Chapitre 1
Affectation, Boucle et Si

Ce chapitre est le premier chapitre de l'année de Première NSI. Il pose les bases de la programmation en Python : les variables et leurs types, l'affectation, les instructions conditionnelles et les boucles. Tous les programmes écrits cette année utiliseront ces constructions élémentaires.

I. Introduction

Définition 1.1.
Un **algorithme** est une séquence finie et univoque d'instructions permettant de résoudre une classe de problèmes.

■ **Exemple 1.1.**
L'addition avec retenue est un algorithme que vous connaissez et utilisez depuis le primaire ! Il permet d'additionner deux entiers positifs quels qu'ils soient. ■

R Un algorithme doit pouvoir s'exécuter sans ambiguïté : chaque étape est parfaitement définie (c'est le sens du mot *univoque*) et le nombre d'étapes est fini : l'algorithme se termine toujours.

II. Variables et instructions élémentaires

1. Types de variables

En informatique, pour stocker les données, on utilise des **variables**. Une variable est définie par un nom qui permet de l'identifier de manière unique. Ce nom permet à l'ordinateur de retrouver la donnée contenue dans sa mémoire qui correspond à la valeur de la variable. Chaque variable est caractérisée par un **type** qui correspond au type de donnée qu'elle contient. Il existe de nombreux types de variables différents en informatique mais nous n'en étudierons que quatre à ce stade.

R Une variable est la donnée de : **1 NOM, 1 type et 1 donnée**.

Type booléen

Une variable de **type booléen** (ou variable booléenne) est une variable qui prend l'une des deux valeurs : **vrai** ou **faux**. En Python, le type booléen se note `bool` et les valeurs associées `True` et `False` respectivement.

Type entier

Le **type entier** désigne, comme son nom l'indique, des variables entières (qui appartiennent donc à l'ensemble $\mathbb{N}$). En Python, le type entier se note `int` (contraction de « integer » qui signifie entier en anglais).

Type flottant

Une variable de **type flottant** (ou variable flottante) contient un nombre qui s'écrit avec un nombre fini de chiffres après la virgule. Le type flottant se note `float` en Python (qui signifie flotter en anglais). Attention, la virgule est remplacée par un point dans l'écriture anglo-saxonne des nombres et donc, en particulier, dans le langage Python.

Type chaîne de caractères

Une variable de **type chaîne de caractères** contient du texte. Le terme caractère désigne les caractères typographiques que sont les lettres (minuscules ou majuscules, avec ou sans accents), les chiffres, les signes de ponctuation, etc. Dans le langage Python, ce type est noté `str` (contraction de « string » qui signifie fil ou enchaînement en anglais). Les chaînes de caractères s'écrivent en Python par du texte entre guillemets simples (par exemple : `'Ceci est un string en Python.'`) ou guillemets doubles (par exemple : `"Ceci est également un string en Python!"`).

Type	Nom en Python	Exemple de valeur
Booléen	`bool`	`True`, `False`
Entier	`int`	`42`, `-3`
Flottant	`float`	`3.14`, `-0.5`
Chaîne de caractères	`str`	`'NSI'`, `"Python"`

2. Affectation

Définition 1.2.

L'**affectation** d'une valeur à une variable est l'action de donner une valeur à cette variable.

On l'écrit en Python avec le signe « = ». Le code suivant affecte à la variable `a` la valeur 3.

```
a = 3
```

Après cette instruction, la variable `a` contient la valeur 3.

> **R** Le signe « = » n'a donc pas le même sens en Python qu'en mathématiques ! En particulier, en Python, « a = x » n'a pas le même sens que « x = a » : la valeur est toujours rangée dans la variable située à gauche du signe « = ».

En programmation, il est possible d'affecter et de **réaffecter** des variables. Dans ce cas, c'est la dernière affectation qui prévaut sur les autres. On peut également mettre à jour une variable en lui réaffectant une valeur qui dépend d'elle-même.

