L1 MISPIC
Programmation fonctionnelle
TP 1

1 S'organiser en TP

La première chose à faire avant de commencer ce TP est d'organiser correctement votre `HOME`. Pour cela, lancez un terminal et créez l'arborescence suivante (en utilisant les commandes `ls`, `cd` et `mkdir`) :

```
L1/S2/OCaml/TP/TP1
```

Dans ce répertoire `TP1`, ouvrez un nouveau fichier `tp1.ml` à l'aide d'*emacs* :

```
emacs tp1.ml &
```

Puis lancez l'interpréteur OCaml dans votre terminal :

```
ledit ocaml
```

Ceci est une version améliorée de l'interpréteur `ocaml` classique, qui propose un historique (accessible avec les flèches haut et bas du clavier).

Dans le fichier `tp1.ml`, vous taperez toutes les instructions et fonctions demandées lors de ce TP. Vous testerez votre code dans l'interpréteur OCaml en utilisant la commande `#use "tp1.ml";;` Ceci permet de charger tout le contenu de votre fichier dans l'interpréteur, sans avoir à copier-coller chaque fonction (attention à bien mettre en commentaires tout ce qui ne serait pas du code dans votre fichier `tp1.ml`). Si l'interpréteur génère des erreurs, vous devez corriger votre code dans *emacs* jusqu'à ce qu'il n'y en ait plus. Ensuite, il faudra encore vous assurer que vos fonctions calculent bien ce que vous souhaitez : testez-les en les appelant avec différentes valeurs de paramètres pour être sûr que ce qu'elles vous renvoient est correct. Vous vous assurerez d'avoir les versions correctes de vos programmes dans votre fichier `tp1.ml`. Ceci vous permettra de sauvegarder votre travail.

Vous êtes maintenant prêts pour travailler. Note : si pour une raison ou une autre vous souhaitez, à un moment donné, quitter l'interpréteur OCaml et reprendre la main sur le terminal, il suffit de faire `Ctrl + d`.

2 Définitions, types et valeurs

Dans cet exercice, il est important de comprendre ce que renvoie l'interpréteur OCaml lorsque vous exécutez les instructions, et pourquoi (valeurs, types).

1. Définir une variable globale `n` comme étant la somme des 5 premiers entiers non nuls.
2. Définir une variable globale `m` comme le cube du nom précédent.
3. Redéfinir `n` comme le produit des 5 premiers entiers non nuls. Que se passe-t-il si on tape `m;;` ? Est-ce normal ?
4. Que répond OCaml si on tape maintenant `n - n;;` ? Et `x - x;;` ?
5. Définir une variable locale `n` égale à 30 pour calculer le cube de 30. Que vaut `n` avant, pendant et après ce calcul ?
6. Proposer une expression permettant de calculer $b^2 - 4ac$ en sachant que $a = 1 + xy$, $b = x - y$, $c = x + 2y$, $x = 156$ et $y = 123$.
7. Proposer une expression permettant de calculer $2\pi r$ en sachant que $\pi = 3.14$, $r = 2 + 6 \times 1.5$.
8. Définir en une instruction deux variables globales `q` et `r` comme le quotient et le reste de 5/2.
9. Calculer la valeur réelle de 5/2.
10. Que répond OCaml si l'on tape `9;;`, puis `'9';;`, puis `"9";;`, puis `9.0;;` ?
11. Définir une variable globale `pi_s` comme la chaîne de caractères `"-3.14"`. Définir ensuite une variable globale `pi_f` comme le réel correspondant à la chaîne de caractères `pi_s`. Vous trouverez des exemples d'utilisation des fonctions de conversion de types à la fin de cet exercice.
12. Définir ensuite une variable globale `pi_i` comme l'entier correspondant à la troncature de la chaîne de caractères `pi_s`. Que se passe-t-il si on essaye d'utiliser la fonction de conversion `int_of_string` ? Pourquoi ? Comment faire alors pour obtenir l'entier correspondant à la chaîne `pi_s` ?
13. Définir une variable globale `pi_s2` comme la concaténation de `pi_s` et de la chaîne `"159"`. Définir les variables globales `pi_f2` et `pi_i2` de la même manière que vous avez définit `pi_i` et `pi_f`, mais avec `pi_s2`. Comparez les valeurs de `pi_f` et `pi_f2`, puis `pi_i` et `pi_i2`.
14. Définir la chaîne de caractères globale `c` comme `"La pie voleuse"`. Définir la variable globale `l` comme la longueur de `c`.
15. Définir les deux variables globales `debut` et `fin` comme le premier et le dernier caractère de `c`. Voir l'exemple en fin d'exercice.

