Cours N°3

Structures de Contrôles

1. Introduction

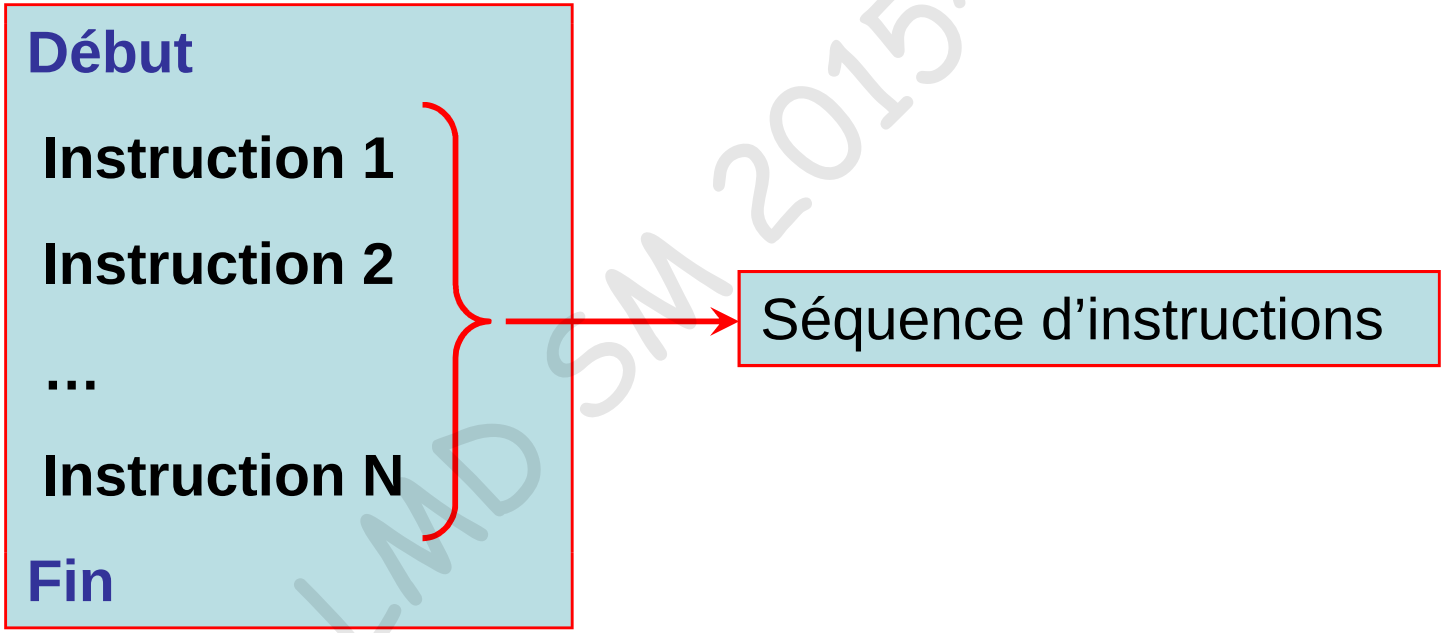
Définition :

Une structure de contrôle sert à contrôler le déroulement d'un traitement.

- Un traitement peut s'exécuter de différentes manières:
  - Séquentielle
  - Alternative (condition)
  - Répétitive (boucle)

2. Le traitement séquentiel

Le traitement séquentiel est une suite d'instruction qui s'exécutent l'une à la suite de l'autre.



Exemple: Addition de deux nombres A et B

2. Le traitement séquentiel (suite)

```
A=float(input('donnez la valeur de A: '))
B=float(input('donnez la valeur de B: '))
Somme = A+B
print('La somme est :',Somme)
```

Suite d'instructions s'exécutant séquentiellement

2. Le traitement séquentiel (suite)

Dans un programme, les instructions sont exécutées dans l'ordre de leur apparition, donc de façon séquentielle. Mais, **"l'intelligence"** d'un programme informatique provient essentiellement:

- De la possibilité de faire des choix entre plusieurs possibilités de traitement en fonction de différents critères (condition);
- De la possibilité d'exécuter rapidement une série d'instructions de façon répétitive (boucle).

