université de BORDEAUX	**Année universitaire :** 2025/2026 **Parcours :** Licence Informatique 3e année UE 4TIN514U **Épreuve :** Devoir surveillé de Programmation fonctionnelle **Date :** mercredi 12 novembre 2025 8 :00 – 9 :20 **Durée :** 1h20 **Documents interdits.**	**Collège Sciences et Technologies**

Le sujet comporte 4 pages. Tous les types des fonctions sont donnés Page 4.

Exercice 1 *(3pts)* Pour chacune des expressions suivantes, donner sa **valeur** et son **type** si elle est correcte, sinon expliquer pourquoi elle est incorrecte.

1. `let x = 2 in x * (let x = 3 + x in let y = -x in x + y)`
2. `let x = 3 in let x = x + 1 and y = 2 * x in y - x`
3. `fun x y -> (x * y, x + y)`
4. `let f x y = 2 * int_of_float x + 3 * y`
5. `f 2.` où `f` est la fonction définie précédement.
6. `f 4. 3` où `f` est la fonction définie précédement.

Exercice 2 *(2pts)* Pour chacun des types suivants, donner une expression ayant ce type.

7. `int -> float -> float`
8. `int -> float -> int * bool`
9. `('a -> bool) -> ('a -> 'a) -> 'a -> 'a`
10. `('a -> int) -> 'a -> int`

Exercice 3 *(4pts)* Soit la fonction `mystery` suivante :

```
let rec mystery n =
  if n = 0 then 2
  else if n = 1 then 3
  else mystery (n - 1) + mystery (n - 2) + 2
```

11. Que retourne l'appel `mystery 3` ?
12. Donner la complexité de la fonction `mystery` en justifiant la réponse.
13. Donner une version linéaire ($O(n)$) `mystery_lin` de cette fonction.

Exercice 4 *(4pts)* Une suite $U_{i\in\mathbb{N}}$ à valeur dans un ensemble $\mathbb{E}$ peut être définie par une fonction *seq* de $\mathbb{N}$ vers $\mathbb{E}$. Pour obtenir l'élément u_i, il suffira d'appliquer *seq* à i.

Par exemple, la suite constante telle que $\forall i \in \mathbb{N}$, $u_i = 4$ peut être définie par `fun (i : int) -> 4`.

On peut donc définir la fonction `seq_constant c` qui retourne la suite de valeur constante `c` de la manière suivante :

```
# let seq_constant c = fun (i : int) -> c;;
val seq_constant : 'a -> int -> 'a = <fun>
# let seq_3 = seq_constant 3;;
val seq_3 : int -> int = <fun>
# seq_3 0;;
- : int = 3
# seq_3 10;;
- : int = 3
```

14. Définir une variable `seq_i` contenant la suite telle que $\forall i \in \mathbb{N}$, $u_i = i$.

Exemples :

```
# seq_i;;
- : int -> int
# seq_i 0;;
- : int = 0
# seq_i 10;;
- : int = 10
```

15. Définir une variable `seq_even` contenant la suite telle que $\forall i \in \mathbb{N}$, $u_i = 2i$.
    Exemples :

```
# seq_even;;
- : int -> int = <fun>
# seq_even 3;;
- : int = 6
# seq_even 10;;
- : int = 20
```

On définit l'*addition* sur les suites.
Soient U et V deux suites, la somme de ces suites $W = U + V$ est définie par $\forall i \in \mathbb{N}$, $w_i = u_i + v_i$.

16. Définir la fonction `seq_sum seq1 seq2` qui retourne la suite somme des suites `seq1` et `seq2`.

Exemples :

```
# let w = seq_sum seq_3 seq_even;;
val w : int -> int = <fun>
# w 0;;
- : int = 3
# w 4;;
- : int = 11
```

17. **En utilisant les fonctions** `seq_sum`, `seq_constant` et `seq_even`, définir la variable `seq_odd` contenant la suite telle que $\forall i \in \mathbb{N}$, $u_i = 2i + 1$.

Exemples :

```
# seq_odd;;
- : int -> int = <fun>
# seq_odd 0;;
- : int = 1
# seq_odd 2;;
- : int = 5
```

18. Définir la fonction `seq_to_list seq n p` qui retourne la liste $[u_n, \ldots, u_p]$.
    Un **bonus** sera attibué si la fonction écrite est **récursive terminale**.

Exemples :

```
# seq_to_list seq_odd 8 3;;
- : int list = []
# seq_to_list seq_odd 3 8;;
- : int list = [7; 9; 11; 13; 15; 17]
```

Exercice 5 *(8pts)*

La méthode des trapèzes permet le calcul numérique d'une intégrale $\int_a^b f(x)\,dx$. Le principe est d'assimiler la région sous la courbe représentative de la fonction f définie sur un segment $[a, b]$ à un trapèze et d'en calculer l'aire :

$$(b-a)\frac{f(a)+f(b)}{2}. \tag{1}$$

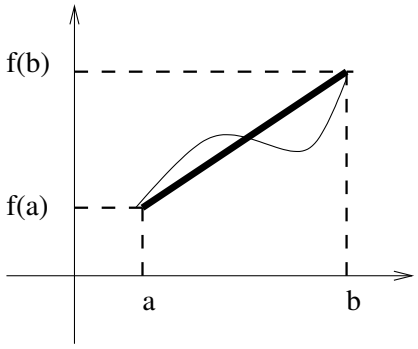

FIG. 1 : Approximation de l'aire par un trapèze

19. Écrire une fonction `trapeze_area a b fa fb` qui calcule l'aire du trapèze représenté Figure 1 en utilisant la formule (1).
Exemple :

```
# trapeze_area 1. 2. 10. 20.;;
- : float = 15.
```

20. En utilisant la fonction `trapeze_area`, écrire la fonction `area_trapeze_method f a b n` qui calcule de manière approchée l'aire sous la courbe `f` et entre `a` et `b` en découpant l'aire en `n` trapèzes comme présenté Figure 2.

