Algorithmique et Programmation

Instructions conditionnelles

Elise Bonzon

elise.bonzon@mi.parisdescartes.fr

LIPADE - Université Paris Descartes
http://www.math-info.univ-paris5.fr/∽bonzon/

Instructions conditionnelles

1. Séquence d'instructions
2. Algèbre de Boole
3. Les instructions conditionnelles simples
4. Les instructions conditionnelles multiples
5. Quelques erreurs courantes
6. Pour conclure

Séquence d'instructions

Séquence d'instructions

Séquence d'instructions

Les instruction d'une **séquence d'instructions** sont exécutées en séquence, les unes après les autres, dans l'ordre dans lesquelles elles apparaissent.

Séquence d'instructions

Séquence d'instructions

Les instruction d'une **séquence d'instructions** sont exécutées en séquence, les unes après les autres, dans l'ordre dans lesquelles elles apparaissent.

```
>>> a, b = 3, 7
>>> a = b
>>> b = a
>>> print(a, b)
7 7
```

Séquence d'instructions

Séquence d'instructions

Les instruction d'une **séquence d'instructions** sont exécutées en séquence, les unes après les autres, dans l'ordre dans lesquelles elles apparaissent.

```
>>> a, b = 3, 7
>>> a = b
>>> b = a
>>> print(a, b)
7 7
```

```
>>> a, b = 3, 7
>>> b = a
>>> a = b
>>> print(a, b)
3 3
```

Instructions composées

- Les séquences d'instructions sont indispensables, mais pas suffisantes
- Nécessité d'aiguiller le déroulement des programmes dans différentes directions, en fonction des circonstances rencontrées
- Ou encore de répéter plusieurs fois des instructions données (*mais ce sera vu au prochain cours*)

$\Rightarrow$ Instructions composées

Instructions composées

Pour identifier les instructions composées, Python utilise la notion d'indentation significative.

Instructions composées

Instruction composée et indentation significative

Une instruction composée se compose :

- d'une **ligne d'introduction** terminée par le caractère "deux-points" (:)
- d'un **bloc d'instructions** indenté par rapport à la ligne d'introduction.

On utilise par convention 4 espaces par indentation.

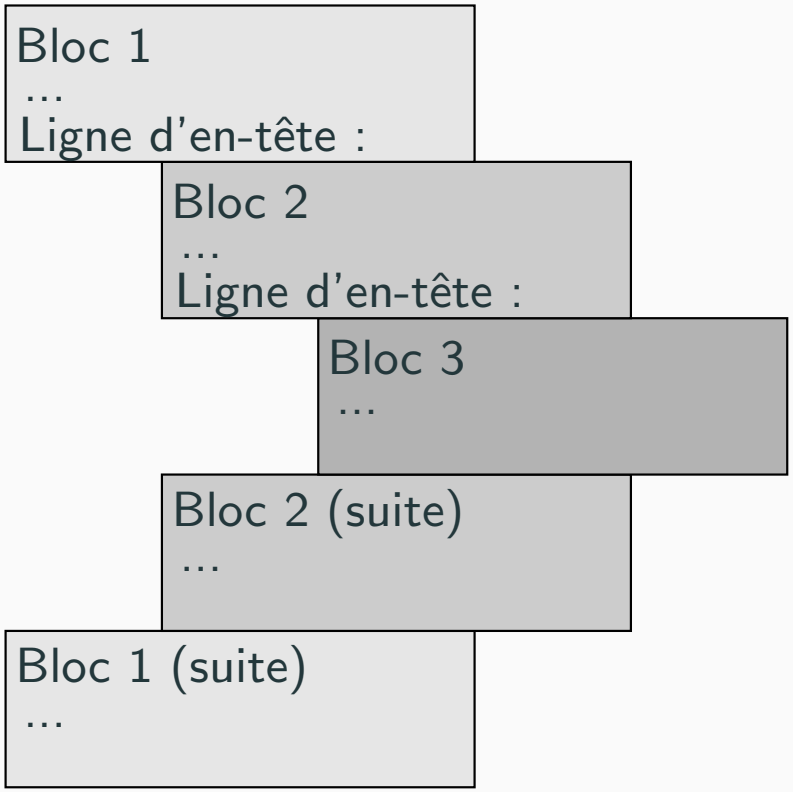
Indentation

```
ligne_introduction_bloc_instruction :
    sous-instruction1
    sous-instruction2
    sous-instruction3
instruction-suivante
```

