

# Chapitre 1-4

## Installer son environnement de travail

### 1. Introduction

Il ne s'agit ici que de CPython, l'implémentation de référence de Python, et non de PyPy ou Jython.

Quel que soit votre système d'exploitation, vous pouvez installer Python en lisant ce chapitre puis, dans un second temps, installer des bibliothèques tierces au gré de vos besoins (cf. section Installer une bibliothèque tierce) et vous pourrez créer des environnements virtuels (cf. section Créer un environnement virtuel).

Si vous souhaitez installer d'un seul coup Python ainsi que Jupyter (anciennement IPython) et la plupart des bibliothèques scientifiques ou d'analyse de données, vous pouvez aller directement à la section Installer Anaconda, pour installer celui-ci en lieu et place de Python. Vous disposerez alors d'autres méthodes pour gérer les environnements virtuels et pour installer des bibliothèques tierces.

### 2. Installer Python

#### 2.1 Pour Windows

Le système d'exploitation Windows requiert usuellement l'utilisation d'un installateur pour pouvoir installer un logiciel quel qu'il soit. Si vous disposez de Windows, vous devriez en avoir l'habitude. Python ne déroge pas à la règle.

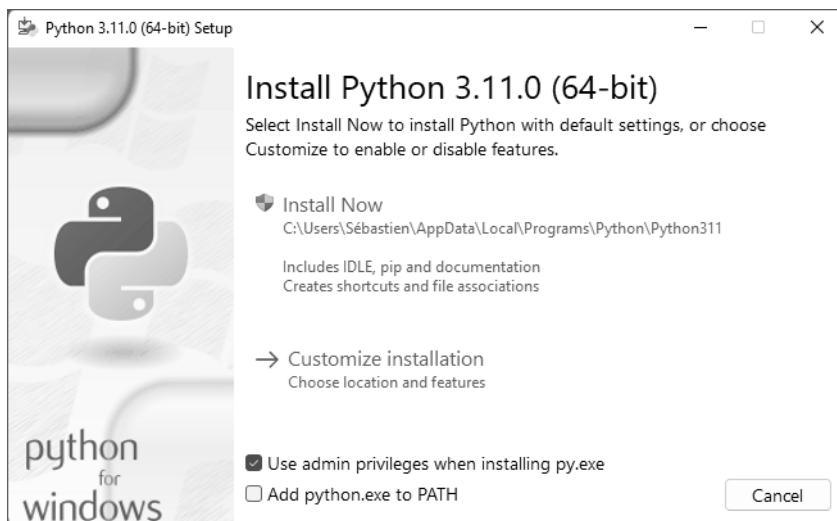
Pour installer Python, vous devez donc aller sur le site officiel (<https://www.python.org/downloads/>) pour télécharger l'installateur adéquat. Comme vous pourrez le constater, on vous met en avant un accès rapide à la dernière version (au moment où ces lignes sont écrites, la 3.11.0), puis un accès aux dernières versions encore actives (actuellement la version 3.10 qui reçoit encore des corrections d'anomalies, puis les versions 3.9 à 3.7 qui reçoivent des corrections de sécurité uniquement).

Il est également possible de télécharger la toute dernière version de la branche 2.7 qui est en fin de vie (elle n'est plus mise à jour), car il existe encore de nombreux projets n'ayant pas encore migré.

Le support correctif dure 2 ans après la première sortie de la version et le support de sécurité dure 5 ans.

Pour notre part, nous vous conseillons la dernière 3.x, mais vous êtes libre d'installer celle que vous souhaitez ou même d'en installer plusieurs suivant vos contraintes, il n'y a pas d'objection à cela.

Une fois le téléchargement effectué, vous devez lancer l'installateur (et éventuellement passer quelques protections de votre système qui vous demande d'accorder votre confiance à cet installateur), pour observer l'écran suivant :

