Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

domingo, 5 de noviembre de 2017

Internacionalización del código (y II)

Programas multilingües con gettext

El módulo **gettext** permite que un mismo programa Python pueda estar disponible en varios idiomas, es decir, favorece la internacionalización del código. Este módulo proporciona una implementación de Python compatible con la biblioteca **GNU** gettext que facilita la traducción de los mensajes de un programa y la gestión de los catálogos que contienen las traducciones.

Afortunadamente, existen herramientas de Python y de otras fuentes que permiten extraer los mensajes a traducir de los programas, crear los catálogos de traducciones y utilizar dichos catálogos durante la ejecución de un programa para mostrar los mensajes en el idioma de preferencia del usuario.

Construir un programa con sus mensajes traducidos a varios idiomas

1) Identificar los mensajes a traducir

La función **_()** se utiliza para identificar en el código fuente las cadenas de los mensajes a traducir durante la ejecución de un programa.

programa.py:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#

import gettext

idiomas = []

# Configurar el acceso al catálogo de mensajes
t = gettext.translation('programa',
                        'locale',
                        languages=idiomas,
                        fallback=True,)
_ = t.gettext

print(_('Ejecutando programa...'))
print(_('El programa ha finalizado correctamente.'))
```

Los textos que se incluyan en las funciones **_()** serán los mensajes a sustituir por los equivalentes en el idioma deseado, que se obtendrán del catálogo siempre que éste esté disponible. Si no existe el catálogo se imprimirá el mensaje original:

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

$ python3 programa.py

```
Ejecutando programa...
El programa ha finalizado correctamente.
```

2) Crear una plantilla de traducción

Una **plantilla de traducción** es un archivo con la extensión **.pot** que se genera a partir de un programa fuente Python (**.py**). Consta de una cabecera con información relativa a su contenido y la recopilación de todos los mensajes a traducir del fuente, en crudo.

Una plantilla se utiliza para crear un archivo con las traducciones para un determinado idioma. Los archivos que se van creando con las traducciones compiladas en distintos idiomas que son leídos en tiempo de ejecución se denominan catálogos.

Para crear una plantilla se puede utilizar el script **pygettext.py** que proporciona Python, la herramienta **xgettext** de GNU, la aplicación **Poedit** de Václav Slavík o las utilidades **po-utils** de François Pinard, entre otros.

Ubuntu

En Ubuntu para crear la plantilla de traducción de **programa.py**, ejecutar:

$ pygettext -d programa programa.py

o bien,

$ xgettext -o programa.pot programa.py

Recuerda: con el comando **which** podemos ver la ubicación de estas herramientas en el sistema, si están instaladas.

$ which pygettext
`/usr/bin/pygettext`

$ which xgettext
`/usr/bin/xgettext`

Windows

En Windows para crear la plantilla con el script pygettext.py, ejecutar:

c:\> python3 pygettext.py -d programa programa.py

Examinando un archivo .pot

Una vez creado el archivo de plantilla **.pot** se puede examinar su contenido con un editor de textos. En el ejemplo siguiente podemos observar que después de las líneas comentadas del comienzo se sitúa la información de la cabecera, a las que siguen después las líneas con los mensajes de texto a traducir.

programa.pot:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR ORGANIZATION
# FIRST AUTHOR , YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"POT-Creation-Date: 2017-11-02 13:27+CET\n"
"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME \n"
"Language-Team: LANGUAGE \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: ENCODING\n"
"Generated-By: pygettext.py 1.5\n"


#: programa.py:13
msgid "Ejecutando programa..."
msgstr ""

#: programa.py:14
msgid "El programa ha finalizado correctamente."
msgstr ""
```

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

Archivo

noviembre 2017 (1)

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

3) Traducir y compilar los mensajes

Para que el programa **programa.py**, además de en castellano/español, esté disponible en inglés y francés tendremos que crear dos copias de la plantilla **.pot**, una para cada idioma e incluir, en cada una de ellas, las traducciones correspondientes.

Cada copia de la plantilla se guardará cambiándole la extensión a **.po** en una carpeta dentro de una estructura de carpetas que sigue un orden preestablecido; en la que los archivos se organizan, básicamente, en carpetas de idiomas y por dominios. El dominio identifica las traducciones específicas de un programa. Por ello, suele tener la misma denominación. Una carpeta de un idioma concreto contiene las traducciones disponibles en ese idioma de múltiples programas multilingües.

