Enset-Mohammedia Semestre N°1 2025-2026

Fonctions et boucles.

Support & TD

Il arrive qu'on veuille, en programmation, répéter une opération un certain nombre de fois. On pourrait, pour cela, recopier l'instruction correspondante autant de fois que nécessaire dans le code source, mais ce serait très long. De plus ce n'est pas toujours possible, car il se peut que le nombre de répétitions à faire ne soit pas connu à l'avance par le programmeur (par exemple, le nombre de répétitions à effectuer pourrait être saisi par l'utilisateur pendant l'exécution du programme, donc une fois le code terminé et compilé).

On utilise alors les boucles, qui sont des instructions indiquant en une seule ligne qu'une opération (ou un bloc d'opérations) sera à répéter.

1 Boucle `for`

Les boucles `for` servent à répéter un bloc d'opérations **un nombre déterminé de fois**.

La forme générale d'une boucle *for* est la suivante, n étant le nombre de fois qu'on souhaite répéter les instructions :

Pour k allant de 1 à n, **faire**	**for** `k in range(1,n+1):`
instructions (I)	instructions (I)
fin pour	Suite du programme
Suite du programme	

Lors de l'exécution de ces lignes de code, l'ordinateur affecte à k la valeur 1 puis exécute le jeu d'instuctions (I), puis affecte à k la valeur 2 et exécute à nouveau les instructions (I), *etc.* Il poursuit ainsi jusqu'à affecter à k la valeur n et exécuter une dernière fois les instructions (I).

- Comme pour la structure conditionnelle si…alors…sinon…fin–si, la fin de la boucle *for* se traduit en Python par un retour à l'indentation initiale.
- **Attention** à ne pas oublier les « deux points ».
- Pour des entiers a et b tels que $a < b$, l'option `range(a,b)` demande à Python de parcourir les entiers compris entre a et $b-1$. Ainsi, `k in range(1,n+1)` fait parcourir à k les entiers entre 1 et n. Lorsqu'on ne précise pas la valeur de a, c'est sous-entendu qu'elle vaut 0 ; ainsi `range(10)` signifie `range(0,10)`.
- La variable k est appelée un **compteur de boucle**. C'est une variable **muette** qui n'a de valeur que le temps de l'exécution de la boucle. On pourrait lui donner un autre nom.

Exercice 1. Répondre sans utiliser l'ordinateur, puis vérifier vos réponses.

1. On exécute les programmes suivants. Pour chacun d'entre eux, prévoir le résultat obtenu à l'écran.

Programme 1

```
for k in range(0, 5):
    print(k)
```

Programme 2

```
for k in range(5):
    print(k)
```

Programme 3

```
for k in range(1, 6):
    print(k)
```

2. On considère le programme suivant :

```
1  a = 1
2  for k in range(1,5):
3      a = 2*a
```

Après exécution de ce programme, quelles seront les valeurs de k et de a ? Vérifier dans la console.

Exercice 2. Dans la partie « code source », recopier le script suivant. Répondre aux questions d'abord sans exécuter le programme, puis vérifier.

```
1 n = input('Entrer un entier naturel: ')
2 n = eval(n)
3 u = 0
4 for k in range(0,n+1):
5     u = u + k**3
6     print(u)
7 print('Le resultat est:', u)
```

1. Que calcule ce programme ?
2. Quelle instruction doit on supprimer pour ne faire apparaître que le résultat et non les calculs intermédiaires ?
3. Sans effectuer la modification sur l'ordinateur, préciser le résultat obtenu en remplaçant la ligne 5 par `u = u + n**3`.

Exercice 3. Écrire un script qui demande à l'utilisateur d'entrer la valeur d'un entier naturel n et d'un réel x et affiche la valeur de la somme :

$$0 + x + 2x^2 + 3x^3 + ... + nx^n.$$

Exercice 4. Écrire un script qui calcule la moyenne de notes (entre 0 et 20) entrées par l'utilisateur. Le programme commencera par demander à l'utilisateur d'entrer le nombre n de notes, demandera n fois à l'utilisateur d'entrer une note puis retournera leur moyenne.

Exercice 5. Écrire une fonction d'en-tête `factorielle(n)` qui prend en entrée un entier naturel n et renvoie le nombre $n!$. On rappelle que $0! = 1$ et, lorsque $n \in \mathbb{N}^*$, $n!$ désigne le produit de tous les entiers compris entre 1 et n.

2 Boucle `while`

On utilise une boucle `while` lorsqu'on répète un bloc d'opérations un nombre de fois non déterminé à l'avance, mais plutôt **tant qu'une condition est vraie.**

Exemple : Si l'on programme un jeu cherchant à faire deviner un nombre entre 0 et 100 à l'utilisateur, on souhaite inviter celui-ci à entrer des propositions jusqu'à avoir trouvé la bonne réponse. Il faut donc lui demander un certain nombre de fois de taper un nombre au clavier, mais **on ne peut pas savoir à l'avance en combien de coups il va gagner.** On ne peut donc pas utiliser de boucle *for*. On utilisera plutôt une boucle *while* : on répétera l'opération **tant que** l'utilisateur n'a pas trouvé la bonne réponse.
La forme générale d'une boucle *while* est la suivante :

Tant que *condition*, **faire**	**while** *condition*:
instructions (I)	instructions (I)
fin tant que	Suite du programme
Suite du programme	

Le programme teste si le booléen *condition* (qui dépend de certaines variables V) est vrai :

- Si oui, il exécute les instructions (I), puis il teste à nouveau si la condition est vraie :
  - Si oui, il exécute les instructions (I), puis il teste à nouveau si la condition est vraie :

· Si oui, il exécute les instructions (I), puis il teste à nouveau si la condition est vraie :

· Et ainsi de suite, tant que la condition est vraie.

Dès que la condition est devenue fausse (on peut supposer que les instructions (I) agissent sur les variables V), on passe à la suite du programme.

— **Attention** à ne pas oublier les « deux points ».

— Comme pour la structure conditionnelle `if`, la boucle `for` et la définition d'une fonction par `def`, c'est le retour à l'indentation initiale qui permet de délimiter la fin d'une boucle `while`.

Exemple : La fonction suivante prend en argument un entier N et renvoie le plus petit entier n tel que $2^n > N$.

```
1 def essai(N):
2     n = 0
3     while 2**n<=N:
4         n = n+1
5     return n
```

Exercice 6. Répondre sans utiliser l'ordinateur, puis vérifier vos réponses.

1. On exécute le programme suivant. Prévoir le résultat obtenu à l'écran.

```
1     valeur = 5
2     while valeur >= 0:
3         valeur = valeur - 1
4     print(valeur)
```

2. On exécute les quatre programmes suivants. Prévoir le résultat obtenu à l'écran.

Programme 1

```
a = 0
while a <= 10:
    a = a + 2
print(a)
```

Programme 2

```
a = 10
while a >= 0:
    a = a + 2
print(a)
```

Programme 3

```
a = 0
while a <= 10:
    print(a)
    a = a + 2
```

Programme 4

```
a = 0
while a <= 10:
    a = a + 2
    print(a)
```

Exercice 7. Soient n un entier naturel et a un nombre réel. Que calcule la fonction suivante ? En comprendre le principe.

```
1 def prog1(a,n):
2     s = 1
3     k = 1
4     while k <= n:
5         k = k+1
6         s = s*a
7     return s
```

Exercice 8.

1. Écrire à l'aide d'une boucle `while` une fonction qui prend en entrée un entier naturel non nul et renvoie la valeur de la somme $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$.
2. Écrire à l'aide d'une boucle `while` un script qui calcule et affiche le plus petit entier naturel n tel que cette somme soit supérieure ou égale à 8.

3 Exercices d'application

Exercice 9. *Devinez le nombre !*

1. Écrire un script effectuant les tâches suivantes :
   - l'ordinateur tire aléatoirement [1] un entier compris entre 1 et 1000 (qu'on propose de faire deviner à l'utilisateur) ;
   - l'ordinateur indique à chaque tentative inexacte de l'utilisateur : « Le nombre recherché est plus grand » ou « Le nombre recherché est plus petit » ;
   - lorsque l'utilisateur parvient à la bonne réponse, l'ordinateur affiche : « Bravo ! Vous avez trouvé la bonne réponse : », suivi du nombre cherché.