▪ Exemple 1.2.
Mise à jour d'un compteur : à chaque ligne, la nouvelle valeur de `n` dépend de l'ancienne. ▪

```
n = 0
n = n + 1
n = n + 1
print(n)
```

Ce programme affiche `2` : la variable `n` a été réaffectée deux fois, et c'est la dernière valeur qui est affichée.

▪ Exemple 1.3.
Calcul pas à pas : on stocke des valeurs dans des variables, puis on combine ces variables pour calculer une moyenne. ▪

```
a = 12
b = 8
moyenne = (a + b) / 2
print(moyenne)
```

Ce programme affiche `10.0`. Remarquez que la division de deux entiers produit ici un flottant.

▪ Exercice 1.1.
Le code Python suivant affecte plusieurs valeurs successives à `a` en réalisant plusieurs affichages (la fonction `print` permet de réaliser un affichage en Python). Sans l'exécuter, que va renvoyer la console Python lorsque l'on exécute ce code ?

```
a = -3
a = 1
print(a)
a = a + 1
print(a)
```

▪

3. Opérations élémentaires

À chaque type est associé un ensemble d'opérations. Vous trouverez ici la liste des opérations les plus élémentaires associées aux quatre types décrits précédemment.

R Tous les types admettent deux opérations universelles : le **test d'égalité** et le **test de différence**. Le test d'égalité, noté « = » en pseudo-code et « == » en Python, permet de vérifier si deux variables sont égales. Le test de différence, noté « $\neq$ » en pseudo-code et « != » en Python, permet de vérifier si deux variables sont différentes. Ces deux opérations renvoient une valeur booléenne.

Exercice 1.2.
Le code Python suivant affecte la valeur 2 à la variable `a` et la valeur -3 à la variable `b`. Sans l'exécuter, que va afficher ce programme ?

```
a = 2
b = -3
print(a == b)
print(a != b)
```

■

Opérations sur les flottants

Pour les flottants, on peut utiliser les opérations classiques entre deux nombres : l'addition, la soustraction, la multiplication, la division, l'élévation à la puissance ; ainsi que les comparateurs d'ordre : inférieur ou égal, inférieur strict, supérieur ou égal, supérieur strict. La liste des symboles utilisés en Python figure dans le tableau ci-dessous :

Opération mathématique	Expression en Python
$a + b$	`a + b`
$a - b$	`a - b`
$a \times b$	`a * b`
a/b	`a / b`
a^b	`a ** b`
$a \leq b$	`a <= b`
$a < b$	`a < b`
$a \geq b$	`a >= b`
$a > b$	`a > b`

Opérations sur les entiers

Parmi les opérations standards entre deux entiers, on retrouve toutes les opérations décrites pour les flottants auxquelles on peut ajouter le quotient et le reste par la division euclidienne (c'est-à-dire la division entière).

Opération mathématique	Expression en Python
Quotient de la division entière de a par b	`a // b`
Reste de la division entière de a par b	`a % b`

■ **Exercice 1.3.**
Dans le code suivant, que valent les variables `a`, `b`, `c`, `d`, `e`, `f` ?

```
a = 3 - 1
b = 1 + 2
c = a ** 3
d = c / 5
e = c // 5
f = (a <= b)
```

■

■ **Exemple 1.4.**
Comparons la division réelle, la division entière et le reste de la division euclidienne pour 17 divisé par 5 : ■

```
print(17 / 5)
print(17 // 5)
print(17 % 5)
```

Ce programme affiche `3.4` puis `3` puis `2`. En effet : $17 = 3 \times 5 + 2$.

■ **Exemple 1.5.**
Le reste de la division euclidienne permet de tester la parité d'un entier : un entier est pair si son reste dans la division par 2 vaut 0. ■

```
reste = 7 % 2
print(reste)
```

Ce programme affiche `1` : le reste de la division de 7 par 2 vaut 1, donc 7 est impair.

Opérations sur les booléens

Les opérations standards sur les booléens sont :
— la **négation**, associée au mot-clé **non** ;
— la **conjonction**, associée au mot-clé **et** ;

— la **disjonction**, associée au mot-clé **ou**.

