**Ejecutar programas externos desde python**

Existen muchas alternativas para ejecutar programas desde python.  
Aquí dejo unos ejemplos:

1. **import** os
2. os.system('ls /home/')
3. *# Existen multiples variantes del comando os.exec(). Por ejemplo,*
4. *# Ejecuta el programa indicado reemplazando el proceso actual*
5. os.execl('/bin/ls')
6. *# Con un listado de parámetros*
7. *#(ojo: la lista de argumentos empieza en 0, no en 1)*
8. os.execl('/bin/ls', '/bin/ls', '-l', '-i')
9. *# Ejecuta el comando buscando en $PATH y reemplazando el proceso actual*
10. os.execlp('ls', 'ls', '-lisa')
11. *#Creando un nuevo proceso primero*
12. **if** os.fork()==0:
13. os.execlp('ls')
14. **else**:
15. **print** 'Fork has failed!'
17. **import** commands
18. *#Devuelve la salida del programa ejecutado*
19. a = commands.getstatusoutput('ls')
20. aList = a.split('**\n**')

## 23

### [Bajar automáticamente archivos de un FTP con python](http://nosololibresig.blogspot.com.es/2008/05/cmo-bajarse-cosas-un-ftp-con-python.html)

A menudo hay que acceder a un ftp y bajarse gran número de ficheros con un determinado nombre de diferentes directorios. Automatizar este proceso puede ser bastante útil en muchos casos. Usar un cliente FTP gráfico no suele ser de ayuda en estos casos. Los clientes FTP en línea de comandos son **interactivos** por lo que no se pueden realizar scripts con ellos, más allá de [ejecutar una lista de comandos](http://www.inlumineconsulting.com:8080/website/scripting.ftp.html) predefinida.

Este simple script se conecta a un servidor, se posiciona en la carpeta 'CATASTRO', y se accede a todos los directorios con profundidad '1' guardando los ficheros que interesan, en este caso [shapefiles](http://es.wikipedia.org/wiki/Shapefile) llamados MASA, en una carpeta local.

1. *#!/usr/bin/python*
2. # DESCRIPTION: Python script that connects to a FTP server
3. # and gets some Shapefiles from
4. # 'CATRASTRO' folder and subfolders
5. # AUTHOR: Nacho Uve
6. # *DATE : May 2008*
7. *# LICENSE: GPL2 or greater*
9. **from** ftplib **import** FTP
10. ftp = FTP('server.com');
11. ftp.login(user='myuser', passwd='mypassword')
12. ftp.cwd("CATASTRO");
13. *# OPTION: To get a 'ls -l' list*
14. *# dirs = []*
15. *# # Used a callback to 'dirs'*
16. *# ftp.dir(dirs.append)*
18. dirs = ftp.nlst()
19. **for** row **in** dirs:
20. **print** row
21. ftp.cwd(row)
22. # Save shapefiles in the local machine with the first word of the folder as filename
23. ftp.retrbinary('RETR MASA.SHP', open(row.split()[0]+'.shp','wb').write)
24. ftp.retrbinary('RETR MASA.SHX', open(row.split()[0]+'.shx','wb').write)
25. ftp.retrbinary('RETR MASA.DBF', open(row.split()[0]+'.dbf','wb').write)
26. ftp.cwd('..')
27. ftp.quit()

# 6.1. Métodos de formato

### 6.1.1. Convertir a mayúscula la primera letra

**Método**: capitalize()

**Retorna**: una copia de la cadena con la primera letra en mayúsculas.

>>> cadena = "bienvenido a mi aplicación"

>>> print cadena.capitalize()

Bienvenido a mi aplicación

### 6.1.2. Convertir una cadena a minúsculas

**Método**: lower()

**Retorna**: una copia de la cadena en minúsculas.

>>> cadena = "Hola Mundo"

>>> print cadena.lower()

hola mundo

### 6.1.3. Convertir una cadena a mayúsculas

**Método**: upper()

**Retorna**: una copia de la cadena en mayúsculas.

>>> cadena = "Hola Mundo"

>>> print cadena.upper()

HOLA MUNDO

### 6.1.4. Convertir mayúsculas a minúsculas y viceversa

**Método**: swapcase()

**Retorna**: una copia de la cadena convertidas las mayúsculas en minúsculas y viceversa.

>>> cadena = "Hola Mundo"

>>> print cadena.swapcase()

hOLA mUNDO

### 6.1.5. Convertir una cadena en Formato Título

**Método**: title()

**Retorna**: una copia de la cadena convertida.

>>> cadena = "hola mundo"

>>> print cadena.title()

Hola Mundo

### 6.1.6. Centrar un texto

**Método**: center(longitud[, "caracter de relleno"])

**Retorna**: una copia de la cadena centrada.

>>> cadena = "bienvenido a mi aplicación".capitalize()

>>> print cadena.center(50, "=")