   La copie d'écran proposée montre ce qui peut se produire après l'exécution de ce script.

```
Entrer un nombre entre 1 et 1000:500

 Le nombre est plus grand
Nouvel essai:800

 Le nombre est plus petit
Nouvel essai:775

 Le nombre est plus petit
Nouvel essai:755

 Le nombre est plus grand
Nouvel essai:758

 Le nombre est plus petit
Nouvel essai:757

 Bravo
```

2. Améliorer le script précédent de façon à ce que :
   - le nombre maximal d'essais accordés au joueur soit huit ;
   - si le joueur n'a pas trouvé le nombre caché au bout de huit essais, l'ordinateur l'avertit qu'il a perdu et affiche le nombre mystère ;
   - si le joueur gagne, l'ordinateur l'avertit et affiche le nombre d'essais effectués.

   Rappelons que la condition de la boucle `while` est une expression booléenne pouvant contenir les opérateurs logiques `or`, `and` et `not`.

Exercice 10. *Valeur approchée de* e

Pour tout $n \in \mathbb{N}$, posons $u_n = \sum_{k=0}^{n} \frac{1}{k!} = 1 + \frac{1}{1!} + \frac{1}{2!} + ... + \frac{1}{n!}$. On peut démontrer que la suite (u_n) converge vers le nombre e et :

$$\forall n \in \mathbb{N}, \quad |u_n - \mathrm{e}| \leqslant \frac{3}{n!}.$$

A l'aide d'une boucle `while`, écrire une fonction d'en-tête `approx(epsi)` qui prend en entrée un réel strictement positif `epsi` et qui renvoie une valeur approchée du nombre e avec une erreur d'approximation inférieure à `epsi`.

Exercice 11. Écrire une fonction qui prend en entrée un entier naturel n et renvoie le terme u_n de la suite définie par :

$$\begin{cases} u_0 = 1 \\ \forall k \in \mathbb{N}, \quad u_{k+1} = \frac{u_k}{u_k + 1} \end{cases}$$

Quelle conjecture peut-on formuler sur l'expression de u_n en fonction de n ?

1. Voir annexe en fin de TP pour le tirage aléatoire.

Exercice 12.

1. Écrire une fonction qui prend en entrée un entier naturel n et renvoie le terme u_n de la suite (dite de Fibonacci) définie par :
$$\begin{cases} u_0 = u_1 = 1 \\ \forall k \in \mathbb{N}, \quad u_{k+2} = u_{k+1} + u_k \end{cases}$$
2. Écrire une fonction qui prend en entrée un entier naturel non nul n et renvoie la valeur de $v_n = u_{n+1}u_{n-1} - u_n^2$. Calculer la valeur de v_n pour quelques valeurs de n. Quelle conjecture peut on formuler ?

4 Annexe : importation de modules

Le langage Python offre de très nombreuses commandes prédéfinies rangées dans des **modules** ou **bibliothèques.** L'accès à un module se fait en utilisant l'une des commandes suivantes :

— **from** *nomdumodule* **import ***
Cette instruction permet d'importer l'ensemble des commandes du module *nomdumodule.*

— **from** *nomdumodule* **import** *nomcommande*
Cette instruction permet d'importer du module *nomdumodule* la commande *nomcommande.*

Exemple :
Le module **math** contient de nombreuses fonctions et constantes mathématiques usuelles.

Commandes Python	Objets mathématiques
`pi`, `e`	π, $\mathrm{e} = \exp(1)$
`exp(x)`, `log(x)`, `sqrt(x)`	$\exp(x)$, $\ln(x)$, $\sqrt{x}$
`pow(x,y)`, `floor(x)`, `abs(x)`, `factorial(n)`	x^y, $\lfloor x \rfloor$, $\lvert x \rvert$, $n!$
`sin(x)`, `cos(x)`, `tan(x)`, `asin(x)`, ...	$\sin(x)$, $\cos(x)$, $\tan(x)$, $\arcsin(x)$, ...

Pour calculer une approximation de $\sqrt{\pi} - \ln(2)$, on peut écrire :

```
1  from math import *
2  sqrt(pi) - log(2)
```

ou bien

```
1  from math import sqrt, pi, log
2  sqrt(pi)-log(2)
```

Pour effectuer des tirages de nombres aléatoires, on importe le module `random`. On donne les deux commandes suivantes (utiles ici seulement dans l'exercice 9, mais nous y reviendrons) :

`randint(a,b)`	choisit aléatoirement et de manière uniforme un entier compris entre a et b.
`random()`	choisit aléatoirement et de manière uniforme un réel compris entre 0 et 1.