Python 3 –TDs-Fipa1

Saison 1 - épisode 4

1 Découvrons et testons le langage Python _____ 1

1.1 Les Fonctions _____ 1

1.2 Variables locales, variables globales _____ 4

2 Mise en pratique _____ 6

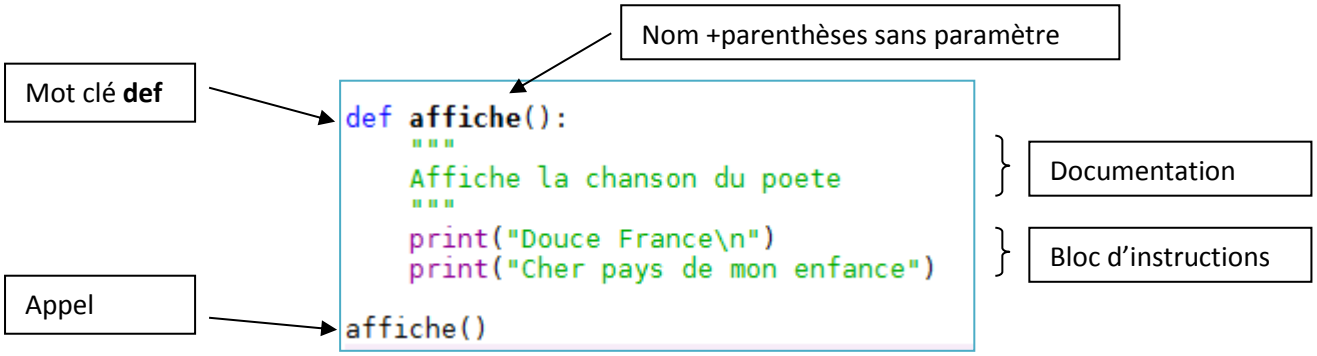
1 Découvrons et testons le langage Python

1.1 Les Fonctions

Une **fonction** est un ensemble d'instructions regroupées sous un nom.

Intérêt : factoriser le code, éviter la duplication de code, réutilisation du code, amélioration de conception d'applications informatiques.

Définition : un mot clef **def** suivi du **nom,** de **parenthèses** entourant ou pas des **paramètres**, d'une **chaine de documentation** indentée comme le corps de la fonction, **du bloc d'instructions** qui constitue le **corps de la fonction.**



Le Passage des arguments par paramètre se fait par affectation :

Chaque argument de l'appel de la fonction correspond dans l'ordre à un paramètre de la fonction. Il faut respecter l'ordre et le type des deux côtés (appel et définition).

Il existe différents paramètres. Le ***paramètre d'entrée*** est celui passé en paramètre à l'appel de la fonction. Le ***paramètre de sortie*** est celui qui est retourné par la fonction.

Le ***paramètre formel*** est celui passé en argument de la fonction dans sa définition ou signature et utilisé dans le corps de la fonction. Le ***paramètre effectif*** est celui passé en argument de la fonction à son appel.

1.1.1 Une fonction avec un ou plusieurs paramètres, pas de retour

Sans l'instruction return, on la nomme aussi **procédure** (qui fait quelque chose).

- Définition et appel dans le script Python :

```
def affiche2(t1,t2):
    """
    Affiche la chanson du poete
    """
    print(t1,"\n",t2)


affiche2("Douce France","Cher pays de mon enfance")
```

- Exécution de la fonction directement dans la console Python :

Ceci

```
>>> affiche2("Douce France","Cher pays de mon enfance")
Douce France
 Cher pays de mon enfance
>>>
```

Ou cela

```
>>> s1="Bercée de tendre insouciance"
>>> s2="Je t'ai gardée dans mon coeur !"
>>> affiche2(s1,s2)
Bercée de tendre insouciance
 Je t'ai gardée dans mon coeur !
>>>
```

1.1.2 Une fonction avec un ou plusieurs paramètres, un ou plusieurs retours

- **Avec un retour unique :**

```
def calcul(a1,a2):
        return a1*a2

print(calcul(4,8))
```

- **Avec un retour multiple :**

```
def calcul2(a1,a2):
        val1=a1*a2
        val2=a1+a2
        return val1,val2

#programme principal
x,y=calcul2(4,8)
print("la somme : ",y,"la multiplication : ",x)
```

1.1.3 Le passage d'une fonction en paramètre

En Python, on peut transmettre une fonction en paramètre.