===========Bienvenido a mi aplicación============

>>> print cadena.center(50, " ")

Bienvenido a mi aplicación

### 6.1.7. Alinear texto a la izquierda

**Método**: ljust(longitud[, "caracter de relleno"])

**Retorna**: una copia de la cadena alineada a la izquierda.

>>> cadena = "bienvenido a mi aplicación".capitalize()

>>> print cadena.ljust(50, "=")

Bienvenido a mi aplicación=======================

### 6.1.8. Alinear texto a la derecha

**Método**: rjust(longitud[, "caracter de relleno"])

**Retorna**: una copia de la cadena alineada a la derecha.

>>> cadena = "bienvenido a mi aplicación".capitalize()

>>> print cadena.rjust(50, "=")

=======================Bienvenido a mi aplicación

>>> print cadena.rjust(50, " ")

Bienvenido a mi aplicación

### 6.1.9. Rellenar un texto anteponiendo ceros

**Método**: zfill(longitud)

**Retorna**: una copia de la cadena rellena con ceros a la izquierda hasta alcanzar la longitud final indicada.

>>> numero\_factura = 1575

>>> print str(numero\_factura).zfill(12)

08 Mar

## [Funciones de cadena imprescindibles en Python](http://www.codejobs.biz/es/blog/2013/03/08/funciones-de-cadena-imprescindibles-en-python)

Publicado en 08/Mar/2013 [CodeJobs](http://www.codejobs.biz/es/user/CodeJobs) [Python](http://www.codejobs.biz/es/blog/category/python) [0 Comentarios](http://www.codejobs.biz/es/blog/2013/03/08/funciones-de-cadena-imprescindibles-en-python/#comments)

Una de las funciones más comunes que podemos utilizar sobre strings es el cálculo dle número de caracteres que contiene. Por ejemplo:

>>> cadena = "Codejobs"

>>> len(cadena)

8

 Otro ejemplo de función que puede ser invocada, sin necesidad de declarar una variable de tipo string, es print().

Respecto a los métodos con los que cuentan los objetos de tipo string, Python incorpora varios de ellos para poder llevar a cabo funcionalidades básicas relacionadas con cadenas de texto. Entre ellas, contamos con métodos para buscar una subcadena dentro de otra, para reemplazar subcadenas, para borrar espacios en blanco, para pasar de mayúsculas a minúsculas, y viceversa.

La función find() regresa el índice correspondiente al primer carácter de la cadena original que coincide con lo que estamos buscando:

>>> cadena = "abcdef"

>>> cadena.find("b")

1

Si el carácter buscado no existe en la cadena, el método find() nos regresará -1.

Para reemplazar una serie de caracteres por otros, contamos con el método replace(). Por ejemplo:

>>> cadena = "Hola Codejobers"

>>> cadena.replace("Hola", "Adiós")

'Adiós Codejobers'

Algo importante de mencionar es que replace() no altera el valor de la variable sobre la que se ejecuta, es decir el valor de cadena seguirá siendo "Hola Codejobers".

Los métodos strip(), lstrip() y rstrip() nos ayudan a eliminar todos los espacios en blanco, solo los que aparecen a la izquierda y solo los que se encuentran a la derecha, respectivamente (es cómo la función trim() de PHP):

>>> cadena = " esta cadena tiene espacios a los lados "

>>> cadena.strip()

"esta cadena tiene espacios a los lados"

>>> cadena.lstrip()

"esta cadena tiene espacios a los lados "

>>> cadena.rstrip()

" esta cadena tiene espacios a los lados"

El método upper() convierte todos los caracteres de una cadena de texto a mayúsculas, mientras que lower() lo hace a minúsculas:

>>> cadena = "Codejobs"

>>> print(cadena.upper())

"CODEJOBS"

>>> print(cadena.lower())

"codejobs"

Relacionados con upper() y lower() hay otro método llamado capitalize(), , el cual sirve para convertir el primer carácter de un string a mayúsculas:

>>> cadena = "codejobs"

>>> cadena.capitalize()

"Codejobs"

Para dividir una cadena en Python se utiliza el método split(), es el equivalente a la función explode() de PHP:

>>> nombres = "Carlos|Cristina|Rodrigo|Hugo"

>>> nombres.split("|")

["Carlos", "Cristina", "Rodrigo", "Hugo"]

El método join() hace lo contrario que split(), es el equivalente a la función implode() de PHP:

>>> caracter = "|"

>>> nombres = ["Carlos", "Cristina", "Rodrigo", "Hugo")

>>> print caracter.join(nombres)

"Carlos|Cristina|Rodrigo|Hugo"

# 9.1. Sobre el objeto File

Al igual que sucede con otras variables, manipular una de ellas como un objeto File, es posible, cuando a ésta, se le asigna como valor un archivo.

