

# Faire bonne impression avec Python

Découvrons ensemble l'éventail des possibilités d'impression de Python.

Il existe de nombreuses possibilités pour effectuer des travaux d'impression avec Python. Ces possibilités sont assez disparates, et pas d'une efficacité constante selon les plates-formes. Nous allons examiner tout cela dans le détail, pour UNIX en général et Linux en particulier, pour Windows, ou, lorsque c'est possible, pour les deux à la fois.

## Si vous êtes pressés

Au départ, lorsque Guido Van Rossum a créé Python, il a voulu faire un langage de script, proposant une alternative au Shell. Guido est issu d'une culture UNIX et ne s'est pas particulièrement intéressé à l'impression, ou plutôt, a considéré que les choses allaient de soi. En effet sous UNIX, on imprime un document en l'injectant dans la file d'impression, tout simplement. Si la file d'impression est bien configurée, bien équipée en filtres, on imprime à peu près ce que l'on veut, même depuis la console, par exemple comme ceci :

```
cat document | lpr
```

Python permet de lancer une commande système, donc imprimer en Python de cette façon est simple. Voici un programme Python 'printlinux.py' qui s'imprime lui même :

```
#!/usr/bin/env python
import os
modele_commande = 'cat %s | lpr'
commande = modele_commande %('printlinux.py')
os.popen(commande)
```

Il est possible d'appliquer cette méthode expéditive sous Windows, en changeant une ligne de code :

```
modele_commande = 'type %s > prn'
```

Malheureusement, sous Windows, les résultats ne sont pas garantis sur facture. Cette méthode fait appel au DOS et il peut arriver que votre imprimante ne soit pas toujours docile. En outre, vous pouvez obtenir un horrible effet d'escalier, si vous tentez d'imprimer un fichier texte écrit sous Linux ou même sous Emacs - Windows. En effet, dans ce cas, les retours à la ligne ne sont indiqués que par le caractère `\xA`, alors que Windows attend la séquence `\xA\xoD` pour assurer le retour du chariot, d'où l'effet d'escalier. Cette méthode n'est donc finalement à retenir que sous Linux.

## Sous Windows

Python dispose de riches extensions pour cette plate-forme. Les extensions Windows ne font pas partie du package Python de base et sont donc à télécharger séparément à <http://www.python.org>. Parmi ces extensions vient Pythonwin, qui est une librairie enveloppe autour de la librairie C++ MFC de Microsoft. MFC est un ensemble de classes plus ou moins bien conçues,

PRATIQUÉ

Python

NIVEAU : DÉBUTANT

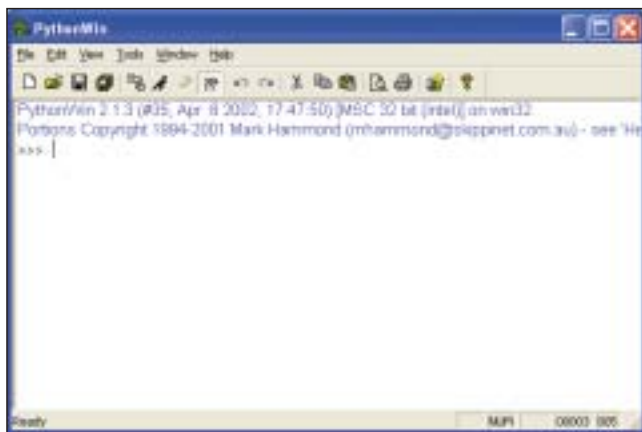
mais accélérant toutefois la programmation sous Windows. Sous Windows, on emploie la même méthode, pour tracer à l'écran et pour imprimer. Cette méthode consiste en l'obtention d'un contexte de périphérique (dit DC pour Device Context) et en l'appel des fonctions graphiques du système avec ce DC en argument. Lorsque le fameux DC est encapsulé dans une classe, comme c'est le cas avec MFC, les méthodes graphiques sont invoquées depuis l'objet. Il n'est pas ici question d'entrer dans le détail des MFC. Voici simplement un exemple de code qui imprime quelque chose :

```
import win32ui

dc = win32ui.CreateDC()
dc.CreatePrinterDC()
dc.StartDoc('Impression avec Python')
dc.StartPage()
dc.TextOut(100, 100, 'Bonjour :)')
dc.EndPage()
dc.EndDoc()

del dc
```

