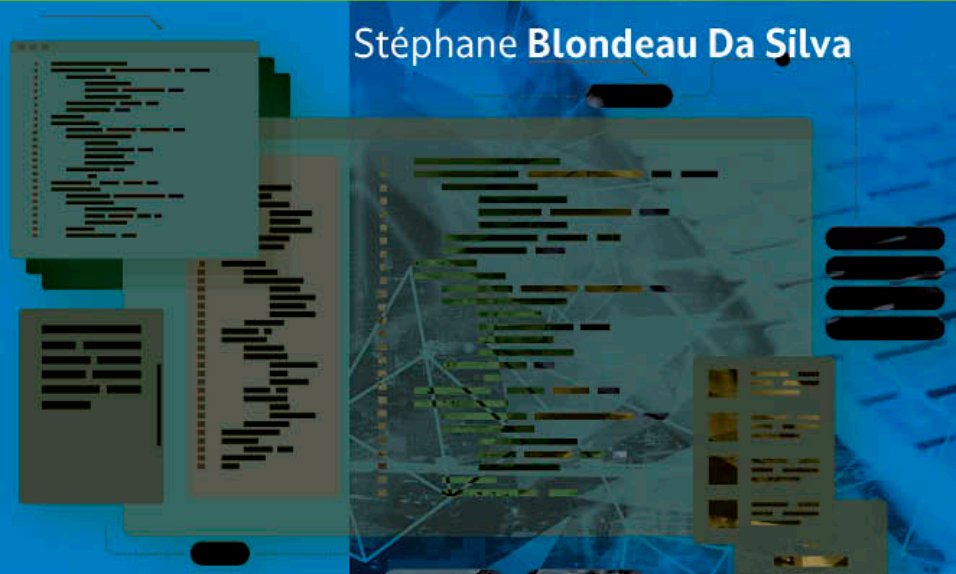


Informatique

BCPST1

Cours et exercices corrigés et commentés

Stéphane **Blondeau Da Silva**



ellipses

Informatique

BCPST1



Informatique

BCPST1

Cours et exercices corrigés
et commentés

Stéphane **Blondeau Da Silva**

Professeur de Mathématiques-Informatique en BCPST1

ellipses

Conception graphique couverture : Éric VIGIER

ISBN 9782340-099562

Dépôt légal : janvier 2025

©Ellipses Édition Marketing S.A., 2025
8/10 rue la Quintinie 75015 Paris



Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5.2° et 3°a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective », et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit constituerait une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

www.editions-ellipses.fr

Avant-propos

Cet ouvrage s'adresse aux étudiants de première année de Classe Préparatoire aux Grandes Écoles Biologie, Chimie, Physique et Sciences de la Terre (CPGE BCPST1) désireux d'asseoir de solides bases en Informatique lors de leur première année de classe préparatoire, bases qu'ils sauront faire fructifier lors de leur deuxième année les menant aux concours. À travers un certain nombre de sujets et d'exercices issus des oraux des concours d'Agro-Véto et G2E, il permet, en outre, dès la première année, de se confronter aux épreuves de ces deux concours et ainsi d'anticiper le travail de préparation de seconde année. Voir que des exercices proposés lors des épreuves orales leur sont accessibles dès la première année participera à rendre les épreuves qui les attendent moins impressionnantes.

Cet ouvrage se veut un catalogue exhaustif des notions inscrites au programme de la classe préparatoire BCPST, définies par le *Bulletin officiel* du 1er juillet 2021, et est donc d'abord un cours d'Informatique. Il respecte l'ordre défini par ce programme, ce qui induit une douce progressivité de la difficulté des notions rencontrées. Les 100 premières pages, en particulier, sont principalement des rappels des notions acquises au lycée, la nécessité d'homogénéiser les connaissances des étudiants en Informatique à leur entrée en classe préparatoire passant par un réexamen complet des bases de la discipline.

Chaque chapitre, outre les définitions, syntaxes, caractéristiques de toutes les notions mises en jeu, est émaillé d'une multitude d'exemples, commentés, illustrant chacun des points abordés. Il se clôt systématiquement par une série d'exercices progressifs, tous corrigés et commentés, permettant à l'étudiant de tester la bonne acquisition des éléments importants du programme. Des conseils et astuces permettent également à ce dernier de gagner en autonomie dans la construction de leurs programmes et algorithmes. Les exercices sont classés par niveau de difficulté :

- ♣ : exercice d'application directe du cours ;
- ♣♣ : exercice faisant intervenir un nombre restreint de notions distinctes du programme ;

- ♣♣♣♣ : exercice difficile exigeant un recul important (ne pas réussir ces exercices d'emblée n'hypothèque évidemment en rien les chances de réussite des étudiants au concours, c'est une évidence).

Cet ouvrage se révèle ainsi un outil complémentaire précieux pour réussir ses deux années d'Informatique en classe préparatoire BCPST.