**Para asignar a una variable un valor de tipo file**, solo es necesario recurrir a la función integrada open(), la cuál está destinada a la apertura de un archivo.

**La función integrada open()**, recibe dos **parámetros**:

* El primero de ellos, es **la ruta hacia el archivo** que se desea abrir
* Y el segundo, **el modo** en el cual abrirlo

### 9.1.1. Modos de Apertura

El **modo de apertura de un archivo**, está relacionado con el objetivo final que responde a la pregunta ¿para qué estamos abriendo este archivo?. Las respuestas a esta pregunta pueden ser varias. Por ejemplo, podemos querer abrir un archivo para leerlo, para escribirlo, para leerlo y escribirlo, para crearlo si no existe y luego escribir en él, etc.

Es necesario saber, que **cada vez que abrimos un archivo estamos creando un puntero, el cuál se posicionará dentro del archivo en un lugar determinado** (al comienzo o al final) y **este puntero podrá moverse** dentro de ese archivo, eligiendo su nueva posición, **mediante el número de byte** correspondiente.

Este puntero, se creará -en inicio- dependiendo del modo de apertura indicado, el cuál será indicado a la función open() como una string en su segundo parámetro. Entre los \***modos de apertura posibles**, podemos encontrar los siguientes:

| **Indicador** | **Modo de apertura** | **Ubicación del puntero** |
| --- | --- | --- |
| r | Solo lectura | Al inicio del archivo |
| rb | Solo lectura en modo binario | Al inicio del archivo |
| r+ | Lectura y escritura | Al inicio del archivo |
| rb+ | Lectura y escritura en modo binario | Al inicio del archivo |
| w | Solo escritura. Sobreescribe el archivo si existe. Crea el archivo si no existe | Al inicio del archivo |
| wb | Solo escritura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe | Al inicio del archivo |
| w+ | Escritura y lectura. Sobreescribe el archivo si existe. Crea el archivo si no existe | Al inicio del archivo |
| wb+ | Escritura y lectura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe | Al inicio del archivo |
| a | Añadido (agregar contenido). Crea el archivo si éste no existe | Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo |
| ab | Añadido en modo binario (agregar contenido). Crea el archivo si éste no existe | Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo |
| a+ | Añadido (agregar contenido) y lectura. Crea el archivo si éste no existe. | Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo |
| ab+ | Añadido (agregar contenido) y lectura en modo binario. Crea el archivo si éste no existe | Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo |

# 9.2. Métodos del Objeto File

El objeto File, entre sus métodos más frecuentes, dispone de los siguientes:

**Método**: seek(byte) Mueve el puntero hacia el byte indicado.

archivo = open("remeras.txt", "r")

contenido = archivo.read()

# el puntero queda

# al final del documento

archivo.seek(0)

**Método**: read([bytes]) Lee todo el contenido de un archivo. Si se le pasa la longitud de bytes, leerá solo el contenido hasta la longitud indicada.

archivo = open("remeras.txt", "r")

contenido = archivo.read()

print contenido

**Método**: readline([bytes]) Lee una línea del archivo.

archivo = open("remeras.txt", "r")

linea1 = archivo.readline()

print linea1

**Método**: readlines() Lee todas las líneas de un archivo.

archivo = open("remeras.txt", "r")

for linea in archivo.readlines():

print linea

**Método**: tell() Retorna la posición actual del puntero.

archivo = open("remeras.txt", "r")

linea1 = archivo.readline()

mas = archivo.read(archivo.tell() \* 2)

if archivo.tell() > 50:

archivo.seek(50)

**Método**: write(cadena) Escribe cadena dentro del archivo.

archivo = open("remeras.txt", "r+")

contenido = archivo.read()

final\_de\_archivo = archivo.tell()

archivo.write('Nueva linea')

archivo.seek(final\_de\_archivo)

nuevo\_contenido = archivo.read()

print nuevo\_contenido

# Nueva linea

**Método**: writelines(secuencia) Secuencia será cualquier iterable cuyos elementos serán escritos uno por línea.

archivo = open("remeras.txt", "r+")

contenido = archivo.read()

final\_de\_archivo = archivo.tell()

lista = ['Línea 1\n', 'Línea 2']

archivo.writelines(lista)

archivo.seek(final\_de\_archivo)

print archivo.readline()

# Línea 1

print archivo.readline()

# Línea 2

**Método**: close() Cierra un archivo.

archivo = open("remeras.txt", "r")

contenido = archivo.read()

archivo.close()

print contenido

# 9.3. Propiedades del objeto file

Se pueden acceder a las siguientes propiedades del objeto file:

* closed: retorna True si el archivo se ha cerrado. De lo contrario, False.
* mode: retorna el modo de apertura.
* name: retorna el nombre del archivo
* encoding: retorna la codificación de caracteres de un archivo de texto

>>> archivo = open("remeras.txt", "r+")

>>> contenido = archivo.read()

>>> nombre = archivo.name

>>> modo = archivo.mode

>>> encoding = archivo.encoding

>>> archivo.close()

>>> if archivo.closed:

... print "El archivo se ha cerrado correctamente"

... else:

... print "El archivo permanece abierto"

...

