

Exemple de notebook dans un autre langage interprété OCaml

January 30, 2020

1 Table of Contents

- 1 Exemple de notebook dans un autre langage interprété : OCaml
 - 1.1 Une faiblesse d'OCaml ?
 - 1.2 Notebooks Jupyter pour le langage OCaml
 - 1.2.1 Un premier petit exemple : fonction factorielle
 - 1.3 Mesure du temps d'exécution
 - 1.4 Exemple : solveur de Sudoku en OCaml
 - 1.5 Conclusion pour ce notebook

2 Exemple de notebook dans un autre langage interprété : OCaml

OCaml (<https://www.ocaml.org/>) est un langage de programmation fonctionnelle, statiquement typé, qui peut être utilisé en mode interprété (comme Python) ou compilé. Conçus par des chercheurs de l'INRIA, hériter de CAML puis `caml_light`, le langage utilisé pour l'enseignement de l'option informatique en classes préparatoires MPSI et MP en France.

OCaml, et les langages fonctionnels en général, n'est pas un langage très populaire dans le monde, en comparaison de C/C++, Java, PHP, Python et Javascript (les cinq plus populaires), mais il a des avantages, et reste très utilisé pour l'enseignement de l'informatique théorique en France.

Liens : - <http://caml.inria.fr/> et <https://www.ocaml.org/> sont les sites officiels du langage, - <http://try.ocamlpro.com/> pour essayer le langage depuis un navigateur, - Par exemple <http://www.janestreet.com/> est un groupe de trading, qui est connu pour utiliser des applications codées en OCaml, - Un autre exemple, Facebook utilise OCaml pour ses outils d'analyse statique de code, comme [flow](#) et [infer](#), et pour son environnement [Reason](#).

2.1 Une faiblesse d'OCaml ?

OCaml est connu pour manquer d'environnement de développement moderne et "séduisant", et la plupart des enseignants utilisent Emacs et son mode Tuareg pour Caml/OCaml, qui manque de modernité.

Les élèves habitués à des logiciels très modernes, faciles d'accès et très visuels, se retrouvent confrontés à un logiciel qui n'a guère évolué depuis les années 1980.

Mon expérience montre que mêmes des élèves de niveau master, préparant l'agrégation de mathématiques option informatique à l'ENS de Rennes, ont un peu de mal à prendre correctement en main le logiciel Emacs et son mode Tuareg. A l'opposé, j'ai vu des élèves de première et deuxième années de prépa (MPSI et PSI) maîtriser très bien l'environnement Jupyter, qu'ils avaient appris à utiliser pour l'oral "maths avec Python" du concours CentraleSupélec.

Liens : - <https://opam.ocaml.org/packages/tuareg/> -
https://pro.yannsalmon.fr/ide#opam_ocaml_emacs_tuareg

2.2 Notebooks Jupyter pour le langage OCaml

Utiliser des notebooks Jupyter avec le *kernel* `ocaml-jupyter` (<https://github.com/akabe/ocaml-jupyter>) est une très bonne solution pour montrer du code, des commentaires, et les résultats d'évaluation des morceaux de code, pour des cours, TD ou TP d'informatique avec OCaml.

```
[53]: let entier = 0
      and flottant = 1.0
      and chaine = "Superman"
      and fonction = fun x -> fun y -> x + y
      ;;
```

```
[53]: val entier : int = 0
      val flottant : float = 1.
      val chaine : string = "Superman"
      val fonction : int -> int -> int = <fun>
```

Comme pour Python, chaque cellule contient du texte (en Markdown), ou du code, et sa sortie une fois exécuté. Il est très facile d'écrire des sujets de TP ou d'examen, qui mélangent tous ces contenus, grâce à Jupyter.

Les documents écrits avec Jupyter peuvent ensuite être convertis en PDF, en HTML, ou d'autres formats.

2.2.1 Un premier petit exemple : fonction factorielle

Exécuter une cellule donne la signature des objets définis dans la cellule, i.e., leurs valeurs et/ou leurs types.

Par exemple, si on définit $\text{fact}(n) = n!$ avec $n! = 1 \times 2 \times \dots \times n$, `fact : int -> int` est la signature de la fonction `fact`.

```
[54]: let rec fact = function
      | 0 -> 1
      | 1 -> 1
      | n -> n * (fact (n-1))
      ;;
```

```
[54]: val fact : int -> int = <fun>
```

```
[61]: for i = 0 to 10 do
      Printf.printf "fact(%i) = %i\n" i (fact i);
done;
flush_all();;
```

```
fact(0) = 1
fact(1) = 1
fact(2) = 2
fact(3) = 6
fact(4) = 24
fact(5) = 120
fact(6) = 720
fact(7) = 5040
fact(8) = 40320
fact(9) = 362880
fact(10) = 3628800
```

```
[61]: - : unit = ()
```

