Langage et Programmation	Chap 02	**Programmer sous condition**

2.1 Les booléens

Dans la vie courante, on prend des décisions en fonction d'une situation : **« s'il pleut, je prends un parapluie »**.

En programmation, c'est pareil : on exécute certaines instructions **seulement si** une condition est vraie. Avant de pouvoir tester des conditions, il faut pouvoir exprimer la notion de **vrai** ou de **faux**.

Les types de base

Dans le chapitre 1, nous avons vu les types de base suivant :

- `int` : nombres entiers (`-3`, `0`, `42`)
- `float` : nombres à virgule (`3.14`, `-0.5`)
- `str` : chaînes de caractères (`"NSI"`, `'Bonjour'`)
- `bool` : valeurs logiques `True` ou `False`

Ce chapitre permet d'expliquer comment fonctionne le type **bool**.

Définition 1 — Booléen

Une valeur **booléenne**, c'est-à-dire de type **bool**, ne peut prendre que deux valeurs : `True` (**vrai**) ou `False` (**faux**).

Exemple 1

Quelques expressions qui produisent un booléen :

```
>>> print(5 > 2)
True
>>> print(3 == 4)
False
>>> test = "NSI" != "SNT"
>>>print(test)
True
```

Exercice 1 — Vrai ou faux ?

Dire si chaque expression vaut `True` ou `False`.

```
3 < 3
7 >= 2*4
"python" == "Python"
10 % 2 == 0
7 == "7"
```

Le tableau suivant récapitule les différents opérateurs qui permettent d'obtenir des valeurs booléennes.

Comparaison	Sens
==	
!=	
<	
<=	
>	
>=	

Attention

Le symbole = désigne l'affectation, alors que le symbole == désigne la comparaison.

2.2 Tester des conditions

En programmation, il ne suffit pas toujours d'exécuter les instructions les unes après les autres.

Souvent, on veut que l'ordinateur prenne une décision en fonction d'une condition : « **si la température est trop basse, afficher un message d'alerte** », « **si un nombre est pair, écrire "pair"** », etc.

Pour cela, on utilise la structure conditionnelle `if`, qui permet de tester une situation et d'exécuter un bloc d'instructions uniquement si la condition est vraie.

Définition 2 — Instruction `if`

La structure de base de l'instruction `if` est :

```
if condition:
    bloc1
    ...
```

— On termine la ligne par le symbole :

— On **indente** (décale à droite).

Le programme ci-dessus se lit comme : "Si la `condition` est vraie, alors on exécute le bloc d'instruction `bloc1`.

Exemple 2

```
temperature = 3
if temperature <= 5:
    print("Couvrez-vous !")
```

Exercice 2 — Premier test

Écrire un programme qui crée une variable x et y qui vaut 12 puis qui affiche `"pair"` si `x` est divisible par 2

Écrire un programme qui crée une variable y qui vaut 27 puis qui affiche `"impair"` si `y` n'est pas divisible par 2

Avec un simple `if`, on peut choisir d'exécuter un bloc d'instructions seulement si une condition est vraie. Mais parfois, on veut aussi préciser ce qui doit se passer quand la condition n'est pas

vérifiée. Par exemple : « **si la température est basse, je mets un manteau ; sinon, je n'en mets pas** ». C'est le rôle du mot-clé `else`.

2.3 Rajouter un cas

Définition 3

La branche `else` s'exécute quand la condition est fausse :

```
if condition: # si condition est vraie
    bloc1 #alors on exécute bloc1
else: #sinon
    bloc2 #alors on exécute bloc2

```

Exemple 3

```
age = 18
if age >= 18:
    print("Majeur")
else:
    print("Mineur")
```

Exercice 3 — Positif / négatif / nul

Créer une variable x qui vaut un entier de votre choix.

Afficher `"positif"` ou `"négatif"` selon le cas.

2.4 Rajouter plusieurs cas

La structure `if ... else` permet déjà de distinguer deux cas possibles. Cependant, dans de nombreux problèmes, il existe plus de deux situations à traiter. Par exemple : attribuer une

mention selon une note (`"Très bien"`, `"Bien"`, `"Assez bien"`, `"Passable"`). Dans ce cas, on peut enchaîner plusieurs conditions grâce au mot-clé `elif`, qui signifie « sinon si ».

Définition 4

`elif` signifie « sinon si ». On peut en chaîner plusieurs.

```
note = 15
if note >= 16:
    mention = "Très bien"
elif note >= 14:
    mention = "Bien"
elif note >= 12:
    mention = "Assez bien"
else:
    mention = "Passable"
print(mention)
```

Exercice 4 — Grille tarifaire

Réaliser un programme qui lit un âge et affiche le prix d'un billet :

- moins de 12 ans : 6€;
- de 12 à 17 ans : 8€;
- 18 ans et plus : 10€.

2.5 Opérateurs logiques

Jusqu'ici, nous avons vu que les comparaisons renvoient une valeur booléenne (`True` ou `False`).

Dans un programme, il est souvent nécessaire de combiner plusieurs conditions : « **je peux prendre le train si j'ai un billet et si le train n'est pas annulé** », ou encore « **je prends un parapluie s'il pleut ou s'il neige** ».

Opérateurs logiques

Pour exprimer des combinaisons logiques, Python met à disposition trois opérateurs : `and`, `or` et `not`.

Opérateur	Effet
`and`	vrai si **les deux** conditions sont vraies
`or`	vrai si **au moins une** est vraie
`not`	inverse `True` ↔ `False`

Exemple 4

```
>>> meteo = "pluie"
>>> parapluie = (meteo == "pluie") or (meteo == "neige")
>>> print(parapluie)
True
```

```
>>> a_billet = True
>>> train_annule = False
>>> peut_voyager = a_billet and (not train_annule)
>>> print(peut_voyager)
True
```

Exercice 5 — Intervalle

Écrire une expression booléenne `test1` qui vaut `True` (vraie) si `x` est dans l'intervalle $[12; 27[$ (5 inclus, 12 exclu) et `False` sinon.

`x` est dans l'intervalle $[12; 27[$ si `x` est plus grand que 12 et strictement plus petit que 27

Etant donné trois variables entières `x`, yet `z`, écrire une expression booléenne `test2` qui vaut `True` (vrai) si `x` est le plus petit des trois entiers.
