Rapport de TP

Projet n°1

Marie Julien

Johan Medioni

utc
Université de Technologie
Compiègne

Julien Marie Gl01
Medioni Johan TC03

Dans ce TP on gère une base de données sous forme d'une liste de personnes ; chaque personne est une liste contenant son nom, prénom, ville, âge, nombre de livres possédés. Le but du TP consistait à écrire une série de fonctions pour manipuler cette base de données.

Une première partie consistait à écrire des fonctions de service, afin de pouvoir les réutiliser éventuellement par la suite dans des fonctions plus complexes.

Table des matières

Fonctions de services

(defun nom (personne) (car personne)) (defun prenom (personne) (second personne)) (defun ville (personne) (third personne)) (defun age (personne) (fourth personne)) (defun livres (personne) (fifth personne))	On définit les fonctions de service qui renvoient le nom, le prénom, la ville, l'âge ou le nombre de livres d'une personne donnée, à l'aide des fonctions car (équivalente à first) et second, third, etc.

Fonctions

(defun F1 (base) (dolist (i base) (print i)))	La fonction F1 affiche toutes les personnes. On parcourt la liste grâce à dolist et on affiche élément par élément. On pourrait afficher seulement le nom et le prénom, mais on a choisi d'afficher toutes les informations. (Il aurait fallu écrire ((nom i) (prenom i)) au lieu de i).
(defun F2 (base) (dolist (i base) (if (eq (first i) 'Perrot) (print i))))	F2 affiche toutes les personnes dont le nom est Perrot. On utilise dolist, et pour chaque élément on vérifie si le nom est Perrot ; si oui on affiche.
(defun F3 (base) (dolist (i base) (if (eq (third i) 'Lyon) (print i))))	De la même façon, F3 affiche les personnes qui habitent à Lyon.
(defun F4 (base nom) (dolist (i base) (if (eq (first i) nom) (print i))))	F4 prend comme arguments la base de personnes ainsi qu'un nom X, et renvoie les personnes qui s'appellent X.
(defun F5 (base X) (dolist (i base) (if (eq X (age i)) (return i))))	F5 renvoie la première personne dont l'âge est X, grâce à la fonction return : dès qu'on trouve une personne qui correspond, return permet de s'arrêter de boucler.
(defun F6 (base) (dolist (i base) (if (> 100 (livres i)) (return i))))	De même F6 cherche la première personne possédant plus de 100 livres.

(defun F7 (base) (dolist (i base) (if (and (eq 'Lyon (ville i)) (< 50 (age i))) (return i))))	F7 renvoie la premiere personne qui habite à Lyon et a plus de 50 ans. Le (and *cond1 cond2*) permet de combiner ces deux conditions.
(defun F8 (base) (let ((somme 0)) (dolist (i base) (setf somme (+ somme (fifth i)))) somme))	F8 doit renvoyer la somme de tous les livres possédés par tout le monde. On utilise let pour créer une variable somme locale, qu'on renvoie à la fin.
(defun F9 (base) (let ((somme 0)) (dolist (i base) (when (eq 'Lyon (third i))(setf somme (+ somme (fifth i))))) somme))	F9 fonctionne de la même façon que F8. On rajoute simplement une condition dans un (when *cond instr*) pour que le (setf) n'ait lieu que si l'habitant est de Lyon.
(defun F10 (base nom) (let ((somme 0)) (dolist (i base) (if (eq nom (first i)) (setf somme (+ somme (fifth i))))) somme))	F10 retourne la somme du nombre de livres possédés par les personnes dont le nom est X. On utilise encore let pour créer une variable somme, ainsi que dolist : pour chaque élément on vérifie si le nom correspond, et si oui on additionne au total.
(defun F11 (base) (let ((nombre 0.0) (total 0.0)) (dolist (i base) (setf total (+ total (fourth i))) (setf nombre (+ nombre 1))) (/ total nombre))	F11 calcule la moyenne des âges de toutes les personnes. Avec let, on crée deux variables nombre et total qu'on initialise à 0.0 (afin d'avoir des variables décimales). Avec dolist, pour chaque élément, on incrémente la variable nombre (nombre de personnes dans la liste) et on ajoute la valeur de l'âge au total. A la fin, on renvoie le résultat de la division de total par nombre. On part du

)	principe que le nombre de personnes est non nul.
(defun F12 (base nom) (let ((nombre 0.0) (total 0.0)) (dolist (i base) (when (eq nom (first i)) (setf total (+ total (fourth i))) (setf nombre (+ nombre 1)))) (if (= 0 nombre) NIL (/ total nombre))))	Même principe que pour F11, mais on utilise le (when) pour imposer une condition au calcul de la moyenne. On vérifie ainsi qu'on ne calcule que la moyenne d'âge des personnes dont le nom est passé en argument.
(defun F13 (base) (setf copie13 (list NIL)) (dolist (i base) (nconc copie13 (list i)))) (setf copie13 (rest copie13)))	F13 réalise une simple copie de la base. Attention elle créée une nouvelle liste, ou affecte à une liste copie13 qui existerait avant l'exécution de la fonction la base passée en argument. On parcourt la base avec dolist, on rajoute chaque élément à la fin de copie13 avec nconc. Ensuite on enlève le premier élément de la liste qui est NIL.
(defun F14 (base) (setf copie14 (list NIL)) (dolist (i base) (nconc copie14 (list (remove (fifth i) i)))) (setf copie14 (rest copie14)))	F14 est une variante de F13. On utilise le remove pour enlever le nombre de livres que chaque personne possède.
(defun F15 (base) (setf copie15 (list NIL)) (dolist (i base) (if (eq 'Lyon (third i)) (nconc copie15 (list i)))) (setf copie15 (rest copie15)))	Même fonctionnement que F13, mais on utilise un if pour ne réaliser la copie que si la condition « habite à Lyon » est réalisée.
(defun F16 (base)	Même fonction que F15, mais en apposant

(setf copie16 (list NIL)) (dolist (i base) (if (not (eq 'Nice (third i))) (nconc copie16 (list i)))) (setf copie16 (rest copie16)))	une négation à la condition => « n'habite pas à Lyon ». On aurait aussi pu utiliser (unless (eq 'Nice (third i))).
(defun F18 (base nom ville) (dolist (i base) (when (and (eq (first i) nom) (eq (third i) ville) (print i)))))	F18 affiche les personnes qui habite dans la ville passée en argument et qui porte le nom passé en argument.

Note : F17 est la même fonction que F4.

Conclusion

Commentaire : print ne fait pas un retour au sens strict, elle ne fait qu'afficher. Une alternative pour avoir en retour une liste de gens est de créer cette liste et de la retourner. Il faut donc utiliser une variable interne (un peu comme ce qui a été fait avec les sommes et les moyennes, où là, il y a eu un vrai retour de valeurs obtenues). Mieux vaut-il utiliser print ou un vrai retour ? Cela dépend de l'utilisation qui doit être faite de la fonction. Si son seul intérêt est d'afficher la liste retournée, autant utiliser directement print. En revanche si on veut travailler sur cette liste, il vaut mieux la créer et la retourner, (f_i *Base arg*) serait alors une liste de sous listes.

Notions importantes utilisées :

- Variables internes et globales : attention aux variables globales. Elles peuvent néanmoins être utiles pour les fonction de copie : on peut générer la copie.
- Quand return est rencontré, on sort de la boucle !
- Boucles conditionnelles : if, when, unless.
- Boucles itératives : dolist.