

IV – Itérations

- Introduction
- itération = répétition de calculs
- élément indispensable en algorithmique
- permet la décomposition d'un problème en sous-problèmes plus simples
- 2 types d'itérations :
 - bornée : nombre d'itérations connu à l'avance
 - non bornée : nombre inconnu, utilisation d'une conditionnelle pour savoir si on continue ou pas

Ré-affectations

- il est possible de modifier autant de fois que l'on veut la valeur d'une variable
- retour sur l'affectation multiple :
 - x, y = 23, 89
- comment échanger les valeurs de x et y ?
 - x, y = y, x
 - ou :
 - tmp = x
 - x = y
 - y = tmp

Mise en œuvre de l'itération

- schéma usuel :

tantque (condition)
corps de l'itération
- fonctionnement : évaluation de la condition (vrai ou faux)
- faux : le corps est ignoré, on passe à la suite
- vrai : exécution du corps (une itération) et on retourne à la condition

Mise en œuvre de l'itération

- en Python, utilisation du mot-clé while :

```
#exemple d'itération
# affichage des 10 premiers carrés et cubes

a=1
while (a<=10):
    #print (a, a**2, a**3)
    print("%1d, %2d, %3d" %(a,a**2,a**3))
    a = a+1
print ("la boucle est finie")
```

Construction d'une itération

- dans l'ordre, procéder aux opérations suivantes :
 - 1) initialisation des variables pour réaliser le calcul
 - 2) actions à réaliser pour passer d'une étape à la suivante
 - 3) condition(s) d'arrêt des calculs (condition(s) de l'itération)

Itérations : conditions à respecter

- variables correctement initialisées
- au moins une instruction de la boucle doit modifier la valeur d'au moins une des variables de la condition
- cette modification doit assurer que l'on sorte de l'itération (la condition devient fausse)
- point difficile mais fondamental pour une bonne programmation !!!

Exemple : suite de Fibonacci

- $\text{fib}(0) = 0$
- $\text{fib}(1) = 1$
- $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$
- calculer les 10 premiers termes (itérations) ?
- appliquer 8 fois la formule (on a déjà les 2 premiers)
- pour calculer la valeur d'un rang donné, il faut :
 - la valeur de rang inférieur (b)
 - la valeur du rang d'avant (a)
 - le rang actuel (pour s'arrêter) (c)

Suite de Fibonacci

```
#fib : vide -> vide
def fib() :
    a = 0
    b = 1
    c = 1
    print ("rang 0 : 0")
    while (c < 10) :
        print ("rang ", c, ": ", a, b)
        tmp = a
        a = b
        b = tmp + b
        c = c + 1
```

Fibonacci version récursive

```
# fib_rec : naturel -> naturel
# n ->
def fib_rec (n):
    "Rend le terme de rang n de la suite de Fibonacci"
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib_rec(n-1) + fib_rec(n-2)
```

Calcul de la racine carrée

- racine d'un nombre a = limite de la suite :
- $x_n = 0.5 \times (x_{n-1} + a / x_{n-1})$
- avec $x_0 = a$
- on ne connaît pas à l'avance le nombre d'itérations
- $\text{eps} = x_n - x_{n-1}$
- si $\text{eps} > \text{précision}$ on continue sinon on s'arrête

Calcul de la racine carrée

```
# Calcul de la racine carrée d'un nombre par la méthode de Newton
#racCar : nombre x nombre+ -> nombre
# a, eps
def racCar (a, eps):
    "Calcul de la racine carrée de a, précision de eps (>0)"
    xAnc = 0
    xNouv = a
    while (abs(xAnc-xNouv)>eps) :
        xAnc=xNouv
        xNouv= (xAnc+(a/xAnc))/2
    return xNouv
```

Les chaînes de caractères

- actions sur les chaînes de caractères :
- accès à un élément de la chaîne (un caractère) : $\text{ch}[5]$
- longueur d'une chaîne : $\text{len}(\text{ch})$
- concaténation de chaînes : $\text{ch} = \text{ch1} + \text{ch2}$
- conversion en nombre : $\text{int}(\text{ch})$, $\text{float}(\text{ch})$
- NB : les chaînes ne sont pas modifiables

Itérations sur les chaînes

- 1^{er} exemple, afficher une chaîne à l'envers :

```
# afficheEnvers : chaîne -> vide
#
#      ch
def afficheEnvers (ch) :
    """affichage à l'envers la chaîne ch"""
    i = len(ch) - 1
    while i >= 0 :
        print(ch[i],end="")
        i=i-1
```

```
afficheEnvers("ceci est un test")
```

81

Itérations sur les chaînes

- 2^e exemple : construire la chaîne inverse de celle en para :

```
#renverse : chaîne -> chaîne
#
#      ch ->
def renverse (ch):
    """rend le contenu de la chaîne ch renversé"""
    chRes = ""
    i = len(ch) - 1
    while i >= 0 :
        chRes=chRes+ch[i]
        i=i-1
    return chRes
```

Itsic – Univ Rennes

82