Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

martes, 14 de febrero de 2017

Las cadenas f

Cuando se programa es frecuente crear cadenas de salida concatenando literales, expresiones y variables que muchas veces obligan a realizar conversiones entre distintos tipos de datos. Desde el nacimiento de Python el lenguaje ha ido incorporando a lo largo del tiempo distintos mecanismos para dar formato a estas cadenas de salida, normalmente, para corregir carencias del lenguaje.

Indudablemente, cualquier evolución que facilita la codificación es siempre bien acogida, pero ofrecer en Python demasiadas opciones para obtener un mismo resultado no suele gustar demasiado porque hace que el lenguaje se vuelva poco *pythonico*.

Esta circunstancia estuvo en el origen de la discusión del PEP 498 que terminó en la propuesta de una nueva forma de interpolar o intercalar expresiones en cadenas y aplicar formatos con conversiones implícitas, basada en lo que ha venido a llamarse **cadenas f**. Un nuevo instrumento disponible desde Python 3.6 que simplifica y unifica en lo posible lo que ya había en este campo.

Las **cadenas f** brindan una forma concisa de construir cadenas formateadas de muy fácil lectura. No obstante, no hay que preocuparse porque el mencionado PEP 498 en ningún caso propone eliminar las opciones de formateo de cadenas que ya existían hasta su aparición.

A continuación, vamos a recordar qué mecanismos hemos utilizado hasta ahora y en qué consiste el nuevo método de formateo de cadenas, para conocer las ventajas que aporta.

Formateo basado en %, en str.format() y string.Template()

Básicamente, los modos para dar formato antes de las **cadenas f** abarcan el **formato de sustitución** que es reconocido por el uso del signo **%** y las funciones **str.format()** y **string.Template()**. A continuación, se ofrecen varios ejemplos para repasar su utilización y señalar algunas limitaciones:

```
# Formateo con %
# --------------

# El formato basado en % sólo es válido para los tipos de
# datos str, int y float:

nombre = 'Claudia'
edad = 35
altura = 1.82

print('Tiene %i años' %edad)  # Tiene 35 años
print('%s tiene %i años y mide %f m.' %(nombre, edad, altura))
# Claudia tiene 35 años y mide 1.820000 m.

# Un objeto datetime (una hora o fecha) y otro tipo de
# objetos no pueden imprimirse sin hacer previamente una
# conversión del tipo de dato.
```

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...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

```
# Formateo con str.format()
# -------------------------

# El formato con str.format() permite fijar la longitud de
# una cadena, aplicar formatos numéricos, establecer la
# alineación, tabular datos y rellenar espacios con un
# determinado caracter:

# Aplica formatos numéricos ajustando la precisión:

valor1 = 8.56767  # asigna flotante
valor2 = 9.45548  # asigna flotante
print('{0:.3} {1:.4}'.format(valor1, valor2))  # 8.57 9.455

# Alinea y rellena con caracteres:

for alin, txt in zip('<^>', ['izquierda', 'centro', 'derecha']):
    print('{0:{fill}{align}30}'.format(txt,
          fill="*",
          align=alin))

# izquierda**********************
# ************centro************
# **********************derecha

# Rellena con guiones bajos:

print('{0:_^30}'.format('Python para impacientes'))
# ___Python para impacientes____

# Tiene el inconveniente de la falta de concisión cuando
# se insertan valores en una cadena:

vel = 120
print('Velocidad permitida: {vel} Km/h.'.format(vel=vel))
# Velocidad permitida: 120 Km/h.

# Incluso en su forma abreviada es algo enrevesado:

print('Velocidad permitida: {} Km/h.'.format(vel))
# Velocidad permitida: 120 Km/h.


# Formateo con string.Template
# ----------------------------

# Con string.Template también se pueden insertar los
# valores de variables y/o expresiones en una cadena,
# aunque la brevedad brilla por su ausencia:

from string import Template
import datetime
perfil = Template('$nom es $pro con un salario de $sal e.')
print(perfil.substitute(nom='Carmen',
                        pro='programadora',
                        sal=1900))
# Carmen es programadora con un salario de 1900 e.

fecha = datetime.date(2017, 2, 10)
contrato = Template('Su contrato se hizo el dia $dia')
print(contrato.substitute(dia=fecha.day))

# Se expresa con bastante claridad pero también con
# demasiada verbosidad.
```

Las cadenas de formato f

A partir de **Python 3.6** las **cadenas f** proporcionan una forma sencilla de integrar variables y expresiones dentro de una cadena empleando una sintaxis muy reducida. A continuación, se muestra su potencial con varios ejemplos:

```
# Una cadena "f" se suele presentar como un literal
# entrecomillado al que le precede el carácter "f" o "F":

f'cadena'

# El resultado de evaluar una cadena "f" se puede
```

que permiten obtener de distintos modos números a...

