# Tests et boucles conditionnelles

Spé NSI - Lycée du parc

Année 2020-2021

**Introduction**

Pour pouvoir s'adapter aux différentes situations qu'il doit traiter, un programme doit pouvoir exécuter une instruction (ou un bloc) plutôt qu'une autre. Ce sont les *tests* qui permettent ce comportement.

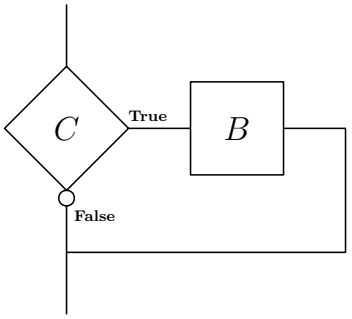
# I La structure conditionnelle

## I.1 if seul

Pour exécuter une instruction seulement si une condition est vérifiée on dispose de l'instruction `if` en Python.

```
if Condition:
    instruction Bloc
    instruction Bloc
    instruction Bloc
instruction
....
```

1




Exemple : pour remplacer un nombre $x$ par son opposé seulement si il est négatif [1], on peut utiliser le code suivant.

```
if x < 0:
    x = -x
```

2

**Exercice 1**

Pour transporter des poteaux sur une ile on dispose d'une barge dont la masse utile est de 1800 kg. Écrire un programme qui demande à l'utilisateur la masse d'un poteau puis le nombre de poteaux et qui affiche le message « Risque de surcharge ! » si la masse utile est dépassée.

1. En mathématique cela revient à remplacer un nombre par sa *valeur absolue*.

**Exercice 2**

On utilise aussi très souvent le `if` seul pour les situations de comptage.

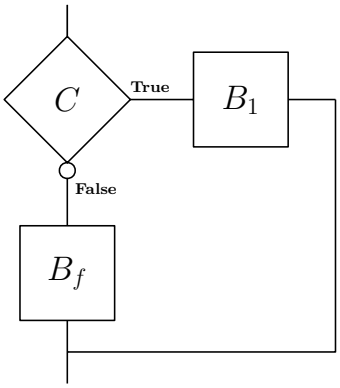
Écire un programme qui demande à l'utilisateur de saisir un texte et qui compte le nombre de « e » dans le texte.

## I.2 if avec else

On peut compléter `if` avec `else` pour pouvoir aussi exécuter une autre instruction (ou un bloc) dans le cas où la condition est fausse :

```
if Condition:
    instruction Bloc1
    instruction Bloc1
    instruction Bloc1
else:
    instruction Bloc final
    instruction Bloc final
    instruction Bloc final
```

3




Exemple : pour afficher le signe d'un nombre on peut utiliser le code suivant.

```
if x > 0:
    print('x est positif)
else:
    print('x est négatif')
```

4

**Exercice 3**

En arithmétique, un nombre parfait est un nombre dont la somme des diviseurs stricts (c.a.d sans le nombre lui-même) est égale à ce nombre.

Compléter le programme suivant afin qu'il permette de tester si un nombre est parfait. On rappelle que a%b donne le reste de la division de a par b.

```
n = int(input('Entrez un nombre : '))
S = ..........
for k in range(1, n):
    if ..........:
        S = .........
if ..........:
    print(n, "est parfait")
else:
    print(n, "n'est pas parfait")
```

5

**Exercice 4**

On veut faire jouer l'ordinateur à un jeu de fléchette : il choisit les deux coordonnées $(x, y)$ d'un point au hasard dans l'intervalle [-1, 1]. Si la distance à l'origine est plus petite que 1 on affiche « Gagné ! » et « Perdu » sinon.

Pour obtenir un nombre aléatoire entre -1 et 1, on pourra utiliser la fonction `random` du module `random` de la manière suivante :

```
>>> random()*2-1
-0.22043013322226201
>>> random()*2-1
0.8646082373924839
```

6

On rappelle que la distance entre deux points se calcule par :

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

**Entraînement 1**

On reprend l'exercice précédent mais on souhaite maintenant compter le nombre de réussites lors du tirage de 4000000 fléchettes.

