Sophie Schüpp

Licence

Python en licence

Algorithme de base et algorithme scientifique



ellipses

Références sciences

Python en licence

Algorithme de base et algorithme scientifique

Sophie Schüpp

ellipses

Collection Références sciences

dirigée par Paul de Laboulaye
paul.delaboulaye@editions-ellipses.fr

Retrouvez tous les livres de la collection et des extraits sur www.editions-ellipses.fr

ISBN 9782340-073258
©Ellipses Édition Marketing S.A., 2022
8/10 rue la Quintinie 75015 Paris

DANGER
LE PHOTOCOPILLAGE TUE LE LIVRE

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

Ce livre est l'aboutissement de nombreuses années d'enseignement de la programmation à travers différents langages, en suivant l'évolution des besoins en codage informatique. Cet ouvrage est principalement destiné aux étudiants de licence mais peut être abordé par tous les scientifiques passionnés d'informatique désirant s'initier au langage Python et à l'algorithmique de base.

L'objectif de cet ouvrage est l'apprentissage du langage de programmation Python, assorti d'une initiation à l'algorithmique pour comprendre les principales structures de contrôles et les structures de données. Les possibilités de Python sont immenses et sont utilisées dans de nombreux domaines de l'informatique : les bases de données, l'administration des réseaux, la gestion d'internet. Ici, nous allons nous concentrer sur la création d'interface, le calcul numérique et la visualisation des résultats.

Tout au long de cet ouvrage, au fur et à mesure de la présentation de la syntaxe, des exemples et des exercices sont présentés pour mettre en œuvre l'algorithmique de base à travers le langage Python. La syntaxe de Python est plus légère que celle d'antres langages antérieurs mais les structures de données sont très sophistiquées. Cette richesse des structures de données permet de stocker tous les résultats intermédiaires, pour optimiser les temps de traitement, les temps de calcul des logiciels construit avec Python.

À l'heure où le traitement de données, le « big data », se développe considérablement, Python se révèle être un langage adapté aux futurs challenges du numérique et des sciences informatiques.

Le propos de cet ouvrage est de présenter les éléments de bases pour bien débuter en programmation, et progresser en algorithmique et vers la maîtrise de différents modules de Python.

Table des matières

Partie 1 : Programmation de base 17

Chapitre 1 : Analyse séquentielle 19

1.1 L'interpréteur : IDLE 19
1.2 Les calculs 20
1.3 Les variables 21
1.4 Quelques opérateurs 23
1.5 Les nombres complexes 24
1.6 Dialogue interactif : saisie au clavier, affichage écran 24

1.6.1 L'instruction « print » : affichage à l'écran 24
1.6.2 L'instruction « input() » : saisie au clavier 25

1.7 Chaînes de caractères 26

1.7.1 Opérations élémentaires sur les chaînes 26
1.7.2 Longueur d'une chaîne : len(chaine) 26
1.7.3 Accès aux éléments de la chaîne ch[i] 27
1.7.4 Sous-chaînes 27
1.7.5 Caractères spéciaux dans une chaîne 28
1.7.6 Formatage de chaînes 29

1.8 Exercices et solutions 31

1.8.1 Enoncés 31
1.8.2 Solutions 32

Chapitre 2 : Analyse conditionnelle 35

2.1 Expression conditionnelle 35

2.1.1 Test d'égalité 35
2.1.2 Opérateurs relationnels 35
2.1.3 Expression logique 35

2.2 L'instruction if elif else : syntaxes 37

2.2.1 Actions si une condition est vraie 37
2.2.2 Actions si une condition est vraie sinon d'autres actions 37
2.2.3 Plusieurs séquences d'actions if …elif…else 38
2.2.4 Ordre des conditions 39

2.2.5 Exemple : l'équation du second degré 40

2.3 Indentation des blocs d'instruction imbriqués 41
2.4 Exercices et solutions 41

2.4.1 Enoncés 41
2.4.2 Solutions 42

Chapitre 3 : Structure de données : les listes 45
3.1 Syntaxe 45
3.2 Fonctions prédéfinies 46

3.2.1 Accès à un élément 46
3.2.2 Longueur 46
3.2.3 Sous-liste 46
3.2.4 Les fonctions « min » et « max » 47
3.2.5 La fonction list 47

3.3 Construction de listes 47

3.3.1 Construction avec les fonctions + et * 47
3.3.2 Construction avec la fonction « range(debut, fin, pas) » 48
3.3.3 Modification d'un élément ou de plusieurs 48
3.3.4 Suppression d'un élément ou de plusieurs : la fonction « del » 49

3.4 Méthodes définies sur une liste 49
3.5 Sous-listes 51

Chapitre 4 : Analyse itérative 53
4.1 Introduction 53
4.2 La boucle for : analyse itérative avec arrêt fixe 53

4.2.1 Syntaxe 53
4.2.2 Exemple : parcours d'une liste d'entiers 53
4.2.3 Exemple : parcours des indices d'une liste 54
4.2.4 Exemple : parcours des éléments d'une liste 54
4.2.5 Construction de liste avec for 54
4.2.6 Exemple 2 : liste des possibilités de lancés de 3 dés 55

4.3 La boucle while : analyse itérative avec arrêt conditionnel 55

4.3.1 Syntaxe 55
4.3.2 Exemple : parcours d'une liste d'entiers 56
4.3.3 Exemple : ordre décroissant 57
4.3.4 Exemple : afficher la table des trois 57
4.3.5 Exemple : parcours des indices d'une liste 57

