INTRODUCTION AUX STRUCTURES DE CONTROLE

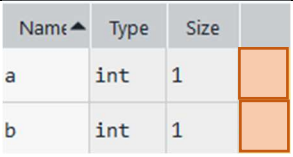
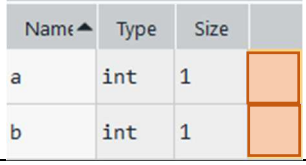
1. Les structures de contrôle

Un programme informatique rationnalise le traitement d'informations dans l'objectif de résoudre un problème donné. Pour cela il lui faut effectuer un certain nombre d'actions et ce, dans un certain ordre. L'algorithme s'attache à décrire de façon structurée les actions à effectuer.
Le programme Python correspondant traduit ces actions structurées en un flux d'exécutions d'instructions qu'il faut contrôler.
Les **structures de contrôle** sont des groupes d'instructions déterminant l'ordre d'exécution. Il en existe de 3 ordres : la séquence d'instructions, la sélection d'instructions et la répétition d'instructions.

2. La séquence d'instructions

Sauf mention explicite, les instructions d'un programme s'exécutent les unes après les autres, dans l'ordre dans lequel elles ont été écrites à l'intérieur du script. Cette suite d'instructions est une séquence d'instructions.

Code	`a, b = 1, 5`
	`a = b`
	`b = a`
		`a, b = 1, 5`
		`b = a`
		`a = b`

Résultat		

3. La sélection ou exécution conditionnelle (`if`, `elif`, `else`)

Une exécution conditionnelle permet d'intervenir dans le choix de l'instruction à venir en fonction d'une condition décrite par une expression booléenne.

3.1. Algorithme

```
si condition
      alors bloc d'instructions 1
      sinon bloc d'instructions 2
fin-du-si
```

signifie que :

- **si** la condition est vérifiée (évaluation de l'expression booléenne à `True`)
  alors le programme exécute le bloc d'instructions 1
- **sinon**, donc si la condition n'est pas vérifiée (évaluation expression booléenne à `False`)
  alors le programme exécute le bloc d'instructions 1.

3.2. Syntaxe Python

```
if condition:
      bloc d'instructions 1
else:
      bloc d'instructions 2
```

.....

PEP8 !

- Deux points après la condition, sans espace.
- Indentation de 4 espaces à chaque nouvelle condition.
- Retour à l'indentation précédente à la fin du bloc d'instructions

➢ Possibilité d'ajout de condition(s) supplémentaire(s): **elif** (« **sinon si** », contraction de else if)

```
if note >= 16:
    mention = "Bravo! Mention Très bien!"
elif note >= 14:
    mention = "Mention Bien"
elif note >= 12:
    mention = "Mention Assez bien"
elif note >= 10:
    mention = "Mention Passable"
else :
    mention = "C'est raté !"
```

- Les expressions booléennes sont évaluées les unes après les autres, de haut en bas, jusqu'à ce que l'une d'entre elles s'évalue en **True**. Le bloc d'instructions correspondant (et seulement celui-ci) est alors exécuté, **puis on sort de la structure conditionnelle.**
- Le bloc correspondant au **else** est exécuté seulement si toutes les expressions conditionnelles situées au-dessus sont évaluées en **False**.

4. La répétition d'instructions en boucle indéfinie ou conditionnelle (**while**)

En programmation, on appelle boucle un système d'instructions qui permet de répéter un certain nombre de fois (voire indéfiniment) toute une série d'opérations.
On peut ainsi être amené à répéter un bloc d'instructions sans savoir combien de fois on devra le répéter (d'où l'appellation de boucle indéfinie !). Dans ce cas, on utilise la boucle « tant que » qui permet de répéter le bloc d'instructions tant qu'une certaine condition est vérifiée.

4.1. Algorithme

```
tant que condition
      faire bloc d'instructions 1
fin-du-tant-que
```

signifie que :

- **si** la condition est vérifiée (évaluation de l'expression booléenne à **True**)
  alors le programme exécute le bloc d'instructions 1
- **sinon**, donc si la condition n'est pas vérifiée (évaluation expression booléenne à **False**)
  alors l'ensemble du bloc d'instructions 1 est ignoré.

4.2. Syntaxe Python

```
while condition:
      bloc d'instructions 1
```

.....

Remarque :

- Si la condition reste toujours vraie, alors le corps de la boucle est répété indéfiniment.
  ➢ Le corps de la boucle doit contenir au moins une instruction qui change la valeur d'une variable intervenant dans la condition évaluée (ou l'instruction **break** vue page suivante).

Boucle infinie	Boucle finie

`condition = 10`
`tant que condition >= 10`
`    faire resultat = "je n'arrête pas"`
`fin-du-tant-que`
	`condition = 1`
	`tant que condition <= 10`
	`    faire condition += 1`
	`    #modification  valeur de la variable à tester`
	`    faire resultat = "Ouf, j'arrête bientôt"`
	`fin-du-tant-que`

- Si la condition est vérifiée avant d'entrer dans la boucle, celle-ci ne sera jamais parcourue !!
  ➢ La condition de boucle doit être correctement initialisée.

5. La répétition d'instructions en boucle définie (for)

On peut être amené à répéter un bloc d'instructions un certain nombre de fois connu au début de la boucle (d'où l'appellation de boucle définie !). Pour cela nous allons utiliser une boucle itérative définie, autrement dit nous allons répéter l'application d'une même séquence d'instructions sur une liste définie à l'avance que nous limiterons pour l'instant à une collection de nombres entiers. Cette boucle est inconditionnelle car elle s'applique sans condition à tous les éléments de la liste, elle-même nommée l'itérable.

5.1. Algorithme

```
pour chaque_element dans liste_iterable répéter
      bloc d'instructions 1
fin-de-la-boucle
```

signifie que :

pour `chaque_element` de la collection d'éléments nommée ici `liste_iterable,` le programme exécute le bloc d'instructions 1.

5.2. Syntaxe Python

Pour générer des collections de nombres entiers on peut utiliser l'instruction `range(n)` :

`range(n)`	Collection de tous les entiers de 0 à n-1 (donc n exclu !)
`range(3, 12)`	Collection de tous les entiers de 3 inclus à 12 exclus
`range(3, 12, 2)`	Collection de tous les entiers de 3 inclus à 12 exclus par pas de 2
`range(12, 3, -1)`	Collection de tous les entiers de 12 inclus à 3 exclus par pas de -1

Exemple:

```
debut = 0
fin = debut
for i in range(4,1,-1):
    fin += 1
```

Tableau de valeurs permettant de lister toutes les valeurs prises par les variables :

Numéro de boucle	début	fin	i

Remarque :

On peut stopper une boucle **for** (ou **while**) avant la fin de l'itération (respectivement avant que la condition ne soit remplie) avec l'instruction **break**.

```
debut_recherche = 2025
fin_recherche = 2030

for a in range(debut_recherche, fin_recherche+1):
      if a % 400 == 0:
            break
      elif a % 100 == 0:
            bissextile = False
      elif a % 4 == 0:
            break
```

Le programme ci-dessus recherche la première année bissextile après 2025. On rappelle qu'une année est bissextile si elle est divisible par 400, ou bien si elle est divisible par 4 sans l'être par 100 !