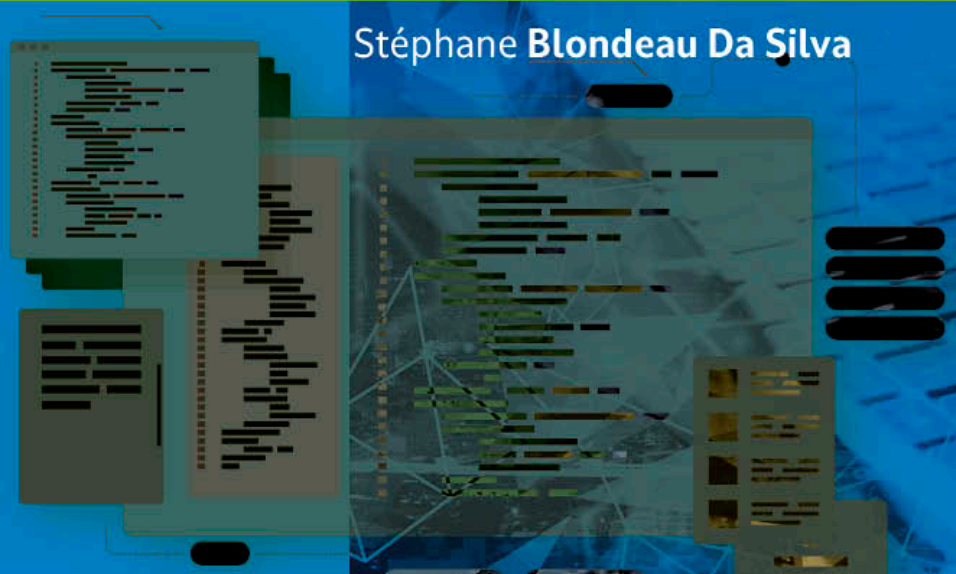
Informatique

BCPST1

Cours et exercices corrigés et commentés

Stéphane **Blondeau Da Silva**



ellipses

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 :

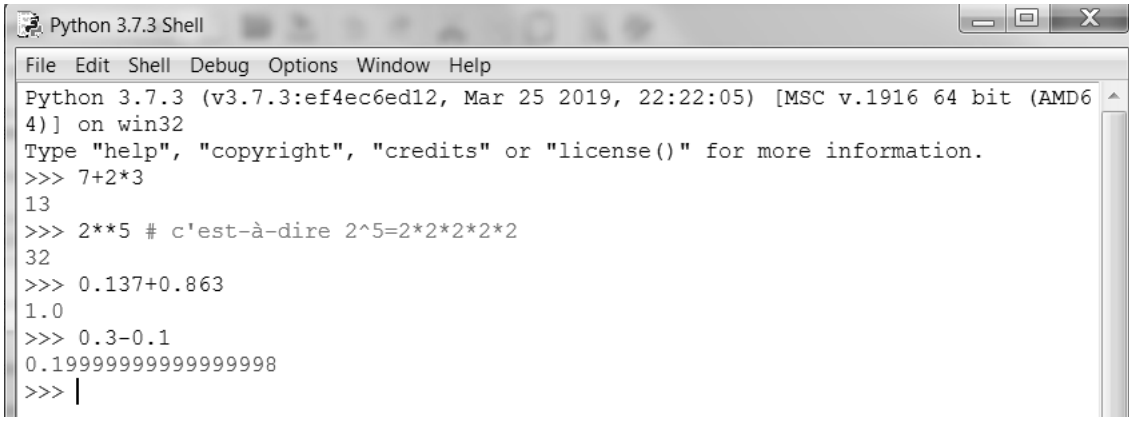
- d'un **interpréteur** aussi appelé **console** ;
- d'un **éditeur**.

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.