Macros

Exercice 1 :
Utiliser macroexpand-1 et macroexpand pour observer l'expansion de diverses macros : first, second, third, my-cond.

1. first
```
(defmacro first (l)
  `(car ,l))

(macroexpand '(first (f (g x y))))
(macroexpand-1 '(first (f (g x y))))
```
2. second
```
(defmacro second (l)
  `(first (cdr ,l)))

(macroexpand-1 '(second x))
(macroexpand '(second x))
```
3. third
```
(defmacro third (l)
  `(caddr ,l))
```
4. my-cond
```
(defmacro my-cond (&body body)
  `(cond
    ,@body
    (t 'error)))

macroexpand-1 '(my-cond
 ((zerop n) 'zero)
 ((= 1 n) 'un)
 ((= 2 n) 'deux)))

(macroexpand '(my-cond
       ((zerop n) 'zero)
       ((= 1 n) 'un)
       ((= 2 n) 'deux)))
```

Exercice 2 :
Définir une macro *double* qui double son argument (c'est une affectation) :

```
(let ((x 1))
  (double x)
  x)
2
```

La tester sur des exemples

Exercice 3 :

1. Définir une fonction $let-to-lambda$ qui traduit une expression let en l'application d'une $lambda-expression$. Le schéma général est le suivant :

```
(let ((p1 a1) ... (pK aK))
  forme1
  ...
  formeN)
```
doit donner
```
((lambda (p1 ...pK)
  forme1
  ...
  formeN)
  a1 ... aK)
```
Vous pouvez découper le travail en plusieurs fonctions auxiliaires.

2. Ecrire une macro "$mon-let$" qui donne une lambda expression en expansion :
```
(defmacro mon-let ???)
  ???)

(macroexpand '(mon-let ((x 45) (y (+ 3 4))) (cons x y)))
; --> ((LAMBDA (X Y) (CONS X Y)) 45 (+ 3 4))
```
Penser à utiliser la fonction précédente

Exercice 4 :

1. Ecrire une macro "mcons" telle que :
```
(macroexpand-1 '(mcons e1 ... en)) -> (cons e1 (mcons e2 ... en))
(macroexpand-1 '(mcons)) -> ()
```
Evaluer les formes suivantes :
```
(macroexpand-1 '(mcons 1 2 3 4 5))
(macroexpand '(mcons 1 2 3 4 5))
(mcons 1 2 3 4 5)
```
2. Ecrire une macro "mmcons" telle que :
```
(macroexpand-1 '(mmcons)) -> ()
(macroexpand-1 '(mmcons e1 ... en)) -> (cons e1 (cons ... (cons en
  ())))
```
Utiliser une fonction auxiliaire pour engendrer l'expansion de la macro :
```
(defmacro mmcons (&body forms)
  (consify forms))
(defun consify (forms)
  '???)
```
Evaluer les formes suivantes :
```
(macroexpand-1 '(mmcons 1 2 3 4 5))
(macroexpand '(mmcons 1 2 3 4 5))
(mmcons 1 2 3 4 5)
```