Elles sont implémentées en Python à l'aide des opérateurs `not`, `and` et `or` respectivement. Si a et b sont deux variables booléennes, on a :

— **non**(a) vaut **vrai** si et seulement si a vaut **faux** ;
— a **et** b vaut **vrai** si et seulement si les deux variables a et b sont égales à **vrai** ;
— a **ou** b vaut **vrai** si et seulement si l'une au moins des deux variables a et b vaut **vrai**.

On peut résumer ces règles dans des tables de vérité :

a	b	a **and** b	a **or** b
`True`	`True`	`True`	`True`
`True`	`False`	`False`	`True`
`False`	`True`	`False`	`True`
`False`	`False`	`False`	`False`

a	**not** a
`True`	`False`
`False`	`True`

■ Exercice 1.4.
Analysez le code Python suivant. Quelles valeurs prennent les variables booléennes `a`, `b`, `c` ?

```
a = (0 < 1) or (0 > 2)
b = not (1 < 2)
c = a and b
```

■

■ Exemple 1.6.
Combinons les opérateurs booléens pour tester un âge : ■

```
age = 16
print(age >= 13)
print(age >= 13 and age < 18)
print(not (age >= 18))
```

Ce programme affiche `True`, `True`, `True` : l'âge 16 est bien supérieur ou égal à 13, il est compris entre 13 et 18, et il n'est pas supérieur ou égal à 18.

Opérations sur les chaînes de caractères

Il existe un certain nombre d'opérations sur les chaînes de caractères en Python mais nous n'en considérons qu'une ici : la **concaténation**. La concaténation permet de créer une nouvelle chaîne de caractères à partir de deux chaînes de caractères a et b, en mettant les caractères de b à la suite des caractères de a. La concaténation de a et b en Python se note `a + b`.

R L'opérateur `*` permet également de répéter une chaîne de caractères : `"ha" * 4` vaut `"hahahaha"`.

■ **Exercice 1.5.**
On considère le code Python suivant. Que valent les variables `a`, `b`, `c`, `d` ?

```
a = "math"
b = a + "ematiques"
c = "ha"
c = c + c
d = c * 4
```

■

4. Instructions conditionnelles

Un père dit à son enfant : « Si tu finis tes légumes, Alors tu auras un dessert ». L'enfant ne finit pas ses légumes mais le père donne quand même à son enfant un dessert. Le père a-t-il menti ?

La réponse est non. Et ce n'est pas pour des raisons d'éducation ou de morale. Le père a dit ce qu'il ferait si son enfant finit ses légumes mais il n'a rien dit s'il ne finissait pas ses légumes.

Dans un algorithme, on peut choisir qu'une instruction ne s'exécute que si certaines conditions sont remplies. En pseudo-code, on peut utiliser les mots-clés **si** et **alors** pour indiquer une condition et la séquence d'instructions à exécuter si la condition est remplie. On peut également utiliser le mot-clé **Fin si** pour indiquer la fin de la séquence d'instructions conditionnelles. Une instruction conditionnelle peut donc s'écrire en pseudo-code sous la forme suivante :

```
Si la condition est vraie
    Alors faire
        instructions
Fin Si
```

En Python, on utilise la commande `if` associée à l'utilisation des deux points `:` et d'une indentation (décalage vers la droite des lignes d'instructions), selon la forme suivante :

```
if condition:
    instruction
```

R Les règles pour les deux points et l'indentation ne sont pas facultatives et le code ne fonctionnera pas correctement si elles ne sont pas respectées. Par ailleurs, l'indentation doit courir sur l'ensemble des instructions couvertes par la condition (donc éventuellement sur plusieurs lignes).