Fonctions de conversion de types :

```
# int_of_float 3.14 ;;
- : int = 3
# int_of_float -4.67 ;;
This expression has type float -> int but is here used with type int
# int_of_float (-4.67) ;;
- : int = -4
# float_of_int (-3) ;;
- : float = -3.0
# string_of_float 3.14 ;;
- : string = "3.14"
# string_of_int (int_of_float 4.77) ;;
```

```
- : string = "4"
# int_of_string "45" ;;
- : int = 45
# float_of_string "coucou" ;;
Exception: Failure "float_of_string".
# float_of_string "    - 4.67 " ;;
Exception: Failure "float_of_string".
# float_of_string "-4.67" ;;
- : float = -4.67
```

Chaînes de caractères :

```
# let chaine = "Vendredi soir";;
val chaine : string = "Vendredi soir"
# String.length "lundi";;
- : int = 5
# chaine.[2];;
- : char = 'n'
```

3 Correction de programmes

Dans cet exercice, vous devez corriger les programmes OCaml contenus dans le fichier `corriger.ml` disponible sur Claroline. Enregistrer ce fichier dans votre dossier `TP1`, et ouvrez-le avec *emacs*.

3.1 Géométrie

Commencer par `la_fonction_de_mathilde` : qu'a-t-elle voulu écrire ? Corriger les erreurs, donner des noms explicites aux variables, indenter correctement. Réécrire cette fonction le plus simplement possible. Tester cette fonction sur un exemple.

Faire de même avec `la_fonction_de_dimitri`.

3.2 La suite

Grâce à votre aide, Mathilde et Dimitri se sentent pousser des ailes et décident de programmer, ensemble, deux autres fonctions. Il semble qu'ils n'aient pas tout compris puisque voici le résultat : la fonction `sourire` et la fonction `musique`. Vous vous demandez ce qu'ils ont bien pu vouloir écrire, mais décidez de les corriger quand même...

4 Fonction de comparaison

Définir une fonction `comparaison : int -> float -> char -> int -> string -> bool` qui renvoie vrai

- si la somme des deux premiers paramètres est supérieure à 10.0
  ou
- si la longueur de la chaîne moins le quatrième paramètre donne un résultat différent de 0, et que le caractère est plus petit ou égal au caractère 'd'.

5 Ou exclusif

Vous connaissez les 3 fonctions prédéfinies sur les booléens suivantes (qui correspondent à des opérateurs logiques) :

— `&&` (et) qui renvoie vrai si ses deux paramètres sont vrais, et faux sinon ;
— `||` (ou) qui renvoie vrai si au moins un des ses paramètres est vrai, et faux sinon ;
— `not` (non) qui renvoie la négation de son paramètre.

Définir une fonction `ou_exclusif : bool -> bool -> bool` qui renvoie vrai si un seul de ses paramètres est vrai, et faux sinon.

6 Conditionnelles

6.1 Droit de vote

Vous disposez des informations suivantes concernant l'âge à partir duquel un citoyen d'un pays a le droit de voter aux élections :

— au Brésil, en Équateur et en Autriche, on peut voter à partir de 16 ans ;
— au Cameroun, il faut attendre d'avoir 20 ans, tout comme au Japon ;
— en France, en Inde, en Égypte, c'est 18 ans ;
— ailleurs, vous ne savez pas.

Définir une fonction `majorite : string -> int -> string` qui, étant donné un nom de pays et un âge, indique si la personne peut voter ou non. Dans le cas d'un pays pour lequel vous n'avez pas l'information, vous gérerez une erreur.

6.2 Signe du produit

Définir une fonction `signe_produit : int -> int -> string` qui, étant donné deux paramètres entiers, indique si leur produit est positif ou négatif, sans le calculer. Vous n'utiliserez qu'un seul `if...then...else...`.

7 Plus petit entier

1. Définir une fonction `min2 : 'a -> 'a -> 'a` qui renvoie le minimum de deux entiers. Définir une fonction `max2 : 'a -> 'a -> 'a` qui renvoie le maximum de deux entiers.
2. Définir une fonction `min3 : 'a -> 'a -> 'a -> 'a` qui renvoie le minimum de trois entiers. Cette fonction fera appel à `min2`. Définir une fonction `max3 : 'a -> 'a -> 'a -> 'a` qui renvoie le maximum de trois entiers. Cette fonction fera appel à `max2`.
3. Définir une fonction `plus_petit_entier : int -> int -> int -> int` qui renvoie le plus petit entier réalisable avec 3 chiffres. Par exemple, le plus petit entier réalisable avec 6, 1 et 8 est 168. Tester votre fonction sur plusieurs exemples pour être sûr qu'elle fonctionne.