STRUCTURES CONDITIONNELLES

1 BRANCHEMENT CONDITIONNEL

A Test simple

La structure avec `if` est la traduction (littérale) de la structure conditionnelle mathématique :

Si *condition,*	**if** `condition`:
Alors *instructions si la condition est vraie,*	`bloc d'instructions si la condition est vraie,`
Sinon *instructions si la condition est fausse.*	**else**:
	`bloc d'instructions si la condition est fausse.`
suite du programme en dehors de la boucle	`suite du programme en dehors de la boucle`

*Le "**alors**" (**then**) est sous-entendu. Il ne faut pas l'écrire.*

```
age = int(input('Entrez votre âge '))
if age<18:
  print('Vous être mineur')
  print('Dans moins de '+str(18-age)+' ans, vous serez majeur')
else:
  print('Vous êtes majeur')
```

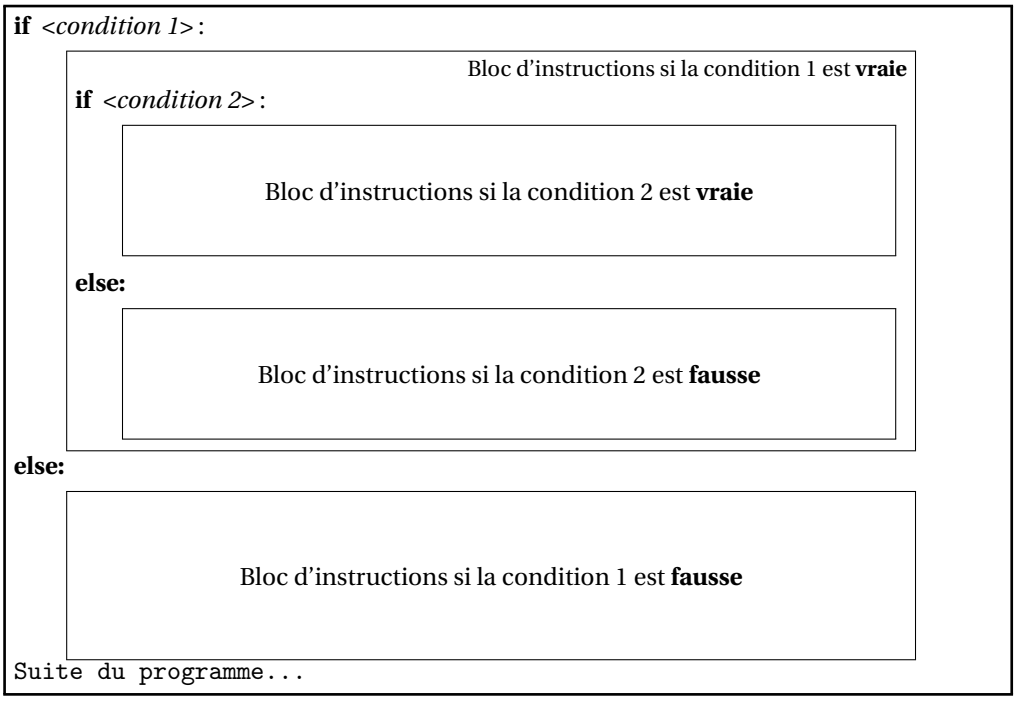
Python 1: Exemple test if

Les **indentations**[1] déterminent les blocs. La partie **else** est optionnelle. Lorsque Python a fini les instructions du bloc qu'il doit exécuter, il passe à la suite du programme.

B Imbrication des blocs

Dans le programme, chaque bloc déterminé par un niveau d'indentation constitue une entité autonome.

Ainsi, dans un bloc **if**, on peut intégrer un autre branchement conditionnel sans problème, et ainsi de suite.



Si toute la structure conditionnelle précédente doit être intégrée au sein d'une fonction, alors, il suffit d'indenter tout ce bloc et de le faire précéder de la commande : **def**.

[1] L'indentation est le fait de décaler la ligne par rapport à la marge. Pour un bloc donné, on s'assurera que toutes les lignes ont la même indentation : le même décalage.

def *<nom fonction>*(*<arguments>*):
    if *<condition 1>* :
        Bloc d'instructions si la condition 1 est **vraie**
        if *<condition 2>* :
            Bloc d'instructions si la condition 2 est **vraie**
        else:
            Bloc d'instructions si la condition 2 est **fausse**
    else:
        Bloc d'instructions si la condition 1 est **fausse**
    Suite du programme...
Suite du programme...

