Université de Gabés
Faculté des Sciences de Gabés

LGLSI 1

Semestre 2

Programmation Python

Dhikra KCHAOU
dhikrafsegs@gmail.com

2023-2024

Chapitre 3

Les structures conditionnelles et les structures itératives

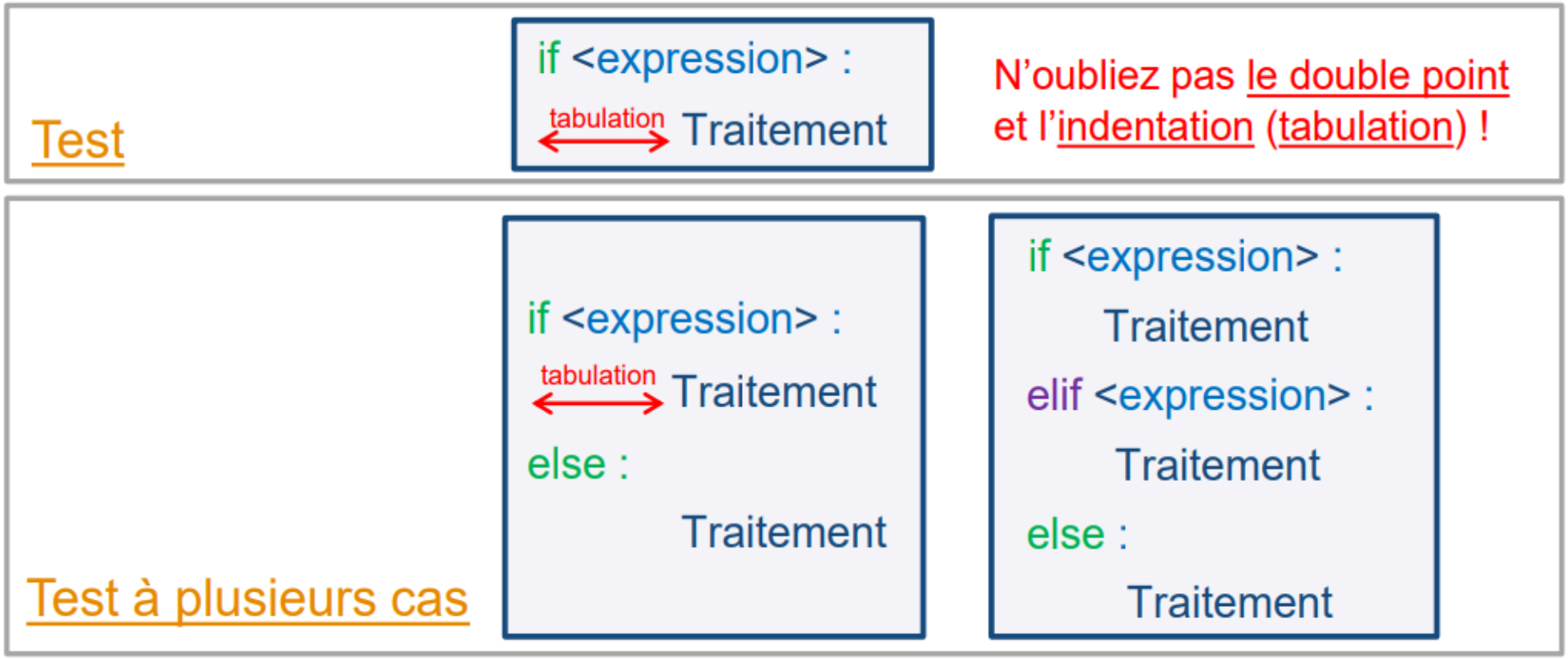
Les structures conditionnelles

Plan du chapitre

1. Présentation des structures conditionnelles
2. Le prédicat
3. Tests multiples
4. Test à plusieurs cas
5. Les structures conditionnelles imbriquées

Les structures conditionnelles

- Une **structure conditionnelle** permet d'exécuter une séquence d'instructions seulement dans le cas où une condition est vraie.



Le prédicat

- Une **condition (ou prédicat)** est une expression logique : le résultat de son évaluation est un booléen.
- Une condition peut représenter des conditions assemblées avec les **opérateurs logiques**.
- Les opérateurs logiques sont:
  - x **or** y : Ou logique
  - x **and** y : Et logique
  - **not** x : Non logique

Exemple

```
>>> x=2
>>> if x==2:
...     print('Le test est vrai')
...

Le test est vrai
>>> if x==3:
...     print('Le test est vrai')
...

>>>
```

- Dans le premier exemple, le test étant vrai, l'instruction **print("Le test est vrai !")** est exécutée.
- Dans le second exemple, le test est faux et rien n'est affiché.
- La ligne qui contient l'instruction if se termine par le caractère **deux-points« : ».**
- Les blocs d'instructions dans les tests doivent **forcément être indentés**.

