

Cours 4

ALTERNATIVE GÉNÉRALISÉE

Contenu du cours

- Retour sur le dernier cours
- Alternative généralisée
- Imbrication d'alternatives
- Laboratoire #3

Retour sur le dernier cours

- Alternative de base
 - Évaluation d'une condition (type booléen)
 - Bloc d'instruction A si c'est *True*
 - Bloc d'instruction B si c'est *False*
 - Indentation du code pour délimiter les blocs d'instructions
- Opérateurs de comparaison
 - Est-ce que deux valeurs sont égales/différentes?
 - Est-ce que l'une est supérieure/inférieure à l'autre?
- Opérateurs logiques
 - « and », « or », « not »
 - Utilisé pour évaluer plusieurs conditions
 - Table de vérité

Alternative généralisée

Définition

Rappel sur l'alternative de base

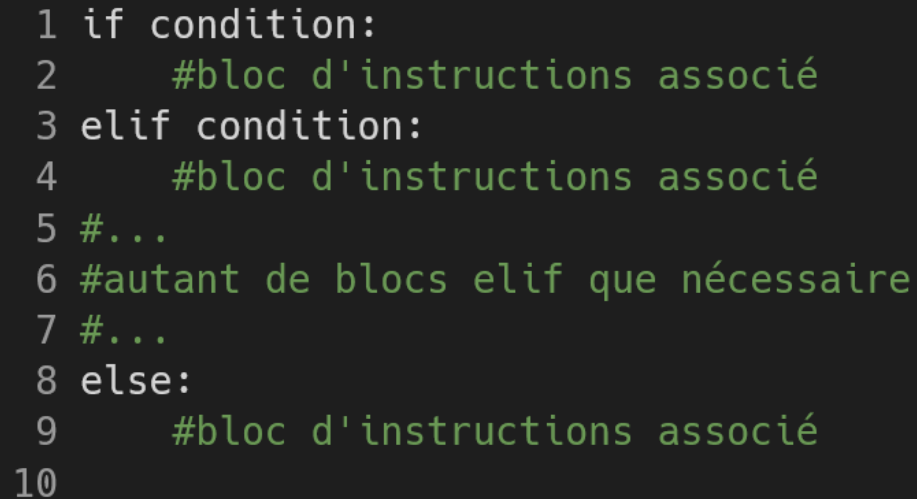
En programmation, une structure d'alternative, également connue sous le nom de structure conditionnelle, est un mécanisme de contrôle du flux d'exécution qui permet à un programme de prendre des décisions en fonction de certaines conditions. Cela signifie que le programme peut choisir d'exécuter ou non un bloc de code spécifique en fonction de la valeur d'une ou plusieurs expressions logiques.

L'alternative généralisée

Il existe une autre forme de l'alternative: l'alternative généralisée. Dans cette forme, on ne se limite plus à n'avoir qu'un bloc "SI" et un bloc "SINON", mais plutôt une séquence de conditions qui prendra plutôt la forme: SI - SINON SI - SINON. Il est à noter qu'une telle alternative peut comporter autant de clauses "SINON SI" que nécessaire.

Syntaxe

En python, l'alternative généralisée utilise les mots réservés `if`, `elif` (contraction de else if) et `else`, respectivement SI, SINON SI et SINON. Elle prend donc la forme:



```
1 if condition:
2     #bloc d'instructions associé
3 elif condition:
4     #bloc d'instructions associé
5 #...
6 #autant de blocs elif que nécessaire
7 #...
8 else:
9     #bloc d'instructions associé
10
```