3. Le traitement alternatif (structures de choix)

- L'instruction de choix permet la sélection entre deux possibilité (appelée sélection binaire).
- La condition s'exprime sous la forme:
  - d'une expression logique booléenne simple (condition simple).

EX: Si (A=B) alors …

  - ou combinée, plusieurs conditions composées avec des opérations logiques ET, OU et NON.

EX: Si (A=B) et (A=C) alors …

3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if …)

La structure conditionnelle simple se présente sous la forme suivante:

SI (condition) alors

séquence

FSI → **lire Fin SI**

Principe :

Si la condition est vérifiée (vraie) alors la séquence d'instructions s'exécute. Dans le cas contraire, ne rien faire.

3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if …)

Traduction en Python:

- Instruction simple:

  if <condition> : Instruction

- Bloc d'instructions:

```
if <condition> :
        Instruction 1
        Instruction 2
        ...
        Instruction n
```

indentation

Bloc d'instructions

3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if …)

L'importance fondamentale de l'indentation en Python:

- Dans un même bloc, deux instructions de même profondeur logique doivent avoir strictement la même indentation (décalage vers la droite).
- Avec une telle convention, il est inutile de marquer le début et la fin d'un bloc par des éléments du langage (comme des accolades { et }, ou les mots réservés **`begin`** et **`end`**).
- Il faut un respect scrupuleux des indentations, mais on est aidé en cela par l'éditeur de **`Idle`**, qui augmente automatiquement l'indentation après chaque instruction d'en tête, et qui conserve cette indentation à l'intérieur du bloc courant.

3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if …)

Exemple: Comparaison de deux variables

```
Début
1) Lire (A)
2) Lire (B)
3) SI A > B ALORS Ecrire (A ,'est supérieur à', B) FSI
Fin
```

3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if …)

Programme: Première écriture

```
A=float(input('A= '))
B=float(input('B= '))
if A > B : print (A,'est supérieur à',B)
```

Cette écriture est déconseillée

Programme: Deuxième écriture

```
A=float(input('A= '))
B=float(input('B= '))
if A > B :
    print (A,'est supérieur à',B)
```

3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if …)

Exemple2: Comparaison de deux variables – Augmentation du nombre d'instructions dans le bloc

```
A=float(input('A= '))
B=float(input('B= '))
if A > B :
      print (A,'est supérieur à',B)
      A=A+B
      print ('La nouvelle valeur de A est:',A)
print('Fin du programme')
```

3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if …)

Exemple2: Comparaison de deux variables – Augmentation du nombre d'instructions dans le bloc

```
A=float(input('A= '))
B=float(input('B= '))
if A > B : print (A,'est supérieur à',B)
           A=A+B
           print ('La nouvelle valeur de A est:',A)
print('Fin du programme')
```

Cette écriture est Fausse - A ne pas utiliser

3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if … else …)

La structure conditionnelle composée se présente sous la forme suivante:

```
SI (condition) ALORS
        séquence 1
SINON
        séquence 2
FSI
```

Principe :

Si la condition est vérifiée alors la séquence1 s'exécute.

Dans le cas contraire, c'est la séquence2 qui va s'exécuter.

3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if … else …)

Traduction en Python:

- **if <condition> : < Instruction 1>**

  else : < Instruction 2>

- **if <condition> :**

  <séquence1>

  else :

  <séquence2>

ATTENTION : le `else` est sur le même niveau d'indentation que le `if` .

