Université Nice Sophia-Antipolis
L1 Info / Math-Info
Algorithmique et Programmation Fonctionnelle
Examen : 15 juin 2018
Durée : 2 heures

Prénom _____

Nom _____

Note

Num. Etu.

Tous les documents sont interdits. Les calculatrices et les téléphones portables doivent être rangés ainsi que tout autre matériel à l'exception d'un stylo et d'un effaceur. Un mémo est donné en fin de sujet

.

Exercice 1 Comptage des solutions d'un système d'inéquations (6 points)

On veut calculer en Scheme le nombre de solutions entières au système d'inéquations suivant [1]

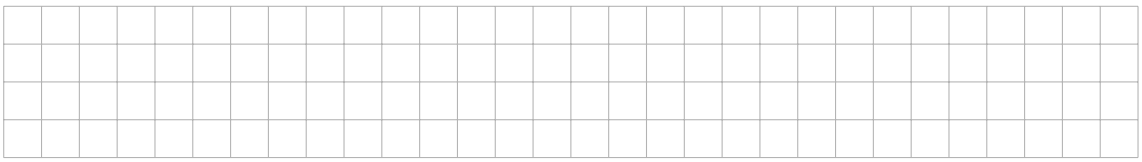
$$\begin{cases} n \geq 3 \\ m \geq 3 \\ \frac{180n(m-2)}{m} < 360 \end{cases}$$

Définissez une fonction `(s n m)` qui prend en argument deux entiers n et m et qui renvoie le nombre $\frac{180n(m-2)}{m}$. Par exemple, `(s 3 4)` renvoie 270.

1. Cela n'a aucun intérêt de le savoir pour répondre à l'exercice, mais chaque solution correspond à un type de polyhèdre régulier (tétrahèdres, cubes, icosaèdre, etc) ; on est donc en train de chercher à calculer combien de types de polyhèdres réguliers différents il existe.

.

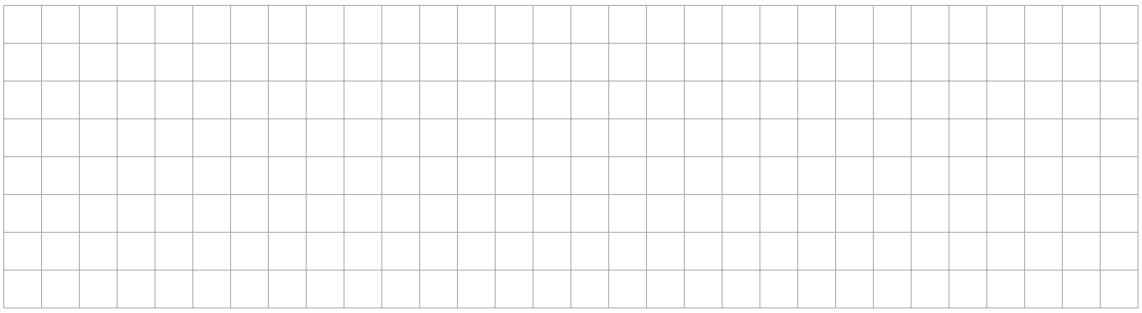
Définissez une fonction `(solution? n m)` qui prend en arguments deux entiers n et m et qui renvoie `#t` si le couple (n, m) est solution du système précédent. Par exemple `(solution? 3 4)` renvoie `#t`, mais `(solution? 4 4)` renvoie `#f` (car $s(4, 4) \geq 360$) de même que `(solution? 2 4)` (car $n < 3$) ou encore `(solution? 3 2)` (car $m < 3$).



À m fixé, la fonction s est croissante en la variable n, il existe donc un entier limite n_{limite} (qui dépend de m) tel que pour tout entier n,

$$(n, m) \text{ est solution} \qquad \text{si et seulement si} \qquad 3 \leq n < n_{\text{limite}}.$$

Définissez une fonction `(n-limite m)` qui prend en argument un entier m et qui renvoie le plus petit entier $n \geq 3$ tel que `(solution? n m)` renvoie `#f`. Par exemple, `(n-limite 4)` renvoie 4 car `(solution? 3 4)` renvoie `#t` et `(solution? 4 4)` renvoie `#f`.



