Librería Estándar

Curso de Python 3 Inicial

César Husillos Rodríguez

IAA-CSIC

1-5 de Abril de 2019

Índice

Índice

¿Qué es?

- Es un añadido del lenguaje PYTHON que incluye
  - Tipos de datos que pueden ser considerados como parte del núcleo de Python (números y listas).
  - Funciones internas y excepciones.
  - Un amplio conjunto de módulos que permiten expandir la funcionalidad de Python.
- Se distribuye junto con el intérprete.
- Con cada nueva versión de Python, se mejora y amplía la funcionalidad.

▸ Librería estándar

Índice

¿Para qué sirve?

- Lo que hace diferente a Python de otros lenguajes de alto nivel es, además de su sencillez, su librería estándar.
- Las decenas de módulos que implementa lo hacen portable a cualquier ordenador (aunque ciertamente algunos son dependientes del Sistema Operativo).
- La librería es **de propósito general**: sirve para cualquier desarrollo.
- Gran parte de nuestro trabajo está ya hecho (REUTILIZACIÓN).
- Los módulos ya han sido probados.
- El tiempo que se invierta en leer la documentación se recupera con creces al programar aplicaciones.

¿Cómo se usa?

- Lo difícil es no perderse en la maraña de módulos existentes.
- Tras buscar en la documentación y localizar el módulo que sirve para nuestra tarea, simplemente lo importamos a nuestro espacio de trabajo y utilizamos lo necesario.
- Visite la web ▸ Python Index Module
  y eche un vistazo rápido a la documentación.

Índice

- Interfaz con el sistema operativo.
- Redirección de entrada y salida (E/S).
- Gestión de procesos en Python.
- Trabajo con fechas y horas.
- Sistema de ficheros. Rutas y búsqueda de patrones.
- Matemáticas.
- Persistencia de la información.
- Números aleatorios.
- Compresión/Descompresión de datos.
- Ficheros CSV.

Interfaz con el Sistema Operativo

Módulo `sys`

- Ya son conocidas sus capacidades para la gestión de parámetros de entrada a scripts Python
  - propiedad `argv`
- También se ha utilizado para modificar la ruta de búsqueda de módulos Python
  - propiedad `path`

Terminar un programa

Forzar el fin de ejecución

Aunque no es programación de estilo, supongamos que, en un momento dado, necesitamos terminar la ejecución de un script

- bien porque el script tarda demasiado en ejecutarse o
- porque se da cierta condición crítica.

Usaremos

```
sys.exit([status])
```

donde `status` es un entero que sirve para informar del motivo de la finalización.

Terminar un programa

Ejemplo

Fichero `salida.py`

```
import sys
if len(sys.argv) < 2:
    sys.exit(1)
print(sys.argv[1:])
```

Desde el intérprete

```
>>> import sys
>>> import subprocess
>>> subprocess.call(['python', 'salida.py'])
1
>>> subprocess.call(['python', 'salida.py', 'parametro'])
['parametro']
0
```

Terminar un programa

Salir del intérprete

También sirve para salir del intérprete

Desde el intérprete

```
cesar@totoro:~/ejemplos$ python
Python 3.7.1 (default, Dec 14 2018, 19:28:38)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> import sys
>>> sys.exit()
cesar@totoro:~/ejemplos$
```

Redirección E/S

- Lo habitual es ejecutar un script y aprovechar para tomar un café. Al regresar vemos por pantalla el resultado.
- Se puede ejecutar varios scripts de forma secuencial (o simultánea), en cuyo caso
  - Las salidas por pantalla se mezclarán o serán demasiado largas para su registro completo en la consola.

  En otras ocasiones se ejecutan programas que pidan información de forma interactiva para poder llevar a cabo sus tareas.
  - ¿Tengo que estar presente para la interacción?

La solución en estos casos pasa por la **Redirección de Entrada/Salida**.

Redirección E/S

Variables

El módulo `sys` se encarga de gestionar las entradas y salidas estándar. Hay tres:

- Entrada estándar (`sys.stdin`)
  Normalmente la entrada se realiza a través del teclado.
- Salida estándar (`sys.stdout`)
  Lo habitual es que la salida de un programa sea la pantalla, aunque también puede ser un fichero.
- Salida estándar de errores (`sys.stderr`)
  Normalmente apunta a la pantalla o a un fichero, como la salida estándar.

Redirección E/S

Ejemplo 1

```
>>> import sys
>>> # guardamos la salida estandar original
>>> out = sys.stdout
>>> fout = open("saludo.out", "w")
>>> # apuntamos la salida estandar a un fichero
>>> sys.stdout = fout
>>> print("Hola mundo")
>>> # Restauramos la salida estandar
>>> sys.stdout = out
>>> fout.close()
>>> print("Hola mundo")
'Hola mundo'
```

Redirección E/S

Ejemplo 1 (Continuación)

Ahora, abrimos el fichero `saludo.out` con un editor de texto plano (*gedit, vi/vim, block de notas*...) y nos encontramos con el texto

```
Hola mundo
```

Ejercicios propuestos

1. Cree un programa que reciba exactamente 2 parámetros: `fichero de entrada` y `fichero de salida`.
   - Si el número de parámetros es distinto de ese valor, muestre el siguiente mensaje de error `'ERROR: Numero de parametros no valido'` y termine el programa.
   - Si el fichero de entrada no existe, imprimir el mensaje `'ERROR: el fichero de entrada no existe'` y termine el programa.
   - Si el fichero de salida no se puede crear, muestre el mensaje `'ERROR: No se ha podido crear el fichero de salida'` y termine.
   - Lea el numero de espacios en blanco que aparecen en el fichero y guarde en el fichero de salida el siguiente mensaje: `'En el fichero FFF se han encontrado XXX espacios en blanco.'`, donde `FFF` es el nombre del fichero leido y `XXX` el numero de espacios encontrados.

Ejercicios propuestos

2. Modifique el ejercicio anterior para que escriba toda la salida por pantalla anterior en el fichero `'out.txt'`.

   NOTA: Redirija la salida estándar `stdout` y la salida estándar de errores `stderr` al fichero de salida `'out.txt'`.

Gestión de procesos

Python se ha hecho popular por varias razones:

- Sintaxis simple y legible.
- Abundante documentación.
- Librería estándar.
- cientos de módulos para trabajo en ciencia, ingeniería, comunicaciones, ...
- Como **lenguaje "pegamento"**. Es sencillo usar código o programas ejecutables escritos en otros lenguajes con el código Python.

Esto último es extremadamente sencillo mediante el uso del módulo `subprocess`.