Parar crear la estructura de carpetas necesaria al mismo nivel de los archivos .py y .pot y copiar las dos copias de la plantilla (para inglés de EE.UU. y francés de Francia), seguir los siguientes pasos:

Ubuntu

```
mkdir locale
mkdir locale/en_US
mkdir locale/en_US/LC_MESSAGES
mkdir locale/fr_FR
mkdir locale/fr_FR/LC_MESSAGES
cp programa.pot locale/en_US/LC_MESSAGES/programa.po
cp programa.pot locale/fr_FR/LC_MESSAGES/programa.po
```

Windows

```
md locale
md locale\en_US
md locale\en_US\LC_MESSAGES
md locale\fr_FR
md locale\fr_FR\LC_MESSAGES
copy programa.pot locale\en_US\LC_MESSAGES\programa.po
copy programa.pot locale\fr_FR\LC_MESSAGES\programa.po
```

A continuación, abrir con un editor de textos el archivo **locale/en_US/LC_MESSAGES/programa.po** que contendrá las traducciones en inglés. Después, completar la información de la cabecera e insertar los mensajes de texto traducidos entre las comillas en cada línea donde aparece **msgstr** (se encuentran debajo de cada mensaje de texto original). Una vez incorporadas las traducciones, guardar y repetir estos últimos pasos con el archivo **locale/fr_FR/LC_MESSAGES/programa.po** que contendrá las traducciones en francés.

Para las traducciones, si lo precisamos, utilizar un **traductor** como el de Google.

Al finalizar, la edición de los archivos **.po** su contenido debe parecerse a:

Traducciones en inglés en locale/en_US/LC_MESSAGES/programa.po:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR ORGANIZATION
# FIRST AUTHOR , YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: programa 1\n"
"POT-Creation-Date: 2017-11-02 13:27+CET\n"
"PO-Revision-Date: 2017-11-02 13:27+CET\n"
"Last-Translator: Python para impacientes \n"
"Language-Team: US English \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8-bit\n"
"Generated-By: pygettext.py 1.5\n"


#: programa.py:13
msgid "Ejecutando programa..."
msgstr "Running program..."

#: programa.py:14
msgid "El programa ha finalizado correctamente."
msgstr "The program has finished correctly."
```

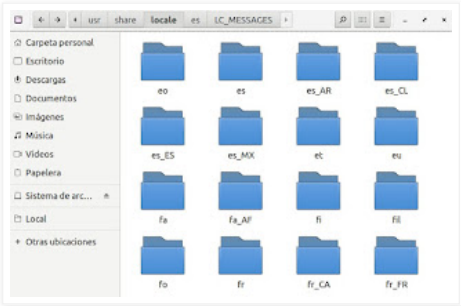
Traducciones en francés en locale/fr_FR/LC_MESSAGES/programa.po:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR ORGANIZATION
# FIRST AUTHOR , YEAR.
#
msgid ""
msgstr ""
"Project-Id-Version: programa 1\n"
"POT-Creation-Date: 2017-11-02 13:27+CET\n"
"PO-Revision-Date: 2017-11-02 13:27+CET\n"
"Last-Translator: Python para impacientes \n"
"Language-Team: FR French \n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8-bit\n"
"Generated-By: pygettext.py 1.5\n"


#: programa.py:13
msgid "Ejecutando programa..."
msgstr "Programme en cours..."

#: programa.py:14
msgid "El programa ha finalizado correctamente."
msgstr "Le programme a bien fini."
```

El dominio es proporcionado por la aplicación o biblioteca y, generalmente, es un valor único que coincide con el nombre de la aplicación o biblioteca. En este caso, el dominio de **programa.py** es '**programa**' porque en el código fuente así se establece en la siguiente línea:

```
t = gettext.translation('programa',
                        'locale',
                        languages=idiomas,
                        fallback=True,)
```