On peut également différencier un certain nombre de cas en utilisant les mots-clés « sinon si » et « sinon » en pseudo-code ou les commandes `elif` et `else` en Python, selon le schéma qui suit :

```
Si la condition1 est vraie :
    Alors faire
        instructions1
Sinon si la condition2 est vraie :
    Alors faire
        instructions2
Sinon si la condition3 est vraie :
    Alors faire
        instructions3
Sinon :
    Faire
        instructions4
Fin Si
```

En Python, cela donne :

```
if condition1:
    instruction1
elif condition2:
    instruction2
elif condition3:
    instruction3
else:
    instruction4
```

■ **Exemple 1.7.**
Deux joueurs s'affrontent aux dés. Le joueur qui obtient la plus grande valeur a gagné. Le programme suivant récupère la valeur obtenue par chacun des joueurs puis annonce qui a gagné la partie (on utilise pour cela les fonctions `int` et `input` de Python qui seront présentées en détails dans la suite de l'année). ■

```
d1 = int(input("Entrez la valeur obtenue par le joueur 1 : "))
d2 = int(input("Entrez la valeur obtenue par le joueur 2 : "))
if d1 > d2:
    print("Le joueur 1 a gagné")
elif d2 > d1:
    print("Le joueur 2 a gagné")
else:
    print("Match nul !")
```

■ **Exemple 1.8.**
Test de parité : un entier est pair si son reste dans la division par 2 vaut 0. ■

```
nombre = 7
if nombre % 2 == 0:
    print("Le nombre est pair")
else:
```

```
    print("Le nombre est impair")
```

Ce programme affiche `Le nombre est impair`.

Exemple 1.9.
Test de majorité : on compare l'âge saisi par l'utilisateur à la majorité légale. ■

```
age = 16
if age >= 18:
    print("Vous êtes majeur")
else:
    print("Vous êtes mineur")
```

Ce programme affiche `Vous êtes mineur`.

5. Boucles

En algorithmique, les boucles permettent de répéter une séquence d'instructions sans avoir à réécrire la séquence. On distingue deux formes de boucles :

— les **boucles bornées** « pour », pour lesquelles la répétition de la séquence d'instructions correspond au parcours de tous les éléments d'un ensemble fini par une variable ;

— les **boucles non bornées** « tant que », pour lesquelles la répétition de la séquence d'instructions est soumise à condition de répétition à chaque tour de la boucle.

Les boucles bornées sont appelées ainsi parce qu'elles s'arrêtent quand on arrive au bout de l'ensemble fini correspondant. Les boucles non bornées sont appelées ainsi parce qu'elles ne s'arrêtent que si la condition de répétition est fausse.

La boucle bornée « pour »

Dans ce cours, on ne considère que les boucles « pour » où la répétition de la séquence correspond au parcours par une variable d'un intervalle d'entiers (par exemple, $\{4, 5, 6, 7, 8, 9\}$) et, le plus souvent, un intervalle d'entiers débutant en 0 (par exemple, $\{0, 1, 2, 3, 4, 5\}$). En pseudo-code, on pourra utiliser le mot-clé « pour » en début de boucle et indiquer la fin de cette boucle par un « fin pour ».

```
Pour i allant de 1 àn
    Faire
        instructions
Fin Pour
```

En Python, on utilise la commande `for` associée à une instruction de la forme `i in range(n)` qui signifie que `i` parcourt l'intervalle $\{0, 1, 2, \ldots, n-1\}$, et suivie d'un « : ». Une indentation permet ensuite d'indiquer les instructions correspondant à la séquence d'instructions à répéter.

```
for i in range(n):
    instructions # n doit être défini avant, sinon NameError
```

■ **Exemple 1.10.**
Calculons la somme des entiers de 1 à 5 à l'aide d'une boucle bornée : ■

```
somme = 0
for i in range(1, 6):
    somme = somme + i
print(somme)
```

Ce programme affiche `15`. La variable `somme` est initialisée à 0 avant la boucle, puis on lui ajoute à chaque tour la valeur de i : on calcule ainsi $0+1+2+3+4+5=15$. Remarquez que la commande `range(1, 6)` fait parcourir à i les entiers de 1 inclus à 6 exclus.

■ **Exemple 1.11.**
Affichons la table de multiplication par 7 : ■

```
for i in range(1, 11):
    print(7, "x", i, "=", 7 * i)
```

Ce programme affiche `7 x 1 = 7`, `7 x 2 = 14`, ..., `7 x 10 = 70`.