Quelques explications. On commence par créer un dc 'générique' puis on spécialise celui-ci pour l'imprimante, en invoquant sa méthode CreatePrinterDC. Remarquez la dernière ligne qui détruit cet objet. Cette ligne n'est pas vraiment utile ici, puisque tous les objets seront quoi qu'il en soit, détruits après exécution du script. Mais si vous insérez ce code dans un autre, alors la destruction du DC est très importante. Faute de quoi, vous vous heurteriez très vite à un épuisement des ressources de votre système. Les travaux d'impression (ici un simple appel à TextOut) doivent impérativement être encadrés dans un double sandwich. La première couche crée un document du point de vue du pilote de l'imprimante. La seconde couche encadre le travail spécifique à une page. Celle-ci est impri-



> Figure 1: L'application Pythonwin

mée et éjectée dès l'appel à EndPage. Plusieurs pages seront donc chacune, encadrées par un sandwich StartPage - EndPage. Cette méthode est pleinement satisfaisante, tant que l'on imprime du texte ou que l'on trace des lignes. Mais si vous souhaitez imprimer des images, c'est à dire des bitmaps sous Windows, vous ris-

quez de rencontrer des problèmes, en raison d'un bug des MFC eux-mêmes, qui ne gèrent pas bien les bitmaps indépendantes du périphérique. Or les pilotes d'imprimantes ne garantissent en général que l'impression de telles bitmaps et non l'impression des bitmaps dépendantes du périphérique. (Pour plus d'informations vous pouvez vous reporter à un article sur le sujet dans Programmez! N°19 et éventuellement à l'article Q195830 de la MSDN de Microsoft).

La question est maintenant de savoir comment aller un peu plus loin. Les classes Pythonwin sont un reflet assez exact des classes MFC. Il existe une documentation un peu rudimentaire au sein de l'application Pythonwin (figure 1). Comprendre : le vocable Pythonwin recouvre deux choses. Une librairie, et une application qui est en fait un utilitaire de programmation. Le code ci-dessous montre comment créer une fonte :

```
import win32ui

dc = win32ui.CreateDC()
dc.CreatePrinterDC()
dc.StartDoc('Impression avec Python')

font_properties = {
    'name': 'Times'
}

font = win32ui.CreateFont(font_properties)

dc.StartPage()
dc.SelectObject(font)
dc.TextOut(100, 100, 'Bonjour :)')
dc.EndPage()
dc.EndDoc()

del dc
```

## Pour les paresseux

Sous Windows, une excellente façon de procéder paresseusement, est de demander à une application d'imprimer le document pour nous. En effet, les applications bureautiques de Microsoft sont avant tout des serveurs Automation. Ceci signifie qu'ils exposent toutes leurs fonctionnalités sous la forme de méthodes pouvant être invoquées par un programme extérieur. Supposons que vous ayez un gros fichier texte à imprimer. Si vous chargez ce fichier dans Word, ce dernier se chargera de toute la mise en page pour vous. Vous pouvez procéder de même avec des feuilles de calcul Excel ou avec des pages HTML, via FrontPage.

Les extensions Python-Windows proposent un outil pour générer automatiquement des classes enveloppes Python autour de ces serveurs. Supposons que nous voulions travailler avec Word. Démarrer d'abord la première fois Pythonwin, puis dans le menu, sélectionnez 'COM makepy Utility'. Puis dans la liste, recherchez quelque chose comme: 'Microsoft Word 9.0 Object Library' (figure 2). Le numéro peut différer selon votre version de Word. Puis cliquez sur Ok. Pythonwin va alors générer ces fameuses enveloppes. Ceci fait, vous pouvez fermer Pythonwin si vous voulez. Vous n'en avez plus besoin car les classes enveloppes sont totale-



