Antoine Pernot • Supervisé par Alain Ploix

Programmer en Python 3

Programmer en Python 3

AC – Programmer en Python 3

Antoine Pernot
Supervisé par Alain Ploix

2017 – 2018

Table des matières

Objectifs du cours

Le présent document va aborder les concepts courants de la programmation, que nous allons appliquer à l'aide du langage de programmation Python dans sa version 3. Cet ouvrage a été conçu afin d'être utilisé en autonomie par un étudiant désireux d'apprendre à programmer mais ne disposant pas d'antécédents dans ce domaine. Il est composé d'un cours complet pouvant être approfondi par des recherches personnelles, de travaux dirigés (TD) et pratiques (TP) corrigés à la fin de ce livre.

Fonctionnement du cours et méthode de travail

Afin d'appréhender au mieux les concepts abordés dans le présent document, il vous est recom mandé de procéder comme suit :

— Lire un chapitre en entier.
— Faire les exercices associés au cours en question. Une correction est disponible en fin de ce livre.
— Approfondir avec des projets personnels pour vous familiariser au mieux avec les techniques abordées.
— Retrouver en fin de ce livre le memento permettant d'utiliser rapidement les éléments essen tiels du cours.

Installation de Python 3

Pour réussir ce cours, nous allons installer les outils nécessaires sur Debian, Ubuntu et Microsoft Windows. La suite de cours sera conçu pour les systèmes Debian et Ubuntu.

Debian et Ubuntu

Pour ces systèmes, il est nécessaire d'installer Python 3 ainsi que la bibliothèque PySide : `1 a p t i n s t a l l py t ho n3 py thon3−p y s i d e`

Il vous sera également nécessaire d'utiliser un éditeur de texte. Libre à vous d'utiliser l'éditeur de votre choix (Atom, Geany, Vim, Emacs, nano . . .).

Microsoft Windows

Nous allons télécharger et installer Python 3. Pour cela, rendez-vous sur . Cliquez sur **Download** puis choisissez **Download Python 3.X.X**. Exécutez l'installeur. Co

 chez **Add Python 3.X to PATH** et cliquez sur **Install Now** *(fig. 1)*

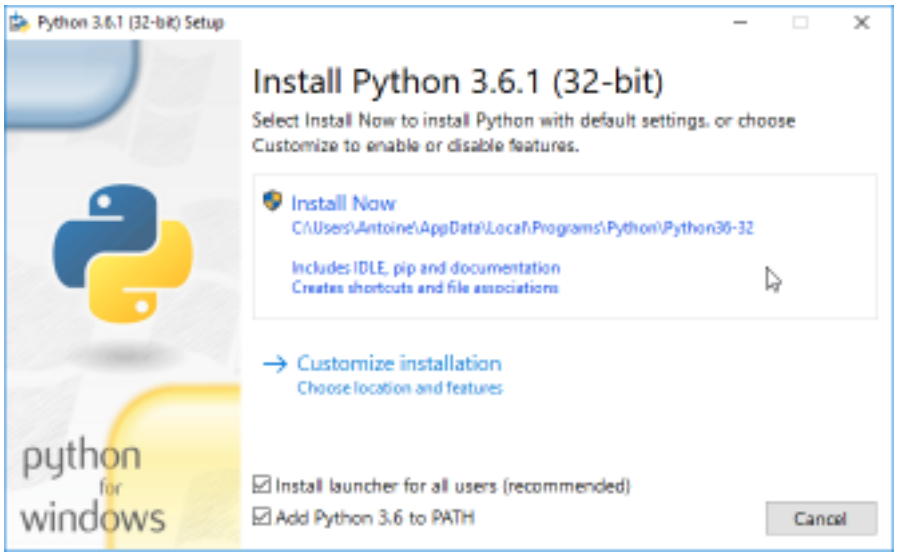

Figure 1 – Installation de Python

Nous allons ensuite installer la bibliothèque PySide, utilisée lors de ce cours. Pour cela, ouvrez un invité de commande et saisissez :

`1 p i p i n s t a l l −U PySi de`

Il vous sera également nécessaire d'utiliser un éditeur de texte. Libre à vous d'utiliser l'éditeur de votre choix (Atom, Geany, Notepad++ . . .).

Chapitre 1

Premiers pas avec Python 3

Nous allons débuter ce cours en effectuant des opérations à l'aide de l'interpréteur Python. En effet, il est possible d'utiliser Python via l'interpréteur ou en interprétant un fichier source. L'utilisation de l'interpréteur est recommandée pour expérimenter une fonctionnalité. Il nous servira de "cahier de brouillon" que vous pourrez utiliser tout au long de ce cours. Les chapitres ultérieurs se concentreront sur l'écriture de scripts destinés à être sauvegardés et interprétés.

Au lancement de l'interpréteur Python, vous obtenez ceci :

```
1 Python 3 . 5 . 2 ( d ef a u l t , Nov 17 2016 , 1 7 : 0 5 : 2 3 )
2 [GCC 5 . 4 . 0 20160609] on l i n u x
3 Type " h e l p " , " c o p y r i g h t " , " c r e d i t s " o r " l i c e n s e " f o r more
      i nf o r m a t i o n .
4 >>>
```

Ces quelques lignes nous indiquent la version de Python (ici 3.5.2) et que l'interpréteur est prêt à exécuter des commandes avec les caractères . Dans les exemples de ce cours, ces mêmes caractères permettent de signaler qu'on utilise l'interpréteur.

Opérations arithmétiques

Pour vous familiariser avec le fonctionnement d'un invité de commandes, nous allons effectuer quelques opérations mathématiques simples :

- 10 + 9
- 5 × 11
- 4 + 7 × 3
- (4 + 7) × 3
- 40 ÷ 6 (division exacte)
- 40 ÷ 6 (division euclidienne soit la partie entière du résultat)
- 40 modulo 6 (reste de la division ci-dessus)
- 27,6 + 4,2

Pour cela, nous allons saisir les commandes suivantes :

```
1 >>> 10+9
2 19
3 >>> 5*11
4 55
5 >>> 4 + 7*3 # L e s e s p a c e s s o n t i g n o r é s
6 25
7 >>> (4+7) * 3
```

```
 8 33
 9 >>> 40/6
10 6.666666666666667
11 >>> 40//6
12 6
13 >>> 40%6
14 4
15 >>> 27 ,6+4 ,2 # Ne fonctionne pas avec la virgule, doit être un poi
       nt
16 >>> 27.6+4.2
17 31.8
```

Voici donc un tableau récapitulatif des différentes opérations arithmétiques de bases abordées :

Opération	**Syntaxe**
Addition	
Soustraction	
Multiplication	
Division exacte	
Division entière	
Modulo	
Puissance (a^b)	
Arrondi de a avec b décimales	

Table 1.1 – Opérations mathématiques de base

Il faut ajouter à cela les opérateurs logiques que nous aborderons plus tard. **N'oubliez pas que Python respecte la priorité des opérations mathématiques.**

Les variables

À l'instar des mathématiques, nous allons utiliser des variables afin de stocker des données et travailler avec. Ces variables contiennent des données numériques (au format binaire) pouvant représenter :

— Des nombres entiers (dits entiers) et des nombres réels (la virgule est symbolisée par un **point**)
— Un texte (ou chaîne de caractères). Syntaxe :
— Des listes. Syntaxe :

— Des listes associatives. Syntaxe :
— Une fonction
— Un booléen (= vrai et = faux)
— . . .

On parle alors de **types** de variables. Nous détaillerons le fonctionnement et les opérations pouvant être effectuées sur ces différents types de variables plus tard.

Chaque variable est identifiée à partir d'un nom que vous donnez. Ce nom est à choisir afin qu'il respecte les consignes suivantes :

— Le nom de la variable doit être court mais indiquer son contenu pour faciliter la lecture du code.
— Ne doit pas être un **nom réservé** *(table 1.2)*.
— Ne doit comporter que des lettres (a → z et A → Z), des chiffres (0 → 9) et le caractère _ (*underscore*). Les **autres symboles sont interdits**.
— Python est **sensible à la casse**, c'est-à-dire que les majuscules et minuscules sont distinguées (exemple : , et sont des variables différentes).
— Il est vivement recommandé de nommer les variables en minuscule, y compris la première lettre et de commencer les mots suivants avec une majuscule pour plus de lisibilité (exemple :). **Dans ce cours, cette convention de nommage sera à appliquer.**

Table 1.2 – Liste des noms réservés en Python

Affectation de variables

Nous allons aborder maintenant comment affecter une valeur à une variable :

```
1 nombre=25
2 t e x t e B i e n v e n u e=" Bo njo u r à t o u s ! "
3 p i =3.14159
```

Avec l'exemple ci-dessus, nous avons créé trois variables :

— La variable nommée contenant l'entier 25.
— La variable nommée contenant le texte "Bonjour à tous !". — La variable nommée contenant la valeur approchée de π à savoir le nombre réel 3,14159.

Chacune de ces lignes effectue les opérations suivantes :

1. Allouer un **espace mémoire** pour cette variable.
2. Affecter un **nom de variable** à cet espace via un **pointeur**.
3. Définir le type de cette variable (entier, texte . . .).
4. Mémoriser la valeur dans l'espace affecté.

Après l'exécution des lignes présentées, voici un extrait de la mémoire vive occupée par notre programme *(table 1.3)* où on y retrouve deux espaces : l'**espace de noms** et l'**espace de valeurs**, liés entre eux par un **pointeur**.

	→ 25
	→ Bonjour à tous !
	→ 3.14159

Table 1.3 – État de la mémoire (extrait)

Il est cependant possible d'effectuer des **affectations multiples**. Un premier exemple consiste à affecter une même valeur pour plusieurs variables :

```
1 t e m p e r a t u r e D i j o n = t em p e ra t u r eRo u e n = 15.3
```

On peut également regrouper l'affectation de valeurs à des variables avec les **affectations paral lèles** :

```
1 t em p e r a t u r eT ro y e s , t e m p e r a t u r e A u x e r r e = 17 , 14.2
```

Dans l'exemple ci-dessus, les variables et prendront respectivement les valeurs 17 et 14,2.

Il est enfin à noter qu'une variable peut contenir le **résultat d'une opération** :

```
1 l o n g u e u r R e c t a n g l e = 25
2 l a r g e u r R e c t a n g l e = 12
3 p e r i m e t r e R e c t a n g l e = ( l o n g u e u r R e c t a n g l e + l a r g e u r R e c t a n g l e ) ∗
2
```

La partie gauche d'un signe égal doit toujours être un nom de variable et non une expression. Par exemple, l'instruction est **invalide**.

Il est cependant très répandu d'avoir ce type d'expression tout à fait impossible en mathématiques : . Cela permet d'ajouter 1 à la valeur de la variable . On dit alors qu'on **incrémente** . Vous pouvez enfin **réaffecter** une valeur à une variable à savoir écraser le contenu d'une variable par une autre valeur.

Afficher la valeur d'une variable

Pour afficher le contenu d'une variable, nous allons utiliser la fonction :

```
1 >>> print(nombre)
2 25
3 >>> print(texteBienvenue)
4 Bonjour à tous !
```

La composition d'instructions

Après avoir vu quelques opérations de base, nous pouvons d'ores et déjà les combiner ensemble pour former des instructions plus complexes. En voici un exemple :

```
1 masseTomatesEnGrammes, masseCurryEnKg = 3600, 0.001
2 print("Vous avez acheté", masseTomatesEnGrammes/1000, "kg de toma te
      s et", masseCurryEnKg*1000, "grammes de curry.")
```

Aide

Pour obtenir de l'aide sur l'utilisation d'une fonction ou d'un module, tapez en remplaçant par la fonction ou le module recherché.

```
1 >>> help(print)
2 print(...)
3 print(value,...,sep=' ', end='\n', file=sys.stdout, flush=False
      )
4
5 Prints the values to a stream, or to sys.stdout by default. 6 Optional keyword arguments:
7 file: a file-like object(stream); defaults to the current sys.stdout.
8 sep: string inserted between values, default a space. 9 end: string appended after the last value, default a newline. 10 flush: whether to forcibly flush the stream.
```

Exercices

1. Cochez les instructions valides :

2. Écrivez les instructions nécessaires pour affecter l'entier 17 à la variable , le nombre réel 14,2 à la variable et le texte "Temps nuageux" à la variable avec et sans les affectations multiples.

3. Écrivez les instructions permettant de calculer l'aire d'un disque. La formule est $A = \pi \times R^2$ avec A l'aire et R le rayon du disque valant ici 5 cm. Le résultat doit être affiché sous la forme "L'aire du disque est de XXX cm2".

4. Décrivez précisément ce que fait le programme suivant :

```
1 longueurPiece = 17 # En mètres
2 largeurPiece = 9 # En mètres
3 longueurCarrelage = 0.3 # En mètres
4 airePiece = longueurPiece * largeurPiece
5 aireCarreauCarrelage = longueurCarrelage **2
6 nombreCarreauxCarrelage = airePiece / aireCarreauCa
rrelage 7 print("Il vous faudra", nombreCarreauxCarrelage
, "carreaux pour recouvrir une pièce de", airePiece, "m2.")
```

Solutions page 139.

Chapitre 2

Le flux d'instructions

Un programme consiste en une suite d'instructions structurées et exécutées par un ordinateur. Cette définition est le cœur de ce chapitre. Nous allons ici écrire nos premiers scripts Python en utilisant les bases que nous avons vues dans le chapitre précédant mais en modifiant le déroulement des opérations en introduisant l'**exécution conditionnelle**.

Les scripts

Nous ne travaillerons plus à partir de l'interpréteur, mais nous éditerons des scripts que nous pourrons éditer et sauvegarder. Un script Python consiste en un fichier texte ayant pour extension **.py** et contenant du code Python. En voici un exemple :

```
1 #! / u s r / b i n / env py t ho n3
2
3 l o n g u e u r P i e c e = 17 # En mè t r e s
4 l a r g e u r P i e c e = 9 # En mè t r e s
5 l o n g u e u r C a r r e l a g e = 0. 3 # En mè t r e s
6 a i r e P i e c e = l o n g u e u r P i e c e * l a r g e u r P i e c e
7 a i r e C a r r e a u C a r r e l a g e = l o n g u e u r C a r r e l a g e **2
8 n om b r e C a r r e a u x C a r r e l a g e = a i r e P i e c e / a i r e C a r r e a u C a
r r e l a g e 9 p r i n t ( " I l vo u s f a u d r a " , nom b r e C a r r e a u x C a r r e l ag e
  , " c a r r e a u x po u r r e c o u v r i r une p i è ce de " , a i r e P i e c e , "m2 . " )
```

Vous aurez remarqué qu'il s'agit des mêmes instructions que nous avons déjà rencontrées. Notez par ailleurs l'ajout de la ligne : cette ligne indique aux systèmes d'ex ploitation de type Unix qu'il s'agit d'un script Python et non d'un autre type de script (tels que Ruby, bash . . .) et lui fournit l'adresse de l'interpréteur avec lequel lire le script (ici). Il s'agit du **shebang**. Dans ce cours, nous **imposons** la présence de ce shebang.

