Programmation Fonctionnelle 1
TDMachine 2 – Fonctions récursives sur les entiers

On vous propose d'écrire quelques fonctions récursives manipulant des entiers.
Pour chaque fonction écrite, **prenez l'habitude de chercher** son type (le vérifier *ensuite* avec le système OCaml)
Vous pourrez, sur quelques fonctions, vous préoccuper du domaine de définition du ou des arguments et utiliser (*proprement*) `failwith`.

1) Ecrire la fonction `sommeCarres` prenant en argument un entier n et calculant la somme des carrés des entiers de 1 à n.
```
# sommeCarres 5;;
- : int = 55
```

2) Ecrire la fonction `sommeFonction` prenant en argument une fonction f et un entier n, et calculant la somme des résultats de l'application de f à chaque entier de 1 à n.
```
# sommeFonction (fun x -> 2 * x) 10;;
- : int = 110
```
En déduire une nouvelle version pour `sommeCarres`.

3) Ecrire la fonction `sommeChiffres` prenant en argument un entier n et calculant la somme des chiffres contenus dans n.

4) Ecrire la fonction `sommeIteree` prenant en argument un entier n et itérant le calcul effectué par `SommeChiffres` jusqu'à ce que le résultat soit inférieur à 10.
```
sommeChriffres 4569 => 15
sommeIteree 456 => 6
```

5) Ecrire la fonction `iterer` prenant en argument une fonction unaire[1] f, un prédicat[2] unaire p et un argument x et itérant l'application de f à x jusqu'à ce que x vérifie le prédicat p, le résultat alors retourné étant x.
Exemple : prendre la moitié de x jusqu'à obtenir une valeur < 10
```
iterer (fun x -> x/2) (fun x -> x < 10) 45 => 5
```

6) Appliquer `iterer` pour redéfinir `sommeIteree`

7) Ecrire la fonction `ackermann`[3] définie sur les entiers N comme suit :
ack(m,n) = n + 1 si m = 0
ack(m,n) = ack(m-1,1) si m > 0 et n = 0
ack(m,n) = ack(m-1,ack(m,n-1)) si m > 0 et n > 0

[1] Une fonction (ou un prédicat) unaire est une fonction (ou un prédicat) à un seul argument
[2] Un prédicat retourne une valeur booléenne
[3] Attention, la fonction croît très rapidement… Ainsi ack(4,2) = 20035...6733 etcontient 19729 chiffres. Eviter de le calculer