Testons !

```
>>> def f(x):
...     return x*2
...
>>> def t(y):
...     return y//2
...
>>> def g(fonc,x):
...     return fonc(x)
...
>>> g(f,2)
4
>>> g(t,4)
2
```

1.1.4 Paramètres avec valeurs par défaut

Il est possible de donner des valeurs par défaut aux paramètres lors de la déclaration de la fonction.

Testons !

```
>>> def jouer(nom,prenom,age,categ="J"):
...         print(nom,prenom,"catégorie", categ)
...
...
>>> jouer('Terieur','Alain',8)
Terieur Alain catégorie J
>>> jouer('Terieur','Alex',18,"S")
Terieur Alex catégorie S
```

Pour une valeur par défaut modifiable on utilise la valeur prédéfinie « ***None*** ».

```
>>> def test(liste=None):
...     if liste is None:
...             liste=[4,5,6,7]
...             return liste
...
>>> print(test())
[4, 5, 6, 7]
```

1.1.5 Passage d'un tuple de valeurs en paramètre

Il est possible de passer un nombre arbitraire d'arguments en utilisant la notation d'un argument final ****args.*** Les paramètres sont passés sous forme de tuple.

```
>>> def totalannuel(*args):
...     tot=0
...     for n in args:
...         tot+=n
...     return tot
...
>>> print(totalannuel(5,10,78,9))
102
```

ou

```
>>> def som(a,b,c):
...     return a+b+c
...
>>> elt=(2,3,5)
>>> print(som(*elt))
10
```

1.1.6 Passage d'un dictionnaire en paramètre

Il est possible de passer un nombre arbitraire d'arguments en utilisant la notation d'un argument final ****kwargs.*** Les paramètres sont passés sous forme de dictionnaire. (kwargs pour Keyword args)

Si la fonction possède plusieurs paramètres, le dictionnaire est en dernier !

Testons !

```
>>> def affiche(**kwargs):
...     print(kwargs)
...
>>> affiche(role1='le Roi',role2='le valet')
{'role2': 'le valet', 'role1': 'le Roi'}
>>>
```

En fournissant un dictionnaire

```
>>> dico={'role1':'le Roi','role2':'le valet'}
>>> affiche(**dico)
{'role1': 'le Roi', 'role2': 'le valet'}
```

1.2 Variables locales, variables globales

Les variables définies dans le corps d'une fonction ne sont accessibles qu'à la fonction elle-même. Ce sont des ***variables locales*** à la fonction.

A chaque appel d'une fonction, Python réserve pour elle un nouvel ***espace de noms***. Les contenus des variables de la fonction sont stockés dans cet espace de noms qui est inaccessible depuis l'extérieur de la fonction.

Les variables définies à l'extérieur d'une fonction sont des ***variables globales***. Leur contenu est visible de l'intérieur d'une fonction mais ne peut pas être modifié par la fonction.

On définit ainsi la ***portée d'une variable***.

Testons !

Variable locale : a et variable globale : b

```
>>> def troca():
...     a=30
...     print(a,b)
...
>>> a,b=4,7
>>> troca()
30 7
>>> print(a,b)
4 7
```

```
>>> def modif(a):
...     a=0
...
>>> b=1
>>> b
1
>>> modif(b)
>>> b
1
```

Commentaire sur modif(a) *: La valeur de la variable* ***b****, paramètre effectif de la fonction, est « copiée » dans le paramètre formel* ***a*** *qui joue le rôle d'une variable locale et dont la portée ne dépasse pas le corps de la fonction. La valeur de b est inchangée.*

Et avec une liste ?

```
>>> def swap(a):
...     v=a[0]
...     a[0]=a[1]
...     a[1]=v
...
>>> b=[1,5]
```

```
>>> print('b : ',b)
b :  [1, 5]
>>> swap(b)
>>> print('b : ',b)
b :  [5, 1]
>>>
>>>
```

Commentaire *: Ici la valeur de* ***b*** *est modifiée. Cela tient à ce qu'est la valeur d'une liste. Le lieu de mémoire où est stocké son contenu. On ne peut pas rediriger cette référence de zone de mémoire mais on peut agir sur son contenu. Idem pour les dictionnaires et les tuples non mutables du reste.*

Quand la fonction ***swap()*** *manipule la liste en interne, elle manipule les valeurs contenues à cette adresse, ce qui correspond aux valeurs contenues par* ***b****. La variable* ***b*** *est modifiée.*

Forcer le caractère global d'une variable dans une fonction

```
>>> def maville():
...     global code,ville
...     ville='Brest'
...     code='29200'
...     print(code,ville)
...
>>> ville,code = 'Quimper','29000'
>>> print(code,ville)
29000 Quimper
>>> maville()
29200 Brest
>>> print(code,ville)
29200 Brest
>>>
```

Définition de la fonction

Exécution

Pour conclure cette première étape

L'usage des fonctions dans un script va rendre le programme plus clair, plus lisible, plus efficace.

La définition des fonctions doit précéder leur utilisation.

2 Mise en pratique

Exercice 1

Refactorisons le code de l'exercice7 du TDSaison1Episode3 !

Ouvrez le fichier « ***saison1epi3Exo7.py*** », copiez le, renommez le, transformez le :

- Observez les blocs d'instructions de consultation et de remplissage ;
- Transformez les blocs en fonctions

Exercice 2

1. Transformons le code suivant en une fonction de conversion d'une chaine de caractères en entier :

```
# Définition de la chaîne de caractères
s = 'Quarante-deux, dit Compute-Un, avec infiniment de calme et de majesté.'
# Conversion en liste d'entiers
l = [ord(x) for x in s]
# Tri de la liste
l.sort()
print(l[43])
```

2. Affichez le contenu de la liste avant et après conversion
3. Ecrivez le code de conversion en 1 ligne

Exercice 3

L'objectif de l'exercice est de réaliser un algorithme de cryptage basé sur le chiffre de César. Cette méthode très classique est par exemple décrite ici :
https://fr.wikipedia.org/wiki/Chiffrement_par_d%C3%A9calage

L'algorithme de chiffrement qu'on souhaite implémenter est basé sur le chiffre de César qui utilise un décalage des lettres de l'alphabet. Par exemple dans la figure 1, si on veut chiffrer un mot avec un chiffre (décalage) de César égal à 2, chaque lettre de mot initial est remplacée par la lettre 2 rangs plus loin dans l'alphabet ; après la lettre 'z', on reprend l'alphabet depuis le début (ainsi, 'a' devient 'c', 'b' devient 'd', 'y' devient 'a' et 'z' devient 'b').

Si on cherche à chiffrer le mot "python", la lettre 'p' est remplacée par 'r', 'y' par 'a', ….etc. Nous obtenons ainsi le mot crypté 'ravjqp' à partir de texte initial 'python'.

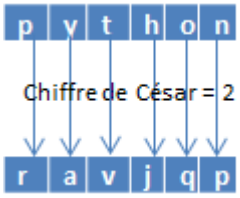


Fig 1. Exemple de chiffrement par un chiffre de César=2

Ecrivons un programme pour chiffrer et déchiffrer un mot !

1. Une première version sera dédiée au mot « python » comme ci-dessus avec un chiffre de César égal à 2. Le code comportera trois fonctions au moins : la première fonction permettra de **fabriquer le dictionnaire de chiffrement pour le mot**, la deuxième fonction **chiffrera le mot initial en mot chiffré**, la troisième fonction **déchiffrera le mot chiffré en mot initial**. Affichez les différents dictionnaires et les valeurs successives des mots.
2. Une deuxième version permettra de **saisir au clavier le chiffre de césar** et pourquoi pas le mot à chiffrer-déchiffrer.