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))
```

- **for** est le mot clé Python qui correspond aux boucle "Pour". La syntaxe est la suivante

```
for variable in range(n):
    # les instructions suivantes vont être exécutées
    # une fois pour variable = 0
    # puis une fois pour variable = 2 ...
    # et enfin une dernière fois pour variable = n - 1
```

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.
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):
    return x + 4
```

On a défini une fonction nommée f qui prend un seul argument (ou paramètre) et dont la valeur de retour est cet argument + 4. Il est tout à fait possible de définir une fonction qui prend plusieurs 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, b):
    return a + b
```

Pour utiliser cette fonction, on tapera **truc(4, n)** par exemple, si la variable n a déjà une valeur.

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.
3. Créer une fonction **puissance(a, n)** qui calcule et retourne a^n

II Listes

Il est recommandé de changer de feuille de script à ce stade. *TD2_listes.py* semble être un nom convenable

2.1 Manipuler des listes

L1, L2, L3 sont des listes Python après ces commandes

```
L1 = []
L2 = [1, 4, True, math.pi]
L3 = list(range(12))
```

La longueur d'une liste est donnée par la fonction **len** :

```
[len(L1), len(L2), len(L3)]
```

Les éléments des listes Python sont numérotés de 0 à longueur - 1.

```
L2[2] # accès à l'élément d'indice 2, c'est à dire au troisième élément de L

for i in range(len(L2)):
    print(L2[i])
```

Opérations sur les listes

Tester les commandes suivantes et essayer de deviner leurs effets

```
L2 + L2
L2 + L3
L2 * 3
[0] * 15
```

Exercice

Créer une fonction **somme(L)** qui prend une liste comme argument et retourne la somme des éléments de L. Comparer à **sum(L)**

Exercice

Créer une fonction **moyenne(L)** qui retourne la moyenne arithmétique des éléments de L.

Exercice

Créer une fonction **sup_moyenne(L)** qui prend une liste L et renvoie le nombre d'éléments de L qui sont supérieurs (ou égaux) à la moyenne de L.

2.2 Construction pas à pas

```
L = [] # une liste vide. L = list() a le même effet
L.append(-1)
L.append(2)
L.append(42)
```

Quel est l'effet de la commande **L.append(element)** ?

Exercice

Créer une fonction **subdiv(a, b, n)** qui étant donné deux nombres $a < b$ et un entier naturel non nul n renvoie une liste de $n+1$ nombres équirépartis dans le segment $[a, b]$, le premier étant a et le dernier b.
Par exemple **subdiv(0, 1, 4)** renvoie $[0, 0.25, 0.5, 0.75, 1]$.
Pour créer cette fonction, une feuille de papier et un crayon sont indispensables pour trouver la méthode générale de construction des nombres demandés.

Interlude d'import

Pour tester nos fonctions et les utiliser, nous aurons besoin d'un peu plus d'outils python. Pour y avoir accès nous allons *importer* des modules, c'est à dire les charger dans la console.

```
import numpy as np
import matplotlib.pyplot as plt
```

Ces lignes chargent de nouvelles fonctions dans la console. Le module numpy contient beaucoup de fonctions que nous utiliserons au cours de l'année.
On peut y accéder par **np.nom_de_la_fonction** (le *as* de la commande d'import cré un alias, c'est à dire que np est un raccourci pour numpy)

```
np.linspace(0, 1, 5)
```

Exercice

Créer une fonction **applique_sin(X)** qui prend une liste X comme argument et retourne la liste $[\sin(X[0]), \sin(X[1]), \dots]$ qui est de même longueur que X.
Tester votre fonction et tapez dans la console :

```
X = np.linspace(0, 2*pi, 51)
Y = applique_sin(X)
plot(X, Y)
```

La fonction **plt.plot** prend deux listes comme arguments : une liste dabscisses et une liste d'ordonnées de mêmes longueurs, et relie par des droites les points dont les coordonnées sont ainsi décrites.

Exercice

Tracer exp, cos, $x \mapsto e^{-x^2}$, $x \mapsto \frac{1}{1+x^2}$ entre -2 et 2 à l'aide de 150 segments (combien de points ?). Pour cela on pourra créer une fonction **applique(f, X)** qui applique la fonction

f à X sur le même modèle que précédemment. On aura donc **applique(math.sin, X) == applique_sin(X)**.
Comment tracer les éventuelles fonctions réciproques ?

Bonus

- Créer la fonction **renverse** qui prend une liste comme argument et retourne la liste ayant les même éléments mais dans l'ordre opposé. Le premier devient le dernier et ainsi de suite.
- Créer une fonction qui prend un argument n entier naturel et retourne la liste de tous les nombres premiers $\leq n$.
- Créer une fonction qui prend un entier n comme argument et renvoie son écriture binaire sous forme de liste.
- Créer la fonction réciproque de la précédente.