TP1

November 22, 2021

1 Programmation fonctionnelle

1.1 TP 1 : Programmer avec des fonctions

1.1.1 Etienne Lozes, Julien Provillard - 2021

1.2 Exercice 0 : Environnement de travail

Installez ocaml sur votre compte de l'université (ou votre machine perso) en suivant les instructions sur la page publique du cours (`http://i3s.unice.fr/~elozes/enseignement/PF`), soit en suivant les instructions pas à pas, soit en lançant le script d'installation (pour les machines de Valrose).

Attendez votre tour pour lancer le téléchargement, si tout le monde lance l'installation en même temps, le réseau sature. Comptez environ 10 minutes pour achever l'installation.

En attendant, vous pouvez commencer le TP en travaillant sous tryocaml, cf `https://try.ocamlpro.com`, cela peut vous servir à la fois de toplevel et d'IDE très rudimentaire.

Une fois l'installation achevée, lancez visual code et créez un fichier `tp1.ml` dans un répertoire dédié aux TPs de PF. Installez l'extension officielle d'Ocaml labs ("Ocaml Platform"). Vous voilà prêt à écrire votre futur projet en OCaml!

Si l'installation ne marche pas pendant le premier TP, revenez dessus plus tard (à la fin du TP, ou à un autre moment de la semaine). Pour une partie du TP 2 vous aurez besoin d'une version de Caml avec la librairie graphics, donc pensez à bien finir votre installation. De manière générale, revenez sur les exercices après la séance de TD ou de TP, les sujets sont difficiles à finir dans le temps imparti.

1.3 Exercice 1 : Fonction usuelles

Définissez les fonctions suivantes:

1. la fonction `val_abs` qui renvoie la valeur absolue d'un réel
2. la fonction `signe` telle que `signe x` renvoie 0 si `x=0`, 1 si `x > 0`, et `-1` sinon
3. la fonction `fac` qui calcule la factorielle de son argument
4. la fonction `est_diviseur` telle que `est_diviseur n d` renvoie `true` si d divise n, `false` sinon
5. la fonction `est_premier` qui indique si son argument est un nombre premier. Veillez à définir une fonction auxiliaire locale (une "sous-fonction").

1.4 Exercice 2 : Hello, world!

1. Écrivez une expression qui affiche *Hello, world!*. Quelle est la valeur de cette expression?
2. Définissez une fonction `hello` qui affiche *Hello, world!*
3. Modifiez la fonction `hello` pour qu'elle prenne en paramètres un nom et un âge et qu'elle affiche une petite phrase de présentation. Par exemple, `hello "Julien" 34` affiche *Hello, my name is Julien, and I am 34*. **Indication**: la fonction `string_of_int` pourrait vous être utile

1.5 Exercice 3 : Approximation de Pi

On représente un point du plan par un couple de nombres flottants `(x, y)`. 1. Définissez une fonction `distance` qui calcule la distance d'un point du plan à l'origine du repère. 2. On considère le cercle de rayon 1 centré à l'origine du repère et le carré [-1,1]x[-1,1] le contenant, de côtés parallèles aux axes. Quelle est la probabilité qu'un point pris au hasard dans le carré soit dans le cercle? 3. Définissez une fonction `rand_point` qui renvoie aléatoirement un point à l'intérieur du carré 4. Définissez une fonction `approche_pi n` qui tire au hasard n points à l'intérieur du carré, compte combien parmi eux sont dans le cercle, et en déduit une approximation de .

1.6 Exercice 4 : Zéro d'une fonction

On considère une fonction *f* continue sur l'intervalle *[a,b]* telle que *f(a)f(b)<0*. On sait alors qu'il existe un réel *c* tel que *f(c)=0*. On se propose de trouver ce réel à une précision donnée près. Plus précisément, on va programmer une fonction `approche0 f a b p` qui renvoie un nombre *c* tel que */f(c)/<p[*.]a,c[*, sinon on poursuit la recherche dans *]c,b[*.

1. Définissez la fonction `approche0`
2. Que calcule le code suivant?

[ ]:
```
let f x = x *. x - 2.0 in approche0 f 1.0 2.0 0.000001
```

3. Déduisez-en une nouvelle façon d'approximer le nombre .

1.7 Exercice 5 : Curryfication, évaluation partielle, fonctions anonymes (`fun x -> ...`)

1. Relisez le cours pour vous rafraichir la mémoire sur la curryfication.
2. Définissez la fonction `min` qui calcule le min de deux nombres. Son type doit être `int -> int -> int`.
3. Déduisez-en la fonction `plafonne_a n` qui renvoie une fonction `f` de type `int -> int` telle que `f m` vaut `m` plafonné à `n`. Le type de `plafonne_a` est le même que celui de `min` (et la ressemblance ne s'arrête pas là).
4. Définissez une fonction `permute_args f` qui prend en argument une fonction `f` ayant pour arguments `a` puis `b` et qui renvoie la fonction `g` ayant pour arguments `b` puis `a`. Par exemple, `permute_args plus_petit` renvoie la fonction `plus_grand`, avec les définitions suivantes:

[ ]:
```
let plus_petit a b = a < b
```

[ ]:
```
let plus_grand a b = a > b
```

Indication : le type de `permute_args` doit être `('a -> 'b -> 'c) -> 'b -> 'a -> 'c`.

1.8 Exercice 6 : Préfixe, suffixe, facteur

On dit qu'un mot u est un préfixe d'un mot v si v commence par u. Par exemple, le mot *tu* est un préfixe du mot *tuyau*. Un suffixe, inversement, est un mot qui termine un autre mot. Par exemple, *au* est un suffixe de *tuyau*. Enfin, un mot est un facteur d'un mot s'il apparait à l'intérieur du mot à une position quelconque. Par exemple *uy* est un facteur de *tuyau* à la position 1 (la position 0 est celle de la lettre t). 1. Définissez une fonction `est_facteur_position u v i` qui renvoie `true` si `u` est un facteur de `v` à partir de la position `i`, et `false` sinon. 2. Déduisez-en les fonctions `est_prefixe`, `est_suffixe`, et `est_facteur`.