Pour créer un nouveau script, utilisez un éditeur de texte [1]tel que Notepad++, Atom ou plus simplement bloc-notes. Pour taper votre code source et enregistrez-le dans un fichier ayant pour

1. À ne pas confondre avec un traitement de texte tel que Word, OpenOffice.org Writer ou LibreOffice

 extension **.py**.

Pour exécuter un script Python, entrez la commande suivante dans votre shell

Linux : 1 py t ho n3 mo n S c ri p t . py

Pour rendre votre code exécutable directement, effectuez ceci :

1 chmod +x mo n S c ri p t . py

Il est possible d'interrompre manuellement l'exécution d'un programme avec la combinaison des touches et .

CTR L

Les types d'erreurs

Malgré tout le soin que vous porterez à programmer, des erreurs se glisseront dans vos pro grammes. Il en existe trois types principaux :

Erreur de syntaxe : Dans un programme, la syntaxe doit être parfaitement correcte : la moindre erreur dans le nom des fonctions ou d'indentation [2] provoquera un arrêt de fonctionnement ou une exécution erratique.

Erreur sémantique : Le programme fonctionne, cependant, vous n'obtenez pas le résultat souhaité. Dans ce cas, c'est que le séquencement des instructions de votre programme n'est pas correct.

Erreur d'exécution : Un élément extérieur vient perturber le fonctionnement normal de votre pro gramme. Ces erreurs, également appelées **exceptions**, sont dues à des circonstances particu lières comme par exemple votre programme doit lire un fichier mais il est absent ou l'utilisateur n'a pas entré la valeur attendue.

Les commentaires

Lors de l'écriture d'un script, il est **très vivement recommandé** de commenter ce que fait le programme. Pour ajouter une ligne de commentaires, celle-ci doit commencer par un #. Cela indique à l'interpréteur d'ignorer la ligne à partir de ce caractère jusqu'à la fin de la ligne.

```
1 a=4 # V o i c i un comme n tai re .
2 # Un comme n tai re s u r t o u t e l a l i g n e .
3 # Exemple :
4 t a i l l e T o u r E i f f e l = 324 # Exp rim é en mè t r e s j u s q u ' à l ' a n t e n n e .
```

Dans ce cours, il sera **exigé** de commenter les opérations complexes pour en définir simplement leur fonctionnement.

2. L'indentation consiste en l'ajout de tabulations en début de ligne pour délimiter un bloc d'instructions.

Il est tout aussi **recommandé** de renseigner une **documentation** sur le script ou la fonction que vous développez pour renseigner comment elles fonctionnent et comment s'interfacer avec. La documentation doit être saisie **au début**, **précédée** et **suivie** par ceci : **"""** (trois fois les double-guillemets). En voici un exemple :

```
1 #! / u s r / b i n / env py t ho n3
```

```
2 """Calculatrice pour la pose de carrelage."""
3
4 longueurPiece = 17 # En mètres
5 largeurPiece = 9 # En mètres
6 longueurCarrelage = 0.3 # En mètres
7 airePiece = longueurPiece * largeurPiece
8 aireCarreauCarrelage = longueurCarrelage **2
9 nombreCarreauxCarrelage = airePiece / aireCarreauCar
relage 10 print("Il vous faudra", nombreCarreauxCarrelage,
  "carreaux pour recouvrir une pièce de", airePiece, "m2.")
```

Cette documentation sera accessible en tapant

Séquences d'instructions

Sauf cas spéciaux, les instructions sont exécutées les unes après les autres. Cette caractéristique peut parfois jouer des tours si l'ordre des instructions n'est pas correct. Voici un exemple dans lequel l'ordre des instructions est crucial. Dans ce cas, intervertir les lignes 2 et 3 donne un résultat différent :

```
1 a, b = 2, 8
2 a=b
3 b=a
4 print(a, b)
```

Interaction utilisateur

Nous allons rendre nos programmes plus interactifs. Pour cela, nous allons demander à l'utilisateur final de saisir des données que nous utiliserons dans nos programmes. Nous allons donc utiliser la fonction . Si vous saisissez une chaîne de caractères entre les parenthèses (on parle de **"passer en argument"** la chaîne de caractères), celle-ci précédera la saisie utilisateur mais ceci est facultatif. La version 3 de Python détecte automatiquement le typage de la saisie utilisateur. Voici un exemple d'utilisation de cette fonction :

```
1 >>> nom = input("Entrez votre nom : ")
2 Entrez votre nom : Michel
3 >>> print("Bonjour", nom)
4 Bonjour Michel
```


Le type de variable retourné est une **chaîne de caractères**. Pour la convertir en réel (dit flottant), utilisez . Pour la convertir en entier, utilisez .

Les conditions

Nous allons introduire un des cas particuliers à l'ordre d'exécution des instructions : **les**

condi tions. Cela permet d'exécuter une portion de code si une certaine condition est remplie, et une autre portion si cette même condition n'est pas remplie. Voici un exemple permettant de clarifier cela :

```
1 nombre = int(input("Entrez un nombre : "))
2 inferieurA5 = False
3 if nombre < 5 :
4 print("Le nombre est inférieur à 5.")
5 inferieurA5 = True
6 else :
7 print("Le nombre est supérieur ou égal à 5.")
```

Dans le cas précédent, les lignes 4 et 5 seront exécutées si l'utilisateur saisit un nombre inférieur à 5. La ligne 7 est exécutée si le nombre ne remplit pas cette condition. De manière plus générale, voici la syntaxe d'une condition :

```
1 if condition1 :
2 début bloc code si la condition 1 est vraie
3 ...
4 fin bloc code si la condition 1 est vraie
5 elif condition2 :
6 début bloc code si la condition 1 est fausse et la condition 2 est
        vraie
7 ...
8 fin bloc code si la condition 1 est fausse et la condition 2 est vraie
9 else :
10 début bloc code si les conditions 1 et 2 sont fausses 11 .
..
12 fin bloc code si les conditions 1 et 2 sont fausses
```

Pour cela, on utilise les instructions (**si** en anglais) auxquelles on juxtapose la **condition**, et terminées par " **:**". Cette condition est à remplir pour exécuter le bloc de code délimité par une **tabulation** en début de chaque ligne.

On peut, ceci est facultatif, mettre une instruction (**sinon si** en anglais) permettant de vérifier une seconde condition si la première est fausse. On peut chaîner autant d'instructions que nécessaire.

Enfin, l'instruction (**sinon** en anglais), elle aussi facultative, exécute un bloc de code si les conditions énumérées avec et n'ont pas été remplies.

Les opérateurs de comparaison

Plusieurs opérateurs de comparaison sont disponibles *(table 2.1)*.

Comparateur	Syntaxe	Types de variables

a égal à b		Tous
a différent de b		Tous
a supérieur à b		Numériques
a supérieur ou égal à b		Numériques
a inférieur à b		Numériques
a inférieur ou égal à b		Numériques
a [pas] dans b		a : Tous, b : Liste ou texte

Table 2.1 – Opérateurs de comparaison

Les blocs d'instructions

Comme abordés dans la partie précédente, nous parlerons des blocs d'instructions. Un bloc d'instructions est un ensemble d'instructions (pléonasme !) associé à une **ligne d'en-tête** (nous avons déjà vu , ou mais on retrouve également Début de la ligne ment , , . . .). Ces lignes d'en-tête se terminent par un double point (:).

Un bloc de code est **délimité par son indentation**. Les lignes d'instructions d'un même bloc de code sont indentées de manière identique (possèdent le même nombre de tabula tions au début de celles-ci). Les blocs peuvent être **imbriqués** comme l'illustre le schéma suivant :

Bloc 1
Ligne d'en-tête :
Bloc 2
Ligne d'en-tête :
Bloc 3
Suite du bloc 2
Suite du bloc 1

Figure 2.1 – Imbrication des blocs

L'exemple suivant met en application l'imbrication de blocs d'instructions :

```
1 nombre = 17
2 i f nombre > 1 0:
3 p r i n t ( " Le nombre e s t sup é r i e u r à 10 " ) 4 i f
nombre < 2 0:
5 p r i n t ( " Le nombre e s t i n f é r i e u r à 20 " ) 6 e l s e :
7 p r i n t ( " Le nombre e s t sup é r i e u r à 20 " ) 8 e l s e :
```

 9 p r i n t (" Le nombre e s t i n f é r i e u r à

10 ")

Exercices

N’oubliez pas le shebang et commentez si nécessaire.

1. Écrivez un programme demandant l’âge de l’utilisateur et affichant si celui-ci est majeur ou mineur.

2. Écrivez un programme permettant, à partir d’un montant hors taxes saisi par l’utilisateur, de calculer le montant de la TVA (20% du montant hors taxes) et du montant TTC (montant hors taxes auquel on ajoute le montant de la TVA) et donnez le détail du calcul.

3. Écrivez un programme vérifiant si dans un texte saisi par l’utilisateur, celui-ci contient le mot "fraise" ou le mot "pêche".

Solutions page 140.

Chapitre 3

Factoriser le code

Plus vos programmes se complexifieront, plus le code source écrit sera volumineux. Pour y remédier, nous allons introduire deux nouveaux concepts : les **boucles** et les **fonctions**. Nous aborderons deux formes de boucles qui répondent à deux modes de fonctionnement différents. Nous poursuivrons ensuite sur les fonctions fournies dans Python 3 et enfin sur comment écrire ces fonctions soi-même. À partir de ce chapitre, la présence du shebang, de la documentation dans le code, les noms de variables explicites et de commentaires sont **obligatoires**.

Boucles "Tant que"

Les boucles ont pour but de répéter un ensemble d'instructions tant qu'une certaine condition est remplie. Dès que cette condition n'est plus remplie, l'interpréteur rompt la boucle et continue l'exécution du script. Il existe deux types de boucles en Python : la boucle **"tant que"** et la boucle **"pour"** que nous aborderons plus tard.

La boucle **"tant que"** (**"while"** en anglais) permet d'exécuter une portion de code tant que la condition fournie est vraie. On utilise le mot **while**. Voici un exemple trivial permettant de mettre en œuvre cette nouvelle fonction :

```
1 comp teu r = 1
2 w h i l e comp teu r <= 1 0:
3 p r i n t ( comp teu r )
4 comp teu r = comp teu r + 1
```

L'exemple ci-dessus affichera les nombres de 1 à 10 inclus à raison de un par ligne.

La boucle n'est pas exécutée si au début la condition n'est pas remplie. Les variables constituant la condition doivent exister avant l'exécution de la boucle.

Faites attention à toujours avoir un facteur modifiant la condition de la boucle. En effet, si la condition est toujours remplie, la boucle s'exécutera pour toujours. On parle alors de **boucle infinie**. Voici un exemple de boucle infinie :


```
1 comp teu r = 1
2 w h i l e comp teu r <= 1 0:
3 p r i n t ( comp teu r )
```

L'instruction

Il peut être nécessaire, lorsque l'on ne connaît pas à l'avance sous quelle condition on va quitter une boucle. C'est notamment le cas des communications réseau ou des lectures de fichiers. Pour ar rêter immédiatement l'exécution d'une boucle à tout moment de son itération, on utilise l'instruction .

Les fonctions

Une fonction est une séquence d'instructions permettant d'effectuer une tâche précise. Nous allons ici réduire le volume de notre code en regroupant les instructions en tâches et créer des fonctions associées. Nous allons voir également que Python fournit beaucoup de fonctions dont nous avons déjà vu quelques unes.

Les fonctions fournies par Python

Nous avons déjà vu dans les chapitres précédents les fonctions **print()**, **help()** et **input()**. Nous allons voir certaines fonctions disponibles utiles.

Trans-typage

Lors de la saisie de données par l'utilisateur, la valeur retournée est une chaîne de caractères. Pour effectuer des opérations mathématiques, il est nécessaire de la convertir en valeur entière ou décimale. Pour ce faire, nous utiliserons les fonctions **int()** pour convertir en entier et **float()** pour convertir en décimal. On peux également tronquer un nombre décimal avec la fonction int() :

```
1 saisieUtilisateur = "75.9"
2 valeurFlottante = float(saisieUtilisateur)
3 valeurEntiere = int(saisieUtilisateur)
```

À l'inverse, il est possible de convertir une donnée en chaîne de caractères avec la fonction **str()**.

Comparer des valeurs

Il est possible de récupérer la valeur minimale et maximale d'une série de valeurs séparées par des virgules avec respectivement les fonctions **min()** et **max()**.

```
1 valeurMaximale = max(4, 17.8, 12)
2 valeurMiminale = min(4, 17.8, 12)
```

Ces fonctions sont notamment utiles lorsque l'on manipule des listes, que nous verrons plus tard.

Importer des modules

Python 3 est fourni avec un ensemble de fonctions disponibles nativement. Cependant, il peut être nécessaire d'importer un module afin d'utiliser une fonction précise. Pour

illustrer comment importer un module Python et utiliser les fonctions qui y sont référencées, nous allons utiliser la fonction du module permettant de tirer un nombre entier au hasard.

Il est d'usage et fortement recommandé d'importer les modules en début de programme, sous le shebang. Nous utiliserons cette convention tout au long de ce cours.

