


3. PARTIE 3 : Le langage de programmation Python : Conditions et boucles

3.1. structure conditionnelle if ... then ... else

3.1.1. structure simple


```
1  if <condition> :
    <instruction1>
3  else:
    <instruction2>
```

- Le mot `then` n'apparaît pas en python
- Le bloc `else` n'est pas obligatoire
- Le `:` symbole est obligatoire après les mots de la structure (IF et ELSE)
- C'est l'indentation qui délimite les blocs d'instruction.
- Il n'y a pas d'instruction `END`

 Exercice 1 : Écrire une fonction `chercheParite()` qui prend pour paramètre un nombre et renvoie `True` si la valeur est paire et `False` sinon. On pourra faire une structure de vérification pour tester si le nombre entré est un entier positif.

```
1  >>> chercheParite(4)
    True
3  >>> chercheParite(9)
    False
```

`.\scripts\part3\exo1.py et exo1_assert.py`

 Exercice 2 : Écrire une fonction `cherchePositif()` qui prend pour paramètre un nombre et renvoie `True` si la valeur est positive et `False` sinon.

```
1  >>> cherchePositif(4)
    True
3  >>> cherchePositif(-9)
    False
```

`.\scripts\part3\exo1.py`

3.1.2. Structure ELIF

```
1  if <condition1> :
    <instruction1>
3  elif <condition2> :
    <instruction2>
5  elif <condition2> :
    <instruction3>
7  else :
    <instruction3>
```



Exercice 3: Écrire une fonction `cherchePositif2()` qui prend pour paramètre un nombre et renvoie

"le nombre est strictement positif", "le nombre est nul" ou "le nombre est strictement négatif" en fonction des cas.

```
1 >>> cherchePositif2(4):
    Le nombre est strictement positif
```

`.\scripts\part3\exo3.py`



Exercice 4: Écrire une fonction `signeProd()` qui reçoit deux entiers a et b relatifs et qui renvoie `True` si le produit est positif et `False` sinon. Attention : on ne doit pas calculer le produit des deux nombres.

```
1 >>> signeProd(4, -2)
    False
```

`.\scripts\part3\exo4.py`

3.1.3. Test d'appartenance

Il est très simple de tester si une valeur appartient ou pas à un ensemble.

```
1 >>> mot = 'abcd'
2 >>> 'a' in mot
    True
4 >>> 'g' in mot
    False
```

Remarque : Les chaînes de caractère doivent être entourées de `'` ou `"`. Utiliser l'un permet de mettre l'autre dans le texte, sur la même ligne.



Exercice 5 : Écrire une fonction `testAppartient()` qui renvoie `True` si la lettre appartient au mot.

```
1 >>> testAppartient('a', 'abracadabra')
    True
3 >>> testAppartient('g', 'abracadabra')
    False
```

Remarque : L'instruction `'a' in mot` renvoie un booléen. (`True` ou `False`)

3.2. La boucle FOR

```
1 for <compteur> in range (a,b,pas):
    <instruction>
```


- On peut mettre un nom de compteur quelconque.
- Le compteur va varier de **a** à **b - 1**. On peut rajouter un **pas** entier. Par défaut : **a = 0** et **pas = 1**

```
1 def compte(n):                # Fonction comptant de 0 a n
    for i in range (n+1):      # i varie de 0 a n
        print(i)              # On affiche la valeur de i

5 def decounte(n):              # Fonction decountant de n a 0
    for i in range(n+1):      # i varie de 0 a n
        print(n-i)           # On affiche la valeur n-i. La valeur decroit


9 def decounte2(n):             # Fonction decountant de n a 0
    for i in range(n,-1,-1):  # i varie de n a 0
        print(i)             # On affiche la valeur i. La valeur decroit
```

.\scripts\part3\exemple1.py

 Exercice 6 : Écrire une fonction `puissance2()` prenant un entier n comme paramètre et affichant les puissances de 2 jusqu'à 2^n


```
1 >>> puissance2(5)           # Affiche les 5 premieres puissances de 2
    2 4 8 16 32               # Affiche de 2**1 a 2**5
```

.\scripts\part3\exo6.py

 Exercice 7 : Écrire une fonction `somme()` prenant un entier n comme paramètre et affichant la somme des n premiers entiers non nuls.


```
1 >>> somme(5)                 # Calcule la somme des 5 premiers entiers non nuls
    15                         # 1 + 2 + 3 + 4 + 5 = 15
```

.\scripts\part3\exo7.py

 Exercice 8 : Écrire une fonction `tableMulti()` prenant un entier n comme paramètre et affichant la table de multiplication de cette valeur. Pour $n = 7$, l'affichage devra ressembler à :

```
1 La multiplication de 0 par 7 est 0
2 La multiplication de 1 par 7 est 7
3 La multiplication de 2 par 7 est 14
4 ...
```

.\scripts\part3\exo8.py

 Exercice 9 : Une somme S est placée à la banque avec des intérêts de p% par an.