Por otro lado, el directorio de dicho dominio es '**locale**', y aunque en el ejemplo lo hemos creado al mismo nivel que **programa.py**, lo recomendado es utilizar un directorio único accesible en el sistema para que todos los usuarios tengan disponibles los mismos catálogos de traducciones (Si no se proporciona el directorio del dominio se utilizará el directorio predetermnado del sistema). En Ubuntu, lo aconsejable es utilizar la ubicación: **/usr/share/locale**. Si accedemos a dicha ruta reconoceremos que la estructura de carpetas nos resulta familiar:



El valor del idioma lo proporciona el entorno del usuario en tiempo de ejecución a través de una de las variables de entorno **LANGUAGE**, **LC_ALL**, **LC_MESSAGES** o **LANG**, según configuración y plataforma. Su valor podría ser "**es**", "**es_ES**", "**es_ES.UTF-8**", "**fr_FR**", "**en_US**", etc. Nótese que hay lenguas como el español/castellano con sus variantes por países que para poder recoger en las traducciones las peculiaridades de cada uno de ellos se identifican de forma diferente: "**es_ES**" (Español de España), "**es_AR**" (Español de Argentina), "**es_MX**" (Español de México), etc.

Con los archivos **.po** terminados el siguiente paso es compilarlos, uno a uno, para crear los catálogos **.mo**, que son la versión legible por el módulo gettext.

Ubuntu

Para compilar, ejecutar:

cd locale/en_US/LC_MESSAGES
msgfmt -o programa.mo programa.po
cd ../../..
cd locale/fr_FR/LC_MESSAGES
msgfmt -o programa.mo programa.po
cd ../../..

Recuerda: con el comando **which** podemos ver la ubicación de esta herramienta, si está instalada.

$ which msgfmt /usr/bin/msgfmt

Windows

Para compilar con el script **msgfmt.py** (), ejecutar:

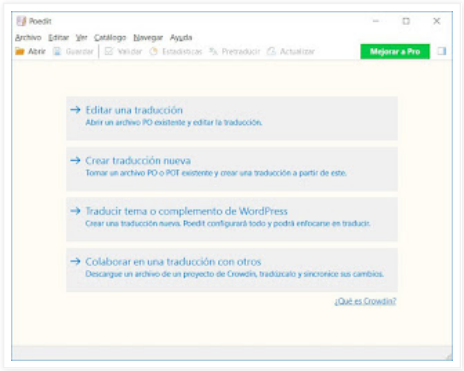
c:\> python3 msgfmt.py -o programa.mo programa.po

Traducir y compilar con Poedit

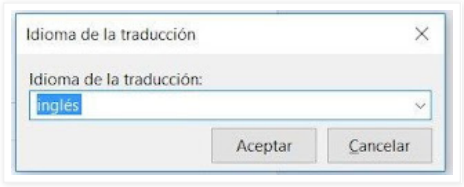
Para que todo sea más fácil podemos instalar la aplicación **Poedit** para traducir los mensajes automáticamente y compilar los catálogos.

Una vez instalado el programa, ejecutar y seguir los siguientes pasos:

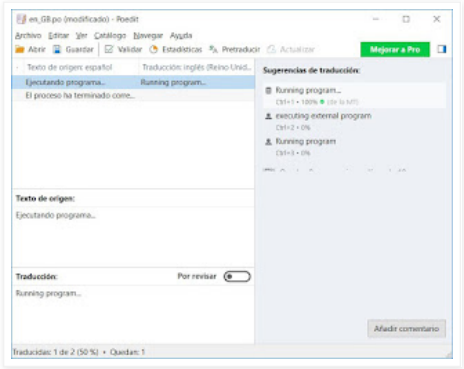
1. Seleccionar el archivo **programa.pot** a generar en el Menú Archivo, Nuevo desde archivo / PO POT:



2. Seleccionar el idioma al que se van a traducir los mensajes:



3. Completar las traducciones seleccionado la sugerencia de traducción más idónea:



4. Para finalizar, guardar el archivo y la aplicación creará los archivos **programa.po** y **programa.mo**.

4) Cargar y usar el catálogo de mensajes apropiado

El último paso consiste en modificar el programa para que cargue y utilice el catálogo de mensajes de un idioma determinado:

Para cargar/usar el catálogo de inglés de Estados Unidos:

programa.py:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
```

```
import gettext

idiomas = ['en_US']