Archivo

febrero 2017 (2)

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

```
# asignar a una variable:

cadenaf = f'cadena'

# También, se puede extender a varias líneas utilizando
# las triples comillas:

cadenaf = f'''línea1
línea2
línea3'''

# Dentro de una cadena "f" se pueden insertar
# variables y expresiones escribiéndolas entre
# llaves {}:

modelo = 'Orbea Alma'
precio = 650
impuestos = precio * 21/100
print(f'Bicicleta {modelo}: {precio+impuestos} euros')
# Bicicleta Orbea Alma: 786.5 euros

# A partir de Python 3.8 un signo igual '=' tras el
# nombre de una variable inserta tanto el nombre de
# la variable como su valor:

importe = 1300
descuento = 15
print(f'Información de la compra: {importe=} €, {descuento=} %')
# Información de la compra: importe=1300 €, descuento=15 %

# Las dobles llaves se utilizan para expresar el
# nombre de una variable o mostrar literalmente una
# expresión y no el valor resultante de su evaluación:

nombre = 'Claudia'
cadena = f'La variable {{nombre}} contiene {nombre}'
print(cadena)
# La variable {nombre} contiene Claudia

# Para concatenar o unir cadenas de textos con
# cadenas "f" se utiliza el espacio o el signo '+'.
# Los espacios en blanco dentro de una expresión
# son ignorados:

arbol = 'secuoya'
alt = 115
print(f'Una {arbol}' ' mide ' f'{alt} metros')
# La secuoya mide hasta 115 metros

print(f'Una { arbol }' + ' mide ' + f'{ alt } metros')
# La secuoya mide hasta 115 metros

# En la declaración de una cadena el carácter "f"
# se puede combinar con el carácter "r" de las
# cadenas row (crudas) para que no se interpreten
# las secuencias de escape (caracteres especiales
# precedidos de la barra invertida "\"):
# "\n" (salto de línea), "\t" (tabulador), etc.
# Sin embargo, "f" no se puede combinar con "b" o
# "u" que se utilizan para representar cadenas de
# Bytes o caracteres Unicode, respectivamente:

cadenacruda = fr'la línea finaliza con \n'
print(cadenacruda)  # la línea finaliza con \n

# Las secuencias de escape no se pueden incluir
# en una expresión:

print(f'Esta cadena genera un {\"error\"}')
print(f'Esta cadena no genera un {"error"}')

# Después de cada expresión se puede indicar un
# especificador para establecer algún tipo de
# conversión. Las conversiones permitidas se expresan
# con '!s' , '!r' y '!a' que son equivalentes a las
# funciones str(), repr() y ascii(), respectivamente:

novela = 'En busca del unicornio'
cadena = f'La novela se llama {novela!r}'
print(cadena)

# La novela se llama 'En busca del unicornio'
```

```
# Los especificadores de formato permiten fijar
# el espacio reservado para la parte entera de
# una expresión numérica y su precisión decimal:

ancho = 10
precision = 5
numpi = 3.14159265358979323846
print(f"Número PI: {numpi:{ancho}.{precision}}")
# Número PI:     3.1416

# Los especificadores que se utilizan para
# formatear fechas y horas se pueden utilizar
# en una cadena "f":

from datetime import datetime
fecha = datetime.now()
print(f'El partido de tenis se jugará el día {fecha:%d}')
# El partido de tenis se jugará el día 10

# Una expresión de una cadena "f" puede incluir
# llamadas a funciones:

def suma(a,b):
    return a+b

a = 10
b = a * 12
rtdo = f'La suma total es {suma(a,b)}'
print(rtdo)  # La suma total es 130

# ...Y la función Lambda es una función más:

b = 10
h = 5
print(f'Área triángulo: {(lambda b,h: b*h/2)(b,h)}')
# Área triángulo: 25.0

# Con las cadenas "f" la alineación y el rellenado
# se expresa de manera más simple:

blog = 'Python para impacientes'
print(f'{blog:_^30}')  # ___Python para impacientes____
print(f'{blog:_<30}')  # Python para impacientes_______
print(f'{blog:_>30}')  # _______Python para impacientes

# Dentro de una cadena "f" cuando se evalúa una
# expresión el valor obtenido se convierte de forma
# automática a cadena de texto (str) para facilitar
# su inserción:

import time
v1 = 10  # Número entero
v2 = 12.34  # Número con decimales (float)
v3 = 'abc'  # Cadena de texto
v4 = 0xF  # Número hexadecimal
v5 = time.localtime().tm_hour  # Horas
v6 = time.localtime().tm_min  # Minutos
v7 = True  # Valor booleano
v8 = (1, 2, 3)  # Tupla
v9 = {'x':0, 'y':0}  # Diccionario
print(f'{v1+v2} {v3} {v4} {v5}:{v6} {v7} {v8} {v9}')
# 22.34 abc 15 11:31 True (1, 2, 3) {'x': 0, 'y': 0}

# Por último, comentar un detalle importante: las
# expresiones de una cadena "f" se evalúan de
# izquierda a derecha:

# Reloj binario de dos dígitos
def cuenta(cnts):
    if cnts[0]==1 and cnts[1]==1:
        cnts[0]=0
        cnts[1]=0
    elif cnts[1] < 1:
        cnts[1]+=1
    elif cnts[0] < 1:
        cnts[0]+=1
        cnts[1]=0
    else:
        cnts[0]=1
        cnts[1]=1
    return cnts
```

```
cnts = [0, 0]
print('Reloj binario:')
print(f'{cnts}, {cuenta(cnts)},')
print(f'{cuenta(cnts)}, {cuenta(cnts)},')
print(f'{cuenta(cnts)}, {cuenta(cnts)}...')

# Reloj binario:
# [0, 0], [0, 1],
# [1, 0], [1, 1],
# [0, 0], [0, 1]...
```

Ir al índice del tutorial de Python

Publicado por Pherkad en 2:10

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua