TP 10 : Les fonctions

⚠
- Penser à ouvrir un nouveau fichier pour chaque exercice et à l'enregister immédiatement en lui donnent le nom proposé dans l'énoncé.
- On documentera chaque fonction écrite en expliquant en une ligne ce que représentent les paramètres et ce qu'elle calcule (comme on l'a vu en cours).

Exercice 1 *Somme des carrés* / ***somme_carres.py***. Le programme suivant calcule et affiche la valeur de la somme

$$\sum_{k=1}^{10} k^2$$

```
s=0
for k in range(1,11):
    s=s+k**2
print(s)
```

(a) Écrire et tester ce programme. Pourquoi **range(1,11)** et pas **range(1,10)** ou **range(10)** ? Pourquoi pas **range(11)** ?
(b) Modifier ce programme afin d'écrire une fonction **somme_carres(n)** qui calcule la somme $\sum_{k=1}^{n} k^2$. Tester cette fonction avec quelques valeurs simples de n.
(c) Écrire une nouvelle fonction **produit_carres(n)** qui calcule le produit $\prod_{k=1}^{n} k^2$. Tester cette fonction avec quelques valeurs simples de n.

Exercice 2 *Paradoxe des anniversaires* / ***anniversaires.py***. La probabilité p_n qu'au moins deux étudiants d'une classe de n étudiants aient leur anniversaire le même jour de l'année est donnée par la formule :

$$p_n = 1 - \prod_{k=1}^{n-1} \left(1 - \frac{k}{365}\right)$$

(a) Écrire une fonction **probabilite(n)** qui calcule p_n.
(b) Calculer p_{35}.

Exercice 3 *Sommes doubles* / ***sommes_doubles.py***. Le programme suivant calcule et affiche la valeur de la somme

$$\sum_{1 \leqslant i,j \leqslant 10} ij$$

```
s=0
for i in range(1,11):
    for j in range(1,11):
        s=s+i*j
print(s)
```

Pour $n \in \mathbb{N}^*$, on définit les sommes $S_n = \sum_{1 \leqslant i,j \leqslant n} \min(i,j)$ et $T_n = \sum_{1 \leqslant i \leqslant j \leqslant n} \frac{i}{j+1}$.

(a) Écrire deux fonctions **S(n)** et **T(n)** qui calculent respectivement S_n et T_n (on pourra utiliser la fonction **min** qui existe dans PYTHON).
(b) Tester ces fonctions en vérifiant les égalités $T_n = n(n+1)/4$ et $S_n = n(n+1)(2n+1)/6$ pour quelques valeurs de n.

◇ Lorsque l'on souhaite programmer la résolution d'un problème complexe, il peut être intéressant de le découper en tâches plus simples qui seront traitées par des fonctions intermédiaires. On utilise ensuite ces fonctions pour effectuer la résolution du problème de départ. Dans l'exercice suivant, on illustre ce principe en programmant une méthode de résolution approchée d'équations différentielles dans un cadre assez général (c'est la méthode d'Euler que l'on a déjà rencontrée).

Exercice 4 ***euler.py***. On considère une équation différentielle avec condition initiale écrite sous sous la forme générale :

$$\begin{cases} y' = f(t,y) \\ y(t_0) = y_0 \end{cases}$$

Par exemple pour traiter l'équation différentielle $y' + ty = 1$ avec la condition initiale $y(0) = 1$, on prendrait $f(t,y) = 1 - ty$, $t_0 = 0$ et $y_0 = 1$.

On souhaite écrire une fonction **`euler(f,t0,tfinal,y0,N)`** qui construit et renvoie les listes **`t`** $= [t_0, \dots, t_N]$ et **`y`** $= [y_0, \dots, y_N]$ correspondant à la résolution approchée de cette équation différentielle sur l'intervalle $[t_0, t_{\text{final}}]$.

(a) Écrire une fonction **`pas_de_temps(t0,tfinal,N)`** qui renvoie le pas de temps Δ_t obtenu en découpant l'intervalle $[t_0, t_{\text{final}}]$ en N intervalles de même taille.

(b) Écrire une fonction **`liste_instants(t0,N,Delta_t)`** qui construit et renvoie la liste $[t_0, \dots, t_N]$ où pour un entier i tel que $0 \leqslant i \leqslant N$, on a posé $t_i = i\Delta_t$.

(c) Écrire une fonction **`liste_constante(y0,N)`** qui renvoie la liste de taille $N+1$ dont tous les éléments sont égaux à **`y0`**.

(d) Avec les fonctions précédentes, écrire la fonction **`euler(f,t0,tfinal,y0,N)`**.