- Le symbole `:` indique à Python qu'il doit attendre un bloc d'instructions
- Python reconnait un bloc d'instructions car il est indenté
- La ligne `instruction-suivante` n'étant pas indenté, Python sait qu'elle ne fait pas partie du bloc d'instructions précédent

Indentation

En Python, l'indentation est obligatoire ! La forme globale d'un programme doit ressembler à ce schéma :



Toutes les instructions au même niveau d'indentation appartiennent au même bloc d'instructions, jusqu'à ce que l'indentation diminue.

Espaces ou tabulation ?

Pourquoi utilise-t'on conventionnellement **4 espaces** plutôt qu'une **tabulation** ?

- L'indentation doit être homogène (soit des espaces, soit des tabulations, mais pas un mélange des deux)
  - Espaces et tabulations sont des codes binaires distincts
  - Si ils sont mélangés pour indenter les lignes d'un même bloc, le résultat parait identique à l'écran
  - Mais Python considérera que ces lignes indentées différemment font partie de blocs différents

  $\rightarrow$ Erreurs difficiles à déboguer
- En fonction des éditeurs de textes utilisés, les tabulations sont transformés par des espaces ou non, ce qui peut poser des problèmes en cas de travail collaboratif/sur plusieurs éditeurs différents

La plupart des programmeurs préfèrent donc se passer des tabulations !

Instructions conditionnelles

Instruction conditionnelle

Une instruction conditionnelle, ou alternative, est une instruction qui permet de **choisir** entre deux séquences d'instruction selon la valeur d'une **condition**.

Par exemple, trouver la valeur absolue d'un nombre :

$$|x| = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{sinon} \end{cases}$$

- On fait le **choix** entre deux calculs (x ou $-x$) selon une **condition** ($x \geq 0$)
- Il faut donc savoir évaluer une condition

Algèbre de Boole

Conditions

Condition

Une **condition** est une expression booléenne, ou booléen, de type `bool`, dont la valeur peut être `True` (vraie) ou `False` (faux).

- Une expression booléenne peut donc être n'importe quelle variable ayant une valeur `True` ou `False`
- Par exemple, les opérateurs de comparaison produisent un résultat de type `bool`

Les opérateurs de comparaison

i	==	j	Egalité
i	!=	j	Différence
i	<	j	Strictement inférieur
i	<=	j	Inférieur ou égal
i	>	j	Strictement supérieur
i	>=	j	Supérieur ou égal

Les opérateurs de comparaison

i	==	j	Egalité
i	!=	j	Différence
i	<	j	Strictement inférieur
i	<=	j	Inférieur ou égal
i	>	j	Strictement supérieur
i	>=	j	Supérieur ou égal

```
>>> 2 > 8
```

Les opérateurs de comparaison

i	==	j	Egalité
i	!=	j	Différence
i	<	j	Strictement inférieur
i	<=	j	Inférieur ou égal
i	>	j	Strictement supérieur
i	>=	j	Supérieur ou égal

```
>>> 2 > 8
False
```

Les opérateurs de comparaison

i	==	j	Egalité
i	!=	j	Différence
i	<	j	Strictement inférieur
i	<=	j	Inférieur ou égal
i	>	j	Strictement supérieur
i	>=	j	Supérieur ou égal

```
>>> 2 > 8
False
>>> 6 % 2 == 0          # 6 est un nombre pair
```

