```
quad :: Int -> Int

quad x = square (square x)
```



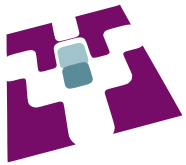
Définir une fonction

- **Carré du carré**

```
quad :: Int -> Int

quad x = square (square x)
```

- **Composition de fonctions : opérateur infixe `(.)`**

```
quadBis :: Int -> Int

quadBis = square . square
```



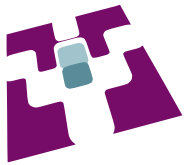
Définir une fonction

- **Minimum de deux entiers**

```
smallest :: Int -> Int -> Int

smallest x y = if (x < y) then x else y
```

- **Moyenne arithmétique de deux flottants**

```
average :: Float -> Float -> Float

average x y = (x+y)/2
```



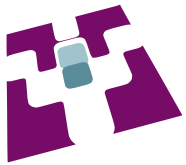
Définir une fonction

- **Factorielle d'un entier**

```
fact :: Int -> Int

fact n = if (n == 0) then 1 else n*(fact (n-1))
```

- **Nombre de chiffres représentant un nombre entier**

```
nbDigits :: Int -> Int

nbDigits n = if (n <= 9) then 1 else 1 + (nbDigits (div n 10))
```



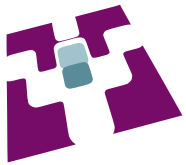
Un coup d'oeil aux listes

- **Listes vs Ensembles**

```
> [1,1] == [1]     ==> False

> length [1]     ==> 1

> length [1, 1]     ==> 2

> [1, 2] == [2, 1]     ==> False
```

- **Listes en abrégé**

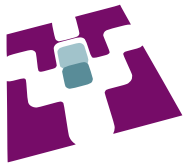
```
> [1..4]     ==> [1,2,3,4]

> length [9..14]     ==> 6
```



Un coup d'oeil aux listes

- **Listes en compréhension (ZF-expressions)**

```
> [x | x <- [1..10]]    ==> [1,2,3,4,5,6,7,8,9,10]

> [x | x <- [1..10], even x]    ==> [2,4,6,8,10]

> [x * x | x <- [1..10], even x]    ==> [4,16,36,64,100]


> [(x,y) | x <- [1..2], y <- [5..7]]

        ==> [(1,5),(1,6),(1,7), (2,5),(2,6),(2,7)]
```



Exemple : nombres premiers inférieurs à k

- **Un nombre est-il premier ?**

```
listDivisors :: Int -> [Int]
listDivisors n = [i | i <- [1..n], (mod n i) == 0]

isPrime :: Int -> Bool
isPrime n = (listDivisors n) == [1,n]
```

- **La liste et combien sont-ils ?**

```
primesLessThan :: Int -> [Int]
primesLessThan k = [n | n <- [1..k], isPrime n]

nbPrimesLessThan :: Int -> Int
nbPrimesLessThan k = length (primesLessThan k)
```