(e) La fonction $t \mapsto \exp(-t) + 1$ est solution de l'équation différentielle $y' + y = 1$. Utiliser ceci pour tester la fonction **`euler`**.

Exercice 5 *Approximation de fonctions* / ***approx_exp.py***. Pour $x \in \mathbb{R}$ et $N \in \mathbb{N}$, on pose :

$$A_{\exp}(N,x) = \sum_{n=0}^{N} \frac{x^n}{n!}$$

(a) Écrire une fonction **`Aexp(N,x)`** qui calcule $A_{\exp}(N,x)$. Vérifier que **`Aexp(10,1)`** est une valeur approchée de $\exp(1)$.

(b) Représenter les fonctions $x \mapsto \mathrm{e}^x$ et $x \mapsto A_{\exp}(3,x)$ sur l'intervalle $[-2,2]$ sur le même dessin. On donne pour cela les commandes :

```
from math import *
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-2,2,100)
plt.plot(x,[exp(xi) for xi in x])
plt.plot(x,[Aexp(3,xi) for xi in x])
plt.show()
```

(c) Représenter les fonctions $x \mapsto \mathrm{e}^x$ et $x \mapsto A_{\exp}(10,x)$ sur l'intervalle $[-2,2]$ sur le même dessin.

Corrections

Ex 1.

```
def somme_carres(n):
    """
    Calcule 1^2+2^2+...+n^2
    avec n entier, n>=1
    """
    s=0
    for k in range(1,n+1):
        s=s+k**2
    return s
print(somme_carres(3))
```

```
14
```

```
def produit_carres(n):
    """
    Calcule 1^2*2^2*...*n^2
    avec n entier, n>=1
    """
    s=1
    for k in range(1,n+1):
        s=s*k**2
    return s
print(produit_carres(3))
```

```
36
```

Ex 2.

```
def probabilite(n):
    """
    Calcule la probabilité que parmi n personnes,
    deux aient leur anniversaire le même jour
    n entier, n>=1 (voire n=0)
    """
    p=1
    for k in range(1,n): # Dans le produit, k va de 1 jusqu'à n-1
        p=p*(1-k/365)
    # p contient le produit (1-1/365)*...*(1-(n-1)/365)
    return 1-p
print(probabilite(35))
```

```
0.8143832388747152
```

Ex 3.

```
def S(n):
    """
    Calcule la somme des min(i,j) pour 1<=i,j<=n
    n entier, n>=1
    """
    s=0
    for i in range(1,n+1):
        for j in range(1,n+1):
            s=s+min(i,j)
    return s
print(S(10), 10*(10+1)*(2*10+1)/6)
```

```
385 385.0
```

```
def T(n):
    """
    Calcule la somme des i/(j+1) pour 1<=i<=j<=n
    n entier, n>=1
    """
    s=0
    for i in range(1,n+1):
        for j in range(i,n+1):
            s=s+i/(j+1)
    return s
print(T(10),10*(10+1)/4)
```

```
27.500000000000004 27.5
```

Deuxième méthode pour **`T(n)`** :

```
def T(n):
    """
    Calcule la somme des i/(j+1) pour 1<=i<=j<=n
    n entier, n>=1
    """
    s=0
    for i in range(1,n+1):
        for j in range(1,n+1):
            if i<=j:
                s=s+i/(j+1)
    return s
print(T(10),10*(10+1)/4)
```