Les opérateurs de comparaison

i	==	j	Egalité
i	!=	j	Différence
i	<	j	Strictement inférieur
i	<=	j	Inférieur ou égal
i	>	j	Strictement supérieur
i	>=	j	Supérieur ou égal

```
>>> 2 > 8
False
>>> 6 % 2 == 0          # 6 est un nombre pair
True
```

Les opérateurs de comparaison

i	==	j	Egalité
i	!=	j	Différence
i	<	j	Strictement inférieur
i	<=	j	Inférieur ou égal
i	>	j	Strictement supérieur
i	>=	j	Supérieur ou égal

```
>>> 2 > 8
False
>>> 6 % 2 == 0          # 6 est un nombre pair
True
>>> 3 + 5 >= 10 - 2
```

Les opérateurs de comparaison

i	==	j	Egalité
i	!=	j	Différence
i	<	j	Strictement inférieur
i	<=	j	Inférieur ou égal
i	>	j	Strictement supérieur
i	>=	j	Supérieur ou égal

```
>>> 2 > 8
False
>>> 6 % 2 == 0          # 6 est un nombre pair
True
>>> 3 + 5 >= 10 - 2
True
```

Autres conditions

D'autres conditions sont possibles (liste non exhaustive) :

```
>>> "a" < "b"
```

Autres conditions

D'autres conditions sont possibles (liste non exhaustive) :

```
>>> "a" < "b"
True
```

Autres conditions

D'autres conditions sont possibles (liste non exhaustive) :

```
>>> "a" < "b"
True
>>> "ami" > "absent"
```

Autres conditions

D'autres conditions sont possibles (liste non exhaustive) :

```
>>> "a" < "b"
True
>>> "ami" > "absent"
True
```

Autres conditions

D'autres conditions sont possibles (liste non exhaustive) :

```
>>> "a" < "b"
True
>>> "ami" > "absent"
True
>>> "mort" in "immortel"
```

Autres conditions

D'autres conditions sont possibles (liste non exhaustive) :

```
>>> "a" < "b"
True
>>> "ami" > "absent"
True
>>> "mort" in "immortel"
True
```

Autres conditions

D'autres conditions sont possibles (liste non exhaustive) :

```
>>> "a" < "b"
True
>>> "ami" > "absent"
True
>>> "mort" in "immortel"
True
>>> len("ami") >= 4
```

Autres conditions

D'autres conditions sont possibles (liste non exhaustive) :

```
>>> "a" < "b"
True
>>> "ami" > "absent"
True
>>> "mort" in "immortel"
True
>>> len("ami") >= 4
False
```

Les opérateurs logiques

Opérateurs booléens de base

Le type `bool` admet 3 opérateurs logiques de base :

- La **négation** (non), notée `not` (opérateur unaire)
- La **conjonction** (et), noté `and` (opérateur binaire)
- La **disjonction** (ou), noté `or` (opérateur binaire)

Principe d'évaluation de la négation

`expression`	`not(expression)`
`True`	`False`
`False`	`True`

Principe d'évaluation de la négation

Evaluation de `expression`, et

- Soit la valeur est `True` , dans ce cas retourner `False`
- Soit la valeur est `False` , dans ce cas retourner `True`

Exemple de la négation

```
>>> not(150 > 100)
False

>>> felix = "chat"
>>> medor = "chien"
>>> not(felix == medor)
True
```

Principe d'évaluation de la conjonction

`expression1`	`expression2`	`expression1 and expression2`
`True`	`True`	`True`
`True`	`False`	`False`
`False`	`True`	`False`
`False`	`False`	`False`

Principe d'évaluation de la conjonction

Evaluation de `expression1`, et

- Soit la valeur est `True` , et dans ce cas
  - Evaluation de `expression2`, et
    - Soit la valeur est `True` , et dans ce cas retourner `True`
    - Soit la valeur est `False` , et dans ce cas retourner `False`
- Soit la valeur est `False` , dans ce cas retourner `False`

Attention ! `expression2` peut ne pas être évaluée ! Mode d'évaluation dit paresseux.

