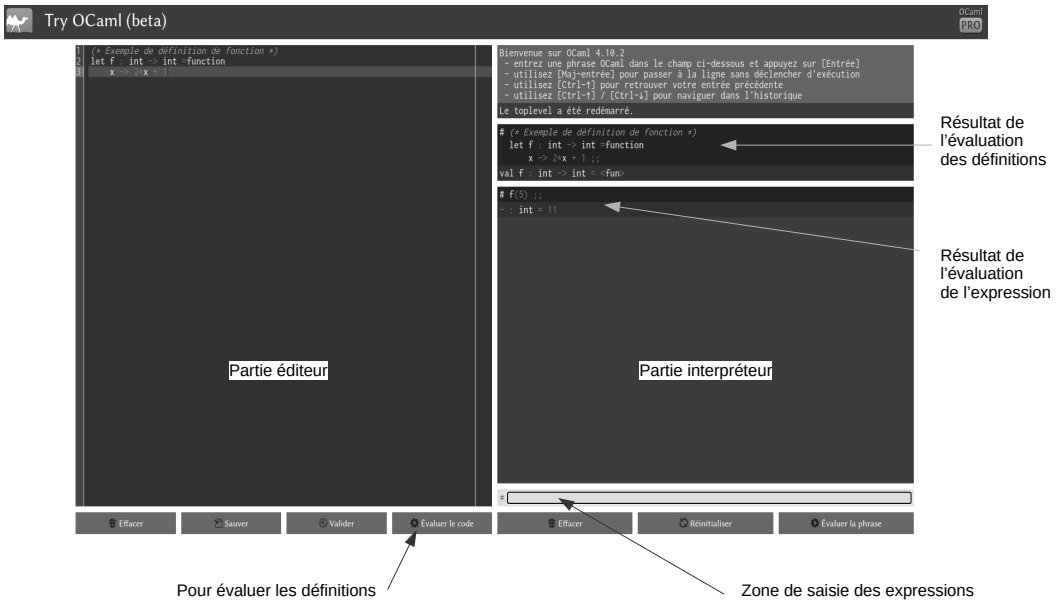
Premier TP : types et expressions OCAML

Pour le premier TP on utilisera l'environnement **TryOCaml** sur le site `ocamlpro.com`. Vous n'avez rien à installer, il vous suffit d'ouvrir un navigateur et d'accéder à la page `https://try.ocamlpro.com`. Vous avez alors à votre disposition un éditeur et à l'interpréteur OCAML :

— L'éditeur vous permet d'écrire, modifier, corriger vos définitions de fonctions, constantes, types, ... Ces définitions sont sauvegardées et peuvent être réutilisées d'une session à l'autre

— L'interpréteur vous permet de saisir et d'évaluer des expressions OCAML



Try OCaml

Expressions

Lorsqu'on saisit une expression dans l'évaluateur OCAML, celui-ci lit l'expression, l'évalue et affiche ses type et valeur. Dans la partie évaluateur de TRYOCAML évaluez :

— les expressions entières :

```
7      5 + 3      6-8      6*8      9/2      9 mod 2
```

— les expressions réelles :

```
3.2      5.2 +. 2.5      6. *. 8.      6.2 -. 1.2      9. /. 2.
```

Attention le typage en OCAML est fort : les types `int` et `float` sont disjoints. Évaluez :

```
3 + 1.2      3 +. 1.2      3.2 + 1.2      3. + 2.      3. mod 2.
```

Il existe des fonctions de conversion de type : `float_of_int, int_of_float`. Évaluez :

```
float_of_int(3)        int_of_float(1.2)        int_of_float(10.0 /. 2.5)
int_of_float(10.0)     int_of_float(2.5)
```

Évaluez les expressions booléennes :

```
true          not(true)          (true || false)
not (true && false)              not (true) && not(false)
(1>2) || (1<2)                   (12. <= 42.) <> (3 = 4)
```

Évaluez les expressions de type `string`

```
"bon"      "123"      "Ha" ^ "Ha"      "HA" ^ "!␣" ^ "HA!"      "HA" ^ "I" ^ "102I"
```

Les opérateurs de comparaison sont des fonctions polymorphes. Leurs arguments sont de type quelconque. Évaluez :

```
true = false      "true" < "false"
```

Mais les arguments doivent être de type identique. Évaluez :

```
1 = 1.0      1 < true      true = "true"
```

Évaluez les expressions conditionnelles :

```
if 12>0 then "positif" else "negatif"
if -12>0 then "positif" else "negatif"
if 12 mod 2=0 then "pair"
if 12 mod 2=0 then "pair" else 0
if 12 mod 2=0 then "pair" else "impair"
(if 3=5 then 8 else 10) + 5
```

