Chapitre 5

Les boucles

Les boucles sont un concept fondamental en programmation.
Elles permettent de répéter une certaine opération autant de fois que nécessaire.

Comment faire par exemple pour :

- Demander un mot de passe tant que celui rentré n'est pas correct ?
- Simuler 1000 lancers de dés ?
- Compter le nombre de voyelles dans un mot ?

Avec des boucles bien sûr !

5.1 La boucle conditionnelle `while`

5.1.1 Un problème de mot de passe...

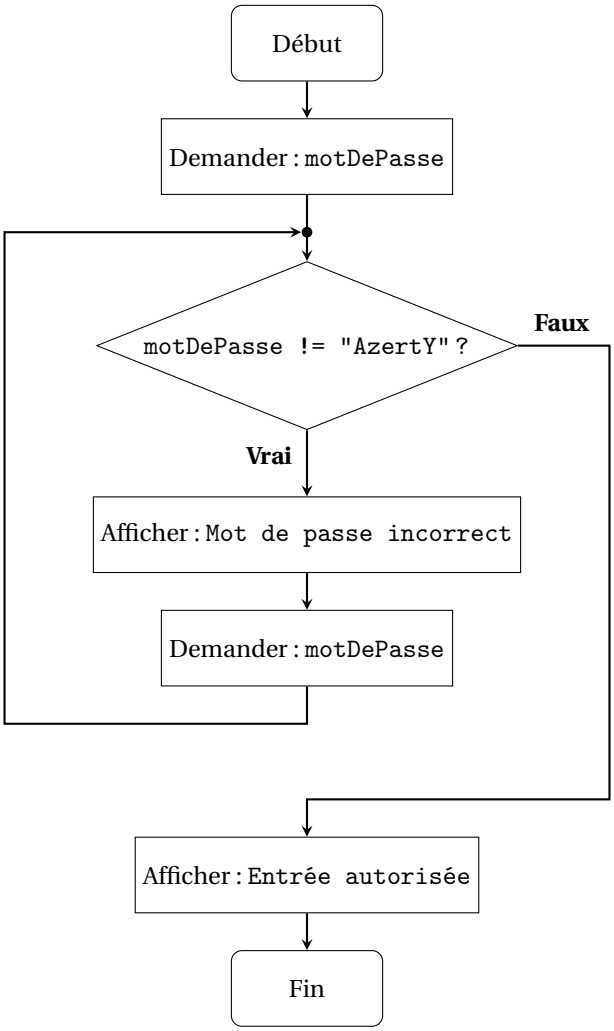
Pour introduire la notion de **boucle conditionnelle**, considérons le problème suivant :

Problème. *Écrire un programme qui demande à l'utilisateur un mot de passe. Tant que ce mot de passe est différent de **AzertY**, afficher une erreur et demander à nouveau le mot de passe. Si le mot de passe est correct, afficher `Entrée autorisée`.*

Un mot a son importance : **tant que**. Le programme doit afficher une erreur et demander le mot de passe **tant que** ce dernier est différent de celui attendu, ici *AzertY*.

▷ ***Combien de fois doit-on afficher l'erreur ? Et demander le mot de passe ?***

On ne sait pas ! Cela dépend des mots de passe donnés par l'utilisateur.
Il faut demander le mot de passe en boucle, tant que celui-ci n'est pas égal au mot de passe correct.
On peut schématiser notre programme de la façon suivante :



On retrouve un schéma très proche de ceux vus dans le chapitre sur les conditions, à une différence près.
Ici, on boucle tant que la condition `motDePasse != "AzertY"` est vraie.

Lorsque cette dernière devient fausse, on arrive à la **condition d'arrêt** de la boucle (ici `motDePasse == "AzertY"`).
On « sort » alors de la boucle, et le programme reprend son déroulement normal.

▷ ***Et comment fait-on tout ça avec Python ?***

Pour déclarer une boucle conditionnelle en Python, on utilise la syntaxe suivante :

```
while condition:
  # Instructions à répéter

# Instructions après la boucle
```

Une fois encore, c'est l'**indentation** qui permet à Python de savoir quelles instructions doivent être répétées.

