INTRODUCTION À PYTHON : FONCTIONS, LISTES

1 Les fonctions

Les fonctions sont des scripts qui prennent un ou plusieurs *arguments* en entrée et ***retournent*** un résultat en fonction de ces arguments.

```
def produit(a,b) :
    return a*b   # la fonction "retourne" le produit de a et b
```

Exécuter le script ci-dessus, puis taper par exemple `produit(5,-2)` dans la console.

Syntaxe d'une fonction. Le mot-clé `def` signifie qu'on introduit une fonction. Ci-dessus la fonction s'appelle `produit` et prend 2 arguments. Il faut ensuite un « : » et indenter la fonction. Le mot-clé `return` permet de choisir ce que renvoie la fonction.

```
def NomDeLaFonction ( arg1 , arg2 , ...) :  # arg pour argument
    ...
    ...          # tant qu'on est indenté, on reste dans la fonction
    ...
    return ...
```

Les fonctions dans le Workspace. Lorsque la fonction est écrite, *ou chaque fois qu'elle est modifiée*, il faut exécuter la portion du script qui contient cette fonction : cela permet « d'enregistrer » la fonction dans le Workspace. Une fois enregistrée, elle peut être utilisée dans la console ou dans le reste du script.

Exercice 1. Écrire une fonction `magie` qui prend comme arguments quatre entiers a,b,c,d et *retourne*

$$(a+b+c+d)(a^2+b^2+c^2+d^2)^2$$

Tester dans la console `magie(2,0,2,3)`. Réponse : « waouh ».

On connait déjà plusieurs fonctions :

- La fonction `print(...)` permet d'afficher son argument dans la console.
- La fonction `range(m,n)`qui est un conteneur avec les valeurs `m,m+1,...,n-1`.
- A noter que `range(n)` équivaut à `range(0,n)`. Ainsi, dans Python, une même fonction peut s'utiliser avec différents nombres d'arguments.

Recopier et compiler le script ci-dessous. Tester la fonction pour différentes valeurs des arguments.

```
def rangeTriple(m,n):
    for k in range(m,n,2):
        print(k)   # affiche les entiers "..." partant de m (inclus) à n
           (exclu)
    return None
```

Question 1. Grâce aux différents tests, déterminer les mots qui correspondent aux « ... » dans le commentaire. À votre avis, que retourne la fonction `range(m,n,p)` où `p` est dans $\mathbb{N}^*$?

Comme le montre la fonction `rangeTriple`, on peut faire un bloc `for` (ou `if` ou `while`) à l'intérieur d'une fonction : il faut cependant faire attention à bien indenter le bloc.

Exercice 2. Compléter le script suivant. Compiler et tester.

```
def nombreDiviseur(n) : # compte le nombre de diviseurs de n
    c = 0
    for d in range(1,n+1) :
        if ... :
            c = c + 1   # si d divise n, on ajoute 1 à c
    return c
```

On notera qu'on peut « imbriquer » les blocs : on peut mettre un bloc `if` dans un bloc `for`, le tout dans une fonction. L'instruction `return` est indentée au même niveau que le `for` : cela marque en même temps la fin des blocs `for` et `if`.

2 Utiliser des fonctions dans d'autres fonctions

Grâce au mot-clé `return`, on peut utiliser le résultat d'une fonction à tout autre endroit du code.

Exercice 3. Compléter le script suivant. Compiler et tester.

```
def estPremier(n) :    # retourne True si n est premier, False sinon
    if nombreDiviseur(n) == ... :
        return True
    else :
        return False
```

Une fois la fonction définie, vous ne devez pas hésiter à la réutiliser ailleurs. C'est la raison d'être d'une fonction ! Dans un DS d'informatique, plusieurs questions préliminaires servent à construire des fonctions basiques. Les questions suivantes demandent d'écrire des fonctions plus complexes, mais les fonctions basiques des premières questions permettent justement de simplifier les choses.

Exercice 4. Écrire une fonction `factorielle` qui prend en argument un entier naturel et qui retourne sa factorielle.

Exercice 5. Écrire une fonction `binom(k,n)` qui retourne le coefficient binomial $\binom{n}{k}$. Il n'est pas nécessaire de vérifier que k, n sont des entiers ou que $0 \leq k \leq n$.

Ici, la fonction `factorielle`, basique, permet de construire facilement la fonction `binom`, plus complexe.

3 Listes

Une liste est un objet Python qui permet de rassembler un ou plusieurs objets. Voici quelques rappels de syntaxe :

- Pour définir une liste : `L = [0, 3.14, "ha"]` est une liste qui contient 3 éléments. `L = []` définit une liste vide.
- Pour accéder à un élement de la liste : `L[0]` pour le **premier** élément, `L[1]` pour le second, etc.
- Pour ajouter un élément à une liste : `L.append(d)`.
- Pour concaténer deux listes `L1` et `L2` en une seule : `L1+L2`. En particulier, `L.append(d)` peut aussi s'écrire `L=L+[d]`.
- Pour obtenir le nombre d'élements d'une liste : `len(L)`.