# Configurar el acceso al catálogo de mensajes
t = gettext.translation('programa',
                        'locale',
                        languages=idiomas,
                        fallback=True,)
_ = t.gettext

print(_('Ejecutando programa...'))
print(_('El programa ha finalizado correctamente.'))
```

Ejecutar para comprobar el resultado:

$ python3 programa.py Running program... The program has finished correctly.

Para cargar/usar el catálogo de francés de Francia:

programa.py:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#

import gettext

idiomas = ['fr_FR']

# Configurar el acceso al catálogo de mensajes
t = gettext.translation('programa',
                        'locale',
                        languages=idiomas,
                        fallback=True,)
_ = t.gettext

print(_('Ejecutando programa...'))
print(_('El programa ha finalizado correctamente.'))
```

Ejecutar para comprobar el resultado:

$ python3 programa.py

```
Programme en cours...
Le programme a bien fini.
```

Si se establece un idioma para el que no existe catálogo se obtendrá la salida en el lenguaje original.

Conocer la disponibilidad y usar catálogos

Con el método **find()** es posible conocer si uno o más catálogos de una lista están disponibles para un determinado programa. Con la función **translation()** quedará habilitado el primero de la lista que esté disponible; y asignando a la variable "**_**" la función ***lambda*** (retornando la misma cadena de entrada) se puede volver al idioma orignal del programa.

Utilizando **install()** en el archivo principal de una aplicación se instala globalmente la función **_()** en su espacio de nombres, permitiendo que todos los archivos de la aplicación puedan utilizar dicha función sin tener que "instalar" explícitamente en cada archivo:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#

import gettext

idiomas = ['fr_CA', 'fr_FR']
catalogos = gettext.find('programa',
                         'locale',
                         languages=idiomas,
                         all=True)
print('Catálogo/s:', catalogos)

if catalogos:
    t = gettext.translation('programa', 'locale', languages=idiomas)
```

```
    t.install()
    print(_('Ejecutando programa...'))

_ = lambda a: a

print(_('El programa ha finalizado correctamente.'))
```

Al ejecutar el programa anterior obtendremos una salida que muestra la ubicación del catálogo disponible (francés de Francia), imprimiéndose el primer mensaje en dicho idioma. Después, se deshabilitará para volver al idioma original:

```
Catálogo/s: ['locale/fr_FR/LC_MESSAGES/programa.mo']
Programme en cours...
El programa ha finalizado correctamente.
```

Usar el catálogo del entorno del usuario

En el siguiente ejemplo se carga y usa el catálogo del entorno del usuario, siempre que esté disponible. Em caso contrario se mostrarán los mensajes en el idioma original.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#

import gettext, locale

entorno_usu = locale.setlocale(locale.LC_ALL, '')
idiomas = [entorno_usu]

print('Entorno de Usuario:', entorno_usu)

# Configurar el acceso al catálogo de mensajes
t = gettext.translation('programa',
                        'locale',
                        languages=idiomas,
                        fallback=True,)
_ = t.gettext

print(_('Ejecutando programa...'))
print(_('El programa ha finalizado correctamente.'))
```

Usar el catálogo preferido de una lista de catálogos

En el siguiente ejemplo se prefiere cargar y usar el catálogo de francés de Canadá 'fr_CA' si existe (que no es el caso); o en otro caso el de francés de Francia 'fr_FR'. Si ninguno de los dos existieran los mensajes se mostrarían en el idioma original.

Se ofrecen dos versiones: en la primera se utiliza install() y en la siguiente, no.

Con install()

```
import gettext

idiomas = ['fr_CA', 'fr_FR']
gettext.install('programa', 'locale')
catalogos = gettext.find('programa',
                         'locale',
                         languages=idiomas,
                         all=True)

if catalogos:
    t = gettext.translation('programa', 'locale', languages=idiomas)
    t.install()

print(_('Ejecutando programa...'))
print(_('El programa ha finalizado correctamente.'))
```

Sin install()

```
import gettext

idiomas = ['fr_CA', 'fr_FR']

# Configurar el acceso al catálogo de mensajes
t = gettext.translation('programa', 'locale', languages=idiomas,
    fallback=True,
)
_ = t.gettext

print(_('Ejecutando programa...'))
print(_('El programa ha finalizado correctamente.'))
```

Ir al índice del tutorial de Python

Publicado por Pherkad en 9:52

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua