TP de Python numéro 2 - Fonctions

*Semaine du **15 Septembre 2022**.*

À partir de maintenant, merci de vous créer un dossier TP dans lequel vous aurez (au moins) un fichier python par TP (par exemple pour celui-ci, TP2.py).

I. Le module `numpy`

Commentaires en programmation

Tout langage de programmation un peu abouti permet de faire des commentaires. Il s'agit juste de vous permettre de commenter votre code pour :

- le rendre plus lisible ou expliquer le rôle d'une ligne,
- ou rendre tout un bout de code inactif ce qui est souhaitable dans plusieurs situations (par exemple, si on veut débugger son code).

En `Python`, le symbole # indique un début de commentaire sur une ligne, et les triples guillemets `"""` délimitent une zone commentée.
Copier coller le code ci-dessous et le lancer, que fait la fonction `np.sqrt` ?

```
import numpy as np #Le bloc commenté ci-dessous explique cette ligne
'''
Cette ligne importe le module numpy qui rajoute des fonctions mathématiques à Python,
comme la fonction sqrt.
Testons ce qu'elle fait à l'aide du code ci-dessous
'''
for i in range(1,20):
    print("pour i valant ", i ," np.sqrt(i) renvoie ", np.sqrt(i))
```

Lors de vos TPs, vous utiliserez les triples guillemets pour garder une trace de vos travaux, sans faire relire à `Python` le travail des exercices précédents à chaque fois.

1. Mise en jambes

Commençons par appliquer ce qui a été vu au premier TP. `Python` vous permet déjà de faire des opérations mathématiques de base, articulées avec les tests `if` et les boucles comme `for`.

Exercice 1. Soit $(u_n)_{n\in\mathbb{N}}$ la suite réelle donnée par $u_0 = 0$ et pour tout entier n : $u_{n+1} = 3u_n + 2^n$. À l'aide d'une boucle `for` , calculer u_{10}, u_{45} et u_{100}.

Exercice 2. Calculer $3^{10} - 2^{10}$, puis $3^{45} - 2^{45}$ et $3^{100} - 2^{100}$. Conjecturer une relation vis à vis de la suite précédente et la démontrer brièvement sur papier.

Sommes et produits

On rappelle les notations suivantes, pour toute suite réelle $(u_n)_{n\in\mathbb{N}}$. Pour tout entier n :

$$\sum_{k=0}^{n} u_k = u_0 + u_1 + u_2 + \ldots + u_n$$

$$\prod_{k=0}^{n} u_k = u_0 u_1 u_2 \ldots u_n.$$

En particulier, on utilisera ici $\displaystyle\sum_{k=0}^{n+1} u_k = \sum_{k=0}^{n} u_k + u_{n+1}$ et la formule analogue pour le produit.

Vous devez répondre à l'exercice suivant sans utiliser l'ordinateur.

Exercice 3. Quelle est la somme affichée par le code ci-dessous ?

```
S=0  #Avant la boucle, S=0
for i in range(0,10):
        S=S+i
print(S)
```

Exercice 4.

- Calculer la valeur des sommes et produits suivants (pour $n = 10, 50, 100$ si la formule dépends de n). Gardez une trace de votre travail en travaillant dans l'éditeur de code.

1. $\displaystyle\sum_{i=0}^{n} i^2$
2. $\displaystyle\sum_{i=4}^{57} \left(-\frac{1}{2}\right)^i$
3. $\displaystyle\prod_{k=0}^{n} k$
4. $\displaystyle\prod_{k=1}^{n} \frac{i}{i+1}$

- Corriger l'énoncé pour rendre la question 3 intéressante et adapter votre code.
- Conjecturer et essayer de démontrer brièvement une égalité liée au dernier produit 4. Pour la conjecture, calculer ce produit pour $n = 2, 5, 10$.

Un exercice faisant intervenir un test conditionnel :

Exercice 5. On s'intéresse aux restes dans la division euclidienne par 7 des nombres de la forme $8^n + 7$, pour n entier. On va utiliser `Python` pour faire des tests et conjecturer.

1. Donner ces restes pour n variant de 0 à 10.
2. Émettre une conjecture portant sur ces restes.
3. Vérifier votre conjecture un peu plus sérieusement, en codant un programme qui teste votre conjecture pour n allant jusqu'à 1000, et affiche un message adéquat s'il trouve un contre exemple (utiliser `if`).

Exercice (à la maison) : démontrer votre conjecture par récurrence.

2. Le module `numpy`

Remarque. Dans le TP de la semaine dernière, nous avons vu des fonctionnalités de base de `Python`, auxquelles il convient d'ajouter la fonction valeur absolue `abs` et la notation scientifique : tapez `1.23e4` et `Python` interprétera `1.23*10**4`.

