Chapitre 4

# Les conditions

Jusqu'à présent, nos programmes sont **linéaires**. Les instructions y sont exécutées les unes après les autres, "en ligne droite".
Il est parfois nécessaire que certaines instructions ne soient exécutées que si une certaine condition est remplie.
On utilise alors des **instructions conditionnelles**.

## 4.1 L'instruction `if`

La syntaxe minimale pour utiliser une condition est la suivante :
`condition` est un **booléen** (aussi appelé **prédicat** en algorithmique).

```
if condition:
  # Instruction 1
  # Instruction 2
  # ...
```

Si `condition` est égal à `True`, alors le bloc d'instructions suivant le `if` est exécuté.
Considérons par exemple le programme suivant :

```
1 age = 16
2
3 if age >= 18:
4   print("Vous êtes majeur")
```

▷ ***J'ai exécuté le programme précédent mais il n'affiche rien...***

Et c'est bien normal ! L'instruction `print("Vous êtes majeur")` n'est exécutée par l'interpréteur Python que lorsque la variable `age` a une valeur supérieure ou égale à 18, ce qui n'est pas le cas ici.
On peut vérifier dans la console (après avoir exécuté le programme ci-dessus) :

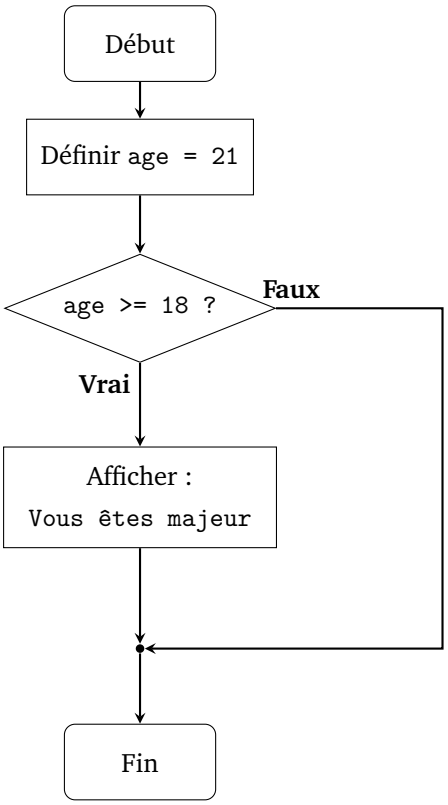
```
>>> age >= 18
False
```

En modifiant la valeur de `age` à 21, le programme affiche alors la phrase tant attendue :

```
Vous êtes majeur
```

**Quelques remarques**

- Ne pas oublier les deux points `:` après la condition.
- On peut représenter graphiquement notre programme précédent de la manière suivante.
  Cela permet de mieux percevoir son déroulement.




- Il est possible d'exécuter plusieurs actions après le `if`, à condition de les **indenter correctement**.

```
1  a = 3
2
3  if a > 0:
4    print("Positif")
5    print("Super cool !")
```

affichera :

```
Positif
Super cool !
```

```
1  a = -5
2
3  if a > 0:
4    print("Positif")
5    print("Super cool !")
```

n'affichera rien.

```
1  a = -5
2
3  if a > 0:
4    print("Positif")
5  print("Super cool !")
```

affichera :

```
Super cool !
```

Dans ce dernier cas, le `print` ne fait pas partie du bloc d'instructions à exécuter si la condition `a > 0` est remplie.

Il est exécuté dans tous les cas.

## 4.2 L'instruction `else`

Il est également possible d'exécuter des instructions dans le cas où la condition donnée n'est pas vérifiée.
La syntaxe à utiliser est la suivante :

```
if condition:
  # Instructions si la condition est vérifiée
else:
  # Instructions si la condition n'est pas vérifiée
```

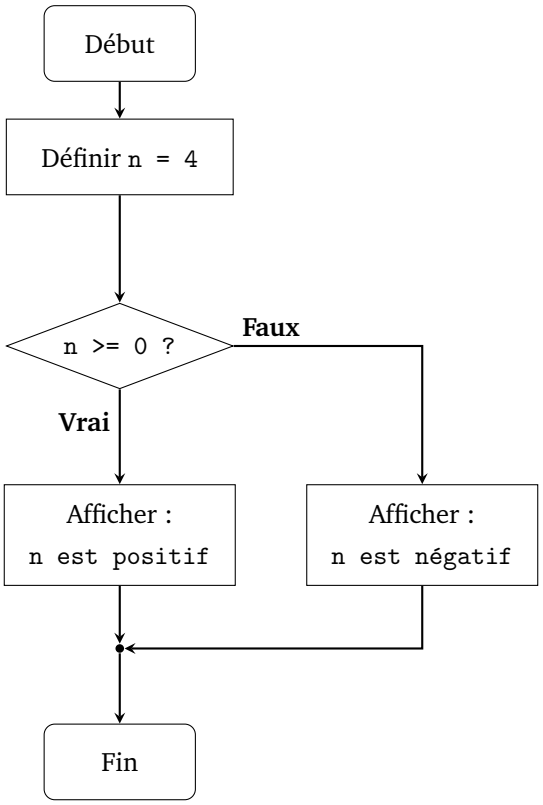
On peut par exemple afficher un message selon qu'un nombre soit positif ou négatif :

```
1 n = 4
2
3 if n >= 0:
4   print("n est positif")
5 else:
6   print("n est négatif")
```

