

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)

~~(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

0.25

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

Le nombre d'appel de fonctions

0.5

de cas particuliers

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:

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

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))
```

0

```
(define (lambda (x)
  (list x 1 'hello)))
```

Wcr

QUESTION 3: Que renvoie l'expression suivante:

```
((lambda (a . b) (cons a b)) 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)
```

1.25

```
(define remove (v lst)
```

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

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

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

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

```
    (else error "else"))
```

cons

si v == 1^{er} élément liste

; on le supprime

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

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)))
```

1

(2 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))
```

0

(2 3)

(3 3)

QUESTION 3: Soient les expressions suivantes:

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

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

1

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.

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)

0.5

(map cdr lst)

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 () ...)))
```

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

0.25

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

oui mais pourquoi ces affectations ?