Exemple de notebook avec OCaml

February 13, 2021

1 Exemple de notebook avec OCaml

1.1 Explications

Le kernel OCaml n'est pas installé par défaut avec Jupyter.

Il faut installer OPAM, puis ocaml-jupyter.

1.2 Exemples

[1]:
```
Sys.command "ocaml -version";;
```

```
The OCaml toplevel, version 4.05.0
```

[1]:
```
- : int = 0
```

[2]:
```
print_endline "Bonjour depuis OCaml !";;
```

```
Bonjour depuis OCaml !
```

[2]:
```
- : unit = ()
```

1.2.1 Une fonction récursive

Pour calculer la fonction $n! := 1 \times 2 \times \cdots \times n$ ($n \in \mathbb{N}$), on peut penser à une solution récursive (qui coutera en espace mémoire à cause de la pile d'appel) et une solution impérative.

[20]:
```
let rec fact (n: int) : int =
    match n with
    | 0 -> 1
    | n -> n * (fact (n-1))
;;
```

[20]:
```
val fact : int -> int = <fun>
```

[22]:
```
for i = 1 to 21 do
    print_endline (Printf.sprintf "fact(%.2i) = %i" i (fact i))
done;;
```

```
fact(01) = 1
fact(02) = 2
fact(03) = 6
fact(04) = 24
fact(05) = 120
fact(06) = 720
fact(07) = 5040
fact(08) = 40320
fact(09) = 362880
fact(10) = 3628800
fact(11) = 39916800
fact(12) = 479001600
fact(13) = 6227020800
fact(14) = 87178291200
fact(15) = 1307674368000
fact(16) = 20922789888000
fact(17) = 355687428096000
fact(18) = 6402373705728000
fact(19) = 121645100408832000
fact(20) = 2432902008176640000
fact(21) = -4249290049419214848
```

[22]:
```
- : unit = ()
```

Et la solution impérative :

[23]:
```
let fact_imp (n: int) : int =
    let f = ref 1 in
    for i = 1 to n do
        f := (!f) * i;
    done;
    !f
;;
```

[23]:
```
val fact_imp : int -> int = <fun>
```

[32]:
```
for i = 15 to 21 do
    print_endline (Printf.sprintf "fact(%.2i) = %i" i (fact_imp i))
done;;
```

```
fact(15) = 1307674368000
fact(16) = 20922789888000
fact(17) = 355687428096000
fact(18) = 6402373705728000
fact(19) = 121645100408832000
fact(20) = 2432902008176640000
fact(21) = -4249290049419214848
```

[32]:
```
- : unit = ()
```

Comme les entiers sont bornés, on dépasse la capacité assez rapidement, et les valeurs calculées deviennent fausses.

1.2.2 Un type non paramétrique récursif et un exemple :

[34]:
```
type formulePropositionnelle =
    | Var of string
    | Non of formulePropositionnelle
    | Ou of (formulePropositionnelle * formulePropositionnelle)
    | Et of (formulePropositionnelle * formulePropositionnelle)
;;
```

[34]:
```
type formulePropositionnelle =
    Var of string
  | Non of formulePropositionnelle
  | Ou of (formulePropositionnelle * formulePropositionnelle)
  | Et of (formulePropositionnelle * formulePropositionnelle)
```

[35]:
```
let x = Var("x")
and y = Var("y")
and z = Var("z")
;;
```

[35]:
```
val x : formulePropositionnelle = Var "x"
val y : formulePropositionnelle = Var "y"
val z : formulePropositionnelle = Var "z"
```

[36]:
```
let p1 = Ou(x, y)
and p2 = Et(y, z)
and p3 = Non(x)
;;

let p4 = Et(p1, p2);;
let p5 = Ou(p3, p4);;
```

```
let p6 = Non(p5);;
```

```
val p1 : formulePropositionnelle = Ou (Var "x", Var "y")
val p2 : formulePropositionnelle = Et (Var "y", Var "z")
val p3 : formulePropositionnelle = Non (Var "x")
```

```
val p4 : formulePropositionnelle =
  Et (Ou (Var "x", Var "y"), Et (Var "y", Var "z"))
```

```
val p5 : formulePropositionnelle =
  Ou (Non (Var "x"), Et (Ou (Var "x", Var "y"), Et (Var "y", Var "z")))
```

```
val p6 : formulePropositionnelle =
  Non (Ou (Non (Var "x"), Et (Ou (Var "x", Var "y"), Et (Var "y", Var "z"))))
```

```
let rec taille (formule: formulePropositionnelle) : int =
    match formule with
    | Var _ -> 1
    | Non phi -> 1 + taille phi
    | Et (phi1, phi2) | Ou (phi1, phi2) -> 1 + (taille phi1) + (taille phi2)
    (* | _ -> 0 *) (* cette ligne est inutile *)
;;
```

```
val taille : formulePropositionnelle -> int = <fun>
```

```
taille x;;
taille y;;
taille z;;
```

```
- : int = 1
```

[38]:
```
- : int = 1
```

[38]:
```
- : int = 1
```

[39]:
```
taille p1;;
taille p2;;
taille p3;;
```

[39]:
```
- : int = 3
```

[39]:
```
- : int = 3
```

[39]:
```
- : int = 2
```

Avec une fonction récursive terminale :

[40]:
```
let taille (formule: formulePropositionnelle) : int =
    let rec aux acc = function
        | Var _ -> 1
        | Non phi -> aux (acc+1) phi
        | Et (phi1, phi2) | Ou (phi1, phi2) -> aux (aux (acc + 1) phi1) phi2
    in aux 0 formule
;;
```

[40]:
```
val taille : formulePropositionnelle -> int = <fun>
```

[41]:
```
taille p4;;
taille p5;;
taille p6;;
```

[41]:
```
- : int = 1
```

[41]:
```
- : int = 1
```

[41]:
```
- : int = 1
```

1.2.3 Un type paramétrique récursif et un exemple :

[42]:
```
type 'a arbreBinaire = Feuille | Noeud of ('a arbreBinaire * 'a * 'a␣
↪arbreBinaire);;
```

[42]:
```
type 'a arbreBinaire =
    Feuille
  | Noeud of ('a arbreBinaire * 'a * 'a arbreBinaire)
```

[43]:
```
Noeud(Feuille, 10, Feuille)
```

[43]:
```
- : int arbreBinaire = Noeud (Feuille, 10, Feuille)
```

1.2.4 D'autres exemples ?

TODO: plus tard !

1.3 Pour en apprendre plus

- Ce petit tutoriel : https://perso.crans.org/besson/apprendre-python.fr.html (sous licence GPLv3) ;
- Ce WikiBooks : https://fr.wikibooks.org/wiki/Programmation_Python (sous licence libre) ;
- Ces deux livres de Python au niveau lycée : https://github.com/exo7math/python1-exo7 et https://github.com/exo7math/python2-exo7 (sous licence Creative Commons).