

INSTRUCTIONS ITÉRATIVES

Les structures de contrôle itératives ou boucles permettent d'exécuter plusieurs fois de suite une ou plusieurs instructions. Nous en présentons deux types dans cette partie : `for` et `while`.

I. La boucle `for ... in range (...)`

Les boucles inconditionnelles servent à exécuter une instruction un nombre connu de fois.

Pour cela on peut utiliser le mot clé `for ... in range(...):` qui peut être traduit par « pour ... dans la gamme... ».

Par exemple, monsieur X est puni et doit recopier 10 lignes. Il doit écrire : « Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement. ». Il est alors possible d'écrire le code suivant :

```
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
print("Je ne dois pas frapper mon ordinateur quand il ne fonctionne pas correctement.")
```

Pour optimiser le code, on utilise la boucle suivante :

```
for i in range(10):
    print("Je ne dois pas frapper (...) quand il ne fonctionne pas correctement.")
```

Le mot clé `range(a,b,n)` s'utilise de la façon suivante :

- `a` est la valeur initiale que va prendre le compteur `i`.
Par défaut, si elle n'apparaît pas, cette valeur est 0.
- `b-1` est la limite où la condition s'arrête.
Ici, nous avons mis 11 car nous voulons 10 lignes ($11 - 1 = 10$).
- `n` est le pas avec lequel `i` augmente (ou diminue).
Par défaut si, elle n'apparaît pas, cette valeur est +1.

```
for i in range(1,11):
    print("ligne",i," : Je ne dois pas frapper (...) fonctionne pas correctement.")
```

Exemple

- `range(12)` retourne les entiers de 0 à 11.
- `range(2, 12, 2)` retourne les entiers de 2, 4, 6, 8 et 10.
- `range(9, 2, -1)` retourne les entiers de 9 à 3 par ordre décroissant.

Remarque

- Utiliser l'instruction `for` lorsqu'on sait à l'avance le nombre d'itérations que l'on souhaite effectuer.
- Utiliser l'instruction `for` pour programmer des calculs de somme ou de produit.

L'idée est d'ajouter (ou multiplier) un à un les termes de la somme dans chaque tour de la boucle.

Exercice 1

Dans un fichier intitulé `factorielle.py`, écrire un script où vous stockez un entier n et qui affiche la valeur de $n!$.

Rappel : pour tout $n \in \mathbb{N}$, $n! = 1 \times 2 \times 3 \times \dots \times (n-1) \times n$.

Pour vérification : $10! = 3\,628\,800$.

Exercice 2

Dans un fichier intitulé `suite_recurrente.py`, écrire un script qui demande d'entrer un entier naturel non nul n et qui affiche en sortie directement la valeur de u_n où $(u_n)_{n \in \mathbb{N}}$ est la suite définie par :

$$u_{n+1} = \frac{1}{2}u_n - 3 \quad \text{et} \quad u_0 = 10$$

Conjecturer alors la limite de la suite $(u_n)_{n \in \mathbb{N}}$. (On ne demande pas de démonstration)

Pour vérification : $u_{10} = -5.984\,375$.

II. La boucle `for ... in ... :`

Pour parcourir du texte (chaîne de caractères), on peut utiliser le mot clé `for ... in ... :`.

Cette boucle parcourt la chaîne de caractères de la première « lettre » à la dernière.

```
lettre_presente = False
prenom = input("Entrer votre prénom.")

for i in prenom:
    if i == "a":
        lettre_presente = True

if lettre_presente:
    print("Votre prénom contient la lettre a.")
else:
    print("Votre prénom ne contient pas la lettre a.")
```

Remarque

Notons l'utilisation du mot clé `input` sans `eval` puisque ici nous travaillons avec des variables de type `str`.

Exercice 3

Dans un fichier intitulé puissance, écrire un script qui demande d'entrer un entier naturel n et un réel x et qui affiche en sortie la valeur de x^n sans utiliser la fonction puissance de Python `x**n` (donc à l'aide de multiplications successives).

