

## COURS : FLOT D'EXECUTION

## 1 Structures de contrôle

L'instruction `if` permet de soumettre l'exécution d'une instruction ou d'un bloc d'instructions à une condition.

```
1 >>> solde_compte = -100
2     if solde_compte < 0:
3         print("Vous êtes à découvert.")
4         print("Veuillez passer à la banque.")
5     print("Bonne journée.")
6
7     Vous êtes à découvert.
8     Veuillez passer à la banque.
9     Bonne journée.
```

Le bloc d'instructions soumis à condition est délimité par l'*indentation*. Par rapport à l'instruction `if`, on décale d'un même nombre d'espaces chaque instruction faisant partie du bloc d'instructions. Par convention, nous choisissons une indentation de 4 espaces. L'instruction suivant la fin du bloc doit avoir le même niveau d'indentation que l'instruction `if`. Le programme précédent vous souhaitera donc une bonne journée quelque soit l'état de votre compte.

Il est possible d'exécuter un bloc d'instructions dans le cas où la condition n'est pas vérifiée.

```
1 >>> est_homme = False
2     if est_homme:
3         salutation = "Monsieur"
4     else:
5         salutation = "Madame"
6     print("Bonjour " + salutation + ".")
7
8     Bonjour Madame.
```

Enfin, il est possible d'exécuter différents blocs selon plusieurs conditions.

```
1 >>> note = 13
2     if note >= 16:
3         print("Mention Très Bien.")
4     elif note >= 14:
5         print("Mention Assez Bien.")
6     elif note >= 12:
7         print("Mention Bien.")
8     elif note >= 10:
```

```
9         print("Vous avez votre Bac.")
10    else:
11        print("Same player shoot again!")
12
13    Mention Assez Bien
```

Seul le bloc correspondant à la première condition vérifiée est exécuté.

## 2 Boucles inconditionnelles

Il est possible de répéter plusieurs fois la même séquence d'instructions en utilisant une boucle `for`. Pour délimiter le groupe d'instructions qui seront répétées, on indente ces instructions d'un même nombre d'espaces. Il est courant d'utiliser 2 espaces. La première instruction ne faisant pas partie de la boucle doit utiliser le même niveau d'indentation que la ligne `for`. Dans le film « The Shining » de Stanley Kubrick, adapté du roman de Stephen King, le personnage principal succombe petit à petit à la folie :

```
1 >>> for k in range(2):
2     print("All work and no play")
3     print("makes Jack a dull boy.")
4     print("Jack Torrance")
5
6     All work and no play
7     makes Jack a dull boy.
8     All work and no play
9     makes Jack a dull boy.
10    Jack Torrance
```

Pour calculer le  $n$ -ième terme de la suite définie par

$$u_0 = 1 \quad \text{et} \quad \forall n \in \mathbb{N} \quad u_{n+1} = \cos(u_n)$$

on utilisera le programme suivant.

```
1 >>> import numpy as np
2 >>> u = 1.0
3     n = 1000
4     for k in range(n):
5         u = np.cos(u)
6     print("u_" + str(n) + " =", u)
```

Il est souvent utile d'avoir un index prenant les valeurs entières successives lors de la boucle. Ainsi, pour afficher les premiers termes de la suite définie par

$$\forall n \in \mathbb{N} \quad u_n = \sum_{k=0}^n \frac{1}{k!}$$

on utilise le programme suivant.

```
1  >>> n = 10
2      u = 0.0
3      for k in range(n):
4          u = u + 1 / math.factorial(k)
5      print("u_" + str(k + 1) + " =", u)
```

La variable  $k$  va prendre successivement les valeurs  $0, 1, 2, \dots, n-1$ . On peut faire commencer la variable  $k$  à une valeur autre que 0 en utilisant `range(a, b)`. Dans ce cas, la variable  $k$  va successivement prendre les valeurs  $a, a+1, \dots, b-1$ . Ainsi, pour calculer le  $n$ -ième terme de la suite

$$\forall n \in \mathbb{N} \quad u_n = \sum_{k=1}^n \frac{1}{k^2}$$

on utilisera le programme

```
1  >>> somme = 0.0
2      n = 1000
3      for k in range(1, n + 1):
4          somme = somme + 1 / k**2
5  >>> somme
```