4.4 Exemples de calcul de somme 58

4.4.1 Calcul de la somme des n premiers carrés 58
4.4.2 Calcul de la somme des multiples de 7 de 0 à 1000 58
4.4.3 Moyenne des entiers pairs et des entiers impairs 58
4.4.4 Calcul de factoriel f = n != n*(n–1)*(n–2)*…*1 59

4.5 Suite définie par récurrence 59
4.6 Les chaînes de caractères : compter le nombre de « e » d'une chaîne 60
4.7 Algorithme de la racine carrée 61
4.8 Boucles imbriquées : décomposition en facteurs premiers 61
4.9 Choix entre for et while 62
4.10 Exercices et solutions 63

4.10.1 Enoncés 63
4.10.2 Solutions 64

Chapitre 5 : Fonction 67

5.1 Fonctions prédéfinies et modules 67

5.1.1 Importer un module de fonctions : module math 67
5.1.2 Module time 69
5.1.3 Module turtle 69
5.1.4 Module random 71

5.2 Syntaxe de la définition d'une fonction 72
5.3 Fonction avec ou sans paramètres et avec ou sans valeur de retour ... 73

5.3.1 Fonction sans paramètre et avec valeur de retour 74
5.3.2 Fonction avec paramètre et sans valeur de retour 74
5.3.3 Fonction sans paramètre et sans valeur de retour 74
5.3.4 Fonction avec paramètre et avec valeur de retour 75

5.4 Paramètres par défaut 75
5.5 Liste, grille et fonction 76
5.6 Exemple d'utilisation des fonctions 77

5.6.1 Calcul d'une suite 77
5.6.2 Liste des entiers parfaits inférieurs à n 77

5.7 Position de l'instruction return 78
5.8 Utilisation de sous-fonction 79
5.9 Variable locale et variable globale 79

5.9.1 Variable locale 79
5.9.2 Variable globale 80

5.10 Structure d'un programme Python 81

5.11 Le paramètre peut être une autre fonction 81
5.11.1 Exemple 1 81
5.11.2 Exemple 2 82
5.12 Quelques algorithmes de base 82
5.12.1 Test d'appartenance 82
5.12.2 Compter le nombre d'éléments, avec une condition 83
5.12.3 Extraire des éléments 83
5.12.4 Stocker des éléments calculés dans une liste 84
5.12.5 Supprimer tous les éléments x de la liste 84
5.13 Exercices et solutions 85
5.13.1 Enoncés 85
5.13.2 Solutions 87

Chapitre 6 : Structure de données : les dictionnaires 91
6.1 Structure de données : tuple 91
6.1.1 Définition 91
6.1.2 Utilisation 91
6.1.3 Valeur renvoyée par une fonction 93
6.1.4 Choix entre liste et tuple 93
6.2 Définition, construction d'un dictionnaire 93
6.2.1 Définition 93
6.2.2 Construction 94
6.2.3 Modification 94
6.2.4 Création d'un dictionnaire « en compréhension » 95
6.3 Fonctions prédéfinies sur un dictionnaire 95
6.4 Méthodes associées aux dictionnaires 96
6.5 Exemples d'utilisation d'un dictionnaire 98
6.5.1 Gestion de stock 98
6.5.2 Travail sur des mots ou des textes 99
6.5.3 Liste de dictionnaires pour gérer une bibliothèque 101
6.5.4 Mémorisation de calculs intermédiaire 102
6.6 Exercices et solutions 103
6.6.1 Enoncés 103
6.6.2 Solutions 104

Chapitre 7 : Le traitement des fichiers 107

7.1 Ouverture et fermeture 108
7.2 Ecriture 108
7.3 Lecture 109
7.4 « chaine = varfichier.read(n) » 109

7.4.1 « Ligne = varfichier.readline() » 110
7.4.2 « Ligne = varfichier.readlines(n) » 110
7.4.3 Lecture avec for 110

7.5 Les erreurs et leur interception par les exceptions 111

7.5.1 L'erreur de syntaxe : « SyntaxError :Invalid syntax » 111
7.5.2 Erreurs à l'exécution 112
7.5.3 Syntaxe de l'instruction try except 112
7.5.4 Exemple 112
7.5.5 La méthode « **isdigit()** » 113

7.6 Bibliothèque « os » et « sys » 113

7.6.1 Module os 113
7.6.2 Exemple 114
7.6.3 Module sys 115
7.6.4 Exemple 115

7.7 Exercices et solutions 115

7.7.1 Enoncés 115
7.7.2 Solutions 118

Partie 2 : Interface avec le module tkinter 121

Chapitre 8 : Introduction et premier exemple d'interface.. 123

8.1 Exemple d'interface minimum simple 123
8.2 Le même exemple avec des classes 125
8.3 Syntaxe minimum pour définir une interface 126
8.4 Conception graphique 126

Chapitre 9 : Les widgets simples 127

9.1 Les options communes à tous les widgets simples 127

9.1.1 Typographie 128
9.1.2 Spécifique aux objectifs des widgets 129
9.1.3 Position 130

9.2 Le widget Button 130
9.2.1 La syntaxe de la définition est la suivante : 130
9.2.2 Les options obligatoires sont les suivantes : 131
9.3 Le widget Label 131
9.3.1 Syntaxe 131
9.3.2 Principales options 131
9.4 Le widget Entry 132
9.4.1 Syntaxe 132
9.4.2 Principales méthodes 132
9.4.3 Principales options 132
9.5 Le widget Text 133
9.5.1 Syntaxe 133
9.5.2 Principales méthodes 133
9.5.3 Principales options 133
9.6 Le widget Frame et LabelFrame 134
9.7 Le widget Radiobutton 135
9.7.1 Syntaxe 135
9.7.2 La variable 135
9.7.3 Option obligatoire 136
9.7.4 Méthodes 136
9.7.5 Exemple 136
9.8 Exemple d'utilisation de widget simple 137

