Programmation fonctionnelle

Contrôle continu 1

L1 – Université Nice Sophia Antipolis

26 février 2018

Durée : 1h30.

- Aucun document ni aucune machine ne sont autorisés.
- Les téléphones doivent être rangés.
- Un mémento de fonctions racket est donné à la fin du sujet
- Les réponses doivent être écrites lisiblement directement sur le sujet.
- L'indentation du code doit être similaire à celle automatique en Dr Racket, ceci afin de permettre d'interpréter favorablement une parenthèse manquante ou inutile. Un code mal indenté conduira a plus de sévérité sur le parenthèsage.
- Les deux premiers exercice sont indépendants du problème.
- Toute question peut être sautée, et une fonction `f` demandée à une question peut être utilisée pour définir une fonction `g` dans une question ultérieure même si la solution pour `f` n'a pas été trouvée.
- Il y a une question bonus "culture générale" à la fin du sujet.

Nom

Prénom

Numéro d'étudiant

Exercice 1 Produit scalaire (2 points)

Écrivez une fonction `(produit-scalaire x1 y1 x2 y2)` qui renvoie le produit scalaire des vecteurs (x_1, y_1) et (x_2, y_2), autrement dit le nombre $x_1x_2 + y_1y_2$.

```
(define (produit-scalaire x1 y1 x2 y2)




)
```

Exercice 2 Syracuse (4 points)

Écrivez une fonction `(syracuse n)` qui à n associe $\frac{n}{2}$ lorsque n est pair et $3n+1$ lorsque n est impair. Par exemple, `(syracuse 2)` renvoie 1 et `(syracuse 3)` renvoie 10

```
(define (syracuse n)




)
```

Le temps de vol d'un nombre n est le nombre fois que l'on peut lui appliquer la fonction de Syracuse avant d'atteindre 1. Par exemple, le temps de vol de 1 est 0, le temps de vol de 2 est 1, ou encore le temps de vol de 3 est 7 ($3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$). Écrivez une fonction qui renvoie le temps de vol d'un nombre n

```
(define (temps-de-vol n)









)
```

Exercice 3 : une animation qui casse des briques

Partie 1 : les briques de base (5 points)

On veut recouvrir une image `IMAGE-FOND` de largeur `LARGEUR` et de hauteur `HAUTEUR` à l'aide d'un mur comportant `R` rangées contenant alternativement `N` et `N+1` briques. Les rangées de `N` briques font exactement la largeur de l'image à recouvrir, et la hauteur du mur fait exactement la hauteur de l'image. On en déduit les dimensions d'une brique.

```
(define LARGEUR-BRIQUE                                    )
(define HAUTEUR-BRIQUE                                    )
```

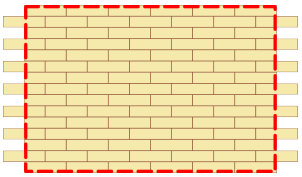
Une brique est représentée par deux rectangles superposés, le plus grand aux dimensions ci-dessus, de la couleur `COULEUR-CIMENT`, et le second de dimensions réduites de `EPAISSEUR-CIMENT` et de couleur `COULEUR-BRIQUE`.

```
(define IMAGE-BRIQUE



)
```

Le mur sera posé sur l'image de fond en tronquant les demi-briques qui dépassent sur les rangées longues ; seule la partie du mur dans le cadre rouge sera visible (voir dessin ci-contre).



On suppose que l'on a les images `IMAGE-FOND` et `IMAGE-MUR-ENTIER` (contenant les briques "qui dépassent"). Définissez l'image `IMAGE-MUR-TRONQUE` correspondant à la partie du mur inclue dans le cadre rouge.

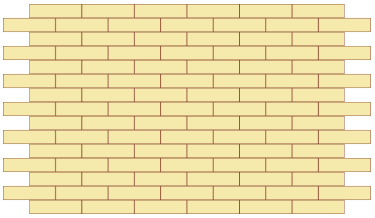
```
(define IMAGE-MUR-TRONQUE




)
```

Partie 2 : au pied du mur (6 points)

On cherche pour le moment seulement à produire l'image `IMAGE-MUR-ENTIER` d'un mur sans trou. On décompose le problème en deux sous-problèmes : construire l'image d'une rangée de k briques, puis construire l'image qui est la juxtaposition de k rangées de `N` ou `N+1` briques. Chaque problème se traite par récurrence.



La première fonction est celle qui renvoie l'image d'une rangée de k briques.

```
(define (rangee-briques k)
    ;; fonction de type entier -> image
    (if                                  ;; test d'arret
                                         ;; cas de base
                                         ;; recurrence
    ))
```

La deuxième fonction appelle la première pour renvoyer l'image d'un mur à k rangées. Elle prend un second argument booléen : s'il vaut `#true`, la première rangée du mur est une rangée courte, sinon elle est longue.

```
(define (mur-plein k premiere-courte?)
    ;; fonction de type entier * booleen -> image




)
```

Grâce à ces deux fonctions, il est aisé de définir l'image d'un mur avec toutes les briques.

```
(define IMAGE-MUR-PLEIN


)
```

Partie 3 : casse-brique (3 points)

Créez une animation où plusieurs images de murs aléatoirement troués se succèdent. Pour la n-ième image, la probabilité qu'à une position donnée il y ait un trou sera $\frac{n}{NR}$. L'animation termine quand $n = NR$.