Si `Python` fournit un cadre de travail dans lequel on peut utiliser les opérations de base des mathématiques, on souhaite aussi s'en servir pour des objets plus compliqués. Par exemple, la fonction exponentielle n'est pas, par défaut, implémentée en `Python`. Mais ce langage est très utilisé en sciences, il serait surprenant que personne n'ait déjà eu envie d'utiliser l'exponentielle...

Les **modules** d'un langage de programmation (ici `Python`) sont des fonctionnalités supplémentaires. Ils vous permettent d'enrichir votre boite à outils. Chaque module vient avec sa documentation, qui explique comment est codée chaque fonction ajoutée. Par exemple, pour implémenter l'exponentielle, on pourrait utiliser la formule (voir semestre 2) suivante :

$$\mathrm{e}^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

valable pour tout réel x, et pour adapter cela à l'ordinateur - qui ne calcule jamais de sommes infinies - on pourrait apprendre à `Python` à utiliser l'approximation :

$$e^1 \simeq \sum_{k=1}^{100} \frac{1}{k!}.$$

Le module `numpy` (*numbers for Python*) est l'un des modules les plus utilisé pour faire ce genre de considérations mathématiques.

Pour s'en servir dans son code, il faut :

1. Le télécharger (avec la commande `pip install numpy` ou `install numpy` selon votre version de `Python`). Normalement, c'est déjà fait (il suffit de le faire une fois sur votre machine).
2. L'appeler au début de votre programme, avec la commande `import numpy`.
3. Les fonctions de `numpy` sont alors disponibles, mais il faut toujours dire à `Python` d'aller les chercher dans `numpy` en écrivant par exemple `numpy.exp` pour la fonction exponentielle `exp` ajoutée par numpy.

En fait, on utilise souvent un raccourci pour gagner du temps.

Importer un module et le renommer

Pour importer le module `numpy`, après l'avoir bien téléchargé, on utilise en début de programme la ligne suivante :
`import numpy as np`
où:

- `import` demande à `Python` d'importer un module,
- `numpy` est le nom du module qu'on veut importer,
- `as` demande à `Python` de renommer ce module,
- `np` est le nom (personnalisable) qu'on décide de donner au module.

Après cette ligne, la fonction exponentielle `exp` de `numpy` est accessible en tapant :
`np.exp`

Exercice 6. Comprendre, en testant sur l'invite de commande, ce que donnent les commandes suivantes de `numpy` (le noter ici):

1. `pi`
2. `e`
3. `sqrt`
4. `exp`
5. `log`
6. `log10`
7. `floor`
8. `ceil`

Exercice 7. Coder un programme demandant à l'utilisateur d'entrer un nombre réel x, et qui :

1. Affiche "Entrez un nombre strictement positif" si le nombre donné est négatif.
2. Sinon, le programme doit afficher (de manière lisible) la racine carrée et le logarithme népérien de x, puis une phrase en français donnant la plus grande des deux quantités précédentes.

II. Les fonctions

1. La notion de "renvoi" et de fonction

La notion de renvoi est importante pour comprendre ce qui se passe lorsque `Python` procède à la lecture de votre code. Considérons par exemple que pendant cette lecture, au cours d'une ligne, `Python` doive effectuer un calcul comme `2+2`. Il va alors :

- interpréter 2+2 comme un calcul qu'il doit effectuer,
- effectuer ce calcul et noter le résultat, 4, puis
- continuer à lire sa ligne *comme si le résultat 4 avait été écrit à la place de 2+2.*

On dit alors que le calcul $2+2$ **renvoie** la valeur 4. La valeur renvoyée par une opération va "prendre sa place" une fois l'opération effectuée.

Un autre exemple : à la lecture de la ligne
`x=2+x`
`Python` va successivement:

- Commencer par comprendre `x=` comme une affectation de variable. Pour continuer, il sait qu'il attend une valeur (à mettre dans la variable `x`).
- Il continue sa lecture : il lit donc l'entier `2` suivi du signe `+`. Il sait alors qu'il va devoir opérer une addition, et doit continuer à lire pour savoir quelle somme effectuer.
- Il lit ensuite la lettre `x`. Il l'interprète ici comme un nom de variable. Il va donc chercher dans sa mémoire...
  - si la variable `x` n'est pas encore attribuée, `Python` s'arrête et affiche une erreur ("Qui est `x` ?").
  - Si la variable `x` est bien attribuée, il va chercher sa valeur v, et le symbole `x` lu ici va **renvoyer** cette valeur.
- Dans le second cas, où `x` a renvoyé une valeur v, `Python` doit donc effectuer le calcul $2+v$. Si cela a un sens, il le fait et ce calcul `renvoie` alors la valeur V trouvée.
- Enfin, `2+x` ayant renvoyé V, la ligne est interprétée finalement comme `x=`V, et `Python` met donc cette valeur V dans la variable `x`.

