# Travaux Pratiques de caml n°8

**Licence d'informatique**

*—Caml—*

---

## Filtrage et Poker

Dans ce TP, nous allons modifier la structure du TP précédent pour manipuler un jeu de cartes : Une carte est représentée par un couple (valeur, couleur) où la couleur est définie par le type

```
type couleur = Trefle | Carreau | Coeur | Pique ;;
```

et la valeur par

```
type valeur = Petite of int | Valet | Dame | Roi | As ;;
```

Exemple : 10 de pique = `(Petite 10, Pique)`.

On considèrera l'ordre des cartes où la plus petite carte est le 2, la plus grande l'as, l'ordre des couleurs est Trèfle < Carreau < Coeur < Pique.

---

### ▸ Exercice 1. *Mise à jour des fonctions du TP précédent*

*Adapter depuis le TP précédent les fonctions de comparaison de cartes, et de génération de la liste des cartes. Remarquer que toutes les autres fonctions marchent encore.*

### ▸ Exercice 2. *Génération de mains*

*On appelle* **main** *toute liste de 5 cartes. On veut pouvoir obtenir des mains à partir d'un jeu complet mélangé (voir TP précédent). Pour cela, il faut pouvoir renvoyer les cartes retenues ainsi que le jeu privé de ces cartes. On utilisera pour cela des fonctions renvoyant des couples ou des listes.*

1. *Programmer la fonction* `tirer_une_carte` *qui prend en paramètre une liste de cartes et qui renvoie le couple formé par la première carte et la liste privée de cette carte :*

```
# tirer_une_carte [ (Petite 7,Pique) ; (Valet,Trefle) ; (Petite 8,Coeur);
  (Valet,Coeur) ; (Petite 7,Trefle) ];;
- : (valeur * couleur) * (valeur * couleur) list = (Petite 7, Pique),
                         [Valet, Trefle; Petite 8, Coeur; Valet, Coeur;
                         Petite 7, Trefle]
```

2. *En utilisant* `tirer_une_carte`*, écrire la fonction* `tirer_n_cartes` *qui prend un nombre* $n$ *de cartes à tirer et une liste de cartes, et qui renvoie le couple formé par les* $n$ *cartes et la liste privée de ces cartes :*

```
# tirer_n_cartes 2 [ (Petite 7,Pique) ; (Valet,Trefle) ; (Petite 8,Coeur);
  (Valet,Coeur) ; (Petite 7,Trefle) ];;
- : (valeur * couleur) list * (valeur * couleur) list = [Petite 7, Pique;
                          Valet, Trefle], [Petite 8, Coeur; Valet, Coeur;
                          Petite 7, Trefle]
```

3. *En déduire une fonction* `tirer_n_mains` *qui prend un nombre* $n$ *et une liste de cartes, et qui renvoie la liste constituée de la liste privée des* $5 * n$ *cartes tirées et des* $n$ *mains :*

```
# tirer_n_mains 1 [ (Petite 9,Carreau) ; (Petite 7,Carreau) ;
  (Petite 8,Carreau) ; (Petite 6,Carreau) ; (Petite 10,Carreau);
  (Petite 9,Carreau) ; (Roi,Coeur) ];;

- : (valeur * couleur) list list * (valeur * couleur) list =
([[(Petite 9, Carreau); (Petite 7, Carreau); (Petite 8, Carreau);
   (Petite 6, Carreau); (Petite 10, Carreau)]],
 [(Petite 9, Carreau); (Roi, Coeur)])
```

### ▸ Exercice 3. *Valeur d'une main*

*On souhaite maintenant pouvoir évaluer la valeur d'une main. Les valeurs possibles que l'on utilisera sont, dans l'ordre, la quinte flush (5 cartes qui se suivent de même couleur), le carré (4 cartes identiques), le full (un brelan plus une paire), la couleur (5 cartes de même couleur), la quinte (5 cartes qui se suivent), le brelan (3 cartes identiques), la double-paire, la paire, et la main qui ne vaut rien.*

*Pour cela, on se dote d'un type* `jeu` *décrivant ces valeurs :*

```
type jeu=
  Quinte_flush of valeur (*carte la plus élevée*)
| Carre of valeur        (*carte du carré*)
| Full of valeur*valeur  (*carte du brelan et de la paire*)
| Couleur of valeur      (*carte la plus élevée*)
| Quinte of valeur       (*carte la plus élevée*)
| Brelan of valeur       (*carte du brelan*)
| Double_paire of valeur*valeur
                         (*carte de la plus grande paire et de la plus petite*)
| Paire of valeur        (*carte de la paire*)
| Rien ;;
```

*En vous aidant de fonctions de tri par valeur et par couleur, écrire les fonctions suivantes (chaque fonction prend une main en paramètre) :*

1. `quinte_flush` *: renvoie* `Quinte_flush(n)` *où* `n` *est la valeur de la plus haute carte si on a une quinte flush et* `Rien` *sinon.*

```
# quinte_flush [ (Petite 7,Carreau) ; (Petite 8,Carreau) ;
  (Petite 6,Carreau) ; (Petite 10,Carreau) ; (Petite 9,Carreau) ];;
- : jeu = Quinte_flush Petite 10
```

2. `carre` *: renvoie* `Carre(n)` *si on a un carré et* `Rien` *sinon.*

```
# carre [ (Petite 7,Pique) ; (Valet,Trefle) ; (Petite 7,Coeur) ;
  (Petite 7,Coeur) ; (Petite 7,Trefle) ];;
- : jeu = Carre Petite 7
```

3. `full` *: renvoie* `Full(n,m)` *si on a un full et* `Rien` *sinon.*

```
# full [ (Petite 7,Pique) ; (Valet,Trefle) ; (Petite 7,Coeur) ;
  (Valet,Carreau) ; (Petite 7,Trefle) ];;
- : jeu = Full (Petite 7, Valet)
```

4. `couleur` *: renvoie* `Couleur(n)` *si on a une couleur et* `Rien` *sinon.*

```
# couleur [ (Petite 7,Pique) ; (Valet,Pique) ; (Petite 7,Pique) ;
  (As,Pique) ; (Petite 3,Pique) ];;
- : jeu = Couleur (As)
```

5. `quinte` *: renvoie* `Quinte(n)` *si on a une quinte et* `Rien` *sinon.*

```
# quinte [ (Petite 7,Pique) ; (Petite 8,Carreau) ; (Petite 6,Coeur) ;
  (Petite 10,Trefle) ; (Petite 9,Pique) ];;
- : jeu = Quinte Petite 10
```

6. `brelan` *: renvoie* `Brelan(n)` *si on a un brelan et* `Rien` *sinon.*

```
# brelan [ (Dame,Pique) ; (Dame,Trefle) ; (Petite 4,Coeur) ;
  (Dame,Coeur) ; (Petite 8,Carreau) ];;
- : jeu = Brelan Dame
```

7. `double_paire` *: renvoie* `Double_paire(n,m)` *si on a une double-paire et* `Rien` *sinon.*

```
# double_paire [ (Petite 8,Trefle) ; (Valet,Trefle) ; (Petite 5,Pique) ;
  (Valet,Coeur) ; (Petite 8,Carreau) ];;
- : jeu = Double_paire (Valet, Petite 8)
```

8. `paire` *: renvoie* `Paire(n)` *si on a une paire et* `Rien` *sinon.*

```
# paire [ (Petite 7,Pique) ; (Petite 8,Carreau) ; (Petite 9,Coeur) ;
  (Petite 10,Trefle) ; (Petite 9,Pique) ];;
- : jeu = Paire Petite 9
```

9. *Naturellement, une même main peut satisfaire plusieurs valeurs (un brelan contient une paire, par exemple). Programmer une fonction* `meilleur_jeu` *qui renvoie la plus forte valeur pour une main donnée.*

```
# meilleur_jeu [ (Petite 8,Pique) ; (Petite 8,Coeur) ; (Petite 9,Coeur) ;
  (Petite 9,Trefle) ; (Petite 9,Pique) ];;
- : jeu = Full (Petite 9, Petite 8)
# meilleur_jeu [ (As,Pique) ; (Dame,Coeur) ; (Petite 2,Carreau) ;
  (Petite 7,Pique) ; (Petite 10,Pique) ];;
- : jeu = Rien
```