Lorsqu'on écrit un programme, on est souvent amené à prendre en compte plusieurs cas de figure et à adapter le comportement de celui-ci en conséquence. Pour ce faire, on utilise la l'instruction de branchement `if` suivi d'une condition logique.

1 Problème : compter les points au mölkky

Au jeu du mölkky, chaque joueur marque à son tour entre 0 et 12 points qui viennent s'ajouter à son score précédent. Le premier joueur à atteindre un score exact de 51 gagne, son score étant ramené à 25 en cas de dépassement.

Tester le programme suivant :

```
score = int(input("Entrer le score :"))
gain = int(input("entrer le gain :"))
nouveau_score = score+gain
print("Nouveau score :", nouveau_score)
```

Ce programme répond au problème dans le cas où le nouveau score est inférieur ou égal à 50. Si le nouveau score est de 51 on peut écrire :

```
print("Gagne !")
```

Et si le nouveau score est supérieur à 51 on peut écrire :

```
nouveau_score = 25
print("Trop grand ! Nouveau score : 25")
```

Selon les cas de figure, on veut pouvoir exécuter l'une ou l'autre de ses trois parties de code. Voyons comment.

2 Conditions et branchements

Pour effectuer un branchement conditionnel, c'est à dire un découpage du code en différentes parties exécutée ou non selon certaines conditions, on utilise les commandes `if/elif/else` :

L'instruction conditionnelle `if` permet de n'exécuter du code qu'à une certaine condition :

```
if <condition>:
    <bloc a executer si la condition est remplie>
```

Celle ci peut être complétée d'un `else` si on veut exécuter un code alternatif quand la condition n'est pas remplie. On parle de branches conditionnelles :

```
if <condition>:
    <bloc a executer si la condition est remplie>
else:
    <bloc a executer sinon>
```

Enfin, on peut rajouter d'autres branches après le premier `if` avec la commande `elif` :

```
if <condition1 >:
    <bloc a executer si la condition 1 est remplie>
elif <condition 2>:
    <bloc a executer si la condition 1 n est pas remplie mais la 2 oui>
elif <condition 3>:
    <bloc a executer si les condition 1 et 2 ne sont pas remplies mais la 3 oui>
...
else:
    <bloc a executer si aucune des conditions precedentes n est remplie>
```

Une fois revenu au niveau d'indentation initial (aligné avec le `if`) on sort du branchement conditionnel et les instructions suivantes sont exécutées normalement. On parle de jonction.

```
if <condition>:
    <bloc execute si la condition est remplie>
else:
    <bloc execute sinon>
<reste du code toujours execute>
```

Pour construire une condition, on a couramment recours à une comparaisons entre deux expression numériques. Tester par exemple le programme suivant :

```
n = int(input("Entrer un entier n :"))
if n>0:
    print("n strictement positif.")
elif n==0:
    print("n nul.")
else:
    print("n strictement negatif.")
```

Ici, on compare l'expression `n` à l'expression `0`. Bien sûr les expressions peuvent être arbitrairement complexes. Les opérateurs de comparaison disponibles sont les suivants :

`>`	plus grand que	`>=`	supérieur ou égal à	`==`	égal à
`<`	plus petit que	`>=`	inférieur ou égal à	`!=`	différent de

Attention : Il ne faut surtout pas confondre les commandes `n=0` et `n==0`. La première affecte la valeur `0` à la variable `a` alors que la seconde teste si la valeur de `a` est égale à `0`. On n'écrira donc jamais (sous peine de déclencher une erreur de l'interpréteur) la commande `if n=0: ...`

On peut maintenant résoudre le problème du mölkky :

```
score = int(input("Entrer le score :"))
gain = int(input("entrer le gain :"))

nouveau_score = score+gain

if nouveau_score < 51:
    print("Nouveau score :", nouveau_score)
elif nouveau_score == 51:
    print("Gagne !")
else:
    nouveau_score = 25
    print("Trop grand ! Nouveau score : 25")

print("Fin du tour")
```

3 Conditions et booléens

On s'attarde ici sur les conditions. Tester les commandes suivantes dans le mode interactif de Python :

- `1==1`
- `1==0`
- `1>-1`
- `1<=2`

On voit que l'interpréteur Python effectue et affiche un calcul dans chaque cas dont la réponse est `True` (vrai) ou `False` (faux). Ces deux valeurs sont appelées des booléens, ici provenant de l'évaluation des conditions.