3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if … else …)

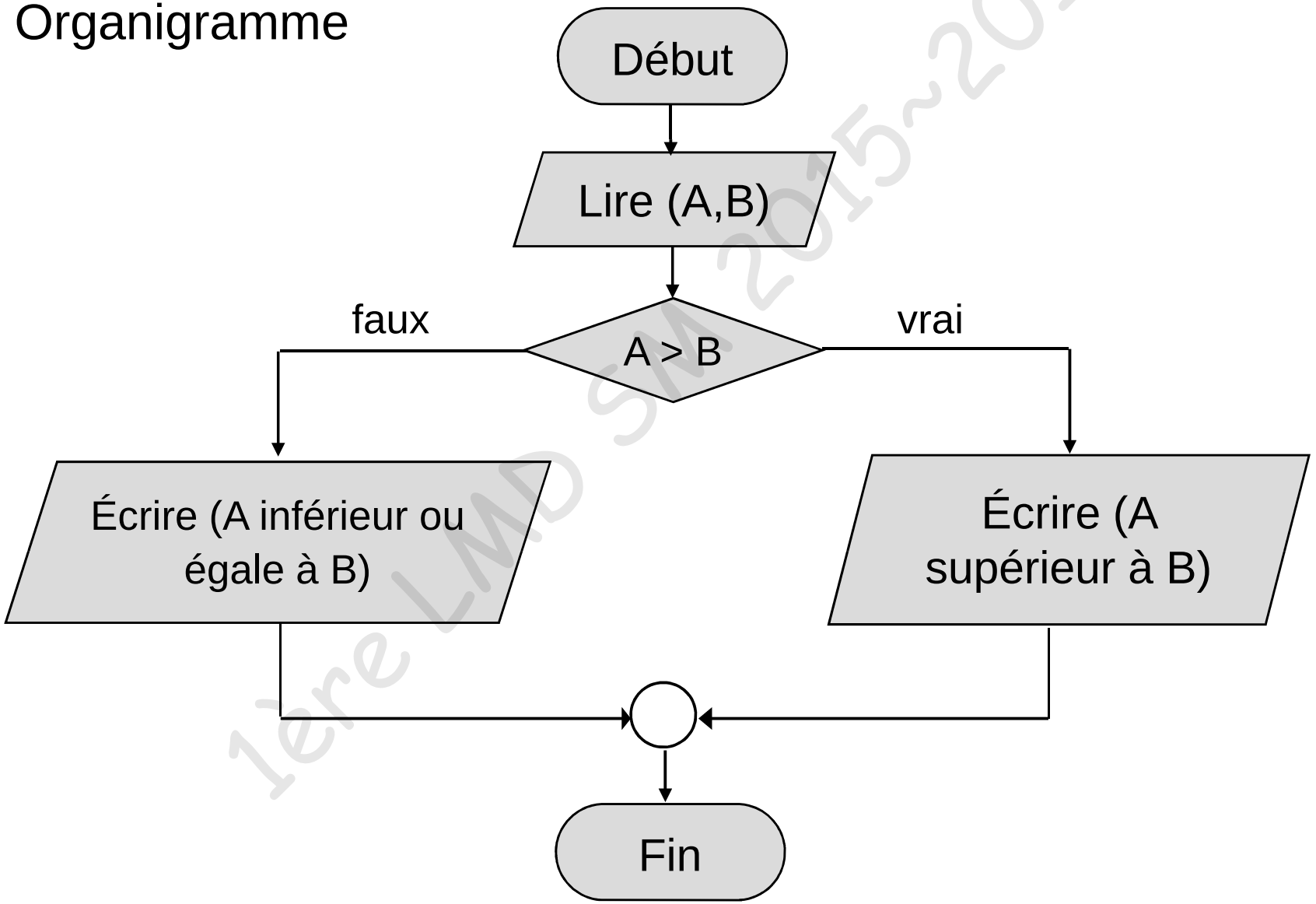
Exemple: Comparaison de deux variables (amélioré)

```
Début
1) Lire (A)
2) Lire (B)
3) SI A > B ALORS Écrire (A ,'est supérieur à', B)
4) SINON Écrire (A, 'est inférieur ou égale à', B) FSI
Fin
```

