Introduction à OCaml

Selon le système que vous utilisez, utiliser OCaml se fait de différente manière.

— Sur *Microsoft Windows* : c'est assez compliqué. L'approche la plus simple semble être d'installer et activer le Windows Subsystem Linux 2 (WSL2) puis de faire comme sur un Mac, d'installer `opam` puis OCaml. Le plus simple est de faire comme on a vu en TP1, avec une machine virtuelle qui démarre un système *GNU Linux* tournant sous une distribution "grand public" comme Ubuntu ;

— Sur un système *GNU Linux* ou *Mac OS*, on peut utiliser le gestionnaire de paquet du système (ex. `apt-get` sur Ubuntu ou Debian, ou `brew` ou `mac port` sur Mac OS) pour installer `opam` puis OCaml.

— Dans tous les cas, suivez les consignes de la page `https://ocaml.org/learn/tutorials/up_and_running.html`

On peut aussi utiliser un système en ligne, basé sur une page web, qui fonctionne directement dans un navigateur web. Vous pouvez essayer REPL.it, sur `https://REPL.it/` qui marche bien, mais demande de créer un compte et de se connecter. Dans ce cas là, pour des petits TP qui n'utilisent pas la compilation du code OCaml mais juste son interprétation par une console, on va préférer des outils qui soient libres (au sens des logiciels libres), sans espionnage ni fuite de données personnelles, hébergés en France, etc. Pour OCaml, je recommande BetterOCaml (qui est le fruit du travail d'un petit groupe d'étudiants en prépa MP) : allez sur `https://BetterOCaml.ml/` ou sur `https://ocaml.besson.link/` et vous aurez accès à une console OCaml interactive et moderne. Il y a le code à gauche, un bouton pour tout exécuter, et la sortie de la console OCaml à droite. On peut exécuter la "phrase" OCaml actuelle avec `Ctrl+Enter`.

Il est aussi possible d'utiliser OCaml dans un notebook Jupyter, mais cette fonctionnalité est encore en développement, elle devrait être intégrée en octobre dans Basthon (`https://notebook.basthon.fr/`).

Des calculs numériques de bases

— Pour ce premier TP, on va travailler avec `https://BetterOCaml.ml/`. Dans la console à droite ou dans le fichier à gauche, une instruction prend une ou plusieurs ligne et se termine par un double point virgule `;;`.

— Tapez dans la console à droite des valeurs numériques de bases avec les deux types primitifs :

  — **Attention** : les entiers ne sont pas en précision arbitraire comme ceux de Python. Ici, le type `int` représente les entiers entre -2^{63} et $2^{63}-1$;

  — Le type `float` représente les flottants comme ceux de Python, avec la même représentation sur 64 bits : un bit pour le signe, 52 bits pour la mantisse et 11 bits pour l'exposant. Les détails seront revus en cours plus tard.

— Faites des calculs numériques de bases :

  — Les calculs sur les `int` sont faits avec `+`, `-`, `*`, `/` et `mod` pour le modulo (`x mod y` en notation dite *infixe*).

  — Attention, les calculs sur les flottants sont faits avec d'autres symboles, en rajoutant un `.` : `+.`, `-.`, `*.`, `/.`. Il y a aussi `**` pour la puissance chez les flottants, mais contrairement à Python, elle n'a pas son équivalent chez les entiers.

— Observez comment OCaml renvoie toujours les résultats avec leur type : c'est un langage dit *fortement typé*, toute valeur a un type bien clairement défini et décidé à la lecture du programme, avant même son exécution.

Par exemple :

```
# 1 + 2;;
- : int = 3
# 3.1415 /. 4.0;;
- : float = 0.785375
```

Il y a plein de fonctions mathématiques qui sont disponibles directement, un peu comme celles qui sont dans le module `math` de Python. Essayez `cos`, `sin`, `atan` et d'autres.

Opérations d'affichage sur les types de bases

— En plus des deux types de bases que l'on a vu (`int`, `float`), il y a aussi les `char` qui sont des caractères ASCII sur 7 bits (TODO : explication), les `string` qui sont des chaînes de caractères correspondant à des tableaux de `char`, et les `bool` qui n'ont que deux valeurs `true` et `false`.

— Pour chacun de ces types (sauf `bool`), il y a une fonction d'affichage correspondante : `print_int`, `print_float`, `print_char`, `print_string`. Il y a aussi `print_endline` qui fait comme `print_string` mais affiche un `\n` en fin de ligne, pour passer à la ligne.

— Faites des affichages de calculs numériques, un pour chaque opérateur, par exemple :

```
print_string "La réponse à la grande question est = ";;
print_int ( 40 + 2 );;
print_string "\n";;
```

— Définissez une fonction `print_bool` qui affiche `"true"` ou `"false"` selon la valeur du booléen donné en argument. La syntaxe ressemble à `let print_bool b = if b then ... else ...;;`. On peut tout écrire sur une ligne ou sauter des lignes pour aérer, OCaml n'est pas dépendant de l'alignement et des espaces contrairement à Python.