Exemple de la conjonction

```
>>> (42 > 21) and (42 != 48)
```

Exemple de la conjonction

```
>>> (42 > 21) and (42 != 48)
True
```

Exemple de la conjonction

```
>>> (42 > 21) and (42 != 48)
True
>>> (42 > 21) and (42 == 48)
```

Exemple de la conjonction

```
>>> (42 > 21) and (42 != 48)
True
>>> (42 > 21) and (42 == 48)
False
```

Exemple de la conjonction

```
>>> (42 > 21) and (42 != 48)
True
>>> (42 > 21) and (42 == 48)
False
>>> (21 > 42) and (42 == (48/0))
```

Exemple de la conjonction

```
>>> (42 > 21) and (42 != 48)
True
>>> (42 > 21) and (42 == 48)
False
>>> (21 > 42) and (42 == (48/0))
False
```

Exemple de la conjonction

```
>>> (42 > 21) and (42 != 48)
True
>>> (42 > 21) and (42 == 48)
False
>>> (21 > 42) and (42 == (48/0))
False
>>> (42 > 21) and (42 == (48/0))
```

Exemple de la conjonction

```
>>> (42 > 21) and (42 != 48)
True
>>> (42 > 21) and (42 == 48)
False
>>> (21 > 42) and (42 == (48/0))
False
>>> (42 > 21) and (42 == (48/0))
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Principe d'évaluation de la disjonction

`expression1`	`expression2`	`expression1 or expression2`
`True`	`True`	`True`
`True`	`False`	`True`
`False`	`True`	`True`
`False`	`False`	`False`

Principe d'évaluation de la disjonction

Evaluation de `expression1`, et

- Soit la valeur est `True` , et dans ce cas retourner `True`
- Soit la valeur est `False` , dans ce cas
  - Evaluation de `expression2`, et
    - Soit la valeur est `True` , et dans ce cas retourner `True`
    - Soit la valeur est `False` , et dans ce cas retourner `False`

Attention ! `expression2` peut ne pas être évaluée ! Mode d'évaluation dit paresseux.

Exemple de la disjonction

```
>>> (42 > 21) or (42 != 48)
```

Exemple de la disjonction

```
>>> (42 > 21) or (42 != 48)
True
```

Exemple de la disjonction

```
>>> (42 > 21) or (42 != 48)
True
>>> (42 < 21) or (42 == 48)
```

Exemple de la disjonction

```
>>> (42 > 21) or (42 != 48)
True
>>> (42 < 21) or (42 == 48)
False
```

Exemple de la disjonction

```
>>> (42 > 21) or (42 != 48)
True
>>> (42 < 21) or (42 == 48)
False
>>> (42 > 21) or (42 == (48/0))
```

Exemple de la disjonction

```
>>> (42 > 21) or (42 != 48)
True
>>> (42 < 21) or (42 == 48)
False
>>> (42 > 21) or (42 == (48/0))
True
```

Exemple de la disjonction

```
>>> (42 > 21) or (42 != 48)
True
>>> (42 < 21) or (42 == 48)
False
>>> (42 > 21) or (42 == (48/0))
True
>>> (21 > 42) or (42 == (48/0))
```

Exemple de la disjonction

```
>>> (42 > 21) or (42 != 48)
True
>>> (42 < 21) or (42 == 48)
False
>>> (42 > 21) or (42 == (48/0))
True
>>> (21 > 42) or (42 == (48/0))
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Un exemple plus complexe

```
>>> (not("a" in "paon") or ("a" in "pantin"))
and
(len("paon") < len("pantin") or len("paon" + "pantin") > 10)
and
not(len("paon") == len("pantin"))
```

Un exemple plus complexe

```
>>> (not("a" in "paon") or ("a" in "pantin"))
and
(len("paon") < len("pantin") or len("paon" + "pantin") > 10)
and
not(len("paon") == len("pantin"))
True
```

Loi de Morgan

Loi de Morgan

Les lois de Morgan permettent de définir la négation des `and` et des `or`.