Voici ce que donne le programme précédent écrit en Python :

```
1 motDePasse = input("Entrez le mot de passe : ") # On commence par demander le mot de passe
2
3 while motDePasse != "AzertY": # Voilà la boucle !
4   print("Mot de passe incorrect !")                # Ces deux instructions sont répétées
5   motDePasse = input("Entrez le mot de passe : ")  # tant que le mot de passe n'est pas correct
6
7 print("Entrée autorisée.") # Si on arrive ici, c'est que l'on est sorti de la boucle
```

☞ *Python est **sensible à la casse**, ce qui signifie que les chaînes de caractères `AzertY` et `azerty` ne sont pas égales pour Python.*

▷ ***Pourquoi commencer par l'instruction*** `motDePasse = input("Entrez le mot de passe : ")` ***?***

C'est obligatoire ! À la ligne 3 du programme, on teste une condition portant sur la variable `motDePasse`.
Si cette dernière n'existait pas, le programme renverrai une erreur du style `NameError: name 'motDePasse' is not defined`.

▷ ***Et si l'utilisateur ne trouve jamais le mot de passe ?***

On fait alors face à la hantise du programmeur : la **boucle infinie**.
Si l'utilisateur ne trouve pas le mot de passe, on lui demande, encore et encore, et le programme ne se termine jamais.

En pratique, on peut laisser 3 chances à l'utilisateur de trouver le mot de passe. Après 3 essais infructueux, on peut sortir de la boucle et afficher un message d'erreur. Nous verrons cela en exercice...

5.1.2 Tables de multiplication

Considérons à présent le problème suivant :

Problème. *Écrire un programme qui affiche la table de multiplication de 7.*

▷ ***Trop facile ! Voilà le programme :***

```
print(1, "*", 7, "=", 1*7)
print(2, "*", 7, "=", 2*7)
print(3, "*", 7, "=", 3*7)
print(4, "*", 7, "=", 4*7)
print(5, "*", 7, "=", 5*7)
print(6, "*", 7, "=", 6*7)
print(7, "*", 7, "=", 7*7)
print(8, "*", 7, "=", 8*7)
print(9, "*", 7, "=", 9*7)
print(10, "*", 7, "=", 10*7)
```

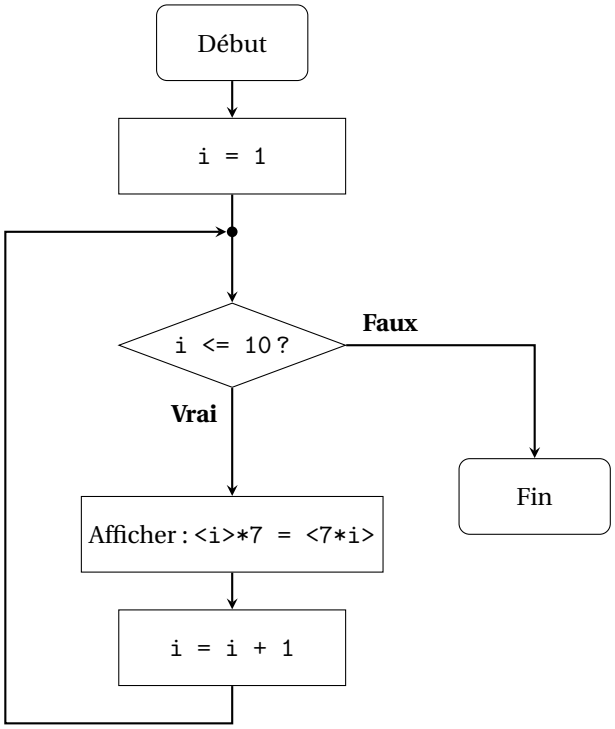
Effectivement, ce programme fonctionne. Mais c'est long et surtout **répétitif**.
Les instructions ne sont pas toutes identiques mais elles sont très similaires. Elles sont toutes de la forme :

```
print(i, "*", 7, "=", i*7)
```

où `i` est un entier variant entre 1 et 10.
Il suffit alors d'utiliser une variable, et le tour est joué :

```
i = 1

while i <= 10: # On boucle tant que i <= 10
  print(i, "*", 7, "=", i*7)
  i = i + 1 # On incrémente i
```



La ligne la plus importante de ce petit programme est certainement la dernière :

```
  i = i + 1 # On incrémente i
```

Essayez de la retirer, et vous verrez pourquoi...

