

NOM: MARIN

PRÉNOM: Alicia

GROUPE: 1

Programmation Fonctionnelle

Contrôle 1.1

QUESTION 1: On suppose que l'on a défini x à 1 et y à 2. Écrire une expression Scheme utilisant la primitive `list` qui produise l'expression suivante:

(x 1 y 2 x+y 3)

(list 'x x 'y y 'x+y (+ x y))

~~(define (x y)~~

~~(print (list 'x x 'y y 'x+y (+ x y)))~~

0.5 BP

QUESTION 2: Que renvoie l'expression Scheme suivante (cons (list 1 2) 3)

(1 2) . 3

((1 2) . 3)

0.25

QUESTION 3: Que compte on lorsqu'on évalue la complexité d'une fonction en Scheme?

Le nombre d'appel de fonctions

de cas particuliers

0.5

? (pas au programme 21-22)

QUESTION 4: Écrire la fonction `contains-int?` qui renvoie `#t` si la liste qui lui est passée en paramètre contient au moins un entier et `#f` sinon. Ainsi par exemple:

(contains-int? '(a b c)) → #f
(contains-int? '(a 1 c)) → #t
(contains-int? '()) → #f

Pour tester si un objet est un entier, on utilisera le prédicat `integer?` qui renvoie `#t` si son paramètre est un entier et `#f` sinon:

(define (contains-int? lst)

(cond

(integer? 1) → #t
(integer? "foo") → #f
(integer? '(1 2 3)) → #f

((null? lst) #f)
((pair? lst) (or (integer? (car lst)) (contains-int? (cdr lst))))
(else #f)))

(define contains-int?

(lambda (l) (cond ((integer? l) #t)

; si c'est un entier

((null? l)

#f)

; si on a fini de parcourir la liste

((pair? l) (contains-int? (cdr l))
(else error "else"))))

Non

; si on n'a pas fini de parcourir la liste

1 BP

NOM: MARIN PRÉNOM: Alicia GROUPE: 1

Programmation Fonctionnelle

Contrôle 2.1

QUESTION 1: Écrire le corps de la fonction `f` dont deux utilisations sont données ci-dessous:

`(define f (lambda (a b) ...))`

`(f 1 2) → (a 1 b 2)`
`(f 200 -6) → (a 200 b -6)`

`(define f
 (lambda (a b)
 (list 'a a 'b b)))`

1
`(define f (lambda (a b)
 (list 'a a 'b b)))`

QUESTION 2: Réécrire le `let` suivant sous forme d'appel de fonction anonyme:

`(let ((foo 1)
 (bar 'hello))
 (list x foo bar))`

`((lambda (foo bar) (list x foo bar)) 1 'hello)`

0
`(define (lambda (x)
 (list x 1 'hello)))` *Wen*

QUESTION 3: Que renvoie l'expression suivante:

`((lambda (a . b) (cons a b)) 100 -5 "foo")` `(100 -5 "foo")`

0.5
`(100 -5 "foo")`

QUESTION 4: Écrire la fonction `remove` qui prend en paramètres une valeur `v` et une liste `lst` et qui renvoie une liste dans laquelle la première occurrence de `v` est supprimée:

`(remove 2 '(1 2 3)) → (1 3)`
`(remove 2 '(1 2 2 2)) → (1 2 2)`
`(remove 5 '(1 2 3)) → (1 2 3)`

`(define (remove v lst)
 (cond
 ((null? lst) ())
 ((pair? lst) (if (equal? v (car lst))
 (cdr lst)
 (cons (car lst) (remove v (cdr lst)))))
 (else lst))))`

`(define remove (v lst)`

`(cond ((null? lst) ()))`

`((pair? lst) (if (= v (car lst))`

`(cdr lst)`

`((car lst) (remove (cdr lst)))`

`(else error "else"))`

si v == 1^{er} élément liste

; on le supprime

; on le met de côté et on appelle la fct sur le reste

cons

2.75

Programmation Fonctionnelle

Contrôle 3.1

QUESTION 1: Quelle est la valeur de l'expression suivante:

```
(let ((x 1)
      (y 2))
  (let ((x y)
        (y x))
    (list x y)))
```

(2 1)

1

QUESTION 2: Soit la fonction suivante (attention ce n'est pas la fonction du cours):

```
(define (f n)
  (let ((c 1))
    (set! c (+ c 1))
    (+ n c)))
```

c = (2)

Quelle est la valeur de l'expression suivante:

```
(let* ((first (f 1))
       (second (f 1)))
  (list first second))
```

(3 3)

0

QUESTION 3: Soient les expressions suivantes:

```
(define foo 10)
(let ((X "hello"))
  (define foo X))
```

```
(define bar 10)
(let ((X "hello"))
  (set! bar X))
```

define crée une variable localement (= dans le champ du let) donc à gauche, ça n'affecte pas le foo de la ligne 1
set! en revanche modifie la valeur d'une variable existante donc à droite ça modifie bien

Expliquez pourquoi lorsqu'on évalue l'expression (list foo bar), on obtient la valeur (10 "hello").

Le (set! bar X) fait l'affectation de X="hello" à bar.
Or, le let définit la variable localement.
mais si on appelle la fonction de manière externe, le 10 est pris en compte.

1

QUESTION 4: Soit la liste lst définie à ((a 10) (b 20) (c 30)). Donner une expression qui, partant de lst, renvoie la valeur (10 20 30)

(map cadr lst)

(map cadr lst)

0.5

cadr

QUESTION 5: Dans le cours, on a défini

```
(define push #f)
(define pop #f)
(define print-stack #f)
```

```
(let ((S '()))
  (set! push (lambda (v) ...))
  (set! pop (lambda () ...))
  (set! print-stack (lambda () ...)))
```

push, pop et print-stack doivent

1. être dans le champ global

2. avoir accès à S qui ne doit pas y être

or, on ne peut accéder à une var que si elle est ≥ le champ actuel

donc les valeurs des 3 fct doivent être définies dans un champ contenant S,

mais elles doivent être stockées dans le champ global

d'où (define) pour les déclarer puis (set!) pour leur affecter une valeur accédant à S

Pourquoi les variables sont définies en dehors du let, et ensuite affectées à l'intérieur du let:

On définit l'implémentation des fonctions dans le let.
Les variables sont définies en dehors du let afin de les déclarer.

0.25

oui mais pourquoi ces affectations ?