

Fonction et récursivité

1 Rappels sur les fonctions

1.1 Paramètres

La syntaxe de définition d'une fonction est :

```
def fonction(param1, param2): # nombre de paramètre arbitraire
    # corps de la fonction
    return # le résultat calculé
```

Pour utiliser cette fonction on l'appellera sous la forme fonction(a, b) où a et b ont des valeurs ou sont directement des objets python.

Dans le corps de la fonction param1 et param2 auront les valeurs de a et de b respectivement.

```
def somme(a, b):
    return a + b

c = somme(5, 12)
d = 4
somme(c, d)
```

21

1.2 Variables locales

Toute variable définie dans le corps d'une fonction est une variable locale, qui n'est accessible que pendant l'exécution de chaque appel, et cache toute variable de même nom définie à l'extérieur de la fonction.

```
a = 12
def f(n):
    a = 5
    b = n
    return a * b

f(10)
```

50

b

NameError

Traceback (most recent

<ipython-input-5-3b5d5c371295> in <module>()
----> 1 b

NameError: name 'b' is not defined

1.3 Exercices

- Définir une fonction "factorielle" qui prend un entier naturel n comme paramètre et retourne n!
- Définir une fonction suite qui prend un nombre u0 et un entier naturel n comme paramètres et retourne la valeur du terme d'indice n de la suite de premier terme u0 et définie par $u_p = \frac{u_{p-1}}{2} + \frac{1}{u_{p-1}}$ pour tout $p \geq 1$

1.4 Combiner les fonctions

Quelle est la différence entre les exemples suivants? Que vaut g(4)

```
def f(n):
    return 2 * n

def g(n):
    a = 1
    for i in range(n):
        a = f(a)
    return a
```

b=g(4)

```
def g(n):
    a = 1

    def f(n):
        return 2 * n
    for i in range(n):
        a = f(a)
    return a
```

b = g(4)

2 Fonctions récursives

Exemples de structures récursives :

2.1 Principe général

Une fonction f est dite récursive si pour calculer sa valeur de retour, elle utilise une valeur de f.

Typiquement, pour une fonction f qui a un paramètre entier n, pour calculer f(n) on va d'abord calculer f(n-1).

```
def f(n):
    return f(n - 1)
```

f(5)

```
-----

RuntimeError                                Traceback (most recent
<ipython-input-11-786fe69aba34> in <module>()
----> 1 f(5)

<ipython-input-10-382384b6672e> in f(n)
----> 2     return f(n - 1)

....

<ipython-input-10-382384b6672e> in f(n)
----> 2     return f(n - 1)

RuntimeError: maximum recursion depth exceeded
```

Evidemment la fonction précédente crée une boucle infinie. Pour toute les fonctions récursives, il faut un cas d'arrêt, c'est à dire une valeur du ou des paramètres qui n'implique plus d'appel à f.

Prenons la définition suivante de n! : $0! = 1 - n! = n \times (n - 1)!$ si $n > 0$

```
def facto(n):
    if n == 0:
        return 1
    return n * facto(n - 1)
```

```
facto(3)
```

6

2.2 Exercice

- Reprendre la suite précédente et créer une fonction `suite_rec` qui répond à la question de manière récursive
- Créer une fonction `puissance(a, n)` qui calcule a^n de manière récursive pour a un nombre et n un entier naturel.

2.3 Cas non trivial

Certain algorithmes s'écrivent naturellement sous forme récursive, et beaucoup moins facilement sous forme itérative.

Reprenons l'exemple du calcul de puissance. On définit a^n pour un nombre a et un entier naturel n par $a^0 = 1$. Pour les $n \geq 1$, il y a deux cas : $a^n = (a^{\frac{n}{2}})^2$ pour les n pairs et $a^n = a \times a^{n-1}$ pour les n impairs.

```
def puissance_rec(a, n):
```

```
1125899906842624
```

L'intérêt majeur ici est la réduction du nombre de multiplication. Une fois n'est pas coutume, nous allons utiliser une variable globale pour compter le nombre d'opérations effectuées.

```
def p(a, n):
    global compteur
```

```
compteur = 0
p(2, 200)
```

```
compteur
```

```
10
```

Pour poursuivre : inverser les éléments d'un tableau, recherche dichotomique.