Ce fonctionnement par blocs est très explicite dans le langage AlgoBox.

Astuce Pyzo : chaque "deux points" ":" annonce une indentation. Lorsque l'on tape les deux points dans Pyzo, celui-ci indente automatiquement la ligne suivante.
Pour indenter tout un paquet de lignes à la fois, on les sélection et on appuie sur `Tab`, la touche tabulation (touche à gauche de la lettre *A* sur un clavier). Pour supprimer une indentation, on sélectionne et on appuie sur `Maj-Tab`.

C Utilisation de elif

Il est possible de spécifier plusieurs alternatives dans un branchement conditionnel. Cela peut éviter d'imbriquer trop de branchements les unes dans les autres. On utilise pour cela la commande **elif** autant de fois que nécessaire. Cette commande se traduit par "sinon, si...".

```
if expressionBooleenne:          #si..., faire
  suite d'instructions
  suite d'instructions
  ...
elif expressionBooleenne2:     #sinon, si..., faire
  suite d'instructions
  suite d'instructions
  ...
elif expressionBooleenne3:     #sinon, si..., faire
  suite d'instructions
  suite d'instructions
  ...
else:                            #sinon,
  suite d'instructions
  suite d'instructions
  ...
suite du programme en dehors du branchement
```

Python 2: syntaxe avec elif

Dès qu'une expression booléenne renvoie `True`, le bloc d'instructions correspondant est exécuté puis le programme sort du test (sans vérifier les conditions suivantes).

Si aucune condition n'est vérifiée, alors le programme exécute le bloc d'instruction de **else**.

```python
def parite(nb):
  if type(nb) != int:
    print("Un nombre entier ! espèce de #&!@*#!")
  elif nb%2 == 0:
    print("Le nombre choisi est pair")
  else:
    print("Le nombre choisi est impair")
```

Python 3: Utilisation de elif

Remarque : cette fonction ne renvoie rien (ou `none`), elle se contente d'afficher un texte à l'écran.

Exercice : quelle est l'erreur dans le programme qui suit ?

```python
def parite(nb):
  if type(nb) != int:
    print("Un nombre entier ! espèce de #&!@*#!")
  elif nb%2 == 0:
    print("Le nombre choisi est pair")
  elif nb == 0:
    print("Le nombre choisi est nul")
  else:
    print("Le nombre choisi est impair")
```

Python 4: Utilisation de elif, erreur dans l'ordre des instructions

Solution :
La condition `nb==0` ne sera jamais testée, car si `nb` vaut zéro, alors la condition `nb%2==0` renverra `True`, le programme affichera que le nombre est pair puis sautera les instructions suivantes. Si on veut que le cas `nb==0` soit testé à part, il faut mettre cette condition plus tôt.

```python
def parite(nb):
  if type(nb) != int:
    print("Un nombre entier ! espèce de #&!@*#!")
  elif nb == 0:
    print("Le nombre choisi est nul")
  elif nb%2 == 0:
    print("Le nombre choisi est pair")
  else:
    print("Le nombre choisi est impair")
```

Python 5: Utilisation de elif, correction de l'erreur dans l'ordre des instructions

D Comportement particulier de la commande return

Exemple (Maximum d'un couple)

Voici une fonction qui donne le maximum de deux nombres (sans utiliser la fonction max).

```python
def maximum(a,b):
  if a>b:
    return a
  else:
    return b
```

Python 6: Maximum d'un couple

Essayez de comprendre pourquoi la deuxième fonction `maximum2` marche aussi.

```python
def maximum2(a,b):
  if a>b:
    return a
  return b
```

Python 7: Maximum d'un couple

Nous avions vu que lorsque Python rencontre "**return**", il renvoie le résultat et sort de la fonction, quelque soit ce qui est écrit après.
Dans notre cas,

- si $a > b$, il réalise l'instruction "**return** `a`" puis sort de la fonction.
- sinon, il sort de la structure conditionnelle et suit l'instruction suivante : `return b`.

Cette fonction renvoie donc bien le maximum.

Exercice

Faites la même chose pour le maximum de trois nombres.

Remarques pour les pros : Lorsque le branchement conditionnel est simple, il est possible d'utiliser une écriture compacte qui correspond à ce que l'on dirait en français.

```
def maximum(a,b):
  return a if a>b else b
```

2 BOUCLE WHILE

Syntaxe et utilisation

La boucle `while` consiste à réaliser des instructions tant qu'une assertion est vraie.