Résultat :

```
n est positif
```

Schématiquement, cela donne :




Il n'est pas nécessaire de préciser une condition dans le `else`. La phrase « n est négatif » est affichée si la condition `n >= 0` est fausse, c'est à dire si la condition `n < 0` est vraie.

## 4.3 L'instruction `elif`

Parfois, il est nécessaire de tester plusieurs conditions s'excluant les unes des autres (ne pouvant être vraies en même temps).
On peut alors utiliser le mot clé `elif` , contraction de *else if*, en français *sinon si*.
Par exemple, l'état de l'eau dépend de sa température :

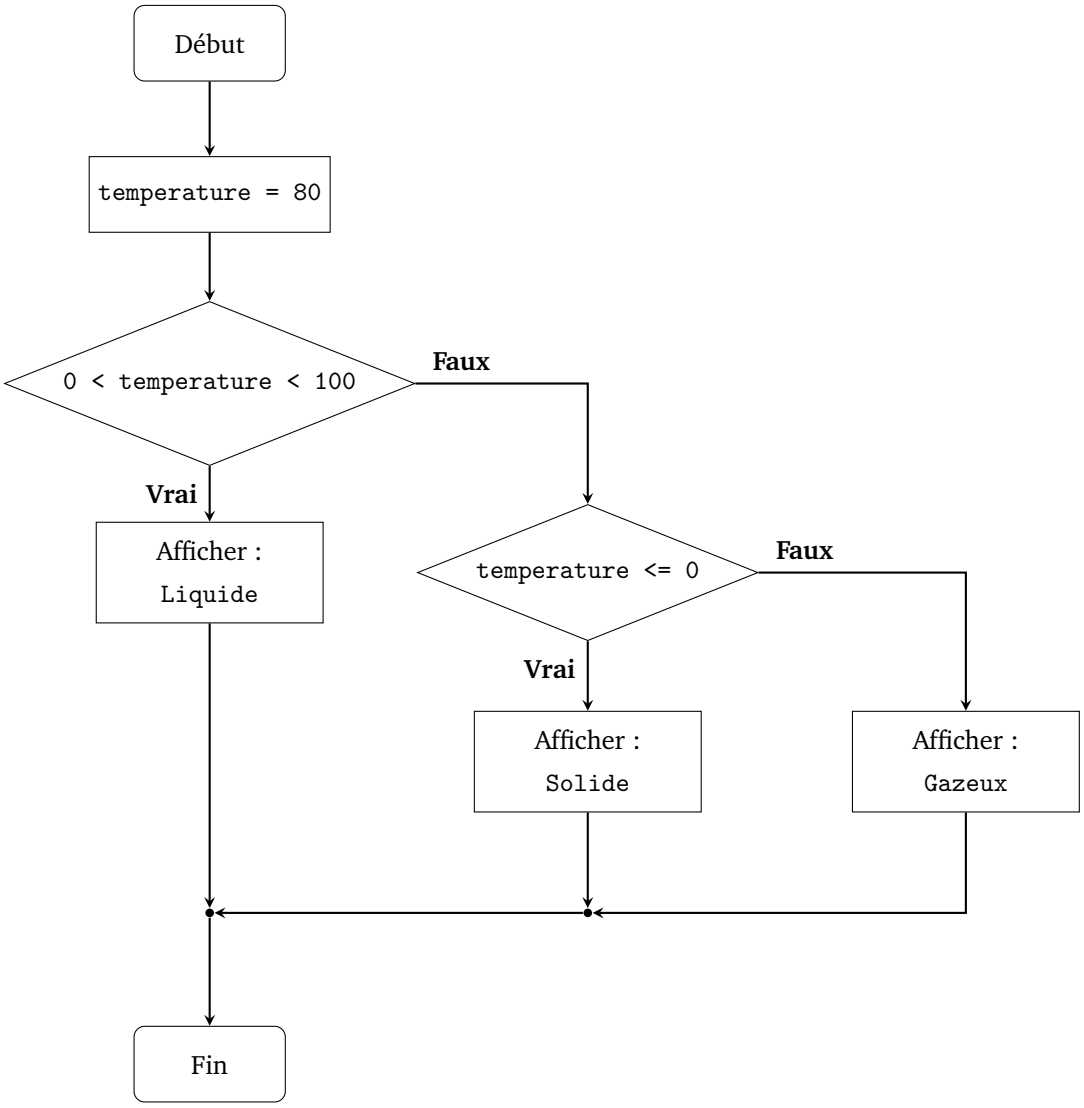
- **Solide** si la température est inférieure à 0°C
- **Liquide** si la température est comprise entre 0°C et 100°C
- **Gazeux** si la température est supérieure à 100°C

