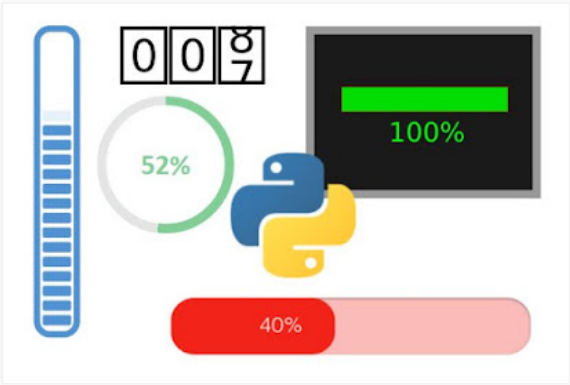
# Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)



**jueves, 19 de abril de 2018**

## Barras de progreso, hélices y contadores




Las **barras de progreso**, las **hélices** y los **contadores** son elementos que se utilizan en un programa para mostrar gráficamente el avance de un proceso. Generalmente, sirven para informar a los usuarios sobre su evolución cuando duran más de unos pocos segundos. Por ello, tienen por objeto dar certidumbre a la persona que espera sobre que un proceso sigue en ejecución informando, a veces, del tiempo transcurrido y del que resta para su finalización. Es un recurso apropiado para mostrar el progreso de unos cálculos prolongados; de una transferencia de archivos; de la instalación de una aplicación, etc.

A continuación, mostraremos algunas de las posibilidades que existen para incorporar este práctico y vistoso elemento a un programa Python. Nos centraremos en los paquetes **Progress** y **TQDM**, ambos disponibles en nuestro repositorio favorito.

### Progress

El paquete **Progress** desarrollado por el griego Georgios Verigakis (Verigak) consta de varios módulos que agrupan un conjunto de clases que permiten declarar barras de progreso, hélices, contadores, etc. en programas para la consola. Son objetos de distintos tipos que tienen la misma finalidad y de los cuales existen algunas variantes.

#### Instalación de Progress

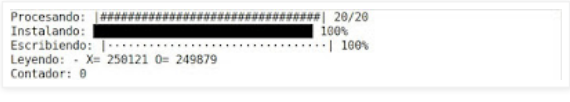
Para instalar Progress con PIP desde la línea de comandos:

```
$ pip install progress
```

#### Objetos Bar, ChargingBar, Spynner y Countdown de Progress

Normalmente los objetos se declaran antes del inicio de un bucle y, después, en cada ciclo del bucle se utiliza el método **Next()** para hacer que avance.

A continuación, varios ejemplos que muestran cómo declarar y usar los objetos: dos tipos de barras, una hélice y un contador. La función **sleep()** de **time** se utiliza a veces para introducir un pequeño retardo de tiempo para que se pueda observar el progreso de los distintos elementos usados.




**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...

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...

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...

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...

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...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute simultáneamente varias operaciones en el mismo espacio de proceso se...

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

Gráficos en IPython
Unos de los motivos que inspiraron el desarrollo de IPython fue contar con una

```
from progress.bar import Bar, ChargingBar
import os, time, random

# Declara un objeto de la clase Bar(). En cada ciclo la barra
# muestra una porción hasta llegar a su máxima longitud en el
# ciclo 20. La barra se representa con el carácter #

bar1 = Bar('Procesando:', max=20)
for num in range(20):
    time.sleep(0.2)
    bar1.next()
bar1.finish()


# Declara un objeto de la clase ChargingBar(). Cuando comienza
# el bucle aparece una barra punteada y durante los ciclos los
# puntos "·" son sustituidos por el carácter "█" hasta alcazar
# el 100%.

bar2 = ChargingBar('Instalando:', max=100)
for num in range(100):
    time.sleep(random.uniform(0, 0.2))
    bar2.next()
bar2.finish()


# Declara un objeto de la clase Bar(). En cada ciclo la barra
# muestra el carácter del atributo "fill" (·) hasta alcanzar
# el 100%.
# Durante el proceso se escribe un archivo de texto con
# 5000 caracteres ('X' y 'O') que son generados por una función
# aleatoria. En este caso el retardo de tiempo es real.
# En el módulo hay otras clases para declarar objetos similares:
# FillingSquaresBar, FillingCirclesBar, IncrementalBar, PixelBar
# y ShadyBar

bar3 = Bar('Escribiendo:', fill='·', suffix='%(percent)d%%')
caracteres = ['X', 'O']
datos = os.getcwd()+os.sep+"datos.txt"
archivo = open(datos, "w")
for i in range(100):
    cadena = ""
    longitud = 5000
    for num in range(longitud):
        cadena += caracteres[random.randint(0, 1)]

    archivo.write(cadena + "\n")
    bar3.next()

bar3.finish()
archivo.close


# Declara un objeto de la clase Spinner(). En cada ciclo una hélice
# gira hasta que se completa la lectura del archivo creado en el
# ejemplo anterior. Cuando se completa la lectura se muestra el
# número total de caracteres encontrados de cada tipo ('X'/'O').
# Una hélice muestra que un proceso se está ejecutando pero no
# no ayuda a prever su final.
# El módulo tienes otras clase para declarar objetos similares:
# PieSpinner, MoonSpinner, LineSpinner y PixelSpinner.

from progress.spinner import Spinner
import time

spinner = Spinner('Leyendo: ')
cuenta_X = 0
cuenta_O = 0
archivo = open(datos, "r")
while True:
    linea = archivo.readline()
    if linea:
        for caracter in linea:
            if caracter == 'X':
                cuenta_X+=1
            elif caracter == 'O':
                cuenta_O+=1
    else:
        break
    time.sleep(0.1)
    spinner.next()
```

herramienta que uniera la posibilidad de realizar cálcu...