Remarque. Cette dernière ligne peut s'écrire plus rapidement à l'aide de l'opérateur `+=` :

```
  i += 1
```

5.2 Parcourir une chaîne de caractères avec `for`

Terminons ce chapitre avec le problème suivant :

Problème. *Écrire une fonction* `nombreVoyelles(mot)` *qui retourne le nombre voyelles présentes dans* `mot` .

Nous connaissons déjà la fonction `len` , qui permet de compter le nombre de lettres d'une chaîne de caractères.
Pour compter le nombre de voyelles, il faudrait regarder **lettre par lettre** et vérifier si c'est une voyelle ou une consonne.

Grâce à l'instruction `for` , il est possible de parcourir la chaîne de caractères lettre par lettre. On utilise la syntaxe suivante :

```
for lettre in chaine:
  # Instructions à répéter
```

Par exemple, on peut faire :

```
1 for lettre in "Bonjour":
2   print(lettre)
```

On affiche lettre après lettre la chaîne `"Bonjour"` .

```
B
o
n
j
o
u
r
```

Dans le programme précédent, la fonction `print` n'est pas appelée une fois mais 7 fois (autant de fois que la chaîne contient de lettres). D'où les 7 affichages successifs.
Plutôt que d'afficher cette lettre (ce qui ne sert pas à grand chose), nous allons tester si c'est une voyelle.
À quelle condition notre lettre est une voyelle ? Si c'est une des lettres suivantes :

a e i o u y A E I O U Y

☞ *Rappelez-vous : Python est* ***sensible à la casse****.*

▷ ***Ça fait beaucoup de conditions à tester... D'abord*** `lettre == "a"` ***, après*** `lettre == "e"` ***...***

C'est vrai, mais nous connaissons le mot clé `in` .
Ainsi, `lettre` sera une voyelle si le booléen `lettre in "aeiouyAEIOUY"` est vrai. Il suffisait d'y penser !
Reprenons notre programme, en ajoutant une instruction conditionnelle dans la boucle. Attention les yeux :

```
1 for lettre in "Bonjour":
2   if lettre in "aeiouyAEIOUY":
3     print("Voyelle")
4   else:
5     print("Consonne...")
```

```
Consonne...
Voyelle
Consonne...
Consonne...
Voyelle
Voyelle
Consonne...
```

▷ *Ok... Mais on ne voulait pas le nombre de voyelles ?*

En effet. Et quoi de mieux qu'une variable pour compter le nombre de voyelles ?

```
1 nbVoyelles = 0                  # On initialise le nombre de voyelles à 0
2
3 for lettre in "Bonjour":        # Pour chaque lettre du mot Bonjour
4   if lettre in "aeiouyAEIOUY":  # Si c'est une voyelle...
5     nbVoyelles += 1             # On incrémente le nombre de voyelles
```

Après exécution du programme, on peut vérifier que *Bonjour* contient 3 voyelles. Étonnant non ?

```
>>> nbVoyelles
3
```

Pour répondre au problème, il nous reste à écrire ce programme à l'intérieur d'une fonction qui retournera le nombre de voyelles du mot passé en paramètre. Attention, le mot n'est plus forcément *Bonjour*...

```
1 def nombreVoyelles(mot):
2   nbVoyelles = 0
3
4   for lettre in mot:
5     if lettre in "aeiouyAEIOUY":
6       nbVoyelles += 1
7
8   return nbVoyelles
```

```
>>> nombreVoyelles("Salut")
2
>>> nombreVoyelles("PROGRAMMATION")
5
```

5.3 QCM

Dans le QCM suivant, chaque question peut admettre **une ou plusieurs bonnes réponses**.