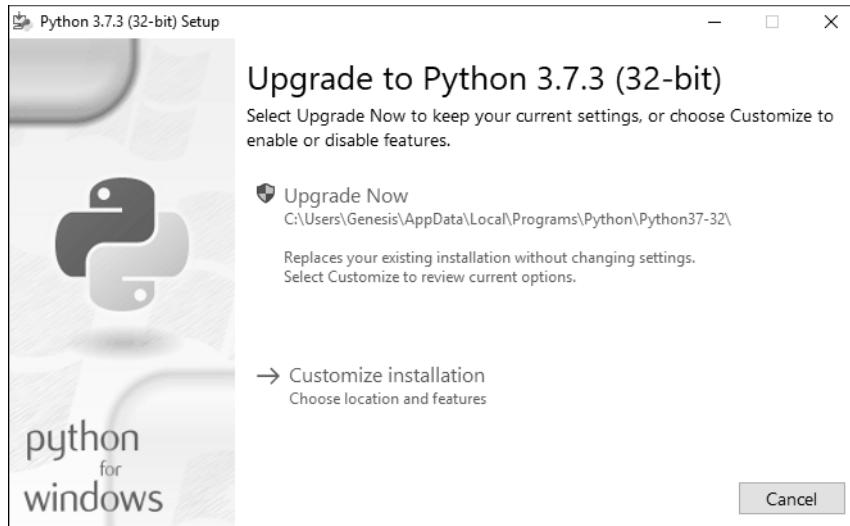


Comme vous pouvez le constater, il est possible de personnaliser l'installation en choisissant le chemin d'installation du logiciel ou en choisissant de ne pas sélectionner quelques fonctionnalités, mais nous ne le conseillons pas.

Nous vous recommandons en revanche de cocher la case **Add python.exe to PATH** afin de configurer la variable PATH du terminal pour rendre Python accessible plus facilement.

## Chapitre 1-4

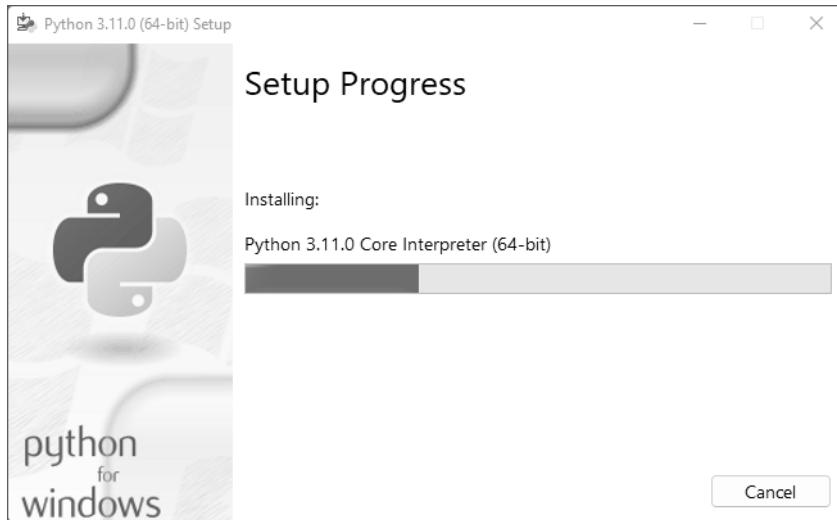
Si vous avez déjà une ancienne version de Python installée de la même branche (dans cet exemple, Python 3.7.2 est déjà installé), vous pourrez la mettre à jour à l'aide du même installeur :



Par contre, si vous avez déjà la version 3.7.1 et que vous installez la version 3.11, cette dernière ne viendra pas remplacer la précédente, mais s'installera à côté. Si vous souhaitez remplacer, il vous faudra donc désinstaller proprement la toute dernière version installée, ce qu'il est possible de faire en relançant l'installeur d'origine.

Nous vous encourageons à garder les installateurs sur votre PC, car ils pourraient devenir indisponibles au téléchargement si trop vieux.

Quel que soit le scénario, vous arriverez devant un écran vous montrant la progression de l'installation et vous n'aurez qu'à fermer la fenêtre une fois celle-ci terminée :



Vous êtes maintenant prêt à utiliser Python.

## 2.2 Pour Mac

Il faut savoir qu'une version de Python est déjà préinstallée sur Mac, car Mac OS X l'utilise pour ses propres besoins et Python est intégré à son propre cycle de développement. Cependant, si vous souhaitez une version différente de celle qui est déjà présente, vous pouvez l'installer, sachant qu'il n'y a pas de contre-indication à posséder plusieurs versions de Python sur la même machine.

Pour installer Python sur Mac OS X, la procédure à suivre est similaire à celle pour Windows. Il faut donc se rendre sur le site officiel (<https://www.python.org/downloads/mac-osx/>), télécharger un installateur correspondant à sa configuration et suivre les étapes.

Pour les utilisateurs de Mac, sachez que Python dispose d'une bonne intégration de ses spécificités, en particulier vis-à-vis de Objective-C, le langage de programmation avec lequel est développé Mac OS X, et Cocoa, interface de programmation de Mac OS X.

## 2.3 Pour GNU/Linux et BSD

Les différentes distributions libres utilisent nativement Python, notamment pour des parties sensibles. Python y est donc tout naturellement déjà installé, généralement sous la dernière version de la branche 2.x. Cependant, ici comme ailleurs, il n'y a pas d'objections à utiliser plusieurs versions de Python.

