

Algorithmique et Programmation (TALA330A... L3 INALCO)

Crédits : M-Anne Moreaux - INALCO, et G. Galisot - Université de Tours

Caroline Parfait
caroline.parfait@sorbonne-universite.fr

2021-2022

Obtic, Sorbonne Center for Artificial Intelligence
STIH EA 4509, Sorbonne Université

1. Les entrées

2. Flux d'exécution

Instructions conditionnelles (if...elif...else)

Instructions répétitives - Boucles

Les entrées

Les objets présents dans l'espace de travail peuvent être :

- **Simple** : une valeur entière, une valeur décimale, un caractère... On les appelle les *scalaires*.
- **Complexes et décomposables en unités plus petites** : collections d'objets dénotés par un seul nom (ensemble (set), liste de valeurs entières, de valeurs décimales, de caractères, listes de chaînes de caractères...)

Lorsque le programme exécutable est en cours d'exécution :

- Les objets **Simples** : indécomposables occupent un seul emplacement mémoire et ne correspondent qu'à une seule adresse.
- Les objets **Complexes et décomposables en unités plus petites** : occupent plusieurs emplacements mémoire, chaque élément simple ayant un adresse.

Durant l'exécution du programme ces objets peuvent :

- voir leur valeur **modifiée** : **Variables**
- **conserver** la même valeur : **Constantes**

Variables et Constantes

	Variable	Constante
Un nom	<i>Identificateur</i>	<i>Identificateur</i> \emptyset nom : <i>constante littérale</i>
valeur durant exe.	état changeant	état immuable
mémoire machine	une variable est une zone de la mémoire dans laquelle une valeur peut être lue ou écrite durant l'exécution du programme	le contenu de la zone mémoire est fixé une fois pour toute. L'état de la zone mémoire ne peut pas être modifié durant l'exécution du programme.

Variables et Constantes

	Variable	Constante
Type	Type changeant (il faut faire attention!!!)	Une constante littérale dénote directement une valeur dont la forme définie Le type. (ex : <i>125</i> , une suite de chiffres ; <i>+10 -32</i> une suite de chiffres précédée de + ou - ; <i>'B'</i> un et un seul caractère entouré de ' ' ; <i>True, False</i> , des booléens)

Flux d'exécution

- **Séquence d'instructions :**

****** Les instructions d'un programme s'exécutent séquentiellement (les unes après les autres dans l'ordre dans lequel elles ont été écrites)

MAIS

- **Instructions composées – blocs d'instructions**

****** *Sous Python, les **blocs d'instructions** ont toujours la même **structure** : une ligne d'en-tête terminée par un **double point**, suivie d'une ou de plusieurs instructions **indentées** sous cette ligne d'en-tête*

Exemple, **bloc d'instruction** :

Ligne d'en-tête : Ne pas oublier double point, puis indentation

première instruction du bloc

... ..

... ..

dernière instruction du bloc

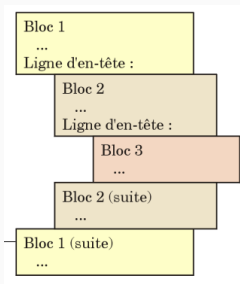
Instruction suivante

Contrôle du flux d'exécution

- **Instructions composées – blocs d'instructions**

La ligne d'en-tête : contient toujours une instruction bien spécifique permettant de contrôler/modifier le flux d'exécution

Commentaires : # sont ignorés



Attention : toutes les lignes d'un même bloc doivent être indentées exactement de la même manière (c'est-à-dire décalées vers la droite d'un même nombre de tabulations)

2. Flux d'exécution

Instructions conditionnelles (if. . . elif. . . else)

Instructions répétitives - Boucles

- Instructions conditionnelles (if...elif...else)

Le bloc : d'instruction est exécuté sous certaines conditions

Si <condition> Alors

<bloc>

Sinon

<bloc>

Finsi

```
if <condition> :
```

```
    _____<bloc1>
```

```
<bloc2>
```

Les instructions : → du bloc1 sont exécutées si la condition est vérifiée.

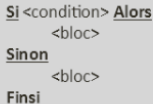
→ du bloc2 sont toujours exécutées.

L'indentation (« ____ ») les distingue du bloc1.

Contrôle du flux d'exécution

- Instructions conditionnelles (if...elif...else)

Le bloc : d'instruction est exécuté sous certaines conditions



Si <condition> Alors
 <bloc>
Sinon
 <bloc>
Finsi



```
if <condition> :  
    _____<bloc1>  
else :  
    _____< bloc2>  
    _____<bloc3>
```

Les instructions : → du bloc1 sont exécutées si la condition est vérifiée.

→ Sinon celles du bloc2 sont exécutées.

→ du bloc3 sont toujours exécutées.

Contrôle du flux d'exécution

- Instructions conditionnelles (if...elif...else)

Le bloc : d'instruction est exécuté sous certaines conditions

```
Si <condition> Alors  
    <bloc>  
Sinonsi <condition>  
    <bloc>  
Sinon  
    <bloc>  
Finsi
```

```
if <condition1> :  
    _____<bloc1>  
elif <condition2>:  
    _____<bloc2>  
else :  
    _____<bloc3>  
    <bloc4>
```

Les instructions : → du bloc1 sont exécutées si la condition1 est vérifiée.

→ Si la condition1 n'est pas vérifiée et si la condition2 l'est, les instructions du bloc2 sont exécutées.

→ Si aucune des conditions ne sont vérifiées, les instructions du bloc3 sont exécutées.

→ du bloc4 sont toujours exécutées.

2. Flux d'exécution

Instructions conditionnelles (if...elif...else)

Instructions répétitives - Boucles

- **Instructions répétitives**

- Répétitions en boucle – L'instruction while
- Une boucle while permet de répéter un certain nombre de fois (voire indéfiniment) une série d'instruction

```
While (condition) :           #Tant_que la condition est vraie  
    bloc d'instructions      # attention à l'indentation  
#suite du programme
```

Exemple :

```
a = 0  
while (a < 10):               # (n'oubliez pas le double point !)  
    a = a + 1                 # (n'oubliez pas l'indentation !)  
    print(" a vaut : " , a)  
#fin de la boucle while
```

Contrôle du flux d'exécution

- **Répétitions en boucle – L'instruction while**

- La variable évaluée dans la condition doit exister au préalable (il faut qu'on lui ait déjà affecté au moins une valeur)
- Si la condition est fausse au départ, le corps de la boucle n'est jamais exécuté.
- Il faut donc veiller à ce que le corps de la boucle contienne au moins une instruction qui change la valeur d'une variable intervenant dans la condition évaluée par while, de manière à ce que cette condition puisse devenir fausse et la boucle se terminer.

Exemple de boucle sans fin (à éviter !) :

`n = 3`

`while n < 5 :`

`print("hello !")`

- **Répétitions en boucle – L'instruction For**
 - Une boucle For permet de répéter un nombre de fois connu à l'avance une série d'instruction
 - Les éléments de la séquence sont issus d'une chaîne de caractères ou bien d'une liste

Exemple :

```
liste = ['Pierre',67.5,18]
```

```
for elmt in liste :
```

```
    print (elmt)      # elmt est la variable d'itération
```

```
print ('Fin de la boucle')
```

- **Répétitions en boucle – L'instruction For**
- Si le nombre d'itérations à effectuer est connu, il faut utiliser de préférence une boucle for.
- On utilise while dans les autres cas
- For est souvent associé avec la fonction range() pour créer des séquences automatiques de nombres entiers

Exemple :

```
for i in range(1,5) :    # Attention range(1,5) renvoie [1,2,3,4]
    print (i)
print ('Fin de la boucle')
```

Contrôle du flux d'exécution

- **Instructions répétitives - break et continue**
 - L'instruction break permet de sortir directement d'un bloc d'instruction sans condition (sortie d'une boucle)
 - L'instruction continue permet de passer à l'itération suivante dans une boucle (sans passer par les instructions qui suivent dans le bloc en cours)

Exemple :

```
import time      # importation du module time
while True :
    # strftime() est une fonction du module time
    print ('Heure courante ', time.strftime('%H : %M : %S'))
    quitter = input('Voulez-vous quitter le programme (o/n) ? ')
    if quitter == 'o' :
        break
    print 'A bientôt'
```

Contrôle du flux d'exécution

- Pas toujours facile à suivre. . .

```
# Suite de Fibonacci
```

```
a, b, c = 1, 1, 1
```

```
while c < 11 :
```

```
    print(b, end = " ")
```

```
    a, b, c = b, a+b, c+1
```

- Une représentation des variables est parfois fort utile.
- CF debuggateur

Variables	a	b	c
Valeurs initiales	1	1	1
Valeurs prises successivement, au cours des itérations	1	2	2
	2	3	3
	3	5	4
	5	8	5

Expression de remplacement	b	a+b	c+1