

OcaML Cheat Sheet

Commandes en vrac:

- caster un type dans un autre:
 - `float_of_int <int>` : renvoie la même valeur mais en `float`
 - `int_of_float <float>` : renvoie la partie entière du `float` (entier arrondi en dessous)
- Afficher un message dans la console avec `Printf.printf "message" (<variable>)`
 - caractères utiles:
 - * `\n` revient à la ligne, souvent utilisé en fin de message.
 - * `%!` vide le buffer du print. Absolument **toujours** l'utiliser en **fin** de message. Ne pas le mettre veut dire ne jamais libérer la mémoire utilisée par le print, donc de plus en plus de mémoire inutilement inutilisé après chaque `printf`. Non je sais pas pourquoi il faut le préciser.
 - pour insérer une variable dans le message, on écrit `%<première_lettre_du_type>`. Les variables vont être insérées dans l'ordre dont elles sont entrées entre les parenthèses.
 - exemples:
 - * `Printf.printf "%i\n%!" (12) -> "12"`
 - * `Printf.printf "Les variables x et b valent respectivement %f et %b\n%!" (x b) -> "Les variables x et b valent respectivement 42.12 et false"`

Types de données, variables et fonctions:

1. types de données:
 - `int` (nombre entiers)
 - `float` (nombre à virgule)
 - `bool` (`true` / `false`)
 - `char` (un seul caractère: `'a'`)
 - `str` (une chaîne de caractères: `"AAAAAA"`)
 - `unit` (ne renvoie rien)
 - toujours faire très attention aux types utilisés.
2. opérateurs:
 - pour les `int`: `+`; `-`; `*` (fois); `/` (division)
 - pour les `float`: pareil que pour les `int`s, mais suivis d'un point (ex: `+.`)
 - pour les `bool`s: `||` (ou); `&&` (et); `not` (inverse la valeur du booléen); `=` (est égal à); `<>` (est différent à); `>` (supérieur) `<=` (inférieur ou égal)
 - pour les `strings`: `^` pour concaténer les chaînes (les mettre bout à bout).
3. déclarer une variable:
 - syntaxe: `let <nom_de_la_variable> : <type> = <valeur>`

- exemples:
 - `let x : int = 69`
 - `let b : bool = true`
 - remarque: une variable est une fonction sans argument qui renvoie une constante.
4. déclarer une fonction:
- syntaxe: `let <nom_de_la_fonction> <arguments> : <type_de_sortie> = <expression>`
 - pour chaque argument, on fait (`<nom_argument>: <type_argument>`)
 - le type de sortie est le type de la valeur renvoyé.
 - exemple:
 - `let carre (x: float) : float = x*x`
 - `let estPair (x: int) : bool = (x mod 2 = 0)` (mod: division euclidienne de deux entiers)
 - `let moyenne (x: float) (y: float) : float = (x + y) / 2`
 - `let message : unit = Printf.printf "un message très utile \n%!"`
5. appeler une fonction:
- syntaxe: `<nom_de_la_fonction> <argument1> <argument2> ... <argumentn>`
 - appeler une fonction revient à utiliser une fonction prédéfinie, avec des variables données en entrée.
 - **ATTENTION** ne pas mettre de parenthèses au tour des arguments, comme en python.
 - exemples:
 - `carre 5` (renvoie 25)
 - `estPair (-3)` (renvoie false, nombre négatifs doivent être entre parenthèses)
 - `moyenne 12.5 16.5` (renvoie 14.5)
 - `moyenne (12.5 16.5)` (erreur, car la fonction voit le groupe de parenthèses comme un seul argument, donc il manque un argument).
 - `message` (renvoie rien, affiche “un message très utile” dans la console)
6. définir un type:
- lol j’ai pas compris, on verra ça plus tard

Tests et conditions:

1. Tests avec if:
- syntaxe:


```
if <expression_booléenne> then
  <expression>
else if <expr_bool> then
  <expression>
```

- ```

...
else
 <expression>

```
- tous les types de sortie doivent être identique.
  - un if qui renvoie `true` ou `false` est (en général) inutile.
  - exemples:
    - `let y = 15`
    - `if x >= 10 then`
    - `moyenne x y`
    - `else`
    - `moyenne 10 y`
    - `if estPair a then a + 1`
2. Les match-expressions (ou pattern matching):
- syntaxe:
 

```

match <expression> with
| <val1> | <val2> | ... | <valn> -> <expression 1>
...
| <variable> -> <dernière expression>

```
  - toutes les expressions de sorties doivent renvoyer le même type.
  - le cas `<variable>` va “récupérer” tous les cas non traité, cette variable peut être utilisé dans l’expression associé.
  - exemple, on verifie que le jour donné est possible:
 

```

match mois with
| 1 | 3 | 5 | 7 | 8 | 10 | 12 -> (1 <= jour) && (jour <= 31)
| 4 | 6 | 9 | 11 -> (1 <= jour) && (jour <= 30)
| 2 -> (1 <= jour) && (jour <= 28)
| n -> Printf.printf "le mois %d n'existe pas\n%!" (n); 1

```

## Composition de fonctions:

1. Le mot clé `in`:
- syntaxe: `let <var/func> = <expression> and <var/func> = <expression> and ... in <expression>`
  - permet de faire de la composition de fonction (comme  $f \bullet g$  en maths).
  - exemples:
    - `let x = carre y in estPair x`
    - `let a = x*2 and b = y*2 in moyenne a b`
    -
- ```

let min4 (a: int) (b: int) (c: int) (d: int) : int =
  let min2 (x: int) (y: int) =
    if x > y then x else y
  in
  let m = min2 a b and n = min2 c d in
  min2 m n

```