**Archivo**

abril 2018 (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

```
print(' X=', cuenta_X, 'O=', cuenta_O)
archivo.close


# Declara un objeto de la clase Countdown(). En cada ciclo un
# contador que comienza en 100 va disminuyendo su valor hasta
# alcanzar 0, que marca el fin del bucle.
# El módulo tiene otras clases para declarar objetos
# similares: Counter, Pie y Stack

from progress.counter import Countdown
import time

contador = Countdown("Contador: ")
for num in range(100):
    contador.next()
    time.sleep(0.05)

print()
```

## TQDM

El módulo **TQDM** de Noam Yorav-Raphael y un grupo de colaboradores a diferencia de progress sólo permite implementar barras de progreso, pero más sofisticadas porque son capaces de ofrecer más información mientras se ejecuta un proceso: el porcentaje ejecutado, el tiempo trascurrido, una estimación del tiempo que resta para su finalización y el número de iteraciones por segundo.

Algunas características interesantes de **TQDM** son que se puede utilizar en el entorno interactivo **IPython**, dentro de un cuaderno **Jupyter** y desde la línea de comandos.

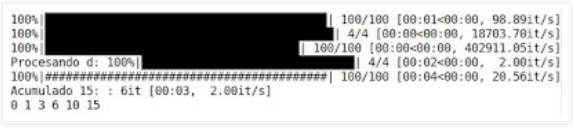
### Instalación de TQDM

Para instalar TQDM con PIP desde la línea de comandos:

```
$ pip install tqdm
```

### Barras de progreso con TQDM

A continuación, varios ejemplos que muestran el uso de las barras de progreso TQDM:




```
# Declara una barra de progreso tqdm. En cada ciclo la barra
# muestra una porción hasta llegar a 100.
# La barra es representada con el carácter "": a su izquierda
# aparece el porcentaje procesado y a su derecha el número de ciclo
# actual, el total de ciclos, el tiempo transcurrido de proceso,
# una estimación del tiempo pendiente y el número de ciclos o
# iteraciones por segundo.

from tqdm import tqdm, trange
import time

for num in tqdm(range(100)):
    time.sleep(0.01)


# Declara una barra de progreso con trange() que es equivalente a
# tqdm(range()) del ejemplo anterior

for num in trange(100):
    time.sleep(0.01)


# Declara una barra de progreso tqdm utilizando una lista,
# correspondiéndose en este caso el número de ciclos con el
# número de elementos de la lista.

texto = ""
for caracter in tqdm(["a", "b", "c", "d"]):
```

```
    texto = texto + caracter


# Declara una barra de progreso tqdm con una lista y
# se establece una descripción para mostrar el elemento
# en curso a la izquierda de la barra.

barra1 = tqdm(["a", "b", "c", "d"])
for caracter in barra1:
    barra1.set_description("Procesando %s" % caracter)
    time.sleep(0.5)


# Declara una barra de progreso tqdm para un bucle de
# 100 ciclos que se actualiza cada 10 ciclos.
# El carácter utilizado para construir la barra es '#'
# y lo establece el atributo ascii con el valor True.

with tqdm(total=100, ascii=True) as barra2:
    for num in range(10):
        barra2.update(10)
        time.sleep(0.5)


# Declara una barra de progreso tqdm en un bucle para
# recorrer y acumular con accumulate() los valores de una lista

from itertools import *
barra3 = tqdm(accumulate([0, 1, 2, 3, 4, 5]))
for acumulado in barra3:
    barra3.set_description("Acumulado %i" % acumulado)
    time.sleep(0.5)


# Declara una barra de progreso tqdm en un bucle para
# recorrer y acumular con accumulate() los valores de una lista.
# En ese ejemplo el atributo disable se establece con el valor
# True para no mostrar la barra de progreso

desactivado = True
barra4 = tqdm(accumulate([0, 1, 2, 3, 4, 5]), disable=desactivado)
for acumulado in barra4:
    barra4.set_description("Acumulado %i" % acumulado)
    if desactivado:
        print(acumulado, end=' ')
    time.sleep(0.5)


# Ejecutado desde la línea de comandos lista los archivos y carpetas
# del directorio /usr/ redirigiendo la salida a un archivo de texto,
# mostrando el número de elementos encontrados y el número de
# iteraciones por segundo.

$ ls /usr/ -R | tqdm >>usr.txt
```

Ir al índice del tutorial de Python

Publicado por Pherkad en 14:09

Etiquetas: IPython, Jupyter, Python3

Entrada más reciente Inicio Entrada antigua

2014-2020 | Alejandro Suárez Lamadrid y Antonio Suárez Jiménez, Andalucía - España
. Tema Sencillo. Con la tecnología de Blogger.