TP 3. Fonctions

Lorsqu'on commence à écrire des programmes plus denses il est préférable d'éviter d'accumuler de grandes quantités de lignes de texte dans un seul et même fichier.

Une approche efficace est de décomposer le programme en sous-programmes avec une tâche bien spéficique et plus élémentaire.

Par ailleurs, si un même bloc d'instructions apparaît à plusieurs reprise dans un programme : il est préférable de le définir une bonne fois pour toute et d'éviter de le reprendre systématiquement.

Une solution dans le langage Python est de définir des **fonctions**.

1 Généralités sur les fonctions

La syntaxe générale est la suivante :

`def nom`(*liste des paramètres*) :
blocs d'instructions ;

L'appel d'une fonction déclenche l'exécution des blocs d'instructions qui la compose.

Exemple. Voici une fonction qui admet un paramètre d'entrée et qui retourne la valeur : carré de l'argument.

```
>>> def carre(x) :
        return x*x
>>> carre(2)
4
```

Remarques.

1. La définition d'une fonction commence par le mot clé `def` et se poursuit par le nom de la fonction. Vous devez éviter les mots réservés (comme `if`, `while`...) et les caractères spéciaux et accentués.
2. La liste des paramètres suit entre parenthèses le nom de la fonction. C'est une variable qui va recevoir les arguments (*paramètres effectifs*) d'entrée de la fonction. Le nom d'une variable d'un programme que nous passons en argument d'une fonction n'a rien à voir avec le nom du paramètre de la fonction.
   Les deux exemples ci-dessous illustre cela :

```
>>> x=1
>>> while x<5 :
        carre(x)
        x=x+1
1
4
9
16
```

```
>>> y=1
>>> while y<5 :
        carre(y)
        y=y+1
1
4
9
16
```

   Note. Les paramètres sont facultatifs. L'exemple ci-dessous montre une fonction qui affiche un message lors de son appel.

```
>>> def coucou() :
        print('Bonjour, ça va ?')
>>> coucou()
Bonjour, ça va ?
```

Exercice 1. En ligne de commande. Définir une fonction `prenom` qui prend en argument une chaine de caractères `X` (disons un prénom) et affiche le message : `Bonjour, X`.

3. Il n'est pas nécessaire de préciser quel *type* doit recevoir un paramètre. Il est cependant possible de commenter sa fonction pour indiquer à l'utilisateur ce qu'elle fait ; il est également possible d'attribuer une valeur par défaut aux paramètres.

   Exemple. Voici l'exemple de la fonction carrée bien documentée.

```
>>> def carre(x=3) :
        """Ceci est la fonction carrée"""
        return x*x
>>> carre()
9
```

Exercice 2. En ligne de commande. Réaliser une fonction valeur absolue (bien documentée) à l'aide d'une structure conditionnelle.

4. L'instruction `return` définit la valeur que la fonction renvoie : elle termine l'appel de la fonction car elle n'est exécuté qu'une fois. Elle est à distinguer de la fonction `print` qui se contente d'un affichage et peut-être présente à plusieurs endroits.

Exercice 3. Rédiger et exécuter les lignes de commandes ci-dessous :

```
>>> def carre(x) :
        return x*x
>>> carre(3) +1
```

```
>>> def carre(x) :
        print(x*x)
>>> carre(3) + 1
```

5. Dans un script : la définition des fonctions doit précéder leur utilisation. Une fois un script exécuté : les fonctions définies dans ce script sont accessibles en ligne de commande.

Exercice 4. Rédiger un programme `compteurVoyelle.py` constitué des deux fonctions suivantes :

1. Une fonction `testVoyelle`, dont le paramètre d'entrée est un caractère, qui renvoie *True* si le caractère est une voyelle et *False* sinon.
2. Une fonction `comptVoyelle`, dont le paramètre d'entrée est une chaine de caractères, qui renvoie le nombre de caractères qui sont des voyelles. *Indication :* une variable k parcourt la chaine de caractère reçue en argument ; un compteur s'incrémente à chaque fois que k tombe sur une voyelle : le test est réalisé à l'aide de la fonction précédente.