Conversions entre types de bases

— Généralement si une conversion est possible entre des valeurs d'un type `a` et la valeur "équivalente" d'un autre type `b`, deux fonctions existent et sont appelées `a_of_b` et `b_of_a`. Par exemple, `int_of_float` et `float_of_int`, ou `string_of_int` et `int_of_string`.

— Essayez ces fonctions.

À l'assaut de la documentation !

La documentation de OCaml est la source officielle d'information sur le langage, qui ne dispose pas de spécification standardisée (contrairement au C). On la trouve en ligne sur `https://ocaml.org/manual/` et **la documentation n'est malheureusement disponible qu'en anglais**.

— Allez lire la documentation du module `Stdlib` (`https://ocaml.org/api/Stdlib.html`), et essayez de trouver un exemple d'utilisation pour *chaque fonction* des morceaux suivants : *"comparisons"*, *"boolean operations"*, *"integer arithmetic"*, *"floating point arithmetic"*. Pour chaque fonction, soyez créatifs et écrivez un exemple d'utilisation. Par exemple :

```
print_string "Le cosinus de 0 vaut = ");
print_float (cos 0.0);
print_endline "";;
```

Interlude : encore des entraînements d'écriture au clavier

Vous pouvez retourner sur `https://www.ratatype.fr/typing-test/test/` ou bien essayer le test sur `https://typing.io/lesson/python/mercurial/merge.py/1` (aller sur `https://typing.io/` puis cliquer sur démo sans s'enregistrer, et choisir Python).

Vos premières fonctions

— Définir des fonctions qui travaillent sur des `int` et d'autres sur des `float`. Par exemple combinez `+`, `-`, `*`, `/` et `mod` sur les entiers, ou `+.`, `-.`, `*.`, `/.` et `**` sur les flottants.

— À l'aide des fonctions de conversion de types `int_of_float` et `float_of_int`, et de la puissance `x ** y` qui fonctionne uniquement pour des arguments de type `float`, écrivez une fonction `puissance_entiere` qui calcule les puissances chez les entiers.

— Même question, écrivez une fonction `puissance_entiere` qui calcule les puissances chez les entiers, mais cette fois en utilisant une boucle `for`, dont voici la syntaxe de base :

```
for i = 1 to n do (* 1 to n, tous les deux inclus *)
    corps de la boucle, autant de lignes qu'on veut
    chaque ligne se termine par un seul séparateur ;
done;;  (* ici il y a le ;; de fin de bloc *)
```

Introduction à la syntaxe des fonctions dans OCaml :

— Chaque ligne intermédiaire qui fait une opération se termine généralement par un point virgule `;`.

— Définir une variable qui ne va pas changer dans le corps d'une fonction se fait avec `let variable = valeur in`.

— Au contraire, si la variable à vocation à changer et évoluer, il est généralement d'usage d'utiliser ce qui s'appelle des **références** en OCaml, comme ici la variable `reponse`. On lit le contenu d'une référence en écrivant `!reponse` (`!`) et on écrit dans une référence avec `reponse := nouvelle_valeur;`.

— La conditionnelle `Si condition Alors bloc1 Sinon bloc2 FinSi` s'écrit simplement `if condition then bloc1 else bloc2` en OCaml.

Suite de l'exercice :

— On peut définir cette fonction `puissance_entiere` d'une autre façon, récursivement, avec l'initialisation `puissance_entiere_rec(x, 0) = 1` et ensuite la récurrence `puissance_entiere_rec(x, n+1) = x * puissance_entiere_rec(x, n)`. En OCaml, il faut utiliser `let rec puissance_entiere_rec x n = ...` avec le mot clé `rec` pour autoriser une fonction à être récursive. Vérifiez que la console OCaml vous interdit de définir la fonction récursivement si on omet le mot clé `rec`. Le message d'erreur devrait être très clair bien qu'étant en anglais.

— Testez vos deux fonctions `puissance_entiere` sur différentes valeurs entières de `x` la valeur et de `n` l'exposant.

Correction partielle du DS numéro 1 en OCaml

Ex.2 : Factorielle et triangle de Pascal

— Écrivez une fonction de signature `int -> int` qui calcule la factorielle, `factorielle`$(n) = n! = \prod_{k=1}^{n} k$, avec une boucle `for` et pas de récursivité ;

— Écrivez une variante récursive de la même fonction. Écrite ainsi naïvement, en OCaml, quel sera l'inconvénient de cette variante récursive comparée à la variante itérative (avec un `for`) ? On apprendra en cours comment écrire des fonctions dites *récursives terminales*, qui n'ont pas cette limitation.