Table des matières

1 - Le langage Python **1**

I – Les algorithmes 1

I.1 Un peu d'histoire 1

I.2 Définition et caractéristiques 2

I.3 Langage de programmation : le choix Python 3

II – Environnement de travail 3

II.1 La console 4

II.2 L'éditeur 4

III – Variables 6

III.1 Définition 6

III.2 Types de variables 7

III.3 Opérateurs 9

IV – Expression et instruction 12

IV.1 Expression 12

IV.2 Instructions 12

IV.3 Exercices 22

V – Les fonctions 27

V.1 Définition, utilisation et syntaxe 27

V.2 Caractéristiques principales 28

V.3 Exercices 32

VI – Les listes 39

VI.1 Définition, syntaxe et création 39

VI.2 Utilisation des listes 41

VI.3 Quelques utiles compléments 46

VI.4 Exercices 47

VII – Utilisation de modules et bibliothèques 67
VII.1 Découverte du fonctionnement d'un module : le module `math` 67
VII.2 Le module `statistics` 68
VII.3 Le module `random` 69
VII.4 Le module `pyplot` de la bibliothèque `matplotlib` 69
VII.5 La bibliothèque `NumPy` 74
VII.6 Exercices 84
VIII –Erreurs et exceptions 96
VIII.1 Gestion des bogues 96
VIII.2 Liste des principales exceptions 99

2 - Lecture et écriture dans un fichier texte **101**
I – Structure d'un script agissant sur un fichier texte 101
II – Lecture des lignes d'un fichier 103
III – Écriture de données dans un fichier 106
IV – Exercices 107
IV.1 Énoncés 107
IV.2 Correction 109

3 - Fonctions récursives **115**
I – Fonctions récursives mathématiques 115
II – Fonctions récursives et fractales 117
III – Exercices 118
III.1 Énoncés 118
III.2 Correction 121

4 - Tris **131**
I – Comment trier une liste ? 131
I.1 Fonctions de tris 132
I.2 Trier avec une fonction clé 132
II – Algorithmes de tris 133
II.1 Tri par insertion 134
II.2 Tri par sélection 135
II.3 Tri par comptage 137
III – Exercices 138

III.1 Énoncés . . . 138
III.2 Correction . . . 139

5 - Recherche dichotomique 145
I – Solution approchée d'une équation . . . 145
I.1 Principe général . . . 145
I.2 Algorithme . . . 146
II – Recherche dichotomique dans un tableau trié . . . 147
II.1 Principe général . . . 148
II.2 Algorithme . . . 149
III – Exercices . . . 150
III.1 Énoncés . . . 150
III.2 Correction . . . 152

6 - Matrices de pixels et images 157
I – Représentation informatique d'une image . . . 157
II – Les outils Python pour le traitement d'images . . . 159
II.1 La fonction `open` . . . 159
II.2 La fonction `fromarray` et la méthode `save` . . . 161
II.3 Les fonctions `zeros_like` et `copy` . . . 162
III – Traitement d'une image pixel par pixel . . . 162
III.1 Obtention d'une image en niveaux de gris . . . 162
III.2 Obtention du négatif d'une image . . . 164
IV – Transformation géométrique d'une image . . . 165
IV.1 Obtention d'une image miroir . . . 165
IV.2 Rotation d'une image . . . 166
IV.3 Agrandissement ou réduction d'une image . . . 168
V – Traitement d'une image par convolution . . . 169
V.1 Principe général . . . 170
V.2 Le script . . . 171
V.3 Floutage d'une image . . . 173
V.4 Amélioration de la netteté . . . 174
V.5 Détection de contours . . . 174
VI – Exercices . . . 175
VI.1 Énoncés . . . 175

VI.2 Correction 176

7 - Objets mutables : dictionnaire **181**

I – Identifiants, valeurs et objets mutables 181

I.1 Identifiants et valeurs 181

I.2 Objets mutables et non mutables 182

I.3 Exercices 188

II – Dictionnaires 190

II.1 Définition, syntaxe et création 191

II.2 Utilisation d'un dictionnaire 192

II.3 Exercices 194

8 - Les graphes **199**

I – Intérêts et applications des graphes 200

II – Vocabulaire et définitions 201

II.1 Les graphes non orientés 201

II.2 Les graphes orientés 202

II.3 Les graphes pondérés 203

III – Représentations informatiques des graphes 204

III.1 Matrices d'adjacence 204

III.2 Listes d'adjacence 205

IV – Parcours en largeur d'un graphe 207

IV.1 Principe du parcours en largeur 207

IV.2 Algorithme 209

V – Exercices 210

V.1 Énoncés 210

V.2 Correction 212

9 - Les bases de données **221**

I – Le modèle relationnel 222

I.1 Vocabulaire 222

I.2 Clé primaire et clé étrangère 224

II – Requêtes SQL 225

II.1 Définition d'une base de données 226

II.2 Le logiciel SQLite 227

II.3 Les requêtes 227
II.4 Requêtes avec jointure 231
II.5 Fonctions d'agrégation 232
III – Exercices 235
III.1 Énoncés 235
III.2 Correction 237