Définissez la variable `nb-solutions` qui contient le nombre de solutions, autrement dit le nombre de couples d'entiers (n, m) tel que `(solution? n m)` renvoie `#t`. On admettra que le nombre de solutions du système correspond à la somme $\sum_{m=3}^{M} (n_{\text{limite}}(m) - 3)$ où M est le premier entier tel que $n_{\text{limite}}(M) = 3$. Vous pourrez vous inspirer du code ci-dessous.

```
(define nb-solutions
   (local [(define (iter m acc) ...)]
    (iter 3 0)))
```

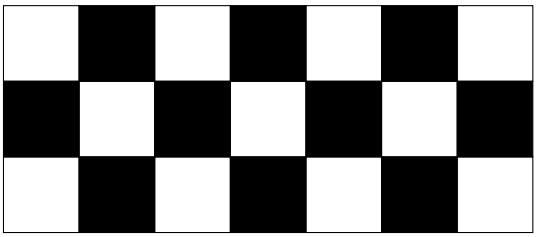
.

Exercice 2 Damier (4 points)

Définissez une fonction `(ligne-damier m solid?)` qui prend en argument un entier m et un booléen `solid?` et qui renvoie une image composée de m carrés 10×10 placés côte à côte, alternativement noir et blanc. Si `solid?` vaut `#t`, le premier carré est noir, sinon il est blanc. Par exemple, `(ligne-damier 6 #t)` et `(ligne-damier 5 #f)` renvoient les images respectives

 et 

Définissez une fonction `(damier n m)` qui renvoie l'image d'un damier de n lignes et m colonnes dont la case en haut à gauche est blanche. Par exemple, `(damier 3 7)` renvoie l'image ci-contre.



.

Exercice 3 Partitionnement de liste (5 points)

Définissez de manière itérative une fonction `(partage garder? L)` qui prend en argument un prédicat `garder?` et une liste `L` et qui renvoie le couple de deux listes `(LIN LOUT)` telles que `LIN` est la liste des éléments `x` de `L` tels que `(garder? x)` renvoie `#t`, et `LOUT` est son complémentaire. Par exemple,
`(partage even? '(1 2 3 4 2 1))` renvoie `'((2 4 2) (1 3 1))`. Vous pourrez vous inspirer du code ci-dessous.

```
(define (partage garder? L)
   (local
    [(define (iter L LIN LOUT) ...)]
    ... ))
```

Définissez cette même fonction en une ligne à l'aide de la fonction d'ordre supérieur `filter` dont on rappelle le principe sur un exemple ci-dessous.

```
;; exemple d'utilisation de filter
> (filter (lambda (x) (= 0 (modulo x 3))) '(0 1 2 3 4 5 6 0))
(0 3 6 0)
```

.

Exercice 4 Structure de données abstraite (3 points)

On souhaite disposer d'une structure de données abstraite `cercle` qui contient les information suivantes :

— coordonnées (x, y) du centre du cercle
— rayon r du cercle
— couleur `coul` du cercle