Et par exemple pour la fonction calculant les termes positifs de la suite de Fibonacci :

```
[21]: let rec fibo = function
      | 0 -> 1
      | 1 -> 1
      | n -> (fibo (n-1)) + (fibo (n-2))
      ;;
```

```
[21]: val fibo : int -> int = <fun>
```

```
[23]: for i = 10 to 20 do
      Printf.printf "fibo(%i) = %i\n" i (fibo i);
done;
flush_all();;
```

```
fibo(10) = 89
fibo(11) = 144
fibo(12) = 233
fibo(13) = 377
fibo(14) = 610
fibo(15) = 987
fibo(16) = 1597
fibo(17) = 2584
fibo(18) = 4181
fibo(19) = 6765
fibo(20) = 10946
```

```
[23]: - : unit = ()
```

2.3 Mesure du temps d'exécution

```
[38]: let timeit f x () =  
      let debut = Sys.time () in  
      let resultat = f x in  
      let fin = Sys.time () in  
      Printf.printf "Calcul effectué en %.3g secondes.\n" (fin -. debut);  
      flush_all();  
      resultat  
      ;;
```

```
[38]: val timeit : ('a -> 'b) -> 'a -> unit -> 'b = <fun>
```

```
[48]: timeit fibo 34 ();;
```

Calcul effectué en 0.276 secondes.

```
[48]: - : int = 9227465
```

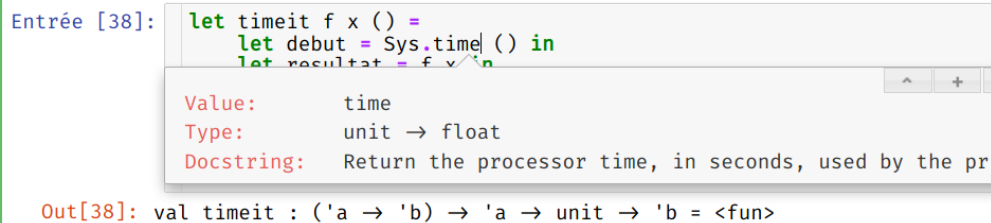
```
[49]: timeit fibo 36 ();;
```

Calcul effectué en 0.712 secondes.

```
[49]: - : int = 24157817
```

Comme avec Python, on peut utiliser Tab pour afficher de l'aide sur les fonctions de la librairie standard du langage (de l'introspection). C'est très pratique :

1.3 Mesure du temps d'exécution



```
Entrée [38]: let timeit f x () =  
             let debut = Sys.time () in  
             let resultat = f x in  
             let fin = Sys.time () in  
             Printf.printf "Calcul effectué en %.3g secondes.\n" (fin -. debut);  
             flush_all();  
             resultat  
             ;;
```

Value: time
Type: unit → float
Docstring: Return the processor time, in seconds, used by the pr

```
Out[38]: val timeit : ('a → 'b) → 'a → unit → 'b = <fun>
```

2.4 Exemple : solveur de Sudoku en OCaml

Pour illustrer l'utilisation de ce langage, voyons un exemple issu du tutoriel officiel (<https://ocaml.org/learn/tutorials/99problems.html#97-Sudoku-medium>) : un solveur de Sudoku.

[65]: `open Printf`

```
module Board = struct
  type t = int array (* 9x9, row-major representation. A value of 0
                      means undecided. *)

  let is_valid c = c >= 1;;

  let get (b: t) (x, y) = b.(x + y * 9);;

  let get_as_string (b: t) pos =
    let i = get b pos in
    if is_valid i then string_of_int i else ".";;

  let with_val (b: t) (x, y) v =
    let b = Array.copy b in
    b.(x + y * 9) <- v;
    b;;

  let of_list l : t =
    let b = Array.make 81 0 in
    List.iteri (fun y r -> List.iteri (fun x e ->
      b.(x + y * 9) <- if e >= 0 && e <= 9 then e else 0) r) l;
    b;;

  let print b =
    for y = 0 to 8 do
      for x = 0 to 8 do
        printf (if x = 0 then "%s" else if x mod 3 = 0 then " | %s"
              else " %s") (get_as_string b (x, y))
      done;
      if y < 8 then
        if y mod 3 = 2 then printf "\n-----+-----+\n"
        else printf "\n      |           |           \n"
        else printf "\n"
      done;
    flush_all();;

  let available b (x, y) =
    let avail = Array.make 10 true in
    for i = 0 to 8 do
      avail.(get b (x, i)) <- false;
      avail.(get b (i, y)) <- false;
    done;
    let sq_x = x - x mod 3 and sq_y = y - y mod 3 in
    for x = sq_x to sq_x + 2 do
      for y = sq_y to sq_y + 2 do
```

```

        avail.(get b (x, y)) <- false;
    done;
done;
let av = ref [] in
for i = 1 (* not 0 *) to 9 do if avail.(i) then av := i :: !av done;
!av;;

let next (x,y) = if x < 8 then (x+1, y) else (0, y+1);;

(** Try to fill the undecided entries. *)
let rec fill b ((_,y) as pos) =
    if y > 8 then Some b (* filled all entries *)
    else if is_valid(get b pos) then fill b (next pos)
    else match available b pos with
        | [] -> None (* no solution *)
        | l -> try_values b pos l
and try_values b pos = function
    | v :: l ->
        (match fill (with_val b pos v) (next pos) with
         | Some _ as res -> res
         | None -> try_values b pos l)
    | [] -> None
;;
end