Procesos

Ejecución

La forma más sencilla de ejecutar un comando del sistema operativo o una llamada a un programa externo es mediante el uso de la función `call`.

```
subprocess.call (comando [,
shell=False, stdin=None, stdout=None,
stderr=None])
```

Ejecución

```
subprocess.call (comando [,
shell=False, stdin=None, stdout=None,
stderr=None])
```

- `comando`, es la orden a ejecutar. Puede ser una lista o una cadena de texto.
- `stdin`, `stdout`, `stderr`, son objetos fichero que representan a la *entrada estándar*, *salida estándar* y *salida estándar de error*.
- `shell`, es un booleano que establece el formato del comando.
  - `shell = False`, comando como lista.
  - `shell = True`, comando como cadena de caracteres.

subprocess.call

Script saludo.py

saludo.py

```
import sys

def miSaludo(name="desconocido"):
    """Retorna un saludo personalizado."""
    print("Hola", name)

if __name__ == '__main__':
    if len(sys.argv) >= 2:
        miSaludo(sys.argv[1])
    else:
        miSaludo()
```

subprocess.call

Ejemplo

shell = False (comando como lista)

```
>>> import subprocess
>>> subprocess.call(['python', 'saludo.py', 'mundo'])
>>> Hola mundo
```

shell = True (comando como cadena)

```
>>> import subprocess
>>> subprocess.call('python saludo.py mundo',
    shell = True)
>>> Hola mundo
```

subprocess.call

Ejemplo 2: salida a fichero.

Salida estándar a fichero

```
>>> fd = open('fichero_salida', 'w')
>>> subprocess.call(['python', 'saludo.py',
'mundo'], stdout = fd)
>>> fd.close ()
```

Ejercicios propuestos

3. Cree el script llamado `miFactorial.py`. El código implementará una función llamada `factorial` que tendrá por argumento un número entero. Como su nombre indica, devolverá el factorial de ese número. Pues bien, su programa se ejecutará desde consola en la forma

   ```
   python miFactorial.py numero
   ```

   y deberá pasar ese `numero` a la función `factorial` y ésta devolver el factorial[1] de ese numero. Para complicarlo un poco más, la salida de ese programa no se imprimirá por pantalla, sino que se guardará en un fichero llamado `'salida.txt'`.
4. Genere un programa que lea el contenido de cualquier directorio que se le pase como parámetro. Muestre su contenido por pantalla (cada elemento en una línea).

   Pista: Use los comandos del sistema operativo para obtener el listado del contenido del directorio (`dir` para Windows, `ls` para MAC y LINUX).

[1] $n! = n * (n-1) * (n-2) * \ldots * 1, n > 1$
$n! = 1, n \in \{0, 1\}$
$\nexists n!, n < 0$

Fechas y horas

- En ocasiones necesito saber la hora a la que se ejecutó un script y/o el tiempo transcurrido.
- A veces necesito saber el intervalo temporal entre dos fechas.
- Quizá necesite parar la ejecución de un programa tras un intervalo de tiempo de espera (una petición web sin respuesta).

Todo esto es posible haciendo uso del módulo de la librería estándar `datetime`.

Módulo `datetime`

Clases del módulo

Clases del módulo `datetime`:

- `date`
  Para gestionar fechas.
- `time`
  Para el trabajo con horas ignorando fechas.
- `datetime`
  Es el módulo principal. Sirve para operar con fechas y horas.
- `timedelta`
  Para realizar operaciones con datos de tipo `datetime`.

Módulo `datetime`

Creación

Para la creación de fechas, horas, fechas-horas e intervalos concretos, usaremos las siguentes funciones

- `datetime.date(año, mes, dia)`
- `datetime.time(hora[, minuto[, segundo[, microsegundo[, tzinfo]]]])`
- `datetime.datetime(año, mes, dia[, hora[, minuto[, segundo[, microsegundo[, tzinfo]]]]])`
- `datetime.timedelta([dias[, segundos[, microsegundos[, millisegundos[, minutos[, horas[, semanas]]]]]]])`
- `datetime.datetime.combine(fecha, hora)`

Módulo `datetime`

Creación

Cuando queremos la fecha-hora, fecha y hora del sistema, usaremos las funciones

- `datetime.datetime.now()`
- `datetime.date.today()`
- `datetime.datetime.now().time()`

pero también podemos usar

- `datetime.datetime.combine(fecha, hora)`

Módulo `datetime`

Atributos (variables)

- Para objetos de tipo `datetime.datetime`
  - `year, month, day, hour, minute, microsecond`.
- Para objetos tipos `datetime.time`
  - `hour, minute, microsecond`.
- Para objetos tipo `datetime.timedelta`
  - `days, seconds, microseconds`.

```
>>> import datetime
>>> today = datetime.datetime.now()
>>> today.year, today.month, today.hour
(2019, 3, 8)
```

Módulo `datetime`

Método `total_seconds()`

- Un método interesante de la clase `timedelta` es

`total_seconds()`

Como su nombre indica, devuelve el intervalo temporal almacenado por el objeto `timedelta` en segundos.

```
>>> import datetime
>>> intervalo = datetime.timedelta(days=1)
>>> intervalo.total_seconds()
86400.0
```

Módulo `datetime`

Generación alternativa de fechas y horas

También se pueden crear objetos de tipo `date` y `time` a partir de objetos `datetime`, mediante los métodos `date()` y `time()` de los objetos de tipo `datetime.datetime`.

```
>>> import datetime
>>> now = datetime.datetime.now()
>>> d = now.date()
>>> type(d)
<type 'datetime.date'>
>>> t = now.time()
>>> type(t)
<type 'datetime.time'>
```

Módulo `datetime`

Generación alternativa de fechas y horas

- Es posible generar objetos `datetime` a partir de cadenas de caracteres que representan fechas.
- Es muy útil cuando se dispone de fechas dadas en un formato no estándar.
- Para eso se usa el método de la clase `datetime`

```
datetime.strptime(cadena, formato)
```
[2]

[2] Más información sobre formatos de fechas en este enlace ▸ formatos

Módulo `datetime`

Formatos de fechas

Ejemplo de formatos

```
>>> import datetime
>>> fecha = '04/05/2019'
>>> dt = datetime.datetime.strptime(fecha,
'%d/%m/%Y')
>>> print(dt)
2019-05-04 00:00:00
>>> # tambien puedo imprimir en otros formatos
>>> print(dt.strftime('%m/%d/%Y'))
05/04/2019
```

Módulo `datetime`

Operaciones con fechas

- Es posible operaciones aritméticas y relacionales básicas.
- Los operandos pueden ser objetos de tipo `datetime`, `date` o `timedelta`.
- Dependiendo de la operación el tipo de operandos tiene sentido o no. Por ejemplo:
  - Tiene sentido restar dos objetos `datetime`, pero no sumarlos.
  - Es razonable sumar o restar un objeto `datetime` o `date` con un intervalo dado por un objeto `timedelta`.
  - También es una operación razonable la suma o resta de dos intervalos temporales (`timedelta`).

