Evaluation - variables - portée

Exercice 1 : Variables lexicales et spéciales

- Formes `let` : déterminer les valeurs des formes suivantes :
  -
```
(let ((x 5)
       (y (sqrt 5))
       (z (* 4 (+ 3 (sqrt 2) (sqrt 4)))))
   (+ x y (* 5 z)))
```
  -
```
(+ (let ((a 12)) (+ a a))
     (let ((b 30)) (+ b (let ((b 20)) (+ b b)))))
```
- Considérer la session suivante ; déterminer les valeurs de `q1` et `q2` :
```
(defparameter x 12)
(defparameter y 5)
(defparameter q1 (let ((x y)
                       (y x))
                   (- x y)))
(defparameter q2 (let* ((x y)
                        (y x))
                   (- x y)))
```

Exercice 2 : Portée lexicale et portée dynamique

- Ecrire la session suivante :

```
* (defun print-base (n base)
    (let (( *print-base* base))
       (print n)))
```
- Essayer la session :
```
* (print-base 6 2)
110
6
* 6
6
```
- Soit la session :
```
* (defparameter *n-fois* 4)
  (defun g (*n-fois* x)
    (if (zerop *n-fois*)
     0
    (f x)))
  (defun f (x)
    (* *n-fois* x))

* (g 2 2 )
```
- Ecrire une expression let déclarant la variable $n-fois$ (lexicale) et contenant les déclarations des fonctions g et f et utilisant $n-fois$ au lieu de $*n-fois*$. Constater la différence de portée de la variable $n-fois$

Exercice 3 : Evaluation des expressions booléennes

L'évaluation de fonctions en lisp (à l'exception de l'évaluation des formes spéciales) se fait de la manière suivante : les arguments sont d'abord tous évalués dans un ordre quelconque puis la fonction leur est

appliquée. Les expressions conditionnelles sont des formes spéciales et sont évaluées différemment. Les manipulations suivantes ont pour but de vous faire bien comprendre comment se fait leur évaluation.

Pour chacune des expressions conditionnelles indiquées ci-dessous, imaginer quelle sera la valeur de a après son évaluation, vérifier en évaluant cette expression au top-level puis en déduire les processus d'évaluation de (*and* arg1 ... argn) et (*or* arg1 ... argn).

```
* (defparameter a 0)
* (and (setf a 20) (= a 20)(setf a  40)(setf a 50))
* (and (setf a 20) (= a 30)(setf a  40)(setf a 50))
* (or (setf a 20) (= a 20)(setf a  40)(setf a 50))
* (or (= a 30) (= a 50)(setf a  40)(setf a 50))
```

Exercice 4 : Formes spéciales

- Ecrire une fonction `new-if` à trois arguments `predicat clause-alors clause-sinon` et faisant appel à la forme `if` avec les mêmes arguments pour réaliser un test.
  - Essayer la session suivante (la fonction `print` effectue un affichage à l'écran) :
    ```
    * (defparameter a 0)
    * (new-if (zerop a) t nil)
    * (new-if (zerop a)
              (print "a est nul")
              (print "a est non nul"))
    ```
    Que constatez-vous ? Expliquez le mécanisme conduisant à ce résultat.
  - Ecrire la fonction `new-factorielle` en utilisant la fonction factorielle dans laquelle vous remplacez l'appel à `if` par un appel à `new-if`. Evaluez l'appel suivant :
    ```
    * (new-factorielle 3)
    ```
    Que constatez-vous ? Expliquez le mécanisme conduisant à cette erreur.

Exercice 5 : Utilisation de *labels*

- En utilisant deux fonctions, écrire l'expression conduisant au calcul de

$$(1+\sqrt{2})(1+\sqrt{2}+\sqrt{3})(1+\sqrt{2}+\sqrt{3}+\sqrt{4})$$

- Faites la même chose que précédemment en définissant une seule fonction au top-level et en utilisant une fonction annexe définie localement.
- Ecrire une fonction `nombre-div` qui calcule le nombre de diviseurs de son argument, ainsi que `somme-div` qui calcule la somme des diviseurs de son argument.