

Les fonctions et les procédures en Python



Définitions

- **Fonction**: Bloc d'instructions nommé et paramétrées réalisant une certaine tâche. Elle admet **zéro, un** ou **plusieurs** paramètres **et renvoie toujours un résultat**
- **Procédure** : bloc d'instructions nommé et paramétrées réalisant une certaine tâche . Elle admet zéro, un ou plusieurs paramètres **mais ne retourne pas de valeur**

Pourquoi des fonctions ?

- Réutiliser un bloc d'instructions
- Regroupement de blocs de code souvent répétés
- Éventuellement dans un module séparé, et réutilisable dans différents programmes
- Découper un problème en sous-tâches
- Code plus facile à tester isolément
- Programme plus simple à comprendre, une fonction étant identifiée pour réaliser une tâche particulière

$$a^n + b^m$$

Algorithme Somme

Variables: a,n,b,m,P1,P2,i: entier

Début

Lire(a,n)

Lire(b,m)

P1 ← 1

Pour i allant de 1 à n faire

P1 ← P1*a

FinPour

P2 ← 1

Pour i allant de 1 à m faire

P2 ← P2*b

FinPour

Ecrire(P1+P2)

Fin

Syntaxe de déclaration

Pour la déclaration d'une
procédure

```
def nomDeLaProcédure(par1,par2,.....parn):  
    " Cette ligne explique à quoi sert la procedure "  
    bloc d'instructions  
    ...
```

Pour la déclaration d'une fonction

```
def nomDeLaFonction(par1,par2,.....parn):  
    "Cette ligne explique à quoi sert la fonction "  
    bloc d'instructions  
    return (resultat)  
    ...
```

Définition d'une fonction

La définition d'une fonction

- commence par le mot-clé **def**,
- suivi du **nom de la fonction**, et d'une liste entre **parenthèses** de **paramètres formels** ; cette première ligne se termine par des double-points **:**
- Les instructions qui forment le corps de la fonction commencent sur la ligne suivante, indentée par **quatre espaces** **(ou une tabulation)**
- La première instruction du corps de la fonction peut être un texte dans une chaîne de caractères ; cette chaîne est la chaîne de documentation de la fonction. On peut la visualiser dans un terminal en tapant l'instruction **help**(nom_de_la_fonction).
- Le **retour à la ligne** signale la **fin de la fonction**
- Dès que l'instruction return est exécutée (si elle est présente), l'exécution de la fonction se termine ;
- la partie du code écrite après l'instruction return n'est jamais exécutée.

Appel d'une fonction

- Une fois qu'elle a été définie, une fonction peut **être appelée** autant de fois que nécessaire pour réaliser le **traitement sur les données**
- À chaque appel **le corps** de la fonction **est exécuté** et travaille sur les **valeurs d'arguments** fournis pour les paramètres

Exemple d'une procédure

```
def table_multiplication(base):  
  
    "Affiche la table de multiplication d'un entier saisi"  
    i = 1  
    while i<11 :  
        print(i, '*', base, " = ", i*base)  
        i = i +1  
  
#le programme principal  
n = int(input('Entrez un entier : '))  
table_multiplication(n)
```

```
Entrez un entier : 11  
1 * 11 = 11  
2 * 11 = 22  
3 * 11 = 33  
4 * 11 = 44  
5 * 11 = 55  
6 * 11 = 66  
7 * 11 = 77  
8 * 11 = 88  
9 * 11 = 99  
10 * 11 = 110
```

Exemple d'une fonction

```
def facto(n):  
    "la fonction calcule la factorielle d'un entier donné"  
    f=1  
    for i in range(1,n+1) :  
        f = f * i  
    return f  
#le programme principal  
a = int(input('Entrez un entier : '))  
print(a, '!= ', facto(a))  
help(facto)  
|
```

```
>>> ===== RESTART =====  
>>>  
Entrez un entier : 8  
8 != 40320  
Help on function facto in module __main__:  
  
facto(n)  
    la fonction calcule la factorielle d'un entier donné
```


Types de paramètres

- Dans les langages de programmation, on parle de:
 - Passage par valeur
 - Passage par adresse (ou par référence)

Algorithmique	Langage de Programmation
Paramètre d'entrée	Passage par valeur
Paramètre d'entrée-sortie	Passage par adresse (par référence)

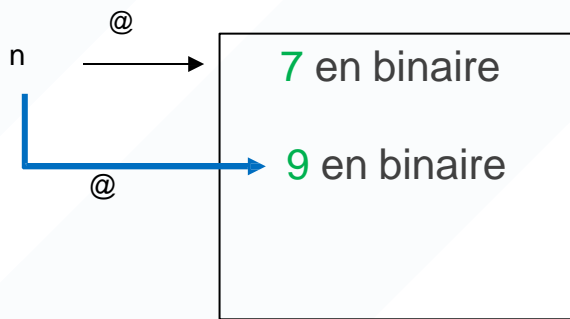
Passage de paramètres en python

Mutable et non mutable

- En Python, il existe deux types d'objets: les **mutables** (listes, dictionnaires....) et les **non mutables** (strings, int, floats, tuples, etc).
- Les mutables sont ceux qu'on peut modifier après leur création. Les non mutables sont ceux qu'on ne peut pas modifier après création.