- `not`(a `and` b) == `not`(a) `or` `not`(b)
- `not`(a `or` b) == `not`(a) `and` `not`(b)

Loi de Morgan

Loi de Morgan

Les lois de Morgan permettent de définir la négation des `and` et des `or`.

- `not`(a `and` b) == `not`(a) `or` `not`(b)
- `not`(a `or` b) == `not`(a) `and` `not`(b)

```
>>> nb = 25
>>> not(nb >= 30 and nb < 35)
```

Loi de Morgan

Loi de Morgan

Les lois de Morgan permettent de définir la négation des `and` et des `or`.

- `not`(a `and` b) == `not`(a) `or` `not`(b)
- `not`(a `or` b) == `not`(a) `and` `not`(b)

```
>>> nb = 25
>>> not(nb >= 30 and nb < 35)
True
```

Loi de Morgan

Loi de Morgan

Les lois de Morgan permettent de définir la négation des `and` et des `or`.

- `not`(a `and` b) == `not`(a) `or` `not`(b)
- `not`(a `or` b) == `not`(a) `and` `not`(b)

```
>>> nb = 25
>>> not(nb >= 30 and nb < 35)
True
>>> nb < 30 or nb >=35
True
```

Loi de Morgan

Loi de Morgan

Les lois de Morgan permettent de définir la négation des `and` et des `or`.

- `not`(a `and` b) == `not`(a) `or` `not`(b)
- `not`(a `or` b) == `not`(a) `and` `not`(b)

```
>>> nb = 25
>>> not(nb >= 30 and nb < 35)
True
>>> nb < 30 or nb >=35
True
>>> nb = 32
>>> not(nb >= 30 and nb < 35)
```

Loi de Morgan

Loi de Morgan

Les lois de Morgan permettent de définir la négation des `and` et des `or`.

- `not`(a `and` b) == `not`(a) `or` `not`(b)
- `not`(a `or` b) == `not`(a) `and` `not`(b)

```
>>> nb = 25
>>> not(nb >= 30 and nb < 35)
True
>>> nb < 30 or nb >=35
True
>>> nb = 32
>>> not(nb >= 30 and nb < 35)
False
```

Loi de Morgan

Loi de Morgan

Les lois de Morgan permettent de définir la négation des `and` et des `or`.

- `not`(a `and` b) == `not`(a) `or` `not`(b)
- `not`(a `or` b) == `not`(a) `and` `not`(b)

```
>>> nb = 25
>>> not(nb >= 30 and nb < 35)
True
>>> nb < 30 or nb >=35
True
>>> nb = 32
>>> not(nb >= 30 and nb < 35)
False
>>> nb < 30 or nb >=35
False
```

Les instructions conditionnelles simples

Instruction conditionnelle simple

$$|x| = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{sinon} \end{cases}$$

- 3 éléments à distinguer :
  - l'expression qui permet de **choisir** : $x \geq 0$
  - l'instruction à appliquer si le choix est vrai : x
  - l'instruction à appliquer si le choix est faux : $-x$
- On nomme :
  - la condition : l'expression qui permet de **choisir**
  - le conséquent : l'instruction à appliquer si le choix est vrai
  - l'alternant : l'instruction à appliquer si le choix est faux

Syntaxe des instructions conditionnelles

```
if condition:
    consequent
else:
    alternant
```

