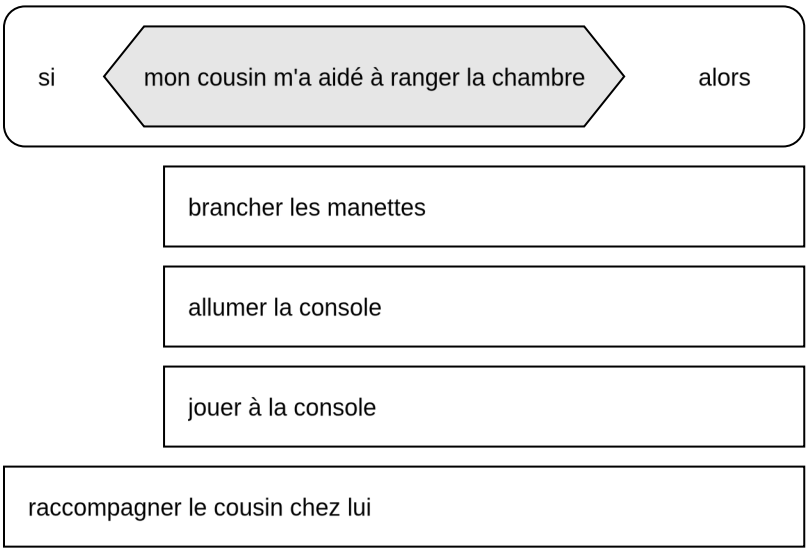
Nom :	Prénom :	Classe :

NSI 1re — Conditions #1 — if, indentation, bloc de code, opérateurs de comparaison

Élément du programme : Langages et programmation, constructions élémentaires → conditionnelles.

À quoi ça sert, une condition ?

Les **conditions** sont des structures de contrôle essentielles en programmation, permettant d'exécuter différentes parties de code selon qu'une condition est **vraie** ou **fausse**.



Dans cet exemple, nous avons défini une **condition**,
condition dont **l'affirmation** est `mon cousin m'a aidé à ranger la chambre`.

- Si cette affirmation est **vraie** (c'est à dire *mon cousin m'a effectivement aidé à ranger la chambre*), alors on exécute le **code contenu dans la condition**, à savoir :
  - brancher les manettes
  - allumer la console
  - jouer à la console
- Mais si cette affirmation est **fausse** (c'est à dire mon cousin ne m'a pas aidé à ranger la chambre), alors on **n'exécute pas** le code contenu dans la condition.
  Donc la manette ne sera pas branchée, la console ne sera pas allumée, et nous n'y jouerons pas.

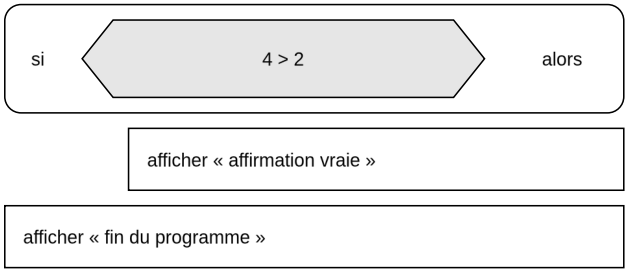
Gardons cependant en tête que le code placé *en dehors* de la condition (c'est à dire le code *raccompagner le cousin chez lui*) sera exécuté, que l'affirmation soit vraie ou fausse.

Et oui, car il ne fait pas parti de la condition, lui !

Condition if, en Python

Rien de mieux qu'une *lecture* d'un code, pour comprendre comment *écrire* ce code !

Schéma	Code Python

	`if 4 > 2:`
	`    print('affirmation vraie')`
	`print('fin du programme')`

Le schéma de gauche correspond au code Python de droite, et vis-versa.

Dans le code ci-dessus, nous avons défini une condition dont l'affirmation est `4 > 2`.

- Si cette affirmation est vraie, alors on exécute le code contenu dans la condition.
  Le message "affirmation vraie" est alors affiché.
- Si cette affirmation est fausse, alors on n'exécute pas le code contenu dans la condition.
  Le message "affirmation vraie" n'est alors pas affiché.