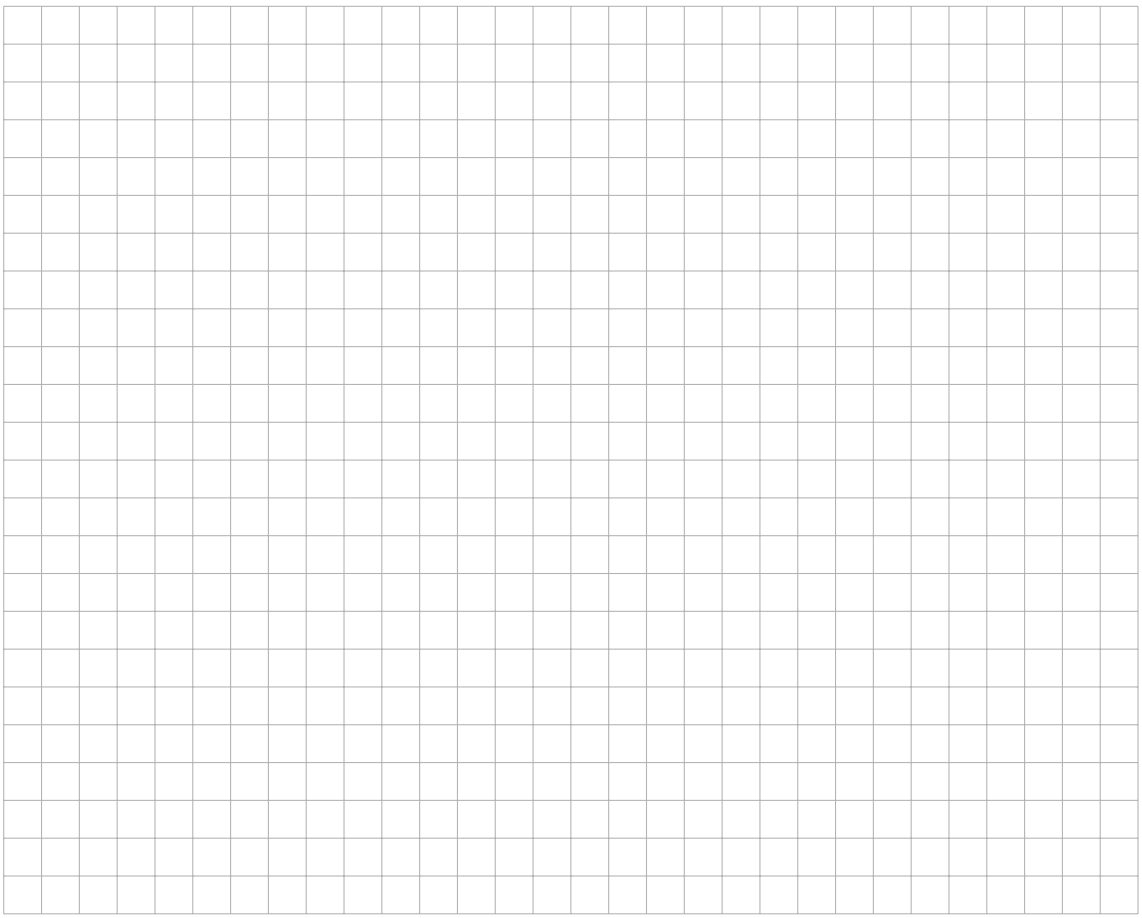
que l'on puisse consulter et dont on peut modifier le rayon (en renvoyant un nouveau cercle). On considère deux implémentations de cette structure de données : à l'aide de listes, et à l'aide de `struct`. Compléter les trous dans le code ci-dessous, en veillant à ce que les fonctions de même nom aient bien le même type dans chaque cas, comme indiqué en commentaire.

```
;; VERSION 1 : implementation par des listes
(define (nouveau-cercle centre r coul)
    ; liste de 2 nombres * nombre * couleur -> cercle
    (list centre r coul))
(define (position-du-cercle C)
    ; cercle -> liste de deux nombres
    (first C))
(define (abscisse-du-cercle C)
    ; cercle -> nombre
    ...)
(define (ordonnee-du-cercle C)
    ; cercle -> nombre
    ...)
(define (rayon-du-cercle C)
    ; cercle -> nombre
    ...)
(define (couleur-du-cercle C)
    ; cercle -> nombre
    ...)
(define (change-rayon-du-cercle C r)
    ; cercle * nombre -> cercle
    ...)
;; VERSION 2 : implementation par des struct
(define-struct pos (x y))
(define-struct cercle (pos-centre rayon couleur))
(define (nouveau-cercle centre r coul)
    ; liste de 2 nombres * nombre * couleur -> cercle
    (make-cercle (make-pos ...) r coul))
(define (position-du-cercle C)
    ; cercle -> liste de deux nombres
    (local [(define centre (cercle-pos-centre C)) ...] ...))
(define (abscisse-du-cercle C)
    ; cercle -> nombre
    ...)
(define (ordonnee-du-cercle C)
    ; cercle -> nombre
    ...)
(define (rayon-du-cercle C)
    ; cercle -> nombre
    ...)
(define (couleur-du-cercle C)
    ; cercle -> nombre
    ...)
(define (change-rayon-du-cercle C r)
    ; cercle * nombre -> cercle
    (make-cercle ... ))
```

.

Version 1 : implémentation par listes

Version 2 : implémentation par struct



.

Exercice 5 Un monde en expansion (2 points)

Complétez le code ci-dessous pour réaliser une animation correspondant à une image 100×100 de fond noir contenant un cercle jaune dont le centre est sur un point tiré au hasard. Le rayon du cercle est de 10 en début d'animation et augmente de 1 à chaque tic d'horloge, jusqu'à ce que le cercle déborde du cadre. Vous utiliserez la structure de données abstraite de l'exercice précédent via les fonctions qui y ont été définies, **en gardant le type cercle abstrait**.