Tant que *condition est vraie,*	**while** `condition`:
Faire *instructions,*	`bloc d'instructions si condition est vraie,`
Fin	
suite du programme	`suite du programme`

Lorsque l'assertion est fausse, le programme saute le bloc indenté.

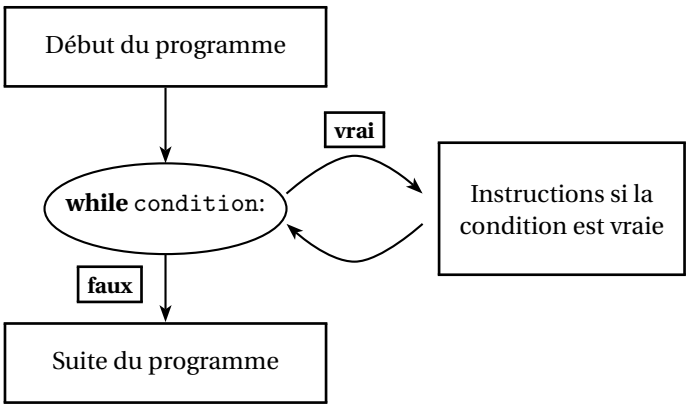


Figure 1: Fonctionnement de la boucle while

Un premier exemple : voici un programme qui compte jusqu'à 10.

```
i = 1
while i <= 10:
  print(i)
  i += 1
print('fini !')
```

Python 8: Boucle while : compter jusqu'à 10

Voici le déroulement du programme :

- `i` prend la valeur 1
- `i<=10` est vrai, donc les instructions de la boucle sont effectuées
  - Affiche `i` : 1
  - Incrémente `i` de 1 : `i=1+1=2`
- Les instructions de la boucle sont terminées, on teste à nouveau la condition `i<=10`. Comme `i=2`, l'assertion est vraie et les instructions de la boucle sont effectuées.
  - Affiche `i` : 2
  - Incrémente `i` de 1 : `i=2+1=3`
- L'assertion est à nouveau testée, elle vaut `True`...
- Et ainsi de suite, jusqu'à ce que `i` vaille 11, alors l'assertion `i<=10` vaut `False` et le programme saute les instructions de la boucle.
- Le programme affiche "fini !"

Pour complexifier, on peut créer une fonction pour compter jusqu'à n (pas forcément 10). On mettra donc la boucle `while` dans une fonction, ce qui revient à indenter tout le bloc.

```
def compter(n):
  i = 1
  while i <= n:
    print i
    i += 1
  print('fini !')
```

Python 9: Boucle while : fonction compter jusqu'à n

Attention aux erreurs

Pour que le programme termine (que l'on sorte un jour de la boucle **while**, il faut que la condition évolue à chaque tour par l'intermédiaire d'une variable que l'on modifie[2]. Dans l'exemple précédent, c'est la variable `i` que l'on incrémente. Si on oublie la ligne correspondante, alors la boucle ne termine jamais

Astuce Pyzo : Lorsque vous testez vos fonctions, il est possible de forcer Python à s'arrêter en cas de boucle infinie (bouton en forme d'éclair au dessus du shell sur Pyzo).

Remarque pour les pros : Pour démontrer mathématiquement qu'une boucle termine, on définit un *variant de boucle*. À chaque occurrence de la boucle, ce variant doit augmenter d'au moins 1 et on doit prouver que la boucle s'arrête lorsque l'incrément prend une valeur supérieure à un certain nombre fixé.
Dans l'exemple (celui qui fonctionne), l'incrément est simplement i.
Mauvaise nouvelle : le théorème de Rice énonce qu'il n'existe pas d'algorithme qui puisse prouver la terminaison de n'importe quel programme.

Exemple

Indiquez ce que fait l'algorithme suivant.
La terminaison de cet algorithme n'a pas été prouvé. Elle est connue sous le nom de conjecture de Syracuse.

```
def syracuse(nb):
  while nb != 1:
    if nb%2 == 0:
      nb = nb/2
    else:
      nb = 3*nb+1
```

Python 10: Conjecture de Syracuse

[2] Il existe une exception avec l'utilisation de **return**.

Astuce de pros : Lorsque l'on met une boucle **while** dans une fonction, on peut arrêter la boucle par la commande **return**.
Par exemple (programme très maladroit, mais permet de comprendre) :

```
# renvoie le nombre de coups pour trouver a
def trouver(a):
  b = 1
  n = 0
  while True:
    n += 1
    if b == a:
      return n
    if b < a:
      b = b+1
    else:
      b = b-1
```