10 - Méthodes numériques **241**
I – Méthode des rectangles 241
I.1 Le principe 241
I.2 L'algorithme 243
I.3 Exercices 243
II – Probabilité 248
II.1 Simulation de variables aléatoires 248
II.2 Estimations d'une probabilité, de l'espérance et de la variance 252
II.3 Exercices 256



Chapitre 1

Le langage Python

I – Les algorithmes

I.1 Un peu d'histoire

Le terme **algorithme** trouve ses racines dans le nom du mathématicien persan du IX^{e} siècle, Muhammad ibn Musa al-Khwarizmi. Al-Khwarizmi a écrit plusieurs ouvrages sur les mathématiques, et son travail sur la résolution systématique des équations linéaires et quadratiques a introduit des méthodes qui ont jeté les bases de l'algèbre. Le mot algorithme dérive de la latinisation de son nom *Algoritmi*.

Cependant, les algorithmes en tant que tels existaient bien avant al-Khwarizmi. Par exemple, l'algorithme d'Euclide pour calculer le plus grand commun diviseur (PGCD) de deux nombres entiers remonte à l'Antiquité grecque et est attribué à Euclide. Les algorithmes ont évolué avec le développement des mathématiques et de l'informatique, et sont devenus cruciaux avec l'avènement des ordinateurs au XX^{e} siècle.

On les retrouve dans des domaines aussi divers qu'en Santé et Médecine (diagnostic médical ou analyse génomique), dans la Finance (tranding algorithmique ou détection de fraude), dans le Transporst et la Logistique (planification des itinéraires ou gestion des stocks), dans le Commerce en ligne (recommandations de produits ou optimisation des prix), dans le domaine de la Sécurité (reconnaissance faciale ou cyber-sécurité), de l'Éducation (apprentissage adaptatif ou détection de plagiats), dans l'Industrie (maintenance prédictive ou optimisation des processus de production), dans le Divertissement (streaming vidéo et musical ou jeux vidéo), dans l'Environnement (prévisions météorologiques ou gestion de l'énergie), etc. Ces exemples montrent à quel point les algorithmes sont omniprésents et essentiels dans de nombreux secteurs de la vie quotidienne et professionnelle.

I.2 Définition et caractéristiques

Un **algorithme** est une suite finie et ordonnée d'instructions permettant de résoudre un problème ou d'accomplir une tâche spécifique. Chaque instruction doit être claire et sans ambiguïté, et l'exécution de l'algorithme doit aboutir à un résultat attendu.

Ils se caractérisent par :

- sa **finitude** : un algorithme doit se terminer après un nombre fini d'étapes ;
- sa **définitude** : chaque étape de l'algorithme doit être précisément définie, sans ambiguïté ;
- ses **entrées** : un algorithme peut avoir zéro ou plusieurs entrées (données initiales).
- ses **sorties** : un algorithme produit au moins une sortie (résultat) ;
- son **efficacité** : l'algorithme doit être efficace en termes de temps et de ressources utilisées.

Exemple 1. *Voici l'algorithme d'Euclide :*

Entrées *: Deux nombres entiers positifs,* A *et* B*, où* $A \geqslant B$*.*
Variables *:* a*,* b *et* r *sont des entiers naturels.*
Initialisation *:* Affecter A *et* B *aux variables* a *et* b*.*
Traitement *:* Tant que b *est différent de* 0*,* Faire
    Calculer *le reste de la division de* a *par* b *et l'*affecter *à la variable* r*.*
    Affecter *à* a *la valeur de* b*.*
    Affecter *à* b *la valeur de* r*.*
Sortie *: Le plus grand commun diviseur est la valeur de* a*.*

Déterminons le PGCD de 48 *et* 18 *en observant l'évolution des variables :*

Variables	*a*	*b*	*r*
Initialisation	48	18	×
Étape 1	18	12	12
Étape 2	12	6	6
Étape 3	6	0	0

Le PGCD de 48 *et* 18 *est* 6*, c'est-à-dire la dernière valeur de* a*.*

I.3 Langage de programmation : le choix Python

Afin de mettre en œuvre un algorithme sur un ordinateur, un **langage de programmation** doit être choisi. Ce dernier, en effet, compte un ensemble d'instructions que l'ordinateur peut comprendre et exécuter.

Le langage **Python** possède un certain nombre de qualités :

- sa lisibilité et sa simplicité : cela rend les algorithmes plus faciles à comprendre et à écrire ;
- l'existence d'une vaste bibliothèque standard qui inclut des modules pour diverses tâches algorithmiques, évitant à l'utilisateur de réimplémenter certaines fonctionnalités ;
- l'existence d'une grande communauté, très active, offrant une abondance de ressources, de tutoriels et de supports ;
- son interopérabilité : Python peut, en effet, facilement s'intégrer dans d'autres langages et technologies, facilitant l'utilisation d'algorithmes dans divers contextes.