Exemple. La fonction suivante permet par exemple de calculer la liste qui contient les carrés des n premiers nombres entiers. Compiler et tester pour différentes valeurs de n.

```
def listeCarre(n) :  # retourne la liste [0**2, 1**2, ..., (n-1)**2]
    L = []
    for i in range(n) :
        L.append(i**2)   # on ajoute i au carré à la liste
    return L
```

Exercice 6. On cherche à écrire une fonction `sommeListe` qui prend en argument une liste de nombres `L` et qui retourne la somme de chaque élément de `L`. Compléter la fonction suivante et tester.

```
def sommeListe(L) :    # L est une liste de nombres
    s = ...
    n = ...
    for k in range(n):
        s = s + L[k]       # on ajoute à s chaque élément de L
    return s
```

Exercice 7. Écrire une fonction `duplication` qui prend en argument une liste `L` et retourne cette liste concaténée avec elle-même.

ATTENTION : si une liste `L` possède `n` éléments, alors :

- `L[0]` retourne le **premier** élément de la liste, et donc...
- `L[n-1]` retourne le **dernier** élément de la liste
- `L[n]` génère toujours une erreur car on dépasse la taille de la liste.

De même, dans le conteneur `range(n)`, il y a les valeurs de `0` à `n-1`, mais pas la valeur `n`.

4 Listes en compréhension

On rappelle la syntaxe de la liste en compréhension :

```
L = [ <f(var)> for <var> in <conteneur> ]
```

Cette instruction crée une liste : la variable `var` parcourt chaque valeur du `conteneur`, et on ajoute à la liste une expression qui peut dépendre de `var`.

```
conteneur : [   a ,   b ,   c , ...]
        L : [ f(a), f(b), f(c), ...]
```

L'instruction `L = [ ... ]` ci-dessus revient à écrire :

```
L = []
for <var> in <conteneur> :
    L.append( <f(var)> )
```

Le principal avantage des listes en compréhension est d'écrire un code plus compact : les 3 lignes ci-dessus n'en font plus qu'une avec l'instruction `L = [ ... ]`.

Exemple. La fonction `listeCarre` qu'on a écrite plus haut peut se réécrire en une ligne avec une liste en compréhension :

```
def listeCarre2(n) :
    return [ i**2 for i in range(n) ]
```

Exercice 8. Compléter la fonction `double` ci-dessous : elle prend en argument une liste `L` de nombres et retourne une liste de même taille où chaque élément a été multiplié par deux.

```
def double(L):
    return [ 2*k for ... ]    # liste en compréhension
```

Exercice 9. Écrire une fonction `listeFactorielle` qui à un entier `n` retourne la liste des `n` premières factorielles (0! puis 1!, etc.).

5 Un mot-clé utile : `in`

L'instruction

`<valeur> in <conteneur>`

renvoie `True` si `<valeur>` est un élément de `<conteneur>`, `False` sinon. Cela fonctionne avec tout type de conteneur : listes, tuples, string, etc.

```
>>> 3 in [0,3]
True
>>> -6 in [12]
False
>>> 'A' in 'ah'
False  # True avec : 'A' in 'Ah'
```

Exercice 10. Écrire une fonction `ajout` qui prend en argument une liste `L` et un objet quelconque `a` : si l'objet `a` n'est pas déjà dans `L`, on l'ajoute à `L`. Sinon, on ne fait rien. Enfin, on retourne la nouvelle liste.

6 Exercices pour s'entrainer

Pour faire les exercices ci-dessous, n'hésitez pas à réutiliser les fonctions de ce TP !

Exercice 11. Modifier la fonction `nombreDiviseur` pour que, partant d'un argument `n`, elle retourne une liste de tous les diviseurs (positifs) de `n`.

Exercice 12. Écrire une fonction `Pascal(n)` qui retourne une liste correspondant à la ligne numéro `n` du triangle de Pascal.

Exercice 13. Écrire une fonction `fibonacci(n)` qui retourne la liste des `n` premières valeurs de la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$u_0 = 0 \qquad u_1 = 1 \qquad \text{et} \qquad \forall n \in \mathbb{N} \qquad u_{n+2} = u_{n+1} + u_n$$

Exercice 14. Écrire une fonction `ListePremier(n)` qui retourne la liste des `n` premiers nombres premiers. *Indication : il faut utiliser une boucle* `while`.

Exercice 15. Écrire une fonction `retourne(L)` qui retourne la liste `L` dans l'ordre inverse. *On ne peut pas utiliser de slicing.*

Exercice 16. Écrire une fonction `demult(n)` qui retourne la liste `[ 1, 2,2, 3,3,3, 4,4,4,4, ... ]` jusqu'à `n` fois le nombre `n`.

Exercice 17 (demi $\star$)**.** Écrire une fonction `noDuplicate(L)` qui prend en argument une liste `L` et retourne la même liste mais sans doublon.