I Nombres et calculs

1. Additions, soustractions, multiplications

À RETENIR

Python permet d'effectuer tout type de **calcul**, comme une calculatrice ordinaire.

EXEMPLE

```
2+5 # Renvoie le résultat de 2 + 5 (ie. 7).
2*5 # Renvoie le résultat de 2 × 5 (ie. 10).
7**2 # Renvoie le résultat de 7² (ie. 49).
```

2. Division

À RETENIR

Il existe deux types de division en Python : la division **euclidienne** avec l'opérateur `//` et la division **décimale** avec l'opérateur `/`. L'opérateur `%` (appelé modulo) permet d'obtenir le reste d'une division euclidienne.

EXEMPLE

```
13 / 5 # Renvoie le quotient de la division décimale de 13 par 5 (ie. 2,6).
13 // 5 # Renvoie le quotient de la division euclidienne de 13 par 5 (ie. 2).
13 % 5 # Renvoie le reste de la division euclidienne de 13 par 5 (ie. 3).
```

II Variables

1. Généralités

À RETENIR

En informatique, les **variables** sont des espaces mémoire « élémentaires » qui associent un nom (l'identifiant) à une valeur. En Python, on peut affecter une valeur à une variable avec la syntaxe `variable = valeur`. Notre variable sauvegarde alors également le **type** de la valeur.

EXEMPLE

```
a = 2 # La variable a prend la valeur 2. Elle est de type "int" (pour entier).
b = 3 # La variable b prend la valeur 3. Elle est aussi de type "int".
c = a + b # La variable c prend la valeur a + b (ie. 5). Elle est également de type "int".
```

2. Incrémentation, décrémentation

À RETENIR

On a souvent besoin d'**incrémenter** une variable (augmenter sa valeur de 1) ou de la **décrémenter** (diminuer sa valeur de 1). Au lieu d'écrire `a = a + 1`, on peut écrire de façon plus courte : `a += 1`. Cette syntaxe raccourcie fonctionne pour n'importe quelle valeur de l'incrément, ainsi que pour les opérateurs +, −, ∗, /, // et %.

EXEMPLE

```
a = -1
a++ # a est égale à 0.
a += 2 # a est égale à 3.
```

III Chaînes de caractères

1. Affichage d'un message dans la console

À RETENIR

La fonction `print()` permet d'afficher une chaîne de caractères. La chaîne de caractère que l'on souhaite afficher doit être écrite entre guillemets (simples ' doubles " ou triples »'). Sans guillemets, c'est le contenu de la variable dont le nom est transmis en paramètre qui sera affiché.

EXEMPLE

```
bonjour = 8
print(bonjour) # On affiche la valeur de la variable dénommée bonjour.
# À ne pas confondre avec :
print("bonjour") # On affiche "bonjour".
```

2. Concaténation

À RETENIR

Le terme **concaténation** désigne l'action de mettre bout à bout au moins deux chaînes de caractères. Il est possible de faire cela en Python avec l'opérateur + appliqué à deux chaînes.

EXEMPLE

```
str1 = "Je"
str2 = "concatène"
resultat = str1 + " " + str2 # "Je concatène"
```

À RETENIR

On ne peut concaténer que des variables de type `str`. Pour convertir une variable d'un autre type en une chaîne de caractère, on peut utiliser la fonction `str()`. Il existe moulte fonctions de conversion :

— `str()` Pour convertir en chaîne de caractères.

— `int()` Pour convertir en nombre entier.

— `float()` Pour convertir en flottant (nombre à virgule).

EXEMPLE

La ligne suivante renvoie une erreur car on tente d'additionner une chaîne de caractères et un nombre.

```
print("J'ai " + 15 + " ans")
```

3. Récupération de l'entrée utilisateur

À RETENIR

On peut demander à l'utilisateur de saisir une valeur avec la fonction `input()`. On peut ajouter en paramètre facultatif un message qui donne des précisions sur la valeur attendue.

EXEMPLE

```
mois = input("Quel est ton mois d'anniversaire ? ")
```

IV Conditions

1. Booléens

À RETENIR

Un **booléen** est une variable qui ne peut contenir que *Vrai* (`True`) ou *Vrai* (`Faux`). Il existe plusieurs symboles de comparaison :

— < pour *strictement inférieur à*.

— <= pour *inférieur ou égal à*.

— > pour *strictement supérieur à*.

— >= pour *supérieur ou égal à*.

— == pour *égal à* (attention à ne pas confondre avec = qui est le symbole d'affectation d'une variable à une valeur).

— `!=` pour *différent de*.

Pour convertir une variable d'un type donné en une variable booléenne, on utilise la fonction `bool()`.

EXEMPLE

```
a = 18
b = (a > 7) # Contient "True".
c = (a == 6) # Contient "False".
```

À RETENIR

On peut effectuer plusieurs opérations sur les booléens.