Chapitre 10 : Le widget Canvas 141
10.1 Le widget Canvas : syntaxe 141
10.2 Effacer le contenu avec la méthode « delete » 141
10.3 Méthodes permettant de créer des objets 141
10.3.1 Les images méthode .create_image() 141
10.3.2 Les textes méthode .create_text() 142
10.3.3 Les figures géométriques 142
10.3.4 Les lignes 143
10.3.5 Les ovales : les cercles et les ellipses 143
10.3.6 Les ovales : les portions et les arcs 143
10.3.7 Les polygones : les carrés ou les rectangles 144
10.3.8 Les polygones 145
10.4 Canvas et fond d'écran de l'application 145

10.5 Exemples d'application avec un widget Canvas....146

10.5.1 Jeu de déplacement 146
10.5.2 Tas de sable....150

Chapitre 11 : Le positionnement et la mise en forme....155

11.1 Les dimensions et le système de coordonnées....155
11.2 La couleur....156
11.3 La police....156
11.4 La méthode grid() 157

11.4.1 Options de positionnement pour la méthode grid()....157
11.4.2 Cacher un widget positionné avec la méthode « grid » 159
11.4.3 Configurer la taille des lignes et des colonnes....159

11.5 La méthode pack() 159
11.6 Récupérer la taille de l'écran attribut winfo… 160

Chapitre 12 : Les évènements et leur liaison....161

12.1 Syntaxe....161
12.2 Noms principaux de liaisons 161
12.3 Exemple d'application 162

12.3.1 L'interface conversion Francs/Euros....162
12.3.2 Le jeu du trésor 162

Chapitre 13 : Les widgets complexes 165

13.1 Le widget ListBox et Scrollbar 165

13.1.1 Syntaxe pour une petite liste 165
13.1.2 Une longue liste avec le widget Scrollbar....166

13.2 Le widget menu 167

13.2.1 Syntaxe....167
13.2.2 Exemple 167

Chapitre 14 : Interface et classe....169

Chapitre 15 : Complément....171

15.1 Couleurs....171
15.2 Méthodes communes à tous les widgets 172

Chapitre 16 : Exercices et solutions 173

16.1 Jeu de dominos 173

16.1.1 Version classique 173
16.1.2 Les dominos tête-à-tête 174
16.1.3 Le domino whist 174
16.1.4 Domino qui perd gagne 174
16.1.5 Décomposition en sous-problèmes 175

16.2 Interface conversion francs/euros 176
16.3 Interface de déplacement 176
16.4 Interface du tas de sable 176
16.5 Interface du jeu du trésor 176

Partie 3 : Calcul scientifique 177

Chapitre 17 : Module sympy : calcul littéral 179

17.1 Opération élémentaire et écriture d'une formule 179

17.1.1 Ecriture et affichage d'une formule 179
17.1.2 Factorisation, développement 180
17.1.3 Evaluation numérique et substitution 181
17.1.4 Transformation fonction sympy en fonction Python 181
17.1.5 Transformation fonction sympy en écriture latex 181

17.2 Tracé de courbe 182
17.3 Limite, dérivée, intégrale 183

17.3.1 Limite d'une fonction en un point a 183
17.3.2 Calcul d'une asymptote oblique 183
17.3.3 Dérivées 184
17.3.4 Intégrale et primitive 185

17.4 Résolution d'équations 187

17.4.1 Equations à une seule variable 187
17.4.2 Equations à plusieurs variables 188
17.4.3 Résolution de systèmes d'équations 188
17.4.4 Equations différentielles 188

17.5 Exercices et solutions 189

17.5.1 Enoncés 189
17.5.2 Solutions 191

Chapitre 18 : Module numpy : calcul matriciel 195

18.1 Création de tableau et copie 195
18.2 Propriétés d'un tableau 196
18.3 Sauvegarde et lecture de fichier 197
18.4 Opération mathématique 198

18.4.1 Opération arithmétique 198
18.4.2 Opération de comparaison et d'affectation 198
18.4.3 Fonctions mathématiques usuelles 198

18.5 Algèbre linéaire et calcul matriciel 198
18.6 Indexation et extraction de sous-tableaux « slicing » 199

18.6.1 Tableau 1D 199
18.6.2 Tableau 2D 200
18.6.3 Extraction conditionnelle 201
18.6.4 L'argument « axis » 203
18.6.5 Opération statistique d'agrégation 203
18.6.6 Autres opérations d'agrégat 204

18.7 Création de tableaux par blocs 205

18.7.1 Fonction np.concatenate() 205
18.7.2 Fonction np.stack() 206
18.7.3 Fonction np.append() 206
18.7.4 Fonction np.tile() 207

18.8 Redimensionnement 207

18.8.1 Fonction d'aplatissement np.ravel() 207
18.8.2 Les fonctions np.reshape() et np.resize() 207
18.8.3 Exemple : manipulation d'un graphe 209

18.9 Exercices et solutions 211

18.9.1 Enoncés 211
18.9.2 Solutions 213

Chapitre 19 : Module matplotlib : tracé de courbe 217

19.1 Superposition de courbe explicite 217

19.1.1 Exemple 1 218
19.1.2 Exemple 2 218
19.1.3 Exemple 3 219
19.1.4 Eléments de mise en forme des courbes 220
19.1.5 Mise en forme des axes 225
19.1.6 Ajouter du texte : titre ou annotation 228