On peut écrire un programme affichant l'état de l'eau en fonction de sa température :

```
1  temperature = 80
2
3  if 0 < temperature < 100:
4    print("Liquide")
5  elif temperature <= 0:
6    print("Solide")
7  else:
8    print("Gazeux")
```

Résultat :

```
Liquide
```



## Bonus 1 : utilisation d'une fonction

Plutôt que de modifier directement la valeur de `temperature` dans le programme précédent, on peut définir une fonction prenant en paramètre `temperature` et faisant exactement la même chose.

```
1 def etat_eau(temperature):
2   if 0 < temperature < 100:
3     print("Liquide")
4   elif temperature <= 0:
5     print("Solide")
6   else:
7     print("Gazeux")
```

☞ ***Attention aux identations ! !***

En exécutant ce programme, il ne se passe rien, mais on peut à présent utiliser la fonction `etat_eau` depuis la console :

```
>>> etat_eau(80)
Liquide
>>> etat_eau(120)
Gazeux
>>> etat_eau(-70.5)
Solide
```

## Bonus 2 : une fonction qui retourne une valeur conditionnelle

Une fonction n'affiche pas nécessairement quelque chose, elle peut aussi (et c'est souvent le cas) retourner une valeur.

```
1 def maximum(a,b):
2     if a > b:
3         return a
4     else:
5         return b
```

```
>>> maximum(81,37)
81
>>> maximum(62, 101)
101
```

▷ ***Deux `return` dans une fonction ? La fonction renvoie plusieurs valeurs ?***

Il y a effectivement deux `return`, mais un seul d'entre eux est « accessible » lors de l'appel de la fonction. On ne peut pas être à la fois dans le cas où `n` est positif et où `n` est négatif. Pas d'inquiétude donc !

## Bonus 3 : Opérateurs logiques

Il est possible d'utiliser les opérateurs logiques `and`, `or` et `not` dans les conditions.
Par exemple, pour savoir si un nombre `n` est divisible par 2 ou par 3, on peut écrire :

```
1 if n % 2 == 0 or n % 3 == 0:
2   print(n, "est divisible par 2 ou par 3")
3 else:
4   print(n, "n'est divisible ni par 2, ni par 3")
```

## 4.4 Un mot sur *l'opérateur ternaire* `... if ... else ...`

Dans un soucis de simplifier au maximum le code, Python permet de définir une valeur « à la volée » selon qu'une certaine condition soit vraie ou non :

```
valeur_si_condition_vraie if condition else valeur_si_condition_fausse
```

Par exemple :

```
>>> age = 21
>>> "Majeur" if age >= 21 else "Mineur"
'Positif'
```

On peut récupérer cette valeur dans une variable :

```
>>> statut = "Majeur" if age >= 21 else "Mineur"
>>> statut
'Majeur'
```

Cela permet entre autres de simplifier l'écriture de certaines fonctions, comme la fonction `maximum(a,b)` vue précédemment :

Version « basique »

```
1 def maximum(a,b):
2   if a > b:
3     return a
4   else:
5     return b
```

Version « raccourcie »

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

## 4.5 QCM

Dans le QCM suivant, chaque question peut admettre **une ou plusieurs bonnes réponses**.

| Questions | Réponses |
|---|---|
| **1.** Quels mots clés permettent d'utiliser les conditions en Python ? | ☐ `if`<br>☐ `si`<br>☐ `else`<br>☐ `else if` |

| Questions | Réponses |
|---|---|
| **2.** Quelles valeurs de `a` et `b` permettent d'afficher `OK` avec le programme ci-dessous ? | ☐ `a = 1` et `b = 1`<br>☐ `a = 1` et `b = 0`<br>☐ `a = 2` et `b = -1`<br>☐ `a = 2` et `b = -2` |

```
1  if a + b > 0 and a * b < 1:
2    print("OK")
3  else:
4    print("Pas OK")
```

| Questions | Réponses |
|---|---|
| **3.** Que va afficher le programme ci-dessous si `a = 4` ? | ☐ `Toto`<br>☐ `Tata`<br>☐ `Titi`<br>☐ Une erreur |

```
1  if 3*a > 2:
2    b = a**2
3    if b <= 10:
4      print("Toto")
5    else:
6      print("Tata")
7  else:
8    print("Titi")
```

| Questions | Réponses |
|---|---|
| **4.** Que va retourner l'instruction `mystere(4)` , où `mystere` est la fonction définie ci-dessous ? | ☐ 4<br>☐ 5<br>☐ 6<br>☐ Une erreur |

```
1  def mystere(x):
2    if x < 10:
3      return x
4    elif x < 5:
5      return x + 1
6    else:
7      return x + 2
```