Módulo `datetime`

Operaciones con fechas

Operación	Operador	Ejemplo
Suma	+	t1 + t2
Resta	-	t1 - t2
Comparación	>, >=, ==, ! =, <, <=	t1 > t2
Valor negativo	-	-t1
Valor absoluto	abs()	abs(t1)

donde

- `t1` y `t2` son objetos de tipo `datetime`, `date` o `timedelta`.
- El tipo de resultado de las operaciones depende del tipo de los operandos[3].

[3] datetime + timedelta = datetime
date + timedelta = date
datetime - datetime = timedelta

...

Módulo `datetime`

Operaciones con fechas

```
>>> import datetime
>>> date1 = datetime.datetime.now()
>>> # Esperamos unos segundos...
...
>>> date2 = datetime.datetime.now()
>>> datediff = date2 - date1
>>> datediff.days, datediff.seconds,
datediff.microseconds
(0, 13, 925023)
>>> datediff.total_seconds()
13.925023
>>> td1 = datetime.timedelta(days=12)
>>> td2 = datetime.timedelta(days=1)
>>> td1 + td2
datetime.timedelta(13)
```

Ejercicios propuestos

Módulo `datetime`

5. ¿Cuántos días faltan para su cumpleaños?
   Si la clase de hoy finaliza a las 14:00 horas, ¿cuanto tiempo le queda para terminar la clase de hoy?

Ficheros y directorios

- ¿Cómo obtengo el directorio en el que estoy ejecutando órdenes en un script o intérprete?
- ¿Cómo creo y borro directorios y ficheros?
- ¿Cómo accedo al contenido de un directorio?
- ¿Cómo copio, muevo o borro un fichero?¿Cómo realizo esas operaciones de forma recursiva sobre un directorio?

Todo esto y más con los módulos `os` y `shutil`.

Directorios

Directorio de trabajo

Usaremos las siguientes funciones del **módulo `os`**

- `os.getcwd()`
  Devuelve una cadena de texto con el directorio de trabajo.
- `os.chdir(otro_directorio)`
  Permite cambiar el directorio de trabajo a otro que se le pasa como parámetro (`otro_directorio`).
- `os.listdir(ruta)`
  Devuelve una lista con el contenido del directorio dado por el parámetro (`ruta`).

Directorio de trabajo

Ejemplo de uso

```
>>> import os
>>> os.getcwd()
'/home/cesar'
>>> os.chdir('/home/cesar/Downloads')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: [Errno 2] No such file or
directory: '/home/cesar/Downloads'
>>> os.chdir('/home/cesar/Descargas')
>>> os.getcwd()
'/home/cesar/Descargas'
>>> os.listdir(os.getcwd())
['IntranetCS.zip', 'python21.jpg',
'workspace', 'UDIT-IAA-2019-02-27T13-27-33.mysql.gz',
'AQ2015_-_002655.pdf', 'Electromagnetismo.pdf']
```

Directorios

Creación

Usaremos las siguientes funciones del **módulo `os`**

- `os.mkdir(ruta)`
  Crea el directorio `ruta` si la ésta es válida, es decir, si existen todos los directorios anteriores. De no ser así, lanza una excepción del tipo `FileNotFoundError`.
- `os.makedirs(ruta)`
  Crea recursivamente los subdirectorios necesarios para generar `ruta`.

Ambas funciones lanzan una excepción del tipo `OSError` cuando se intenta crear un directorio que ya existe.

Creación de directorio

Ejemplo

```
>>> import os
>>> os.mkdir('ejemplo')
>>> os.mkdir('ejemplo')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: [Errno 17] File exists: 'ejemplo'
>>> os.mkdir('ejemplo/dir1/subdir1')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or
  directory: 'ejemplo/dir1/subdir1'
>>> os.makedirs('ejemplo/dir1/subdir1')
>>> os.chdir('ejemplo/dir1/subdir1')
>>> os.getcwd()
'/home/cesar/Descargas/ejemplo/dir1/subdir1'
```

Directorios

Eliminación

Podemos usar las siguientes funciones del **módulo `os`**

- `rmdir(ruta)`
  Borra el directorio `ruta`.
- `removedirs(ruta)`
  Es un super `rmdir`. Borra el directorio final de `ruta`. Si lo consigue, intenta eliminar el directorio padre del borrado y así, sucesivamente, hasta el directorio base de `ruta`.

Ambas funciones lanzan una excepción del tipo `OSError` cuando se intenta borrar directorios no vacíos.

Pero quizá sea de más ayuda el **módulo `shutil`**, con la función

- `rmtree(ruta)`
  Borra el directorio `ruta` y todo su contenido.

Directorios

Ejemplo de eliminación

```
>>> os.listdir('ejemplo')
['dir1']
>>> os.listdir('ejemplo/dir1')
['subdir1']
>>> os.listdir('ejemplo/dir1/subdir1')
[]
>>> os.rmdir('ejemplo/dir1/subdir1')
>>> os.removedirs('ejemplo')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3.6/os.py", line 238, in removedirs
    rmdir(name)
OSError: [Errno 39] Directory not empty: 'ejemplo'
>>> os.removedirs('ejemplo/dir1') # Atencion que ahora si se puede
>>> os.listdir('ejemplo')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'ejemplo'
```

Directorios

Copia

Usaremos las siguientes funciones del **módulo `shutil`**.

```
shutil.copytree(origen, destino[,
symlinks = False])
```

- Copia recursivamente el contenido del directorio `origen`, en `destino`.
- El directorio de destino **NO DEBE** existir.
- Se creará `destino`, así como los directorios necesarios hasta llegar hasta la ruta dada.
- Si `symlinks = False`, se copian los enlaces simbólicos. Si es `True`, se copia el contenido enlazado en el nuevo directorio.