L'ensemble de ces raisons font de Python un choix pertinent de langage de programmation.

II – Environnement de travail

Python, placé sous licence libre, a été conçu par Guido van Rossum au Centre de Recherche pour les Mathématiques et l'Informatique (CWI) aux Pays-Bas. Ce dernier commence à travailler sur Python en tant que projet de loisir pendant les vacances de Noël et, finalement, Python est devenu un des langages de programmation les plus influents et largement utilisés au monde.

Il existe plusieurs environnements de travail pour utiliser Python ; IDLE et Spyder en font partie. Le premier est disponible sur le site officiel de Python : `https://www.python.org`, le second l'est à l'adresse suivante : `https://spyder-ide.org/`. Ils se composent tous deux principalement :

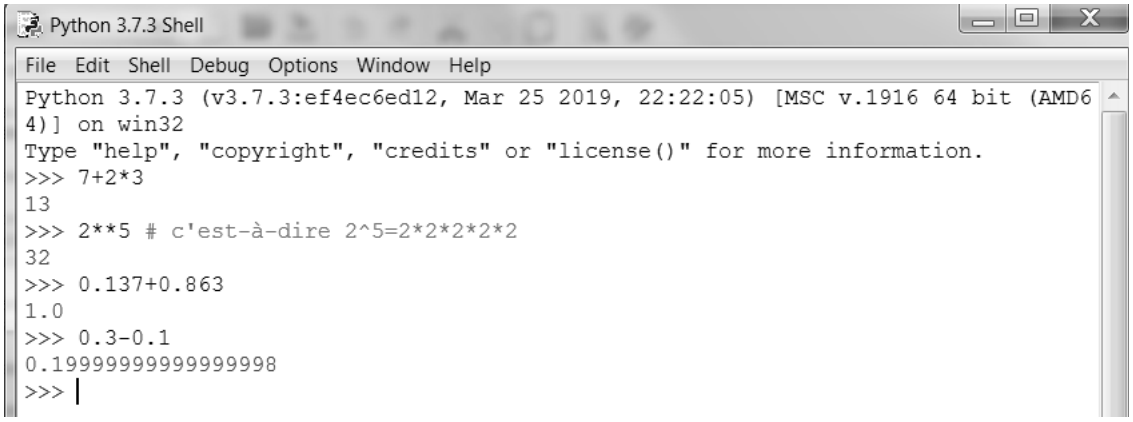
- d'un **interpréteur** aussi appelé **console** ;
- d'un **éditeur**.

II.1 La console

La console Python est un environnement de programmation interactif où on saisit des commandes Python et on voit immédiatement les résultats. C'est un outil pratique pour tester des fragments de code, exécuter des commandes, et expérimenter avec le langage Python en temps réel.

On entre une instruction après le « `In[1]:` » de l'invite de commande dans Spyder (ou le triple symbole « `>>>` » dans IDLE), puis l'interpréteur l'exécute.

Exemple 2. *La console d'IDLE :*



On remarque que Python rencontre parfois quelques difficultés avec les nombres décimaux !

Exemple 3. *La console de Spyder :*

```
In[1] : 39/5  # division décimale
Out[1] : 7.8
In[2] : 39//5  # quotient de la division euclidienne
Out[2] : 7
In[2] : 39%5  # reste de la division euclidienne
Out[2] : 4
```

II.2 L'éditeur

L'éditeur Python sert à écrire et modifier des scripts et programmes en langage Python. Il fournit des fonctionnalités qui facilitent le développement de code, comme la coloration syntaxique, l'indentation automatique, et la complétion de code. Ils incluent aussi des outils de débogage et d'exécution de code.

Le contenu de l'éditeur, contrairement à celui de l'interpréteur, est destiné à être sauvegardé, sous la forme d'un fichier au format *.py*, c'est-à-dire d'un **script** Python. Il s'agit ainsi souvent de programmes plus longs et, plus seulement de simples instructions. Une fois sauvegardé, le script a vocation à être exécuté dans l'interpréteur.

Exemple 4. *Un script, pouvant être sauvegardé, dans l'éditeur IDLE :*

I1I.py - D:/Ellipse/I1I.py (3.7.3)

File Edit Format Run Options Window Help

```
# Résolution d'équations du second degré à coefficients réels du type
# ax^2+bx+c=0 où a est différent de 0

a=float(input("Quelle est la valeur de a ?"))
b=float(input("Quelle est la valeur de b ?"))
c=float(input("Quelle est la valeur de c ?"))

delta=b**2-4*a*c # calcul du discriminant

if delta>0: # 1er cas : discriminant strictement positif
    x1=(-b-delta**0.5)/(2*a)
    x2=(-b+delta**0.5)/(2*a)
    print("Il y a deux solutions distinctes : {} et {}.".format(x1,x2))

elif delta==0: # 2e cas : discriminant nul
    x0=-b/(2*a)
    print("Il y a une unique solution : {}.".format(x0))

else: # 3e cas : discriminant strictement négatif
    x1c=(-b-delta**0.5j)/(2*a) # racines complexes conjuguées où...
    x2c=(-b+delta**0.5j)/(2*a) # ... j est le nombre complexe i
    print("Il y a deux solutions distinctes : {} et {}.".format(x1c,x2c))
```

Dans la console IDLE on obtient :

Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD6
4)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========================== RESTART: D:/Ellipse/I1I.py ==========================
Quelle est la valeur de a ?4
Quelle est la valeur de b ?0
Quelle est la valeur de c ?-1
Il y a deux solutions distinctes : -0.5 et 0.5.
>>>
========================== RESTART: D:/Ellipse/I1I.py ==========================
Quelle est la valeur de a ?1
Quelle est la valeur de b ?-2
Quelle est la valeur de c ?1
Il y a une unique solution : 1.0.
>>>
========================== RESTART: D:/Ellipse/I1I.py ==========================
Quelle est la valeur de a ?1
Quelle est la valeur de b ?1
Quelle est la valeur de c ?1
Il y a deux solutions distinctes : (-0.588648892694378-0.054266503388404386j) et
 (-0.4113511073056219+0.054266503388404386j).
```

Le contenu de la console, lui, ne peut être sauvegardé !

Remarque 1. *Les **commentaires**, présents après l'octothorpe #, servent à améliorer la lisibilité du script, en en expliquant le code. Tout ce qui suit ce symbole, et ce jusqu'au changement de ligne, est ignoré par l'interpréteur.*

III – Variables

III.1 Définition

Une **variable** est un espace mémoire élémentaire qui associe un nom (**l'identifiant**) à une valeur. Elle peut être modifiée pendant l'exécution d'un programme. Les variables permettent aux programmes de manipuler et de gérer les données de manière flexible et dynamique.

Une variable se caractérise par :

- son **nom** : il est unique et signifiant, qui sert d'identifiant ;
- son **type** : il est unique et détermine les valeurs que peuvent prendre la variable et les opérations qui peuvent lui être appliquées ;
- sa **valeur** : il s'agit de l'information stockée dans la variable, elle peut changer au cours de l'exécution du programme ;
- sa **portée** : elle détermine jusqu'où l'on accéder à cette variable. Elle peut être de **portée locale** (variable définie dans une fonction uniquement) ou de **portée globale** (variable accessible dans tout le programme).

En Python, nul besoin de **déclaration** de variable : la variable est implicitement déclarée lors de sa première **affectation**, c'est-à-dire lorsqu'une valeur (et donc un type) lui est donnée.

Exemple 5.

```
In[1] : dep=69  # affectation qui crée la variable dont le nom est dep
In[2] : dep  # instruction pour connaître la valeur de dep...
Out[2] : 69  # ... qui est 69 ; son type est, bien sûr : entier
```

Remarque 2. *Le choix du nom d'une variable est loin d'être anodin : il doit en effet renseigner sur la nature de la variable dont il est l'identifiant. Une variable qui est utilisée pour effectuer un décompte sera ainsi classiquement nommé* `compteur`.

Notons, pêle-mêle, que Python est sensible à la casse (`Poisson` et `poisson` sont deux variables distinctes) et que certains noms sont déjà réservés à des fonctions Python (la coloration syntaxique permettant d'éviter cet écueil).

Une variable peut être supprimée avec la fonction `del` de la façon suivante :

Exemple 6.