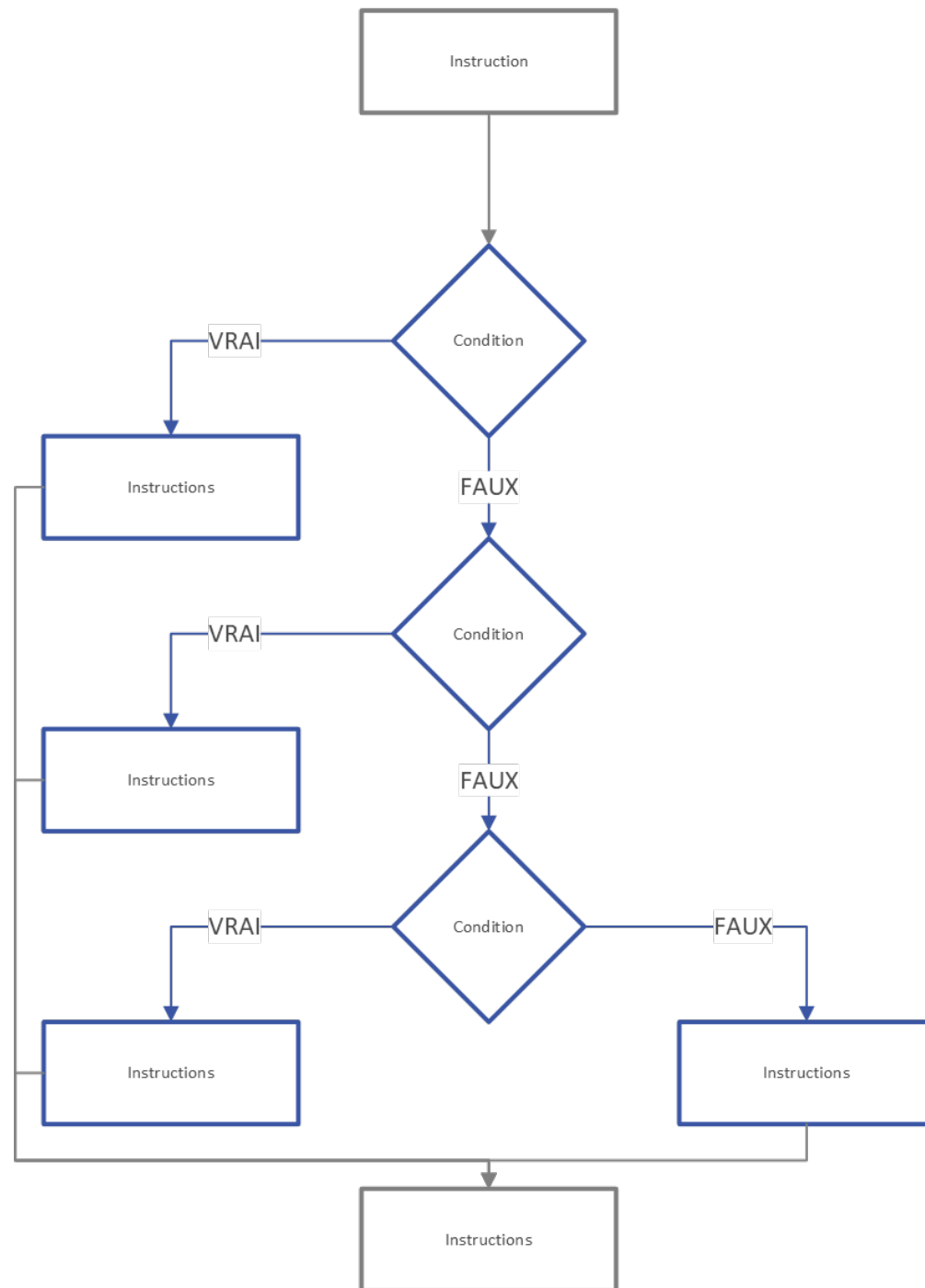


Schéma:

SI –
SINON SI –
SINON SI –
SINON

Principe

Le schéma précédent nous montre qu'aussitôt qu'une des conditions de l'alternative est vraie, on exécute le bloc d'instructions qui y est rattaché et on poursuit l'exécution à la première instruction qui suit immédiatement la fin de l'alternative (donc après le dernier SINON-SI ou après le SINON).

Mise en œuvre

Supposons que l'on veuille écrire un programme qui permet de transformer une note chiffrée en "cote" (A, B, C, etc...).