— On rappelle que le nombre de façons de choisir k élément parmi une liste de n valeurs (avec remises) se note $\binom{n}{k}$, et qu'une façon de le définir est $\binom{n}{k} = n!/(k! * (n-k)!)$. Écrivez une fonction `choixParmi` qui prend (k, n) comme argument et renvoie $\binom{n}{k}$ en utilisant votre fonction `factorielle`.

— On donne les cas de base $\binom{n}{1} = n$, et la relation de récurrence du triangle de Pascal est $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$. Écrivez une autre fonction `choixParmiRec` qui utilise ces cas de base et cette relation de récurrence pour calculer la même chose.

Ex.3 : Figures en mode texte

Écrivez une fonction `afficheCarre` qui affiche une figure comme celle-ci, pour n la hauteur (et la largeur) donnée en argument de la fonction.

Conseil : utilisez `print_string` pour afficher des chaînes sans sauter à la ligne, combinée avec des boucles `for`, et des `print_endline ""` pour sauter à la ligne sans rien afficher.

```
xxxxxx
x    x
x    x
x    x
x    x
xxxxxx
```


Ex.5 : Pseudo-code à corriger

Traduisez le pseudo-code suivant en OCaml, qui donne une fonction qui permet de vérifier que toutes les valeurs du tableau `tab` donné en argument sont des multiples de 93 ?

```
Fonction tousMultiples93(tab) =
    N = longueur(tab)
    reponse = true
    Pour chaque i = 0 à N-1 (inclus) faire
        Si (tab.(i) mod 93) != 0 Alors
            reponse = false (* écriture de la référence *)
    Fin Pour
    reponse (* lecture de la référence *)
    (* le contenu de la dernière ligne est la valeur renvoyée par la fonction *)
;;
```

Suite de l'introduction à la syntaxe de OCaml :

— On utilise ici un tableau d'entiers `tab`, qui est de type `int array`. Lire la `i`-ième case d'un tableau se fait avec `tab.(i)` pour `i` allant de `0` à `n-1` si `n` est la longueur du tableau. Cette longueur s'obtient avec la fonction `Array.length tab`, c'est-à-dire avec une fonction `length` qui vient du module `Array`. Pour votre curiosité et la suite, sachez aussi que l'on écrit en case `i` du tableau `tab` avec la syntaxe `tab.(i) <- nouvelle_valeur`.

Ex.6 : Racines du second degré

On suppose que $a \neq 0$, et a, b, c sont trois nombres flottants. Écrivez une fonction appelée `racinesSecondDegre` qui renvoie les deux racines (ou deux fois la racine simple) de l'équation du second degré $ax^2 + bx + c = 0$.

— La plupart des fonctions numériques du module `math` de Python sont disponibles dans OCaml par défaut, sans avoir besoin d'utiliser un module supplémentaire. Vous disposez donc de la fonction `sqrt : float -> float` par défaut.

— On peut se restreindre au cas où les racines seront réelles.

Ex.7 : Signatures curryfiées ou non

— Soit $f(a, b) = (7 * a + b) \bmod 10$, donner son code en OCaml, et donner sa signature sous la forme curryfiée (à la OCaml, qui permet l'application partielle), et décurryfiée (à la Python, qui ne permet pas l'application partielle). Vérifiez que la console OCaml donne les deux signatures différentes, selon si on les définit comme `let f a b = ...` (curryfiée) ou `let f (a, b) = ...` (décurryfiée).

— Même question avec la fonction `racinesSecondDegre` de l'exercice 6.

Ex.8 Moyennes arithmétique, géométrique et harmonique

On va supposer avoir un tableau `tab` de `float` (donc une valeur `tab` de type `float array`). On donne les formules suivantes pour calculer respectivement $Ma(\texttt{tab})$, $Mg(\texttt{tab})$ et $Mh(\texttt{tab})$ les moyennes arithmétique, géométrique et harmonique du tableau `tab`.

— $Ma(\texttt{tab}) = \frac{1}{n}\sum_{i=0}^{n-1}\texttt{tab}[i]$;

— $Mg(\texttt{tab}) = \prod_{i=0}^{n-1}\texttt{tab}[i]^{\frac{1}{n}}$;

— $Mh(\texttt{tab}) = n/\Big(\sum_{i=0}^{n-1} 1/\text{tab}[i]\Big)$.

Écrivez trois fonctions OCaml qui calcule ces formules, avec une boucle `for`, une référence initialisée à `0` pour les sommes partielles, et en utilisant les opérateurs sur les flottants (`+.` et `*.` pour l'addition et la multiplication, et la double `**` pour la puissance).

Pour initialiser un tableau, on utilise par exemple `tab = [| 2; 4; 5; 18 |]`.

Pour quelques tableaux différents, affichez dans la console les résultats des trois fonctions pour ce tableau. Est-ce qu'une relation d'ordre entre les trois moyennes semble apparaître ?