Écrire une fonction `capital()` prenant trois valeurs en paramètres (la somme versée, le nombre d'année et le pourcentage d'intérêt) et renvoyant le capital accumulé au bout de ce nombre d'années.

```
1 >>> capital(100,15,5)        # Test pour S = 100, n = 15 et interet = 5%
    207.89281794113666
```

.\scripts\part3\exo9.py

 Exercice 10 : Écrire une fonction `affichePair()` qui affichera toutes les valeurs paires entre 0 et un entier n.

```
1 >>> affichePair(9):
    0 2 4 6 8
```

`.\scripts\part3\exo10.py`

3.3. La boucle FOR pour balayer une liste

Il est possible d'utiliser la boucle FOR sans pour autant créer un compteur

```
mot = 'bonjour'          # chaine de caractere
for let in mot:          # let prend pour valeur chaque lettre de mot
    print(let, end="-")  # affichage
```

```
>>>
*** Console de processus distant Réinitialisée ***
b-o-n-j-o-u-r-
```

`.\scripts\part3\for_balayer_liste.py`

Il existe également un type de variable appelé `liste`

```
>>> maliste = [1,5,8,4,3]
>>> maliste [1]
5
>>> maliste [4]
3
>>> maliste [5]
Traceback (most recent call last):
File "<interactive input>", line 1, in <module>
IndexError: list index out of range
```

Cette liste :

- s'écrit entre crochets. Les valeurs sont séparées par des virgules
- les valeurs à l'intérieur de la liste peuvent être de n'importe quel type.
- chaque élément de la liste est identifié par un indice. Les index commencent à 0.
- ainsi une liste de n éléments ont des indices compris entre 0 et n-1


Les deux boucles FOR suivantes affichent exactement la même chose

```
1 maliste = [1,5,8,4,3]
2 n = len(maliste)      # len( ) permet de recuperer la longueur de la liste
3 for i in range(n):    # on balaye les indices de la liste 0 a n-1
    print(maliste[i], end="_")

6 for elt in maliste:   # on fait prendre elt chaque valeur de la liste
    print(elt,end="_")

9 >>>
10 1 5 8 4 3 1 5 8 4 3
```

`.\scripts\part3\for4.py`

 Exercice 11 : Écrire une fonction `estDansListe` qui teste si une valeur est dans une liste. Proposez deux versions des programmes : l'une avec un balayage d'indice et l'autre sans.

```
1 >>> maliste = [4,2,1,5]
2 >>> estDansListe(2,maliste)
3 True
4 >>> estDansListe(7,maliste)
5 False
```

`.\scripts\part3\exo11.py`

3.4. La boucle WHILE

```
1 while <condition vraie>:      # Tant que la condition est vraie
    <instruction1>              # Instruction a effectuer
```

- La boucle est effectuée tant que la condition est vraie.
- Dès que la condition est fausse, on sort de la boucle while et on continue le programme.
- On pourra toujours remplacer une boucle FOR par un WHILE mais pas forcément l'inverse.
- Le WHILE est à utiliser lorsqu'on ne sait pas à priori combien de boucles il faut effectuer.

Voici quelques exemples utilisant la boucle WHILE


```
1 n = 0                                # Initialisation de n
2 while n<5:
    print(2**n, end="_")              # Affichage de 2**n de n=0 jusqu'à n=4
    n = n + 1                         # Incrementation de n
5 >>>
6 1 2 4 8 16

8 n = 0                                # Initialisation de n
9 u = 1                                # Initialisation de u
10 while u > 0.00001:                 # Recherche de seuil
    u = u * 0.9                       # Mise a jour de u (diminution de 10%)
    n = n + 1                         # Incrementation de n
13 print(u,n)
14 >>>
15 9.261387130997904e-06 110

17 cherche = True                     # Initialisation de cherche (booléen)
18 n = 0                               # Initialisation de n
19 while cherche:                     # Tant que cherche est True
    if 2**n > 1000:                   # Recherche de seuil
        cherche = False              # Mise a jour de cherche si au dessus du seuil
    else:
        n = n + 1                    # Incrementation de n
24 print(n)
25 >>>


28 while True:                        # Boucle infinie !!!!!
    print('a')
```

.\scripts\part3\exemplesWhile.py

 Exercice 12 : Écrire une fonction `sommeEntier()` en utilisant la boucle while qui effectue la somme des n premiers entiers non nuls.

```
1 >>> sommeEntier(8)
2 36
```

.\scripts\part3\exo12.py

 Exercice 13 : Écrire une fonction `sommeEntier2()` en utilisant la boucle while qui renvoie la somme des n premiers entiers non nuls jusqu'à ce que cette somme soit supérieure à une valeur de seuil.

.\scripts\part3\exo13.py

🔥 Exercice 14 : Une piscine remplie de 150 litres d'eau fuit et perd 1% de son volume tous les jours. Pour compenser cette perte, on ajoute 1 litre par jour.

1. Écrire une fonction `calculVolume()` en utilisant la boucle while qui renvoie le volume d'eau présent dans la piscine au bout de n jours.

```
1 >>> calculVolume(100)
2 118.30161706366135
```

`.\scripts\part3\exo14.py`

2. Écrire une fonction `rechercheSeuil()` en utilisant la boucle while qui renvoie le nombre de jours pour lequel la quantité d'eau dans la piscine sera inférieure à 120 litres.

```
1 >>> rechercheSeuil(120)
2 92
```

`.\scripts\part3\exo14seuil.py`

🔥 Exercice 15 : Écrire une fonction `plusOuMoins()` qui renvoie le nombre d'essais pour trouver un nombre aléatoire entre 0 et 100.

Pour permettre à Python de pouvoir choisir un nombre aléatoire il faudra importer la bibliothèque `random`

```
1 from random import *
2 a = randint(0,100)      # Genere un entier compris entre 0 et 100
3 b = random()            # Genere un flottant compris entre [0;1[
```

Pour demander une valeur au clavier, on utilisera exceptionnellement la commande `input`

```
a = int(input("Entrer un nombre")) # La chaine de caractere est convertie en entier
```

Le résultat final sera le suivant :

```
1 >>> plusOuMoins()
2 C'est moins
3 C'est plus
4 Bravo trouvé en 2 coups
```

`.\scripts\part3\exo15.py`