Directorios

Ejemplo de copia

```
>>> os.chdir('/home/cesar/Desktop')
>>> os.listdir()
['Curso_Astronomia_UDIT', 'inventario_optica.csv']
>>> os.listdir('Curso_Astronomia_UDIT')
['Fotometria.ppt', 'AstronomiaEspacio2.ppt',
'variabilidad.ppt', 'RelatividadGeneral.ppt']
>>> import shutil
>>> shutil.copytree('Curso_Astronomia_UDIT',
'Curso_Astronomia_UDIT_copia')
'Curso_Astronomia_UDIT_copia'
>>> os.listdir('Curso_Astronomia_UDIT_copia')
['Fotometria.ppt', 'AstronomiaEspacio2.ppt',
'variabilidad.ppt', 'RelatividadGeneral.ppt']
```

Directorios

Renombrar

Usamos la función `rename` del **módulo `os`**.

```
os.rename(origen, destino)
```

- Falla cuando `destino` existe.
- Sirve también para ficheros. En este caso, si `destino` existe, lo reemplazará si se dispone de permisos.

Directorios

Ejemplo de renombrado

```
>>> os.chdir('/home/cesar/Desktop')
>>> os.listdir()
['Curso_Astronomia_UDIT',
 'Curso_Astronomia_UDIT_copia',
 'inventario_optica.csv']
>>> os.rename('Curso_Astronomia_UDIT_copia',
   'Curso_Astronomia_UDIT_2')
>>> os.listdir()
['Curso_Astronomia_UDIT',
 'Curso_Astronomia_UDIT_2',
 'inventario_optica.csv']
```

Directorios

Listar contenido

Podemos usar las siguientes funciones del **módulo `os`**.

- `os.listdir(ruta)`
  Ya vista. Devuelve en forma de lista el contenido del directorio dado por `ruta`.
- `os.walk(ruta)`
  Proporciona todos los nombres de ficheros y directorios que cuelgan del directorio `ruta`.
  - Devuelve un objeto de tipo `generator` que se usa en bucles.
  - En cada iteración para cada directorio encontrado a partir de `ruta` produce una tupla

    ```
    (directorio, subdirectorios, ficheros)
    ```

Directorios

Ejemplo de listado de contenido

```
>>> os.listdir('ejemplo')
['dir2', 'dir1']
>>> for el in os.walk('ejemplo'):
...     print(el)
...
('ejemplo', ['dir2', 'dir1'], [])
('ejemplo/dir2', [], ['fichero22.txt', 'fichero21.txt'])
('ejemplo/dir1', ['subdir1'], ['fichero1.txt'])
('ejemplo/dir1/subdir1', [], ['fichero11.txt'])
```

Ficheros

Copiado y borrado

- `shutil.copy(origen, destino)`
  Copia el fichero `origen` en el fichero `destino`. Copia, además, los permisos del fichero original en el destino.
- `os.remove(ruta)`
  Elimina el fichero dado por el parámetro `ruta`.

Ficheros

Ejemplo de copia y borrado

```
>>> import os, shutil
>>> os.listdir('Descargas')
['Anaconda3-2018.12-Linux-x86_64.sh',
   'E10397_RP_AC56_QSG.pdf']
>>> shutil.copy('Descargas/E10397_RP_AC56_QSG.pdf',
   'Descargas/E10397_RP_AC56_QSG_copia.pdf')
'Descargas/E10397_RP_AC56_QSG_copia.pdf'
>>> os.listdir('Descargas')
['Anaconda3-2018.12-Linux-x86_64.sh',
'E10397_RP_AC56_QSG.pdf',
'E10397_RP_AC56_QSG_copia.pdf']
>>> os.remove('Descargas/E10397_RP_AC56_QSG_copia.pdf')
>>> os.listdir('Descargas')
['Anaconda3-2018.12-Linux-x86_64.sh',
 'E10397_RP_AC56_QSG.pdf']
```

Ficheros

Mover ficheros

- `os.rename(origen, destino)`
  Ya mencionada previamente para el caso de directorios. Mueve el fichero `origen` al fichero `destino`.
- `shutil.move(origen, destino)`
  Mueve un fichero o (recursivamente) un directorio `origen` a `destino`.

Ficheros

Ejemplo de copia y borrado

```
>>> import os, shutil
>>> os.listdir('Escritorio')
['cursos_Python']
>>> os.listdir('Escritorio/cursos_Python')
['aplpy', 'libreria_estandar', 'cursoPython3',
'python', 'cursoPython3_final']
>>> shutil.move('Escritorio/cursos_Python/aplpy',
'Escritorio/new_aplpy')
'Escritorio/new_aplpy'
>>> os.listdir('Escritorio')
['cursos_Python', 'new_aplpy']
>>> os.listdir('Escritorio/cursos_Python')
['libreria_estandar', 'cursoPython3', 'python',
'cursoPython3_final']
```

Gestión del sistema de ficheros

- Dada una ruta, ¿cómo separo el nombre del fichero y su directorio?
- ¿Qué hago para verificar que la ruta a un fichero o directorio existe?¿Cómo sé si se trata de un fichero o un directorio?
- ¿Cómo extraigo la extensión de un fichero?
- Dado un fichero, ¿cómo determino la ruta absoluta?
- ¿Cómo genero una nueva ruta a partir de rutas parciales?
- Necesito conocer el tamaño que ocupa un fichero. ¿Cómo se hace?

Todo lo que necesitamos está disponible en el módulo `os.path`

Sistema de ficheros

Separar ruta-nombre de fichero

- `os.path.split(ruta)`

  Esta función descompone `ruta` y la devuelve en forma de tupla de dos elementos en la forma `(directorio, nombre)`

```
>>> import os.path
>>> os.path.split('/home/cesar/fichero.txt')
('/home/cesar', 'fichero.txt')
>>> os.path.split('/home/cesar/Escritorio')
('/home/cesar', 'Escritorio')
```

Sistema de ficheros

Separar la extensión de un fichero de su ruta

- `os.path.splitext(ruta)`

  Esta función descompone `ruta` en forma de tupla de dos elementos:
  `(ruta2, extension)`
  donde `ruta2` contiene `ruta` hasta el último punto que corresponde a la extensión.

```
>>> import os.path
>>> os.path.splitext('/home/cesar/fichero.txt')
('/home/cesar/fichero', '.txt')
>>> os.path.splitext('/home/cesar/Escritorio')
('/home/cesar/Escritorio', '')
```

Sistema de ficheros

Rutas absolutas

- `os.path.abspath(ruta)`
  Esta función obtiene la ruta absoluta `ruta`. Es equivalente concatenar la ruta absoluta del directorio actual con `ruta`.

```
>>> import os.path
>>> os.listdir(os.getcwd())
['.profile', '.bash_logout', '.python_history',
'.viminfo',  '.bash_history', 'Documentos',
'.mozilla', 'anaconda3', '.ipython', '.config',
'.cache', '.ssh', '.bashrc', 'Escritorio']
>>> os.path.abspath('.ipython')
'/home/cesar/.ipython'
>>> os.path.abspath('Escritorio')
'/home/cesar/Escritorio'
```

Sistema de ficheros

Concatenación de rutas

- `os.path.join(directorio, nombre)`
  Esta función compone y devuelve una nueva ruta dada por `directorio + nombre`, donde `+` depende del sistema operativo
  - \, en Windows.
  - /, en Linux o Mac.

```
>>> import os.path
>>> directorio = '/home/users/udit'
>>> directorio2 = '/home/users/dae/'
>>> nombre_fichero = 'cesar/stdout.txt'
>>> os.path.join(directorio, nombre_fichero)
'/home/users/udit/cesar/stdout.txt'
>>> os.path.join(directorio2, nombre_fichero)
'/home/users/dae/cesar/stdout.txt'
```

Sistema de ficheros

Validación de rutas

- `os.path.exists(ruta)`
  Devuelve:
  - `True`, si `ruta` es un fichero o directorio válido.
  - `False`, si `ruta` es un fichero o directorio **no válido o no existente**.

```
>>> import os.path
>>> os.path.exists('/home/cesar/Escritorio')
True
>>> os.path.exists('/home/cesar/fichero_raro.csv')
False
```

Sistema de ficheros

¿Directorio o fichero?

- `os.path.isdir(ruta)`
  Devuelve:
  - `True`, si `ruta` es un directorio válido.
  - `False`, si `ruta` no es un directorio válido o es un fichero.
- `os.path.isfile(ruta)`
  Devuelve:
  - `True`, si `ruta` es un fichero válido.
  - `False`, si `ruta` no es un válido válido o es un directorio.

¿Directorio o fichero?

Ejemplo

```
>>> import os.path
>>> # directorio existente
>>> os.path.isdir('/home/cesar/Escritorio')
True
>>> # directorio inexistente
>>> os.path.isdir('/home/cesar/Escritorio2')
False
>>>  # fichero existente
>>> os.path.isfile('/home/cesar/.bashrc')
True
>>> # fichero inexistente
>>> os.path.isfile('/home/david/.bashrc')
False
>>> # directorio
>>> os.path.isfile('/home/cesar')
False
```

Sistema de ficheros

Tamaño de ficheros

- `os.path.getsize(ruta)`
  Devuelve el tamaño *en bytes* del fichero referenciado por `ruta`.

```
>>> import os.path
>>> fichero = '/home/cesar/.bashrc'
>>> if(os.path.isfile(fichero)):
...     print('El fichero "%(ruta)s" ocupa
%(bytes)d bytes.' % {'ruta': fichero,
'bytes': os.path.getsize(fichero)})
...
El fichero "/home/cesar/.bashrc" ocupa 4345 bytes.
```

Búsquedas aproximadas en el sistema de ficheros

Usaremos la función `glob` del **módulo `glob`**.

- `glob.glob(pathmodel)`
  - Esta función, devuelve una lista con las rutas que cumplen con el patrón aproximado establecido en el parámetro `pathmodel`.
  - El módulo `glob` admite la sintaxis de caracteres comodín de la shell (*wildcars*).

Comodín	Descripción
`*`	Reemplaza a cero o muchos caracteres.
`?`	Reemplaza a un carácter simple.
`[caracteres]`	Reemplaza a cualquiera de los caracteres dentro de los corchetes.
`[!caracteres]`	Reemplaza a cualquiera de los caracteres excepto los dados.

Búsquedas aproximadas

Ejemplo

```
>>> import glob
>>> glob.glob('*.pdf')
['Iniciacion_Python_Clases.pdf',
'curso-python-para-principiantes.pdf']
>>> glob.glob('*python*')
['python-logo.png',
'curso-python-para-principiantes.pdf',
'poo_python.ppt']
>>> glob.glob('*.[pc]??')
['Iniciacion_Python_Clases.pdf',
'curso-python-para-principiantes.pdf',
'pyfits.ppt', 'python-logo.png',
'pystyle.css', 'pyraf.ppt', 'poo_python.ppt']
```

Ejercicios propuestos

Directorios y ficheros

6. Script que lea el contenido de cualquier directorio que se le pase como parámetro. Muestre su contenido por pantalla (cada elemento en una línea).
7. Realice una copia de seguridad de su directorio de trabajo (donde guarda sus scripts de Python). La copia será un directorio llamado `python_backup`. Si ese directorio existe, pida permiso para actualizarlo. Si lo obtiene, hágalo. Si no, muestre el mensaje: `Copia de seguridad abortada`.
8. Cree un directorio llamado `python`. Dentro de ese directorio cree otro llamado `texto` y otro llamado `scripts`. Copie en el directorio `texto` los ficheros de su directorio de trabajo que tengan extensión `.txt`. Copie en el directorio `scripts` todos aquellos ficheros de su directorio de trabajo que tengan extensión `.py`.

Ejercicios propuestos

Directorios y ficheros

9. Liste recursivamente el contenido de su directorio de usuario o de trabajo. Muestre cada ruta en una línea, y al lado, etiquete cada una como `DIRECTORIO` o como `FICHERO`.
   ¿Cuanto tiempo tarda el programa en realizar esa operación?
10. Liste todos los archivos ejecutables
    - (extensión `.exe`) del directorio `C:\Windows\system32` (para WINDOWS).
    - de los directorios `/usr/bin` y `/usr/local/bin` si existen (para LINUX y MAC).
11. Realice un listado de ficheros del directorio actual. Almacene en un fichero llamado `listado_ficheros.txt` la ruta y tamaño de cada fichero con el formato siguiente: ruta;tamaño con una linea para cada fichero encontrado.
12. Use fichero `listado_ficheros.txt` para mostrar todas las extensiones disponibles y el número de ficheros que tienen igual extensión.
13. ¿Cual es el fichero de mayor tamaño de los registrados en `listado_ficheros.txt`?

Matemáticas

- Todo lo referente a operaciones matemáticas está incluido en el módulo `math`.
- Contiene
  - definiciones de constantes.
  - funciones matemáticas básicas.

Matemáticas

Constantes y funciones del módulo `math`

Constante	Descripción
`pi`	Número π.
`e`	Número e (base del logaritmo neperiano).

Función	Descripción
`degrees(x)`	Convierte `x` de radianes a grados.
`radians(x)`	Convierte `x` de grados a radianes.