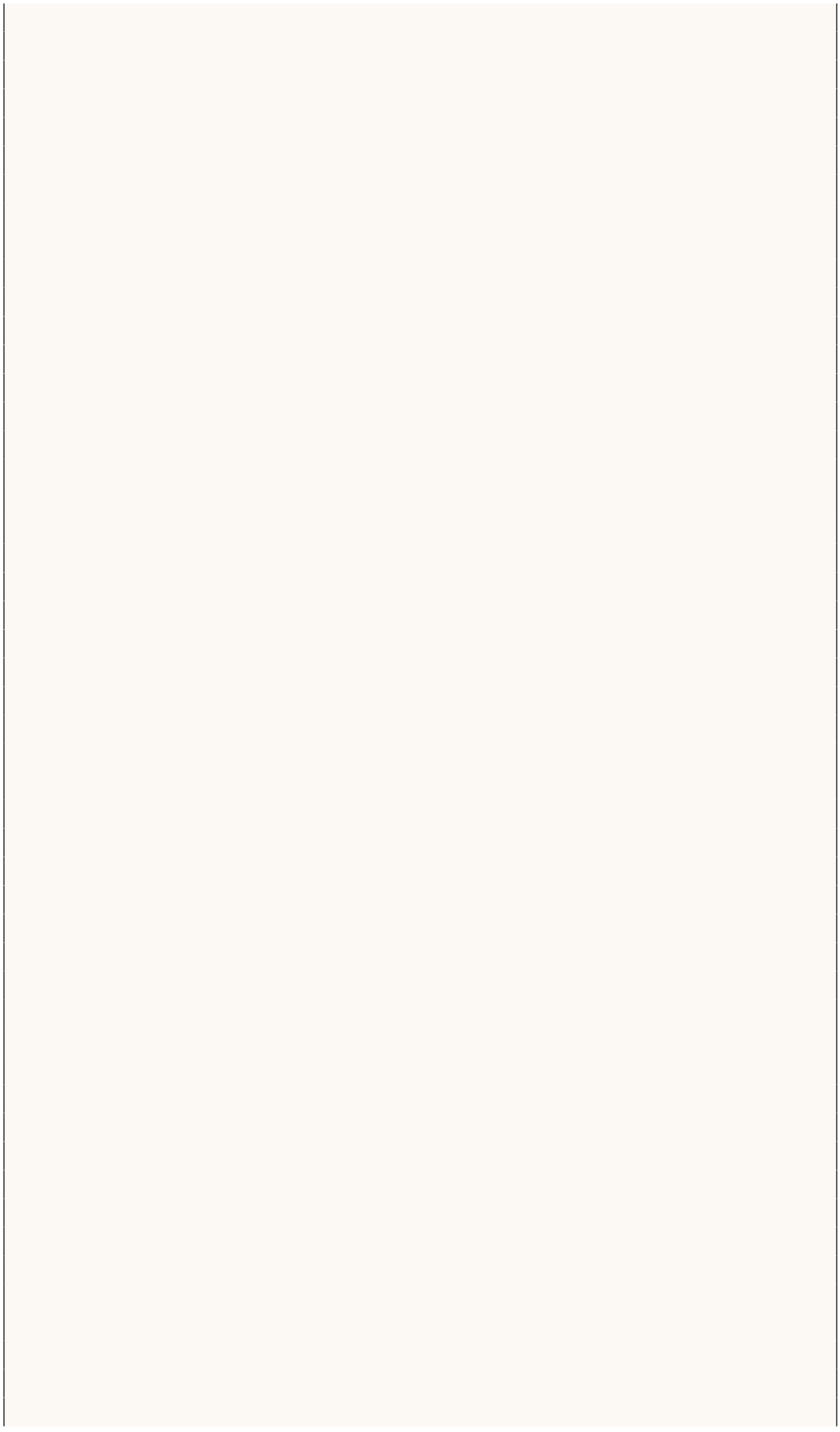
Indications

— Vous pourrez utiliser l'image d'une brique invisible

```
(define IMAGE-BRIQUE-INVISIBLE
    (rectangle LARGEUR-BRIQUE HAUTEUR-BRIQUE
               'solid TRANSPARENT))
```

— Vous pourrez introduire une fonction `image-brique` qui renvoie au hasard l'image d'une brique pleine ou celle d'une brique invisible, et vous inspirer du code de la partie 2.

```
;; Votre code ici!
```



```
;; ouf... c'est fini!
```

Exercice 3 Bonus culture générale (0,5 points)

Que s'est-t-il passé le 9 novembre 1989 ?

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

(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
```

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