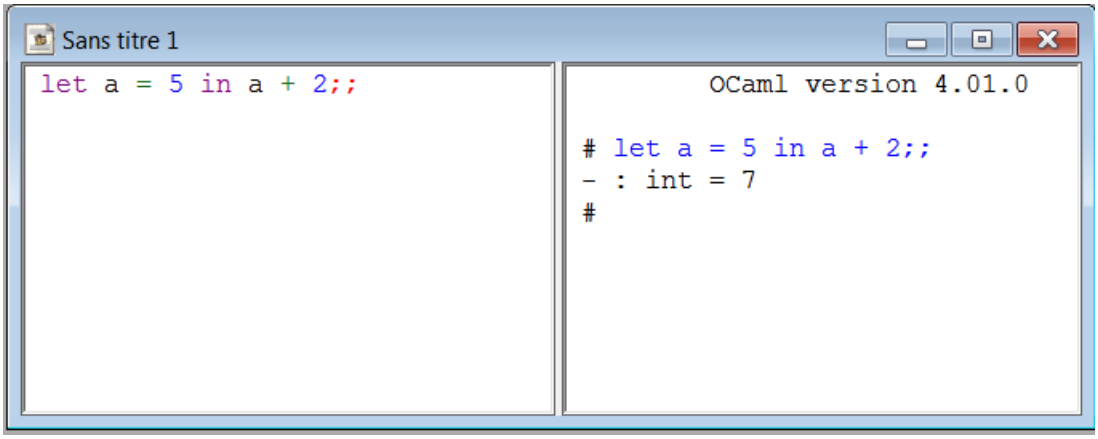
1 Découverte de l'environnement Wincaml

Lancer Wincaml. Vérifier dans le menu CamlTop que le langage est bien OCaml. Une fenêtre Wincaml ressemble à cela :



La fenêtre de gauche sert à écrire du code et la fenêtre de droite affiche le résultat des différentes exécutions. Pour exécuter du code, il faut le terminer par `;;` et appuyer sur la combinaison de touches Ctrl + Entrée.

1. Faire les calculs suivants (attention aux types !) :
   - $3 + 2$
   - $16 \mod 5$
   - $0,5 \times 7$
   - 3^5
2. Écrire les fonctions de $\mathbb{N}$ dans $\mathbb{N}$ suivantes :
   - $f : x \longrightarrow x + 2$
   - $g : x \longrightarrow x^2$
   - $h : x \longrightarrow x^2 + 2$

2 Fonctions de base sur les listes

1. Avec les définitions données dans le cours, écrire les fonctions sur les listes suivantes :
   - `length : 'a list -> int`
   - `mem : 'a -> 'a list -> bool`
   - `append : 'a list -> 'a list -> 'a list`
   - `map : ('a -> 'b) -> 'a list -> 'b list`
   - `iter : ('a -> unit) -> 'a list -> unit`

   On écrira les fonctions suivantes en écrivant un message d'erreur si la dimension de la liste ne permet pas de renvoyer de résultat. On utilisera des fonctions locales si nécessaire.

2. Fonction `last : 'a list -> 'a` qui renvoie le dernier élément d'une liste.
3. Fonction `nth : 'a list -> int -> 'a` qui renvoie le n-ème élément d'une liste.
4. Fonction `sum : int list -> int` qui calcule la somme des éléments d'une liste.
5. Fonction `sorted : 'a list -> bool` qui teste si la liste est triée par ordre croissant.
6. Fonction `max : 'a list -> 'a` qui renvoie l'élément maximal dans une liste.
7. Fonction `rev : 'a list -> 'a list` qui renverse une liste.
8. Fonction `flatten : 'a list list -> 'a list` qui concatène les éléments d'une liste de listes.

9. Fonction `partition : ('a -> bool) -> 'a list -> 'a list * 'a list` telle que `partition p l` renvoie une partition `l1, l2` telle que `l1` est l'ensemble des éléments de `l` vérifiant le prédicat `p`, et `l2` ceux qui ne le vérifient pas. Par exemple, `partition (function x -> x < 0) [-1; -2; 3; -4; 5]` renvoie `[-1; -2; -4], [3; 5]`.
10. Fonction `readint : int list -> int` qui renvoie l'entier obtenu en écrivant les nombres dans l'ordre de la liste. Par exemple, `readint [1; 23; 4]` renvoie `1234`.
11. Fonction `sublist : 'a list -> 'a list -> bool` qui teste si une liste est une sous-liste d'une autre. Par exemple, `[2; 4]` est une sous-liste de `[1; 2; 4; 5]` et `[1; 2; 3; 4; 5]`, mais pas de `[5; 4; 3; 2]`.
12. Fonction `pattern : 'a list -> 'a list -> bool` qui teste si une liste est un motif dans une autre. Par exemple, `[2; 4]` est un motif de `[1; 2; 4; 5]`, mais pas de `[1; 2; 3; 4; 5]`.

3 Des suites et des listes

Dans cette partie, on souhaite, à partir d'une suite $(u_n)_{n\in\mathbb{N}}$, construire la liste des $n+1$ premiers termes de u, soit $[u_0, u_1, \ldots, u_n]$. Écrire les fonctions suivantes :

1. Fonction `geo : int -> int -> int -> int list` telle que `geo q u0 n` construit les $n+1$ premiers termes de la suite géométrique de premier terme `u0` et de raison `q`.
2. Fonction `rec_seq : ('a -> 'a) -> 'a -> int -> 'a list` telle que `rec_seq f u0 n` construit les $n+1$ premiers termes de la suite définie par $u_0 =$ `u0` et $u_{k+1} =$ `f`(u_k). Vérifier avec les bons arguments qu'on peut bien retrouver la suite précédente.
3. Fonction `fibo : int -> int list` qui construit les $n+1$ premiers premiers termes de la suite de Fibonacci définie par $u_0 = 0$, $u_1 = 1$ et $u_{k+2} = u_{k+1} + u_k$.
4. Fonction `look_say : int -> int list` qui construit la suite *look and say*, c'est-à-dire la suite telle que $u_0 = 1$ et le calcul de u_{k+1} se fait en lisant à haute voix le terme u_k. Les premiers termes sont 1, 11, 21, 1211, 111221, 312211,...
5. Appliquer la fonction précédente à $n = 10$. Que remarque-t-on ? Comment expliquer ces valeurs ?
6. Pour éviter ce problème, on propose de travailler avec des listes au lieu d'entiers. Réécrire la fonction avec une signature `look_say : int list -> int list list`.