Votre programme demande à l'utilisateur de saisir une chaine de caractères et lui répond le nombre de voyelles contenues dans la chaine.

Remarque. Il est possible de définir une ou plusieurs fonctions dans un premier fichier texte et d'y faire appel dans un autre script. **Attention :** pour ne pas avoir à spécifier le chemin d'accès, il est préférable de placer les deux scripts dans un même dossier. Pour accéder aux fonctions placées dans un autre fichier on utilise la même syntaxe que pour les modules prédéfinis avec le mot clé `import` et le nom du fichier.

Exercice 5. **Valuation 2-adique.** Définir une fonction `val2` qui prend un entier en argument et renvoie la plus grande puissance de 2 qui divise (en nombre entier) cet entier.
Exemple. La valuation 2-adique de 24 est 3 car 8 divise 24 mais pas 16.

Exercice 6. Autour des palindromes.

Un mot est un palindrome si l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche.
Exemples : les mots « radar » et « non » sont des palindromes.

On se propose de créer des scripts Python qui vont nous permettre de tester si une chaine de caractères est un palindrome.

Rappels sur le type `str`.

- Par accéder à un caractère d'une chaine de caractère `Ch`, on utilise la syntaxe `Ch[k]` où k désigne l'indice du caractère (la numérotation commence à 0).
- La fonction `len` donne le nombre de caractères d'une chaine.
- Par ajouter un caractère à une chaine, on utilise l'opérateur +.
- On teste l'égalité de deux chaines de caractères avec l'opérateur ==.

```
>>> Ch = 'coucou'
>>> Ch[2]
'u',
>>> len(Ch)
6
>>> 'couco' + 'u' :
'coucou'
>>> Ch == 'coucou' :
True
>>> Ch == 'bonjour' :
False
```

1. **Méthode 1**. Ouvrir et enregistrer un nouveau fichier `palindrome1.py`. Vous rédigez les commandes dans ce fichier.
   (a) Définir une fonction `egalite`, qui prend en entrée deux chaines de caractères `Ch1` et `Ch2` et qui retroune `True` si `Ch1` et `Ch2` sont égales et `False` sinon.
   (b) Définir une fonction `symetrique`, qui prend en entrée une chaine de caractères `Ch` et qui retourne la chaine formée par les caractères de `Ch` lus de la droite vers la gauche.

   Exemple.
   ```
   >>> symetrique('coucou') :
   uocuoc
   ```
   (c) Pour compléter votre programme : demandez à l'utilisateur de saisir un `mot` ; votre programme réalise l'affichage du texte : `Ce mot est un palindrome`, si c'est bien le cas et : `Ce mot n'est pas un palindrome` sinon.
2. **Méthode 2 : directe.** La méthode présentée dans la question précédente nécessite de tester l'égalité de deux chaines et de créer une nouvelle chaine de caractère. On envisage dans cette question une méthode moins coûteuse.
   Ouvrir et enregistrer un nouveau fichier `palindrome2.py`. Vous rédigez les commandes dans ce fichier.

   Fonction : palindrome(Ch).
   Entrée : une chaine de caractères Ch
   Sortie : un booléen
   n = longueur(Ch)
   $k = 0$ `# indice qui parcourt la moitié de la chaine`
   reponse=Vrai
   Tant que *reponse* **et** $k \leq n//2$:
   reponse= ($Ch[k] == Ch[n-1-k]$)
   $k = k + 1$ `# incrémentation de l'indice`
   Retourner *reponse*

   (a) Lire et comprendre l'algorithme ci-dessus puis l'implanter en Python.
   (b) Le compléter pour qu'il tienne compte de la chaine vide.
   (c) Reprendre la question 1. (c) avec la fonction palindrome.
3. On convient qu'une **phrase** est un palindrome lorsque les lettres des mots qui la compose peuvent être lus dans les deux sens **sans tenir compte des espaces**.
   Adapter votre programme précédent pour qu'il teste si une phrase est un palindrome. Indication : chercher sur internet une **méthode** qui permet d'enlever les espaces d'une chaine de caractères.
   Tester votre programme sur la locution latine :

   IN GIRUM IMUS NOCTE ET CONSUMIMUR IGNI

2 Variables locales et variables globales

Les variables utilisées dans le corps d'une fonction sont des **variables locales**. Elles sont affectées au moment de l'appel de la fonction et sont détruites à la sortie.