```

```

[65]: module Board :
sig
    type t = int array
    val is_valid : int -> bool
    val get : t -> int * int -> int
    val get_as_string : t -> int * int -> string
    val with_val : t -> int * int -> int -> int array
    val of_list : int list list -> t
    val print : t -> unit
    val available : t -> int * int -> int list
    val next : int * int -> int * int
    val fill : t -> int * int -> t option
    val try_values : t -> int * int -> int list -> t option
end

```

```

[66]: let sudoku b = match Board.fill b (0,0) with
    | Some b -> b
    | None -> failwith "sudoku: no solution";;

```

```

[66]: val sudoku : Board.t -> Board.t = <fun>

```

```
[67]: (* The board representation is not imposed. Here "0" stands for "." *)
let initial_board =
  Board.of_list [[0; 0; 4; 8; 0; 0; 0; 1; 7];
                [6; 7; 0; 9; 0; 0; 0; 0; 0];
                [5; 0; 8; 0; 3; 0; 0; 0; 4];
                [3; 0; 0; 7; 4; 0; 1; 0; 0];
                [0; 6; 9; 0; 0; 0; 7; 8; 0];
                [0; 0; 1; 0; 6; 9; 0; 0; 5];
                [1; 0; 0; 0; 8; 0; 3; 0; 6];
                [0; 0; 0; 0; 0; 6; 0; 9; 1];
                [2; 4; 0; 0; 0; 1; 5; 0; 0]];
```

```
[67]: val initial_board : Board.t =
  [[0; 0; 4; 8; 0; 0; 0; 1; 7; 6; 7; 0; 9; 0; 0; 0; 0; 0; 5; 0; 8; 0; 3; 0;
    0; 0; 4; 3; 0; 0; 7; 4; 0; 1; 0; 0; 0; 6; 9; 0; 0; 0; 7; 8; 0; 0; 0; 1;
    0; 6; 9; 0; 0; 5; 1; 0; 0; 0; 8; 0; 3; 0; 6; 0; 0; 0; 0; 0; 6; 0; 9; 1;
    2; 4; 0; 0; 0; 1; 5; 0; 0]]
```

```
[68]: Board.print initial_board;;
```

```
. . 4 | 8 . . | . 1 7
      |       |
6 7 . | 9 . . | . . .
      |       |
5 . 8 | . 3 . | . . 4
-----+-----+-----
3 . . | 7 4 . | 1 . .
      |       |
. 6 9 | . . . | 7 8 .
      |       |
. . 1 | . 6 9 | . . 5
-----+-----+-----
1 . . | . 8 . | 3 . 6
      |       |
. . . | . . 6 | . 9 1
      |       |
2 4 . | . . 1 | 5 . .
```

```
[68]: - : unit = ()
```

```
[69]: Board.print (sudoku initial_board);;
```

```
9 3 4 | 8 2 5 | 6 1 7
      |       |
6 7 2 | 9 1 4 | 8 5 3
      |       |
```

```

5 1 8 | 6 3 7 | 9 2 4
-----+-----+-----
3 2 5 | 7 4 8 | 1 6 9
      |       |
4 6 9 | 1 5 3 | 7 8 2
      |       |
7 8 1 | 2 6 9 | 4 3 5
-----+-----+-----
1 9 7 | 5 8 2 | 3 4 6
      |       |
8 5 3 | 4 7 6 | 2 9 1
      |       |
2 4 6 | 3 9 1 | 5 7 8

```

```
[69]: - : unit = ()
```

```
[87]: timeit sudoku initial_board ();;
```

Calcul effectué en 0.000219 secondes.

```
[87]: - : Board.t =
[|9; 3; 4; 8; 2; 5; 6; 1; 7; 6; 7; 2; 9; 1; 4; 8; 5; 3; 5; 1; 8; 6; 3; 7; 9;
 2; 4; 3; 2; 5; 7; 4; 8; 1; 6; 9; 4; 6; 9; 1; 5; 3; 7; 8; 2; 7; 8; 1; 2; 6;
 9; 4; 3; 5; 1; 9; 7; 5; 8; 2; 3; 4; 6; 8; 5; 3; 4; 7; 6; 2; 9; 1; 2; 4; 6;
 3; 9; 1; 5; 7; 8|]
```

2.5 Conclusion pour ce notebook

Ce petit notebook montre que la même technologie (les notebooks Jupyter) peut être utilisée pour d'autres langages que Python.

Une très longue liste de *kernel* existe, avec notamment :

- <https://github.com/jupyter-xeus/xeus-cling> pour C++
- <https://github.com/JuliaLang/IJulia.jl> pour Julia
- <https://github.com/jupyter/jupyter/wiki/Jupyter-kernels> et <https://jupyter.readthedocs.io/en/latest/projects.html> pour la liste des kernels qui existent.