El archivo se ha cerrado correctamente

>>> nombre

'remeras.txt'

>>> modo

'r+'

>>> encoding

None

# 9.4. Cerrando archivos de forma automática

Desde la versión 2.5, Python incorpora una manera elegante de trabajar con archivos de forma tal, que se cierren de forma automática sin necesidad de invocar al método close(). Se trata de un bloque with:

with open("remeras.txt", "r") as archivo:

contenido = archivo.read()

print archivo.closed

# True

Cuando una estructura with finaliza, Python, automáticamente invoca al método close(), como se puede ver en el valor de la propiedad closed.

Como también se deja ver en el ejemplo, la sentencia with utiliza un alias para el objeto file, lo que permite acceder al objeto file, justamente, por el alias indicado.

# 10.1. Módulos de sistema

Entre los módulos de sistema que Python nos provee a través de su librería estándar, podemos destacar tres: os, sys, y subprocess. Haremos una breve reseña de cada uno de ellos, a continuación.

### 10.1.1. Módulo os

El módulo os nos permite acceder a funcionalidades dependientes del Sistema Operativo. Sobre todo, aquellas que nos refieren información sobre el entorno del mismo y nos permiten manipular la estructura de directorios (para leer y escribir archivos, ver capítulo 9). [Referencia oficial](http://docs.python.org/library/os.html).

#### 10.1.1.1. Archivos y directorios

El módulo os nos provee de varios métodos para trabajar de forma portable con las funcionalidades del sistema operativo. Veremos a continuación, los métodos más destacados de este módulo.

| **Descripción** | **Método** |
| --- | --- |
| Saber si se puede acceder a un archivo o directorio | os.access(path, modo\_de\_acceso) |
| Conocer el directorio actual | os.getcwd() |
| Cambiar de directorio de trabajo | os.chdir(nuevo\_path) |
| Cambiar al directorio de trabajo raíz | os.chroot() |
| Cambiar los permisos de un archivo o directorio | os.chmod(path, permisos) |
| Cambiar el propietario de un archivo o directorio | os.chown(path, permisos) |
| Crear un directorio | os.mkdir(path[, modo]) |
| Crear directorios recursivamente | os.mkdirs(path[, modo]) |
| Eliminar un archivo | os.remove(path) |
| Eliminar un directorio | os.rmdir(path) |
| Eliminar directorios recursivamente | os.removedirs(path) |
| Renombrar un archivo | os.rename(actual, nuevo) |
| Crear un enlace simbólico | os.symlink(path, nombre\_destino) |

Para ver al módulo os trabajando con funcionalidades del sistema de archivos y directorios, ejecutar python os\_examples de la carpeta sources de este capítulo.

#### 10.1.1.2. El módulo os y las variables de entorno

El módulo os también nos provee de un diccionario con las variables de entorno relativas al sistema. Se trata del diccionario environ:

import os

for variable, valor in os.environ.iteritems():

print "%s: %s" % (variable, valor)

#### 10.1.1.3. os.path

El módulo os también nos provee del submódulo path (os.path) el cual nos permite acceder a ciertas funcionalidades relacionadas con los nombres de las rutas de archivos y directorios. Entre ellas, las más destacadas se describen en la siguiente tabla:

| **Descripción** | **Método** |
| --- | --- |
| Ruta absoluta | os.path.abspath(path) |
| Directorio base | os.path.basename(path) |
| Saber si un directorio existe | os.path.exists(path) |
| Conocer último acceso a un directorio | os.path.getatime(path) |
| Conocer tamaño del directorio | os.path.getsize(path) |
| Saber si una ruta es absoluta | os.path.isabs(path) |
| Saber si una ruta es un archivo | os.path.isfile(path) |
| Saber si una ruta es un directorio | os.path.isdir(path) |
| Saber si una ruta es un enlace simbólico | os.path.islink(path) |
| Saber si una ruta es un punto de montaje | os.path.ismount(path) |

**Nota** Para conocer más sobre os.path, visitar [la documentación oficial](http://docs.python.org/library/os.path.html).

### 10.1.2. Módulo sys

El módulo sys es el encargado de proveer variables y funcionalidades, directamente relacionadas con el intérprete.