Matemáticas

Funciones trigonométricas del módulo `math`

Función	Descripción
`cos(x)`[4]	Devuelve el coseno de `x`[5]
`sin(x)`	Devuelve el seno de de `x`.
`tan(x)`	Devuelve la tangente de `x`.
`acos(x)`	Función inversa del coseno de `x`[6].
`asin(x)`	Devuelve el arcoseno de `x`.
`atan(x)`	devuelve la arcotangente de `x`.

[4] También incluye la versión hiperbólica (`cosh(x)` y su inversa (`acosh(x)`) para todas las funciones trigonométricas.

[5] `x` **en radianes**. Lo mismo para las otras funciones trigonométricas `sin` y `tan`.

[6] El resultado viene dado **en radianes**. Lo mismo para las otras funciones inversas trigonométricas.

Matemáticas

Otras funciones del módulo `math`

Función	Descripción
`exp(x)`	Devuelve la exponencial de `x`.
`log(x[,base])`	Si no le pasamos `base`, devuelve el logaritmo neperiano de `x`.
`log10(x)`	Devuelve el logartimo decimal de `x`.
`pow(x, y)`	Devuelve la potencia x^y.
`sqrt(x)`	Devuelve la raíz cuadrada de `x` ($\sqrt{x}$).
`hypot(x, y)`	Devuelve la norma euclídea ($\sqrt{x^2+y^2}$).

Matemáticas

Funciones de redondeo y caracterización del módulo `math`

Función	Descripción
`fabs(x)`	Devuelve el valor absoluto de `x`.
`factorial(x)`	Devuelve el factorial de `x`. Lanza una excepción *ValueError* si `x` no es entero o es menor que 0.
`floor(x)`	Devuelve el mayor entero $<= x$.
`ceil(x)`	Devuelve el menor entero $>= x$.
`fmod(x, y)`	Devuelve el resto de la división entera (de acuerdo al lenguaje C).

Matemáticas

Ejemplo

```
>>> import math
>>> math.pi
3.141592653589793 1
>>> math.e
2.718281828459045 1
>>> math.degrees(math.pi)
180.0
>>> math.radians(360)
6.283185307179586 2
>>> math.exp(1)
2.718281828459045 1
>>> math.log(4,2)
2.0
>>> math.log10(1000)
3.0
>>> math.pow(3,2.5)
15.588457268119896
>>> math.hypot(1,1)
1.4142135623730951
```

Matemáticas

Ejemplo

```
>>> math.fabs(-3.5)
3.5
>>> math.ceil(3.5)
4.0
>>> math.ceil(3.4)
4.0
>>> math.fmod(5,2)
1.0
>>> math.fmod(5.4,2)
1.4000000000000004
>>> math.ceil(3.5)
4.0
>>> math.ceil(3.4)
4.0
```

Ejercicios propuestos

Matemáticas

14. Cree un script que determine el logaritmo decimal de los 1000 primeros numeros enteros. Guardará el resultado en un fichero cuyas filas tendrán el siguiente formato `x y` (espacio entre medias), donde `x` será el entero e `y` su correspondiente logaritmo decimal.
El nombre del fichero de salida será un parámetro de entrada del script.
15. El fichero `randomNumbers.txt` contiene miles de números aleatorios que siguen una distribución gausiana. Determine su media y su desviación estándar.

NOTA:

$$\text{desviación estándar} = \sqrt{\left(\frac{1}{N-1}\right)\sum_{i=1}^{N}(x_i-\mu)^2}$$

donde x_i representa cada uno de los números aleatorios, N es el número total de aleatorios y μ es su valor medio.

Persistencia de objetos

Serialización

- La **serialización** consiste en el almacenado rápido en fichero de cualquier objeto python para su posterior carga y uso.
  - Por ejemplo, un diccionario con valores o rangos de parámetros válidos para la ejecución de un script, un conjunto de templates estelares o extragalácticos, ...
- El módulo `Pickle` codifica y decodifica objetos de cualquier tamaño, por muy complicada que sea su estructura interna[7].
- `cPickle` es otra de las posibilidades.
  Está implementado en C y se estima en unas 1000 veces más veloz que Pickle[8].
- Ambos módulos implementan funciones similares.

[7] Podemos usarlo como superclase de la que heredar.

[8] El problema es que no es heredable.

Serialización

Métodos

Sólo necesitamos dos métodos de este módulo:

- `pickle.dump(objeto, objetoFichero)`. Almacena el `objeto` en un `objetoFichero` con **permisos de escritura**. Tendremos que crearlo de esta forma:

  ```
  objetoFichero = open('rutaFichero', 'w')
  ```

- `pickle.load(objetoFichero)`
  Este método devuelve los objetos almacenados en el `objetoFichero`, que se ha **abierto en modo de lectura**.
  Abrimos un fichero en modo lectura y se lo pasamos a `load`

  ```
  objetoFichero = open('rutaFichero', 'r')
  ```

Serialización

Ejemplo

```
>>> import pickle
>>> data = {'entero': 3, 'float': 2.33,
'complejo': 1+3j, 'lista': range(10), 'tupla':
(1,2,3), 'diccionario': {'nombre': 'Sara',
'edad': 20}}
>>> fout = open('fichero.pick', 'w')
# modo 'wb' si da problemas la funcion 'dump'
>>> pickle.dump(data, fout)
>>> fout.close()
>>> fin = open('fichero.pick', 'r')
# mode 'rb' si se uso 'wb' para escribir
>>> new_data = pickle.load(fin)
>>> new_data['entero']
3
>>> new_data
{'tupla': (1, 2, 3), 'float': 2.3300000000000001,
'complejo': (1+3j), 'entero': 3,
'lista': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
'diccionario': {'edad': 20, 'nombre': 'Sara'}}
```

Serialización

Ejercicios

16. Haga un listado de todos los ficheros con extension `.py` de su directorio de trabajo.
Almacénelos en un diccionario que tendrá como claves los nombres de los ficheros y como valor, la ruta absoluta en el disco. Guarde este diccionario en un fichero llamado `ejercicios.pic` haciendo uso del módulo `Pickle`.
17. Recupere el contenido del fichero `ejercicios.pic`. Determine el número de líneas que ha programado hasta ahora en el curso.

    NOTA: No cuentan las líneas en blanco ni las líneas que sean comentarios.

Números aleatorios

Introducción

- El módulo `random` implementa la generación de *números pseudo-aleatorios* para varias distribuciones (uniforme, normal, lognormal...).
- La primera función que debe llamarse es la de la generación de la semilla de números aleatorios `random.seed([x])` donde `x` es la semilla. Si no se especifica una, se utiliza el valor del tiempo actual del sistema.
  - Semillas distintas, dan distintas secuencias de números aleatorios.

Números aleatorios

Funciones

Cuadro: Funciones para números enteras.