La boucle non bornée « tant que »

Dans la boucle « tant que », la boucle se répète tant qu'une condition de répétition est vraie. En pseudo-code, on peut indiquer le commencement d'une telle boucle par « tant que » et la fin des instructions de la boucle par « fin tant que ».

```
Tant que la condition est vraie
    Faire
        instructions
Fin Tant que
```

En Python, la boucle « tant que » passe par la commande `while`, associée à une condition, et suivie d'un deux-points. Comme usuellement en Python, une indentation permet d'indiquer les instructions correspondant à la séquence d'instructions à répéter.

```
while condition:
    instruction
```

> **R** Attention, si la condition est toujours vérifiée, la boucle ne s'arrêtera jamais. On dit que le programme tourne en **boucle infinie**. Généralement, ce n'est pas souhaitable. Il faut donc s'assurer que la condition de répétition puisse passer de la valeur vrai à la valeur faux lors de la séquence d'instructions de la boucle. Si cela arrive, il faut en général interrompre le programme « de force » en tapant **Control-C**.

■ **Exemple 1.12.**
Un compte à rebours avec une boucle non bornée : ■

```
i = 10
while i > 0:
    print(i)
    i = i - 1
print("Décollage !")
```

Ce programme affiche `10`, `9`, ..., `1`, puis `Décollage !`. La variable `i` diminue à chaque tour : la condition `i > 0` finit par devenir fausse et la boucle s'arrête.

On remarquera que dans une boucle non bornée, la variable qui apparaît dans la condition est affectée à une valeur avant le début de la boucle. Ceci est un schéma classique en programmation qui s'appelle **l'initialisation**. On dit, par exemple, que la variable `i` est initialisée à 0 avant une boucle de comptage croissant, et initialisée à 10 avant un compte à rebours.

> **R** L'initialisation d'une variable avant une boucle est indispensable : elle fixe l'état de départ de la boucle. Pour une boucle bornée, la variable qui parcourt l'intervalle est initialisée automatiquement par la commande `range` ; pour une boucle non bornée, il faut initialiser soi-même la variable qui apparaît dans la condition.

6. Erreurs et bugs

Les erreurs font partie de la vie du programmeur ! Il est important de savoir les reconnaître et les corriger. Voici les grandes familles d'erreurs rencontrées en Python :

- **SyntaxError** : les erreurs de syntaxe sont dues au non-respect des règles d'écriture de Python, comme un oubli de parenthèse ou un manque de « : ».
- **NameError** : les erreurs de définition sont dues à l'usage d'un nom qui n'a pas été défini, par exemple une variable ou une fonction qui n'a pas été définie.
- **TypeError** : les erreurs de type sont dues à des valeurs dont le type n'est pas compatible avec l'expression dans laquelle elles apparaissent.
- **Erreur d'exécution** : les erreurs sont dues à des instructions que l'ordinateur ne sait pas exécuter comme une division par zéro.
- **Erreur de logique** : ces dernières n'occasionnent en général pas de message d'erreur car il s'agit d'erreurs dans la logique du programme, qui n'enfreignent pas les règles de Python. Ce sont les erreurs les plus difficiles à résoudre. Une erreur de logique courante est la boucle infinie : le programme ne s'arrête jamais car la condition d'une boucle non bornée ne devient jamais fausse.

▪ Exemple 1.13.
Une parenthèse non fermée provoque une **SyntaxError** : ▪

```
print("Bonjour"
```

▪ Exemple 1.14.
L'utilisation d'un nom qui n'a jamais été défini provoque une **NameError** : ▪

```
print(x)
```

■ **Exemple 1.15.**
Additionner un entier et une chaîne de caractères provoque une **TypeError** : ■

```
x = 3 + "chat"
```

■ **Exemple 1.16.**
Diviser par zéro provoque une **ZeroDivisionError** : ■

```
y = 1 / 0
```

III. Exercices

Les exercices suivants reprennent et complètent les notions du chapitre. Pensez à bien lire les énoncés, à tester vos programmes, et à interpréter les messages d'erreur éventuels.

Exercices d'automatismes

Les exercices d'automatismes exercent les réflexes fondamentaux : calcul mental d'expressions Python, lecture de code court, prédiction d'affichage, complétion de conditions. Chaque question doit pouvoir être traitée en moins de trente secondes.

■ Exercice 1.6.

Calculer mentalement la valeur des expressions Python suivantes.

1. Calculer mentalement la valeur de l'expression Python suivante.

```
2 + 3 * 4
```

2. Donner les valeurs affichées par le programme suivant.

```
print(17 // 5)
print(17 % 5)
```

3. Calculer mentalement la valeur de l'expression Python suivante.

```
(7 + 5) % 4
```

4. Calculer mentalement la valeur de l'expression Python suivante (attention à l'ordre des opérations).

```
20 - 2 * 3 ** 2
```

■

■ Exercice 1.7.

Déterminer le type ou la valeur des expressions Python suivantes.

1. Quel est le type de la valeur renvoyée par chacune des deux expressions suivantes ?

```
10 / 2
10 // 2
```

2. Donner la valeur de chacune des deux expressions suivantes.

```
"3" + "2"
"3" * 2
```

3. Quel est le type de la valeur renvoyée par l'expression suivante ?

```
3 == 3
```

■

■ **Exercice 1.8.**
Prédire l'affichage des programmes suivants, sans les exécuter.

1. Qu'affiche le programme suivant ?

```
x = 5
y = 2
print(x + y)
```

2. Qu'affiche le programme suivant ?

```
n = 4
n = n * 2
n = n + 1
print(n)
```

3. Qu'affiche le programme suivant ?

```
a = 3
b = 8
a = b
b = a
print(a, b)
```

4. Qu'affiche le programme suivant ?

```
total = 0
for i in range(4):
    total = total + i
print(total)
```

■

■ **Exercice 1.9.**
Comparaisons, opérateurs logiques et conditions : indiquer si chacune des expressions suivantes vaut `True` ou `False`, puis déterminer la valeur prise par la variable dans chacun des programmes suivants.

1. Indiquer si chacune des expressions suivantes vaut `True` ou `False`.

```
7 > 4
3 == 5
"chat" == "Chat"
```

2. Indiquer si chacune des expressions suivantes vaut `True` ou `False`.

```
3 > 2 and 5 < 1
5 > 3 or 2 > 10
not (4 == 4)
```

3. Quelle valeur prend la variable `categorie` après exécution du programme suivant ?

```
age = 15
if age < 13:
    categorie = "enfant"
elif age < 18:
    categorie = "adolescent"
else:
    categorie = "adulte"
```

4. Quelle valeur prend la variable `resultat` après exécution du programme suivant ?

```
x = 7
if x > 10:
    resultat = "grand"
elif x > 5:
    resultat = "moyen"
else:
    resultat = "petit"
```

■

■ **Exercice 1.10.**

Boucles et itérations : déterminer le nombre d'exécutions ou la valeur affichée par chacun des programmes suivants, et compléter la condition demandée.

1. Combien de fois le corps de chacune des deux boucles suivantes est-il exécuté ?

```
for i in range(10):
    pass
```

```
for i in range(2, 9):
    pass
```

2. Quelle valeur est affichée par le programme suivant ?

```
compteur = 0
while compteur < 3:
    compteur = compteur + 1
print(compteur)
```

3. Quelle valeur est affichée par le programme suivant ?

```
n = 1
```

```
while n < 20:
    n = n * 2
print(n)
```

4. Recopier et compléter la condition de la boucle pour que la boucle s'arrête lorsque `compteur` atteint la valeur 5.

```
compteur = 0
while _____:
    compteur = compteur + 1
```

5. Le programme suivant provoque-t-il une boucle infinie ? Expliquer la réponse en une phrase.

```
i = 0
while i < 5:
    print(i)
```

■

Exercices d'application

Les exercices d'application reprennent les notions du chapitre, du plus simple au plus difficile. Ils demandent de lire ou d'écrire de courts programmes.