3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if … else …)

Organigramme



3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if … else …)

Programme: Première écriture

```
A=float(input('A= '))
B=float(input('B= '))
if A > B : print (A,'est supérieur à',B)
else : print (A,'est inférieur ou égale à',B)
```

Cette écriture est déconseillée

Programme: Deuxième écriture

```
A=float(input('A= '))
B=float(input('B= '))
if A > B :
     print (A,'est supérieur à',B)
else :
     print (A,'est inférieur ou égale à',B)
```

3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if … else …)

Exemple2: Ecrire un programme python qui lit un entier x et affiche ensuite s'il est pair ou impair.

Solution : Vu pendant la séance de cours.

3. Le traitement alternatif (suite)

C. Structure conditionnelle multiple (if … elif … else …)

La structure conditionnelle multiple se présente sous la forme suivante:

```
SI (condition1) ALORS
        séquence 1
SINON SI (condition2) ALORS
        séquence 2
SINON SI (condition3) ALORS
        séquence 3
…
SINON
        séquence n
FSI
```

3. Le traitement alternatif (suite)

C. Structure conditionnelle multiple (if … elif … else …)

Principe :

Si la **condition1** est vérifiée alors la **séquence 1** s'exécute. Sinon si la **condition2** est vérifiée alors la **séquence 2** s'exécute. Sinon si la **condition3** est vérifiée alors la **séquence 3** s'exécute … Dans le cas contraire (SINON) , c'est la **séquence n** qui vas s'exécuter.

Remarque :

Cette structure de contrôle présente plusieurs cas d'exécutions du traitement mais un seul cas sera exécuté.

3. Le traitement alternatif (suite)

C. Structure conditionnelle multiple (if … elif … else …)

Traduction en Python:

```
if <condition1> :
        <séquence 1>
elif <condition2> :
        <séquence 2>
elif <condition3> :
        <séquence 3>
          ... ... ... ...
else :
        <séquence n>
```

- Le mot clé **elif** est une contraction de « **else if** », que l'on peut traduire très littéralement par « **sinon si** ».
- **ATTENTION :** De même que le **else**, le **elif** est sur le même niveau d'indentation que le **if** initial. Il se termine aussi par deux points. Cependant, entre le **elif** et les deux points se trouve une nouvelle condition.