Le plus simple reste d'utiliser votre gestionnaire de paquets, ce qui peut se faire via un outil graphique, comme Synaptic pour Debian :



Il suffit alors de faire une recherche sur le mot-clé **python** pour voir les différentes versions (sur une ancienne Debian, ce sont Python 2.6, 2.7 et 3.2).

Par contre, tous les paquets **python3-xxxxx** que vous pouvez voir ici sont des bibliothèques tierces et non Python lui-même. Nous en parlerons plus tard dans ce chapitre.

Une fois les paquets souhaités sélectionnés, il ne manque plus qu'à les installer en cliquant sur le bouton **Appliquer**.

Notez que tout ceci peut se faire par la simple ligne de commande, toujours en utilisant votre gestionnaire de paquets qui peut être apt-get, aptitude, yum, emerge, pkg\_add ou autre.

Voici par exemple pour une distribution Debian ou Ubuntu :

```
█ $ sudo aptitude install python3
```

Ceci ne permet cependant pas de choisir la version que l'on souhaite, à moins d'aller trouver des sources alternatives. Si l'on veut avoir la toute dernière version de Python, il faudra la plupart du temps passer par la compilation.

## 2.4 Par la compilation

Compiler Python n'est pas en soi une tâche très complexe. C'est par contre souvent une tâche imposée lorsque l'on ne travaille pas avec des conteneurs. En effet, en entreprise, on développe souvent des applications qui sont destinées à être hébergées. Il est alors impératif de travailler sur votre propre poste avec une version de Python qui soit la même que celle existante sur la machine de production.

Sous GNU/Linux, mais aussi sous d'autres systèmes, il est possible de compiler la version de Python que l'on souhaite. Après tout, Python n'est rien d'autre qu'un programme écrit en C. Pour ce faire, il faut aller télécharger le code source (<https://www.python.org/downloads/source/>), qui prend la forme d'une archive, puis décompresser celle-ci, se placer dans le répertoire ainsi obtenu et taper ces quelques commandes :

```
$ ./configure --prefix=/path/to/my/python/directory  
$ make  
$ sudo make altinstall
```

Notez que dans cette dernière ligne, nous n'utilisons pas la commande **make install**, qui aurait pour effet de remplacer votre Python système par le Python que vous compilez, ce qui pourrait avoir des conséquences indésirables voire désastreuses.

Notez également que vous choisissez lors de la configuration le chemin dans lequel vous placerez vos bibliothèques Python. En général, l'usage veut que l'on utilise **/opt**, mais il n'y a pas de règle, tout dépend des pratiques de votre entreprise ou votre expérience en la matière.

Si vous venez d'installer Python 3.5 par cette méthode, vous aurez alors maintenant accès à ce programme en l'appelant ainsi, depuis votre terminal :

```
$ python3.5
```

Par cette même méthode, vous pouvez installer les dernières versions (<https://www.python.org/download/pre-releases/>) de Python qui ne sont pas encore sorties (alphas ou betas), ce qui vous permet de les tester en avant-première !

Notons que, par cette méthode, toutes les bibliothèques de Python ne fonctionneront pas. En effet, lorsqu'elles ont besoin d'autres bibliothèques C, il faut effectuer des compilations croisées et utiliser les différents en-têtes de ces bibliothèques. C'est le cas par exemple pour faire fonctionner Curses, ReportLab (génération de fichiers PDF) ou encore PyUSB (accès aux ports matériels USB).

Dans ce cas-là, la commande **./configure** devra recevoir des arguments supplémentaires et vous devrez trouver un tutoriel en ligne pour vous indiquer la démarche, laquelle peut être plus ou moins complexe.

## 2.5 Pour un smartphone

Installer une machine virtuelle Python sur un smartphone est possible. Pour Android, la procédure est assez simple puisqu'il existe un produit dédié (<http://qpython.com/>), tout comme sur Windows Phone (<https://apps.microsoft.com/store/detail/python-39/9P7QFQMJRFP7>). Pour iOS, c'est une autre paire de manches (<https://github.com/linusyang/python-for-ios>) étant donné que l'utilisateur est enfermé dans un système sur lequel il n'a aucun contrôle.

### 3. Installer une bibliothèque tierce

#### ■ Remarque

*Si vous abhorrez le terminal, sachez que vous pouvez installer une bibliothèque tierce depuis votre IDE, ce qui sera probablement plus aisé pour vous.*

#### 3.1 À partir de Python 3.4

Pour installer une bibliothèque tierce, vous devez simplement connaître son nom. Celui-ci est généralement assez intuitif. Par exemple, la bibliothèque permettant de communiquer avec un serveur Redis s'appelle `redis`.