```
In[1] : ASSE=42  # création de la variable ASSE
In[2] : ASSE
Out[2] : 42
In[3] : del(ASSE)  # suppression de ASSE (autre syntaxe : del ASSE)
In[4] : ASSE  # quelle est la valeur de ASSE ?
Out[4] : NameError: name 'ASSE' is not defined  # logique !
```

III.2 Types de variables

Le **type** d'une variable définit la nature des données que cette variable stocke et les opérations qui peuvent être effectuées sur ces données.

Chaque variable Python possède un unique type, mais celui-ci peut changer au gré des affectations successives. On parle de **typage dynamique**. Dans d'autres langages de programmation, à **typage statique**, il est nécessaire de déclarer explicitement le type de la variable avant de l'utiliser, ce type demeurant par la suite permanent. La flexibilité accrue et la simplification des programmes qu'autorise le typage dynamique en Python, demande, ainsi, en contrepartie, une réelle vigilance sur le type des variables utilisées.

Voici quelques types utilisés en Python :

Type	Nature de la variable	Exemple
`int`	entier	-169
`float`	flottant (un nombre réel)	2.718
`str`	string (chaîne de caractères)	"Mme Petit"
`bool`	booléen	True
`tuple`	n-uplet	$(3, 5, 17, 257, 65537)$
`list`	liste	["Nini", 24, 11, "IMBE"]

Remarques 3. ◇ *Le type `int` est stocké en valeur exacte contrairement au type `float` (on voit l'erreur liée au caractère approximatif dans l'Exemple 2). La présence du point (`.`) permet de différencier les variables de type `int` et `float` : `5` est un entier tandis que `5.0` est un flottant.*

- *Le type `str` doit être noté entre quotes simples (’), doubles (”) ou même triples (ce dernier cas autorisant les sauts de ligne).*
- *Le type `bool` ne possède que deux valeurs : `True` ou `False`.*
- *Les types `list` et `tuple`, on le voit, contiennent des objets de type divers. La différence fondamentale entre les listes et les tuples est leur **mutabilité** : les listes sont modifiables (on dit **mutables**), ce qui signifie qu’il est possible d’ajouter, de supprimer ou de modifier des éléments après la création de la liste, ce qui n’est pas le cas avec les tuples !*
- *La fonction `type` prend en paramètre une variable et renvoie son type.*
- *Les fonctions `int`, `float`, `bool`, `str` permettent de modifier le type d’une variable.*
- *Le séparateur décimal est, on l’a vu, le point (.). La virgule (,), elle, permet de séparer les différents objets des listes ou des tuples. Le point-virgule (;), quant à lui, permet d’écrire plusieurs instructions sur une même ligne.*
- *La **classe** d’un objet en Python définit la structure et le comportement de cet objet, spécifiant ses attributs, c’est-à-dire ses propriétés, et ses méthodes, c’est-à-dire ses fonctions associées.*

Exemple 7.

```
In[1] : MJ=23
In[2] : type(MJ)
Out[2] : <class 'int'>  # 23 est un entier, sans surprise !
In[3] : Sot=2.45
In[4] : type(Sot)
Out[4] : <class 'float'>  # 2.45 est un flottant
In[5] : float(MJ)  # on convertit la valeur 23 en flottant...
Out[5] : 23.0  # ... 23 est devenu 23.0 ; ⚠ MJ est inchangée
In[6] : Sot=int(Sot)  # conversion en entier et affectation
In[7] : Sot  # quelle est la valeur de Sot ?
Out[7] : 2  # un entier évidemment (la troncature de 2.45) !
In[8] : type("ficelle")
Out[8] : <class 'str'>  # une chaîne de caractères bien sûr!
```

À travers cet exemple, on voit que le type de la variable `Sot` a changé : typage dynamique.

III.3 Opérateurs

Un **opérateur** est un symbole ou un mot clé qui indique une opération à effectuer sur des valeurs ou des variables.

III.3.1 Opérateurs arithmétiques

Opérateur	Opérations
`+, *, -, /`	addition, multiplication, soustraction, division
`**`	exponentiation
`//`	quotient de la division euclidienne
`%`	reste de la division euclidienne

Exemple 8.

```
In[1] : 2.7+2.3  # deux flottants au départ...
Out[1] : 5.0  #... le résultat est un flottant !
In[2] : 2*2.5  # un entier et un flottant...
Out[2] : 5.0  # ... le résultat est encore un flottant !
In[3] : mer1=(3,7,31,127);mer2=(8191,131071)  # deux affectations !
In[4] : mer1+mer2
Out[4] : (3,7,31,127,8191,131071)
In[5] : "Lily "+"est"+" tatouée"
Out[5] : 'Lily est tatouée'  # concaténation de chaînes de caractères
In[6] : 2*"Gé";5*(1,2)
Out[6] : 'GéGé'
(1,2,1,2,1,2,1,2,1,2)
In[7] : [0,0,0]-[-7,-5,-3]  # différence entre deux listes...
TypeError: unsupported operand type(s) for -: 'list' and 'list'
In[8] : 3+[-7,-5,-3]  # somme entre un entier et une liste...
TypeError: unsupported operand type(s) for +: 'int' and 'list'
In[9] : 91/7  # division décimale...
Out[9] : 13.0  # ... le résultat est un flottant
In[10] : (-93)//7;(-93)%7  # quotient et reste division euclidienne...
Out[10] : -14  # eh oui, -93 = -14 × 7 + 5, en effet...
5  # ... le reste est compris entre 0 inclus et 7 exclu !
```

Remarques 4.
- ◇ *La somme de deux entiers donne un entier, de deux flottants donne un flottant, jusque là rien de bien surprenant.*
- ◇ *La somme de deux listes, de deux tuples, de deux chaînes de caractères renvoie respectivement une liste, un tuple et une chaîne de caractères. On observe le phénomène de* ***concaténation****, les listes, tuples et chaînes de caractères étant mis bout à bout.*
- ◇ *Le produit d'un entier naturel par une liste, un tuple et une chaîne de caractères renvoie respectivement une liste, un tuple et une chaîne de caractères dupliquée.*
- ◇ *Les opérations entre deux variables de types différents ne sont généralement pas possibles, à l'exception bien sûr, de variables de type* `int` *et* `float`*. Un flottant est dès lors renvoyé.*
- ◇ *La division décimale renvoie toujours un flottant, la division euclidienne un entier.*

III.3.2 Opérateurs de comparaison

Opérateur	Test effectué
`a==b`	a est-il égal à b ?
`a!=b`	a est-il différent de b ?
`a>b`	a est-il strictement supérieur à b ?
`a<b`	a est-il strictement inférieur à b ?
`a>=b`	a est-il supérieur ou égal à b
`a<=b`	a est-il inférieur ou égal à b

Ces tests sont évalués en booléens comme on le voit ci-dessous :

Exemple 9.