Questions	Réponses
1. Quels mots clés permettent de déclarer une boucle en Python ?	☐ `while` ☐ `if` ☐ `since` ☐ `for`

On considère le programme suivant :

```
1  a = 5
2
3  while a <= 100:
4    a = a / 2
5    b = 10 * a
6    print(b)
```

Questions	Réponses
2. Quel est le problème du programme précédent ?	☐ Une variable n'est pas initialisée ☐ Il y a une boucle infinie ☐ Il y a une erreur de syntaxe ☐ Un problème ? Lequel ?

On considère le programme suivant :

```
1  mot = "Salut"
2  compteur = 1
3
4  for l in mot:
5    compteur += len(mot)
6
7  print(compteur)
```

Questions	Réponses
3. Qu'affiche le programme précédent ?	☐ Rien du tout ☐ 6 ☐ 26 ☐ 106

5.4 Exercices

5.4.1 Échauffement

Boucle `while`

Exercice 01 Écrire un programme qui affiche les nombres entiers de 1 à 20.

Exercice 02 Écrire un programme qui affiche les nombres entiers de 20 à 37.

Exercice 03 Écrire un programme qui affiche les nombres entiers de 1 à 15, dans l'ordre décroissant.

Exercice 04 Écrire un programme qui affiche les nombres impairs de 7 à 29.

Exercice 05 Écrire une fonction `motivation(n)` qui écrit `n` fois la phrase « Je finirais par comprendre les boucles ».

Exercice 06 Écrire une fonction `calculMental()` qui demande à l'utilisateur « Combien font 3×4 ? » tant qu'il ne donne pas la bonne réponse (qui est 12, au cas où...).

Boucle `for`

Exercice 07 Écrire une fonction `epeler(mot)` qui affiche successivement les lettres de `mot`.

```
>>> epeler("salut")
s
a
l
u
t
```

Exercice 08 Écrire une fonction `voyelles(mot)` qui affiche uniquement les voyelles de `mot`. Les consonnes ne sont pas affichées.

```
>>> voyelles("salut")
a
u
```

Exercice 09 Modifier la fonction précédente pour qu'elle renvoie la chaîne de caractères correspondante, sans affichage.
On utilisera une variable temporaire afin d'enregistrer la chaîne à retourner.

```
>>> voyelles("salut")
au
```

Exercice 10 Écrire une fonction `cacherVoyelles(mot)` qui remplace les voyelles de `mot` par `*` et retourne la chaîne correspondante.

```
>>> cacherVoyelles("salut")
s*l*t
```

Exercice 11 Écrire une fonction `doubler(mot)` qui retourne la chaîne de caractères formée en doublant les lettres de `mot`.

```
>>> double("salut")
ssaalluutt
```

Exercice 12 Écrire une fonction `serpent(mot)` qui triple tous les `s` de `mot`.

```
>>> serpent("salut")
sssalut
```

5.4.2 Entraînement

Exercice 01 Écrire une fonction `tableMultiplication(n)` qui affiche la table de multiplication de `n`.

Exercice 02 Écrire une fonction `sommeEntiers(n)` qui calcule la somme des entiers de 1 à `n`.

```
>>> sommeEntiers(10) # 1 + 2 + ... + 10
55
>>> sommeEntiers(89) # 1 + 2 + ... + 89
4005
```

Exercice 03 Écrire une fonction `demanderMotDePasse()` qui demande à l'utilisateur de saisir un mot de passe deux fois de suite, pour vérification.

- Si l'utilisateur a saisi deux mots de passe identiques, on affiche *Mot de passe enregistré.*
- Sinon, tant que les mots de passe sont différents, on affiche *Les mots de passe sont différents* et on lui demande à nouveau de saisir le mot de passe deux fois de suite.

Exercice 04 ***Nombres premiers***

Un nombre premier est un entier naturel possédant exactement deux diviseurs positifs : 1 et lui même.