- Avec
  - la condition : expression booléenne, de type `bool`
  - le conséquent : instruction ou suite d'instructions
  - l'alternant : instruction ou suite d'instructions
- Remarques :
  - Le bloc `else` peut ne pas être présent
  - Attention à ne pas oublier le `:` et l'indentation
- Attention ! la condition est une expression booléenne !
  - On n'écrit jamais : `condition == True`
  - On n'écrit jamais : `condition == False`

Exemples d'instruction conditionnelle simple

```
# Programme permettant d'afficher la valeur absolue
# d'un entier entré au clavier
```

Exemples d'instruction conditionnelle simple

```
# Programme permettant d'afficher la valeur absolue
# d'un entier entré au clavier

x = int(input("Entrez un entier : "))
```

Exemples d'instruction conditionnelle simple

```
# Programme permettant d'afficher la valeur absolue
# d'un entier entré au clavier

x = int(input("Entrez un entier : "))

if x >= 0:
```

Exemples d'instruction conditionnelle simple

```
# Programme permettant d'afficher la valeur absolue
# d'un entier entré au clavier

x = int(input("Entrez un entier : "))

if x >= 0:
    print(x)
```

Exemples d'instruction conditionnelle simple

```
# Programme permettant d'afficher la valeur absolue
# d'un entier entré au clavier

x = int(input("Entrez un entier : "))

if x >= 0:
    print(x)
else:
    print(-x)
```

Exemples d'instruction conditionnelle simple (2)

```
# Programme permettant de déterminer si un entier entré au
# clavier est pair ou impair
```

Exemples d'instruction conditionnelle simple (2)

```
# Programme permettant de déterminer si un entier entré au
# clavier est pair ou impair

x = int(input("Entrez un entier : "))
```

Exemples d'instruction conditionnelle simple (2)

```
# Programme permettant de déterminer si un entier entré au
# clavier est pair ou impair

x = int(input("Entrez un entier : "))

estPair = ((x % 2) == 0) #estPair est un booléen
```

Exemples d'instruction conditionnelle simple (2)

```
# Programme permettant de déterminer si un entier entré au
# clavier est pair ou impair

x = int(input("Entrez un entier : "))

estPair = ((x % 2) == 0) #estPair est un booléen

if estPair:
    print(x, "est un entier pair")
else:
    print(x, "est un entier impair")
```

Exemples d'instruction conditionnelle simple (3)

```
# Division d'un nombre y par un nombre x

x, y = 0, 5
```

Exemples d'instruction conditionnelle simple (3)

```
# Division d'un nombre y par un nombre x

x, y = 0, 5

if not(x == 0) :
    nb = y/x
    print(nb)
```

Exemples d'instruction conditionnelle simple (3)

```
# Division d'un nombre y par un nombre x

x, y = 0, 5

if not(x == 0) :
    nb = y/x
    print(nb)
else :
    print("Ce nombre ne peut pas être calculé, division par 0")
```

Exemples d'instruction conditionnelle simple (3)

```
# Division d'un nombre y par un nombre x

x, y = 0, 5

if not(x == 0) :
    nb = y/x
    print(nb)
else :
    print("Ce nombre ne peut pas être calculé, division par 0")
```

Autre solution, en utilisant le transtypage :

Exemples d'instruction conditionnelle simple (3)

```
# Division d'un nombre y par un nombre x

x, y = 0, 5

if not(x == 0) :
    nb = y/x
    print(nb)
else :
    print("Ce nombre ne peut pas être calculé, division par 0")
```

Autre solution, en utilisant le transtypage :

```
# Division d'un nombre y par un nombre x

x, y = 0, 5
nb = ("Ce nombre ne peut pas être calculé, division par 0")
if not(x == 0):
    nb = y/x
print(nb)
```

Exemples d'instruction conditionnelle simple (4)

```
# Minimum et maximum
if x <= y:
    minimum = x
    maximum = y
else:
    minimum = y
    maximum = x
```

Les instructions conditionnelles multiples

Choix multiples

Problème

Imaginons que l'on veuille écrire un programme qui, à partir d'un pH donné, indique si c'est un acide, un basique ou un neutre.