Exemples :

```
# area_trapeze_method (fun x -> x) 0. 1. 1;;
- : float = 0.5
# area_trapeze_method (fun x -> x *. x) 0. 1. 1;;
- : float = 0.5
# area_trapeze_method (fun x -> x *. x) 0. 1. 10;;
- : float = 0.335
# area_trapeze_method (fun x -> x *. x) 0. 1. 100;;
- : float = 0.333350000000000035
```

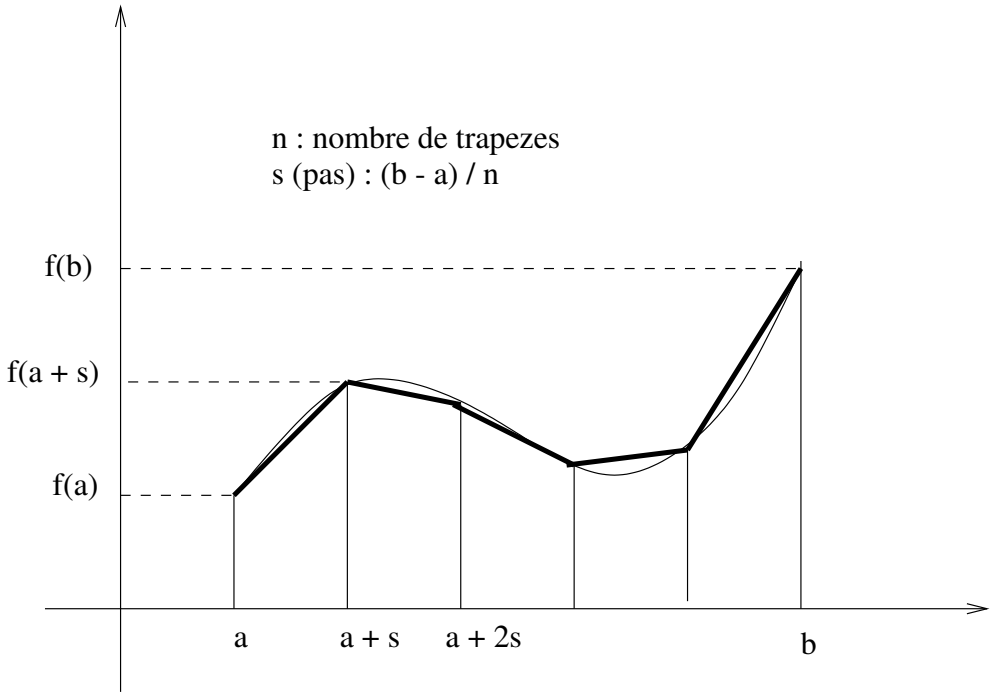

FIG. 2 : Approximation de l'aire par la méthode des trapèzes

21. En utilisant la fonction `area_trapeze_method`, écrire une fonction `integrale f n` qui retourne la fonction qui à `a` et `b` associe l'aire approchée sous la courbe `f` entre `a` et `b`.

Exemples :

```
# integrale (fun x -> x) 10;;
- : float -> float -> float = <fun>
# integrale (fun x -> x) 10 0. 1.;;
- : float = 0.5
# integrale (fun x -> x *. x) 10 0. 1.;;
- : float = 0.335
```

22. En utilisant la fonction `integrale f n`, écrire la fonction `primitive f a n`, qui calcule la fonction $x \mapsto \int_a^x f(x)\,dx$.

Exemples :

```
# primitive (fun x -> 2. *. x) 0. 10;;
- : float -> float = <fun>
# primitive (fun x -> 2. *. x) 0. 10 6.;;
- : float = 36.
# (fun x -> x *. x) 6.;;
- : float = 36.
# primitive (fun x -> 3. *. x *. x) 0. 100 6.;;
- : float = 216.010800000000074
# (fun x -> x *. x *. x) 6.;;
- : float = 216.
```

Remarques :
La primitive de la fonction $x \mapsto 2x$ est $x \mapsto x^2$.
La primitive de la fonction $x \mapsto 3x^2$ est $x \mapsto x^3$.

Types des fonctions

```
val mystery : int -> int = <fun>
val mystery_lin : int -> int = <fun>
val trapeze_area : float -> float -> float -> float -> float = <fun>
val area_trapeze_method :
    (float -> float) -> float -> float -> int -> float =
  <fun>
val integrale : (float -> float) -> int -> float -> float -> float = <fun>
val primitive : (float -> float) -> float -> int -> float -> float = <fun>
val seq_constant : 'a -> int -> 'a = <fun>
val seq_i : int -> int = <fun>
val seq_even : int -> int = <fun>
val seq_sum : (int -> int) -> (int -> int) -> int -> int = <fun>
val seq_odd : int -> int = <fun>
val seq_to_list : (int -> 'a) -> int -> int -> 'a list = <fun>
```

FIN