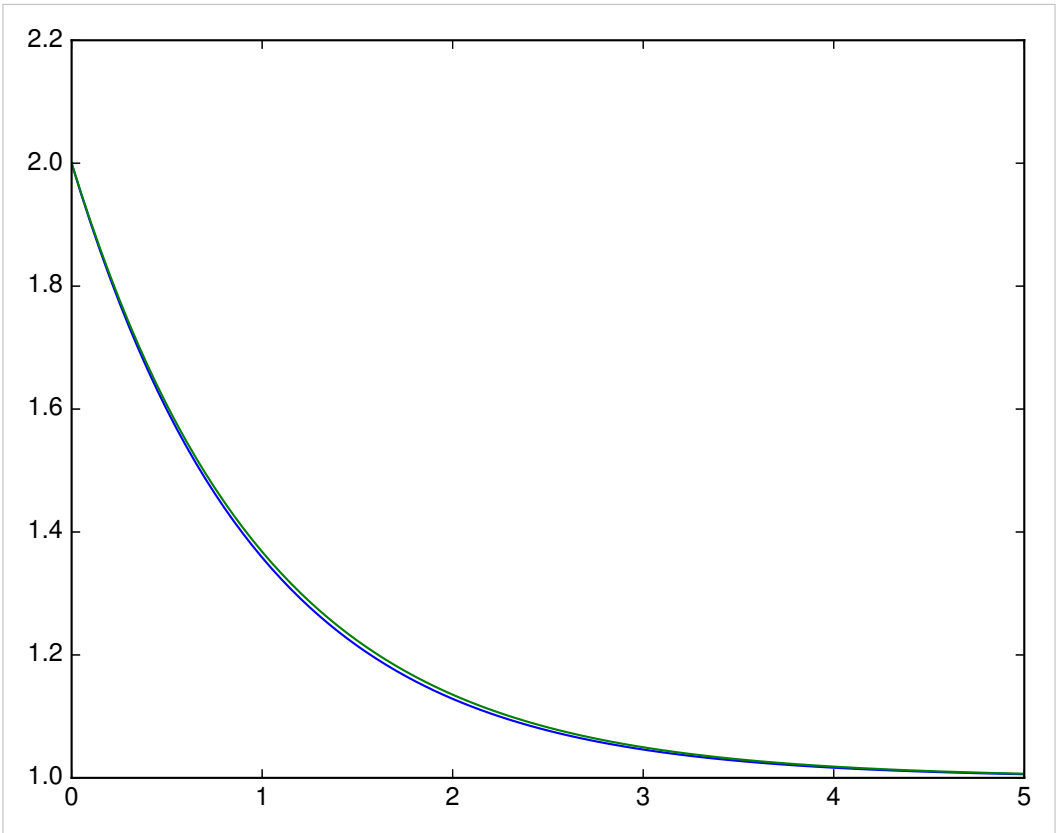
```
27.500000000000004 27.5
```

Ex 4.

```
def pas_de_temps(t0,tfinal,N):
    """
    N est un entier >0
    t0 et tfinal de type float avec tfinal>t0
    """
    return (tfinal-t0)/N

def liste_instants(t0,N,Delta_t):
    """
    N est un entier, N>0
    t0 et Delta_t de type float
    """
    t = []
    for i in range(0,N+1):
        t.append(i*Delta_t)
    return t

def liste_constante(y0,N):
    """
    N est un entier >0
    """
    y = []
    for i in range(0,N+1):
        y.append(y0)
    return y

def euler(f,t0,tfinal,y0,N):
    """
    t0, tfinal, y0 de type float avec tfinal>t0
    N entier >0
    f est une fonction de 2 variables
    """
    Delta_t = pas_de_temps(t0,tfinal,N)
    t = liste_instants(t0,N,Delta_t)
    y = liste_constante(y0,N)
    for i in range(0,N):
        y[i+1] = y[i]+Delta_t*f(t[i],y[i])
    return (t,y)
```

Exemple :

```
from math import *
import matplotlib.pyplot as plt

def f(t,y):
    return 1-y

(t,y) = euler(f,0,5,2,100)
plt.plot(t,y)
plt.plot(t,[exp(-ti)+1 for ti in t])
```

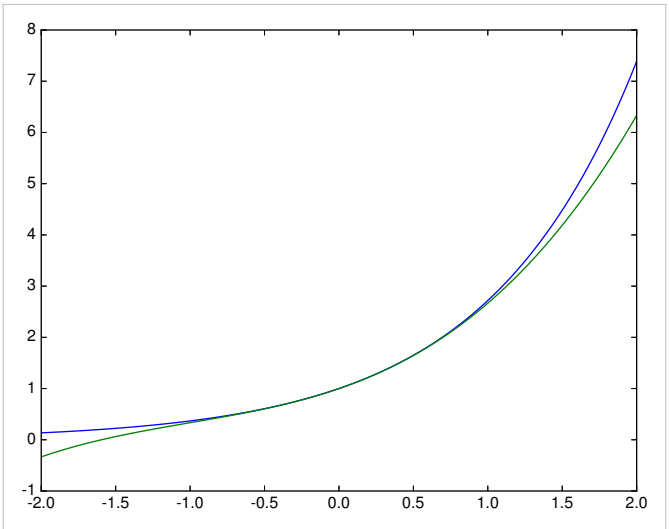


Ex 5.

```
from math import *
def Aexp(N,x):
    """
    Calcule x^0/0!+x^1/1!+...+x^N/N!
    N entier, N>=0
    """
    s=0
    for n in range(N+1):
        s=s+x**n/factorial(n)
    return s
print(Aexp(10,1))
```

```
2.7182818011463845
```

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-2,2,100)
plt.plot(x,[exp(xi) for xi in x])
plt.plot(x,[Aexp(3,xi) for xi in x])
```




```
x=np.linspace(-2,2,100)
plt.plot(x,[exp(xi) for xi in x])
plt.plot(x,[Aexp(10,xi) for xi in x])
```