Importer des fonctions d'un module

Si vous utilisez un faible nombre de fonctions d'un même module, il est recommandé d'importer que ces dernières et ainsi faire l'économie des fonctions inutiles du module :

```
1 from random import randint, shuffle # Importe les fonctions randint
et shuffle (inutile dans cet exemple) du module random 2 aleatoire = ran
    dint(1,10) # Nombre aléatoire entre 1 et 10 inclus
```

Importer un module complet

Si vous utilisez une grande partie des fonctions d'un module, il est moins fastidieux d'importer les fonctions une à une. Dans ce cas, on importe le module complet :

```
1 import random # Importe le module random
2 aleatoire = random.randint(1,10) # Nombre aléatoire entre 1 et
    10 inclus (fonction du module random)
```

Importer un autre fichier Python

Vous pouvez également importer un autre script Python comme module. Pour cela, il faut que votre module soit écrit dans le même dossier que votre script. Pour l'importer, utilisez l'une des deux méthodes ci-dessus en renseignant le nom du fichier module sans son extension.

ATTENTION : Lorsque vous importez un module dans vos programmes, le code non présent dans des fonctions ou classes s'exécute. Par exemple :
Fichier monmodule.py

```
1 def maFonction():
2 print("Ceci est ma fonction")
3 print("Bonjour le monde !")
```

Fichier monscript.py

```
1 import monmodule
2 print("Le ciel est bleu")
```

Dans cet exemple, la ligne 3 du fichier sera exécutée lors de l'import.

Créer des fonctions

Nous allons aborder maintenant comment créer nous-mêmes des fonctions. Pour cela, nous utiliserons l'instruction **def**. Une fonction peut prendre des **arguments** en entrée et

retourner une valeur en fin avec l'instruction **return**, tout cela est facultatif. Voici un exemple que nous détaillerons ensuite :

```
1 def addition(a,b):
2 resultat=a+b
3 return(resultat)
4 resultatAddition=addition(4,7)
5 print(resultatAddition)
```

La première ligne introduit notre nouvelle fonction nommée addition et nécessitant deux argu ments, les variables a et b. **Ces variables, ainsi que celles déclarées au sein de cette fonc tion, ne sont valables que dans cette fonction.** Le corps de la fonction, composé des lignes 2 et 3, indique d'effectuer une opération sur les variables et de retourner la valeur de la variable

. **L'exécution de la fonction s'arrête lors du , même si d'autres instructions la suivent.**

Une fois la fonction définie, nous l'utilisons à la ligne 4 en lui fournissant comme arguments les valeurs a=4 et b=7. On récupère le résultat de cette fonction dans la variable . On affiche le contenu de la variable à la ligne 5. **Si le résultat d'une fonction n'est pas récupéré, il est perdu.**

Il est possible d'utiliser plusieurs fois une même fonction :

```
1 def addition(a,b):
2 return(a+b)
3 resultatAddition=addition(4,7)
4 print(resultatAddition)
5 resultatAddition=addition(8,4.12)
6 print(resultatAddition)
```

Lorsqu'une fonction est créée, elle prévaut sur celle fournie par défaut par Python. Si vous appelez une de vos fonctions , celle-ci sera appelée en lieu et place de celle fournie de base.

Lors de l'utilisation d'une fonction, tous les arguments sont obligatoires. Vous pouvez rendre un argument facultatif en lui fournissant une **valeur par défaut** :

```
1 def tableAddition(valeur,fin=10):
2 compteur = 1
3 while compteur <= fin : # Par défaut, la valeur de fin est à 10. 4 print(
compteur , "+" , valeur , "=" , compteur+valeur ) 5 compteur += 1 # Équivalen
t à compteur = compteur + 1
```



```
6 tableAddition(4)# La variable fin aura pour valeur 10 7 t
ableAddition(6,15)# La variable fin aura pour valeur 15
```

Vous pouvez fournir les arguments dans le désordre lors de l'appel d'une fonction en nommant vos variables :

```
1 def addition(a,b):
2 return(a+b)
3 resultatAddition=addition(b=2,a=9)
4 print(resultatAddition)
```

Ici, la fonction aura pour arguments a=9 et b=2. Ce type d'appel peut être utile pour des fonctions ayant un grand nombre d'arguments pour plus de clarté.

Enfin, il est également possible de retourner plusieurs valeurs à la fin d'une fonction :

```
1 def valeursExtremes(valeurs):
2 return(min(valeurs),max(valeurs))
3 valeurMinimale,valeurMaximale=valeursExtremes([14,96,57,10,0.7])
```

Exercices

N'oubliez pas le shebang et commentez si nécessaire.

1. Écrivez un programme permettant d'effectuer les opérations de base (+, −, ×, ÷) en mathé matiques en créant des fonctions.

2. Créez un programme tirant un nombre au hasard entre 1 et 20 et demandant à l'utilisateur de deviner ce nombre en lui indiquant si sa proposition est supérieure ou inférieure au nombre tiré.
   Exemple :

3. Écrivez un programme permettant de demander à l'utilisateur la réponse à la multiplication de deux nombres tirés aléatoirement. Votre programme lui posera 15 questions et calculera un score noté sur 20.

Solutions page 141.

Chapitre 4

Les séquences

Les séquences en Python sont des structures de données composées d'entités plus petites. Dans cette catégorie, on retrouve les **chaînes de caractères**, les **listes** et les **dictionnaires**. Ces types de données ont en commun des fonctions permettant d'avoir des informations sur elles ou de les modifier, ainsi qu'une boucle parcourant un à un les éléments de celles-ci. Nous allons étudier en détail ces structures, les fonctions, ainsi que la boucle "Pour" dans ce chapitre.

Les chaînes de caractères

Nous avons déjà abordé précédemment les chaînes de caractères mais sans entrer dans le détail. Il est tout d'abord primordial d'avoir à l'esprit que les chaînes de caractères sont composées de caractères accessibles par un **indice**. Le schéma suivant permet d'illustrer les caractères associés à leurs indices *(fig. 4.1)*.

Figure 4.1 – Caractères associés à leurs indices

On peut donc accéder aux caractères un à un à partir de leurs indices et n'utiliser qu'une partie de la chaîne en demandant un fragment de celle-ci :

```
>>> chaineDeCaracteres = " Montagne "
>>> print(chaineDeCaracteres[0])
M # Premier caractere
>>> print(chaineDeCaracteres[3])
t # Quatrieme caractere
>>> print(chaineDeCaracteres[-1])
e # Dernier caractere
>>> print(chaineDeCaracteres[-2])
n # Avant-dernier caractere
>>> print(chaineDeCaracteres[-3])
```



```
g # Avant-avant-dernier caractère
>>> print(chaineDeCaracteres[2:5])
n ta # Tranche du troisième au cinquième caractère 14
>>> print(chaineDeCaracteres[:4])
Mont # Tranche du premier au quatrième caractère 16
>>> print(chaineDeCaracteres[6:])
ne # Tranche du septième au dernier caractère
>>> print(chaineDeCaracteres[7:2:-1])
e ngat # Tranche du huitième au troisième caractère dans le sens inverse
>>> print(chaineDeCaracteres[::-1])
engatnoM # La chaîne complète dans le sens inverse
>>> print(chaineDeCaracteres[::2])
Mnan # Les lettres d'indice pair
>>> print(chaineDeCaracteres[1::2])
o tg e # Les lettres d'indice impair
```

Comme nous l'avons vu dans l'exemple précédent, on peut accéder aux caractères d'une chaîne en entrant le nom de la variable suivi de l'indice entre crochets. Un indice négatif permet d'accéder aux caractères à partir de la fin.

La syntaxe générale est : (par défaut 0) (par défaut la fin de la chaîne) (par défaut 1) avec inclus, exclu et le pas.

Si une valeur n'est pas renseignée, sa valeur par défaut est appliquée. Si une seule valeur est entrée entre crochets, seulement le caractère à l'indice du début est retourné (exemple lignes 2 à 10).

Enfin, il est possible de **concaténer** [1] des chaînes de caractères **uniquement** à l'aide du symbole + :

```
1 >>> prenom = " Arnaud "
2 >>> t e x t e = " Bo njo u r " + prenom + " , comment va s-t u ? " 3 >>> p
r i n t ( t e x t e )
4 Bo njo u r Arnaud , comment va s-t u ?
```

Modifier la casse d'une chaîne de caractères

La casse désigne le fait de distinguer les lettres majuscules des lettres minuscules. Python possède les méthodes , , , et permettant de modifier la casse d'une chaîne de caractères :

1. Enchaîner, mettre bout à bout deux chaînes de caractères.


```
1 >>> t e x t e = "É c r i t pa r A n to i n e de S a i n t -Exup é r y "
2 >>> p r i n t ( t e x t e . l ow e r ( ) ) # Tout en m i n u s c u l e
3 é c r i t pa r a n t o i n e de s a i n t -exup é r y
4 >>> p r i n t ( t e x t e . u p p e r ( ) ) # Tout en m a j u s c u l e
5 ÉCRIT PAR ANTOINE DE SAINT-EXUPÉRY
6 >>> p r i n t ( t e x t e . t i t l e ( ) ) # M aj u s c u l e à chaque mot 7 É c r i
t Pa r A n to i n e De S a i n t-Exup é r y
8 >>> p r i n t ( t e x t e . c a p i t a l i z e ( ) ) # M aj u s c u l e en dé b u t de p h r
a s e 9 É c r i t pa r a n t o i n e de s a i n t -exup é r y
10 >>> p r i n t ( t e x t e . swa pca s e ( ) ) # I n v e r s e l a c a s s e
11 éCRIT PAR aNTOINE DE sAINT-eXUPÉRY
```

Compter les occurrences dans une chaîne ou une liste

La méthode permet de compter le nombre d'occurrences de la sous-chaîne dans la chaîne ou liste :

```
1 >>> t e x t e = "É c r i t pa r A n to i n e de S a i n t -Exup é r y "
```

```
2 >>> texte.count("a")#Sensibleàlacasse
3 2
4 >>> texte.lower().count("a")
5 3
```

Les listes et les tuples

Une liste et un tuple Python sont un ensemble **ordonné** d'éléments de tous types. Elles peuvent contenir en leur sein des chaînes de caractères, des nombres, des autres listes, des objets . . . Elles peuvent contenir des éléments de plusieurs types à la fois.

La différence entre une liste et un tuple est qu'une liste est modifiable et un tuple, non. Chaque élément est séparé par une virgule. Voici ci-dessous la syntaxe pour déclarer une liste et un tuple :

```
1 exempleListe = [27, 24.8, "Bonjour"]
2 exempleTuple = (27, 24.8, "Bonjour")
3 listeDansUneListe = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

À l'instar des chaînes de caractères, on peut accéder aux éléments d'une liste ou d'un tuple par son indice entre crochets :

```
1 >>> exempleListe = [27, 24.8, "Bonjour"]
2 >>> listeDansUneListe = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
3 >>> exempleListe[0]
4 27
5 >>> exempleListe[1:]
```



```
6 [24.8, 'Bonjour']
7 >>> listeDansUneListe[0]
8 [1, 2, 3]
9 >>> listeDansUneListe[0][1]
10 2
```

Il est possible de modifier, d'ajouter ou de supprimer une valeur d'une liste **et non d'un tuple**. Nous allons aborder toutes ces opérations.

Ajouter un élément en fin de liste

Pour ajouter un élément en fin de liste, on utilise la méthode :

```
1 >>> fournitures = ["cahier","crayon","stylo","trousse",
"gomme"] 2 >>> fournitures.append("ciseaux")
3 >>> print(fournitures)
4 ['cahier','crayon','stylo','trousse','gomme','ciseaux']
```

Modifier un élément

La modification d'un élément se fait en réaffectant la nouvelle valeur à la place de l'ancienne :

```
1 >>> fournitures=["cahier","crayon","stylo","trousse",
"gomme"] 2 >>> fournitures[1]="équerre"
3 >>> print(fournitures)
4['cahier','équerre','stylo','trousse','gomme']
5 >>> fournitures[3]=["trombones","calque"]
6 >>> print(fournitures)
7['cahier','équerre','stylo',['trombones','calque'],'gomme']
```

Ajouter un élément au cœur de la liste

Ajouter un élément se fait en modifiant une tranche de la liste dont le début et la fin de la tranche sont identiques :

```
1 >>> fournitures=["cahier","crayon","stylo","trousse","gomme"
]2 >>> fournitures[2:2]=["cartable"]# Doit être dans une liste ou un t
uple.
3 >>> print(fournitures)
4['cahier','crayon','cartable','stylo','trousse','gomme'
]5 >>> fournitures[4:4]=["règle","feuilles"]
6 >>> print(fournitures)
7['cahier','crayon','cartable','stylo','règle','feuilles','t
    rousse','gomme']
```


Supprimer un élément

Il existe deux méthodes pour supprimer un élément d'une liste : et .

La méthode

Cette méthode permet de supprimer la première occurrence de l'élément passé en argument :

```
1 >>> fournitures=["cahier","crayon","stylo","trousse",
    "gomme","stylo"]
2 >>> fournitures.remove("stylo")
3 >>> print(fournitures)
4['cahier','crayon','trousse','gomme','stylo']
```

La méthode

Cette méthode permet de supprimer un élément par son indice et retourne l'élément supprimé :

```
1 >>> fournitures=["cahier","crayon","stylo","trousse",
"gomme"] 2 >>> element=fournitures.pop(3)
```

```
>>> print(fournitures)
['cahier','crayon','stylo','gomme']
>>> print(element)
trousse
```

Diviser une chaîne en liste

On peut séparer une chaîne de caractères en liste en utilisant la méthode . Cette méthode utilise en argument une chaîne délimitant chaque élément :

```
>>> listeFournitures="cahier;crayon;stylo;trousse;gomme"
>>> fournitures=listeFournitures.split(";")
>>> print(fournitures)
['cahier','crayon','stylo','trousse','gomme']
```

Assembler une liste en chaîne

La méthode permet de concaténer chaque élément de la liste séparé par un séparateur :

```
>>> fournitures=["cahier","crayon","stylo","trousse","gomme"]
>>> listeFournitures=";".join(fournitures)
>>> print(listeFournitures)
cahier;crayon;stylo;trousse;gomme
```


Trier une liste

La méthode permet de trier dans l'ordre croissant une liste selon ses valeurs :

```
>>> fournitures=["cahier","crayon","stylo","trousse","gomme"]
>>> fournitures.sort()
>>> print(fournitures)
['cahier','crayon','gomme','stylo','trousse']
```

Inverser l'ordre d'une liste

La méthode permet d'inverser l'ordre des valeurs d'une liste :

```
>>> fournitures=["cahier","crayon","stylo","trousse","gomme"]
>>> fournitures.reverse()
>>> print(fournitures)
['gomme','trousse','stylo','crayon','cahier']
```

Mélanger une liste

La méthode du module permet de mélanger aléatoirement les valeurs d'une liste :