## 4.6 Exercices

☞ *Ces exercices sont auto-corrigés en suivant ce lien.*

**Exercice 01** Écrire une fonction `divisiblePar7(n)` qui affiche si le nombre entier `n` est divisible ou non par 7.

```
>>> divisiblePar7(49)
49 est divisible par 7
>>> divisiblePar7(100)
100 n'est pas divisible par 7
```

**Exercice 02** Définir la fonction mathématique suivante :

$$f(n) = \begin{cases} \frac{n}{2} & si \quad n \text{ est pair} \\ 3n+1 & si \quad n \text{ est impair} \end{cases}$$

```
>>> f(32)
16
>>> f(9)
28
```

**Exercice 03** Écrire une fonction `max2(a,b)` qui retourne le maximum des deux nombres `a` et `b`.

**Exercice 04** Écrire une fonction `max3(a,b,c)` qui retourne le maximum des trois nombres `a`, `b` et `c`.

**Exercice 05** ***Ça va être long...***

Écrire une fonction `mois(n)` qui retourne le nom du `n`-ième mois en français, `n` étant un entier entre 1 et 12. Si `n` n'est pas un entier entre 1 et 12, alors la fonction doit retourner ¡¡ Erreur ¿¿.

```
>>> mois(1)
Janvier
>>> mois(4)
Avril
>>> mois(13)
Erreur
```

**Exercice 06** ***Résultats du BAC***

Écrire une fonction `resultatBAC(note)` qui affiche le résultat obtenu au BAC avec une note égale à `note`, sachant que :

- Un étudiant est déclaré :
  - **Admis** s'il a une note supérieure ou égale à 10
  - **Admis au 2nd groupe** (rattrapage) s'il obtient une note entre 8 (inclus) et 10 (exclu)
  - **Recalé** sinon
- Il existe trois mentions différentes :
  - **Très bien** si la note est supérieure ou égale à 16
  - **Bien** si la note est comprise entre 14 (inclus) et 16 (exclu)
  - **Assez Bien** si la note est comprise entre 12 (inclus) et 14 (exclu)

```
>>> resultatBAC(15.4)
Admis avec la mention Bien
>>> resultatBAC(10.9)
Admis
>>> resultatBAC(7)
À l'année prochaine !
```

**Exercice 07** ***Bien choisir son mot de passe***

En tant qu'expert informatique, vous êtes amené à vérifier la sécurité des mots de passe d'une société.
Écrire une fonction `verification(motDePasse)` qui vérifie que le mot de passe `motDePasse` donné est correct.
Pour être correct, un mot de passe doit :

1. Contenir le caractère @
2. Contenir entre 8 et 20 caractères

```
>>> verification("abcdefghij")
Le mot de passe est incorrect.
>>> verification("superMotDePasse@")
Le mot de passe est correct.
```

**Exercice 08** ***Années bissextiles (*)***

Écrire une fonction `estBissextile(annee)` qui affiche si `annee` est bissextile ou non, sachant qu'une année est bissextile si elle est divisible par 4 et non divisible par 100 ou si elle est divisible par 400.

**Exercice 09** ***Nature d'un triangle (*)***

1. Écrire une fonction `estRectangle(a,b,c)` qui retourne `True` si le triangle de côtés `a`, `b` et `c` est rectangle, et `False` sinon.
2. Écrire une fonction `estIsocele(a,b,c)` qui retourne `True` si le triangle de côtés `a`, `b` et `c` est isocèle, et `False` sinon.
3. Écrire une fonction `natureTriangle(a,b,c)` qui affiche la nature d'un triangle de côtés `a`, `b` et `c`.
   Un triangle peut être :
   - Quelconque
   - Isocèle
   - Équilatéral
   - Rectangle
   - Rectangle isocèle

   On pourra utiliser les fonctions `estRectangle` et `estIsocele` définies précédemment.

**Exercice 10** ***Choisir la bonne direction***

Soient $A(x_A; y_A)$ et $B(x_B; y_B)$ deux points du plan.

Écrire une fonction `direction(xa,ya,xb,yb)` qui retourne la direction à suivre pour aller du point $A$ au point $B$.

La direction doit être une chaîne de caractères correspondante à un des 8 repères cardinaux suivants : $N$, $NE$, $E$, $SE$, $S$, $SO$, $O$ et $NO$.