Función	Descripción
`random.randrange([start], stop[, step])`	Devuelve un número aleatorio entre `start` y `stop`, con intervalo entre valores `step`.
`random.randint(a, b)`	Devuelve un número entero entre `a` y `b` (ambos inclusive).

Números aleatorios

Funciones

Cuadro: Funciones para secuencias.

Función	Descripción
`random.choice(seq)`	Devuelve un elemento seleccionado al azar de la lista `seq`.
`random.shuffle(seq)`	Baraja la lista `x`.
`random.sample(population, k)`	Devuelve una muestra de `k` elementos seleccionados al azar y sin repetición de la lista `population`.

Números aleatorios

Funciones

Cuadro: Funciones para distribuciones matemáticas.

Función	Descripción
`random.random()`	Devuelve un número float aleatorio en [0, 1).
`random.uniform(a, b)`	Devuelve un número aleatorio en [a, b] (distribución uniforme).
`random.gauss(mu, sigma)`	Devuelve un número aleatorio según una distribución gausiana de media `mu` y desviación estándar `sigma`.

Números aleatorios

Ejemplo

```
>>> import random
>>> random.random() # float aleatorio en [0, 1)
0.37444887175646646
>>> random.uniform(1, 10) # float aleatorio en [0, 10)
1.180014607311752 3
>>> random.randint(1, 10) # Entero desde 1 a 10
7
>>> # Entero par desde 0 hasta 100
>>> random.randrange(0, 101, 2)
26
>>> random.choice('abcdefghij') # saca un elemento
'c'
>>> items = [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(items) # baraja (reordena)
>>> items
[7, 3, 2, 5, 6, 4, 1]
>>> # Elige 3 elementos
>>> random.sample([1, 2, 3, 4, 5], 3)
[4, 1, 5]
```

Ejercicios propuestos

Números aleatorios

18. Cree un programa que simule el lanzamiento de un dado. Lance el dado un millón de veces. ¿Está cargado?
NOTA: Está cargado si el valor medio de cada puntuación posible difiere notablemente de $\frac{1}{6}$.

19. Cree un programa que simule el lanzamiento de dos dados.

    Lance esos dados 1000 veces.

    - ¿Cuantas veces ha obtenido una suma de dados igual a dos?
    - ¿Cuantas veces ha obtenido una suma de dados igual a un número par?

Ejercicios propuestos

Números aleatorios

20. Suponga una baraja de 10 cartas. Cada una tiene un número que va del 1 al 10. Baraje y reparta 5 para usted y 5 para el ordenador. Muestre su jugada y la del ordenador. Gana el que obtenga mayor puntuación. Imprima el ganador. En caso de empate, juegue de nuevo.
NOTA: No puede repartir una carta que ya halla sido repartida.

21. Juguemos a cara o cruz.
    - Inicialmente dispone de 1000 euros.
    - Cada apuesta es de 100 euros.
    - Si gana duplica su apuesta.
    - Si pierde, se queda sin 100 euros.

    ¿En cuantas jugadas lo pierde todo?

Ficheros comprimidos

Introducción

- Cuando el espacio en disco es limitado o cuando hay que mover ficheros por la red, su tamaño sí importa.
- En la librería estándar de Python existen módulos para el trabajo con los formatos comunes de archivado y compresión, incluyendo
  - `zlib`.
  - `gzip`.
  - `bz2`.
  - `zipfile`.
  - `tarfile`.

Nos centraremos en el **módulo `zipfile`**.

Ficheros comprimidos

Funciones

Función	Descripción
`zipfile.ZipFile (file[, mode])`	Clase de ficheros ZIP. El parámetro `mode` será `r` (lectura) , `w` (escritura), `a` (añadir o crear, si no existe).
`zipfile.is_zipfile(filename)`	Devuelve `True` si `filename` es un fichero ZIP válido. `False` en caso contrario.

Ficheros comprimidos

Métodos

Método	Descripción
`namelist()`	Devuelve la lista de nombres de elementos dentro del paquete ZIP.
`getinfo(name)`	Devuelve un objeto ZipInfo con información del archivo miembro del paquete ZIP llamado `name`.
`open(name[, mode[, pwd]])`	Extrae el miembro llamado `name` del paquete ZIP. `mode` es el modo de extracción ('r' por defecto y siempre sólo lectura). `pwd` es la contraseña para ficheros encriptados.
`close()`	Cierra el fichero ZIP.
`extract(member[, path[, pwd]])`	Extrae el miembro `member` al directorio actual. `path` especifica un directorio alternativo de extracción.

Ficheros comprimidos

Métodos

Método	Descripción
`extractall([path[, members[, pwd]]])`	Extrae todo elcontenido del ZIP al directorio `path`. Por defecto `path` es el directorio actual.
`printdir()`	Muestra el contenido del fichero por pantalla (`stdout`).
`testzip()`	Comprueba el fichero ZIP. devuelve el nombre del primer fichero corrupto del paquete. `none` si todo está bien.
`write(filename[, arcname[, compress_type]]`	Escribe el fichero filename en `arcname`. Si este último no se da, sobreescribe el primero. El paquete ZIP debió abrirse en formato 'w' o 'a'. De no ser así se lanzará la exceptión del tipo `RuntimeError`.

Ficheros comprimidos

Ejemplo

```
>>> fzip = zipfile.ZipFile('cosa.zip','a')
>>> os.listdir(os.getcwd ())
['cosa.zip', 'ejercicio1.py', 'ejercicio2.py',
'ejercicio3.py', 'ejercicio4.py']
>>> fzip.write('ejercicio1.py')
>>> fzip.write('ejercicio2.py')
>>> fzip.printdir()
File Name     Modified             Size
ejercicio1.py 2012-03-07 20:59:22 681
ejercicio2.py 2012-03-08 00:02:45 207
>>> f = fzip.extract('ejercicio1.py')
>>> fzip.close()
```

Ejercicios propuestos

Ficheros comprimidos

22. Cree un fichero comprimido llamado `python_iniciacion.zip`. Contendrá todos los ficheros y subdirectorios que ha creado durante el curso de iniciación a Python.

    Posteriormente, pruebe a abrirlo con cualquiera de los programas disponibles en su sistema operativo que gestione ficheros comprimidos.

Ficheros CSV

Introducción

- Los ficheros CSV no son más que ficheros de texto con un formato especial.
  - Se puede trabajar con ellos igual que hemos hecho antes con la funciones nativas de PYTHON3.
- Su uso está muy extendido a todos los niveles, es por eso que PYTHON3 incluye un módulo específico para su gestión.
  - Algunas aplicaciones tan conocidas como *Microsoft Office*, *LibreOffice* y sistemas gestores de bases de bases de datos como *MySQL*, *ORACLE* o *SQLServer*, son capaces de exportar e importar información en este formato.