Sachant que l'on donne:

- A à quiconque a une note supérieure à 90
- B pour une note supérieure à 80
- C pour une note supérieure à 70
- D pour une note supérieure à 60
- E sinon

Solution imparfaite:



```
1 note = 83
2 if (note > 90):
3     print("Vous avez obtenu un A")
4 if (note > 80):
5     print("Vous avez obtenu un B")
6 if (note > 70):
7     print("Vous avez obtenu un C")
8 if (note >= 60):
9     print("Vous avez obtenu un D")
10 else:
11     print("Vous avez obtenu un E")
12
```

Résultat:



```
1 Vous avez obtenu un B
2 Vous avez obtenu un C
3 Vous avez obtenu un D
```

Mise en œuvre

Ceci est dû au fait que chaque `if` est évalué indépendamment des autres. Ainsi, après avoir exécuté le bloc d'instructions associé à la condition `if (note > 80)`, on évalue ensuite la condition `if (note > 70)` ; comme la note est de 83, cette condition s'avère être également vraie, et on exécute alors le bloc d'instructions qui y est rattaché. La même séquence d'événements se reproduit donc avec tous les autres `if` de l'alternative.

On peut corriger le problème observé avec la solution précédente simplement en bornant mieux chacune des conditions. On obtient alors le programme suivant:

Solution:



```
1 note = 83
2 if (note > 90):
3     print("Vous avez obtenu un A")
4 if (note <= 90) and (note > 80):
5     print("Vous avez obtenu un B")
6 if (note <= 80) and (note > 70):
7     print("Vous avez obtenu un C")
8 if (note <= 70) and (note > 60):
9     print("Vous avez obtenu un D")
10 else:
11     print("Vous avez obtenu un E")
```

Résultat:



```
1 Vous avez obtenu un B
```

Mise en œuvre

La dernière sortie correspond au résultat que l'on attend du programme.

Comme chaque intervalle est maintenant mieux défini, on évite que chacune des conditions qui suivent la première qui est évaluée à `True` soit elle aussi déclenchée.

Voyons maintenant à quoi pourrait ressembler le même programme si on utilise une alternative généralisée:

Solution:



```
1 note = 83
2
3 if (note > 90):
4     print("Vous avez obtenu un A")
5 elif (note > 80):
6     print("Vous avez obtenu un B")
7 elif (note > 70):
8     print("Vous avez obtenu un C")
9 elif (note > 60):
10    print("Vous avez obtenu un D")
11 else:
12    print("Vous avez obtenu un E")
```

Résultat:



```
1 Vous avez obtenu un B
```

Résultat

On remarque que chaque condition a repris sa forme plus simple, mais que le programme produit malgré tout le bon résultat. Ceci tient au fait que l'instruction `elif` fait en sorte qu'aussitôt qu'une condition est évaluée à `True`, l'ensemble des conditions qui suivent sont ignorées, et que l'exécution se poursuit à la ligne suivant l'instruction `else`.

Alternative imbriquée

Contexte

Dans certaines situations, on doit organiser la prise de décision en plus d'une étape. Par exemple, imaginons que l'on souhaite écrire un programme qui affiche le type d'acide ou de base d'une solution en fonction du PH observé: c'est-à-dire, déterminer s'il s'agit d'une solution neutre, acide ou basique, tout en spécifiant si l'on est face à un acide/base fort ou faible.

On obtient alors le programme suivant:



```
1 valeurPH = float(input("Veuillez entrer la valeur de PH observée: "))
2
3 if (valeurPH < 7.0): #On a affaire à un acide
4     message = "Votre solution est un acide "
5
6     if (valeurPH < 3.0): #Acide fort
7         message = message + "fort (ph < 3.0)"
8     else:
9         message = message + "faible (3.0 <= ph < 7.0)"
10
11 elif (valeurPH == 7.0): #solution neutre
12     message = "Solution neutre"
13
14 else: #On a affaire à une base
15     message = "Votre solution est une base "
16
17     if (valeurPH > 11.0): #base forte
18         message = message + "forte (ph > 11.0)"
19     else:
20         message = message + "faible (7.0 < ph <= 11.0)"
21
22 print(message)
```

Contexte

On peut donc voir que la composition du message à afficher s'effectue en deux temps. La première structure d'alternative rencontrée détermine d'abord le type de solution (base, neutre ou acide), ce qui permet de commencer la composition du message.

La seconde partie du message est ensuite déterminée par une seconde alternative qui permet de qualifier la force de l'acide ou de la base. Le message ainsi assemblé est finalement affiché à la toute dernière ligne du programme.

Laboratoire #3