```
In[1] : 5//2==2.0  # un entier et un flottant...
Out[1] : True  # ... le résultat est un booléen
In[2] : 2<=5/2<7  # cette syntaxe est valide...
Out[2] : True  #... et l'inégalité est vraie
Int[3] : tup=(-3<-5,"test")  # un booléen dans un tuple
Int[4] : tup  # quelle est la valeur de tup ?
Out[4] : (False,'test')
Int[5] : "Biologie">"Informatique"  # Nulle inquiétude à avoir...
Out[5] : False  # ... c'est l'ordre lexicographique qui prévaut !
```

```
In[6] : (1,1.2,"vulcain")<=(1,-2,"citron",2<3)  # comparaisons...
Out[6] : False  # ... élément par élément ; 1.2 > −2 : False
```

Remarques 5.
- ◇ *On peut comparer des entiers entre eux, des flottants entre eux, un entier avec un flottant ou même des chaînes de caractères entres elles ; dans ce dernier cas, c'est l'ordre lexicographique qui prévaut.*
- ◇ *Des variables de types différents ne peuvent, en général, pas être comparées.*
- ◇ *Les listes (et les tuples) peuvent parfois être comparées entre elles. Dans ce cas, les éléments sont comparés deux à deux dans l'ordre de leur indexation (à la condition qu'ils soient de même type !). Si les deux premiers éléments sont égaux, on passe au suivant et ainsi de suite...*

III.3.3 Opérateurs logiques

Les opérateurs logiques agissent sur des variables de type `bool`. Les voici :

Opérateur	Test effectué
`A and B`	`True` si `A` **et** `B` valent `True`
`A or B`	`True` si `A` **ou** `B` valent `True`
`not A`	`True` si `A` vaut `False`

Exemple 10.

```
In[1] : (Périgueux,Cahors)=(24,46)  # affectation d'un tuple
In[2] : Périgueux<13 or Cahors==46
Out[2] : True
In[3] : not Cahors>Périgueux
Out[3] : False
```

III.3.4 Opérateurs d'appartenance

Les opérateurs d'appartenance `in` et `not in` donnent des informations sur la présence ou l'absence d'un objet dans une collection (c'est-à-dire une chaîne de caractères, une liste ou un tuple).

Opérateur	Test effectué
`obj in C`	`True` si `obj` appartient à `C`
`obj not in C`	`True` si `obj` n'appartient pas à `C`

Exemple 11.

```
In[1] : 3 in (2,3.0,"Scutigère véloce")
Out[1] : True  # 3 est un élément du tuple
In[2] : "véloce" in (2,3.0,"Scutigère véloce")
Out[2] : False  # "véloce" n'est pas un élément du tuple
Int[3] : "aba" not in "fabacées"
Out[3] : False
```

Remarque 6. *Les éléments d'une chaîne de caractères sont les différents caractères de la chaîne.*

IV – Expression et instruction

IV.1 Expression

Une **expression** est une combinaison de valeurs, de variables, d'opérateurs et de fonctions qui sont **évaluées** pour produire une valeur. Les expressions sont des éléments fondamentaux du langage de programmation, car elles permettent de réaliser des calculs, d'évaluer des conditions et de manipuler des données.

Voici quelques exemples d'expressions :

Exemple 12.

```
In[1] : (len("bloot")+2)*2  # len renvoie la longueur (lenght)...
Out[1] : 14  # ... d'une chaîne de caractères
In[2] : "antennates"=="mandibulates" and True
Out[2] : False
```

IV.2 Instructions

Une **instruction** (ou **commande**) est une unité de code exécutable qui indique à l'interpréteur Python d'effectuer une action spécifique. Contrairement à une expression qui produit une valeur, une instruction peut exécuter des actions, modifier l'état du programme, contrôler le flux d'exécution (la séquence dans laquelle les instructions d'un programme sont lues et exécutées par l'interpréteur), ou appeler des fonctions. Les instructions sont les composants de base d'un programme Python ; elles permettent d'orga-

niser le comportement du programme. Le catalogue des principales instructions va être dressé dans cette partie.

IV.2.1 Affectation

Une **affectation** est l'opération qui consiste à attribuer une valeur à une variable. Cette opération fondamentale permet de stocker et de manipuler des données dans un programme.

Syntaxe 1. ***Affectation***

Elle se compose :

- ⋆ *du **nom** de la variable suivi du caractère « = » ;*
- ⋆ *d'une **expression** dont la valeur sera assignée à la variable.*

```
nom_variable=valeur_variable
```

Voici quelques exemples d'affectations :

Exemple 13.

