Python Programming

Módulo 3

Ejecución de comandos

Ejecución de comandos o programas

Una tarea frecuente en el desarrollo de *scripts* es la de tener que invocar un programa o comando del sistema desde Python, y ocasionalmente leer el resultado que imprime en pantalla. Para ello la librería estándar provee el módulo `subprocess`.

Por ejemplo, supongamos que queremos ejecutar el siguiente comando de Windows que crea una nueva carpeta en la ruta desde donde se esté ubicado en la terminal:

```
> mkdir nueva_carpeta
```

Si el nombre de la nueva carpeta contiene espacios, entonces debemos encerrarlo entre comillas:

```
> mkdir "Nueva carpeta"
```

Para ejecutar este comandos del sistema operativo desde Python usaremos la función `subprocess.run()` del siguiente modo.

```
import subprocess

subprocess.run(["mkdir", "Nueva carpeta"],
shell=True)
print("¡La carpeta ha sido creada!")
```

El primer argumento de `run()` es una lista en donde el primer elemento es el nombre del comando o programa que se quiere ejecutar, y los subsiguientes son los argumentos para dicho comando o programa (tal como lo vimos al comienzo de esta clase con `sys.argv`). El parámetro `shell=True` indica que se trata de un comando de la terminal o símbolo del sistema de Windows.

Probemos ahora un comando que imprima información en la consola y queramos analizar desde Python. Por ejemplo, `hostname`, en Windows indica el nombre del host actual, o `whoami`, que imprime el nombre del host y el del usuario.

```
> hostname
DESKTOP-JL5G5BG

> whoami
desktop-jl5g5bg\usuario
```

¿Podremos ejecutar el comando `hostname` desde Python y guardar el resultado (es decir, lo que imprime en consola; en este caso, DESKTOP-JL5G5BG) en una variable?

Sí, de la siguiente manera:

```
p = subprocess.run("hostname",
capture_output=True, encoding="cp850")
# p.stdout contiene el resultado del comando
como una cadena.
print(p.stdout)
```

Este último procedimiento tiene varias diferencias con el primer comando que ejecutamos:

- El primer argumento es una cadena y no una lista. Esto es válido porque el comando no contiene espacios, entonces, se puede pasar directamente una cadena.
- El argumento `capture_output=True` le indica a la función `run()` que debe capturar el resultado del comando (esto es, el nombre del host).
- El argumento `encoding="cp850"` indica la codificación de caracteres usada por defecto por la consola de Windows.
- No es relevante en este momento, simplemente recuerda usarlo tal como lo hemos hecho recién siempre que quieras leer el resultado de un comando desde Python.
- No especificamos el parámetro `shell=True`. Esto es porque `hostname` es en realidad un programa, no un comando. Podemos observar esto escribiendo `where hostname` en la consola, lo cual nos revelará la ubicación del programa (por lo general, `C:\Windows\System32\HOSTNAME.EXE`). Siempre que ejecutemos un programa en lugar de un comando debemos evitar el parámetro `shell`.

Nota. En las distribuciones de Linux la codificación de la terminal es, por lo general, UTF-8, de modo que el argumento pasado a `run()` deberá ser `encoding="utf-8"`. La variable os.name contiene el nombre del sistema operativo, en el cual nos podemos basar para determinar el valor correcto del argumento: `"cp850"` si `os.name == "nt"` o `"utf-8"` si `os.name == "posix"`.

Una vez que el comando se ha ejecutado, Python guardará su resultado en la variable `p.stdout` (cuyo nombre viene de ***st****andar****d out****put*).

Si ejecutamos este mismo código desde la consola interactiva, veremos que p.stdout contiene el resultado del comando.

```
>>> import subprocess
>>> p = subprocess.run("hostname",
capture_output=True, encoding="cp850")
>>> p.stdout
'DESKTOP-JL5G5BG\n'
```

Recordar que podemos eliminar el salto de línea al final de la cadena vía `strip()`:

```
>>> p.stdout.strip()
'DESKTOP-JL5G5BG'
```

El primer argumento de `run()` puede ser la ruta completa a un programa, o directamente el llamado al mismo, por ejemplo:

```
>>>datos =
subprocess.run(["python","--version"],capture
_output=True,encoding="cp850")
>>> datos.stdout
'Python 3.9.5\n'
```

Esto invoca el programa "*python*" con el argumento `--version`, que imprime en consola la versión de Python.

¡Muchas gracias!

¡Sigamos trabajando!