Algorithmique et programmation 1

L1 Mathématiques - L1 Informatique

Semestre 1

Chapitre 3 : Instructions Répétitives - Compléments

Dans ce chapitre vous allez apprendre à :

- quitter une boucle sans que la condition d'arrêt implicite soit atteinte ;
- éviter que certaines itérations ne soient réalisées.

Remarque : Ces compléments sur les boucles sont destinés à des **programmeurs avertis** !!!

Sortie impérative de boucles : `break` et `continue`

Nous allons voir dans cette partie deux instructions `break` et `continue`. Cependant, ces instructions sont reservées aux programmeurs confirmés, qui ont bien compris la notion de boucle. Autrement dit :

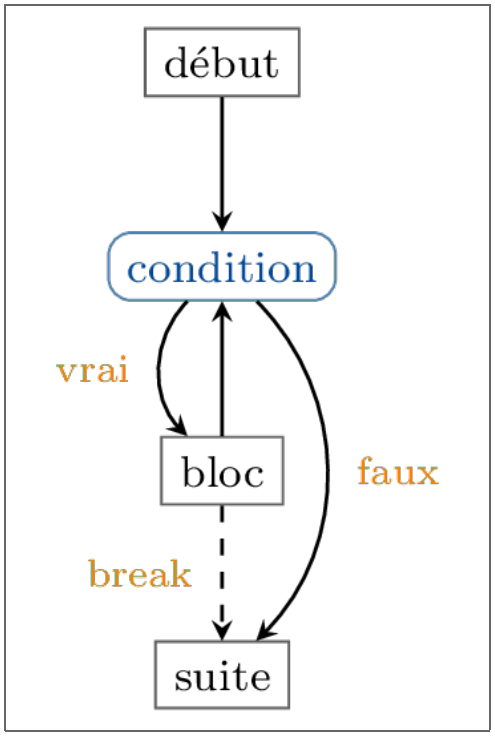
vous devez utiliser break et continue avec prudence.

En effet, trop de clauses `break` et `continue` produiront une boucle avec de nombreux points de sortie et rendront le programme difficile à lire.

1) Instruction `break`

On peut sortir prématurément d'une boucle `while`

- Instruction `break` dans le corps (en général dans une conditionnelle)
- Force une sortie de boucle et passe directement à la suite du programme
- On ne réévalue pas la condition du `while`



Exemple 1 : Une saisie contrôlée en évitant la duplication de l'instruction `input`

In [1]:

```
while True:
    note_cc1 = float(input('Note du premier contrôle : '))

    # Si la saisie est correcte, on termine la boucle
    if 0 <= note_cc1 <= 20:
        break

    print('Erreur de saisie.')
```

```
Note du premier contrôle : -2
Erreur de saisie.
Note du premier contrôle : 12
```

**Exemple 2 : ** Une première méthode pour savoir si un nombre est premier
Un nombre `N` est dit **premier** lorsqu'il possède exactement deux diviseurs, `1` et `N`.
Par exemple, `2`, `3`, `5`, `7`, `11`, `13`, `17`, `19` sont des nombres premiers. Par contre, `1` n'est pas premier car il n'admet qu'un seul diviseur : `1` ...

Pour tester si un nombre est premier, a priori, il faut chercher ses diviseurs pamis *tous les nombres compris entre * 2 *et* n - 1 . Mais, on peut faire mieux :
*ne chercher les diviseurs que parmi les nombre `2` et `rce(n)` où `rce(n)`désigne la racine carrée entière de `n`.  En effet, si `n = pq`, avec `p` et `q` différent de `1`, alors nécessairement `p * p < n` ou `q * q < n`. Sinon, on aurait `p * p > n` et `q * q > n`, d'où `n * n = p * q * p * q = p * p * q * q > n * n`...

In [2]:

```python
n = int(input("Donnez moi un nombre : "))

#Calcul de la racine entière
rce_plus_1 = 1
while rce_plus_1 * rce_plus_1 <= n:
    rce_plus_1 += 1
#Recherche des diviseurs
premier = False
for div in range(rce_plus_1):
    if div != 0 and div != 1 and n % div == 0: #Teste si div >= 2 est bien un diviseur de n
        print(n, "est divisible par", div)
        premier = True
        break #On a trouvé un diviseur, donc n n'est pas premier : on s'arrete, car on a la reponse.
if not premier:
    print(n, "est un nombre premier")
```

```
Donnez moi un nombre : 197
197 est un nombre premier
```

2) Instruction `continue`

Au cours d'une boucle, on peut passer prématurément à l'itération suivante grâce à l'instruction `continue` . En tête du corps de la boucle :

- on teste si l'on est dans certains cas problèmatiques pour le traitement désiré ; dans ce cas, l'instruction `continue` permet de passer à l'itération suivante.
- si l'on n'est pas dans un cas problèmatique, on évalue le reste du corps de la boucle.

In [3]:

```
for i in range(12):
    if i % 5 == 0:
        continue
    print(i)
```

```
1
2
3
4
6
7
8
9
11
```

Remarques : On peut **toujours** se passer de l'instruction `continue`, en utilisant des conditionnelles. Simplement, dans des programmes de grandes envergures, cela peut coûter cher en temps d'exécution ou créer beaucoup d'indentation (ce qui rends le programme moins lisible)

In [4]:

```
for i in range(12):
    if i % 5 != 0:
        print(i)
```

```
1
2
3
4
6
7
8
9
11
```