- Il y a donc **3** solutions possibles
- On pourrait imbriquer des `if`

Exemple du pH, imbrications d'alternatives

```
ph = float(input("pH ? "))

if ph <= 7:
    if ph == 7:
        print("C'est un neutre")
    else:
        print("C'est un acide")
else:
    print("C'est une base")
```

Exemple du pH, imbrications d'alternatives

```
ph = float(input("pH ? "))

if ph <= 7:
    if ph == 7:
        print("C'est un neutre")
    else:
        print("C'est un acide")
else:
    print("C'est une base")
```

Plus simple : les instructions conditionnelles multiples permettent de choisir entre plus de deux valeurs

Les instructions conditionnelles multiples

```
if condition1:
    conséquent1
elif condition2:
    conséquent2
elif ...

else:
    alternant
```

Evaluer `condition1`.

1. Si `condition1` est `True`, exécuter uniquement `conséquent1`
2. Si `condition1` est `False`, évaluer `condition2`
   - 2.1 Si `condition2` est `True`, exécuter uniquement `conséquent2`
   - 2.2 Si `condition2` est `False`, évaluer `condition3`
     - 2.2.1 ...
3. Si aucune des conditions n'est vraie, on évalue `alternant`

Exemple du pH, instruction conditionnelle multiple

```
ph = float(input("pH ? "))

if ph == 7:
    print("C'est un neutre")
elif ph < 7:
    print("C'est une base")
else:
   print("C'est un acide")
```

Exemple : mention

```
#Affichage de la mention en fonction de la note obtenue

moyenne = int(input("Quelle moyenne avez-vous obtenue ? "))

if moyenne < 10 :
    print("C'est un échec")
elif moyenne < 12 :
    print("Mention passable")
elif moyenne < 14 :
    print("Mention Assez-bien")
elif moyenne < 16 :
    print("Mention Bien")
elif moyenne < 18 :
    print("Mention Très Bien")
else :
    print("Félicitations!")
```

Quelques erreurs courantes

Mauvaise utilisation d'une suite de `if`

Qu'affiche le programme suivant ?

```
age = 9
if age < 10:
    print('Moins de 10 ans')
if age < 15:
    print('Moins de 15 ans')
if age < 18:
    print('Moins de 18 ans')
```

Mauvaise utilisation d'une suite de `if`

Qu'affiche le programme suivant ?

```
age = 9
if age < 10:
    print('Moins de 10 ans')
if age < 15:
    print('Moins de 15 ans')
if age < 18:
    print('Moins de 18 ans')
```

```
Moins de 10 ans
Moins de 15 ans
Moins de 18 ans
```

Mauvaise utilisation d'une suite de `if`

On aurait voulu :

```
age = 9
if age < 10:
    print('Moins de 10 ans')
elif age < 15:
    print('Moins de 15 ans')
elif age < 18:
    print('Moins de 18 ans')
```

Mauvaise utilisation de else avec une suite de `if`

Qu'affiche le programme suivant ?

```
note = 9
if note < 10:
    print("Echec")
if note >= 10 and note < 15:
    print("Bien")
if note >= 15 and note < 18:
    print("Très bien")
else:
    print("Excellent!")
```

Mauvaise utilisation de else avec une suite de `if`

Qu'affiche le programme suivant ?

```
note = 9
if note < 10:
    print("Echec")
if note >= 10 and note < 15:
    print("Bien")
if note >= 15 and note < 18:
    print("Très bien")
else:
    print("Excellent!")
```

```
Echec
Excellent!
```

Pour conclure

Résumé du cours

Aujourd'hui, on a vu

- Ce qu'est une séquence d'instructions
- L'importance de l'indentation en Python
- L'évaluation des expressions booléennes
- Les instructions conditionnelles simples et multiples