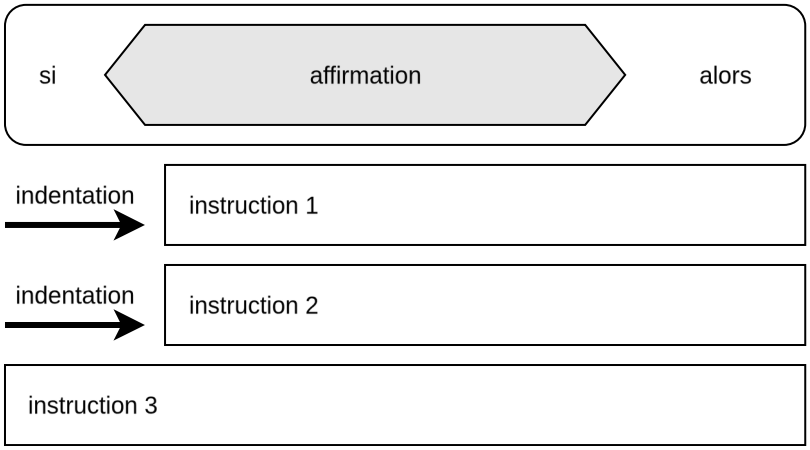
Puis, que cette affirmation soit vraie ou fausse, on affiche le message "fin du programme".

Avez-vous vu les **quatre espaces** devant l'instruction `print("affirmation vraie")` ?

C'est ce qu'on appelle **l'indentation** !

Indentation

L'indentation en Python correspond aux **quatre espaces** placés **au début** des lignes de code.
L'indentation sert à *regrouper des lignes* qui doivent être exécutées ensemble.



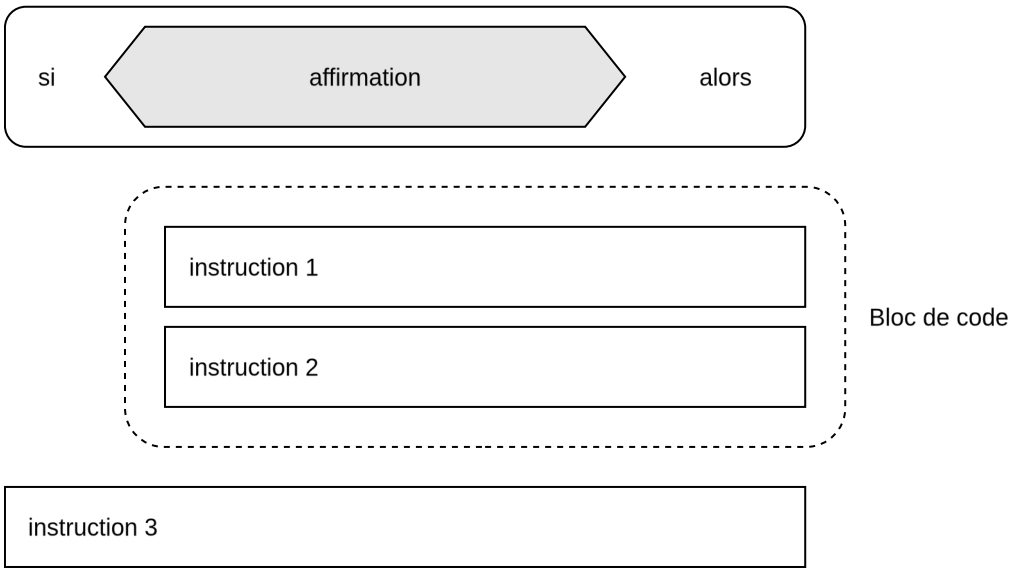
Dans ce schéma, les instructions 1 et 2 possèdent une **indentation**. Elles seront toutes deux exécutées si (et uniquement si) l'affirmation est vraie.

On dit que les instructions 1 et 2 sont indentées.

L'instruction 3 n'est pas indentée. Donc elle ne fait pas partie de la condition, et sera exécutée quoi qu'il arrive.

Code indenté = Bloc de code

En programmation, le code qui est indenté forme un groupe — un bloc — que l'on appelle un « **bloc de code** » :



Dans cet exemple, les instructions 1 et 2 sont indentées, elles forment un **bloc de code**.

Ce *bloc de code* sera exécuté si l'affirmation est vraie.

➤ Exercice 1 (à vous de jouer)

Analysez les programmes Python suivants, puis déterminez ce qui est affiché ou non.

Code Python	Message(s) affiché(s)	Message(s) non affiché(s)

`if 4 > 2:`
`  print('affirmation vraie')`
`print('fin du programme')`		

`if 123 < 18:`
`  print('affirmation vraie')`
`print('fin du programme')`		

`if 18 > 123:`
`  print('affirmation vraie')`
`print('fin du programme')`		

`if 10/2 > 4:`
`  print('affirmation vraie')`
`  print('merci et à toute')`
`print('fin du programme')`		

Les opérateurs de comparaison

Les **opérateurs de comparaison** sont des symboles utilisés pour comparer deux valeurs dans une **affirmation**.