3. Le traitement alternatif (suite)

C. Structure conditionnelle multiple (if … elif … else …)

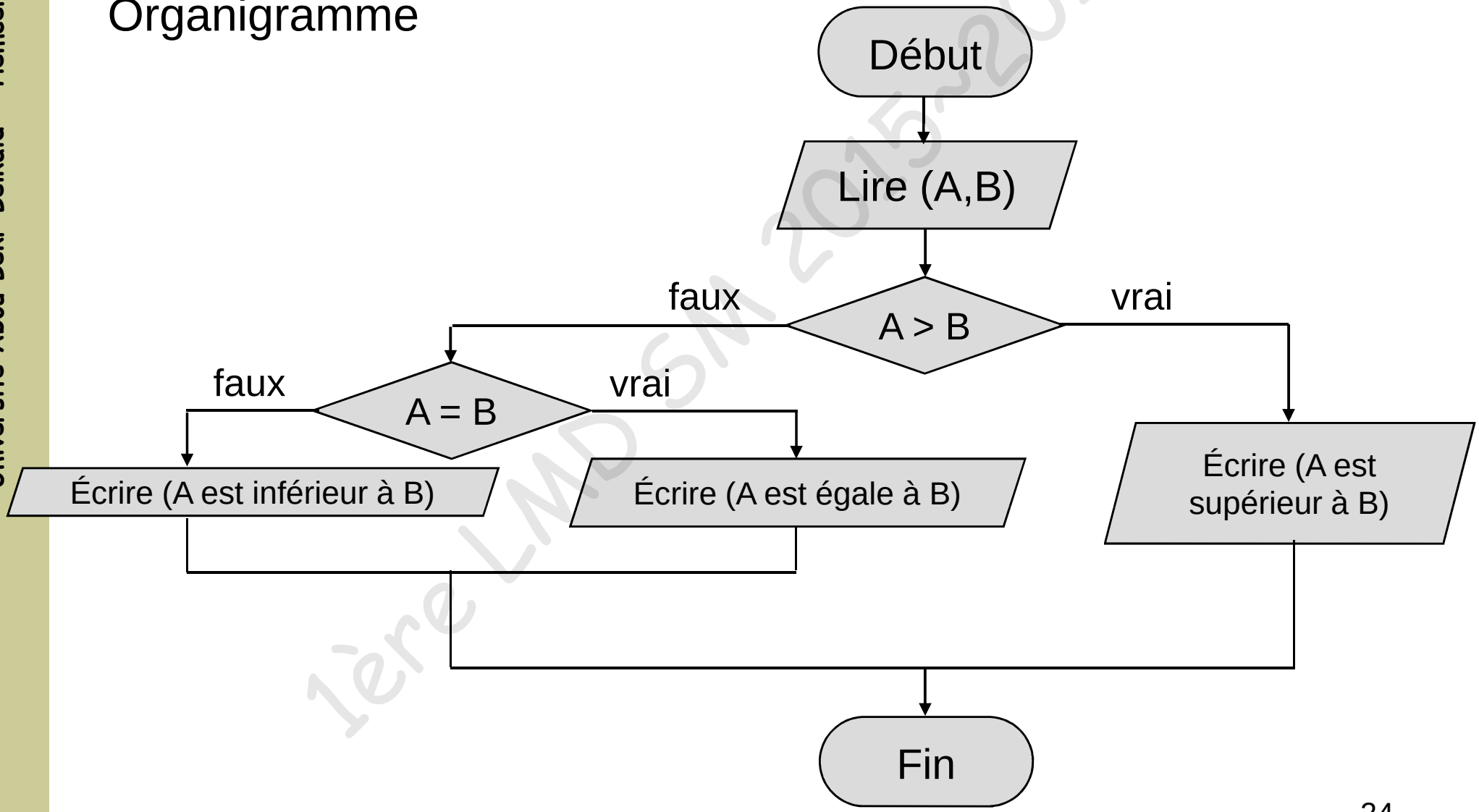
Exemple: Comparaison de deux variables (plus amélioré)

```
Début
1) Lire (A)
2) Lire (B)
3) SI A > B ALORS Écrire (A ,'est supérieur à', B)
4) SINON SI A = B Écrire (A, 'est égale à', B)
5) SINON Écrire (A, 'est inférieur à', B) FSI
Fin
```

3. Le traitement alternatif (suite)

C. Structure conditionnelle multiple (if … elif … else …)

Organigramme



3. Le traitement alternatif (suite)

C. Structure conditionnelle multiple (if … elif … else …)

Programme

```
A=float(input('A= '))
B=float(input('B= '))
if A > B :
      print (A,'est supérieur à',B)
elif A == B :
      print (A,'est égale à',B)
else :
      print (A,'est inférieur à',B)
```

3. Le traitement alternatif (suite)

C. Structure conditionnelle multiple (if … elif … else …)

Exemple: Calcul du produit, somme et la moyenne de trois réels suivant un choix

```
Début
1) Lire (nb1, nb2, nb3)
2) Lire (Choix)
3) SI Choix='1' Alors Écrire ('Le produit est : ', nb1*nb2*nb3)
4) SINON SI Choix='2' Alors Écrire ('La somme est : ',nb1+nb2+nb3)
5) SINON SI Choix='3' Alors Écrire ('La moyenne est : ',(nb1+nb2+nb3)/3)
6) SINON Écrire ('Choix incorrecte') FSI
Fin
```

Organigramme : Vu pendant la séance de cours.