```
In[1] : salle=410
In[2] : boule= (not salle>=400)
In[3] : bonheur=3/2
In[4] : salle, boule, bonheur  # valeurs des trois variables ?
Out[4] : (410, False, 1.5)  # expressions évaluées deviennent valeurs
```

Remarques 7.

- ◇ ⚠ *La comparaison, avec la syntaxe* `==`*, et l'affectation, avec la syntaxe* `=`*, ne doivent pas être confondues ! L'affectation est une instruction (qui est exécutée), tandis que la comparaison est une expression (qui est évaluée en* `True` *ou* `False`*).*
- ◇ *Il est possible d'effectuer des **affectations simultanées** ou des **affectations multiples**. Voici un exemple de chaque :*

```
In[1] : marjo,app_pi,cheval="Blaise Pascal",round(22/7,2),True
In[2] : app_pi  # 3 affectations simultanément effectuées
Out[2] : 3.14  # round renvoie l'arrondi au centième : paramètre 2
Int[3] : var1=var2=7  # affectations multiples, les variables valent 7
```

Les opérateurs d'affectation permettent de combiner des opérations arithmétiques avec l'affectation de valeurs aux variables. Ils sont largement utilisés pour simplifier et clarifier le code.

Opérateur	Équivalent à...
`a+=b`	a=a+b
`a-=b`	a=a-b
`a*=b`	a=a*b
`a/=b`	a=a/b
`a//=b`	a=a//b
`a%=b`	a=a%b
`a**=b`	a=a**b

Exemple 14.

```
In[1] : a=702;a/=9;a  # deux affectations successives
Out[1] : 78.0
Int[2] : a%=9;a  # une nouvelle affectation
Out[2] : 6.0
```

IV.2.2 Instruction de sortie : `print`

La fonction `print` est utilisée pour afficher des messages, des valeurs de variables, des résultats de calculs, etc., sur la sortie standard, qui est généralement l'écran de la console. C'est l'une des fonctions les plus couramment utilisées en Python, particulièrement utile pour le débogage, l'interaction avec l'utilisateur et l'affichage de résultats.

Syntaxe 2. *`print`*

Elle se décline de la façon suivante :

- ⋆ *le* ***mot clé*** *`print` suivi d'une parenthèse ouvrante ;*
- ⋆ *la liste des paramètres :*
  - ∗ *un ou plusieurs objets à afficher. Ils peuvent être de n'importe quel type (chaînes, nombres, listes, etc.). Les objets sont convertis en chaînes de caractères avant d'être affichés ;*
  - ∗ *une chaîne de caractères qui sépare les objets (par défaut, un espace : `sep=' '`) ;*

- ∗ *une chaîne de caractères ajoutée à la fin de l'affichage (par défaut, un saut de ligne (newline) :* `end='\n'`*).*

⋆ *une parenthèse fermante.*

```
print(objets, sep=' ', end='\n')
```

Exemple 15.

```
In[1] : print("Linotte a ",1," an !")  # trois objets s'affichent
Linotte a 1 an !
In[2] : print(8,7,sep="*",end="=");print(56)  # deux affichages...
8*7=56  # ... avec un « = » au lieu du saut de ligne
In[3] : note=19
In[4] : print("{} a eu {} au bac en SVT".format("Léontine",note));
print("Bravo !")  # deux affichages...
Léontine a eu 19 au bac en SVT  # ... avec saut de ligne
Bravo !
```

Remarques 8. ◇ *La fonction* `print` *ne renvoie rien (dans la console, absence de sortie :* Out[ ] *), elle se contente d'afficher des objets. Cet affichage ne peut pas être réexploité ni affecté à une variable !*

◇ *La* ***méthode*** *(c'est-à-dire une fonction associée strictement à une classe d'objets)* `format` *permet d'insérer la valeur des variables mises en paramètre dans la chaîne de caractères à la place des accolades* `{}`*.* `format` *est une méthode de la classe* `str`*.*

IV.2.3 Instruction d'entrée clavier : `input`

L'instruction `input` est utilisée pour lire une ligne de texte saisie par l'utilisateur à partir de l'entrée standard, généralement le clavier. Elle est couramment utilisée pour permettre l'interaction avec l'utilisateur en lui demandant de fournir des informations ou des choix pendant l'exécution d'un programme.

Syntaxe 3. `input`

Elle se décline de la façon suivante :

⋆ *le* ***mot clé*** `input` *suivi d'une parenthèse ouvrante ;*