Exemples :

Code Python	Affirmation	Opérateur de comparaison utilisé
`if 18 > 10:` `    print('affirmation vraie')`	`18 > 10`	`>`
`if 18 < 10:` `    print('affirmation vraie')`	`18 < 10`	`<`

Les opérateurs permettent d'établir des *relations* entre les valeurs comme *égalité*, *différence*, *supériorité* ou *infériorité*.

Voici une liste des opérateurs de comparaison disponibles dans Python :

Opérateur	Description	Exemple
`>`	Strictement supérieur à	`if 100 > 20:` `    print('100 est strictement supérieur à 20')`
`<`	Strictement inférieur à	`if 100 < 20:` `    print('100 est strictement inférieur à 20')`
`>=`	Supérieur ou égal à	`if 100 >= 20:` `    print('100 est supérieur ou égal à 20')`
`<=`	Inférieur ou égal à	`if 100 <= 20:` `    print('100 est inférieur ou égal à 20')`
`==`	Égal à	`if 100 == 20:` `    print('100 est égal à 20')`
`!=`	Non égal à	`if 100 != 20:` `    print('100 est non égal à 20')`

➤ Exercice 2

Dans les exemples du tableau donné ci-dessus, quelles sont les affirmations[1] qui sont vraies ?

1 Les « affirmations » sont aussi appelées « expressions booléenne », car elles retournent True ou False.

Peut-on comparer les valeurs stockées dans les variables ?

C'est vrai que nous n'avons pas parlé des variables !

Il est bien sûr possible de comparer les valeurs qu'elles contiennent ; c'est même l'usage premier des conditions.

```
1  age = 18
2  if age >= 18:
3      print('bonjour')
4      print('vous êtes majeur(e)')
5  print('fin du programme')
```

Explication du code ci-dessus :

Ligne	Explication
1	Assignation de la valeur `18` à une variable age.
2	Condition dans laquelle on affirme que la valeur stockée dans age est supérieure ou égale à `18`.
3 et 4	Bloc de code qui sera exécuté si l'affirmation est vraie (ce bloc de code affiche deux messages).
5	Cette ligne n'est pas dans le bloc de code. Elle sera exécutée quoi qu'il arrive.

➤ Exercice 3

Analysez les programmes Python suivants, puis déterminez ce qui est affiché ou non.

Code Python	Message(s) affiché(s)	Message(s) non affiché(s)

`age = 15`
`if age >= 14:`
`  print('Voiture sans permis')`		

`age = 15`
`if age >= 18:`
`  print('Majeur(e)')`		

`a = 100`
`if a == 100:`
`  print('Ça vaut 100')`
`if a != 100:`
`  print('Ça ne vaut pas 100')`		

`a = 99`
`if a == 100:`
`  print('Ça vaut 100')`
`if a != 100:`
`  print('Ça ne vaut pas 100')`		

La suite page suivante ...

Code Python	Message(s) affiché(s)	Message(s) non affiché(s)

`i = 20`
`if i >= 20:`
`  print('B')`
`if i <= 20:`
`  print('R')`
`if i > 20:`
`  print('T')`
`if i != 22:`
`  print('A')`
`  print('V')`
`print('O')`		

`i = 19`
`if i >= 18:`
`  print('A')`
`if i > 19:`
`  print('B')`
`if i <= 19:`
`  print('C')`
`if i < 20:`
`  print('D')`
`if i == '19':`
`  print('E')`
`print('Fin')`		

➤ Exercice 4

Observez le programme ci dessous, puis répondez aux différentes questions posées.

```
1   age = input('Votre âge svp ? ')
2   age = int(age)
3   if age > 13:
4       print('✅ scooter 50 cm3')
5   if age >= 14:
6       print('✅ voiture sans permis')
7   if age >= 15:
8       print('✅ conduite accompagnée')
9   if age >= 17:
10      print('✅ permis de conduire')
```

Quel est le rôle de `int(age)` ?

Quel est le rôle de la ligne 1 ?

Vous avez répondu aux deux questions ci-dessus ? Alors en voici d'autres :

Quel(s) message(s) serai(en)t affiché(s) si l'utilisatrice ou l'utilisateur saisit un âge valant 15 ?

Quel(s) message(s) serai(en)t affiché(s) si l'utilisatrice ou l'utilisateur saisit un âge valant 13 ?