3. Le traitement alternatif (suite)

Programme

```
# Programme menu
print('Entrer trois nombres réels: ')
nb1=float(input('Donnez la valeur du 1er nombre: '))
nb2=float(input('Donnez la valeur du 2ème nombre: '))
nb3=float(input('Donnez la valeur du 3ème nombre: '))
print(20*'-')
choix=input('Donnez un choix= ')
print(20*'-')
if choix == '1' :
        print ('Le produit est:',nb1*nb2*nb3)
elif choix == '2' :
        print ('La somme est:',nb1+nb2+nb3)
elif choix == '3' :
        print ('La moyenne est:',(nb1+nb2+nb3)/3)
else :
        print ('Choix incorrect')
```

4. Le traitement répétitif (les boucles)

Dans un programme, il arrive souvent qu'une ou plusieurs instructions doivent être exécuter plusieurs fois dans une structure répétitive appelée boucle.

Dans une boucle, le nombre de répétitions:

- Peut dépendre d'une condition permettant l'arrêt et la sortie de cette boucle.
- Peut être connu, fixé à l'avance.

Dans le langage Python on dispose de deux structures pour contrôler un traitement répétitif:

- La boucle "*Tant que* ".
- La boucle "*Pour* " .

4. Le traitement répétitif (suite)

A. La boucle "Tant que " (while)

La boucle "*Tant que* " se présente sous la forme suivante:

Tant que (condition) faire

séquence

Fin tant que

Principe :

Tant que la condition est vérifiée la séquence s'exécute. Elle ne s'arrêtera que lorsque la condition est invérifiable.

4. Le traitement répétitif (suite)

A. La boucle "Tant que " (while)

Traduction en Python:

- **Première écriture:**

WHILE <condition> : < Instruction >

- **Deuxième écriture**

WHILE <condition> :

<séquence>

- Attention à l'indentation !!! Lorsqu'on a un bloc d'instructions après l'instruction while, il est délimité par l'indentation.
- L'instruction Python "**while**" exécute de manière répétitive le bloc indenté tant que le test de condition est réalisé. Si la condition est d'emblée fausse le bloc ne sera jamais exécuté.

4. Le traitement répétitif (suite)

A. La boucle "Tant que " (while)

Exemple: Afficher les 100 premiers nombre entiers positifs

Début

1) i ← 1

2) Tant que i ≤ 100 faire

3) Écrire(i)

4) i ← i+1 Fin tant que

Fin

Programme

```
i=1   # Initialisation
while i<=100 :
  print(i,end=' ')
  i+=1   # Incrémentation
```

4. Le traitement répétitif (suite)

A. La boucle "Tant que " (while)

Attention ! Il faut bien choisir le test d'arrêt de la boucle sinon on obtient une ***boucle infinie***.

Exemple:

```
r = 1.5          # initialisation du nombre réel
while r > 0 : # condition vraie étant donné l'initialisation
        print(r)
        r += 2 # Incrémentation
```

C'est une **boucle infinie** parce que l'incrémentation du nombre **r** par 2 renvoie toujours une valeur positive. La condition **r > 0** est toujours vraie.

On peut forcer l'arrêt d'un script Python en tapant **[Ctrl] [C]**

4. Le traitement répétitif (suite)

A. La boucle "Tant que " (while)

Exemple: Afficher la table de multiplication d'un nombre

Solution : Vu pendant la séance de cours.

