Exercice 1 : Le type produit

Nous pouvons représenter un nombre complexe z comme un couple de deux réels (a, b), où a et b représentent respectivement la partie réelle et la partie imaginaire de z : $z = a + ib$. En Ocaml, on utilise le produit cartésien pour représenter un tel couple.

1. Définir les fonctions `preelle` et `pimg` retournant respectivement la partie réelle et la partie imaginaire d'un nombre complexe donné en paramètre
2. Définir la fonction `somme_c` calculant la somme de deux nombres complexes ;
3. Définir la fonction `module_c` calculant le module d'un nombre complexe (noté $|a + ib|$) ;
4. Définir la fonction `arg_c` calculant l'argument d'un nombre complexe.

NB : soit θ l'argument du nombre complexe $z = (a + ib)$. C'est-à-dire $z = |z|(\cos(\theta) + i\sin(\theta))$. Or nous savons que $\cos(\theta) = a/(|a + ib|)$ et $\sin(\theta) = b/(|a + ib|)$, ce qui permet de calculer θ (il convient cependant de distinguer les cas où la partie réelle de z (i.e. a) est respectivement négative, nulle ou positive). Dans la plupart des cas, on peut exprimer θ au moyen de l'arctan. En effet, sauf pour les cas où la partie réelle nulle, nous avons $\tan(\theta) = b/a$.

Rappel :

Somme complexe :	$(a + ib) + (c + id) = (a + c) + i(b + d)$
Module :	$\|a + ib\| = \sqrt{a^2 + b^2}$
Argument :	θ tel que $cos(\theta) = a/(\|a + ib\|)$ et $sin(\theta) = b/(\|a + ib\|)$

Exercice 2 : Les suites récurrentes

Cet exercice porte sur la traduction des suites récurrentes par des fonctions récursives en Ocaml. Prenons l'exemple de la factorielle. On peut la définir par les relations de récurrences :

$$u(0) = 1 \text{ et } u(n + 1) = (n + 1) \times u(n)$$

que l'on peut traduire en Ocaml par

```
# let rec fac n =
if n = 0 then 1 else n * fac (n - 1);;
```

`fac n` calcule $u(n)$ pour tout entier n, ce calcul se faisant de n à 0 et non de 0 à n.

```
Application:
# fac 4;;
Etapes de calculs:
Premier appel: fac 4 = if 4 = 0 then 1 else 4 * fac(4-1)==> fac 4 = 4*fac 3
Second appel: 4 * fac 3 = 3 * fac 2
Troisieme appel :             fac 2 = 2 * fac 1
Quatrieme appel :                         fac 1 = 1 * fac 0
Cinquieme appel:                                      fac 0 = 1
Ensuite on remplace fac i par sa valeur:
fac 1 = 1 * 1 = 1; fac 2 = 2 * 1 = 2 ; fac 3 = 3 * 2 = 6;fac 4 = 4 * 6 = 24.
```

Les termes de la suite sont : `fac 0, fac 1, fac 2, fac 3, fac 4`. Donc `fac n` renvoie le nième terme de la suite. C'est à dire `fac n` représente exactement la fonction $n \mapsto u_n$. La fonction `fac` peut aussi s'écrire au moyen du filtre `match with` :

```
# let rec fac n = match n with
0 -> 1
(* u(0) = 1 *)
| n -> n * fac(n - 1) (* u(n+1) = (n+1) * u(n) ; de 0 à n équivalent à *)
                      (* u(n)=n * u(n-1) ; de n à 0 *) ;;
```

Carré d'un entier

Le carré d'un entier peut être défini par :

$$x^2 = (x-1)^2 + 2 \times x - 1$$

Déduire de cette définition une relation de récurrence, et écrire en Ocaml une fonction récursive `carre x` calculant cette relation (`carre 1` vaut `0 + 1`).

Calcul du nombre de chiffres

On veut compter le nombre de chiffres d'un nombre entier positif en base 10. Exemples : 34 a 2 chiffres, et 3 a 1 chiffre. Ce calcul peut utiliser la propriété suivante :

$$\text{nbchiffres}(n) = \begin{cases} 1 & \text{si } n < 9 \\ 1 + \text{nbchiffres}(n/10) & \text{sinon} \end{cases}$$

Écrire une fonction récursive Ocaml utilisant cette propriété.

L'algorithme d'Euclide

L'algorithme d'Euclide permet de calculer le plus grand commun diviseur (pgcd) de 2 entiers. Cet algorithme se base sur la propriété suivante :

$$\text{pgcd}(a,b) = \begin{cases} a & \text{si } b = 0 \\ \text{pgcd}(b, a \mod b) & \text{sinon} \end{cases}$$

Écrire une fonction récursive Ocaml utilisant cette propriété.

Présence d'un caractère dans une chaine

On considère les fonctions suivantes :

```
# let car ch = String.get ch 0 ;; (* retourne le premier car. de ch *)
val car : string -> char = <fun>
# let cdr ch = String.sub ch 1 ((String.length ch) - 1);;
(* retourne ch sans son premier element *)
val cdr : string -> string = <fun>;
# cdr "azerty";;
- : string = "zerty";
(* on utilise cdr pour parcourir la chaîne *)
```

Écrire une fonction récursive `presence` qui prend en argument un caractère c et une chaine ch et retourne $true$ si c est dans ch sinon $false$ (utiliser les fonctions car et cdr).

Exercice 3 : Modélisation de Σ

Le calcul de la somme d'une série de la forme :

$$\Sigma_{i=n}^{i=m} f(i) = f(m) + f(m+1) + \cdots + f(n)$$

se traduit en Ocaml par la fonction récursive suivante :

```
# let rec somme f m n = if m > n then 0
                        else (f m) + (somme f (m+1) n);;
(* f argument de type fonction*)
(* m borne inférieure de l'intervalle *)
(* n borne supérieure de l'intervalle *)
```

ou encore

```
# let rec somme f m n = if m < n then 0
                 else (f n) + (somme f m (n-1));;
```

Quel est le type de `somme` ?

Donner un exemple d'appel de la fonction `somme` pour chacun des calculs suivants :

1. La somme des entiers de 1 à n.
2. La somme des entiers compris dans l'intervalle $[m, n]$.
3. La somme des cubes (x^3) des entiers compris dans l'intervalle $[m, n]$.
4. La somme des factorielles compris dans l'intervalle $[1, n]$.