Ficheros CSV

Ejemplo

listado.csv

```
Identificador,Nombre,Calificacion
23,Pepe,8.44
24,Manuel,5.68
25,Azucena,9.20
```

Ficheros CSV

Generalidades

- Pese a que la forma de trabajar con los ficheros es la misma (apertura, lectura/escritura, cierre), existen marcadas diferencias entre el modo nativo y el de este módulo en cuanto a:
  - Funciones de entrada/salida.
  - Formato en que devuelve el contenido del fichero.
  - La forma de acceder a registros (líneas) y campos (elementos entre ","").

Ficheros CSV

Apertura

- No existe una función específica para la apertura de un fichero en el módulo `csv`.
  - Se lleva a cabo con la función `open()` de PYTHON3.

```
objetoFichero = open('ruta', 'modo')
```

Ficheros CSV

Lectura

- La lectura se realiza mediante la función `csv.reader()`
- En su forma más simple recibe como parámetro obligatorio un `objetoFichero`, de los que devuelve la función nativa `open()`.
  - El modo de apertura de ese fichero determina las operaciones que se pueden realizar sobre él.
- La función `reader()` devuelve un objeto del tipo `_csv.reader` (iterador), a través del cual se accede al contenido del fichero.

Ficheros CSV

Ejemplo de lectura

```
>>> import csv
>>> fichero = open("listado.csv")
>>> lector = csv.reader(fichero)
>>> for linea in lector:
...     # 'lector' funciona como un iterador
...     print(linea)
...
["Identificador","Nombre","Calificacion"]
["23","Pepe","8.44"]
["24","Manuel","5.68"]
["25","Azucena","9.20"]
```

Ficheros CSV

Lectura

- **Nota importante:** Una vez se han leído todas las filas de un fichero, no se pueden volver a leer a menos que
  1. se cierre el fichero,
  2. se vuelva a abrir en modo lectura.
- Esto es porque el iterador mueve el *punto de lectura* de una línea a la siguiente hasta el final. Una vez alcanzado el final, no se puede mover hacia atrás.

Ficheros CSV

Lectura

- El objeto `reader` del módulo `csv` devuelve una lista de cadenas de caracteres por cada línea leída.
- El acceso a esos elementos se realiza a través del operador `[]` con la posición del campo a extraer.

Extraer los nombres

```
>>> nombres = list()
>>> with open("listado.csv") as fichero:
...     lector = csv.reader(fichero)
...     for linea in lector:
...         nombres.append(linea[1])
...
>>> print(nombres)
["Pepe", "Manuel", "Azucena"]
```

Ficheros CSV

Escritura

- Para la escritura necesitamos de un `objetoFichero` abierto en modo de escritura.
- Ese `objetoFichero` se pasa a la función `csv.writer()` del módulo `csv` como parámetro obligatorio.
- Devuelve un objeto *escritor* que es capaz de escribir en el fichero.
- El objeto *escritor* escribe información a través del método `objEscritor.writerow(listaCadenas)` que recibe como parámetro una lista con las cadenas de caracteres (`listaCadenas`) a escribir.

Ficheros CSV

Cierre de fichero

- Una vez terminadas las operaciones de escritura, se cierra el `objetoFichero` para que
  - los cambios se graben en disco.
  - se libere la memoria que ocupa.

Ficheros CSV

Ejemplo de escritura

```
>>> import csv
>>> fichero_salida = open("listado3.csv", "w")
>>> writer = csv.writer(ficheroSalida,
    delimiter=",", quotechar='"')
>>> info = list()
>>> info.append(["Year", "Month", "Day", "Login"])
>>> info.append(["2016", "10", "31", "ppd"])
>>> info.append(["2018", "2", "12", "mrh"])
>>> for fila in info:
...     writer.writerow(fila)
...
>>> fichero_salida.close()
```

Ficheros CSV

Función `csv.writer()`

- Los parámetros opcionales del método writer() permiten personalizar el formato de escritura de la información. Algunos de ellos son:
  - `delimiter`
    Es el carácter usado para delimitar campos.
  - `quotechar`
    Carácter que delimita el contenido de un campo. Eso permite obviar la presencia de caracteres `delimiter` dentro de un campo.

    Por ejemplo, si `quotechar = comillas` y `delimiter = espacio`, entonces `"hola amigo"` sería considerado como un campo (no como dos, aunque contenga el carácter dado a `delimiter`).

Ficheros CSV

Función `csv.writer()`

- (Continuación)
  - `quoting`
    Determina cuándo deben generarse los caracteres de delimitación de campos (`quotechar`) por parte del `objetoEscritor`.
    Posibles valores:
    - `csv.QUOTE_ALL` (todos los campos).
    - `csv.QUOTE_MINIMAL` (sólo aquellos campos que contienen caracteres especiales).
    - `csv.QUOTE_NONNUMERIC` (para delimitar los campos no numéricos).
    - `csv.QUOTE_NONE` (para no delimitar campos).

Ejercicios propuestos

Ficheros CSV

23. Genere un script que lea el fichero `listado.csv`. Debe agregar una columna llamada `Nota final` y rellenar, para cada línea que figure, un valor aleatorio entre 0 y 10. Guarde el contenido en otro fichero llamado `listado_de_calificaciones_2018.csv`.

Descargue el fichero `NucleosPoblacionAndaluces.csv`. Responda a las siguientes preguntas en diferentes ejercicios:

24. ¿Cuántos Municipios tienen más de 100000 habitantes?¿Cuál es la segunda ciudad más poblada?¿Qué posición ocupa Granada en el ranking de las más pobladas?
25. Escriba los nombres de los 10 municipios con menos población.
26. ¿Cuántos municipios de Sevilla tienen más de 5000 habitantes?

Ejercicios propuestos

Ficheros CSV

27. ¿Cuál es el municipio situado más al Norte? (Usar el valor de la coordenada `Y` que representa la latitud en grados). Proporcione también la provincia a la que pertenece y su población.
28. ¿Cual es el municipio de la provincia de Huelva situado más al Este? ¿Cual es el situado más al Oeste en Huelva?
29. Dígame los nombres de los Municipios más cercano y más lejano a Granada. Para ello debe calcular la distancia en todos ellos y Granada. Por supuesto, Granada no cuenta.
30. ¿Cuántos Municipios hay en un radio de 5 grados de la ciudad de Granada?