■ **Exercice 1.11.**
On considère le programme suivant. Recopier et compléter un tableau donnant, après chaque ligne du programme, les valeurs des variables `a`, `b` et `c`. Que vaut la variable `a` à la fin du programme ?

```
a = 5
b = 2
a = a + b
c = a * b
b = c - a
a = a - b
```

■

■ **Exercice 1.12.**
Écrire un programme Python qui :
1. affecte la valeur 120 à la variable `prix_ht` ;
2. calcule le prix TTC en appliquant une TVA de 20 % (le prix TTC vaut `prix_ht` $\times$ 1,2) ;
3. affiche le prix TTC à l'aide de la fonction `print`.

On pourra tester le programme avec d'autres valeurs de `prix_ht`. ■

■ **Exercice 1.13.**
On considère le code Python suivant.

1. Que fait le programme si l'utilisateur rentre la valeur 15 ?
2. Et la valeur 1515 ?

```
x = int(input("Veuillez saisir un nombre entier."))
if x < 100:
    print("Votre nombre est bien faible !")
else:
    print("Quel beau nombre !")
print("Au revoir !")
```

■

■ **Exercice 1.14.**
On considère le code Python suivant.

1. Quelle valeur est contenue dans la variable `a` après exécution du programme ?
2. Comment modifier le code pour produire un fou rire ?

```
a = "A"
for k in range(2):
    a = a + "ha"
a = a + "!"
print(a)
```

■

■ **Exercice 1.15.**
Écrire un programme qui demande un nombre entier à l'utilisateur (avec les fonctions `input` et `int`), puis affiche si ce nombre est positif, négatif ou nul. ■

■ **Exercice 1.16.**

1. Faire une boucle qui affiche les noms de toute votre famille.
2. Faire une boucle qui compte tous les nombres de 1 à 1000.
3. Faire une boucle qui dit le nombre de lettres pour le nom de chaque personne de votre famille (utiliser la fonction `len()`).

■

■ **Exercice 1.17.**
Écrire un programme qui demande un entier n à l'utilisateur puis affiche :
— « pair » si n est divisible par 2 ;
— « multiple de 3 » si n est divisible par 3 ;
— « pair et multiple de 3 » si n vérifie les deux propriétés.
Indication : n est divisible par 2 si $n \,\%\, 2 == 0$. ■

■ **Exercice 1.18.**

1. Écrire un programme qui calcule et affiche la somme des entiers de 1 à 100 à l'aide d'une boucle bornée.
2. Modifier le programme pour calculer la somme des entiers pairs de 1 à 100 (les entiers pairs sont les multiples de 2).

■

■ **Exercice 1.19.**
Écrire un programme qui affiche un compte à rebours de 10 à 1 à l'aide d'une boucle non bornée, puis affiche « Décollage ! ». Attention : la variable de la boucle doit évoluer à chaque tour, sinon le programme tourne en boucle infinie ! ■

■ **Exercice 1.20.**
On considère les trois séquences d'instructions suivantes en pseudo-code.

1. On suppose que la variable x contient la valeur 2 avant l'exécution de la séquence. Dans chacun des cas, déterminez la valeur contenue dans x après exécution de la séquence.
2. Identifiez la ou les séquences pour lesquelles, si x est initialisée à a avant l'exécution de la séquence, alors x contient $(a-1)^2+(a+1)^2$ après exécution de la séquence.

```
# Séquence 1
x = 3 # On donne une valeur àx
x = x + 1
b = x ** 2
a = x - 1
c = a ** 2
x = b - c
```

```
# Séquence 2
```

```
x = 2 # On donne une valeur àx
x = x - 1
a = x ** 2
x = x + 2
b = x ** 2
x = a + b
```

```
# Séquence 3
x = 4 # On donne une valeur àx
a = x - 1
b = a ** 2
c = x + 1
d = c ** 2
x = b + d
```

■

Exercices de réflexion

Les exercices de réflexion demandent plus d'analyse : attention aux indentations, repérage d'erreurs, modélisation d'un problème. Ils sont classés du plus abordable au plus exigeant.

