I Rappels

1.1 Commandes utiles

- **range** est une commande python qui permet d'utiliser des boucles for. Voici quelques commandes à taper dans la console (interpréteur) pour comprendre son fonctionnement.

```
list(range(8))
list(range(1,4))
list(range(0,10,2))
```

  Donner une commande équivalente à la première, mais utilisant 2 arguments.
- **for** est le mot clé Python qui correspond aux boucle "Pour". La syntaxe est la suivante

```
for variable in range(a, b + 1):
    # les instructions suivantes vont être exécutées
    # une fois pour variable = a
    # puis une fois pour variable = a + 1 ...
    # et enfin une dernière fois pour variable = b
```

  On utilise souvent i comme nom de variable (appelée parfois variable de boucle), mais ce n'est pas une obligation. Les instructions répétées sont indentées (il y a plus d'espace pour commencer la ligne) par rapport à la commande "for". Remarquez aussi le symbole " :" en fin de ligne.
  L'utilisation la plus classique est :

```
a = # une valeur de départ
for i in range(n):
    a = # un truc en fonction de a
    # le range(n) permet de répéter n fois l'opération
```

  Voici un premier exemple.

```
for i in range(17):
    print(i**2)
```

  qui affiche à l'écran les carrés des entiers naturels ≤ 16.
- **def** est le mot clé qui permet de définir une nouvelle fonction. Voici un exemple de syntaxe

```
def f(x, y):
    return x + y
```

  On a défini une fonction nommée f qui prend deux arguments (ou paramètres) et dont la valeur de retour est la somme des arguments.
- La commande "n % k" calcule le reste dans la division de n par k et permet donc de savoir si n est un multiple de k.
- l'égalité entre quantité se teste avec la commande ==. Le simple "=" est réservé à l'affectation.

1.2 Activités

Nommage

Donner un nom correct à la fonction suivante :

```
def truc(a, n):
    res = 0
    for i in range(n):
        res = res + a
    return res
```

On pourra la tester dans la console éventuellement.

A vous de jouer

A chaque fois que vous pensez en avoir fini avec une fonction, testez là ! Ceci vaut pour toute l'année, voire pour toute la vie... Les fonctions suivantes ont tout à fait leurs places dans un fichier nommé *TD2_fonctions.py*

1. Créer une fonction **basel(n)** qui calcule, pour un entier n donné, la somme $\sum_{k=1}^{n} \frac{1}{k^2}$.
   Comparer la valeur obtenue pour n un peu grand (10^4 par exemple) avec $\frac{\pi^2}{6}$.
2. Créer une fonction **diviseurs(n)** qui affiche tous les diviseurs de l'entier n. Pour afficher une valeur, on utilise

```
print(valeur)
```

3. Créer une fonction **puissance(a, n)** qui calcule et retourne a^n.
4. Créer une fonction **est_premier(n)** qui renvoie une valeur booléenne indiquant si n est un nombre premier.

II Textes

2.1 Rappel et manipulations de base

Création de texte

Pour créer des données de texte (que l'on appelle chaînes de caractères, chaque lettre étant un caractère) en python :

```
t1 = ''Ceci est un texte, on remarque la présence de guillemets.''
t2 = ''123456''  # ceci aussi est un texte
# Ne pas faire de copier-coller directement dans la console.
```

Opérations possibles

Tester les commande suivantes dans la console, en ayant d'abord créé deux variables t1 et t2 contenant du texte (pourquoi pas les exemples précédents)

```
t1[0],t1[1]
```

De manière générale, quel est l'effet de `t1[i]` ? Quelles sont les valeurs de `i` admissibles ?

```
len(t1)
```

```
t1 + t2, t1 + ''3''
```

```
''a''*7
```

2.2 Premiers algorithmes

Exercice

1. Créer une fonction `occurences(txt, c)` qui retourne le nombre de fois où le caractère `c` apparaît dans la chaîne `txt`.
2. Créer une fonction `demi_escalier(n)` telle que :

```
>>> demi_escalier(4)
#
##
###
####
```

On demande d'afficher cet escalier, pas de retourner la chaîne de caractères.

3. Même question, mais avec un escalier pour monter cette fois (dans le sens de lecture, bien entendu) !
4. Même question pour la fonction `escalier(hauteur, largeur)`

```
>>> escalier(3,5)
  #####
 #######
#########
```

Plus difficile

1. Ecrire une fonction qui teste si les caractères d'une chaîne sont 2 à 2 distincts.
2. Ecrire une fonction `est_sous_chaine_pos(txt, ch, k)` qui teste si la chaîne `ch` est une sous-chaîne de `txt` à partir de la position k, ou encore, en notant m la longueur de `ch`, $\forall i \in [0, m-1]$ $txt[k+i] == ch[i]$. En déduire une fonction qui teste si une chaîne `ch` est une sous-chaîne de `txt`.
3. En utilisant la fonction `chr` (et la fonction `ord` auparavant dans la console), créer la fonction `caractere_majoritaire(txt)` qui retourne le caractère apparaissant le plus de fois dans la chaîne `txt` (le premier trouvé en cas d'égalité). On considère que toutes nos chaînes sont composées de lettres minuscules, sans accent ni caractères spéciaux. On ne comptera pas les espaces.