Travaux dirigés – Feuille d'exercices n°1

Exercice I : Premières fonctions

Q1 – Définir une fonction `double` qui calcule l'entier $2x$ à partir de l'entier x. Quel est le type de cette fonction ?

```
# (double 3);;
- : int = 6
```

Q2 – En utilisant la fonction `double` pour calculer les entiers $2x$ et $2y$, définir une fonction `somme_double` qui calcule $2x + 2y$ à partir des entiers x et y. Quel est le type de cette fonction ?

```
# (somme_double 3 4);;
- : int = 14
```

Q3 – Redéfinir la fonction `somme_double` en utilisant une seule fois la fonction `double`.

Q4 – En utilisant la fonction `double`, définir une fonction `is_double` qui étant donnés deux entiers x et y retourne `true` si $y = 2x$ et `false` sinon. Quel est le type de cette fonction ?

```
# is_double 2 3;;
- : bool = false
# is_double 2 4;;
- : bool = true
```

Q5 – En utilisant les fonctions `double` et `is_double`, définir une fonction `something_is_double` qui étant donnés deux entiers x et y retourne `true` si $y = 2x$ ou $y = 4x$ et retourne `false` sinon. Quel est le type de cette fonction ?

```
# (something_is_double 2 4);;
- : bool = true
# (something_is_double 2 8);;
- : bool = true
# (something_is_double 2 16);;
- : bool = false
```

Q6 – Définir une fonction `make_even` qui étant donné un entier x retourne x si x est pair est retourne $2x$ sinon. Quel est le type de cette fonction ?

```
# (make_even 6);;
- : int = 6
# (make_even 5);;
- : int = 10
```

On pourra utiliser la fonction prédéfinie `mod` qui calcule le reste de la division entière de deux entiers :

```
# 5 mod 2;;
- : int = 1
# 4 mod 2;;
- : int = 0
```

Exercice II : Expressions alternatives, expressions booléennes

Q1 – Montrez que les expressions

1. `if a then true else (f a)`
2. `if a then a else (f a)`
3. `a || (f a)`

ont les mêmes valeurs pour tout booléen `a` et toute fonction booléenne (unaire) `f`.

Q2 – Montrez que les expressions

1. `if a then (f a) else false`
2. `if a then (f a) else a`
3. `a && (f a)`

ont les mêmes valeurs pour tout booléen `a` et toute fonction booléenne (unaire) `f`.

Exercice III : Expressions fonctionnelles, application partielle

On considère les fonctions `square` et `is_greater_square` définies par :

```
let square = fun x -> x*x;;
let is_greater_square = fun x -> fun y -> y >= (square x);;
```

Q1 – Quel est le résultat de l'évaluation de l'expression `(square 4)` ? Quel est le type de la fonction `square` ?

Q2 – Quel est le résultat de l'évaluation des expressions `(is_greater_square 4 10)` et `(is_greater_square 4 18)` ? Quel est le type de la fonction `is_greater_square` ?

Q3 – Quel est le type de l'expression `(is_greater_square 4)` ? Que désigne la valeur de cette expression ?

Q4 – On définit la fonction `foo` comme suit :

```
let foo = (is_greater_square 4);;
```

Quel est le type de cette fonction ? Quel est le résultat de l'évaluation des expressions `(foo 10)` et `(foo 18)`? Que fait cette fonction ?

Q5 – Définir une fonction `make_is_greater_square` dont l'argument est un entier n et dont le résultat est une fonction qui étant donné un entier x retourne `true` si $x \geq n^2$ et `false` sinon. Quel est le type de cette fonction ?

```
# ((make_is_greater_square 4) 10);;
- : bool = false
# ((make_is_greater_square 4) 18);;
- : bool = true
```

Q6 – Redéfinir la fonction `foo` en utilisant la fonction `make_is_greater_square`.

Exercice IV : À faire à la maison

Cette petite série d'exercices vous permettra de vérifier si vous avez assimilé ou non l'utilisation des parenthèses pour écrire les applications de fonctions. Vous pouvez les faire en utilisant une machine. L'objectif est que vous compreniez pourquoi certaines expressions sont correctes et d'autres non.

Q1 – Soit `f : int -> bool -> int -> int`

Parmi les expressions ci-dessous, dites lesquelles sont des applications correctes de la fonction `f`.

1. `f(3,true,5)`
2. `f(3 true 5)`
3. `(f (3 true 5))`
4. `(f 3 true 5)`
5. `(f 3 5 true)`
6. `(f true 3 5)`
7. `(f 3 (3 < 5) 5)`
8. `(f 2+1 true 5)`
9. `(f (2+1) true 5)`
10. `(f 5 true f 4 false 3)`
11. `(f 5 true (f 4 false 3))`
12. `((f 5 true 4) 3)`

Q2 – On considère encore `f : int -> bool -> int -> int`. Même question que ci-dessus. Donnez en plus le type de l'expression.

1. `(f 4 true) 3`
2. `((f 4) true 3)`
3. `(f) 4 true 3`
4. `(f 4 true)`
5. `((f 4 true) 3)`
6. `(((f 4) true) 3)`

Q3 – On définit `f` de la façon suivante:

```
let f (x:int) (y:bool) (z:int) : int =
  if y then x else z
```

Les applications suivantes sont-elles correctes ? Si oui, quelles sont les valeurs obtenues ?

1. `(f 4 true 2+1)`
2. `(f 4 true (2+1))`
3. `(f 4 true 2) + 1`

Q4 – Pour chacune des expressions suivantes, dire si elle est une expression valide du langage OCaml, dans le cas où l'expression est valide, donner son type et sa valeur ainsi que les étapes de calcul de cette valeur dans le modèle équationnel. Dans l'autre cas, expliquez l'erreur.

1) `((fun x -> (2 * x)) 7)`

2) `((fun x -> (2 + x)) (3 * 5))`

3) `((fun x -> (2 + x)) 3 * 5)`

4) `(((fun x -> (2 + x)) 3) * 5)`

5) `(((fun x -> (fun y -> (x * y))) 3) 7)`

6) `((fun x -> fun y -> (x * y)) (3 7))`

7) `((fun x -> fun y -> (x * y)) 3 7)`

8) `((fun f -> (f 2)) 7)`

9) `((fun f -> fun x -> (f x) + x) (fun x -> (x - 1)) 3)`