En programmation, **la notion de fonction** est capitale. Elle permet d'écrire un sous programme et de s'en servir facilement dans la suite. Par exemple, regardons le code suivant.

```
def reste2(x):
        return x%2
```

Une fois ce bloc lu par `Python`, une nouvelle fonction nommée `reste2` est crée. Elle prends un argument, ici noté `x`, et renvoie le résultat du calcul `x%2` (rappel : il s'agit du reste de x dans sa division euclidienne par 2). Autrement dit, quand `Python` lira `reste2(34)`, il ira chercher le code précédent, l'interprétera avec `x=34`, et ici continuera sa lecture comme s'il avait lu `34 %2` .
Autrement dit, la fonction `Reste2` appliquée à une donnée `x` **renvoie** la donnée `x%2`. Dans `Python`, le mot clé `return` indique à `Python` ce qu'il devra renvoyer, dans le cadre d'une fonction.

2. Définir une fonction

Voici un autre exemple :

```
def fun(n,k):
        for i in range(n):
                print(i)
        if k < n:
                return k
        else:
                return n
```

Définir une fonction en python

Pour définir une fonction :

- On annonce la création d'une fonction avec `def`,
- on choisit un nom pour la fonction, ici `fun`,
- on spécifie le nombre de ses arguments et on les nomme. Ici, il y a deux arguments nommés `k` et `n`.
- Une fois ceci fait, on utilise les deux-points et on indente le code correspondant à `fun`.
- Dans notre exemple, la fonction commence par afficher tous les entiers de `0` à `n-1`.
- Ensuite, la fonction **renvoie** l'argument `k` si le test `k<n` donne `True`, et renvoie `n` sinon.

Quand une fonction renvoie une valeur avec `return`, `Python` considère que le travail de la fonction est terminé et **cesse de lire** le code de la fonction. `return` a donc un rôle tout a fait particulier. On peut très bien définir une fonction sans argument, ou sans valeur renvoyée.

Les arguments d'une fonction sont aussi appelées ses entrées. Lorsque `Python` applique une fonction à une valeur donnée, on dit qu'il fait un appel de la fonction ("call" en anglais).

Exercice 8. On veut une fonction `f` qui, étant donné deux données x et y, renvoie la donnée $(x+1)(y-1)$. Déterminer les erreurs de syntaxe dans la définition de fonction ci-dessous.

```
def f(x,y)
        a=x+1
        b=y-1
return a*b
```

3. Exercices

Exercice 9. Définir en python la fonction `c` donnée, pour x un réel convenable, par $c(x) = \left(1+\frac{1}{x}\right)\left(1-\frac{1}{1+x}\right)$. Faire calculer des valeurs de c, émettre une conjecture et la démontrer.

Exercice 10. Soient a et b deux réels strictement positifs. On appelle :

- Moyenne **arithmétique** de a et b le réel $\frac{a+b}{2}$,
- moyenne **géométrique** de a et b le réel $\sqrt{ab}$
- moyenne **harmonique** de a et b le réel $\frac{2}{\frac{1}{a}+\frac{1}{b}}$.

Coder trois fonctions `ma,mg,mh` prenant en entrée deux réels supposés strictement positifs et renvoyant respectivement leurs moyennes arithmétique, géométrique et harmonique.

Exercice 11. En utilisant les fonctions de l'exercice précédent, comparer (au sens de $\leq$) ces trois notions de moyenne pour beaucoup de valeurs de a et b. Vérifier un résultat démontré en TD.

Exercice 12.

1. Compléter le code suivant pour qu'il définisse une fonction d'entête `def estCarre(n):` prenant en entrée un entier `n` et renvoyant `True` si `n` est le carré d'un entier, et `False` sinon.
```
def estCarre(n):
        reponse=False
        for k in range(n+1):
                if n... :
                        reponse=True
        return(reponse)
```
2. Même question, avec une structure différente.
```
def estCarre(n):
        for k in range(n+1):
                if n... :
                        return(True)
        return(...)
```
3. Coder une fonction d'entête `def test11(n):` prenant en entrée un entier `n` et renvoyant `True` si $\texttt{n}^2+1$ est le carré d'un entier, et `False` sinon.
4. Soit n un entier. A quelle condition n^2+1 est-il le carré d'un entier? Conjecturer une réponse à l'aide de la question précédente.
5. Démontrer par l'absurde votre conjecture.

Exercice 13.

1. Écrire une fonction prenant en entrée deux entiers naturels `n` et `p` et renvoyant la valeur de la somme :
$$\sum_{k=0}^{n} k^p.$$
2. Á l'aide de cette fonction, émettre une conjecture reliant les sommes $\sum_{k=0}^{n} k$ et $\sum_{k=0}^{n} k^3$, pour tout entier n.
3. (Maths) En déduire une formule simple donnant $\sum_{k=0}^{n} k^3$ en fonction de $n \in \mathbb{N}$ puis la démontrer.