Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

martes, 16 de diciembre de 2014

Fundamentos para procesar imágenes con Pillow (II)

Después de ver en el capítulo anterior algunas funciones de edición de imágenes del módulo **Image**, vamos a mostrar qué hacer para convertir nuestras imágenes de formato y de tipo. Para ello, es importante comprender que no siempre es posible cambiar una imagen de formato. En ocasiones para poderlo hacer tendremos que cambiar antes su tipo.

Para concluir mostraremos algunos ejemplos en los que se utilizan funciones que aplican filtros a las imágenes, que pertenecen al módulo **ImageFilter**.

Tipos de imágenes y formatos

Los formatos de las imágenes que se pueden procesar con **Pillow** son los de uso más extendido (BMP, EPS, GIF, IM, JPEG, MSP, PCX, PNG, TIFF, PPM, WebP, ICO, PSD, PDF, etc). Los tipos de imágenes dependen sobre todo del número de bits que se utilizan para representar la unidad de información de una imagen o píxel.

Clave	Tipo de imagen
1	1-bit por pixel, blanco y negro, almacena 1 pixel/byte
L	8-bit por píxel, blanco y negro
P	8-bit por píxel, asignada a cualquier otro modo usando una paleta de colores
RGB	3x8-bit por píxel, color verdadero
RGBA	4x8-bit por píxel, color verdadero con true color con máscara de transparencia
CMYK	4x8-bit por píxel, separación de colores
YCbCr	3x8-bit por píxel, formato de vídeo en color
LAB	3x8-bit por píxel, the L*a*b color space
HSV	3x8-bit por píxel, Hue, Saturation, Value color space)
I	32-bit signed integer pixels
F	32-bit floating point pixels

Convertir imágenes

Convertir el tipo de imagen y su formato:

No todos los tipos de imagen son compatibles con cualquier formato. En el ejemplo siguiente la imagen de tipo "P" y con formato PNG no puede convertirse a JPEG:

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

amapolas.png

```
imagen = Image.open("amapolas.png")  # Se abre la imagen PNG
imagen.mode  # Comprobamos el modo de la imagen: P
imagen.save("amapolas-8bit.jpg") # Se intenta cambiar formato, a JPEG
```

Se intenta cambiar de formato, a JPEG pero se produce la siguiente excepción porque no es posible convertir la imagen:

OSError: cannot write mode P as JPEG

Si embargo, si antes la convertimos a "**RGB**" podremos convertirla a JPEG.

```
imagenrgb = imagen.convert("RGB")
imagenrgb.mode   # RGB
imagenrgb.save("amapolasrgb.jpg")
```

amapolasrgb.jpg

Cambiar el formato de una imagen directamente:

Para convertir de formato una imagen también puede hacerse así:

```
try:
    Image.open("amapolas-in.png").save("amapolas-out.jpg")

except IOError:
    print("No se puede convertir la imagen")
```

Cambiar una imagen de color RGB a Blanco/Negro

```
imagen = Image.open("amapolasrgb.jpg")
imagen.mode -> RGB
imagenbn = imagen.convert("L")
imagenbn.show()
imagenbn.save("amapolasbn.jpg")
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

diciembre 2014 (3)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

Filtros

Para aplicar con **Pillow** un filtro a una imagen se utiliza el módulo **ImageFilter**.

Debemos tener en cuenta que, a veces, cuando se aplica un filtro los cambios a observar serán más o menos sutiles en función al tipo de imagen que procesemos.

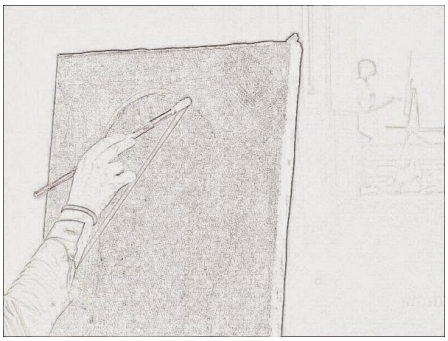
Para los ejemplos utilizaremos la siguiente imagen:

BLUR: Desenfocar una imagen:

```
from PIL import Image, ImageFilter
imagen = Image.open("pintando.jpg")
desenfocada = imagen.filter(ImageFilter.BLUR)
desenfocada.show()
desenfocada.save("desenfocada.jpg")
```

CONTOUR: Marcar contornos:

```
contorneada = imagen.filter(ImageFilter.CONTOUR)
contorneada.show()
contorneada.save("contorneada.jpg")
```



DETAIL: Resaltar detalle:

```
detallar = imagen.filter(ImageFilter.DETAIL)
detallar.show()
detallar.save("detallada.jpg")
```

EDGE_ENHANCE y EDGE_ENHANCE_MORE: Realzar bordes:

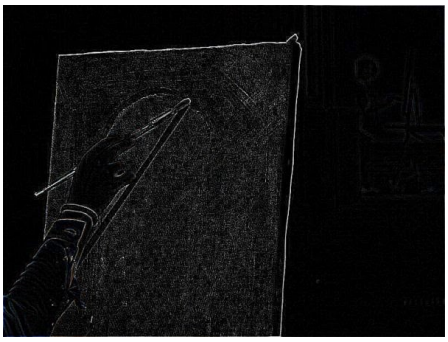
```
realzarbordes = imagen.filter(ImageFilter.EDGE_ENHANCE_MORE)
realzarbordes.show()
realzarbordes.save("realzarbordes.jpg")
```

EMBOSS: Obtener relieve:

```
relieve = imagen.filter(ImageFilter.EMBOSS)
relieve.show()
relieve.save("relieve.jpg")
```

FIND_EDGES: Buscar límites:

```
limites = imagen.filter(ImageFilter.FIND_EDGES)
limites.show()
limites.save("limites.jpg")
```



SMOOTH y SMOOTH_MORE: Suavizar la imagen:

```
suavizar = imagen.filter(ImageFilter.SMOOTH_MORE)
suavizar.show()
suavizar.save("suavizar.jpg")
```

SHARPEN: Afinar:

```
afinar = imagen.filter(ImageFilter.SHARPEN)
afinar.show()
afinar.save("afinar.jpg")
```

Continuar en: Fundamentos para procesar imágenes con Pillow (y III)

Ir al índice del tutorial de Python

Publicado por Pherkad en 14:37

Etiquetas: Pillow

Entrada más reciente Inicio Entrada antigua