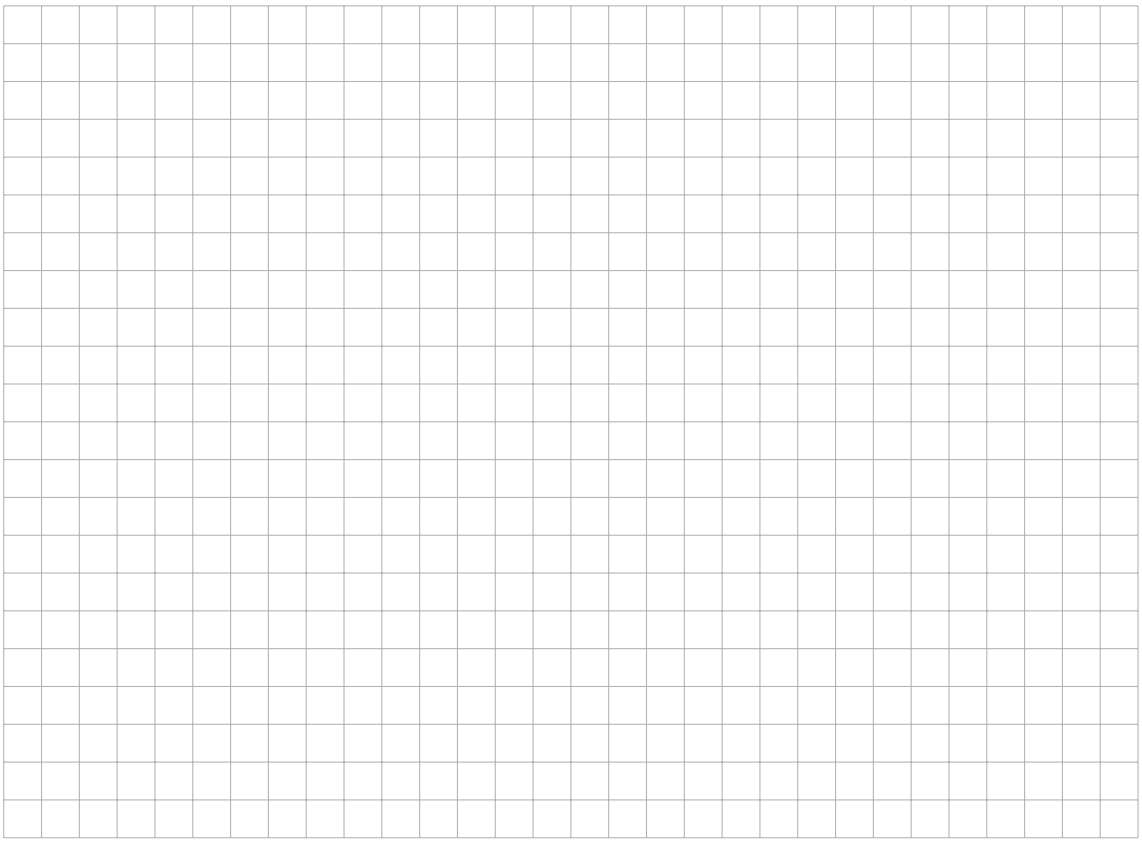
```
(define monde-initial ; le monde est un cercle
    (nouveau-cercle ...))

(define (dessine C) ; cercle -> image
    ...)

(define (suivant C) ; cercle -> cercle
    ...)

(define (final? C) ; cercle -> booleen
    ...)

(big-bang monde-initial
   (on-tick suivant)
   (on-draw dessine)
   (stop-when final?))
```



.

Memento

```
;; MATH

(quotient n m) ; quotient de la division de n par m
(modulo n m)   ; reste de la division de n par m
(random n)     ; renvoie un nombre au hasard
               ; compris entre 0 et n-1


;; IMAGES

empty-image           ; l'image vide
(rectangle l h style color) ;; un rectangle
(circle r style color) ;; un cercle
(underlay IMG1 IMG2) ; superposition centree
                     ; sans redimensionnement
(beside IMG1 IMG2)   ; juxtaposition horizontale
                     ; centree a mi-hauteur
(above IMG1 IMG2)    ; juxtaposition verticale
                     ; centree a mi-largeur
(place-image IMG1 x y IMG2)
                     ; IMG1 au-dessus de IMG2
                     ; avec son centre en x y
                     ; par rapport a l'angle
                     ; superieur gauche de IMG2,
                     ; voir exemple ci-dessous

;; LISTES : on suppose L = (L[1] ... L[n])

(cons x L)           ; = (x L[1] ... L[n])
(first L)            ; = L[1]
(rest L)             ; = (L[2] ... L[n])
(second L)           ; = L[2]
(third L)            ; = L[3]
(length L)           ; = n
```

```
(place-image (circle 100 'solid "black")
             60 60
             (rectangle 400 200 'solid "red"))
```