Pour vérification : $2^{10} = 1024$ et attention à traiter à part le cas $n = 0$.

Exercice 4

Dans un fichier intitulé somme_cube, écrire un script dans lequel vous stockez un entier non nul n et qui affichera en sortie les valeurs de :

$$S_n = \sum_{k=1}^n k^3 = 1^3 + 2^3 + \dots + n^3 \quad \text{et} \quad T_n = \left(\frac{n(n+1)}{2} \right)^2$$

Pour vérification : $S_{100} = 25\,502\,500$.

Modifier votre script afin qu'il compare (on fera un test Python) les valeurs de S_n et T_n pour toutes les valeurs de n entre 1 et 100. Qu'observe-t-on ?

Exercice 5

1) Écrire un programme Python utilisant une boucle `for` qui, pour chaque seconde de $t = 0$ à $t = 3$ s, calcule la distance parcourue, la vitesse et affiche les valeurs sous la forme :

$t = 0$ s : distance = 0.00 m, vitesse = 0.00 m/s.

$t = 1$ s : distance = 4.91 m, vitesse = 9.81 m/s

...

Données : $g = 9,81 \text{ m.s}^{-2}$, $d(t) = \frac{1}{2}gt^2$ et $v(t) = gt$.

2) À partir de quelle valeur de t l'objet a-t-il parcouru plus de 300 m ?

III. La boucle `while`

Les boucles conditionnelles peuvent servir à exécuter une instruction un nombre de fois dépendant de la réalisation d'une certaine condition.

Pour cela on peut utiliser le mot clé `while ... :` qui peut être traduit par « tant que ... ».

Par exemple, monsieur Y est puni et doit recopier 10 fois une ligne. Il doit écrire : « Je dois lever la main avec de prendre la parole. ». Il est alors possible d'écrire le code suivant :

```
i = 1
while i <= 10:
    print("ligne",i,": Je dois porter correctement mon masque au lycée.")
    i += 1
```

Remarque

Notons l'utilisation d'une variable aléatoire « muette » i qui est ce que l'on appelle un compteur. Elle représente en général le numéro de la boucle que l'on est en train de parcourir et intervient parfois dans le test d'arrêt. Il est nécessaire de l'incrémenter (`i += 1`) à chaque boucle afin de pouvoir compter le nombre de boucles.

IV. Les booléens dans une boucle

De la même façon que pour les conditions, il est possible d'utiliser les booléens dans les boucles.

Par exemple, on peut utiliser le code suivant :

```
entier = False

while not entier:
    n = eval(input("Entrer un entier : "))
    if type(n) == int:
        entier = True
    else:
        print(n, "n'est pas un entier")
```

Expliquer ce que fait le programme.

Exercice 6

Dans un fichier intitulé `suite_recurrente_2`, écrire un script qui affichera en sortie la plus petite valeur de n telle que $|u_n - (-6)| \leq 10^{-3}$ où $(u_n)_{n \in \mathbb{N}}$ est la suite définie par :

$$u_{n+1} = \frac{1}{2}u_n - 3 \quad \text{et} \quad u_0 = 10$$

(La suite $(u_n)_{n \in \mathbb{N}}$ est proche de sa limite conjecturée au millièmè.)

Pour vérification : $n = 14$.

Exercice 7

Dans un fichier intitulé `somme_double.py`, écrire un script qui affiche les valeurs des sommes suivantes :

$$A = \sum_{i=0}^{100} i^2, \quad B = \sum_{i=0}^{100} \sum_{j=0}^{100} ij, \quad C = \sum_{i=0}^{100} \sum_{j=i}^{100} ij, \quad D = \sum_{i=0}^{100} \sum_{j=0}^i ij$$

Pour vérification : $A = 338\,350$, $B = 25\,502\,500$, $C = 12\,920\,425$, $D = 12\,920\,425$.