```
>>> from random import shuffle
```

```
2 >>> nomb res = [ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 ]
3 >>> s h u f f l e ( nomb res )
4 >>> p r i n t ( nomb res )
5 [ 7 , 9 , 3 , 1 , 8 , 4 , 6 , 5 , 2]
6 >>> s h u f f l e ( nomb res )
7 >>> p r i n t ( nomb res )
8 [ 2 , 1 , 9 , 4 , 6 , 8 , 5 , 3 , 7]
```

Trouver l'index d'un élément

La méthode retourne l'indice du premier élément passé en argument :

```
1 >>> f o u r n i t u r e s = [ " c a h i e r " , " c r a y o n " , " s t y l o " , " t r o u s s e " ,
      "gomme" , " s t y l o " ]
2 >>> f o u r n i t u r e s . i n d e x ( " s t y l o " )
3 2
```

Copie de liste

Nous allons étudier la copie de liste. On peut instinctivement tenter cette opération avec la commande suivante mais sans succès :


```
1 >>> f o u r n i t u r e s = [ " c a h i e r " , " c r a y o n " , " s t y l o " , " t r o u s s e " ,
"gomme" ] 2 >>> c o p i e F o u r n i t u r e s = f o u r n i t u r e s
3 >>> f o u r n i t u r e s . remove ( " c r a y o n " )
4 >>> p r i n t ( f o u r n i t u r e s )
5 [ ' c a h i e r ' , ' s t y l o ' , ' t r o u s s e ' , 'gomme ' ]
6 >>> p r i n t ( c o p i e F o u r n i t u r e s )
7 [ ' c a h i e r ' , ' s t y l o ' , ' t r o u s s e ' , 'gomme ' ]
```

Lors de la ligne 2, Python crée un alias à la nouvelle liste et n'effectue pas de copie. Chaque opération apportée sur chaque variable affectera la liste qui est accessible par ses alias. Ainsi, pour effectuer une véritable copie d'une liste, il est nécessaire d'utiliser la fonction du module :

```
1 >>> f rom copy im po r t dee pco py
2 >>> f o u r n i t u r e s = [ " c a h i e r " , " c r a y o n " , " s t y l o " , " t r o u s s e " ,
"gomme" ] 3 >>> c o p i e F o u r n i t u r e s = dee pco py ( f o u r n i t u r e s )
4 >>> f o u r n i t u r e s . remove ( " c r a y o n " )
5 >>> p r i n t ( f o u r n i t u r e s )
6 [ ' c a h i e r ' , ' s t y l o ' , ' t r o u s s e ' , 'gomme ' ]
7 >>> p r i n t ( c o p i e F o u r n i t u r e s )
8 [ ' c a h i e r ' , ' c r a y o n ' , ' s t y l o ' , ' t r o u s s e ' , 'gomme ' ]
```

Création rapide d'une suite de nombre

Il peut être utile de générer une suite de nombres. Pour cela, il est recommandé d'utiliser la fonction (inclus, par défaut 0) (exclu) (par défaut 1).

```
1 >>> l i s t e = [ i f o r i i n r a n g e ( 0 , 2 0 , 2 ) ]
2 >>> p r i n t ( l i s t e )
3 [ 0 , 2 , 4 , 6 , 8 , 10 , 12 , 14 , 16 , 1 8]
```

Cette syntaxe peut être utilisée pour initialiser une liste avec des valeurs identiques :

```
1 >>> l i s t e = [ None f o r i i n r a n g e (10 ) ]
2 >>> p r i n t ( l i s t e )
3 [ None , None , None , None , None , None , None , None , None , None ]
```

Boucles "Pour"

Une boucle **"Pour"** (**"For"** en anglais) permet d'exécuter une portion de code pour chaque élément d'une liste en les affectant à une variable. Une fois que la liste est terminée, l'exécution normale du programme se poursuit.

Voici un exemple de l'utilisation de cette structure :

```
1 f o u r n i t u r e s = [ " c a h i e r " , " c r a y o n " , " s t y l o " , " t r o u s s e " ,
"gomme" ] 2 f o r e l em e n t i n f o u r n i t u r e s :
3 p r i n t ( e l em e n t )
```

L'exemple ci-dessus affichera les éléments de la liste à raison de un par ligne. Il est possible d'utiliser la fonction directement :

```
1 f o r comp teu r i n r a n g e (15 ) :
2 p r i n t ( comp teu r )
```

L'exemple ci-dessus affichera les nombres de 0 à 14 inclus à raison de un par ligne.

Récupérer l'élément et son indice

La fonction retourne l'indice et l'élément un à un :

```
1 >>> f o u r n i t u r e s = [ " c a h i e r " , " c r a y o n " , " s t y l o " , " t r o u s s e " ,
"gomme" ] 2 >>> f o r i n d e x , e l em e n t i n e n ume ra te ( f o u r n i t u r e s ) :
3 . . . p r i n t ( " I n d i c e : " + s t r ( i n d e x ) + " => " + e l em e n t ) 4 . . .
5 I n d i c e : 0 => c a h i e r
6 I n d i c e : 1 => c r a yo n
7 I n d i c e : 2 => s t y l o
8 I n d i c e : 3 => t r o u s s e
```

```
9 Indice : 4 => gomme
```

Parcourir deux listes en même temps

La fonction permet de parcourir plusieurs listes en même temps :

```
1 >>> a = [1, 2, 3]
2 >>> b = [4, 5, 6]
3 >>> c = [7, 8, 9]
4 >>> for i in zip(a, b, c):
5 ...     print(i)
6 ...
7 (1, 4, 7)
8 (2, 5, 8)
9 (3, 6, 9)
```

Les dictionnaires

Un dictionnaire est un ensemble **désordonné** d'éléments de tous types. Chaque élément est identifié à l'aide d'une **clé**. Voici la syntaxe pour déclarer un dictionnaire. :


```
1 exempleDictionnaire = {"livre": 74, 85: "tulipe", 74.1: "rose",
      "coquelicot": False, "agrumes": ["citron", "orange", "
      pamplemousse"]}
```

Le dictionnaire étant un ensemble désordonné, l'ordre de déclaration des éléments qui le composent n'a pas d'importance. Chaque élément est accessible par sa clé :

```
1 >>> exempleDictionnaire = {"livre": 74, 85: "tulipe", 74.1: "ros
e", "coquelicot": False, "agrumes": ["citron", "orange", "
pamplemousse"]} 2 >>> exempleDictionnaire[74.1]
3 'rose'
4 >>> exempleDictionnaire['coquelicot']
5 False
6 >>> exempleDictionnaire['coquelicot'] = 'Rouge'
7 >>> exempleDictionnaire['coquelicot']
8 'Rouge'
9 >>> exempleDictionnaire["agrumes"]
10 ['citron', 'orange', 'pamplemousse']
11 >>> exempleDictionnaire["agrumes"][1]
12 'orange'
13 >>> exempleDictionnaire['livre'] += 1
14 >>> print(exempleDictionnaire['livre'])
15 75
```

On peut ajouter un élément dans un dictionnaire comme suit :

```
>>> quantiteFournitures={"cahiers":134,"stylos":{"rouge":41,"bleu":74},"gommes":85}
>>> quantiteFournitures["agrafes"]=49
>>> quantiteFournitures["stylos"]["noir"]=16
>>> print(quantiteFournitures)
{'stylos':{'bleu':74,'noir':16,'rouge':41},'cahiers':134,'gommes':85,'agrafes':49}
```

Supprimer un élément

À l'instar des listes, on utilise la méthode pour supprimer un élément par sa clé et retourner l'élément supprimé :

```
>>> quantiteFournitures={"cahiers":134,"stylos":{"rouge":41,"bleu":74},"gommes":85}
>>> element=quantiteFournitures.pop("gommes")
>>> print(quantiteFournitures)
{'stylos':{'bleu':74,'rouge':41},'cahiers':134} 5
>>> print(element)
85
```


Lister les clés d'un dictionnaire

La méthode retourne la liste des clés du dictionnaire :

```
>>> quantiteFournitures={"cahiers":134,"stylos":{"rouge":41,"bleu":74},"gommes":85}
>>> cles=quantiteFournitures.keys()
>>> forfournitureincles:
...print(fourniture,"Quantité:",quantiteFournitures[fourniture])
...
stylosQuantité:{'bleu':74,'rouge':41}
cahiersQuantité:134
gommes Quantité:85
>>> print(list(cles))
['stylos','cahiers','gommes']
```

Lister les valeurs d'un dictionnaire

La méthode retourne la liste des valeurs du dictionnaire :

```
>>> quantiteFournitures={"cahiers":134,"stylos":{"rouge":41,"bleu":74},"gommes":85}
>>> valeurs=quantiteFournitures.values()
>>> print(values)
dict_values([{'bleu':74,'rouge':41},134,85])
```

Copier un dictionnaire

La méthode permet de créer une copie indépendante d'un dictionnaire :

```
1 >>> quantiteFournitures={"cahiers":134,"stylos":{"rouge":4
      1,"bleu":74},"gommes":85}
2 >>> inventaire=quantiteFournitures.copy()
3 >>> quantiteFournitures.pop("cahiers")
4 134
5 >>> print(quantiteFournitures)
6 {'stylos':{'bleu':74,'rouge':41},'gommes':85}
7 >>> print(inventaire)
8 {'stylos':{'bleu':74,'rouge':41},'gommes':85,'cahiers':134}
```

Parcourir un dictionnaire

On peut parcourir un dictionnaire par ses clés ou ses clés et ses valeurs avec la méthode :


```
1 >>> quantiteFournitures={"cahiers":134,"stylos":{"rouge":4
      1,"bleu":74},"gommes":85}
2 >>> for cle in quantiteFournitures:
3 ...     print(cle)
4 ...
5 stylos
6 cahiers
7 gommes
8 >>> for cle,valeurs in quantiteFournitures.items():
9 ...     print(cle,valeurs)
10 ...
11 stylos {'bleu':74,'rouge':41}
12 cahiers 134
13 gommes 85
```

Exercices

N'oubliez pas le shebang et commentez si nécessaire.

1. Écrivez un programme simulant le fonctionnement d'une banque en stockant le solde des comptes dans un dictionnaire. Il devra permettre le dépôt et le retrait de sommes d'argent.

2. Écrivez un programme de loterie en demandant à l'utilisateur de choisir 6 numéros entre 1 et 50 inclus et d'effectuer un tirage aléatoire sur les mêmes critères. Enfin, il

devra vérifier le nombre de numéros gagnants.

3. Écrivez un programme permettant de vérifier si un mot ou une phrase saisis par l'utilisateur est un palindrome, à savoir un mot lisible à la fois à l'endroit ou à l'envers tel que Serres, radar, rotor ou "Ésope reste ici et se repose" :

4. Écrivez un programme permettant de générer le calendrier d'une année non-bissextile dont le premier janvier tombe un samedi (telle que 2011). Les jours et les mois seront stockés dans des tuples. L'affichage final sera du type :

5. Écrivez un programme permettant de calculer la quantité d'ingrédients de la recette ci-dessous en fonction du nombre de biscuits fourni par l'utilisateur. La liste

d’ingrédients doit être stockée dans un dictionnaire.

Biscuits écossais (20 biscuits)

300g de farine • 75g de beurre • 75g de sucre roux • 1 œuf • 50 ml de lait • ½ sachet de levure chimique • 1 sachet de sucre vanillé.

Mélanger farine et levure, ajouter le beurre, le sucre et pétrir avec les doigts. Ajouter l’œuf battu et le lait. Bien mélanger. Fariner la boule de pâte, étaler la pâte sur ½ cm et découper des formes avec un emporte-pièce. Cuire 12 à 15 minutes à 190°C.

 6. Écrivez un programme affichant le mot le plus long d’une phrase entrée par l’utilisateur.

7. Écrivez un programme permettant de trier une liste de nombres sans utiliser la méthode . Réécrivez une fonction de tri de liste avec l’algorithme de tri à bulles qui consiste à comparer deux valeurs consécutives d’une liste et de les permuter quand elles sont mal triées et de répéter cela jusqu’à ce que la liste soit triée. Vous utiliserez les nombres tirés aléatoirement.

Solutions page 142.

Chapitre 5

Manipuler les fichiers

Il peut être nécessaire de lire ou d'écrire des fichiers stockés sur l'ordinateur exécutant vos scripts. Consigner des données dans des fichiers permet de simplifier un programme en externalisant les données et peut être un moyen de s'interfacer avec d'autres programmes et systèmes ainsi qu'avec les utilisateurs. Nous utiliserons la fonction fournie par défaut . Avant tout, il est nécessaire de voir comment naviguer dans l'arborescence.

Navigation dans l'arborescence

En fonction du répertoire dans lequel est exécuté votre script, il peut être nécessaire de changer de répertoire de travail du script. Pour se faire, nous utiliserons la fonction dans le module pour changer de répertoire de travail. Nous utiliserons également la fonction du même module. Il est possible de créer un dossier avec :

```
1 >>> f rom o s im po r t getcwd , c h d i r , mk di r
2 >>> p r i n t ( ge tcwd ( ) )
3 /home/ a n t o i n e
4 >>> c h d i r ( ' e s s a i s ' )
5 >>> p r i n t ( ge tcwd ( ) )
6 /home/ a n t o i n e / e s s a i s
7 >>> mk di r ( ' t e s t ' )
```

Ouvrir un fichier

Pour lire un fichier, il faut tout d'abord ouvrir un flux de lecture ou d'écriture de fichier avec la fonction (par défaut : 'rt') avec l'adresse du fichier à ouvrir et , le type de flux à ouvrir. Le mode est composé de deux lettres, les droits d'accès () et l'encodage (). Voici le tableau détaillant les modes de flux de fichier *(table 5.1)*.

Pour fermer le flux de fichier avec la méthode sur la variable représentant le flux.

Il est important de **fermer le flux** une fois les opérations sur le fichier terminé.

Caractère	Action
	Ouvrir en lecture seule (défaut)
	Ouvrir en écriture. Écrase le fichier existant.
	Ouvrir en écriture si et seulement si le fichier n'existe pas déjà.
	Ouvrir en écriture. Ajoute au fichier existant.
	Mode binaire.
	Mode texte (défaut).

Table 5.1 – Mode pour

Lire un fichier

Une fois le flux en lecture ouvert, on peut utiliser les méthodes qui retournent une chaîne de caractères contenant l'intégralité du fichier ou retournant une liste où chaque élément est une ligne du fichier.