De même que l'on a des opérations pour les nombres, on a des opérations sur les booléens / les conditions. Si `c1` et `c2` sont des conditions, on peut en construire de nouvelles :

Commande	**Nom**	**Description**
`c1 and c2`	conjonction	vraie lorsque les deux conditions sont vraies
`c1 or c2`	disjonction	vraie lorsque au moins l'une des deux condition est vraie
`not c1`	négation	vraie lorsque la condition est fausse

Tester le programme suivant et faire un schéma pour comprendre les conditions données ici. Les occasions d'utiliser des instructions conditionnelles en géométrie sont très fréquentes.

```
xa = int(input("Abscisse A :"))
ya = int(input("Ordonnee A :"))
xb = int(input("Abscisse B :"))
yb = int(input("Ordonnee B :"))
if xa==xb and ya==yb:
    print("Points confondus")
elif xa==xb or ya==yb:
    print("Points alignes verticalement ou horizontalement")
else:
    print("Point independants")
```

Priorité des opérateurs booléens : Lorsqu'on utilise plusieurs opérateurs booléens dans une expression, les priorités de calculs sont parenthèse, `not`, `and` puis `or`. Par exemple, l'expression `x>0 and y<3 or not z==0` est équivalent avec des parenthèses à `(x>0 and y<3) or (not (z==0))`. Cependant, même si ces règles existent, on préférera utiliser des parenthèses pour rendre les conditions plus lisibles.

Paresse des opérateurs booléens : Dans certains cas, la connaissance de la valeur de vérité de la première condition d'un `and` ou d'un `or` rend inutile l'évaluation de la seconde. Plus précisément :

— si `c1` est vraie, alors `c1 or c2` est vraie quelque soit la valeur de `c2` ;

— si `c1` est fausse, alors `c1 and c2` est fausse quelque soit la valeur de `c2`.

Dans ces cas là, l'interpréteur n'effectue pas le calcul de `c2`. Cela s'avérera très utile pour les boucles conditionnelles (`while`). Par exemple dans le programme suivant :

```
x = int(intput("Entrer un nombre x :"))
if 2<=x and x<=7:
    print("x dans l'intervalle [2,7]")
```

Si `x` a pour valeur `1`, seule la première condition est évaluée. En revanche, si `x` a pour valeur `5` les deux conditions sont évaluées.

Chaînes de comparaisons : En Python (ce n'est pas le cas dans tous les langages) il est possible d'écrire des inégalités ou égalités de plusieurs termes. Par exemple `2<=x<=7` ou encore `x==y==2`. Ces chaînes de comparaisons sont tout simplement interprétées comme des conjonctions, ici respectivement `2<=x and x<=7` ou encore `x==y and y==2`.

```
x = int(input("Entrer un nombre :"))
y = int(input("Entrer un nombre :"))
z = int(input("Entrer un nombre :"))
if x<=y<=z:
    print("Nombres donnes dans l ordre ")
```

Des variables booléennes : On a vu qu'une condition était à un moment évaluée sous la forme d'un booléen (`True` ou `False`). Il est tout à fait possible de stocker l'évaluation d'une condition dans une variable, variable que l'on peut utiliser de la même manière qu'une condition avec les instructions `if`/`elif`/`else` :

```
x = int(input())
y = int(input())
b = x<y
if b:
    print("x plus petit que y")
```

Dans ce cas de figure on essaiera d'éviter l'erreur classique des débutants : `if b == True:` qui a strictement le même effet mais en faisant faire un calcul inutile à l'interpréteur.

4 Instructions conditionnelles imbriquées

Le bloc d'instruction d'un branchement conditionnel est un bloc d'instruction quelconque qui peut lui aussi contenir des branchements conditionnels. On parle alors d'instructions conditionnelles imbriquées.

Par exemple, on peut chercher à résoudre le problème suivant : étant donnés trois nombres `a`, `b` et `c` donnés dans n'importe quel ordre, a-t-on `b` compris entre `a` et `c`. On peut voir deux cas de figure :

— si `a<=b` alors la réponse est équivalente à `b<=c` ;

— sinon (`a>b`) alors la réponse est équivalente à `b>=c`.

On peut donc écrire :

```
a = int(input())
b = int(input())
c = int(input())
if a<=b:
    if b<=c:
        print("Dedans")
    else:
        print("Dehors")
else:
    if b>=c:
        print("Dedans")
    else:
        print("Dehors")
```