#### 10.1.2.1. Variables del módulo sys

Entre las variables más destacadas podemos encontrar las siguientes:

| **Variable** | **Descripción** |
| --- | --- |
| sys.argv | Retorna una lista con todos los argumentos pasados por línea de comandos. Al ejecutar python modulo.py arg1 arg2, retornará una lista: ['modulo.py', 'arg1', 'arg2'] |
| sys.executable | Retorna el path absoluto del binario ejecutable del intérprete de Python |
| sys.maxint | Retorna el número positivo entero mayor, soportado por Python |
| sys.platform | Retorna la plataforma sobre la cuál se está ejecutando el intérprete |
| sys.version | Retorna el número de versión de Python con información adicional |

#### 10.1.2.2. Métodos del módulo sys

Entre los métodos más destacados del módulo sys, podemos encontrar los siguientes:

| **Método** | **Descripción** |
| --- | --- |
| sys.exit() | Forzar la salida del intérprete |
| sys.getdefaultencoding() | Retorna la codificación de caracteres por defecto |
| sys.getfilesystemencoding() | Retorna la codificación de caracteres que se utiliza para convertir los nombres de archivos unicode en nombres de archivos del sistema |
| sys.getsizeof(object[, default]) | Retorna el tamaño del objeto pasado como parámetro. El segundo argumento (opcional) es retornado cuando el objeto no devuelve nada. |

**Nota** Más información sobre el módulo sys, puede obtenerse en [la documentación oficial](http://docs.python.org/library/sys.html).

### 10.1.3. Módulo subprocess

El módulo subprocess es aquel que nos permite trabajar de forma directa con órdenes del sistema operativo.

**Advertencia** El módulo subprocess\*\* se presenta en este capítulo solo con fines educativos, mostrando ejemplos básicos y sencillos. Por lo tanto, se recomienda tener mucho cuidado en el uso de este módulo, desaconsejando su uso para órdenes que puedan comprometer el sistema\*\*.

Entre los métodos más comunes de subprocess, podemos encontrar subprocess.call(). Este método, suele ser útil, para ejecutar órdenes sencillas, como por ejemplo, limpiar la pantalla:

from subprocess import call

call('clear')

El método call, esperará recibir como primer argumento, el comando a ser ejecutado, como se mostró en el ejemplo anterior. Sin embargo, si el comando requiere argumentos, como primer parámetro, call necesitará recibir una lista donde el primer elemento será el comando y el segundo, sus argumentos. Un ejemplo de ello, es el siguiente código encargado de hacer un **listado de archivos y directorios**:

from subprocess import call

comando\_y\_argumentos = ['ls', '-lha']

call(comando\_y\_argumentos)

El módulo subprocess también nos provee del submódulo Popen, el cuál nos permite, no solo ejecutar órdenes al igual que call, sino mantener un mejor control sobre las salidas.

#### 10.1.3.1. Capturando la salida con Popen

El manejo y captura de las salidas, puede resultar un poco complejo. Por eso, intentaremos explicarlo paso a paso a fin de evitar confusiones.

Lo primero que debemos tener en cuenta, es que Popen (al igual que call), como primer argumento, recibirá el comando a ser ejecutado o una lista de dos elementos, donde el primero sea el comando y el segundo, la lista de argumentos. Hasta aquí, no hay diferencia con call. Sin embargo, si la salida del proceso abierto con Popen no es tratada, el proceso quedará abierto.

Por ejemplo, el siguiente código quedaría en ejecución:

from subprocess import Popen

Popen(['ls', '-lha']

A diferencia de call, Popen no es un método de subprocess, sino, un objeto. Cómo tal, la forma correcta de iniciar un proceso con Popen, será entonces, crear un objeto Popen para poder acceder a sus métodos, y así evitar, que el proceso quede abierto en ejecución. De esta forma, creamos el objeto y luego, llamamos al método wait() de Popen, el cual espera a que el proceso finalice.

from subprocess import Popen

proceso = Popen(['ls', '-lha'])

proceso.wait()

Si ejecutamos el código anterior, al igual que sucede con call, la salida obtenida es directamente plasmada en pantalla. Es aquí donde entra en juego, el manejo de las salidas que Popen nos permite hacer.

#### 10.1.3.2. Entradas y salidas que pueden ser capturadas con Popen

* **stdout**: nomenclatura correspondiente a la salida estándar en sistemas UNIX-Like. Es la encargada de almacenar la salida de un programa.
* **stdin**: nomenclatura correspondiente a la entrada estándar en sistemas UNIX-like. Es la encargada de enviar información a un programa.
* **stderr**: al igual que las anteriores, se utiliza como referencia a los errores producidos en la salida de un programa.

#### 10.1.3.3. Utilizando tuberías para capturar la salida

Popen nos permite capturar tanto la entrada como la salida estándar o su error. Para efectuar dicha captura, tanto stdout como stdin y/o stderr se pasan como argumentos clave a Popen. El valor de dichos argumentos, deberá ser un archivo o una tubería que funcione como tal. Y para esto, Popen, también nos provee de una tubería para capturar dichas entradas y salidas, llamada PIPE.