Une expression peut être un nom qui désigne une constante de type `int, float, bool, string` ou une fonction. Dans ce cas OCAML affiche son type (sa signature) et indique que c'est une fonction. Évaluez :

```
float_of_int      int_of_float      not      abs      sqrt      floor
```

D'après ces évaluations, est-ce que les expressions `sqrt(2)` et `abs(-1.5)` sont bien typées ? Évaluez-les.
Quelle est la différence entre les fonctions `ceil` et `int_of_float` ? Évaluez les expressions :

```
int_of_float(1.5)      floor(1.5)      int_of_float(1.5) + 1      floor(1.5) + 1
```

Définitions

On peut définir de nouveaux noms. Par exemple pour associer la valeur 3.14 à `pi`, évaluez :

```
let pi = 3.14 ;;
```

À présent `pi` vaut `3.14` (on dit que `pi` est lié au flottant `3.14`).
Par exemple pour calculer le périmètre d'un cercle de rayon 2.4 évaluez :

```
2. *. pi *. 2.4 ;;
```

On peut également définir des noms localement à une expression. Évaluez :

```
let x=sqrt(2.) in x /. (x +. 1.) ;;
```

La portée de la liaison $(x, \sqrt{2})$ est alors limitée à l'expression $\frac{x}{x+1}$. Vérifiez en évaluant :

```
x ;;
```

Plusieurs définitions peuvent être imbriquées ou au même niveau. Évaluez et comparez les 2 expressions :

```
let x = 2 in let y = x+1 in x+y ;;

let x = 2 and y = x+1 in x+y ;;
```

En utilisant le nom `pi`, calculez

- la valeur de l'expression $(\sqrt{\pi} + \ln \pi) \times (\sqrt{\pi} - 2 \ln \pi)$ en appliquant qu'une fois les fonctions `sqrt` et `log`.
- la valeur de l'expression $\pi^2 + \pi^4 + \pi^6$ en n'exécutant que 3 multiplications (*.) et sans utiliser d'opérateur puissance.

Fonctions

Les définitions de fonction seront écrites dans la partie éditeur pour pouvoir plus facilement les corriger, sauvegarder, réutiliser. Dans la partie éditeur, saisissez la définition de la fonction f :

```
let f : int -> int = function
  x -> 2*x+1 ;;
```

Évaluez cette définition puis évaluez :

```
f(4) ;;
```

Un exemple de fonction avec plusieurs paramètres [1]. La traduction OCaml de la fonction $moy : \begin{array}{ccc} \mathbb{R}\times\mathbb{R} & \longrightarrow & \mathbb{R} \\ (x,y) & \longmapsto & \frac{(x+y)}{2} \end{array}$ est

```
let moy: float * float -> float = function
(x,y) -> (x+.y)/. 2. ;;
```

Évaluez cette définition puis évaluez :

```
moy(1.2,3.) ;;
```

On peut utiliser les noms définis globalement pour définir une fonction. Par exemple, si le nom `pi` est lié à la valeur 3.14, on peut définir la fonction calculant le périmètre de la fonction calculant le périmètre d'un cercle de rayon `r` par :

```
let pi=3.14;;
let perimCercle: float -> float = function
  r -> 2. *. pi *. r;;
```

Évaluez cette définition puis `perimCercle(1.2)` :

```
# perimCercle(1.2);;
- : float = 7.536
```

Si à présent on modifie la valeur de `pi` :

```
let pi=3.14159;;
```

Cette modification n'a pas d'effet sur `perimCercle` :

```
# perimCercle(1.2);;
- : float = 7.536
```

On dit que la liaison est statique : la valeur de `pi` associée à la fonction `perimCercle` est déterminée lors de la définition de la fonction et non de son application. Pour tenir compte de la nouvelle liaison, il faut réévaluer la définition de `perimCercle` :

```
# let perimCercle: float -> float = function
  r -> 2. *. pi *. r;;
val perimCercle : float -> float = <fun>
# perimCercle(1.2);;
- : float = 7.53981599999999919
```

Écrivez la définition

- d'une fonction qui calcule la surface d'un disque de rayon `r`
- d'une fonction qui calcule la surface d'un triangle rectangle de côtés `a` et `b` (côtés autres que l'hypothénuse)
- d'une fonction qui calcule le périmètre d'un triangle rectangle de côtés `a` et `b` (côtés autres que l'hypothénuse)
- de la fonction $puis8 : \begin{array}{ccc} \mathbb{N} & \longrightarrow & \mathbb{N} \\ x & \longmapsto & x^8 \end{array}$ en n'utilisant que 3 multiplications.

Suite des TP en `Learn Ocaml`

Pour les séances suivantes nous utiliserons l'outil `Learn OCaml`, qui contient des énoncés d'exercices et un environnement pour saisir, tester et évaluer vos solutions.

1. Nous verrons plus tard que cette fonction n'a en fait qu'un paramètre : le couple `(x,y)`