Il peut y avoir des variations. Par exemple, la bibliothèque de référence pour traiter du XML est `lxml` et, plus complexe, celle pour BeautifulSoup est `bs4`. En recherchant comment répondre à un besoin sur le Net ou sur PyPi (<https://pypi.python.org/pypi>), vous trouverez rapidement une bibliothèque de référence.

Sur des sujets plus confidentiels, il vous arrivera de trouver plusieurs petites bibliothèques. Vous pourrez alors les tester et choisir celle que vous utiliserez pour votre projet.

Sachez que vous pouvez aussi conduire une recherche directement depuis votre terminal :

```
■ $ pip search xml  
■ $ pip search soup
```

Cela vous donnera une liste de bibliothèques accompagnée d'une courte description, à la manière de ce que font les gestionnaires de paquets sous Linux (lesquels sont écrits en Python, au passage).

Sachez que **pip** existe quel que soit votre système d'exploitation (vous devez être familier avec le terminal de votre système, cependant) et que depuis la version 3.4 de Python, il est installé automatiquement avec celui-ci. Si ce n'est pas votre cas, consultez la section suivante : Pour une version inférieure à Python 3.4.

**pip** est un outil formidable. Si vous utilisez une version de Python qui est celle du système, vous utiliserez alors la commande **pip** pour gérer les bibliothèques. Si vous utilisez une autre version, telle que Python 3.5, alors vous utiliserez la commande **pip-3.5**. Pour Python 3.3, ce sera **pip-3.3**. Dans les exemples suivants, il vous faudra prendre en compte cette particularité.

Cet outil vous permettra d'installer une bibliothèque à sa dernière version ainsi que toutes les bibliothèques dépendantes. En effet, il n'est pas rare qu'une bibliothèque de Python ait besoin d'une autre bibliothèque (ou de plusieurs) pour fonctionner. Par exemple, l'installation de `redis` se fait par cette commande :

```
■ $ pip install redis
```

On peut aussi choisir la version à installer :

```
■ $ pip install -Iv redis==2.10.5
```

Ou mettre à jour la bibliothèque à une version précise :

```
■ $ pip install -U redis==2.10.5
```

Ou à la dernière version :

```
■ $ pip install -U redis
```

Et on peut la désinstaller :

```
■ $ pip uninstall redis
```

Une fonctionnalité très importante permet d'obtenir la liste des bibliothèques déjà installées (quelle que soit la manière dont elles ont été installées) :

```
■ $ pip freeze
```

Ce que l'on peut mettre dans un fichier :

```
■ $ pip freeze > requirements.txt
```

Pour installer tous les paquets ainsi listés, il faut procéder ainsi :

```
■ $ pip install -r requirements/base.txt
```

Cette méthode est particulièrement utile dans le cadre d'un environnement virtuel ; nous y reviendrons.

Il est possible de retrouver des informations sur un paquet déjà installé :

```
■ $ pip show django-redis
---
Name: django-redis
Version: 4.3.0
Location: /path/to/my/env/lib/python3.4/site-packages
Requires: redis
```

On voit ici que le paquet **django-redis** a une dépendance vers **redis** : en l'installant, on installe automatiquement **redis**.

Mettre à jour ce paquet met à jour automatiquement les dépendances :

```
■ $ pip install -U django-redis
```

Si on ne veut pas mettre à jour les dépendances, on peut procéder ainsi :

```
■ $ pip install -U --no-deps django-redis
```

On peut aussi installer plusieurs bibliothèques en même temps :

```
■ $ pip install django-redis==4.3.0 bs4 lxml
```

Cette commande installera donc automatiquement redis s'il n'est pas installé, car il est déclaré comme dépendance.

Cette commande a cependant des limites. En effet, si vous installez une bibliothèque tierce qui utilise une bibliothèque C, vous devrez disposer des en-têtes C correspondants (paquets **dev** pour Debian ou **devel** pour Fedora). Il faut donc avoir un peu de pratique dans ce genre de situation pour savoir déjouer ces pièges.

# Chapitre 1.3

## Génération de contenu

### 1. PDF

#### 1.1 Présentation

##### 1.1.1 Format PDF

PDF est le sigle de *Portable Document Format* qui est un format de document utilisant un langage de description de page qui est PDL (sigle de *Page Document Language*) et un protocole d'impression non dépendant d'un constructeur qui est une évolution de Postscript. Il est devenu progressivement le standard pour l'impression des documents et est devenu une norme ISO.

Aujourd'hui, de très nombreux formats de données (textes, dessins, images...) proposent des fonctionnalités d'export vers PDF.

##### 1.1.2 Avantages

Ses qualités principales sont :

- la concordance entre l'affichage à l'écran et ce qui sera réellement imprimé ;
- son indépendance par rapport au système d'exploitation ;
- son indépendance par rapport au matériel ;
- la présence de logiciels ou bibliothèques libres pour ce format ;
- la très grande diffusion de lecteurs de documents PDF, dont certains sont libres.

##### 1.1.3 Inconvénients

Les inconvénients sont de plusieurs ordres :

- Les droits liés à chaque document dépendent à la fois de ceux liés au format, mais également à ceux liés à tout ce qui est contenu dans le document, à savoir les droits d'auteurs sur les textes, les images embarquées, mais également les polices embarquées.

- L'évolution du format est majoritairement liée à la politique d'un seul éditeur.

### 1.1.4 Présentation de la bibliothèque libre

ReportLab est une bibliothèque externe écrite en Python qui offre des outils simples et performants pour générer des documents PDF. Cette bibliothèque est maintenue par un éditeur qui propose deux branches, dont une à destination de la communauté qui est celle que nous allons utiliser.

Elle est portée sous Python 3, plus précisément à partir de Python 3.3, et voici le dépôt de code : <https://github.com/nakagami/reportlab>

## 1.2 Bas niveau

### 1.2.1 Bibliothèque de données

La première chose pour créer un document est d'avoir des données préfabriquées. En effet, il est agréable d'avoir des formats tout prêts correspondant à du A4, A3 ou tout autre format usuel, des couleurs prédéfinies prêtes à l'emploi et un système permettant de convertir les mesures que l'on connaît avec celles utilisées par ReportLab, à savoir en **point** car c'est l'unité de mesure des écrans et des imprimantes.

Voici ce que l'on peut trouver en fouillant un peu dans cette bibliothèque de données :

```
>>> from reportlab import lib
>>> dir(lib)
['RL_DEBUG', '__builtins__', '__doc__', '__file__', '__name__',
 '__package__', '__path__', '__version__', 'boxstuff', 'colors',
 'logger', 'os', 'pagesizes', 'rltempfile', 'units', 'utils']
```

Voici pour commencer le module dédié aux formats standards des imprimantes courantes :

```
>>> from reportlab.lib import pagesizes
>>> dir(pagesizes)
['A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'B0', 'B1', 'B2', 'B3',
 'B4', 'B5', 'B6', 'ELEVENSEVENTEEN', 'LEGAL', 'LETTER', '_BH',
 '_BW', '_H', '_W', '__builtins__', '__doc__', '__file__',
 '__name__', '__package__', '__version__', 'cm', 'elevenSeventeen',
 'inch', 'landscape', 'legal', 'letter', 'portrait']
```

Voici quelques-unes de ces unités :

```
>>> units.inch
72.0
>>> units.cm
28.346456692913385
```

Il y a donc 72 points par pouce et 28 par centimètre. On retrouve facilement le fait qu'un pouce fait 2,54 centimètres. On préférera parler dans une unité qui est homogène à celle du système international, mais la bibliothèque elle-même utilise les unités anglaises :

```
>>> units.inch / units.cm
2.54
```

## Chapitre 1.3

On peut donc ainsi convertir les mesures de pages en centimètres :

```
>>> [s / cm for s in pagesizes.A4]  
[21.0, 29.7]
```

Il y a un dernier module particulièrement important, permettant de ne pas se compliquer la vie pour créer des couleurs, en disposant des plus courantes d'entre elles :

```
>>> from reportlab.lib import colors  
>>> dir(colors)
```

Le résultat de cette commande est relativement long, signe du nombre important de couleurs. Il y a fort à parier que LA couleur que vous voulez en fasse partie.

Voici comment se présentent ces couleurs :

```
>>> colors.yellow  
Color(1,1,0,1)  
>>> colors.yellowgreen  
Color(.603922,.803922,.196078,1)  
>>> colors.turquoise  
Color(.25098,.878431,.815686,1)  
>>> colors.olive  
Color(.501961,.501961,0,1)
```

Il s'agit en réalité d'une sorte de 4-uplet qui ne présente pas une couleur au format CMJN (Cyan, Magenta, Jaune, Noir), contrairement à ce que l'on pourrait penser de prime abord, étant donné que c'est le format pour les imprimantes, mais un format RVB.

Il est possible d'obtenir des informations sur une couleur selon beaucoup de méthodes différentes :

```
>>> c = colors.turquoise  
>>> c.red, c.green, c.blue, c.alpha  
(0.25098039215686274, 0.8784313725490196, 0.8156862745098039, 1)  
>>> c.hexval(), c.hexvala()  
('0x40e0d0', '0x40e0d0ff')  
>>> c.rgb(), c.rgb()  
((0.25098039215686274, 0.8784313725490196, 0.8156862745098039),  
(0.25098039215686274, 0.8784313725490196, 0.8156862745098039, 1))  
>>> c.bitmap_rgb(), c.bitmap_rgba()  
((64, 224, 208), (64, 224, 208, 255))
```

Et si aucune couleur ne convenait, il est toujours possible d'en créer. Voici par exemple un « Azur clair », dont les valeurs proviennent de Wikipédia ([http://fr.wikipedia.org/wiki/Liste\\_des\\_couleurs](http://fr.wikipedia.org/wiki/Liste_des_couleurs)) :

```
>>> c = colors  
>>> c = colors.Color(116./255, 208./255, 241./255, 1)  
>>> c  
Color(.454902,.815686,.945098,1)
```

De la même manière, il est possible de se fabriquer un format d'impression, puisque ce n'est finalement rien d'autre qu'un 2-uplet, mais cela est plus rare, ReportLab ayant tout ce qu'il faut à ce niveau-là.

Un dernier point extrêmement important est la gestion des polices. Par convention, 14 d'entre elles sont standardisées et disponibles donc sans efforts particuliers, il suffit d'utiliser leur nom. Voici ces noms :

```
>>> from reportlab.pdfbase import pdfmetrics  
>>> pdfmetrics.standardFonts  
('Courier', 'Courier-Bold', 'Courier-Oblique', 'Courier-BoldOblique',  
'Helvetica', 'Helvetica-Bold', 'Helvetica-Oblique', 'Helvetica-BoldOblique',  
'Times-Roman', 'Times-Bold', 'Times-Italic', 'Times-BoldItalic', 'Symbol',  
'ZapfDingbats')
```

Leur utilisation est libre. Il est toujours possible d'utiliser d'autres polices d'écritures, qui peuvent potentiellement être soumises à licence, mais il faut alors les charger et les embarquer dans le document.

## 1.2.2 Canvas

Voici un petit script, disponible parmi ceux fournis avec cet ouvrage, qui permet d'utiliser le canvas. L'objectif est de montrer ce qu'il est possible de faire à bas niveau. D'abord les imports nécessaires :

```
>>> from reportlab.pdfgen.canvas import Canvas  
>>> from reportlab.lib.units import cm
```

Voici comment créer un canvas et rajouter des métadonnées qui sont visibles par les systèmes d'exploitation et les lecteurs PDF :

```
>>> canvas = Canvas("hello.pdf")  
>>> canvas.setTitle("Premier document")  
>>> canvas.setSubject("Creation de document PDF avec ReportLab")  
>>> canvas.setAuthor('SCH')  
>>> canvas.setKeywords(['PDF', 'ReportLab', 'Python'])  
>>> canvas.setCreator('sch')
```

Voici comment positionner un texte en précisant sa police d'écriture et sa taille :

```
>>> canvas.setFont("Helvetica", 36)  
>>> canvas.drawCentredString(12.0 * cm, 18.0 * cm, "Hello world")
```

Voici un autre exemple intéressant :

```
>>> canvas.setFont("Times-Roman", 12)  
>>> canvas.drawString(1.0 * cm, 1.0 * cm, "O")  
>>> canvas.drawString(2.0 * cm, 1.0 * cm, "X")  
>>> canvas.drawString(1.0 * cm, 2.0 * cm, "Y")
```

On a positionné ainsi un repère O, X, Y qui permet de voir dans quel sens sont prises les mesures. Comme on peut le voir en lisant le document ainsi généré, le point O est en bas à gauche, l'axe X est vers la droite et l'axe Y vers le haut.

Tant que l'on ne change pas de police d'écriture, on continue de l'utiliser. On n'a pas besoin de le préciser à chaque écriture.

## Chapitre 1.3

Voici maintenant la notion d'alignement qui est mise en évidence :

```
>>> canvas.drawString(8.5 * cm, 16.0 * cm, "G")
>>> canvas.drawRightString(8.5 * cm, 15.0 * cm, "D")
>>> canvas.drawCentredString(8.5 * cm, 14.0 * cm, "C")
>>> canvas.drawString(12.5 * cm, 16.0 * cm, "Aligné à gauche")
>>> canvas.drawRightString(12.5 * cm, 15.0 * cm, "Aligné à droite")
>>> canvas.drawCentredString(12.5 * cm, 14.0 * cm, "Aligné au centre")
```

Enfin, est mis en évidence le fait que le changement de ligne ne peut se faire simplement et que les textes ne passent pas à la ligne automatiquement :

```
>>> canvas.drawCentredString(10.5 * cm, 10.0 * cm,
    '\n'.join(["Aligné plein centre"] * 5))
>>> canvas.drawString(18.5 * cm, 12.0 * cm, "Mal Aligné" * 5)
>>> canvas.drawRightString(2.5 * cm, 12.0 * cm, "Mal Aligné" * 5)
>>> canvas.save()
```

Finalement, on enregistre et on crée ainsi le fichier.

## 1.3 Haut niveau

### 1.3.1 Styles

Un élément clé de la génération de document avec des outils de haut niveau est la manipulation de styles.

Voici pour commencer le moyen de récupérer une feuille de style standard :

```
>>> from reportlab.lib.styles import getSampleStyleSheet
>>> styles = getSampleStyleSheet()
>>> styles.list()
```

Voici ce que cette commande donne pour chaque style :

```
BodyText None
    name = BodyText
    parent = <ParagraphStyle 'Normal'>
    alignment = 0
    allowOrphans = 0
    allowWidows = 1
    backColor = None
    borderColor = None
    borderPadding = 0
    borderRadius = None
    borderWidth = 0
    bulletFontName = Helvetica
    bulletFontSize = 10
    bulletIndent = 0
    firstLineIndent = 0
    fontName = Helvetica
    fontSize = 10
    leading = 12
    leftIndent = 0
```

```
rightIndent = 0
spaceAfter = 0
spaceBefore = 6
textColor = Color(0,0,0,1)
textTransform = None
wordWrap = None
```

Et voici la liste des autres styles standards :

- |              |            |            |          |
|--------------|------------|------------|----------|
| – Bullet     | – Heading1 | – Heading4 | – Italic |
| – Code       | – Heading2 | – Heading5 | – Normal |
| – Definition | – Heading3 | – Heading6 | – Title  |

Voici comment créer un nouveau style de paragraphe :

```
>>> from reportlab.lib.styles import ParagraphStyle
>>> from reportlab.lib.enums import TA_JUSTIFY
>>> mon_style = ParagraphStyle(name='mon_style',
    alignment=TA_JUSTIFY, fontName = "Helvetica", fontSize = 14)
```

Et voici comment le rajouter à la liste ci-dessus :

```
>>> styles.add(mon_style)
```

Ceci fait également partie du module vu en introduction.

L'utilisation de paramètres nommés est la règle dès lors que l'on en a beaucoup, ce qui permet une relecture de code plus claire, y compris lorsque l'on ne connaît pas par cœur les signatures des méthodes :

```
>>> help(ParagraphStyle)
```

On a donc les moyens de créer des styles spécifiques à la demande avant de commencer à créer le contenu. Tous sont applicables à des textes.

Cependant, il y a une particularité pour laquelle l'application de styles est particulière, il s'agit des tableaux :

```
>>> style_tableau = [
...     ('ALIGN',      (0,0),  (-1,-1), "LEFT"),
...     ('VALIGN',     (0,0),  (-1,-1), "TOP"),
...     ('LEFTPADDING', (0,0),  (-1,-1), 0*cm),
...     ('RIGHTPADDING', (0,0),  (-1,-1), 0*cm),
...     ('TOPPADDING',  (0,0),  (-1,-1), 0*cm),
...     ('BOTTOMPADDING', (0,0),  (-1,-1), 0*cm),
... ]
```

Cette feuille de style est une liste de 4-uplets qui désignent chacun respectivement le nom du style, la case en haut à gauche et la case en bas à droite qui englobent les cases du tableau concernées et la valeur associée au style.

## Chapitre 1.3

Il est possible de créer un autre style à partir de celui-ci, pour éviter une double saisie :

```
>>> style_tableau1 = style_tableau[:]
>>> style_tableau1.append(('LINEABOVE', (0,0), (-1, 0), 1,
colors.turquoise))
>>> style_tableau1.append(('LINEABOVE', (0,1), (-1,-1), 0.5,
colors.darkturquoise))
```

Il est très important de ne pas oublier les `[ : ]` pour faire une copie, sans quoi on ne fait que créer un pointeur vers la même liste.

### 1.3.2 Flux de données

Pour un document textuel qui est une suite de paragraphes, voire d'objets comme des images ou des tableaux, on écrit alors un flux de donnée.