Tests multiples: conditions assemblées

```
>>> x=2
>>> y=2
>>> if x==2 and y==2:
...     print("le test est vrai")
...
...
le test est vrai
```

- Respectez bien la casse des opérateurs **and** et **or** qui, en Python, s'écrivent en minuscule.

Test à plusieurs cas « else »

- Si on a **deux conditions**, on peut se servir des instructions **if** et **else**

```
>>> if x==2:
...     print('le test est vrai')
... else:
...     print('le test est faux')
...
...
le test est vrai
>>>
```

Tests à plusieurs cas « elif »

- Si on a **plus que deux conditions**, on peut utiliser l'instruction **elif** (utilisé pour tester **plusieurs valeurs d'une même variable**.)

```
>>> a=0
>>> if a > 0 :
...     print("a est positif")
... elif a < 0 :
...     print("a est négatif")
... else:print("a est nul")
...
a est nul
>>>
```

Les structures conditionnelles imbriquées

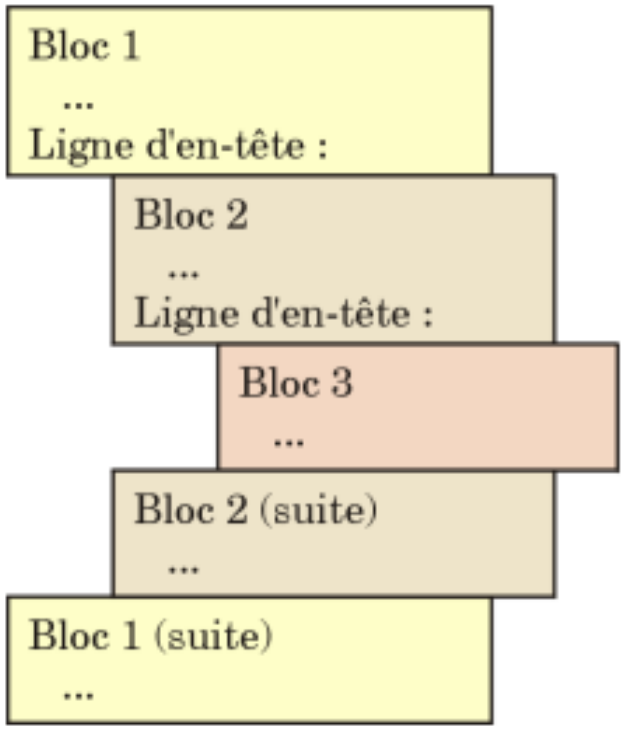
- Il est possible d'imbriquer des instructions conditionnelles les unes dans les autres, de manière a réaliser des structures de décision complexes.
- **Exemple**: Écrire un programme Python qui résout une équation du premier degré : ax+b=0

```
a=eval(input("Insérer la valeur de a"))
b=eval(input("Insérer la valeur de b"))
if a==0:
    if b==0:
        print("toute valeur de x est possible")
    else:
        print("Equation incorrecte")
else:
    print("la valeur possible de x est",-b/a)
```

```
Insérer la valeur de a5
Insérer la valeur de b4
la valeur possible de x est -0.8
```

Importance de l'indentation

- Les blocs d'instructions sont toujours associés à une ligne d'en-tête contenant une instruction bien spécifique (if, elif, else, etc.) se terminant par **un double point**.
- Les blocs sont délimités par **l'indentation** : toutes les lignes d'un même bloc doivent être indentées exactement de la même manière (c'est-à-dire décalées vers la droite d'un même nombre d'espaces). Le nombre d'espaces à utiliser pour l'indentation est quelconque, mais la plupart des programmeurs utilisent des multiples de 4.



Exercice

Ecrire un programme qui **lira** au clavier l'heure (h) et les minutes (m), et il affichera l'heure qu'il sera une minute plus tard

Exemples

h=21 et m=32 →"Dans une minute, il sera 21:33 "

h=14 et m=59 →"Dans une minute, il sera 15:0"

NB. supposer que l'utilisateur entre une heure et minutes valides (hϵ[0…23] et mϵ[0…59])

Exercice

```
h=eval(input("donner l'heure"))
m=eval(input("donner la minute"))
if m==59:
    if h==23:
        h=0
        m=0
    else:
        h+=1
        m=0
else:
    m=m+1
print(f"dans une minute, il sera {h}:{m}")
```

```
donner l'heure23
donner la minute59
dans une minute, il sera 0:0
```

```
donner l'heure12
donner la minute4
dans une minute, il sera 12:5
```