19.1.7 Mise en forme de la zone de traçage 230

19.2 Tracé de courbe implicite et courbe paramétrée 230
19.3 Fenêtre avec plusieurs graphiques 231

19.3.1 La fonction « plt.subplots(nrows = 1, ncols = 3) » 231
19.3.2 La fonction « plt.subplot2grid(nrows = 3 ; ncols = 2) » 232

19.4 Graphique 3D 233
19.5 Nuage de points : méthode « scatter » 234
19.6 Tracé d'un histogramme : diagramme en barres 235

19.6.1 Fonction « plt.bar() » 235
19.6.2 Fonction « plt.hist() » 237
19.6.3 Histogramme de deux séries 237

19.7 Graphique en secteur : pie 238
19.8 Tracé avec marge d'erreur : « errorbar » 239
19.9 Ensemble de flèche orienté : « quiver » 240

19.9.1 Tracé d'un vecteur 240
19.9.2 Tracé d'un champ de vecteurs 241

19.10 Tracé de courbe avec deux variables 242

19.10.1 La fonction « numpy.meshgrid() » 242
19.10.2 Tracés de fractales 242

19.11 Afficher une image 244
19.12 Exercices et solutions 245

19.12.1 Enoncés 245
19.12.2 Solutions 246

Chapitre 20 : Module scipy : méthode numérique 249

20.1 Racines d'équation à une inconnue 249

20.1.1 Méthode de la dichotomie 250
20.1.2 Méthode de la sécante 251
20.1.3 Méthode de Newton 251
20.1.4 Comparaison des différentes méthodes 252

20.2 Système de n équations à n inconnues 253

20.2.1 Equation non linéaire 253
20.2.2 Système d'équations non linéaires 254
20.2.3 Equations différentielles 254

20.3 Interpolation 255

20.3.1 Fonction « np.interp », « scipy.interpolate.interp1d » 255
20.3.2 Interpolation par la méthode des moindres carrés..... 256
20.3.3 Interpolation polynomiale..... 261
20.3.4 Interpolation par des splines cubiques 261
20.3.5 Interpolation avec « scipy.interpolate.CubicSpline » 262
20.3.6 Interpolation 2d « scipy.interpolate.interp2d » 263
20.3.7 Interpolation 2d « scipy.interpolate.griddata » 263

20.4 Intégration 265

20.4.1 Méthode de Newton-Cotes..... 265
20.4.2 Méthode des rectangles..... 265
20.4.3 Méthode des trapèzes..... 266
20.4.4 Méthode de Simpson..... 266
20.4.5 Fonction du module « scipy.integrate » 266
20.4.6 Erreurs des différentes méthodes 268
20.4.7 Intégrales multiples..... 268

20.5 Exercices et solutions 270

20.5.1 Enoncés..... 270
20.5.2 Solutions 270

Chapitre 21 : Méthode numérique avancée..... 275

21.1 Gradient..... 275

21.1.1 Formule de base dans « numpy »..... 275
21.1.2 Fonction de « scipy.misc image »..... 276
21.1.3 Formule du gradient dans « scipy.ndimage »..... 279

21.2 Convolution 280
21.3 Récursivité 281

Chapitre 22 : Programmation Orientée Objet (POO) 283

22.1 Philosophie..... 283
22.2 La définition des objets : nom, attribut et méthodes 284

22.2.1 Les espaces de noms 284
22.2.2 Constructeur de la classe : méthode « __init__(self,.) » 284

22.3 L'utilisation de ces objets : l'instanciation..... 286
22.4 Propriétés des méthodes : héritage, surcharge..... 286

22.4.1 Surcharge ou polymorphisme 286
22.4.2 Héritage..... 287

22.5 Exemples interface 288

Bibliographie 289

Table des illustrations 291

Annexe 293

Index 295

Partie 1 : Programmation de base

Chapitre 1 : Analyse séquentielle

1.1 L'interpréteur : IDLE

Nous utiliserons IDLE qui est spécialement dédié à Python (et qui se télécharge automatiquement en même temps). On peut utiliser Python directement (non conseillé ensuite…). Le symbole « >>> » précise qu'on a la main. Notre premier écran Python :

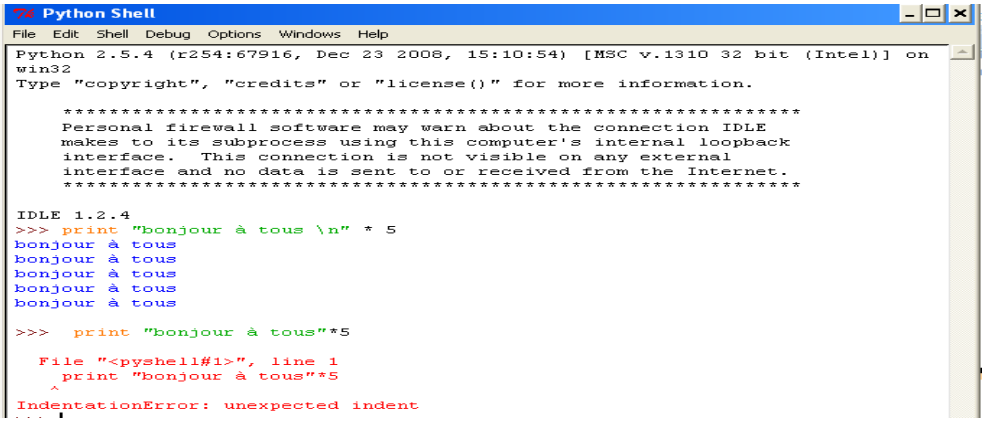


Fig. 1 Interface IDLE partie interpréteur

La particularité de Python est que le début et la fin de bloc d'instructions ne sont pas indiqués par un mot comme dans les autres langages structurés (C++, Java, …). Les instructions du bloc imbriqué dans le précédent doivent être indentées. i.e. des espaces en nombre fixe et paramétrable doivent être écrits en début de ligne, sinon il peut y avoir un message d'erreur. Le message d'erreur contient le nom du fichier, le numéro de ligne, l'instruction et le type d'erreur. Les erreurs de syntaxe sont gérées l'une après l'autre et provoque un message d'erreur bloquant.

Il est plus pratique et efficace d'utiliser un éditeur (comme emacs ou autres...). L'éditeur IDLE qui est téléchargé avec Python est très fonctionnel. Il permet de récupérer le numéro de ligne et de colonne du curseur. Les erreurs sont immédiatement localisées. Il permet d'ouvrir des fichiers pour y écrire des scripts (des programmes) variés, de les sauvegarder, puis de les exécuter. L'utilisation est très simple :

- On ouvre un fichier (ouverture ou création d'un nouveau fichier avec le menu déroulant)
- On écrit son script

- On le sauvegarde (« CTRL » S ou « enregistrer » ou « enregistrer sous » dans le menu déroulant)
- On l'exécute (F5 ou « **run** » dans le menu déroulant).

Chaque appel de « **run** » remet la mémoire à zéro (affichage de RESTART).
Le caractère # ouvre une ligne de commentaire qui ne sera pas lue par le programme (pas de symbole de fin de commentaire).

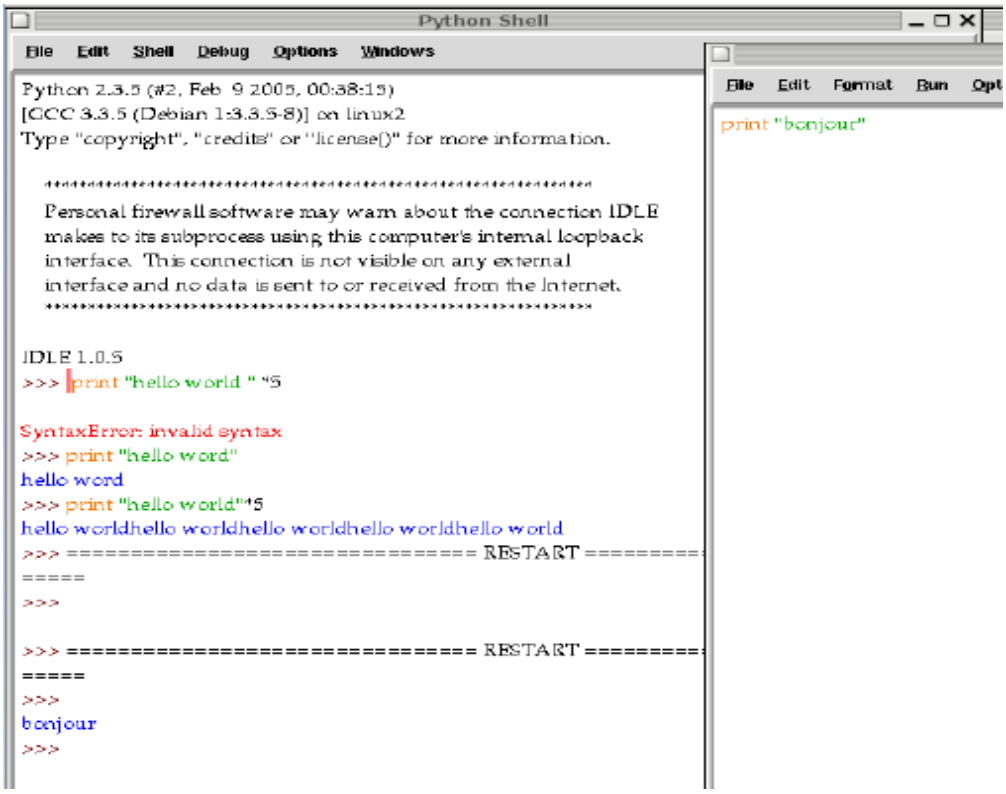


*Fig. 2 Exemple de d'utilisation d'un fichier *.py*

1.2 Les calculs

Division entière opérateur

Python et l'éditeur associé IDLE peuvent être utilisé comme calculatrice. Une fois le calcul entré, on fait « enter » au lieu de taper « = ». Dans les versions supérieures à 3.0, la division entre deux nombres entiers ou réels est réelle. Pour avoir une division entière il faut utiliser « // ». Le résultat est l'entier inférieur le plus proche. Si un des deux nombres est de type réel, le résultat est de type réel, sinon de type entier.

```
x = 9/2
n = 9//2
z = 9.0//2
print("x = ", x, "n = ", n, "z = ", z)
x = 4.5 y = 4 z = 4.0
```

Ordre de priorité

En Python, les règles de priorité sont les mêmes que celles des mathématiques. On les mémorise à l'aide de l'acronyme PEMDAS : P pour parenthèses, E pour exposants, M et D pour multiplication et division, qui ont la même priorité, A pour addition, S pour soustraction.
Les parenthèses ont la plus haute priorité. Elles permettent donc de « forcer » l'évaluation d'une expression dans l'ordre désiré. Ainsi (1+1)**(5–2) = 8 et 2*(3–1) = 4. Les puissances sont évaluées ensuite (après les expressions entre parenthèses), avant les autres opérations. Ainsi 2**1+1 = 3 (et non 4), et 3*1**10 = 3 (et non 59049!). Les multiplications et les divisions ont la même priorité et sont évaluées avant l'addition A et la soustraction S, lesquelles sont donc effectuées en dernier lieu. Ainsi 2*31 = 5 (plutôt que 4), et 6/31 = 1
Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite. Ainsi dans l'expression 59*100/60, la multiplication est effectuée en premier, et la machine doit donc ensuite effectuer 5900/60, ce qui donne 98. Si la division était effectuée en premier, le résultat serait 59 (puisqu'il s'agit d'une division entière).

1.3 Les variables

Règles du choix des noms

En Python, les noms de variables doivent obéir à quelques règles simples. Un nom de variable est une séquence de lettres (a ... z, A ... Z) et de chiffres (0…9), qui doit toujours commencer par une lettre (il est conseillé de prendre des noms explicites !!!). Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, à l'exception du caractère _ (tiret du bas). En particulier – est interdit. La casse est significative (les caractères majuscules et minuscules sont distingués).
Attention : largeur, Largeur et LARGEUR sont donc des variables différentes, x et X peuvent cohabiter (mais ce n'est pas conseillé !!!).
On écrit généralement les noms de variables en caractères minuscules (y compris la première lettre, mais ce n'est pas une obligation). On n'utilisera généralement les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans « **`tableDesMatieres`** », par exemple.

En plus de ces règles, certains mots sont réservés, ils ne peuvent donc pas servir de nom de variable. En particulier, les mots du langage Python sont interdits. Ils sont nombreux car Python est un langage Orientés Objets et possède de plus de nombreuses bibliothèques de fonctions prédéfinies. Les mots ci-dessous sont les plus courants :

```
and assert class def del elif else for from global if import in is lambda not
or pass print raise return while as
```

Il faut aussi tenir compte de toutes les fonctions prédéfinies qui ne peuvent pas servir de nom de variables. Un message d'erreur plus la coloration du mot en orange vous prévient de la confusion possible.

Affectation

L'affectation se fait avec le symbole « = », pour une définition ou une modification, avec le nom en premier. Une affectation multiple est possible. Les variables sont séparées par des virgules, ainsi que les valeurs respectives dans le même ordre. Si l'affectation est mal faite, il y a un message d'erreur.

```
x = 5
phrase = ”bonjour tout le monde”
x, y = 2.4, 8
z = x + y
```

L'affectation crée et mémorise un nom de variable, attribue un type donné, crée et mémorise une valeur particulière, établit un lien (système de pointeurs) entre le nom et l'emplacement mémoire de la valeur correspondante. En Python : la déclaration de type est inutile. Le typage se fait automatiquement (typage dynamique). La fonction prédéfinie permet de connaître le type affecté à la variable.
Une **affectation** utilise le signe « = » et elle signifie que l'on veut mettre un nombre dans une variable. L'ordinateur détermine (évalue) tout d'abord la valeur à droite du signe « = », en faisant tous les calculs nécessaires, elle est transférée dans la mémoire dont le nom est indiqué à gauche du « = ». Cela n'a pas grand-chose à voir avec l'égalité mathématique.

On peut écrire « a = 5 » mais pas « 5 = a »
On ne peut pas écrire « a + b = 0 »
« a = a+1 » signifie : regarder combien vaut a, ajouter 1, puis stocker le résultat dans « a » (son ancienne valeur étant alors écrasée par la nouvelle).

Type de variables

```
x = 4
y = 15.3
ph = ”bonjour”
print( type(x)) → <type 'int'>
```

```
print(type (y)) → <type 'float'>
print(type(ph)) → <type 'str'>
```

- int : un entier entre –2^31 +1 et 2^31 –1, arrondi au plus près à un entier.
- long : taille de l'entier non limitée x = 2345655555555
- float : nombre à virgule
- complex

1.4 Quelques opérateurs

- « **x**n** » ou pow(x, n) renvoie « x » à la puissance « n ».

```
x = 4
print(4**3) → 64
print(4**0.5) → 2.0
print(4**(1/2)) → 2.0
n1 = 3
n2 = 1/2
n3 = –1
y1 = x**n1
y2 = x**n2
y3 = x**n3
print(x, "à la puissance", n1, " = ", y1, "–", x, "à la puissance", n2, " = ",
y2, "–", x, "à la puissance", n3, " = ", y3)
➔ 4 à la puissance 3 = 64 – 4 à la puissance 0.5 = 2.0 – 4 à la
   puissance –1 = 0.25
```

- « **a%b** » Renvoie le reste de la division euclidienne de « a » par « b »:

  si a = b q + r alors a%b renvoie r

```
print(24%5) → 4
```

- La fonction « **min** » renvoie le minimum de la liste des paramètres, de même « **max** » renvoie le maximum.

```
a = 1
b = 2
c = 1
delta = b*b–4*a*c
valmin = min(a, b, c, delta)
valmax = max(a, b, c, delta)
print("a=",a,"b=",b,"c=",c) → a= 1 b= 2 c= 1
print("delta=",delta,"minimum=",valmin,"max=",valmax)
 → delta= 0 minimum= 0 max= 2
```

1.5 Les nombres complexes

Python gère automatiquement les nombres complexes. Mais attention, le complexe « a + ib » se note « a + jb » et il faut obligatoirement écrire « b » (même si b = 1); on peut considérer les imaginaires purs : « 1j ». Python gère : addition, soustraction, multiplication et division entre complexes, puissance…

```
print((3+5j)3+7j)        → 12j
print((3+5j)(3+7j) )     → 2j
print( (3+5j)/(3+7j) )   → (0.758620689655172380.10344827586206896j)
print( (3+5j)*( 2+7j) ) →(29+31j)
print( (1+2j)**2 )       →(3+4j)
```