■ **Exercice 1.21.**
Pour chacun des programmes suivants, dire quelle valeur prend la variable `d` après exécution du programme. Attention aux indentations !

```
# Programme 1
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a + b
    b = b + a
    c = c + b
    d = d + c
```

```
# Programme 2
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a + b
    b = b + a
    c = c + b
d = d + c
```

```
# Programme 3
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a + b
    b = b + a
c = c + b
d = d + c
```

```
# Programme 4
a = 1
b = 1
c = 1
d = 0
for k in range(2):
    a = a + b
b = b + a
c = c + b
d = d + c
```

■

■ **Exercice 1.22.**
Le programme suivant contient trois erreurs : une `SyntaxError`, une `NameError` et une `TypeError`. Identifier chaque erreur, expliquer pourquoi elle se produit, puis écrire une version corrigée du programme.

```
print("Bonjour"
x = 5
y = "chat"
z = x + y
print(t)
```

■

■ **Exercice 1.23.**
À la fête foraine, l'accès aux montagnes russes est réservé aux personnes mesurant au moins 1 m 20 (compris) et au plus 2 m 10 (compris). Faites un programme Python qui demande à un utilisateur de rentrer sa taille en mètres et l'informe, en fonction de sa réponse, s'il :
- peut monter dans l'attraction ;
- est trop petit ;
- est trop grand.

■

■ **Exercice 1.24.**
Aladin propose le code Python suivant pour échanger la valeur de deux variables a et b.
1. Expliquer pourquoi le code fourni par Aladin ne remplit pas ses objectifs.
2. Proposer une autre solution qui échange effectivement les deux variables.

```
a = 42
b = 23
a = b
b = a
```

■

■ **Exercice 1.25.**
On considère la séquence d'instructions en pseudo-code qui suit.
1. Quelle valeur est affichée lorsque l'on exécute cette séquence ?
2. Implémentez cette séquence en Python et exécutez votre code afin de vérifier

votre réponse.

```
x = 3
y = 11
k = 1
Tant que x < y
    Faire
        x = 3x + 2
        y = 2y + 1
        k = k + 1
Fin Tant que
Afficher k
```

■

■ **Exercice 1.26.**
Un étudiant fauché place 10 euros sur son livret A en 2020, rémunéré 0,5 % d'intérêt par an. À partir de quelle année cet étudiant aura-t-il au moins 20 euros sur son livret ? Réalisez un code Python qui permette de répondre à la question.
■

■ **Exercice 1.27.**
En utilisant le module `random` (taper `import random`) et la commande `random.randint(a, b)` qui tire un nombre au hasard entre a et b (inclus) :

1. Faites une boucle conditionnelle tirant au hasard un nombre entre 0 et 50 jusqu'à obtenir le numéro 1. On affiche le numéro dans la boucle.
2. Cette boucle est-elle infinie et pourquoi ?
3. Faire une boucle qui tire des nombres au hasard jusqu'à avoir un nombre pair.
4. Faire une boucle où le nombre augmente de 3 en 3 jusqu'à dépasser le numéro 1802.

```
import random
x = random.randint(0, 50) # x prend une valeur au hasard entre 0 et 50
```

■

■ **Exercice 1.28.**
Dans le programme Python qui suit, l'une des deux boucles tourne en boucle infinie. Laquelle ? Justifier votre réponse.

```
# Boucle 1
i = 0
while i >= 0:
    i = i + 1
    print(i)
```

```
# Boucle 2
i = 10
while i >= 0:
    i = i - 1
    print(i)
```

■

■ **Exercice 1.29.**

Le jeu du « plus grand, plus petit ». Écrire un programme qui :

1. tire un nombre secret entre 1 et 100 avec la commande `random.randint(1, 100)` ;
2. demande à l'utilisateur de proposer un nombre ;
3. tant que le nombre proposé n'est pas le nombre secret, affiche « Trop grand ! » ou « Trop petit ! » puis redemande un nombre ;
4. lorsque l'utilisateur trouve le nombre secret, affiche « Bravo ! » et le nombre d'essais effectués.

Indication : utiliser une boucle non bornée et une variable compteur d'essais. ■