Manipulation d'images

Auteur: Olivier Laurent
Contact: oli@aragne.com
Organisation: Python Blanc Bleu Belge
Date: 18/02/2005

Table des matières

1 Introduction

Nous allons tenter, dans ce tutoriel, de créer un objet capable de manipuler des images jpg, gif ou png.

> **Note**
>
> Ce didacticiel repose sur une bibliothèque spécifique : la **Python Imaging Library**, communement appelée PIL. Vous devrez la télécharger et l'installer par vous même. Sources et binaires peuvent être téléchargé ici: http://www.pythonware.com

2 Création de l'objet image

En plus d'importer les habituels modules os et sys, nous devrons importer le module Image de la bibliothèque PIL.

Le constructeur de notre objet prendra le chemin vers le fichier comme argument, ce qui est modélisé par :

```
def __init__(self, path)
```

Comme pour l'objet créé dans la leçon précédente, nous créons aussi une fonction **main()** externe à l'objet. Nous pouvons ainsi tester notre objet en même temps que sa création.

Un fichier image n'est pas ouvert avec la méthode **open** standard mais avec la méthode **open** du module **Image**. Il faut bien entendu tester si le fichier reçu en paramètre de l'objet est effectivement un fichier image. Si ce n'est pas le cas, une erreur d'entrée/sortie sera générée (**IOError**). Cette erreur, interceptons la avec une exception :

```
try:
    self.im = Image.open(self.imgPath)
except IOError:
    error_msg = "Erreur : '%s' n'est pas une image valide" % (self.imgPath, )
    sys.exit(error_msg)
```

Nous ajoutons trois attributs à notre objet : le nom du fichier, le répertoire le contenant et le format du fichier (jpeg, png, ...). Voilà notre module créé.

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

import os, sys
import Image

class Img:

    def __init__(self, path):
        self.imgpath = os.path.abspath(os.path.normpath(path))
        try:
            self.im = Image.open(self.imgpath)
        except IOError:
            error_msg = "Erreur : '%s' n'est pas une image valide" % (self.imgpath, )
            sys.exit(error_msg)

        self.imgname = os.path.basename(self.imgpath)
        self.basepath = os.path.dirname(self.imgpath)
        self.imgformat = self.im.format      # JPEG, GIF, ...

def main():
    try:
        path = sys.argv[1]
    except IndexError:
        sys.exit('Vous devez fournir une image comme argument')

    mon_image = Img(path)
    print mon_image

if __name__ == "__main__":
    main()
```

Ajoutons une méthode **__str__** après la méthode **__init__** pour afficher les caractéristiques de l'objet plus clairement :

```
    def __str__(self):
        out = ''
        out += "Chemin: %s\n" % self.imgpath
        out += "Format: %s\n" % self.imgformat
        return out
```

Il serait intéressant de pouvoir obtenir la taille du fichier et la taille de l'image (largeur, hauteur). Voyons comment comment faire. Obtenir la taille du fichier est très facile, nous l'avons déjà fait dans les leçons précédentes. Deux lignes suffisent:

```
def __init__(self, path):
    [...]
    self.imgweight = os.path.getsize(self.imgpath)
    self.imgsize = self.im.size
    self.img_width = self.imgsize[0]
    self.img_height = self.imgsize[1]
```

Notre méthode **__str__** peut donc se développer:

```
def __str__(self):
    out = ''
    out += "Chemin: %s\n" % self.imgpath
    out += "Poids: %s octets\n" % self.imgweight
    out += "Format: %s\n" % self.imgformat
    out += "Taille (L x H) : %sx%s\n" % (self.img_width, self.img_height)
    return out
```

Voilà ! Notre objet nous donne maintenant les caractéristiques principales de l'image. (Vous pouriez facilement en rajouter d'autres, il suffit d'aller puiser dans le fichier Image.py du package PIL).

Notre objet est, jusqu'à présent assez fainéant. Faisons le travailler un peu.

3 Faire des thumbnails

Dans cette leçon, nous allons découvrir la façon de créer un thumbnail [1] à partir d'une image grâce à la méthode thumbnailize que nous allons écrire. notre méthode thumbailize appelera la méthode thumbnail du module Image (cfr Image.py).

Dans un premier temps, notre méthode acceptera un paramètre: la largeur du thumbnail. Voyons à quoi ressemble cette méthode. Les explications suivent.

[1] c'est-à-dire, pour les non anglophones, une image de petite taille pour publier sur le Web

```
def thumbnailize(self, size):
    """ 'size' est la taille du thumbnail
    """
    self.ratio = self.img_height / self.img_width
    self.size = (size, size * self.ratio)

    thumbname = "tn_%s"%(self.imgname)
    output = os.path.join(self.basepath, thumbname)
    try:
        self.im.thumbnail(self.size)
        self.im.save(output)
    except IOError, error_message:
        errorlog = open("%s%s%s"%(os.curdir, os.sep, "error.log"), 'a')
        msg = "Erreur sur le fichier %s : %s\n\n"%(self.imgpath, error_message)
        print msg
        errorlog.write(msg)
        errorlog.close()
```

Les deux lignes qui suivent la chaîne de documentation sont un subterfuge pour pouvoir sauver le thumbnail avec une largeur fixe. En effet, la méthode thumbnail définie dans la Python Imaging Library possède une mécanique bien à elle pour préserver le ratio hauteur/largeur de l'image. En effet, si votre image est plus haute que large, le paramètre largeur que vous envoyez à la méthode servira en fait à déterminer la hauteur. C'est assez difficile à expliquer d'une manière simple. Le plus simple est que vous consultiez vous-même le module Image pour déterminer comment cela fonctionne.

Donc, si l'on veut être certain que la largeur donnée en paramètre soit effectivement la largeur du thumbnail, il faut multiplier la hauteur par le ratio hauteur/largeur. Le premier élément du tuple largeur/hauteur reçoit donc la vraie largeur (ou la largeur effectivement voulue). Le second élément du tuple largeur/hauteur reçoit, quant à lui, une hauteur factice. Désolé pour ces explications laborieuses.

> **Note**
>
> Vous pouvez utiliser un mécanisme similaire pour que ce soit la hauteur qui soit fixe.

Les lignes