Voici les éléments vus précédemment nécessaires :

```
>>> from reportlab.lib import colors
>>> from reportlab.lib.styles import getSampleStyleSheet
>>> from reportlab.lib.styles import ParagraphStyle
>>> from reportlab.lib.enums import TA_JUSTIFY
>>> from reportlab.lib.pagesizes import A4
>>> from reportlab.lib.units import cm
```

Le module utilisé est `platypus` et l'élément central est :

```
>>> from reportlab.platypus import SimpleDocTemplate
```

Il faut créer une liste d'autres objets de ce même module :

```
>>> flowables = []
```

Une fois que l'on a défini les styles comme présenté dans la sous-section Styles de ce chapitre, un paragraphe est rajouté ainsi :

```
>>> from reportlab.platypus import Paragraph
>>> flowables.append(Paragraph("Fichier PDF Généré", styles["Heading1"]))
>>> flowables.append(Paragraph("Sébastien CHAZALLET", styles["Normal"]))
>>> flowables.append(Paragraph("http://www.inspyration.com",
styles["Code"]))
```

Pour chaque paragraphe, le style est précisé. Il est possible d'insérer un contenu sur plusieurs lignes, mais celles-ci ne formeront qu'une seule ligne dans le résultat final :

```
>>> content = """Ce document est généré par le script 02_flux.py.
... Ce script est livré avec le présent ouvrage.
... Vous pouvez le modifier à souhait pour faire vos propres expériences"""
>>> flowables.append(Paragraph(content, styles["Normal"]))
```

Il existe également un objet particulier pour effectuer un saut de ligne :

```
>>> from reportlab.platypus import Spacer
>>> flowables.append(Spacer (0, 0.2*cm))
```

Un autre pour gérer le saut de page :

```
>>> from reportlab.platypus import PageBreak
>>> flowables.append(PageBreak())
```

Voici comment écrire un tableau. Il s'agit d'une liste de listes de paragraphes qui chacun portent un style (en plus du style propre au tableau) :

```
>>> data, line = [], []
>>> line.append( Paragraph ("Technologie", styles["Normal"])) )
>>> line.append( Paragraph ("Logiciel", styles["Normal"])) )
>>> line.append( Paragraph ("Alternatives", styles["Normal"])) )
>>> data.append(line)
>>> line = []
>>> line.append( Paragraph ("OS", styles["Normal"])) )
>>> line.append( Paragraph ("Debian", styles["Normal"])) )
>>> line.append( Paragraph ("Ubuntu, Fedora", styles["Normal"])) )
>>> data.append(line)
```

Pour rajouter un tel tableau dans le flux principal, il faut procéder ainsi :

```
>>> from reportlab.platypus import Table
>>> flowables.append(Table(data, colWidths=[5*cm, 5*cm, 8*cm],
style=style_tableau))
```

Voici comment rajouter une image :

```
>>> from reportlab.platypus import Image
>>> flowables.append(Image('hello.pdf.png', height = 5 * cm, width
= 8 * cm))
```

Voici enfin comment créer le document final à partir du travail effectué :

```
>>> pdf = SimpleDocTemplate('test.pdf', pagesize = A4, title =
'Premier test', author = 'SCH')
>>> pdf.build(flowables)
```

La complexité de la création d'un fichier PDF est extrêmement basse, puisque linstanciation de quelques objets et l'utilisation de listes et de n-uplets permettent de créer très simplement des documents contenant du texte sur lequel on peut appliquer des styles, des sauts de lignes, des sauts de page, des tableaux et des images comme montré ici.

Cette bibliothèque externe est relativement bien documentée et il suffit d'utiliser help sur différents objets ou méthodes pour retrouver des informations sur la manière de les utiliser. Il existe une documentation qui apporte notamment quelques exemples à reproduire. Une fois que l'on maîtrise un peu la bibliothèque, réaliser des composants permettant d'automatiser la création de documents selon des règles métiers précises est relativement aisé.

Il est nécessaire d'aller un peu plus loin pour voir les possibilités offertes, par exemple la possibilité de créer des visuels, comme cela sera présenté dans la sous-section Création d'un visuel de ce chapitre, ou encore la possibilité de créer un traitement spécifique pour la première page via la signature de la méthode build, ou enfin créer des templates de pages permettant d'automatiser un numéro de page, ce qui sera présenté dans la sous-section Template de page de ce chapitre.

Il est également à noter que la classe Image de la bibliothèque PIL et celle de la bibliothèque ReportLab sont compatibles et il est possible d'ouvrir une image, d'effectuer des traitements dessus (modifier les dimensions...) et la transformer en image ReportLab.