— Avec l'opérateur *ET*. Il est défini de la manière suivante : `a` *ET* `b` est *vrai* si et seulement si `a` est *vrai* et `b` est *vrai*. En Python, au lieu d'écrire *ET*, on écrit `and`.

— Avec l'opérateur *OU*. Il est défini de la manière suivante : `a` *OU* `b` est *vrai* si et seulement si `a` est *vrai* ou `b` est *vrai* (ou les deux le sont). En Python, au lieu d'écrire *OU*, on écrit `ou`.

— Avec l'opérateur *NON*. Il est défini de la manière suivante : *NON*(`a`) est *vrai* si et seulement si `a` est *faux*. En Python, au lieu d'écrire *NON*, on écrit `not`.

EXEMPLE

```
a = 12 > 11 and 10 <= 9 # Contient "False".
b = 12 > 11 or 10 <= 9 # Contient "True".
c = not a # Contient "True".
d = not b # Contient "False".
```

2. Blocs `if` / `else`

À RETENIR

Les **conditions** permettent de rendre les programmes adaptatifs en leur donnant la possibilité de se comporter différemment selon qu'une certaine condition est réalisée ou pas. En Python, on utilise les mots-clés `if` (pour *si*) et `else` (pour *sinon*).

EXEMPLE

Le code suivant va demander l'âge de l'utilisateur, convertir l'entrée en nombre entier, puis effectuer la comparaison avec le nombre 18 : si l'entrée est plus grande, alors l'utilisateur est majeur ; sinon, il est mineur.

```
age = int(input('Quel est votre âge ? ')) # On demande l'âge de l'utilisateur.
if age < 18: # On effectue la comparaison avec le nombre 18.
    print('Vous êtes mineur.') # Si l'utilisateur l'âge de l'utilisateur est
    strictement inférieur à 18 ans, on affiche ce message.
else:
    print('Vous êtes majeur.') # Sinon, on affiche plutôt ce message.
print('Merci, et au revoir !') # Enfin, on dit merci !
```

Le dernier `print` est exécuté après la comparaison.

3. Blocs `elif`

À RETENIR

La clause `elif` permet d'ajouter une autre condition à tester : si la condition `if` est fausse, alors Python teste la condition `elif`. Si la condition `elif` est fausse, Python applique les instructions contenues dans le bloc `else`.

EXEMPLE

Le code ci-dessous est un raffinement de l'exemple précédent.

```
age = int(input('Quel est votre âge ? ')) # On demande l'âge de l'utilisateur.
if age < 18: # On effectue la comparaison avec le nombre 18.
    print('Vous êtes mineur.') # Si l'utilisateur l'âge de l'utilisateur est
    strictement inférieur à 18 ans, on affiche ce message.
elif age < 21: # On effectue la comparaison avec le nombre 21.
    print('Vous êtes majeur en France, mais pas à l\'international.') # Si l'âge de l
    'utilisateur est plus grand que 18, mais moins que 21, on affiche ce message.
else:
    print('Vous avez la majorité internationale !') # Si l'âge de l'utilisateur est
    supérieur ou égal à 21 ans, on affiche ce message.
print('Merci, et au revoir !') # Enfin, on dit merci !
```

V Boucles

À RETENIR

En informatique, une **boucle** est une instruction qui permet de répéter un morceau de code un certain nombre de fois, connu ou non. À ce titre, on dispose principalement de deux types de boucles :

— Les boucles dites **bornées**, qui sont utilisées lorsque l'on veut exécuter un nombre déterminé de fois un même bloc d'instructions.

— Les boucles dites **conditionnelles**, qui sont utilisées pour répéter un bloc d'instructions tant qu'une condition est vérifiée.

1. Boucles bornées

À RETENIR

En Python, pour écrire une boucle bornée, on utilise l'instruction `for`.

EXEMPLE

Le programme ci-dessous est un exemple simple d'utilisation de cette boucle.

```
for i in range(5):
    print('Bonjour')
```

On répète ici 5 fois l'instruction `print('Bonjour')`.

À RETENIR

Le bloc d'instructions à répéter doit être **indenté**, c'est-à-dire décalé de 4 espaces vers la droite.

La variable `i` est l'**indice** de boucle (ou **compteur** de boucle). On peut lui donner le nom que l'on veut (`i`, `n`, `compteur`, `nb`, ...). La valeur de l'indice de boucle est le nombre de fois que l'on a exécuté celle-ci.

EXEMPLE

Dans l'exemple ci-dessus, on affiche 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, puis de nouveau 9 car, à la fin de l'exécution du bloc `for`, la variable i existe toujours et contient la dernière valeur qui lui a été affectée.

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

2. Boucles conditionnelles

À RETENIR

La boucle `for` a l'avantage d'être simple à utiliser, mais elle nécessite de connaître à l'avance (avant d'entrer dans la boucle) le nombre de tours de boucles qui vont être effectués. Dans certaines situations, on souhaiterait demander au programme de sortir de la boucle lorsqu'une certaine condition est réalisée. La boucle `while` permet de répondre à ce besoin.