Plus généralement, si  $\delta > 0$ , l'instruction `range(a, b, delta)` est utilisée pour boucler sur les entiers  $a, a+\delta, a+2\delta$  jusqu'au plus grand entier de la forme  $a+k\delta$  strictement inférieur à  $b$ . Lorsque  $\delta < 0$ , cette même instruction permet d'itérer sur les entiers  $a, a+\delta, a+2\delta$  jusqu'au plus petit entier de la forme  $a+k\delta$  strictement supérieur à  $b$ . En particulier, le programme suivant permet de calculer la même somme en sommant dans l'ordre inverse.

```
1  >>> somme = 0.0
2      n = 1000
3      for k in range(n, 0, -1):
4          somme = somme + 1 / k**2
5  >>> somme
```

Remarquons que le résultat est légèrement différent du premier. Ce phénomène est dû à la non associativité de l'addition sur les nombres flottants, conséquence des arrondis effectués à chaque opération. Vous lisez bien, j'ai bien parlé de la non associativité de l'addition sur les flottants et non de sa non commutativité. Et pour cause, l'addition sur les nombres flottants est bien commutative.

Il est possible d'imbriquer les boucles les unes dans les autres. Vous pouvez par exemple générer les tables de multiplication très facilement de la manière suivante.

```
1  >>> for a in range(1, 10):
2      for b in range(1, 10):
3          print(a, "*", b, "=", a * b)
```

## 3 Boucles conditionnelles

Les boucles `for` nous ont permis d'exécuter plusieurs fois un bloc d'instructions lorsque le nombre d'itérations est connu avant de rentrer dans la boucle. Lorsque ce nombre n'est pas connu avant de rentrer dans la boucle, typiquement lorsque l'on doit exécuter un bloc tant qu'une condition est vérifiée, on utilise l'instruction `while`. Le programme suivant calcule le nombre de chiffres utilisés pour écrire l'entier  $u$  en base 10.

```
1  >>> u = 123
2      m = 0
3      v = 1
4      while v <= u:
5          v = v * 10
6          m = m + 1
7  >>> m
```

La condition «  $v \leq u$  » est évaluée en début de boucle. Si elle est vérifiée, le bloc d'instruction est exécuté. Puis la condition est à nouveau évaluée. La boucle s'arrête lorsque la condition est fausse. Pour la valeur initiale de 123, la variable  $v$  prend successivement les valeurs 1, 10, 100 et 1000. On passe donc 3 fois dans la boucle ce qui nous assure que le nombre s'écrit bien avec 3 chiffres.

Contrairement à ce qui se passe pour une boucle conditionnelle pour laquelle on est assuré de sortir de la boucle, il est tout à fait possible qu'une boucle inconditionnelle ne sorte jamais de la boucle. On dit alors que le programme part en boucle infinie. Dans le film, « Groundhog day », le personnage joué par Bill Muray se réveille chaque matin au son de la radio qui lui annonce le jour de la marmotte.

```
1  >>> while True:
2      print("This is Groundhog day!")
```

Vous pourrez interrompre le programme en appuyant à la fois sur la touche « control » et la touche « c ».

Le calcul du pgcd par l'algorithme d'Euclide se fait en remplaçant  $a$  et  $b$  respectivement par  $b$  et  $a \% b$ . Lorsque  $b$  est nul, on obtient le pgcd dans la variable  $a$ .

```
1  >>> a = 42
2      b = 18
3      while b != 0:
4          c = a % b
5          a = b
6          b = c
7  >>> a
```

La suite de Syracuse est définie en initialisant  $u_0$  avec un entier  $m$  et en définissant :

$$\forall n \in \mathbb{N} \quad u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

L'expérience montre que, quel que soit l'entier  $m$  choisit, la suite fini par prendre les valeurs 1, 4, 2, 1, 4, 2, 1, etc. Afin de vérifier ce phénomène, on écrit un programme permettant d'afficher toutes les valeurs de la suite, jusqu'à ce que  $u_n = 1$ .

```
1  >>> u = 17
2      n = 0
3      while u != 1:
4          if u % 2 == 0:
5              u = u // 2
6          else:
```

```
7              u = 3 * u + 1
8          n = n + 1
9          print("u_" + str(n) + " =", u)
```

À ce jour, personne n'a pu montrer que la suite de Syracuse finissait par prendre la valeur 1 quelle que soit la première valeur choisie. On a cependant vérifié que c'était bien le cas pour de nombreux entiers. Ce programme est un exemple très simple pour lequel aucune preuve de terminaison n'existe à ce jour.

Voici enfin un exercice pour vérifier que ce chapitre est bien compris.