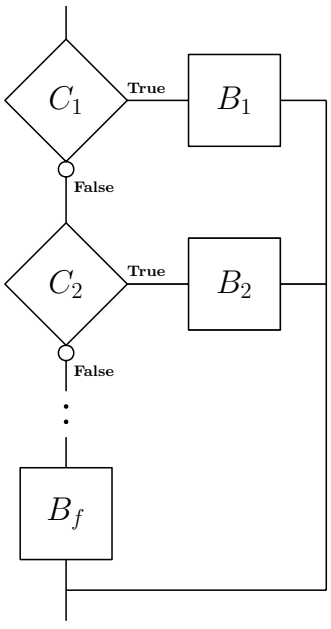
Faire afficher le nombre de réussites et diviser ce nombre par 1000000. Que remarquez-vous ?

## I.3 if, elif .. et else

Lorsque l'on veut un test avec plusieurs alternatives on intercale autant de conditions que l'on veux entre le `if` et le `else` avec des `elif` (contraction de *else* et *if*). On a alors la structure générale d'un test :

```
if Condition1:
    instruction Bloc1
    instruction Bloc1
    ...
elif Condition2:
    instruction Bloc2
    instruction Bloc2
    ...
elif ...
    ....
else:
   instruction Bloc final
   instruction Bloc final
   ....
```

7



Seul le premier mot-clé `if` est obligatoire, les `elif` et le `else` sont optionnels. Noter que `else` n'est pas suivi d'une condition. Les conditions sont des variables de type booléen [2] ou des expressions qui sont évaluées sous forme de booléen.

Un seul des blocs d'instructions peut être exécuté : le premier possible. Ceci a lieu y compris si l'état courant rend plusieurs des conditions valides.

Les conditions sont souvent construites à l'aide des opérateurs de comparaison :

| Opérateur | test effectué |
|---|---|
| == | égalité |
| != | différent |
| <= | inférieur ou égal |
| >= | supérieur ou égal |
| < | inférieur strict |
| > | supérieur strict |

**Exercice 5**

On rappelle que les années bissextiles sont celles qui sont multiples de 4 sauf si elles sont multiples de 100 mais pas de 400. Compléter le code suivant pour qu'il affiche « Année bissextile » ou « Année non bissextile »selon la valeur du nombre rentré par l'utilisateur.

```
n = int(input("Entrez le nombre de l'année : "))

if ..................:
    print('Année non bissextile')
elif ................:
    print('Année bissextile')
elif ................:
    print('Année non bissextile')
else:
    print('Année bissextile')
```

8

## I.4 Opérateur booléen

Les conditions d'un test sont toujours évaluées sous la forme d'un booléen (on rappelle que le type booléen ne possède que deux valeurs : `True` et `False`).

Les opérateurs booléens permettent de combiner les conditions, il y en a trois, du plus prioritaire au moins prioritaire : `not`, `and` et `or`. Les opérations effectuées s'appellent respectivement *négation*, *conjonction* et *disjonction*.

négation

| C | not C |
|---|---|
| True | False |
| False | True |

conjonction

| C1 | C2 | C1 and C2 |
|---|---|---|
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

disjonction

| C1 | C2 | C1 or C2 |
|---|---|---|
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

2. Que l'on utilise **directement** : pas de « if flag == True : » mais simplement « if flag : ».

**Exercice 6**

Les opérateurs `and` et `or` sont paresseux en Python. C'est à dire que dans `C1 and C2` et `C1 or C2` après l'évaluation de la première condition la deuxième n'est pas évaluée si cela ne sert à rien pour déterminer le résultat.

Pour chacun des deux opérateurs, indiquez précisément dans quelle situation la condition C2 n'est pas évaluée.