EXEMPLE

Voici un programme qui permet de demander à l'utilisateur un mot de passe. Tant qu'il n'est pas correct, on redemande :

```
mdpCorrect = 'azerty' # Le mot de passe correct est "azerty".
mdpUtilisateur = '' # On initialise une variable nommée "mdpUtilisateur". Peu importe sa valeur.
while mdpUtilisateur != mdpCorrect: # Tant que "mdpUtilisateur" est différente de "azerty", on va exécuter le bloc ci-dessous.
    mdpUtilisateur = input('Veuillez entrer le mot de passe : ') # On demande à l'utilisateur d'entrer un mot de passe.
print('Vous avez entré le bon mot de passe !') # On est sorti de la boucle while : c'est que l'utilisateur à entré le bon mot de passe.
```

EXEMPLE

Il est également possible de réecrire les boucles `for` en boucles `while`. Par exemple, les deux morceaux de code suivants sont équivalents :

```
for i in range(5):
    print('Bonjour')
```

```
i = 0
while i < 5:
    print('Bonjour')
    i++
```

3. Interruption de boucles

À RETENIR

L'instruction `break` permet d'interrompre l'exécution d'une boucle.

EXEMPLE

Dans le code ci-dessous, on arrête l'exécution de la boucle une fois que la valeur de la variable `nombre` est égale à 5.

```
nombre = 0
for nombre in range (10):
    if nombre == 5:
        print("On va s'arrêter MAINTENANT !")
        break
    print(nombre)
```

À RETENIR

L'instruction `continue` demande au programme de revenir immédiatement au début de la boucle pour effectuer le tour suivant. Cette instruction permet dans certains cas d'améliorer un peu la lisibilité du programme.

EXEMPLE

Dans le code ci-dessous, on passe un tour de boucle une fois que la valeur de la variable `nombre` est égale à 5.

```
nombre = 0
for nombre in range (10):
    if nombre == 5:
        print("On va sauter cette étape !")
        continue
    print(nombre)
```

VI Fonctions

1. Vocabulaire

À RETENIR

Le rôle d'une fonction est d'exécuter un bloc d'instructions, puis éventuellement de renvoyer une **valeur de retour**. L'utilisateur n'a pas à connaître son fonctionnement interne. Afin que le comportement de la fonction puisse s'adapter, on lui fournit généralement un ou plusieurs **paramètres**. On peut ensuite faire appel à cette fonction autant de fois qu'on le souhaite, en donnant son nom et la liste des valeurs des paramètres éventuels.

EXEMPLE

Le code ci-dessous demande deux nombres à l'utilisateur et affiche le plus grand.

```
a = int(input('Entrer un premier nombre : '))
b = int(input('Entrer un deuxième nombre : '))
if a > b:
  print(a)
else:
  print(b)
```

On peut réécrire ce code en utilisant la fonction `max` de Python.

```
a = int(input('Entrer un premier nombre : '))
b = int(input('Entrer un deuxième nombre : '))
print(max(a, b))
```

2. Paramètres

À RETENIR

La définition d'une fonction en Python commence avec le mot-clé `def`. On indique ensuite le nom de nouvelle fonction que l'on souhaite créer. Les parenthèses qui suivent sont obligatoires : elles indiquent qu'il s'agit d'une fonction et non d'une variable ordinaire.
À l'intérieur des parenthèses, on indique la liste des noms des paramètres de la fonction. Les paramètres sont optionnels : on peut créer une fonction sans paramètres (avec des parenthèses vides).

EXEMPLE

```
# Définition de la fonction "direBonjour".
def direBonjour(personne):
print('Bonjour ' + personne + ' !')

# Exécution de la fonction avec le paramètre "mon très cher voisin".
direBonjour('mon très cher voisin')
```

3. Valeur de retour

À RETENIR

Les fonction que l'on a écrites précédemment n'ont pas de valeur de retour. Il est cependant (et en général, préférable), pour une fonction, de renvoyer quelque chose à la fin de son exécution.

Le mot-clé `return` permet de mettre fin à la fonction et/ou de renvoyer une valeur de retour.

EXEMPLE

Voici l'exemple d'une fonction 'plus' qui renvoie la somme de deux nombres `a` et `b` :

```
def plus(a, b):
  resultat = a + b
  return resultat
```

On aurait aussi pu l'écrire de manière plus courte :

```
def plus(a, b):
  return a + b
```

4. Fonctions récursives

À RETENIR

Une fonction peut faire appel à elle-même dans sa propre définition. On parle alors de fonction **récursive**.

EXEMPLE

Voici un exemple de fonction qui permet de calculer la somme $1+2+3+\cdots+n$ pour un entier n donné.

```
def somme(n):
  if n == 0:
    return 0
  return n + somme(n-1)
```