1.6 Dialogue interactif : saisie au clavier, affichage écran

1.6.1 L'instruction « print » : affichage à l'écran

L'instruction « **print** » permet l'affichage de données (n'importe quel type de données). Pour afficher le contenu d'une variable, il faut indiquer le nom de la variable sans guillemet. Si le nom de la variable est encadré de guillemet, Python interprète le paramètre comme une chaîne de caractères et n'affiche pas le contenu de la variable mais le nom de la variable. Une virgule entre les données à afficher permet de mettre plusieurs affichages sur une même ligne. A partir de la version 3.0, il faudra mettre les arguments dans des parenthèses.
A partir de la version 3, la fonction print possède des paramètres qui permettent de spécifier le séparateur affiché (« **sep** ») et d'empêcher le passage à la ligne (« **end** »). Par défaut, le paramètre « **sep** » a la valeur « espace » et le paramètre « **end** » a pour valeur « **\n** » qui est un passage à la ligne.

```
print(2+3) → 5
print( "résultat du calcul" , "2+3 = ", 2+3) → résultat du calcul 2+3 = 5
ph1 = "marquise"
ph2 = 'vos beaux yeux'
ph3 = "d'amour"
ph4 = 'mourir'
ph5 = 'me font'
ph7 = 'hello world'
print( ph7) → hello world
print( ph1, ph3, ph2, ph4, ph5) → marquise d'amour vos beaux yeux
mourir me font
x = "premier cours de Python"
y = x[7:18]
```

```
print("x", x, sep = " = ", end = " ## ")
print("y = ", y)          → x = premier cours de Python ## y =  cours de P
```

1.6.2 L'instruction « input() » : saisie au clavier

La saisie au clavier et l'affichage à l'écran font partie de la gestion des entrées/sorties. Un programme est une suite finie et ordonnée d'instructions (un algorithme) pour résoudre un problème. Le programme a besoin de données initiales – les entrées – et calcul et renvoie les résultats – les sorties –. Les entrées sont données aux programmes sous différentes formes : saisies au clavier en interactif, lues dans un fichier, résultats issus d'un autre programme, mesures de capteurs, … De même pour les sorties, il existe différentes formes, les résultats peuvent s'afficher à l'écran, s'écrire dans un fichier, être transmis à un autre programme, …
L'appel de « **input(message)** » va provoquer l'affichage de la chaîne message (ce doit être une chaîne) et attendre que l'utilisateur écrive quelque chose puis tape sur « **enter** » pour valider la saisie. Attention, la saisie de l'utilisateur est obligatoirement une chaîne de caractères qu'il faut transformer en nombre pour faire les calculs.
La plupart des scripts élaborés nécessitent à un moment ou un autre une intervention de l'utilisateur (entrée d'un paramètre, clic de souris sur un bouton, etc.). Dans un script simple en mode texte (comme ceux que nous avons créés jusqu'à présent), la méthode la plus simple consiste à employer la fonction intégrée « **input()** ». Cette fonction provoque une interruption dans le programme courant. L'utilisateur est invité à entrer des caractères au clavier et à terminer avec la touche « **enter** ». Lorsque cette touche est enfoncée, l'exécution du programme se poursuit et la fonction fournit en retour une valeur correspondant à ce que l'utilisateur a entré. Cette valeur peut alors être assignée à une variable quelconque. On peut invoquer la fonction « **input()** » en laissant les parenthèses vides. On peut aussi y placer en argument un message explicatif destiné à l'utilisateur.

Exemple :

```
print( 'Veuillez entrer un entier positif quelconque : '),
nn = int(input())
print( 'Le carré de', nn, 'vaut', nn**2) ou encore :
prénom = input('Entrez votre prénom : ')
print( 'Bonjour, ', prénom )
```

Remarques importantes :
• La fonction « **input()** » renvoie une valeur dont le type est une chaîne de caractères. Dans notre exemple, la variable « **nn** » contient donc une chaîne de caractères qui est transformée en entier avec la fonction « **int()** ». La fonction « **input()** » renvoie toujours *une chaîne de caractères*. Vous pouvez ensuite convertir cette chaîne en nombre à l'aide de « **int()** » ou de « **float()** ».

Exemple :

```
a = input('Entrez une donnée : ')
Entrez une donnée : 52.37
print(type(a))   → <type 'str'>
b = float(a) # conversion en valeur numérique
print( type(b))  → <type 'float'>
 x = int(input( “age”))
25
print(x)   → 25
 nom = input (“votre nom “)
“Dupont”
print(nom)   → “DuPont”
```

La fonction **raw_input(“message”)** n’existe plus à partir de la version 3.

1.7 Chaînes de caractères

Une chaîne de caractères (c'est le type « **string** ») est une suite de caractères délimitée par des guillemets ou des apostrophes (ces guillemets ou apostrophes ne faisant bien évidemment pas partie de la chaîne). On mettra obligatoirement des guillemets si la chaîne contient une apostrophe et des apostrophes si la chaîne contient des guillemets. On doit impérativement mettre le même symbole au début et à la fin de la chaîne.
Attention : 2 apostrophes ne font pas des guillemets.

1.7.1 Opérations élémentaires sur les chaînes

L’addition + et la multiplication * s'appliquent aux chaînes. L’addition permet de concaténer (mettre bout à bout) les chaînes de caractères ou les variables représentants des chaînes. La multiplication permet de répéter n fois une chaîne.

```
print(ph1+ph2+ph3+ph4+ph5)      →
"marquisevos beaux yeuxd'amourmourirme font"
print(ph7 * 2)   → 'hello worldhello world'
```

1.7.2 Longueur d'une chaîne : len(chaine)

La fonction « **len()** » renvoie la longueur (le nombre de caractères la composant) de la chaîne passée en paramètre.

```
print(len (“bonjour”))      → 7
print(len(ph2+ph3))         → 21
```

1.7.3 Accès aux éléments de la chaîne ch[i]

Les caractères de la chaîne sont numérotés, on dit qu'ils ont un indice. Attention, en informatique, on numérote généralement à partir de 0. Si « **ch** » est une chaîne, « **ch[i]** » renverra le caractère de la chaîne dont l'indice est « **i** », Il y a une double indexation : de la gauche vers la droite avec des nombres positifs de 0 à n – 1 (si « **n** » est la taille de la chaîne) et de la droite vers la gauche avec des nombres négatifs commençant à la valeur « –1 ».

Sur un exemple :

B	O	n	j	o	u	R	!	
0	1	2	3	4	5	6	7	8
–9	–8	–7	–6	–5	–4	–3	–2	–1

c	o	u	r	s		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11
–12	–11	–10	–9	–8	–7	–6	–5	–4	–3	–2	–1

```
x = ”cours Python”
print(x[3])    → t
n = len(x)
print(n)   → 12
print(x[7] )   → y
print(x[0] )   → c
print(x[23] ) → Traceback (most recent call last): File "<pyshell#23>", line
1, in <module> IndexError: string index out of range
print(x[–1] ) → n
print(x[–2] ) → o
```

1.7.4 Sous-chaînes

A partir d'une chaîne, on peut extraire une « tranche » de cette chaîne (une sous-chaîne). En anglais, cette extraction se nomme le » slicing ». Si x est une chaîne, alors x[n :p] renvoie la tranche de x comprise entre les indices n (inclus) et p (exclu). Les paramètres de début et de fin peuvent être pris indifféremment dans les deux échelles d'indices. L'ordre considéré est celui de la chaîne de caractères. Si l'indice de fin correspond à un caractère situé avant le caractère représenté par l'indice de début, la chaîne renvoyée est vide.
Les deux paramètres sont facultatifs. Si le paramètre de début est omis, par défaut la valeur est 0 inclus (le premier caractère est inclus). Si le paramètre de fin est omis, par défaut la valeur est le dernier indice de la chaîne inclus (le dernier caractère est exclu).

```
x = ”cours de Python”
print(x[0:3])          → cou
print(x[13:6])         # 13<6 donc renvoie chaîne vide
print(x[8:250] )       → Python
print(x[1:–3] )        → ours de pyt
print(x[3:–1] )        → rs de pytho
pront(x[–5:–2] )       → yth
print(x[–9:8] )        → de
```

Les indices sont facultatifs. Par défaut, le début est 0 et la fin est –1 inclus.

```
print(x[:8] )  → Cours de
print(x[8:] )  → Python
print(x[:–9] )→ Cours
print(x[:–1] )→ premier cours de pytho # indice de fin est exclu, il manque
le 'n'
print(x[:])→ Cours de Python
print(x[3:3] )→  # indice de fin est exclu, renvoie chaîne vide
```

Attention on ne peut pas modifier directement une chaîne. Pour modifier il faut créer une autre chaîne en recopiant ou en modifiant les morceaux de la première.

```
y = "les fotes"
y[5] = "au"
Traceback (most recent call last):
 File "<pyshell#4>", line 1, in <module>
 y[5] = "au"
TypeError: 'str' object does not support item assignment
z = y[:5]+"au"+y[6:]
```

1.7.5 Caractères spéciaux dans une chaîne

On peut mettre dans une chaîne des caractères spéciaux en utilisant « \ ». Il permet d'insérer des caractères spéciaux dans une chaîne : en particulier, permet d'insérer une apostrophe dans une chaîne délimitée par des apostrophes. Les lettres accentuées sont codées, il est préférable d'utiliser l'instruction « **print** » pour les afficher.

- « **n** » provoque un saut à la ligne
- « **t** » ajoute un nombre d'espace fixe (une tabulation)

```
ch = "bonjour \n Python est un langage puissant"
print( ch)→ bonjour
Python est un langage puissant
ch = "bonjour \t Python est un langage puissant"
print( ch)     → Bonjour   Python est un langage puissant
x = "c'est bientôt l'été "
```

```
x       → "c'est bien\xc3\xb4t l'\xc3\xa9t\xc3\xa9 "
print( x)  → c'est bientôt l'été
```

1.7.6 Formatage de chaînes

On peut « fabriquer » de nouvelles chaînes en insérant des variables. Dans la chaîne que l'on veut ainsi fabriquer, le signe « **%s** » permet d'insérer une variable dans la chaîne qui s'affiche, puis ensuite indiquer le nom de la variable considéré avec « **%** » ou la liste de variable et de calculs entre parenthèses séparée par un espace de la chaîne de caractère à afficher.

```
x, y = 3 , 5
print( "on a x = %s " %x)       → on a x = 3
print("la somme de %s et %s est égale à %s " %(x, y, x + y))       →
la somme de 3 et 5 égale 8 '
```

1.7.6.1 Quelques fonctions et méthodes définies sur des chaînes

Fonction

Pour obtenir la liste exhaustive des méthodes appliquées à des chaînes, il faut taper l'instruction « **help(str)** ».

Ch = « cours de Python »	
N = int(Ch)	transforme une chaîne en entier (ou provoque une erreur)
X = float(Ch)	transforme une chaîne en nombre réel (ou provoque une erreur)
Ch = str(n)	transforme l'entier n en chaîne
Y = eval(Ch)	transforme une chaîne en nombre entier ou réel (ou provoque une erreur si la chaîne contient du texte)