



LES CONSTRUCTIONS ÉLÉMENTAIRES : CONDITIONS, BOUCLES ET FONCTIONS

0. Indentation

En Python, les instructions appartenant à un même bloc doivent être indentées (décalées vers la droite). L'indentation est obligatoire : elle indique quelles instructions sont exécutées dans le if, le for, le while ou la fonction.

1. Les conditions : faire un choix

On utilise **if**, **elif** et **else** pour faire un choix.



- **if** : exécute un bloc d'instructions **si une condition est vraie**.
- **elif** (*sinon si*) : permet de tester **une autre condition** si la précédente est fausse.
- **else** (*sinon*) : exécute un bloc d'instructions **si aucune des conditions précédentes n'est vraie**.

À toi de jouer : Qu'affiche ce programme ?

```
1 moyenne=14
2 ▾ if moyenne >= 16:
3     print("Mention Très Bien")
4 ▾ elif moyenne >= 14:
5     print("Mention Bien")
6 ▾ elif moyenne >= 12:
7     print("Mention Assez Bien")
8 ▾ elif moyenne >= 10:
9     print("Admis")
10 ▾ else:
11     print ("Ajourné")
```



...

2. Les boucles for : répéter des instructions

Une boucle permet de répéter automatiquement une ou plusieurs instructions. Elle évite de recopier plusieurs fois le même code et rend les programmes plus courts, plus lisibles et plus faciles à modifier.

Par exemple, un bus doit annoncer chaque arrêt.

Au lieu d'écrire plusieurs fois la même instruction :

```
print("Arrêt numéro 1")
print("Arrêt numéro 2")
print("Arrêt numéro 3")
print("Arrêt numéro 4")
print("Arrêt numéro 5")
```

Avec une boucle, on écrit :

```
for arret in range(1, 6):
    print("Arrêt numéro", arret)
```

Idée : fusée et tables

À toi de jouer : Complète le tableau d'après le programme suivant :

La variable `i` prend successivement les valeurs produites par `range()`.

Passage	i	Affichage
1	0	
2	1	
3	2	
4	3	
5	4	

```
for i in range(5) :
    print(i**2)
```



Écriture	Utilisation	Affichage
<code>for i in range(5): print(i)</code>	Répéter 5 fois	0 1 2 3 4
<code>for i in range(1,6): print(i)</code>	Commencer à une valeur	1 2 3 4 5
<code>for i in range(2,11,2): print(i)</code>	Avancer avec un pas	2 4 6 8 10
<code>for i in range(3,0,-1): print(i)</code>	Parcourir les valeurs dans l'ordre décroissant	3 2 1
<code>for _ in range(5) : print("Bonjour")</code>	Faire 5 fois sans avoir besoin d'indice de boucle	Bonjour Bonjour Bonjour Bonjour Bonjour

À retenir :



range(debut, fin, pas) produit une suite de nombres allant de debut jusqu'à **fin exclue**, en avançant du pas indiqué.

Une fusée doit effectuer un compte à rebours avant le décollage.

Voici le programme :

```

1 for secondes in range(5, 0, -1):
2     print(secondes)
3
4 print("🚀 Décollage !")

```



À toi de jouer : Complète

Passage dans la boucle	Valeur de seconde	Affichage
1		
2		
3		
4		
5		

Après la boucle :

Valeur de seconde	Affichage

À retenir :



En Python, la variable de boucle conserve sa dernière valeur après la fin de la boucle. Toutefois, il est préférable de ne pas la réutiliser ensuite : cette variable est destinée à contrôler la boucle et non à être utilisée dans la suite du programme.

3. L'accumulateur : mémoriser un résultat

Un **accumulateur** conserve un résultat calculé au fil des tours de boucle.

Par exemple, un robinet verse **2 litres** d'eau chaque minute. Après 5 minutes, combien de litres y a-t-il dans le réservoir ?

Avant la boucle, on **initialise** l'accumulateur à 0.

```
eau = 0
for _ in range(5):
    eau = eau + 2
print(eau)
```



Passage dans la boucle	Valeur de tour	Valeur de eau
1	0	2
2	1	4
3	2	6
4	3	8
5	4	10

4. Les boucles while : répéter tant qu'une condition est vraie

Une boucle while est très utile lorsque l'on souhaite **répéter une action jusqu'à ce qu'une condition soit satisfaite**.

Par exemple, on peut demander à un utilisateur de saisir un mot de passe tant qu'il n'est pas correct.

```
mot = ""
while mot != "NSI":
    mot = input("Entrez le mot de passe : ")
print("Mot de passe correct !")
```

5. Les fonctions : réutiliser du code

Depuis le début de ce chapitre, nous avons utilisé plusieurs fonctions de Python, comme `print()`, `input()` ou `range()`. Nous allons maintenant apprendre à créer nos propres fonctions.

a. Qu'est-ce qu'une fonction ?

Une **fonction** est un bloc d'instructions qui réalise une tâche précise. Elle peut être fournie par Python ou créée par le programmeur.

Une fois définie, une fonction peut être **appelée** autant de fois que nécessaire dans un programme.

Les fonctions permettent :

- D'éviter de répéter plusieurs fois le même code ;



- De rendre un programme plus lisible ;
- De faciliter les modifications et la correction des erreurs.

b. Définir une fonction

Pour créer une fonction, on utilise le mot-clé `def`.

```
def bonjour():  
    print("Bonjour !")
```

Cette instruction **définit** la fonction, mais ne l'exécute pas.

Pour l'utiliser, il faut l'**appeler** :

```
bonjour()
```

Résultat :

Bonjour !

c. Les paramètres

Une fonction peut recevoir une ou plusieurs **valeurs**, transmises grâce à des **paramètres**.

```
def bonjour(prenom):  
    print("Bonjour", prenom)
```

`prenom` est un **paramètre** ; `"Alice"` est un **argument**.

On peut ensuite appeler cette fonction avec différentes valeurs :

```
bonjour("Alice")
```

Résultat :

Bonjour Alice

Le même code est ainsi réutilisé avec des données différentes.

d. Renvoyer une valeur



Une fonction peut également **renvoyer un résultat** grâce au mot-clé `return`.

```
def carre(nombre):  
    return nombre ** 2
```

On peut récupérer cette valeur dans une variable :

```
x = carre(5)  
print(x)
```

Résultat :

25

e. À retenir :

- `print()` affiche un résultat à l'écran.
- `return` renvoie un résultat qui pourra être réutilisé dans le programme.

7. Bilan



À la fin de ce chapitre, tu sais :

- Utiliser une condition (`if`, `elif`, `else`) pour faire un choix ;
- Répéter des instructions avec une boucle `for` et `range()` ;
- Utiliser une boucle `while` pour répéter des instructions tant qu'une condition est vraie ;
- Utiliser un accumulateur pour calculer une somme ;
- Créer et appeler une fonction simple.

```
1 # Condition
2 if condition:
3     ...
4
5 # Boucle for
6 for i in range(...):
7     ...
8
9 # Boucle while
10 while condition:
11     ...
12
13 # Fonction
14 def nom(...):
15     ...
16
17 # Retour d'une fonction
18 return valeur
```