De esta forma, si quisiéramos capturar la salida estándar o error de nuestro código, debemos pasarle a Popen, stdout y stderr como argumentos claves, con PIPE como valor de cada uno de ellos, para lo cual, también debemos importar PIPE:

from subprocess import PIPE, Popen

proceso = Popen(['ls', '-lha'], stdout=PIPE, stderr=PIPE)

Al capturar la salida estándar en una tubería, ya no será necesario poner en espera al proceso, puesto que directamente será capturado por la tubería, permitiéndonos acceder a la lectura tanto de stdout como de stderr, como si se tratara de cualquier otro archivo:

proceso = Popen(['ls', '-lha'], stdout=PIPE, stderr=PIPE)

error\_econtrado = proceso.stderr.read()

listado = proceso.stdout.read()

Capturando la salida, como bien se puede ver en el ejemplo, stdout y stderr, son tratados como archivos (de hecho, lo son ya que hemos utilizado una tubería). Por lo tanto, deben ser cerrados una vez leídos:

proceso = Popen(['ls', '-lha'], stdout=PIPE, stderr=PIPE)

error\_econtrado = proceso.stderr.read()

proceso.stderr.close()

listado = proceso.stdout.read()

proceso.stdout.close()

Luego, podremos manipular dichas lecturas, como cualquier string:

if not error\_encontrado:

print listado

else:

print "Se produjo el siguiente error:\n%s" % error\_encontrado

**Nota** Para conocer más sobre subprocess, ingresar en [la documentación oficial](http://docs.python.org/library/subprocess.html)

# 10.2. Módulos para el programador

### 10.2.1. Debuguear código con Pdb

El módulo pdb nos sirve para controlar paso a paso, la ejecución de nuestros programas. Pdb se utiliza solo para debuguear y su utilidad consiste en permitirnos conocer el lugar exacto y el por qué, nuestro script falla.

Imagina que tienes un archivo que genera errores y no logras descubrir la solución. Puedes importar el módulo pdb y hacer una llamada a pdb.Pdb().set\_trace() en la línea, a partir de la cuál, deseas que tu script comience a "caminar paso a paso" para así, descubrir el error:

# -\*- coding: utf-8 -\*-

import pdb

from subprocess import call, Popen, PIPE

# Limpiar la pantalla

call("clear")

pdb.Pdb().set\_trace()

proceso = Popen(['ls', '-lha'], stdout=PIPE, stderr=PIPE)

error\_encontrado = proceso.stderr.read()

proceso.stderr.close()

listado = proceso.stdout.read()

proceso.stdout.close()

if not error\_encontrado:

print listado

else:

print "Se produjo el siguiente error:\n%s" % error\_encontrado

A partir de la línea donde pdb.Pdb().set\_trace() se encuentra, al ejecutar python tu\_archivo.py, Pdb comenzará a ejecutar tu archivo línea por línea, esperando tu respuesta. Por ejemplo, en el código anterior, tras la ejecución del archivo, la pantalla se limpiará y Pdb comenzará a actuar, mostrándote la línea que sigue:

-> proceso = Popen(['ls', '-lha'], stdout=PIPE, stderr=PIPE)

(Pdb)

Pdb estará esperando tu orden para continuar. Entre las órdenes más usuales, puedes indicar:

n (next) ejecuta el código mostrado y salta a la siguiente línea de tu archivo

s (step) te mostrará paso a paso el camino recorrido

hasta poder ejecutar la siguiente línea de tu archivo

c (continue) ejecuta el archivo hasta encontrar un punto de quiebre

q (quit) abandonar el debugger

Pero no solo un comando, puede ser ordenado a Pdb. También es posible, depurar el código de tu archivo, ejecutando alguna instrucción:

-> listado = proceso.stdout.read()

(Pdb) n

> /home/eugenia/Cursos/Python para Principiantes/Módulo

5/sources/subprocess\_examples/examples.py(13)<module>()

-> proceso.stdout.close()

(Pdb) listado.upper()

'TOTAL 12K\nDRWXRWXR-X 2 EUGENIA EUGENIA 4,0K 2012-07-07 17:34 .\nDRWXRWXR-X 8

EUGENIA EUGENIA 4,0K 2012-07-07 17:34 ..\n-RW-RW-R-- 1 EUGENIA EUGENIA 429

2012-07-07 20:48 EXAMPLES.PY\n'

(Pdb)

**Nota** Puedes obtener más información sobre Pdb ingresando en [la documentación oficial](http://docs.python.org/library/pdb.html).

### 10.2.2. Documentar tu app con pydoc

Con tan solo colocar los docstrings correspondientes en cada módulo y/o función de tu Python app, ejecutar en la terminal pydoc tu\_app será suficiente para acceder a toda la documentación:

~$ pydoc tienda

Help on package tienda:

NAME

tienda

FILE

/home/eugenia/pythonapps/tienda/\_\_init\_\_.py

PACKAGE CONTENTS

\_\_main\_\_

administracion (package)

buscador (package)

core (package)

Alternativamente, también puedes obtener la documentación en formato HTML. Para ello, deberás ejecutar: pydoc -w nombre\_del\_modulo

Otra alternativa, es iniciar un servidor Web local, que te permita navegar por la documentación de tu app. Para ello, simplemente ejecuta pydoc -p n (donde n, es el número del puerto por el cual accederás. Por ejemplo, pydoc -p 8080 inicia el servidor en http://localhost:8080/).

