T.P. III - Modules

Python est un des langages les plus utilisés de nos jours. De nombreuses fonctions mathématiques et algorithmes classiques ont déjà été écrites dans ce langage et sont disponibles pour tous les utilisateurs. Si Python les chargeait toutes lors de son lancement, le TP ne serait pas encore commencé ! Pour pouvoir utiliser une fonction utilisée dans un module, il faut donc *demander* à Python de charger le module.

.1 - Module `numpy`

Par exemple, le module `numpy` permet d'utiliser de nombreuses fonctions mathématiques.

```
import numpy

print(numpy.log(3))
print(numpy.exp(1.14))
```

qui affiche

```
1.0986122886681096
3.1267683651861553
```

Vous remarquerez la syntaxe `numpy.log` qui *dit* à Python qu'il faut aller chercher la fonction `log` du module `numpy`. Cette syntaxe est lourde et il est préférable de donner un *surnom* au module `numpy` de manière à écrire moins de texte. On le surnommera généralement `np`. On écrira alors :

```
import numpy as np

print(np.log(3))
print(np.exp(1.14))
```

qui affiche

```
1.0986122886681096
3.1267683651861553
```

Le module `numpy` permet également de créer des listes de nombres avec un espace régulier :

- `arange(a, b, pas)` crée la liste des nombres compris entre `a` (inclus) et `b` (exclus), avec un espacement constant égal à `pas`.
- `linspace(a, b, n)` crée une liste de `n` nombres compris entre `a` (inclus) et `b` (inclus), avec un espace constant.

```
print(np.arange(1, 2, 0.1))

print(np.linspace(1, 2, 11))

[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9]
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```

.2 - Rendus graphiques

Les rendus graphiques en Python sont possibles à l'aide du module `matplotlib.pyplot`, importé ici avec le *surnom* `plt`. La fonction

- `plt.figure()` permet de créer un nouveau graphique,
- `plt.plot(abscisses, ordonnees)` permet de tracer les points dont la liste des abscisses est `abscisses` et la liste des ordonnées est `ordonnees`,
- `plt.show()`. permet d'afficher le graphique.

Pour tracer le graphe de la fonction exponentielle sur l'intervalle $[-5, 5]$, on utilisera ainsi :

```
import matplotlib.pyplot as plt
X = np.arange(-5, 5.1, 0.1)
Y = np.exp(X)

plt.figure()
plt.plot(X, Y)
plt.show()
```

Pour tracer un graphe, nous avons donc besoin de la liste des ordonnées, c'est-à-dire de la liste des images des abscisses par une fonction. Dans l'exemple précédent, `Y = np.exp(X)` permet de stocker dans `Y` la liste des images des éléments de `X` par la fonction `np`. Pour construire l'image des éléments de la liste `X` par la fonction `f`, il existe plusieurs solutions :

* si `f` est une fonction `numpy` *simple* (définie sans utiliser de conditionnelle), on peut écrire comme précédemment `Y = f(X)`.

```
def f(x):
    return x**2 * np.exp(x)

X = np.arange(1, 3, 0.5)
Y = f(X)
print(Y)
```

qui affiche

```
[ 2.71828183 10.08380041 29.5562244  76.14058725]
```

* sinon, on peut utiliser la notion de *liste par compréhension* : `Y = [f(x) for x in X]`. Ceci ce lit *Y est la liste des éléments* `f(x)` *lorsque* `x` *parcourt* `X`.

```
def f(x):
    if x < 2:
        return x
    else:
        return x + 1

X = np.arange(1, 3, 0.5)
Y = [f(x) for x in X]
print(Y)
```

qui affiche

```
[1.0, 1.5, 3.0, 3.5]
```

* enfin, on peut utiliser un initialiser `Y` avec un vecteur rempli de zéros à l'aide de la fonction `np.zeros` puis remplacer chacun des éléments par la *bonne* valeur :

```
def f(x):
    if x < 2:
        return x
    else:
        return x + 1

X = np.arange(1, 3, 0.5)
Y = np.zeros((len(X), 1))
for i in range(0, len(Y)):
    Y[i] = f(X[i])

print(Y)
```

qui affiche un vecteur colonne

```
[[1. ]
 [1.5]
 [3. ]
 [3.5]]
```

La fonction `len` renvoie la longueur d'un vecteur.

I - Exercices

Exercice 1. (Suite arithmético-géométrique) Soit (u_n) la suite définie par $u_0 = 1$ et

$$\forall\, n \in \mathbb{N},\ u_{n+1} = \frac{1}{2}u_n + 3.$$

1. Compléter le code suivant qui permet de calculer le n-ième terme de la suite puis d'afficher les valeurs de u_n pour $0 \leqslant n \leqslant 20$. Quelle conjecture pouvez vous effectuer sur le comportement de la suite ?

```
def u(n):
    u = 1
    for i in range(1, ...):
        u = u / 2 + ...
    return u

X = np.arange(0, ..., 1)
Y = [u(n) for n in X]

plt.figure()
plt.plot(..., ..., 'o')
plt.show()
```

2. On souhaite déterminer puis afficher le plus petit entier naturel n_0 tel que $u_{n_0} \geqslant 5.5$. Compléter le code suivant :

```
n = 0
u = 1
while u <= ...:
    u = ...
    n = ...

print(...)
```

Exercice 2. (Série géométrique) Soit $q \in \mathbb{R}$. On souhaite étudier les suites définies par $v_0 = 1$ et, pour tout n entier naturel, $v_{n+1} = qv_n$.

1. a) Compléter le code suivant qui permet définir une fonction `suite_geom` tel que l'appel `suite_geom(n, q)` renvoie le terme v_n :

```
def suite_geom(n, q):
    v = 1
    for i in range(..., ...):
        v = ...
    return ...
```

b) Compléter le code qui permet d'afficher les valeurs de v_n lorsque :

* $n = 10$ et $q = 0,1$.
* $n = 100$ et $q = 2$.
* $n = 110$ et $q = 0,5$.

```
print("n = 10, q = 0.1", v(..., ...))
print("n = 100, q = 2", v(..., ...))
print("n = 110, q = 0.5", v(..., ...))
```

On souhaite maintenant cacluler les termes successifs de la suite $(w_n)_{n\in\mathbb{N}}$ définie par $w_0 = 0$ et $w_n = \sum_{k=1}^{n} v_k$ pour tout entier naturel n.

2. Compléter le code de la fonction suivante qui renvoie le terme w_n :

```
def serie_geom(n, q):
    s = ...
    for i in range(..., ...):
        s = s + ...
    return s
```

a) On souhaite afficher les 100 premiers termes de cette suite pour $q = 0.1$. Compléter le code suivant :

```
import matplotlib.pyplot as plt
n = ...
s = [serie_geom(...) for n in range(..., ...)]

plt.plot(range(...), ...)
...
```

b) Repréndre la question précédente avec $q = 1.2$.