---

---

---

---

**Exercice 7**

Compléter le programme suivant pour tester si le point M est dans le rectangle défini par les deux sommets opposés A et B. La réponse sera donnée par un affichage du type : « M est dans le rectangle » ou « M n'est pas dans le rectangle ».

```
(x_A, y_A) = (1, 5)
(x_B, y_B) = (4, 3)

x = float(input("Entrez l'abscisse de M : "))
y = float(input("Entrez l'ordonnée de M : "))
```

**Entraînement 2**

Compléter le programme suivant pour tester si le point M est dans le cercle de centre A et de rayon r. La réponse sera donnée par un affichage du type : « M est dans le cercle » ou « M n'est pas dans le cercle ».

```
(x_A, y_A) = (2, -3)
r =4

x = float(input("Entrez l'abscisse de M : "))
y = float(input("Entrez l'ordonnée de M : "))
```

**Entraînement 3**

Écrire un programme qui demande un nombre entier à l'utilisateur et qui affiche un message pour indiquer si ce nombre est premier.

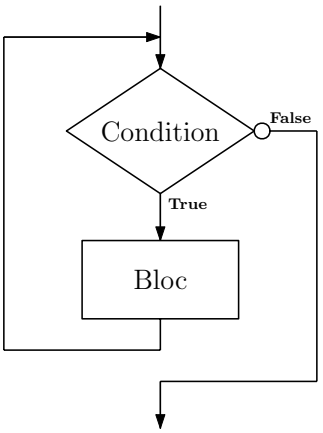
Rappel : Un nombre est premier lorsqu'il n'est divisible que par deux entiers (1 et lui-même).

# II Boucle conditionnelle

Dans une boucle **for**, le nombre d'itérations est connu à l'avance. Lorsque l'on ne sait pas à l'avance combien de fois la boucle devra être exécutée, on utilise une boucle **while**.

La syntaxe est la suivante :

```
while condition:
    instruction du bloc
    instruction du bloc
    instruction du bloc
```




La condition est une expression qui doit pouvoir être évaluée sous forme de booléen. Tant que sa valeur est True, le bloc d'instructions est exécuté.

**Attention,** lors de l'utilisation de **while**, il faudra bien prendre gare à ce que la condition finisse par prendre la valeur False sans quoi la boucle ne se terminera pas!!

Remarque : Pour faire une opération jusqu'à obtenir une certaine condition il suffit d'ajouter l'opérateur logique **not**().

Un exemple typique est celui de la division euclidienne par soustractions successives. Ici `n` est un entier positif et $\texttt{d} > 0$.

```
n = int(input('Entrez le dividende :'))
d = int(input('Entrez le diviseur :'))
q = 0
while n >= d:
    n = n - d
    q = q + 1
print('Le quotient est : ', q)
print('Le reste est : ', n)
```

9

**Exercice 8**

Compléter un tableau d'état pour l'exécution dans le cas de la division de 25 par 7.

**Exercice 9**

Montrer qu'une boucle **for** k **in range**(a, b, p) peut être obtenue à l'aide d'un **while**.

**Exercice 10 : Saisie contrainte**

Écrire un programme qui force l'utilisateur à entrer un nombre entier compris entre 1 et 100 (si la saisie n'est pas conforme, elle est redemandée).

**Entraînement 4** 

Écrire un programme qui permet de saisir un mot de passe sous la forme d'une chaîne de caractères et qui ensuite redemande ce mot de passe jusqu'à ce qu'il soit correct.

---

**Exercice 11**

On appelle logarithme entier d'un nombre réel $x \geqslant 1$, le nombre de fois qu'il faut le diviser par 2 pour obtenir un nombre inférieur à 1.

Écrire un programme qui permet de saisir un nombre $x$ et qui affiche son logarithme entier.

**Exercice 12**

Écrire un programme qui permet à l'utilisateur de rentrer une chaine de caractères et qui indique si la sous-chaine `'NSI'` est présente dans cette chaîne : Si oui, on affiche la position où apparait la sous-chaîne pour la première fois, sinon on affiche `'NSI' n'est pas présent`.

Aide : pour obtenir le caractère en position `i` dans une chaîne `C`, on utilise simplement `C[i]`. Attention les indices commencent à 0.