4. Le traitement répétitif (suite)

B. La boucle "Pour " (for … in …)

❑ Notion d'intervalle – Fonction prédéfinie "range" :

- On est souvent amené à répéter plusieurs fois un bloc d'instructions en fonction des valeurs successives d'un compteur.
- Une façon simple d'indiquer la plage des valeurs possibles de ce compteur est d'utiliser un intervalle (ou encore "**range**" dans le langage Python).
- La syntaxe de la fonction "**range**" est :

```
range(a,b,h)
```

où **a**, **b**, **h** sont des ***valeurs entières***

- La valeur **a** représente le début de l'intervalle, et par défaut elle vaut 0.
- La valeur **b** représente la fin de l'intervalle.
- La valeur **h** représente le pas de l'intervalle, et vaut 1 par défaut.

4. Le traitement répétitif (suite)

B. La boucle "Pour " (for … in …)

❑ Notion d'intervalle – Fonction prédéfinie "range" :

- L'intervalle **`range(a,b,h)`** doit être vu comme une succession de valeurs, en partant de **a**, et en progressant vers **b** **(sans jamais l'atteindre !)**, dans le sens croissant ou décroissant selon que le pas est positif ou négatif. Ainsi :

o L'intervalle **`range(7)`** représente la succession des valeurs: 0 1 2 3 4 5 6

o L'intervalle **`range(1,7)`** représente la succession des valeurs: 1 2 3 4 5 6

o L'intervalle **`range(1,7,2)`** représente la succession des valeurs: 1 3 5

o L'intervalle **`range(7,2)`** est vide (ici le pas a sa valeur par défaut, c-à-d 1)

o L'intervalle **`range(7,2,-1)`** représente la succession des valeurs: 7 6 5 4 3

4. Le traitement répétitif (suite)

B. La boucle "Pour " (for … in …)

La boucle "*Pour* " se présente sous la forme suivante:

Pour compteur dans Intervalle faire

<séquence>

Fin Pour

Principe :

Pour la variable de contrôle allant d'une valeur initiale jusqu'à une valeur finale d'un intervalle, exécuter la séquence d'instruction.

4. Le traitement répétitif (suite)

B. La boucle "Pour " (for … in …)

Traduction en Python:

for <compteur> in range(<Plage des données>) :

<séquence>

- Le compteur est initialisé par l'instruction **`for`** à *la valeur initiale* et tant que *la valeur d'arrêt* n'est pas atteinte, il est automatiquement incrémenté après l'exécution des instructions du bloc à répéter.
- Dans ce cas on a :

b = valeur d'arrêt + 1

4. Le traitement répétitif (suite)

B. La boucle "Pour " (for … in …)

Exemple: Afficher les 100 premiers nombre entiers positifs

```
Début
1) Pour i Dans (1,101) faire
2)       Écrire(i) Fin Pour
Fin
```

Programme

```
for i in range(1,101):
    print(i,end=' ')
```

4. Le traitement répétitif (suite)

B. La boucle "Pour " (for … in …)

Exemple: Afficher la table de multiplication d'un nombre

Solution : Vu pendant la séance de cours.

4. Le traitement répétitif (suite)

C. Les boucles imbriquées

Les boucles peuvent être imbriquées les unes dans les autres. Une boucle "*tant que …* " peut contenir une autre boucle "*tant que*", ou une autre boucle "*pour … dans*".

Autrement dit, une boucle peut contenir une autre boucle qui elle-même peut contenir une autre boucle ainsi de suite.

Exercice:

Ecrire les programmes Python qui permetent de calculer les opérations suivantes: ($\forall N \in \mathbb{N}$)

1. $S1 = 1 + 2 + 3 + \cdots + N = \sum_{i=1}^{N} i$
2. $S2 = 1 \times 2 \times 3 \times \cdots \times N = \prod_{i=1}^{N} i$
3. $S3 = -\frac{1}{2}N + \frac{1}{2}N + \frac{5}{2}N + \cdots + \frac{N^2 + N - 1}{2}N$
4. $S = \sum_{i=-N}^{N} \frac{i^2 + i - 1}{i + 1}$

Solutions : Vu pendant la séance de cours.