### 10.2.3. Probar el código antes de enviarlo a producción con doctest

El módulo doctest de Python, nos permite indicar fragmentos de código en los comentarios de nuestros módulos, que emulen instrucciones del intérprete interactivo, ejecutándolas de forma tal, que podamos automatizar las pruebas de nuestra aplicación.

import doctest

def sumar\_dos\_numeros(a, b):

"""Suma dos números y retorna su resultado

Argumentos:

a -- primer sumando

b -- segundo sumando

Test:

>>> sumar\_dos\_numeros(25, 10)

35

>>> sumar\_dos\_numeros(30, 20)

50

"""

return a + b

if \_\_name\_\_ == "\_\_main\_\_":

doctest.testmod()

Si vemos el texto debajo de Test:, luce como el intérprete interactivo. Aquí estoy invocando a la función:

>>> sumar\_dos\_numeros(25, 10)

Y debajo, estoy simulando el resultado que arrojaría en el intérprete interactivo. Esto, será interpretado por doctest, como el resultado esperado:

35

Para correr los test, solo bastará con ejecutar:

~$ python modulo.py -v

Y obtendremos un resultado similar a:

eugenia@cochito:~/pythonapps/doctest\_examples$ python suma.py -v

Trying:

sumar\_dos\_numeros(25, 10)

Expecting:

35

ok

Trying:

sumar\_dos\_numeros(30, 20)

Expecting:

50

ok

1 items had no tests:

\_\_main\_\_

1 items passed all tests:

2 tests in \_\_main\_\_.sumar\_dos\_numeros

2 tests in 2 items.

2 passed and 0 failed.

Test passed.

**Nota** Puedes obtener más información sobre doctest visita [la documentación oficial](http://docs.python.org/library/doctest.html).

# Obtener fecha y hora actual en python (datetime - time)

By Diego Caraballo  |  14:13 [11 Comentarios](http://www.pythondiario.com/2014/05/obtener-fecha-y-hora-actual-en-python.html#comment-form)

¿Como puedo saber la **fecha y hora actual en python**? ¿Cuál es la **función o módulo** que debo usar para obtener la hora actual en el [**lenguaje de programación python**](http://www.pythondiario.com/)?  
  
Podemos utilizar el **módulo time** (de bajo nivel) que proporciona diversas funciones relacionadas con el tiempo, sin embargo no todas las funciones están disponibles en todas las plataformas. Por lo tanto es necesario utilizar el **módulo datetime** que proporciona clases (ver [**clases y objetos en python**](http://www.pythondiario.com/2014/10/clases-y-objetos-en-python-programacion.html)) para la manipulación de fechas y horas simples y complejas.  
  
**Obtener hora actual en python con el modulo time**  
  
**Formato:** time.strftime(formato)  
  
**Ejemplo:**  
  
>>>import time  
>>>time.strftime("%H:%M:%S") #Formato de 24 horas  
20:08:40  
  
>>>import time  
>>>time.strftime("%I:%M:%S") #Formato de 12 horas  
08:08:40  
  
**Imprimir la fecha actual:**  
  
Formato: dd/mm/yyyy  
  
>>>import time  
>>>print (time.strftime("%d/%m/%y"))  
22/05/14  
  
**Obtener fecha y hora local en python**  
  
# -\*- coding: utf-8 -\*-  
#!/usr/bin/python  
  
import time  
  
ahora = time.strftime("%c")  
## representacion de fecha y hora  
print "Fecha y hora " + time.strftime("%c")  
  
## representacion del tiempo  
print "Fecha "  + time.strftime("%x")  
  
## representacion de la hora  
print "Hora " + time.strftime("%X")  
  
## Muestra fecha y hora actual a partir de la variable   
print ("Fecha y hora de la variable %s"  % ahora )  
  
**Salida:**  
  
Fecha y hora 05/22/14 20:35:11  
Fecha 05/22/14  
Hora 20:35:11  
Fecha y hora de la variable 05/22/14 20:35:11  
  
**Los distintos formatos**  
  
Las siguientes directivas se pueden utilizar en el formato de cadena:  
  