## encadré1

```
# listing partiel wxpythonprint.py

def OnPrintBouton(self, event):
    pd = wxPrintData()
    pd.SetPrinterName("")
    pd.SetOrientation(wxPORTRAIT)
    pd.SetPaperId(wxPAPER_A4)
    pd.SetQuality(wxPRINT_QUALITY_DRAFT)
    pd.SetColour(false) # impression noir et blanc
    pd.SetNoCopies(1)
    pd.SetCollate(true)

    pdd = wxPrintDialogData()
    pdd.SetPrintData(pd)
    pdd.SetSetupDialog(false)
    pdd.SetMinPage(1)
    pdd.SetMaxPage(1)
    pdd.SetFromPage(1)
    pdd.SetToPage(1)
    pdd.SetPrintToFile(false)

    printdialog = wxPrintDialog(self, pdd)
    ok = printdialog.ShowModal()
    if not ok:
        return

    dc = printdialog.GetPrintDC()
    dc.StartDoc("Mon document")
    dc.StartPage()
    dc.SetMapMode(wxMM_POINTS)
    dc.DrawText("Bonjour :-)", 72, 72)
    dc.EndPage()
    dc.EndDoc()
    del dc
```

La deuxième méthode, plus complexe, permet aussi une meilleure gestion des pages. L'encadré 2 en présente le listing complet. En dehors de l'initialisation de wxPrintData et wxPrintDialogData, qui fait partie des meubles, on utilise cette fois un objet wxPrinter, dont la méthode Print s'attend à recevoir un objet dérivant de wxPrintout. Cette classe dérivée DOIT surcharger quatre méthodes :

- le constructeur dans lequel on appelle le constructeur de base.
- HasPage. Tant que cette méthode renvoie true, la fonction callback OnPrintPage est automatiquement appelée.
- GetPageInfo, qui retourne un tuple contenant les premières, dernières pages et le début et la fin de l'impression.



> Figure 4: Sous Linux, tout finit par aboutir dans la file d'impression.

- OnPrintPage, dans laquelle sont effectués les travaux d'impression proprement dits. Ici le DC obtenu est préfabriqué par la classe et NE DOIT PAS être libéré par le programmeur. Cette méthode doit retourner 'true'. Si elle retourne false, le job d'impression est annulé. ■

Frédéric Mazué  
fmazue@programmez.com

## encadré2

```
#!/usr/bin/env python

from wxPython.wx import *

class MonPrintout(wxPrintout):
    def __init__(self, titre):
        wxPrintout.__init__(self, titre)

    def HasPage(self, page):
        if (page <= 1):
            return true
        else:
            return false

    def GetPageInfo(self):
        return (1, 1, 1, 1)

    def OnPrintPage(self, page):
        dc = self.GetDC()
        dc.SetMapMode(wxMM_POINTS)
        dc.DrawText("Bonjour :-)", 72, 72)
        return true

class MainWindow(wxFrame):
    def __init__(self, parent, id, titre):
        wxFrame.__init__(self, parent, -1, titre, size = (500, 500),
            style=wxDEFAULT_FRAME_STYLE | wxNO_FULL
            _REPAINT_ON_RESIZE)
        id_bouton = wxNewId()
        wxButton(self, id_bouton, 'Imprimer', wxPoint(200, 200))
        EVT_BUTTON(self, id_bouton, self.OnPrintBouton)

    def OnPrintBouton(self, event):
        pd = wxPrintData()
        pd.SetPrinterName("")
        pd.SetOrientation(wxPORTRAIT)
        pd.SetPaperId(wxPAPER_A4)
        pd.SetQuality(wxPRINT_QUALITY_DRAFT)
        pd.SetColour(false) # impression noir et blanc
        pd.SetNoCopies(1)
        pd.SetCollate(true)

        pdd = wxPrintDialogData()
        pdd.SetPrintData(pd)
        pdd.SetSetupDialog(false)
        pdd.SetMinPage(1)
        pdd.SetMaxPage(1)
        pdd.SetFromPage(1)
        pdd.SetToPage(1)
        pdd.SetPrintToFile(false)

        printer = wxPrinter(pdd)
        monprintout = MonPrintout("mon objet d'impression")
        printer.Print(self, monprintout, true)

    def OnCloseWindow(self, event):
        self.Destroy()

class App(wxApp):
    def OnInit(self):
        frame = MainWindow(None, -1, "Demo d'impression avec wxPython")
        self.SetTopWindow(frame)
        frame.Show(true)
        return true

app = App(0)
app.MainLoop()
```