Les structures itératives

Les structures itératives

- Pour répéter une ou plusieurs instructions, on doit utiliser une boucle.
- Python propose deux sortes de boucles:
  - La boucle **while** : le nombre de répétitions est inconnu à l'avance
  - La boucle **for** : le nombre de répétitions est connu à l'avance

La boucle while

```
while <expression> :
    Instructions à répéter
```

tabulation ⟷

- La boucle **while** permet de répéter une ou plusieurs instructions tant qu'une « Expression » est vérifiée (valeur = True)
- Lorsque l'expression n'est plus vérifiée (valeur = False), aucune instruction n'est exécutée.

➡ **Remarque** : La vérification de la condition s'effectue avant l'exécution des traitements. Alors, si la condition est fausse dès le départ, **le traitement n'est jamais exécuté.**

Exemple 1

```
>>> i=1
>>> while(i<=4):
...     print(i)
...     i=i+1
...
...

1
2
3
4
```

- Il est nécessaire d'indenter le bloc d'instructions correspondant au corps de la boucle.
- Si on oublie l'indentation, Python renvoie un message d'erreur:

```
IndentationError: expected an indented block
```

Exemple 2

```
>>> a=0
>>> while a<3:
...     print(a)
```

Boucle infinie !!

Ne s'arrête jamais!

- La boucle doit contenir au moins une instruction qui **change la valeur** d'une **variable intervenant dans la condition** évaluée par while, de manière à ce que cette condition puisse devenir fausse.

La boucle while

- Une boucle while nécessite généralement trois éléments pour fonctionner correctement :

1. **Initialisation de la variable** d'itération avant la boucle.
2. **Test de la variable d'itération** associée à l'instruction while.
3. **Mise à jour de la variable** d'itération dans le corps de la boucle.

La saisie contrôlée: input + while

- La boucle while combinée à la fonction input() peut être utilisé lorsqu'on souhaite demander à l'utilisateur une valeur numérique spécifique.

```
n=int(input('Entrez un entier entre [1..10]:'))
while not (1<= n <=10):
    n=int(input('Entrez un entier entre [1..10],S.V.P:'))
```

```
Entrez un entier entre [1..10]:11
Entrez un entier entre [1..10],S.V.P:14
Entrez un entier entre [1..10],S.V.P:15
Entrez un entier entre [1..10],S.V.P:5
>>>
```

Parcours d'une séquence

- Ecrire un programme permettant d'afficher le résultat suivant:

val-0 val-1 val-2 val-3 val-4

- Utiliser la boucle while:

```
i=0
while i<5:
    print("val",i, end=" ",sep="-")
    i+=1
```

```
val-0 val-1 val-2 val-3 val-4
```

- Remarque: le nombre d'itération est connu auparavant !

➡Utiliser la **boucle for**

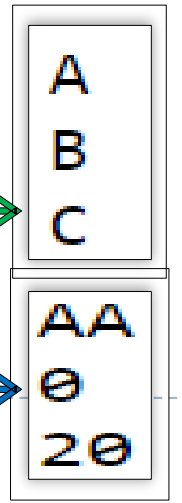
La boucle for

- La boucle for répète une séquence d'instructions autant de fois que le nombre d'éléments dans une séquence (**Conteneur ou Intervalle**)

```
for <element> in < sequence >
tabulation Instructions à répéter
```

  - Conteneur : chaîne de caractère, liste, tuple, ensemble, et dictionnaire.

```
chaine="ABC"
liste=["AA",0,20]
for val in chaine:
    print(val)
for val in liste:
    print(val)
```



La boucle for : la fonction range

- Intervalle : **range**

  range(val) : Énumère les entiers de **0→val-1**

  range(val1,val2) : Énumère les entiers de **val1→val2-1**

  range(val1,val2,val3) : Énumère les entiers de **val1→val2-1** par pas de **val3**

```
for x in range(3):
    print ("Itération", x)
```

```
Itération 0
Itération 1
Itération 2
```

```
for x in range(4,8):
    print ("Itération", x)
```

```
Itération 4
Itération 5
Itération 6
Itération 7
```

```
for x in range(4,8,2):
    print ("Itération", x)
```

```
Itération 4
Itération 6
```

```
for x in range(8,4,-2):
    print ("Itération", x)
```

```
Itération 8
Itération 6
```

```
for i in range(8,4):
    print(i)
for i in range(8,4,2):
    print(i)
```

La boucle for : la fonction range

```
for i in range(5):
    print(i, end=" ")
    i+=2
```