n=7

n=9



Mémoire

Passage de paramètres en python

- Les variables **numériques et non modifiables (non mutables)** passent **par valeur**
- les variables **modifiables (mutables)** (liste et dictionnaire) passent par **référence**
(ou adresse)

Variables locales et variables Globales

- Les **variables locales** sont définis à l'intérieur du corps d'une fonction, ces variables ne sont **accessibles** qu'à la **fonction elle-même**.
- Les variables **définies à l'extérieur** d'une fonction sont des **variables globales**. Leur contenu est « **visible** » de l'intérieur d'une fonction, mais la fonction ne **peut pas le modifier**

Exemple

```
def mask():  
    p = 20  
    print (p, q)  
  
####Program Principal  
  
p, q = 15, 38  
mask()  
print (p, q)
```

20 38

15 38

le même nom de variable **p** a été utilisé ici à deux reprises, **pour définir deux variables différentes** : l'une est globale et l'autre est locale

Règle de priorité :
à l'intérieur d'une fonction ce sont les variables **définies localement** qui ont la priorité.

Une variable locale peut avoir le même nom qu'une variable de l'espace de noms global mais elle reste néanmoins indépendante.

Comment définir une variable globale à l'intérieur d'une fonction

- Par le mot clé **global**

```
def monter():  
    global a  
    a = a+1  
    print (a)  
#Program principal  
a = 15  
monter()  
monter()
```

```
16  
17
```

Exercise

Qu'affiche ce programme ? :

```
def test():
```

```
    b = 5
```

```
    print(a, b)
```

```
a = 2
```

```
b = 7
```

```
test()
```

```
print(a, b)
```

2 5

2 7

Valeurs par défaut pour les paramètres

- il est possible de définir une valeur **par défaut** pour chacun des paramètres d'une fonction. **On obtient ainsi une fonction qui peut être appelée avec une partie seulement des arguments attendus.**

Exemples :

```
def message (destinataire,jour,heure, salle=24):  
    "une fonction qui affiche les information sur un rattrapage"  
    print("les étudiants de la filère", destinataire,"auront un rattrapage le", jour," à ", heure," à la salle", salle)
```

```
##### Programme principale"
```

```
message("GI","mardi","8h")
```

```
message("GI","mardi","8h","17")
```

```
===== RESTART: C:/Users/AS/Desktop/exemple python/parametres_fonction.py =====  
les étudiants de la filère GI auront un rattrapage le mardi à 8h à la salle 24  
les étudiants de la filère GI auront un rattrapage le mardi à 8h à la salle 17  
>>> |
```


Les étiquettes pour les paramètres

- il est possible de donner **des étiquettes** aux **paramètres** afin de donner plus de flexibilité dans l'ordre des paramètres
- On peut faire appel à la fonction en fournissant les arguments correspondants **dans n'importe quel ordre**, à la condition de **désigner nommément les paramètres** correspondants

Exemples :

```
def afficher(civ="M.:", Nom="Chafik BAIDADA"):  
    print("Bonjour ", civ)  
    print(Nom)  
  
afficher()  
afficher("Madame", "Bouchra")  
afficher(Nom="Ali Alaoui", civ="M.:" )
```

```
Bonjour M.:  
Chafik BAIDADA  
Bonjour Madame  
Bouchra  
Bonjour M.:  
Ali Alaoui  
>>>
```

Les Arguments arbitraires

- afin de donner plus de flexibilité dans l'appel des fonctions. il est possible de passer **nombre variable** de **paramètres** En utilisant l'opérateur *****;
- Ceci est réalisable grâce au mécanisme de **tuple** (on va voir après)
- Exemple :

```
def ma_function(*enfant):  
    print("Mon premier enfant est", enfant[0])  
  
ma_function("Rim", "Taj Eddine", "Rita")
```

```
===== RES IAK  
Mon premier enfant est Rim
```

Les valeurs de retours des fonctions

- Les fonctions en python peuvent avoir **plusieurs valeurs de retour**
- Les valeurs de retour doivent être séparés par des virgules
- **Exemple :**

```
>>> def addmulti(a,b):  
    return a+b,a*b
```

```
>>> addmulti(3,2)  
(5, 6)
```

Les fonctions récursives

- Une **fonction récursive** est une fonction qui s'appelle elle-même plusieurs fois.
- Chaque **appel** à la fonction est **indépendant des autres**, avec ses **propres variables**.
- Il faut définir une **condition d'arrêt** qui correspond généralement au cas simple:

Une récursion a toujours la forme suivante :

if (cas simple):

(solution immédiate)

else: (solution récursive,

impliquant un cas plus simple que le problème original)

- **Exemple:**

```
def factorielle1(n):  
    if n == 1:  
        return n  
    else:  
        f = factorielle1(n-1)*n  
        print("--- factorielle(",n,") =",f)  
        return f
```

```
>>> factorielle1(5)  
--- factorielle( 2 ) = 2  
--- factorielle( 3 ) = 6  
--- factorielle( 4 ) = 24  
--- factorielle( 5 ) = 120
```