```
1 >>> fichier = open("texte.txt",'rt')
2 >>> texte = fichier.read()
3 >>> print(texte)
4 Lorem ipsum dolor sit amet, consectetur adipiscing elit.
5 Pellentesque gravida erat ut lectus convallis auctor. 6 Fusce mollis sem id tellus auctor hendrerit.
7 >>> lignes = fichier.readlines()
8 >>> print(lignes)
9 ['Lorem ipsum dolor sit amet, consectetur adipiscing elit.\n','Pell
    entesque gravida erat ut lectus convallis auctor.\n','Fu
```

```
sce m o l l i s sem i d t e l l u s a u c t o r h e n d r e r i t . \ n ' ]
10 >>> f i c h i e r . c l o s e ( )
```

Chaque ligne est terminée par "" qui représente un retour à la ligne. Il s'agit d'un caractère spécial. Si vous écrivez ce caractère dans une chaîne de caractères, Python produira un retour à la ligne :

```
1 >>> t e x t e = " P remi è r e l i g n e \ nDeuxi ème l i g n e "
2 >>> p r i n t ( t e x t e )
3 P remi è r e l i g n e
4 De uxi ème l i g n e
```

Écrire un fichier

On peut écrire un fichier si le flux est ouvert en écriture. Les trois flux possibles sont "", "" et "". À l'instar de et , on utilisera pour écrire une chaîne de caractères et avec une liste ou un tuple dont chaque élément est une ligne à écrire. **N'oubliez pas le caractère en fin de ligne pour revenir à la ligne.**


```
1 >>> f i c h i e r = open ( " t e x t e . t x t " , 'wt ' )
2 >>> f i c h i e r . w r i t e ( " Lorem ipsum d o l o r s i t amet , c o n s e c t e t u r a d i p i s c
i n g e l i t . \ n P e l l e n t e s q u e g r a v i d a e r a t u t l e c t u s c o n v a l l i s a u c t o r
. \ nF u sce m o l l i s sem i d t e l l u s a u c t o r h e n d r e r i t . " ) 3 >>> f i c h i e r . c l o s
e ( )
```

```
1 >>> f i c h i e r = open ( " t e x t e . t x t " , 'wt ' )
2 >>> f i c h i e r . w r i t e l i n e s ( [ " Lorem ipsum d o l o r s i t amet , c o n s e c t e t u
r a d i p i s c i n g e l i t . \n " , " P e l l e n t e s q u e g r a v i d a e r a t u t l e c t u
s c o n v a l l i s a u c t o r . \n " , " F u sce m o l l i s sem i d t e l l u s a u c t o r h
e n d r e r i t . " ] )
3 >>> f i c h i e r . c l o s e ( )
```

Formats de fichiers

Il existe différents formats standards de stockage de données. Il est recommandé de favoriser ces formats car il existe déjà des modules Python permettant de simplifier leur utilisation. De plus, ces formats sont adoptés par d'autres programmes avec lesquels vous serez peut-être amené à travailler.

Le format CSV

Le fichier *Comma-separated values* (CSV) est un format permettant de stocker des tableaux dans un fichier texte. Chaque ligne est représentée par une ligne de texte et chaque colonne est séparée par un **séparateur** (virgule, point-virgule . . .).

Les champs texte peuvent également être délimités par des guillemets. Lorsqu'un

champ contient lui-même des guillemets, ils sont doublés afin de ne pas être considérés comme début ou fin du champ. Si un champ contient un signe utilisé comme séparateur de colonne (virgule, point-virgule . . .) ou comme séparateur de ligne, les guillemets sont obligatoires afin que ce signe ne soit pas confondu avec un séparateur.

Voici des données présentées sous la forme d'un tableau et d'un fichier CSV *(fig. 5.1)* :

```
1 Nom; Pr énom ; Age
2 " D u boi s " ; " Ma rie "; 2 9
3 " Duval " ; " J u l i e n " " Pa ul " " " ; 4 7 4
J a c q u e t ; B e r na r d ; 5 1
5 Ma r ti n ; " L u c i e ; C l a r a "; 1 4
```

Nom	Prénom	Age
Dubois	Marie	29

Duval	Julien "Paul"	47
Jacquet	Bernard	51
Martin	Lucie ;Clara	14

(a) Données sous la forme d'un fichier CSV (b) Données sous la forme d'un tableau

Figure 5.1 – Exemple de fichier CSV

Le module de Python permet de simplifier l'utilisation des fichiers CSV.

Lire un fichier CSV

Pour lire un fichier CSV, vous devez ouvrir un flux de lecture de fichier et ouvrir à partir de ce flux un lecteur CSV. Pour ignorer la ligne d'en-tête, utilisez :

```
1 im po r t c s v
2 f i c h i e r = open ( "noms . c s v " , " r t " )
3 l e c t e u r C SV = c s v . r e a d e r ( f i c h i e r , d e l i m i t e r=" ; " ) # O u v e r t u r e du
l e c t e u r CSV en l u i f o u r n i s s a n t l e c a r a c t è r e s é p a r a t e u r ( i c i " ; " ) 4 f
o r l i g n e i n l e c t e u r C SV :
5 p r i n t ( l i g n e ) # Exemple a v e c l a 1 e l i g n e du f i c h i e r d ' exem ple : [ 'Nom ' , '
Pr énom ' , 'Age ' ]
6 f i c h i e r . c l o s e ( )
```

Vous obtiendrez en résultat une liste contenant chaque colonne de la ligne en cours.
Vous pouvez modifier le délimiteur de champs texte en définissant la variable :

```
1 im po r t c s v
2 f i c h i e r = open ( "noms . c s v " , " r t " )
3 l e c t e u r C SV = c s v . r e a d e r ( f i c h i e r , d e l i m i t e r=" ; " , q u o t e c h
a r=" ' " ) 4 # Dé f i n i t l ' a p o s t r o p h e comme dé l i m i t e u r de champs t e
x t e 5 f o r l i g n e i n l e c t e u r C SV :
6 p r i n t ( l i g n e )
7 f i c h i e r . c l o s e ( )
```

Vous pouvez également lire les données et obtenir un dictionnaire par ligne contenant

les données en utilisant au lieu de :

```
1 import csv
2 fichier = open("noms.csv","rt")
3 lecteurCSV = csv.DictReader(fichier,delimiter=";")
4 for ligne in lecteurCSV:
5 print(ligne)# Résultat obtenu : {'Age':'29','Nom':'Dubois','Prénom':'Marie'}
6 fichier.close()
```

Écrire un fichier CSV

À l'instar de la lecture, on ouvre un flux d'écriture et on ouvre un écrivain CSV à partir de ce flux :

```
1 import csv
2 fichier = open("annuaire.csv","wt")
3 ecrivainCSV = csv.writer(fichier,delimiter=";")
4 ecrivainCSV.writerow(["Nom","Prénom","Téléphone"])# On écrit la ligne d'en-tête avec le titre des colonnes
```



```
5 ecrivainCSV.writerow(["Dubois","Marie","0198546372"])6
ecrivainCSV.writerow(["Duval","Julien\"Paul\"","0399741052"])7ecrivainCSV.writerow(["Jacquet","Bernard","0200749685"])8ecrivainCSV.writerow(["Martin","Julie;Clara","0399731590"])9fichier.close()
```

Nous obtenons le fichier suivant :

```
1 Nom;Prénom;Téléphone
2 Dubois;Marie;0198546372
3 Duval;"Julien""Paul""";0399741052
4 Jacquet;Bernard;0200749685
5 Martin;"Julie;Clara";0399731590
```

Il est également possible d'écrire le fichier en fournissant un dictionnaire par ligne à condition que chaque dictionnaire possède les mêmes clés. On doit également fournir la liste des clés des dictionnaires avec l'argument :

```
1 import csv
2 bonCommande = [
3 {"produit":"cahier","reference":"F452CP","quantite":41,
        "prixUnitaire":1.6},
4 {"produit":"stylobleu","reference":"D857BL","quantite":18,
        "prixUnitaire":0.95},
5 {"produit":"stylonoir","reference":"D857NO","quantite":18,
        "prixUnitaire":0.95},
```

```
6 {"produit":"équerre","reference":"GF955K","quantite":4
        ,"prixUnitaire":5.10},
7 {"produit":"compas","reference":"RT42AX","quantite":13,"
        prixUnitaire":5.25}
8 ]
9 fichier = open("bon-commande.csv","wt")
10 ecrivainCSV = csv.DictWriter(fichier,delimiter=";",fiel
      dnames= bonCommande[0].keys())
11 ecrivainCSV.writeheader()# On écrit la ligne d'en-tête avec
      cle titre des colonnes
12 for ligne in bonCommande :
13 ecrivainCSV.writerow(ligne)
14 fichier.close()
```

Nous obtenons le fichier suivant :

```
1 reference;quantite;produit;prixUnitaire
2 F452CP;41;cahier;1.6
```



```
3 D857BL;18;stylobleu;0.95
4 D857NO;18;stylonoir;0.95
5 GF955K;4;équerre;5.1
6 RT42AX;13;compas;5.25
```

Par défaut, Python placera les guillemets autour des chaînes contenant des guillemets, une virgule ou un point virgule afin que ceux-ci ne soient pas confondus avec un délimiteur de champs ou le séparateur. Afin que tous les champs soient encadrés par les guillemets, nous allons modifier l'argument pour ou :

```
1 import csv
2 fichier = open("annuaire.csv","wt")
3 ecrivainCSV = csv.writer(fichier,delimiter=";",quotecha
      r="'",quoting=csv.QUOTE_ALL)# quotechar modifie le cara
      ctère délimitant un champ (par défaut :")
4 ecrivainCSV.writerow(["Nom","Prénom","Télé phone"])# On écr
      it la ligne d'en-tête avec le titre des colonnes
5 ecrivainCSV.writerow(["Dubois","Marie","0198546372"])6
ecrivainCSV.writerow(["Duval","Julien\"Paul\"","0399741052
"])7 ecrivainCSV.writerow(["Jacquet","Bernard","
0200749685"])8 ecrivainCSV.writerow(["Martin","Julie;Cla
ra","0399731590"])9 fichier.close()
```

Nous obtenons le fichier suivant :

```
1 'Nom';'Prénom';'Télé phone'
2 'Dubois';'Marie';'0198546372'
3 'Duval';'Julien"Paul"';'0399741052'
```

```
4 ’ Jac q ue t ’ ; ’ Be rna rd ’; ’ 0 2 0 0 7 4 9 6 8 5 ’
5 ’ Ma r tin ’ ; ’ J u l i e ; C la ra ’; ’ 0 3 9 9 7 3 1 5 9 0 ’
```

Le paramètre peut prendre les valeurs suivantes *(table 5.2)*.

Valeur	**Action**
	Met tous les champs entre guillemets.
	Met les guillemets autour des chaînes contenant des guillemets et le séparateur de champs (par défaut).
	Met les guillemets autour des valeurs non-numériques et indique au lecteur de convertir les valeurs non contenues dans les guillemets de les convertir en nombres réels.
	Ne met aucun guillemet.

Table 5.2 – Valeurs de l’argument

Le format JSON

Le format *JavaScript Object Notation* (JSON) est issu de la notation des objets dans le langage JavaScript. Il s’agit aujourd’hui d’un format de données très répandu permettant de stocker des données sous une forme structurée.

Il ne comporte que des associations clés → valeurs (à l’instar des dictionnaires), ainsi que des listes ordonnées de valeurs (comme les listes en Python). Une valeur peut être une autre association clés → valeurs, une liste de valeurs, un entier, un nombre réel, une chaîne de caractères, un booléen ou une valeur nulle. **Sa syntaxe est similaire à celle des dictionnaires Python.**

Voici un exemple de fichier JSON :

```
 1 {
 2 " D ij o n " : {
 3 " nomDepartement " : "Cô t e d ’ Or " ,
 4 " c o d e P o s t a l " : 21000 ,
 5 " p o p u l a t i o n " : {
 6 " 2 0 0 6 ": 151504 ,
 7 " 2 0 1 1 ": 151672 ,
 8 " 2 0 1 4 ": 153668
 9 }
10 } ,
11 " T roy e s " : {
12 " nomDepartement " : " Aube " ,
13 " c o d e P o s t a l " : 10000 ,
14 " p o p u l a t i o n " : {
```

```
15 " 2 0 0 6 ": 61344 ,
16 " 2 0 1 1 ": 60013 ,
17 " 2 0 1 4 ": 60750
18 }
19 }
20 }
```

Il est également possible de compacter un fichier JSON en supprimant les tabulations et les retours à la ligne. On obtient ainsi :

```
1 {" D ij o n " : { " nomDepartement " : " Côt e d ' Or " , " c o d e P o s t a l ": 2 1 0 0 0 , " p o
    p u l a t i o n ": { " 2 0 0 6 ": 1 5 1 5 0 4 , " 2 0 1 1 ": 1 5 1 6 7 2 , " 2 0 1 4 ": 1 5 3 6 6
    8 } } , " T roy e s " : { " nomDepartement " : " Aube " , " c o d e P o s t a l ": 1 0 0 0 0 ,
    " p o p u l a t i o n ": { " 2 0 0 6 ": 6 1 3 4 4 , " 2 0 1 1 ": 6 0 0 1 3 , " 2 0 1 4 ": 6 0 7 5
    0 } } }
```

Pour lire et écrire des fichiers JSON, nous utiliserons le module fourni nativement avec Python.

Lire un fichier JSON

La fonction permet de décoder le texte JSON passé en argument et de le transformer en dictionnaire ou une liste.

```
1 >>> im po r t j s o n
2 >>> f i c h i e r = open ( " v i l l e s . j s o n " , " r t " )
3 >>> v i l l e s = j s o n . l o a d s ( f i c h i e r . r e a d ( ) )
4 >>> p r i n t ( v i l l e s )
5 { ' T roy e s ' : { ' p o p u l a t i o n ' : { ' 2006 ' : 61344 , ' 2011 ' : 60013 , ' 2014 ' : 60750} ,
    ' c o d e P o s t a l ' : 10000 , ' nomDepartement ' : 'Aube ' } , ' D ij o n ' : { ' p o p u l
    a t i o n ' : { ' 2006 ' : 151504 , ' 2011 ' : 151672 , ' 2014 ' : 153668 } , ' c o d e P o s t
    a l ' : 21000 , ' nomDepartement ' : "Côt e d ' Or " }}
6 >>> f i c h i e r . c l o s e ( )
```

Écrire un fichier JSON

On utilise la fonction pour transformer un dictionnaire ou une liste en texte JSON en fournissant en argument la variable à transformer. La variable permet de trier les clés dans l'ordre alphabétique.