- **i** prend les différentes valeurs imposées par range, même si elle est modifiée dans le corps de la boucle for.

```
>>> for i in range(5):
...     print(i)
...
...
0
1
2
3
4
>>> for i in range(5):
...     print(i)
...     i+=2
...
...
0
1
2
3
4
```

La boucle for : la fonction range

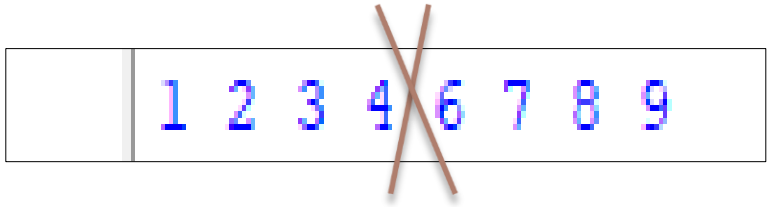
for compteur in range (val1,val2, pas)

- **val1**: représente la borne inférieure du compteur, c'est le point de départ, si on ne met pas la borne inférieure, par défaut la borne inférieure commence par 0
- **val2** : représente la borne supérieure du compteur, c'est le point d'arrêt du compteur, (cette valeur est exclue), donc il faut obligatoirement mentionner cette valeur
- **Pas** : représente la valeur d'avancement du compteur, par défaut le **pas=1**

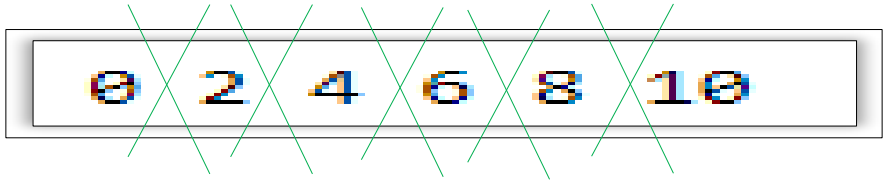
break et continue

- **break** : sort violemment de la boucle et passe à la suite
- **continue** : saute directement à l'itération suivante sans exécuter la suite du bloc d'instructions de la boucle

```
a=0
while (a>=0):
    a+=1
    if(a==5):
        continue
    elif (a==10): break
    else: print(a,end=" ")
```




```
for i in range(11):
    if i%2:continue
    else: print(i, end=" ")
```



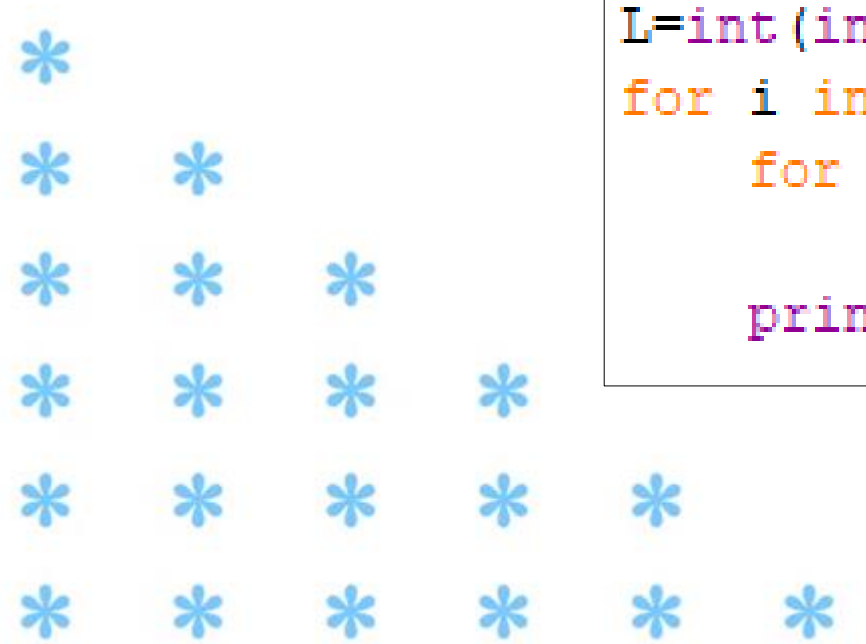
Exercice 1

- Ecrire un programme qui demande un entier entre [1...9] et qui affiche son factoriel

```
n=int(input("donner un entier"))
while not(1<=n<=9):
    n=int(input("donner un entier"))
f=1
for i in range(1,n+1):
    f=f*i
print(f)
```

Exercice 2

- Ecrire un programme qui étant donné un nombre L de lignes permet de réaliser un « triangle d'étoiles »



```
L=int(input("saisir le nomre de lignes: "))
for i in range (1,L+1):
    for j in range(i):
        print("*", end="")
    print("")
```