Exercice 7. Cet exercice illustre le fonctionnement des variables locales

1. Définir en ligne de commande la fonction ci-dessous. Prédire les retours des deux dernières lignes. Le vérifier.

```
>>> def fonctionTest(x) :
        var1='bonjour'
        return (x,var1)
>>> fonctionTest(0)
?.....
>>> print(var1)
?.....
```

2. Même question en définissant à l'extérieur de la fonction une variable `var1`.

```
>>> var1 = 'jenesuispaslocale'
>>> def fonctionTest(x) :
        var1='bonjour'
        return (x,var1)
>>> fonctionTest(0)
?.....
>>> print(var1)
?.....
```

Les variables utilisées à l'extérieur d'une fonction sont des **variables globales**.

Exercice 8. Définir la fonction suivante et affecté `var1` et `var2` comme indiqué. Prédire les retours des deux dernières lignes. Le vérifier.

```
>>> def fonctionTest2() :
        var1='abc'
        return (var1,var2)
>>> var1='xyz'
>>> var2='123'
>>> fonctionTest2()
?.....
>>> print(var1,var2)
?.....
```

Commentaires.

- À l'intérieur de la fonction `fonctionTest2` une variable `var1` reçoit `'abc'`. C'est une variable **locale**.
- À l'extérieur de la fonction nous définissons `var1` et `var2` qui reçoivent respectivement `'xyz'` et `'123'` : ce sont des variables **globales**.
- La variable `var1` est utilisé deux fois. Lors de l'appel de la fonction elle reçoit temporairement une valeur. À la fin de l'appel, cette affectation temporaire est terminée et elle retrouve l'affectation globale qu'elle avait.
- **Conclusion.** Dans un programme, on peut définir des fonctions et utiliser des variables locales dans ces fonctions sans se préoccuper de savoir si elles existent ailleurs dans le script. Les affectations temporaires à l'appel de la fonction ne risquent pas d'interférer avec les autres variables.

On peut cependant modifier la valeur d'une variable globale en la déclarant comme telle dans le corps de la fonction. On utilise l'instrution `global`.

Exercice 9. Définir et tester la fonction suivante.

```
>>> def incremente() :
        global a
        return a + 1
>>> a=2
>>> incremente()
?.....
```

Remarque. L'utilisation d'une variable globale dans une fonction permet de modifier la valeur d'une variable globale.

Exercice 10. Définir et tester la *procédure* suivante.

```
>>> def ecrase(x) :
        global a
        a=x
>>> a=1
>>> ecrase('bonjour')
>>> a
?.....
```

Commentaire.

- La fonction ci-dessus ne retourne pas de valeur : on parle plutôt de procédure. Elle a cependant en effet non négligeable sur la variable locale `a`.

3 Passage des arguments

Tout comme les *variables locales* les variables qui représentent les **paramètres formels** ne sont affectés qu'au moment de l'appel de la fonction même s'ils ont un nom identique à celui d'une variable globale.
L'exemple ci-contre illustre cela.

```
>>> def carre(x) :
        return x**2
>>> a,x=2,3
>>> carre(a),carre(x)
(4,9)
>>> print(a,x)
(2,3)
```

On peut passer de tout type d'objet en argument d'une fonction. Il faut alors traiter le paramètre formel en cohérence avec le type qu'il doit recevoir.

Exercice 11.

1. Définir une fonction `somme` dont les paramètres d'entrée sont une fontion `f` et un nombre entier `N`. La fonction retourne la valeur $f(1) + f(2) + ... + f(n)$.
2. Tester votre fonction sur les fonctions $x \mapsto x^2$ et $x \mapsto \exp(x)$.