```
self.thumbname = "tn_%s"%(self.imgname)
self.output = os.path.join(self.basepath, self.thumbname)
```

fixent le nom du fichier thumbnail : c'est-à-dire le nom de l'image préfixé de **tn_**. Ainsi que le chemin vers ce fichier.

Les lignes suivantes créent le thumbnail proprement dit et le sauvent sur disque:

```
self.im.thumbnail(self.size)
self.im.save(self.output)
```

Mais comme nous ne sommes jamais à l'abri d'une erreur, nous intercepterons cette erreur à l'aide d'une exception. Si un grand nombre d'erreurs est généré, comme dans le cas d'un traitement par lot (ou *batch processing*), vous voudrez peut-être garder une trace de ces erreurs. C'est pourquoi j'ai pensé à vous : les messages d'erreurs seront sauvés dans le fichier error.log créé à cet effet (et sauvegardé dans le répertoire courant).

Une fonctionalité très utile manque à notre méthode: pouvoir sauver nos thumbnail dans un autre répertoire que le répertoire courant. Procédons:

Nous aurons besoin d'un paramètre supplémentaire à passer à notre méthode. Nous l'appelerons *output* et il sera optionnel. Voici ce que devient notre méthode. Les explications suivent.

```
    def thumbnailize(self, size, output=None):
        """ 'size' est la taille du thumbnail
            'output' est le chemin du fichier à sauvegarder
        """
        self.ratio = self.img_height / self.img_width
        self.size = (size, size * self.ratio)

        thumbname = "tn_%s"%(self.imgname)
        if not output:
            output = os.path.join(self.basepath, thumbname)
        try:
            self.im.thumbnail(self.size)
            self.im.save(output)
        except IOError, error_message:
            errorlog = open("%s%s%s"%(os.curdir, os.sep, "error.log"), 'a')
            msg = "Erreur sur le fichier %s : %s\n\n"%(self.imgpath, error_message)
            print msg
            errorlog.write(msg)
            errorlog.close()
```

La ligne:

```
    if not output:
```

teste si l'argument **output** est passé (par défaut, il est à **None**). Sinon, on le définit comme étant égale au nom de l'image préfixé par **tn_**.

Modifions également notre méthode **main** pour prendre en compte ce second argument:

```
def main():
    try:
        path = sys.argv[1]
    except IndexError:
        sys.exit('Vous devez fournir une image comme argument')

    try:
        output = sys.argv[2]
    except IndexError:
        output = None

    mon_image = Img(path)
    mon_image.thumbnailize(64, output)
```

Et voici notre programme de création de thumbnails au complet:

```
#!/usr/bin/env python
# -*- coding: latin1 -*-

import os, sys
import Image

class Img:

    def __init__(self, path):
        self.imgpath = os.path.abspath(os.path.normpath(path))
        try:
            self.im = Image.open(self.imgpath)
        except IOError:
            error_msg = "Erreur : '%s' n'est pas une image valide" % (self.imgpath, )
            sys.exit(error_msg)

        self.imgname = os.path.basename(self.imgpath)
        self.basepath = os.path.dirname(self.imgpath)
        self.imgformat = self.im.format      # JPEG, GIF, ...

        self.imgweight = os.path.getsize(self.imgpath)
        self.imgsize = self.im.size
        self.img_width = self.imgsize[0]
        self.img_height = self.imgsize[1]

    def thumbnailize(self, size, output=None):
        """ 'size' est la taille du thumbnail
            'output' est le chemin du fichier à sauvegarder
        """
        self.ratio = self.img_height / self.img_width
        self.size = (size, size * self.ratio)

        thumbname = "tn_%s"%(self.imgname)
        if not output:
            output = os.path.join(self.basepath, thumbname)
        try:
            self.im.thumbnail(self.size)
            self.im.save(output)
        except IOError, error_message:
            errorlog = open("%s%s%s"%(os.curdir, os.sep, "error.log"), 'a')
            msg = "Erreur sur le fichier %s : %s\n\n"%(self.imgpath, error_message)
            print msg
            errorlog.write(msg)
            errorlog.close()

    def __str__(self):
        out = ''
        out += "Chemin: %s\n" % self.imgpath
        out += "Poids: %s octets\n" % self.imgweight
        out += "Format: %s\n" % self.imgformat
        out += "Taille (L x H) : %sx%s\n" % (self.img_width, self.img_height)
        return out
```

```
def main():
    try:
        path = sys.argv[1]
    except IndexError:
        sys.exit('Vous devez fournir une image comme argument')

    try:
        output = sys.argv[2]
    except IndexError:
        output = None

    mon_image = Img(path)
    mon_image.thumbnailize(64, output)

if __name__ == "__main__":
    main()
```