

# Informatique fondamentale

## Structures de contrôle

R.Gosswiller

1 Structures de contrôle

2 Itératif et récursif

3 Boucles

4 Les tours de Hanoï

# Structures de contrôle

# Structure de contrôle

## Définition

Une structure de contrôle est une instruction de branchement du code

## Principe

Point de déviation du code

Interruption du flux d'exécution

# Structures de contrôle

## Exemples de structures de contrôle

- Appel de fonctions
- Instructions conditionnelles
- Boucles
- Instructions différées
- Exceptions

# Instructions différées

## Principe

Les instructions différées sont des points de branchement dans le code permettant de jongler d'une adresse à une autre

Instructions go / goto

## Exemple

```
1          Label A :  
2              FAIRE ACTION  
3              ALLER B  
4  
5          Label B : FAIRE ACTION  
6              SI CONDITION VRAIE  
7                  ALLER A
```

# Instructions différées

## Inconvénients

Code "spaghetti"

Peu intuitif

## Solutions

Recourir à des boucles

# Boucles

## Définition

Une boucle est une structure logique permettant de conditionner la répétition d'une instruction ou d'une séquence d'instructions

## Détails

A l'opposé du récursif

Définition syntaxique dédiée (mot-clés)

Modèle universel



# Boucles

Il existe plusieurs types de boucles

## Boucle POUR

Réitération d'une tâche d'après un ensemble de valeurs ou d'éléments

Exemple : "Pour chaque chat, compter une moustache"

## Boucle TANTQUE

Réitération d'une tâche tant qu'une condition est vraie

Exemple : "Tant qu'il y a de l'essence, roule"

# Boucle infinie

## Principe

Une boucle infinie est une boucle dont la condition d'arrêt n'est jamais satisfaite

Les paramètres ne convergent pas vers une satisfaction de la condition d'arrêt

Cas d'erreur courant en programmation/Algorithmique

# Itératif et récursif

# Mécanisme itératif

Problématique

Comment effectuer un traitement de manière répétée ?

Solution

Employer des mécanismes récursifs et itératifs

# Itératif et récursif

## Limites du récursif

- Approche mathématique
- Peu orienté fonctionnel

## Solution

Faire appel à des mécanismes itératifs

# Itératif

## Définition

L'itératif est la répétition conditionnée d'un ensemble d'instructions.

## Détails

Par convention, on oppose itératif et récursif

Récursif : appel mutuel et successif de fonctions

Itératif : boucles conditionnées

# Conditions d'arrêt

## Principe

Tout mécanisme récursif ou itératif doit être implémenté avec une condition d'arrêt

C'est cette condition qui détermine l'exécution ou non de la tâche.

## Récursif

En récursif, la condition d'arrêt va être basée sur la valeur des arguments d'entrée

## Itératif

En itératif, la condition d'arrêt est définie explicitement au début de la boucle

# Conditions d'arrêt

## Exemple

```
1      TANT QUE i < 10
2          j = j * i
3          i = i + 1
```



# Boucles

# Range

## Principe

Génération d'une liste d'entiers

## Détails

Ensemble de valeurs

Conçu pour les itérations

Conditions d'itération

Type range

# Range

## Syntaxe

```
1  range([start], stop[, step])  
2  
3  range(5) -> [0,1,2,3,4]  
4  range(1,4) -> [1,2,3]  
5  range(0,10,2) -> [0,2,4,6,8]
```

Paramètres entiers (pas de chaîne)

Démarrage à 0

Fonction xrange : Python 2

# La boucle for

## Principe

La boucle for permet d'implémenter une répétition sur un nombre de fois donné en paramètre

La valeur de départ et la condition d'arrêt sont fixées

## Syntaxe

```
1         for i in range(init,limit):  
2             do()
```

# La boucle for

## Principe

Nombre d'itérations connu

Variable, constante

## Syntaxe

```
1         for var in ensemble:  
2             perform_action()
```

# La boucle for

## Exemples

```
1         for i in range(0,10):  
2             print i  
3  
4         for i in range(5):  
5             print i
```

# La boucle while

## Principe

La boucle while permet d'itérer tant qu'une condition est vraie  
Cela conditionne l'exécution d'une tâche à la validation d'une condition

## Syntaxe

```
1      while Condition :  
2          do ()
```

# La boucle while

## Exemples

```
1         while doStuff()==val:
2             doLoop()
3
4         while i < 10:
5             print i
6             i = i +1
7
8         while cond==True:
9             print "Validity"
10            i = i+1
11            if (i%50 == 0):
12                break;
```



# La boucle forEach

## Principe

La boucle `forEach` permet d'itérer sur le contenu d'un ensemble  
Alternative à la boucle `for`  
Itération sur les éléments (et non sur un compteur)

## Syntaxe

```
1         for elt in ens:  
2             do()
```

# La boucle forEach

## Exemples

```
1      l = ["a","b","c"]
2      for i in l:
3          print i
4
5      for i in ["a","b","c"]:
6          print i
7
8      for i in dico.keys():
9          print dico[i]
```

# Le mot-clé break

## Principe

Le mot-clé **break** permet de forcer l'arrêt d'une boucle  
Il s'agit d'une instruction de sortie

## Syntaxe

```
1         while(getEssence() > 0):  
2             rouler()  
3             if(mur==True):  
4                 break
```

# Les tours de Hanoï

# Les tours de Hanoï

## Problème

On suppose 3 piquets, numérotés 1,2,3. Sur le piquet 1 sont empilés  $n$  disques, chaque disque  $i$  étant plus petit que les  $i - 1$  premiers.

On souhaite disposer les  $n$  disques sur le piquet 3, sans jamais déposer un disque plus grand sur un disque plus petit, et en n'en déplaçant qu'un à la fois.

## Analyse

Se ramener à une solution simple

Déplacements élémentaires successifs

# Les tours de Hanoï

## Solution itérative

```
1 def hanoi(n):
2     i = 1; d = 1; a = 2
3     if(n%2 == 0):
4         a = 3
5     for j in range(2, 2^n - 1):
6         if(n%2 == 0):
7             k = v_2(j)
8             i = k + 1
9             if(k%2 == 0):
10                (d, a) = (6 - d - a, d)
11            else:
12                a = 6 - d - a
13        else:
14            i = 1
15            if(k%2 == 0):
16                (d, a) = (6 - d - a, d)
17            else:
18                d = 6 - d - a
19
20    print("i" + d + "->" + a)
```

# Les tours de Hanoï

## Complexité

$$c(n+1) = 2 * c(n) + 1$$

$$c(n) = 2^n - 1$$

# Les tours de Hanoï

Décomposition en sous-tâches

Solution récursive

Déplacer  $n$  disques de  $d$  à  $a$  :

Déplacer  $n - 1$  disques de  $d$  à  $i +$

Déplacer 1 disque de  $d$  à  $a +$

Déplacer  $n - 1$  disques  $i$  à  $a$



# Les tours de Hanoï

## Solution récursive

```
1 def hanoi(n,d,i,a):  
2     if(n==1):  
3         print(d+"->" +a)  
4     else:  
5         h(n-1,d,a,i)  
6         h(1,d,i,a)  
7         h(n-1,i,d,a)
```

## Complexité

$$c(n+1) = 2 * c(n) + 1$$

$$c(n) = 2^n - 1$$

# Les tours de Hanoï

## Conclusion

Même complexité

Modélisation fonctionnelle plus simple en récursif

# Synthèse

- Boucles for, while
- Mécanismes itératif/récuratif
- Fonction range
- Conditions et branchements