%a - Nombre del día de la semana  
%A - Nombre del día completo  
%b - Nombre abreviado del mes  
%B - Nombre completo del mes  
%c - Fecha y hora actual  
%d - Día del mes  
%H - Hora (formato 24 horas)  
%I - Hora (formato 12 horas)  
%j - Día del año  
%m - Mes en número  
%M- Minutos  
%p - Equivalente de AM o PM  
%S - Segundos  
%U - Semana del año (domingo como primer día de la semana)  
%w - Día de la semana  
%W - Semana del año (lunes como primer día de la semana)  
%x - Fecha actual  
%X - Hora actual  
%y - Número de año (14)  
%Y - Numero de año entero (2014)  
%Z - Zona horaria  
  
**Obtener la fecha y hora actual en Python usando el módulo datetime**  
  
**Formato:**  
  
>>>ahora = datetime.datetime.now()  
>>>ahora.hour  
>>>ahora.minute  
>>>ahora.year  
>>>ahora.day  
>>>ahora.month  
  
**Ejemplos:**  
  
# -\*- coding: utf-8 -\*-  
#!/usr/bin/python  
  
import datetime  
  
x = datetime.datetime.now()  
  
print ("Fecha y hora = %s" % x)  
  
print ("Fecha y hora en formato ISO = %s" % x.isoformat() )  
  
print (u"Año = %s" %x.year)  
  
print ("Mes = %s" %x.month)  
  
print ("Dia =  %s" %x.day)  
  
print ("Formato dd/mm/yyyy =  %s/%s/%s" % (x.day, x.month, x.year) )  
  
print ("Hora = %s" %x.hour)  
  
print ("Minutos = %s" %x.minute)  
  
print ("Segundos =  %s" %x.second)  
  
print ("Formato hh:mm:ss = %s:%s:%s" % (x.hour, x.month, x.second) )  
  
**Salida:**  
  
Fecha y hora = 2014-05-23 18:05:04.597000  
Fecha y hora en formato ISO = 2014-05-23T18:05:04.597000  
Año = 2014  
Mes = 5  
Dia = 23  
Formato dd/mm/yyyy = 23/5/2014  
Hora = 18  
Minutos = 5  
Segundos = 4  
Formato hh:mm:ss = 18:5:4  
  
Bueno, esto es todo por ahora sobre estos módulos, espero le sea de utilidad. Como siempre; cualquier duda o sugerencia dejen el comentario debajo de esta entrada. Saludos, Diego.

**Trabajando con archivos y directorios con Python**

**Listado de archivos en un directorio**

Para buscar todos los archivos con una extensión, por ejemplo .jpg:

import glob  
lista = glob.glob("\*.jpg")

Para listar todos los archivos de un directorio:

import os  
ficheros = os.listdir('/home/alumno/ejercicios/python') # linux  
ficheros = os.listdir(r'c:\Documents and Settings\alumno\Escritorio\ejercicios\python') #windows: cuidado con el caracter \

Directorio actual:

os.getcwd()  
os.curdir

**Tipos de ficheros**

print michero, 'es un',   
if os.path.isfile(mifichero):  
    print 'fichero'  
if os.path.isdir(mifichero):  
    print 'directorio'  
if os.path.islink(mifichero):  
    print 'enlace'

**Último acceso a un fichero**

ultimo\_acceso = os.path.getatime('foto.jpg')  
ultima\_modificacion = os.path.getmtime('foto.jpg')  
tiempo\_en\_dias = (time.time()- ultimo\_acceso)/ (60\*60\*24)  
print tiempo\_en\_dias

**Eliminar ficheros y directorios**

os.remove('mifoto.jpg')  
for foto in glob.glob('\*.jpg') + glob.glob('\*.tif'):  
    os.remove(foto)

Eliminar directorio:

import shutil  
shutil.rmtree('midirectorio')

**Copiar y renombrar ficheros**

import shutil  
shutil.copy(mifichero, copiafichero)  
  
# copia también tiempo de último acceso y última modificación  
shutil.copy2(mifichero, copiafichero)  
  
# copia un árbol de directorios  
shutil.copytree(raiz\_de\_directorio, copia\_directorio)

**Manipulando los paths y nombres**

Rutas

>>> os.path.split('/home/alumno/python/ejercicios/ej1.py')  
('/home/alumno/python/ejercicios', 'ej1.py')  
>>> os.path.basename('/home/alumno/python/ejercicios/ej1.py')  
'ej1.py'  
>>> os.path.dirname('/home/alumno/python/ejercicios/ej1.py')  
'/home/alumno/python/ejercicios'

amaño del ficheroExtensiones

>>> os.path.splitext('pelicula.avi')  
('pelicula', '.avi')

**Crear y moverse entre directorios**

directorioOriginal = os.getcwd()  
directorio = os.path.join(os.pardir, 'miNuevoDir')  
if not os.path.isdir(directorio):  
    os.mkdir(directorio)  
os.chdir(directorio)  
...  
os.chdir(directorioOriginal) # vuelve al directorio inicial  
os.chdir(os.environ['HOME']) # cambia al directorio home

**Tamaño del fichero**

os.path.getsize ( " /la /es /la /file\_path.txt " )