```
1 im po r t j s o n
2 q u a n t i t e F o u r n i t u r e s = {" c a h i e r s " : 1 3 4 , " s t y l o s " : { " ro ug e " : 4 1
    , " b l e u " : 7 4 } , "gommes " : 85}
3 f i c h i e r = open ( " q u a n t i t e F o u r n i t u r e s . j s o n " , "wt " )
4 f i c h i e r . w r i t e ( j s o n . dumps ( q u a n t i t e F o u r n i t u r e s ) )
5 f i c h i e r . c l o s e ( )
```

Gestion des erreurs

Lors de l'écriture de vos premiers scripts, vous avez peut-être rencontré ce type de message d'erreurs :

```
1 >>> r e s u l t a t = 42/0
2 T race back ( most r e c e n t c a l l l a s t ) :
3 F i l e "<s t d i n >" , l i n e 1 , i n <module>
4 Z e r o D i v i s i o n E r r o r : d i v i s i o n by z e r o
```

Ce message signale l'erreur ainsi que la ligne à laquelle a été faite cette erreur. Or il est courant que les erreurs ne soient pas des erreurs lors de la programmation mais une mauvaise manipulation de la part de l'utilisateur. Par exemple, vous demandez à l'utilisateur de fournir un nombre et celui-ci vous fournit un texte ou que le fichier, que vous cherchez à lire, n'existe pas.

Il faut alors gérer ce type d'erreurs afin d'éviter une interruption involontaire de notre application. Pour cela, nous utiliserons les structures .

Voici un exemple sur lequel nous allons ajouter un mécanisme de gestion d'erreurs :

```
1 age = i n p u t ( " Quel e s t v o t r e age : " )
2 age = i n t ( age )
```

Voici l'erreur obtenue si la chaîne de caractères n'est pas un nombre :

```
1 >>> age = i n p u t ( " Quel e s t v o t r e age : " )
2 Quel e s t v o t r e age : A l a i n
3 >>> age = i n t ( age )
4 T race back ( most r e c e n t c a l l l a s t ) :
5 F i l e "<s t d i n >" , l i n e 1 , i n <module>
6 V a l u e E r r o r : i n v a l i d l i t e r a l f o r i n t ( ) w i t h ba s e 1 0: ' A l a i n '
```

La structure se présente ainsi :

```
1 t r y :
2 # La p o r t i o n de code à t e s t e r
3 e x c e p t :
4 # Que f a i r e en c a s d ' e r r e u r
```

Nous obtenons donc pour notre exemple :

```
1 >>> age = i n p u t ( " Quel e s t v o t r e age : " )
2 Quel e s t v o t r e age : A l a i n
3 >>> t r y :
4 . . . age = i n t ( age )
```

```
5 ...except:
6 ...print("Erreurlorsdelasaisie.")
7 ...
8 Erreurlorsdelasaisie.
```

Il est possible d'identifier l'erreur et d'effectuer une action en conséquence. Nous pouvons donc chaîner les instructions **en fournissant le type d'erreur**. Le bloc (optionnel) est exécuté s'il n'y a eu aucune erreur.

```
1 try:
2 quantiteBoites=quantitePieces/nombresPiecesParBoi
tes 3 except TypeError: # Type incompatible avec l'opération 4
print("Au moins une des variables n'est pas un nombre.") 5 except
NameError: # Variable non définie
6 print("Au moins une des variables n'est pas définie.") 7 ex
cept ZeroDivisionError: # Division par 0
8 print("Le nombres de pièces par boites est égal à 0.") 9 els
e:
10 print("Il faut commander" + str(quantiteBoites) + "boîtes.")
```

 Le bloc (optionnel) est exécuté dans tous les cas (s'il y a eu des erreurs ou non).

Enfin, l'instruction ne fait rien : elle permet de laisser un bloc vide ce qui est utile pour les exceptions.

Gérer les fichiers

Python fournit deux modules permettant de gérer les fichiers et les répertoires. Le module permet de lister des fichiers et des répertoires, d'effectuer des opérations sur les URI [1]. Le module permet de copier, déplacer, supprimer des éléments sur les systèmes de fichiers.

Dans ce chapitre, nous travaillerons sur des systèmes de fichiers Unix (GNU/Linux, MacOS . . .).

Les chemins de fichiers

Nous allons étudier les fonctions permettant de manipuler les chemins de fichiers ou de répertoires du module . La fonction renvoie le nom du fichier de l'adresse fourni en argument. À l'inverse, la fonction renvoie le chemin jusqu'au fichier, sans le nom du fichier. **Ces fonctions fonctionnent même si le fichier n'existe pas.**

```
1 >>> import os.path
2 >>> chemin = "/tmp/dir/dir2/monFichier.txt"
3 >>> print(os.path.basename(chemin))
4 monFichier.txt
5 >>> print(os.path.dirname(chemin))
6 /tmp/dir/dir2
```

Différentier les fichiers et les répertoires

La fonction renvoie True si le fichier ou le répertoire fournis en argument existent. Les fonctions et permettent respectivement de vérifier si le chemin mène à un fichier ou un répertoire et renvoie True si c'est le cas.

```
>>> import os.path
>>> chemin = "/tmp/dir/dir2/monFichier.txt"
>>> print(os.path.exists(chemin))
True
>>> print(os.path.isfile(chemin))
True
>>> print(os.path.isdir(chemin))
False
>>> print(os.path.isdir(os.path.dirname(chemin)))
True
```

1. *Uniform Resource Identifier*, la chaîne de caractères permettant d'identifier un élément sur un système de fichiers. On parle couramment de chemin.

Lister le contenu d'un répertoire

La fonction du module retourne le contenu du répertoire passé en argument sans distinction entre les fichiers et les répertoires.

```
>>> import os.path
>>> print(os.listdir("/tmp/dir"))
['villes.json','quantiteFournitures.json','dir2']
```

Copier un fichier ou un répertoire

Il existe deux méthodes dans le module permettant d'effectuer une copie. La fonction permet de copier un fichier, alors que en fait de même avec les répertoires.

```
import shutil
shutil.copytree("/tmp/dir/dir2","/tmp/dir/dir3")
shutil.copy("/tmp/dir/dir2/monFichier.txt","/tmp/dir/exemple.txt")
```

Déplacer un fichier ou un répertoire

La fonction du module permet de déplacer un fichier ou un répertoire. Cela peut également servir à renommer le fichier ou le répertoire.

```
import shutil
shutil.move("/tmp/dir/dir3","/tmp/dir/perso")
```

Supprimer un fichier ou un répertoire

La méthode du module et la fonction du module permettent respectivement de supprimer un fichier ou un répertoire.

```
1 im po r t os , s h u t i l
2 o s . remove ( " /tmp/ d i r / exem ple . t x t " )
3 s h u t i l . rm t r e e ( " /tmp/ d i r / p e r s o " )
```

Sauvegarder des variables

Le module permet de sérialiser des variables quelles qu'elles soient vers un fichier ouvert en flux binaire et de les restaurer. Cela équivaut à sauvegarder et restaurer l'état des variables.

La fonction permet d'exporter une variable vers un fichier et la fonction retourne la variable lue depuis le fichier.

L'ordre de sauvegarde et de restauration des variables doit être identique.

```
1 >>> im po r t p i c k l e
2 >>> t e x t e = "É c r i t par A n to i n e de S a i n t −Exup é r y "
3 >>> q u a n t i t e F o u r n i t u r e s = {" c a h i e r s " : 1 3 4 , " s t y l o s " : { " ro ug e " : 4
    1 , " b l e u " : 7 4 } , "gommes " : 85}
4 >>> f o u r n i t u r e s = [ " c a h i e r " , " c r a yo n " , " s t y l o " , " t r o u s s e " ,
"gomme" ] 5 >>> f i c h i e r S a u v e g a r d e = open ( " do n n ee s " , "wb" )
6 >>> p i c k l e . dump ( t e x t e , f i c h i e r S a u v e g a r d e )
7 >>> p i c k l e . dump ( q u a n t i t e F o u r n i t u r e s , f i c h i e r S a u v
e g a r d e ) 8 >>> p i c k l e . dump ( f o u r n i t u r e s , f i c h i e r S a u v e
g a r d e )
9 >>> f i c h i e r S a u v e g a r d e . c l o s e ( )
```

```
1 >>> im po r t p i c k l e
2 >>> f i c h i e r S a u v e g a r d e = open ( " do n n ee s " , " r b " )
3 >>> t e x t e = p i c k l e . l o a d ( f i c h i e r S a u v e g a r d e )
4 >>> q u a n t i t e F o u r n i t u r e s = p i c k l e . l o a d ( f i c h i e r S a u v
e g a r d e ) 5 >>> f o u r n i t u r e s = p i c k l e . l o a d ( f i c h i e r S a u v e
g a r d e )
6 >>> f i c h i e r S a u v e g a r d e . c l o s e ( )
7 >>> p r i n t ( t e x t e )
8 É c r i t par A n to i n e de S a i n t−Exup é r y
9 >>> p r i n t ( q u a n t i t e F o u r n i t u r e s )
10 { ' s t y l o s ' : { ' b l e u ' : 74 , ' ro ug e ' : 41} , ' gommes ' : 85 , ' c a h i e r s ' : 134} 11
>>> p r i n t ( f o u r n i t u r e s )
```

12 ['cahier','crayon','stylo','trousse','gomme']

Exercices

N'oubliez pas le shebang et commentez si nécessaire.

1. Écrivez un programme permettant à un utilisateur de deviner un mot choisi au hasard dans un fichier dans lequel chaque ligne comporte un mot en capitale. L'utilisateur a 7 chances pour découvrir le mot en proposant une lettre à chaque fois. Si la lettre proposée n'est pas dans le mot, une chance lui est retirée.
   Exemple :

   Vous pouvez télécharger un fichier de mots ici :

2. Écrivez un programme permettant de chiffrer et déchiffrer un fichier texte à l'aide du chif frement par décalage dans lequel chaque lettre est remplacée par une autre à distance fixe choisie par l'utilisateur. Par exemple, si la distance choisie est de 4, un A est remplacé par un D, un B par un E, un C par un F . . . Le résultat sera écrit dans un fichier texte.
   Texte en clair :
   Texte chiffré (distance 7) :

3. Écrivez un programme permettant à partir d'un fichier texte en français d'en déduire la fré quence d'apparition des lettres qu'il contient. Le résultat de cette analyse sera consigné dans un fichier JSON. Pour des statistiques fiables, prenez un texte assez long. Vous pouvez utiliser une copie de *Zadig*, écrit par Voltaire, disponible ici :

4. Le message suivant a été chiffré à l'aide de la technique de chiffrement par décalage :

```
1 HFCMSG GS GWHIS ROBG ZS UFOBR SGH RS ZO TFOBQS OI
2 QSBHFS RI RSDOFHSASBH RS Z OIPS RCBH SZZS SGH ZS
3 QVST ZWSI SH O Z CISGH RS ZO FSUWCB UFOBR SGH ZO
4 QCAAIBS G SHSBR ROBG ZO DZOWBS RS QVOADOUBS
5 QFOMSIGS O DFCLWAWHS RI DOMG R CHVS SH RI DOMG R
6 OFAOBQS QSHHS JWZZS RS DZOWBS OZZIJWOZS G SHOPZWH
7 ROBG ZO JOZZSS RS ZO GSWBS
```

À l'aide des statistiques d'apparition des lettres issues de l'exercice précédent, déduisez le message en clair du texte ci-dessus.

5. Écrivez un programme permettant de calculer la moyenne d'un étudiant dont les notes sont consignées dans un fichier CSV. Chaque colonne correspond à une matière. Vous devrez écrire un fichier JSON consignant ces moyennes ainsi que la moyenne générale. Toutes les notes et les matières sont au cœfficient 1. Un exemple de fichier CSV est disponible ici :

Solutions page 146.

Chapitre 6

Interagir avec les bases de données

Ce chapitre aborde les bases de données SQL. Si toutefois ce type de base de données vous est inconnu, un chapitre permettant de vous y introduire est disponible en annexe A, page 119.

Peu à peu, nos programmes manipuleront un très grand nombre de données et nécessiteront un système plus performant pour stocker et lire ces données. Pour cela, nous ferons appel à des **bases de données**. Les bases de données permettent de stocker de grands volumes de données sous une forme normalisée et qui peut être utilisée par plusieurs programmes différents. Nous utiliserons dans ce cours deux SGBD gratuits :

MariaDB Copie de MySQL, disponible gratuitement et sous licence libre (GPL). Le couple MariaDB et MySQL fait partie des SGBD les plus utilisés et les plus massivement déployés. **SQLite3** Il s'agit d'une bibliothèque écrite en C dans le domaine public permettant d'interagir avec des bases de données stockées dans des fichiers. À l'inverse de MariaDB, il ne repose pas sur une architecture client/serveur. Ce système est particulièrement adapté pour les petites bases de données stockées localement, comme alternative aux fichiers texte.

Utiliser une base de données SQLite3

Créer la base et insérer des données

Nous allons tout d'abord importer le module . Dans l'exemple qui suit, nous allons créer une base de données avec une table contenant un répertoire téléphonique contenant le nom, le prénom, l'adresse et le numéro de téléphone fixe des contacts. Nous verrons comment insérer quelques enregistrements.

```
1 im po r t s q l i t e 3
2 baseDeDonnees = s q l i t e 3 . co n n e c t ( ' c o n t a c t s . db ' )
3 c u r s e u r = baseDeDonnees . c u r s o r ( )
4 c u r s e u r . e x e c u t e ( "CREATE TABLE Co n t a c t s ( i d INTEGER PRIMARY
      KEY AUTOINCREMENT, nom TEXT NOT NULL , prenom TEXT NOT NULL , a d r
      e s s e TEXT NOT NULL , t e l e p h o n e F i x e TEXT) " ) # Cr é a t i o n de l a ba
      s e de donn é e s
5 baseDeDonnees . commit ( ) # On e n v o i e l a r e q u ê t e SQL
```



```
6 c u r s e u r . e x e c u t e ( "INSERT INTO Co n t a c t s (nom , prenom , a d r e s s e , t
      e l e p h o n e F i x e ) VALUES ( ? , ? , ? , ? ) " , ( " Dupont " , " Pa ul " , " 15 r u e L
```

```
ouis Pasteur 10000 Troyes " , " 0325997452 " ) ) # On ajoute un enregistrement
7 baseDeDonnees . commit ( )
8 baseDeDonnees . close ( )
```

Il est possible d'ajouter un enregistrement depuis un dictionnaire. Dans l'exemple, on ajoute plusieurs enregistrements avec une boucle :

```
1 import sqlite3
2 baseDeDonnees = sqlite3 . connect ( 'contacts . db ' )
3 curseur = baseDeDonnees . cursor ( )
4 personnes = [
 5 {"nom" : " Chabot " , " prenom" : " Martin " , " adresse " : " 18 rue Général Leclerc 13600 La Ciotat " , " telephoneFixe " : " 0499506373 " } , 6 {"nom" : " Delbois " , " prenom" : " Julie " , " adresse " : " 35 rue du Château 77176 Savigny le Temple " , " telephoneFixe " : " 0199836074 " } , 7 {"nom" : " Rivard " , " prenom" : " Christelle " , " adresse " : " 83 rue de Qué bec 83400 Hyères " , " telephoneFixe " : " 0499687013 " }
8 ]
9 for contact in personnes :
10 curseur . execute ( "INSERT INTO Contacts (nom , prenom , adresse , telephoneFixe ) VALUES ( : nom , : prenom , : adresse , : telephoneFixe ) " , contact ) # On ajoute un enregistrement depuis un dictionnaire
11 baseDeDonnees . commit ( )
12 idDernierEnregistrement = curseur . lastrowid # Récupère l' ID de la dernière ligne insérée .
13 baseDeDonnees . close ( )
```

L'exemple suivant illustre comment modifier des données :

```
1 import sqlite3
2 baseDeDonnees = sqlite3 . connect ( 'contacts . db ' )
3 curseur = baseDeDonnees . cursor ( )
4 curseur . execute ( "UPDATE Contacts SET telephoneFixe = ? WHERE id = ? " , ( " 0598635076 " , 2 ) )
5 baseDeDonnees . commit ( )
6 baseDeDonnees . close ( )
```


Récupérer des données

Pour récupérer les données, il est possible de récupérer le premier résultat avec ou de retourner tous les résultats avec . Voici un premier exemple utilisant :

```
1 >>> im po r t s q l i t e 3
2 >>> baseDeDonnees = s q l i t e 3 . co n n e c t ( ' c o n t a c t s . db ' )
3 >>> c u r s e u r = baseDeDonnees . c u r s o r ( )
4 >>> c u r s e u r . e x e c u t e ( "SELECT nom , prenom , t e l e p h o n e F i x e FROM Co
      n t a c t s WHERE i d = ? " , ( " 2 " , ) )
5 >>> c o n t a c t = c u r s e u r . f e t c h o n e ( )
6 >>> p r i n t ( c o n t a c t )
7 ( ' Chabot ' , ' Ma r ti n ' , ' 0598635076 ' )
8 >>> baseDeDonnees . c l o s e ( )
```

Dans l'exemple ci-dessus, la variable contient un tuple avec les valeurs du premier enregistrement retourné par la requête.

Voyons à présent comment récupérer plusieurs enregistrements avec la commande :

```
1 >>> im po r t s q l i t e 3
2 >>> baseDeDonnees = s q l i t e 3 . co n n e c t ( ' c o n t a c t s . db ' )
3 >>> c u r s e u r = baseDeDonnees . c u r s o r ( )
4 >>> c u r s e u r . e x e c u t e ( "SELECT nom , prenom , t e l e p h o n e F i x e FROM Co
      n t a c t s " )
5 >>> f o r c o n t a c t i n c u r s e u r . f e t c h a l l ( ) :
6 . . . p r i n t ( c o n t a c t )
7 . . .
8 ( ' Dupont ' , ' Pa ul ' , ' 0325997452 ' )
9 ( ' Chabot ' , ' Ma r ti n ' , ' 0598635076 ' )
10 ( ' D e l b o i s ' , ' J u l i e ' , ' 0199836074 ' )
11 ( ' R i v a r d ' , ' C h r i s t e l l e ' , ' 0499687013 ' )
12 >>> baseDeDonnees . c l o s e ( )
```

Utiliser une base de données MariaDB/MySQL

Pour cette partie, vous devez avoir installé le paquet .

Créer la base et insérer des données

Nous allons utiliser le module pour nous connecter au serveur MariaDB ou MySQL et créer notre base de données permettant de stocker un catalogue de produits.

```
1 im po r t my s ql . c o n n e c t o r
```



```
2 baseDeDonnees = my s ql . c o n n e c t o r . co n n e c t ( h o s t=" l o c a l h o s t " ,
u s e r=" c a t a l o g u e " , pa s swo r d=" JieTh8Th " , d a t a b a s e=" C a t a log u e
      " )
3 c u r s o r = baseDeDonnees . c u r s o r ( )
4 c u r s o r . e x e c u t e ( "CREATE TABLE P r o d u i t s ( r e f e r e n c e CHAR( 5 )
```

```
    NOT NULL PRIMARY KEY, nom TINYTEXT NOT NULL , p r i x FLOAT NOT
    NULL)ENGINE= InnoDB DEFAULT CHARSET=u tf 8 ; " )
5 baseDeDonnees . c l o s e ( )
```

Nous allons à présent insérer des données dans cette table.

```
1 im po r t my s ql . c o n n e c t o r
2 baseDeDonnees = my s ql . c o n n e c t o r . co n n e c t ( h o s t=" l o c a l h o s t " ,
u s e r=" c a t a l o g u e " , pa s swo r d=" JieTh8Th " , d a t a b a s e=" C a t a log u e
                                                    " )
3 c u r s e u r = baseDeDonnees . c u r s o r ( )
4 c u r s e u r . e x e c u t e ( "INSERT INTO P r o d u i t s ( r e f e r e n c e , nom , p r i x )
VALUES (%s , %s , %s ) " , ( "ARB42" , "Canap é deux p l a c e s n o i r " , 1 9 9. 9 9 ) ) 5
baseDeDonnees . commit ( )
6 baseDeDonnees . c l o s e ( )
```

Il est également possible d'insérer des données depuis un dictionnaire :

```
1 im po r t my s ql . c o n n e c t o r
2 baseDeDonnees = my s ql . c o n n e c t o r . co n n e c t ( h o s t=" l o c a l h o s t " ,
u s e r=" c a t a l o g u e " , pa s swo r d=" JieTh8Th " , d a t a b a s e=" C a t a log u e
                                                    " )
3 c u r s e u r = baseDeDonnees . c u r s o r ( )
4 p r o d u i t s = [
5 {" r e f e r e n c e " : " EIS3P" , "nom" : " C h a i s e de s a l l e à manger " , " p r i x " : 2
        5 } ,
6 {" r e f e r e n c e " : "BA9KI " , "nom" : "Commode b l a n c h e " , " p r i x " : 1 3 9 . 9 0
} , 7 {" r e f e r e n c e " : "OI4HE" , "nom" : " Ta ble b a s s e " , " p r i x " : 2 4 . 9 5 } , 8 {"
r e f e r e n c e " : "IOM9X" , "nom" : " L i t d o u b l e " , " p r i x " : 6 9 9. 9 9 } 9 ]
10 f o r f i c h e i n p r o d u i t s :
11 c u r s e u r . e x e c u t e ( "INSERT INTO P r o d u i t s ( r e f e r e n c e , nom , p r i
    x ) VALUES (%( r e f e r e n c e )s , %(nom)s , %( p r i x ) s ) " , f i c h e )
12 baseDeDonnees . commit ( )
13 baseDeDonnees . c l o s e ( )
```

Récupérer des données

À l'instar de SQLite, on peut utiliser pour récupérer le premier résultat ou retourner tous les résultats avec . Voici comment récupérer le premier résultat d'une requête .

```
1 >>> im po r t my s ql . c o n n e c t o r
```



```
2 >>> baseDeDonnees = my s ql . c o n n e c t o r . co n n e c t ( h o s t=" l o c a l h o s t " ,
u s e r=" c a t a l o g u e " , pa s swo r d=" JieTh8Th " , d a t a b a s e=" C a t a log u e " )
3 >>> c u r s e u r = baseDeDonnees . c u r s o r ( )
4 >>> c u r s e u r . e x e c u t e ( "SELECT r e f e r e n c e , nom , p r i x FROM P r o d
```

```
uits")5 >>> print(curseur.fetchone())
6 ('ARB42', 'Canapé deux places noir', 199.99)
7 >>> baseDeDonnees.close()
```

On peut retourner tous les résultats avec :

```
1 >>> import mysql.connector
2 >>> baseDeDonnees = mysql.connector.connect(host="localhost",
  user="catalogue", password="JieTh8Th", database="Catalogue")
3 >>> curseur = baseDeDonnees.cursor()
4 >>> curseur.execute("SELECT reference, nom, prix FROM Produ
its")5 >>> for ligne in curseur.fetchall():
6 ... print(ligne)
7 ...
8 ('ARB42', 'Canapé deux places noir', 199.99)
9 ('BA9KI', 'Commode blanche', 139.9)
10 ('EIS3P', 'Chaise de salle à manger', 25.0)
11 ('IOM9X', 'Litdouble', 699.99)
12 ('OI4HE', 'Table basse', 24.95)
13 >>> baseDeDonnees.close()
```

Exercices

N'oubliez pas le shebang et commentez si nécessaire.

Vous êtes nouvellement embauché dans le secrétariat de scolarité d'une université. Votre travail est d'optimiser la gestion des étudiants, des enseignants, des matières enseignées, des inscriptions des étudiants à ces dernières et des résultats obtenus. À votre arrivée, une collègue vous fournit les fichiers tableurs permettant d'accomplir ces tâches (au format CSV) :

Liste des étudiants

Liste des enseignants

Liste des matières

Liste des inscriptions

Relevé des résultats

Les exercices suivant permettront d'effectuer cela en scindant le travail en différentes sous-tâches. Chaque sous-tâche fera l'objet d'un nouveau programme. Tous ces programmes utiliseront la même base de données SQLite3. Toutes les moyennes auront deux décimales.

1. Écrivez un programme permettant de créer une base de données SQLite3 nommée et de créer la structure de table adaptée au stockage des données. Importez le contenu des fichiers CSV dans cette base.

2. Écrivez un programme permettant de générer des statistiques pour l'université au format JSON dont nous stockerons les moyennes par matière, la moyenne maximale et minimale par matière, le nombre d'étudiants inscrits par matière, la moyenne de toutes les matières et le nombre d'étudiants par département (les deux premiers nombres du code postal).

3. Écrivez un programme permettant de générer un bulletin de notes par étudiant sous la forme d'un courrier stocké dans un fichier texte individuel. Chaque fichier aura pour nom le nom et le prénom de l'étudiant, séparés par un trait d'union () et pour extension et sera stocké dans un dossier créé pour cela. Chaque courrier adoptera ce modèle :

> Université Claude Chappe
> 15 avenue de Moulincourbe
> 28094 Clairecombe
>
> Lionel Paulin
> 48 Ruelle de Locvaux
> 74019 Mivran
>
> Madame, Monsieur,
>
> Veuillez trouver dans le récapitulatif ci-dessous les résultats de vos examens. Matière Moyenne
> AI90 16.93
> PQ84 12.7
> UE21 12.0
> VO38 12.49
> XO83 13.05
> ZI51 16.33
> Moyenne générale 13.92
>
> Ce document constitue les résultats officiels. Pour toute contestation, contactez le service scolarité.

4. Écrivez un programme permettant l'inscription d'un nouveau étudiant à l'université et son inscription aux matières.

5. Écrivez un programme permettant la saisie des notes obtenues aux examens.

Solutions page 150.

Chapitre 7

La programmation réseau

Nos programmes peuvent à présent effectuer des tâches complexes et peuvent s'interfacer entre eux par le biais de fichiers ou de bases de données. Voyons à présent comment faire communiquer plusieurs programmes fonctionnant sur des ordinateurs différents *via* le réseau informatique. L'objectif de ce chapitre est d'aborder la communication entre plusieurs ordinateurs avec le mécanisme de sockets.

Un *socket*, que nous pouvons traduire par connecteur réseau, est une interface aux services réseaux offerte par le système d'exploitation permettant d'exploiter facilement le réseau. Cela permet d'initier une session TCP, d'envoyer et de recevoir des données par cette session. Nous utiliserons pour ce chapitre le module .

Nous travaillerons avec deux scripts, le serveur permettant d'écouter les demandes des clients et d'y répondre. Le client se connectera sur le serveur pour demander le service. Il est possible d'exécuter à la fois le client et le serveur sur un même ordinateur. Pour cela, il vous suffit de renseigner comme adresse IP pour la partie client.

Créer un serveur socket

Nous allons commencer par construire une application serveur très simple qui reçoit les

connexions clients sur le port désigné, envoie un texte lors de la connexion, affiche ce que le client lui envoie et ferme la connexion.

```
1 #! / u s r / b i n / env py t ho n3
2 im po r t s o c k e t
3 s e r v e u r = s o c k e t . s o c k e t ( s o c k e t . AF_INET , s o c k e t
.SOCK_STREAM) 4 s e r v e u r . b i n d ( ( ' ' , 50000 ) ) # Éco u t e s u r l e p o r
t 50000 5 s e r v e u r . l i s t e n ( 5 )
6 w h i l e True :
7 c l i e n t , i n f o s C l i e n t = s e r v e u r . a c c e p t ( )
8 p r i n t ( " C l i e n t co n n e c t é . A d r e s s e " + i n f o s C l i e n t [ 0 ] ) 9 r e q u
e t e = c l i e n t . r e c v (255 ) # Re ç o i t 255 o c t e t s . Vous po uvez c h a ng e r
po u r r e c e v o i r p l u s de donn é e s
```



```
10 p r i n t ( r e q u e t e . deco de ( " u tf −8" ) )
11 r e p o n s e = " Bonjou r , j e s u i s l e s e r v e u r "
12 c l i e n t . se n d ( r e p o n s e . e nco de ( " u tf −8" ) )
13 p r i n t ( " Co n nexio n fe rm é e " )
14 c l i e n t . c l o s e ( )
15 s e r v e u r . c l o s e ( )
```

Vous remarquerez la présence de l'instruction . Cette indication demande à Python de convertir la chaîne de caractères en flux binaire UTF-8 pour permettre son émission sur le réseau. L'instruction permet d'effectuer l'opération inverse. Nous ne pouvons pas tester le programme tant que nous n'avons pas écrit la partie cliente.

Créer un client socket

L'application cliente présentée ici permet de se connecter à un serveur spécifié sur un port désigné. Une fois la connexion établie, il enverra un message au serveur et affichera le message que le serveur lui enverra en retour.