1. Écrire une fonction `estPremier(n)` qui retourne `True` si `n` et premier, et `False` sinon.
2. Écrire une fonction `nombresPremiersInferieursA(N)` qui affiche les nombres premiers inférieurs ou égaux à `N`.

```
>>> estPremier(37)
True
>>> nombresPremiersInferieursA(100)
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

☞ *Pour écrire les nombres en ligne, on pourra utiliser le* ***paramètre nommé*** `end` *de la fonction* `print`.
En écrivant `print(variable, end=" ")`, *Python écrira le contenu de* `variable` *puis un espace, sans aller à la ligne.*

Exercice 05

1. Combien y a-t-il de 0 dans l'écriture décimale de 2^{2019} ?
2. Écrire une fonction `nombreZeros(n)` qui retourne le nombre de zéros dans l'écriture décimale d'un nombre entier `n`.

Exercice 06 Écrire une fonction `verlan(mot)` qui retourne la chaîne de caractères `mot` **écrite à l'envers**.

☞ *Pensez à utiliser une variable pour stocker progressivement le mot à l'envers...*

Exercice 07 ***Un peu de hack...***

Réécrire la fonction `fonctionSecrete` suivante :

☞ *La fonction* ***renvoie*** *une valeur, et n'affiche rien.*

```
>>> fonctionSecrete("abc")
abbccc
>>> fonctionSecrete("test")
teesssttt
>>> fonctionSecrete("aEiOuY")
aEEiiiOOOOuuuuuYYYYYY
```

Exercice 08 ***Nombres parfaits***

1. Écrire une fonction `sommeDiviseurs(n)` qui retourne la somme des diviseurs d'un nombre entier `n`.
2. Un nombre est dit **parfait** si la somme de ses diviseurs est égale au double de ce nombre.

   Écrire une fonction `estParfait(n)` qui retourne `True` si le nombre `n` est parfait, et `False` sinon.
3. Écrire une fonction `nombresParfaitsInferieursA(N)` qui affiche les nombres parfaits inférieurs ou égaux à `N`.

Exercice 09 ***Suite de Syracuse***

La suite de Syracuse est une suite de nombres définie de la façon suivante :

- on choisit un entier N non nul
- s'il est pair, on le divise par 2
- sinon, on le multiplie par 3 et on lui ajoute 1
- on recommence l'opération avec le résultat obtenu

On obtient ainsi une suite de nombres. Ce qui est remarquable avec cette suite, c'est que quelque soit le nombre N choisi au départ, on finit toujours par tomber sur 1 au bout d'un certain nombre d'essais. Personne n'a aujourd'hui réussi à démontrer que ce résultat était vrai pour tous les entiers...

1. Vérifiez « à la main » qu'en partant de 6, on finit par tomber sur 1.
2. Écrire une fonction `syracuse(N)` qui affiche la suite de Syracuse en partant de N, jusqu'à 1.

```
>>> syracuse(5)
5 16 8 4 2 1
>>> syracuse(7)
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

5.4.3 Étirements

Exercice 10 ***Encore des mots de passe... (*)***

☞ *Dernier exercice sur les mots de passe, promis !*

Écrire une fonction `verificationMotDePasse(motDePasse)` qui vérifie si un mot de passe est suffisamment robuste.

Pour être suffisamment robuste, le mot de passe doit :

- contenir au moins 8 caractères
- contenir au moins 3 chiffres
- contenir au moins un caractère spécial parmi les suivants : `$ # ? ! @ _`

Exercice 11 ***Conversions (*)***

1. Écrire une fonction `dec2bin(n)` qui retourne la valeur en binaire du nombre décimal `n`. La valeur de retour doit être une chaîne de caractères, précédée des caractères `"0b"`.
2. Écrire une fonction `dec2hex(n)` qui retourne la valeur en héxadécimal du nombre décimal `n`. La valeur de retour doit être une chaîne de caractères, précédée des caractères `"0x"`.
3. (**) Écrire les fonctions `bin2dec(b)` et `hex2dec(h)` qui retournent la valeur décimale du nombre correspondant. `b` et `h` sont des chaînes de caractères commençant par `"0b"` ou `"0x"`. La valeur de retour doit être un entier.

   ☞ *On pourra utiliser l'**algorithme de Hörner***

```
>>> dec2bin(11)
'0b1011'
>>> dec2hex(170)
'0xAA'
>>> bin2dec("0b1111")
15
>>> hex2dec("0xAB")
171
```