```
1 #! / u s r / b i n / env py t ho n3
2 im po r t s o c k e t
3 a d r e s s e I P = " 1 2 7 . 0 . 0 . 1 " # I c i , l e p o s t e l o c a l
4 p o r t = 50000 # Se c o n n e c t e r s u r l e p o r t 50000
5 c l i e n t = s o c k e t . s o c k e t ( s o c k e t . AF_INET , s o c k e t
.SOCK_STREAM) 6 c l i e n t . co n n e c t ( ( a d r e s s e I P , p o r t ) )
7 p r i n t ( " Connec t é au s e r v e u r " )
8 c l i e n t . se n d ( " Bonjou r , j e s u i s l e c l i e n t " . e nco de ( " u tf −8" ) )
9 r e p o n s e = c l i e n t . r e c v (255 )
10 p r i n t ( r e p o n s e . deco de ( " u tf −8" ) )
11 p r i n t ( " Co n nexio n fe rm é e " )
12 c l i e n t . c l o s e ( )
```

Cela fonctionne mais le serveur présente un défaut, il ne peux pas gérer plusieurs clients à la fois. Nous allons y remédier dans la suite de ce chapitre.

L'exécution de fonctions en parallèle : le multithread

Le multithread permet l'exécution de plusieurs opérations simultanément sur les mêmes res sources matérielles (ici, l'ordinateur). Les différents threads sont traités à tour de rôle par l'ordinateur pendant un temps très court ce qui donne cette impression d'exécution parallèle.

Nous abordons cette technique pour pouvoir élaborer un serveur pouvant gérer plusieurs connexions client en même temps. Tout d'abord, nous allons nous familiariser avec le module qui met

en œuvre le multithread pour les fonctions et les objets Python. Voici un exemple simple montrant le fonctionnement de cette technique :

```
1 #! / u s r / b i n / env py t ho n3
2 im po r t t h r e a d i n g
3 d ef comp teu r ( nomThread ) :
4 f o r i i n r a ng e ( 3 ) :
5 p r i n t ( nomThread + " : " + s t r ( i ) )
6 t h r ea dA = t h r e a d i n g . Th read ( None , compteu r , None , ( " Th read A" , ) , {} ) 7 t
h rea dB = t h r e a d i n g . Th read ( None , compteu r , None , ( " Th read B" , ) , {} ) 8 t h r
ea dA . s t a r t ( )
9 t h rea dB . s t a r t ( )
```

Voici un des résultats possibles lors de l'exécution du script ci-dessus. On observe que l'affichage du thread A et B sont confondus :

```
1 Th read A : 0
2 Th read A : 1
3 Th read B : 0
4 Th read B : 1
5 Th read A : 2
6 Th read B : 2
```

Nous allons détailler la fonction créant le thread : dont voici les arguments :

groupe Doit être à , réservé à un usage futur.

cible Le nom de la fonction à exécuter.

nom Le nom du thread (facultatif, peut être défini à).

arguments Un tuple donnant les arguments de la fonction cible.

dictionnaireArguments Un dictionnaire donnant les arguments de la fonction cible. Avec

l'exemple ci-dessus, on utiliserait .

Créer un serveur socket acceptant plusieurs clients

Voici donc la combinaison du serveur de socket vu précédemment et la technique du multithrea ding pour obtenir un serveur plus complexe créant un nouveau thread à chaque client connecté. Nous avons apporté une petite modification au client car désormais, le client demande à l'utilisateur de saisir un message qui sera affiché sur le serveur. Ce dernier répondra à tous les messages du client jusqu'à ce que l'utilisateur saisisse le mot sur le client ce qui termine la connexion. Voici le script serveur :

```
1 #! / u s r / b i n / env py t ho n3
2 im po r t s o c k e t
```



```
3 im po r t t h r e a d i n g
4 t h r e a d s C l i e n t s = [ ]
5 d ef i n s t a n c e S e r v e u r ( c l i e n t , i n f o s C l i e n t ) :
6 a d r e s s e I P = i n f o s C l i e n t [ 0 ]
7 p o r t = s t r ( i n f o s C l i e n t [ 1 ] )
8 p r i n t ( " I n s t a n c e de s e r v e u r p r ê t po u r " + a d r e s s e I P + " : " + p o r t ) 9
me s sage = " "
10 w h i l e me s sage . u p p e r ( ) != " FIN" :
11 me s sage = c l i e n t . r e c v (255 ) . deco de ( " u tf −8" )
12 p r i n t ( " Message r e çu du c l i e n t " + a d r e s s e I P + " : " + p o r t + " : " + me s sage
                )
13 c l i e n t . se n d ( " Message r e çu" . e nco de ( " u tf −8" ) )
14 p r i n t ( " Co n nexio n fe rm é e a v e c " + a d r e s s e I P + " : " + p o r t ) 15
c l i e n t . c l o s e ( )
16 s e r v e u r = s o c k e t . s o c k e t ( s o c k e t . AF_INET , s o c k e t
.SOCK_STREAM) 17 s e r v e u r . b i n d ( ( ' ' , 50000 ) ) # Éco u t e s u r l e p o
r t 50000 18 s e r v e u r . l i s t e n ( 5 )
19 w h i l e True :
20 c l i e n t , i n f o s C l i e n t = s e r v e u r . a c c e p t ( )
21 t h r e a d s C l i e n t s . append ( t h r e a d i n g . Th read ( None , i n s t a n c e S e
                r v e u r , None , ( c l i e n t , i n f o s C l i e n t ) , {} ) )
22 t h r e a d s C l i e n t s [ −1]. s t a r t ( )
23 s e r v e u r . c l o s e ( )
```

Et voici le nouveau script client :

```
1 #! / u s r / b i n / env py t ho n3
2 im po r t s o c k e t
3 a d r e s s e I P = " 1 2 7 . 0 . 0 . 1 " # I c i , l e p o s t e l o c a l
4 p o r t = 50000 # Se c o n n e c t e r s u r l e p o r t 50000
5 c l i e n t = s o c k e t . s o c k e t ( s o c k e t . AF_INET , s o c k e t
```

```
.SOCK_STREAM) 6 c l i e n t . co n n e c t ( ( a d r e s s e IP , p o r t ) )
7 p r i n t ( " Connec t é au s e r v e u r " )
8 p r i n t ( "Tapez FIN po u r t e r m i n e r l a c o n v e r s a t i o n . " )
9 me s sage = " "
10 w h i l e me s sage . u p p e r ( ) != " FIN" :
11 me s sage = i n p u t ( "> " )
12 c l i e n t . se n d ( me s sage . e nco de ( " u tf −8" ) )
13 r e p o n s e = c l i e n t . r e c v (255 )
14 p r i n t ( r e p o n s e . deco de ( " u tf −8" ) )
15 p r i n t ( " Co n nexio n fe rm é e " )
16 c l i e n t . c l o s e ( )
```


À chaque nouvelle connexion d'un client, le serveur crée un thread dédié à ce client, ce qui permet au programme principal d'attendre la connexion d'un nouveau client. Chaque client peut alors avoir une conversation avec le serveur sans bloquer les autres conversations.

Voici un exemple de l'affichage sur le serveur :

```
1 I n s t a n c e de s e r v e u r p r ê t po u r 1 2 7 . 0 . 0 . 1 : 5 7 2 8 2
2 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 2 8 2 : Je s u i s c l i e n t 1
3 I n s t a n c e de s e r v e u r p r ê t po u r 1 2 7 . 0 . 0 . 1 : 5 7 3 6 5
4 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 3 6 5 : Je s u i s c l i e n t 2
5 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 2 8 2 : On
6 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 3 6 5 : peux
7 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 2 8 2 : e n v o y e r
8 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 3 6 5 : d e s
9 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 2 8 2 : me s sage s
10 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 3 6 5 : e n s em bl e
11 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 2 8 2 : FIN
12 Co n nexio n fe rm é e a v e c 1 2 7 . 0 . 0 . 1 : 5 7 2 8 2
13 Message r e ç u du c l i e n t 1 2 7 . 0 . 0 . 1 : 5 7 3 6 5 : FIN
14 Co n nexio n fe rm é e a v e c 1 2 7 . 0 . 0 . 1 : 5 7 3 6 5
```

Créer un serveur Web

Nous allons ici utiliser la bibliothèque pour créer rapidement un serveur Web capable de servir des fichiers à un navigateur Web. Le script ci-dessous montre comment créer cela en spécifiant le numéro de port sur lequel notre serveur Web va écouter et quel dossier il va servir.

```
1 im po r t h t t p . s e r v e r
2 p o r t E c o u t e = 80 # Po r t Web pa r dé f a u t
```

```
3 adresseServeur=("",portEcoute)
4 serveur=http.server.HTTPServer
5 handler=http.server.CGIHTTPRequestHandler
6 handler.cgi_directories=["/tmp"]# On sert le dossier/tmp 7
print("Serveur actif sur le port",portEcoute)
8 httpd=serveur(adresseServeur,handler)
9 httpd.serve_forever()
```

Ce serveur retournera par défaut le fichier du dossier servi. Si ce fichier n'existe pas, le serveur retournera la liste des fichiers du dossier.

Utiliser des services Web

De plus en plus de programmes utilisent et proposent aujourd'hui des interfaces de programma tion nommées API [1]. Cela permet de standardiser l'interaction entre les différentes applications et de découper les différentes fonctionnalités d'une application en divers modules qui communiquent ensemble avec ces interfaces.

Nous allons voir comment utiliser des services Web proposés pour enrichir nos applications Python en utilisant le module . Notre premier exemple sera d'afficher la position de la Station Spatiale Internationale (ISS) qui nous est fournie par l'API qui nous renvoie un texte au format JSON sous cette forme :

```
1 {
2 "iss_position":{
3 "longitude":"-100.8325",
4 "latitude":"-12.0631"
5 },
6 "time stamp": 1493971107,
7 "message":"success"
8 }
```

Voici le code source permettant de récupérer la donnée et en récupérer les parties utiles :

```
1 import urllib.request
2 import json
3 requete=urllib.request.Request('http://api.open-notify.org/
    iss-now.json')# La requête de l'API
4 reponse=urllib.request.urlopen(requete)# Récupérer le fi
    chier JSON
5 donneesBrut=reponse.read().decode("utf-8")# Décoder le texte
reçu 6 donneesJSON = json.loads(donneesBrut)# Décoder le fichier
JSON 7 position = donneesJSON["iss_position"]
8 print("La station spatiale internationale est située à une longit
    ude"+position["longitude"]+"et à une latitude"+position["
    latitude"]+".")
```

Nous allons utiliser une seconde API permettant de trouver les communes associées à un code postal. L'API où est le code postal recherché. Voici un exemple de données retournées :

```
{
"postcode": "21000",
"country": "France",
```

1. *Application Programming Interface*


```
"countryabbreviation": "FR",
"places": [
{
"placename": "Dijon",
"longitude": "5.0167",
"state": "Bourgogne",
"stateabbreviation": "A1",
"latitude": "47.3167"
}
]
}
```

Voici le programme permettant d'afficher les communes associées à un code postal :

```
import urllib.request
import json
codePostal=input("Entrez le code postal:")
requete=urllib.request.Request('http://api.zippopotam.us/FR/'+codePostal)
reponse=urllib.request.urlopen(requete)
donneesBrut=reponse.read().decode("utf-8")
donneesJSON=json.loads(donneesBrut)
listeCommunes=donneesJSON["places"]
print("Voici les communes ayant pour code postal"+codePostal+":")
for commune in listeCommunes:
print("-"+commune["placename"])
```

Exercices

N'oubliez pas le shebang et commentez si nécessaire.

Vous êtes nouvellement embauché dans une banque pour mettre au point le système de commu nication entre les distributeurs automatiques et le système central de gestion des comptes. Votre travail est de développer les applications sur ces deux systèmes pour

permettre aux distributeurs de communiquer avec le système central pour effectuer les transactions.

On souhaite créer une base de données SQLite3 stockant les soldes des comptes, les informations les concernant, ainsi que les transactions effectuées. On utilisera un serveur de socket pour effectuer les communications entre les deux systèmes. Voici les différents messages pris en charge avec (C→S) un message du client vers le serveur et (S→C) un message du serveur vers le client :

— (C→S) : Vérifier si le code PIN saisi est correct. — (S→C) : La vérification du code PIN est validée.

— (S→C) : La vérification du code PIN n'est pas valide.
— (C→S) : Demande un retrait du montant défini. — (S→C) : Le retrait est validé.
— (S→C) : Le retrait est refusé.
— (C→S) : Demande un dépôt du montant défini. — (S→C) : Le dépôt est validé.
— (C→S) : Demande un transfert du montant défini entre deux comptes.
— (S→C) : Le transfert est validé.
— (S→C) : Le transfert est refusé.
— (C→S) : Demande le solde du compte
— (S→C) : Renvoie le solde du compte demandé
— (C→S) : Demande les 10 dernières opérations du compte — (S→C) : Renvoie les 10 dernières opérations du compte au format CSV (date, libellé, montant déposé ou retiré).
— (S→C) : Signale que l'opération demandée n'est pas valide.

La direction de la banque vous fournit les fichiers CSV contenant :

Liste des clients

Liste des comptes

Liste des opérations

Toute opération doit être précédée d'une vérification du code PIN. Les exercices suivants per mettront d'effectuer cela en scindant le travail en différentes sous-tâches. Chaque tâche fera l'objet d'un nouveau programme.

1. Écrivez un programme permettant de créer sur le serveur une base de données SQLite3 nom mée et de créer la structure de table adaptée au stockage des données. Importez le contenu des fichiers CSV dans cette base.

2. Écrivez les programmes du serveur central de la banque et des distributeurs automatiques. Programme centrale bancaire :