Escritura y lectura de archivos

CUIDADO

Tenga cuidado con los siguientes ejercicios. Afectaran los archivos del sistema y pueden eliminar algun archivo que afecte el fucionamiento del equipo. Mantenga este archivo en un directorio independiente para las mas modificaciones en lo archivos se mantengan dentro de este directorio.

Python puede acceder a los archivos del sistema y tanto leer como escribir archvos, ya sea de texto o binarios. Esto permitirá tener un reporitorio de información no volatíl, al que podrems retornar para recuperar la inforamación almacenada.

La manipulación de archivos requiere la importación del módulo para el control del sistema operativo:

In [2]:

```
import os
```

Una vez importado el modulo, podemos tener acceso a ciertas consultas importantes sobre el systema operativo y el sistema de archivos (file system). Por ejemplo, validemos que nos encontramos en un directorio donde vamos a poder hacer nuestras pruebas:

In [4]:

```
# Validamos que el directorio de trabajo sea el correcto:
print("Directorio de trabajo:", os.getcwd())
```

```
Directorio de trabajo: C:\Users\lmuno\Desktop\Programacion de Computadoras 2017-1\Semana 07
```

Vamos a crear un sub-directorio para probar las funciones de modulo os:

In [19]:

```
# Se crea un directorio dentro del directorio actual si es que este no existe
if not (os.path.exists('TextFiles')):
    print("Creando el directorio 'TextFiles'\n")
    os.mkdir('TextFiles')
else:
    print("El directorio 'TextFiles' ya existe\n")

# Se muestra el contenido de el directorio padre
print("+--", os.getcwd(), ':')
for f in os.listdir():
    print(" '--", f)

# Se ingresa al directorio creado 'TextFiles'
os.chdir('TextFiles')
print("\nDirectorio de trabajo:", os.getcwd())
```

```
El directrio 'TextFiles' ya existe

+-- C:\Users\lmuno\Desktop\Programacion de Computadoras 2017-1\Semana 07 :
 '-- .ipynb_checkpoints
 '-- 11 - Conjuntos.ipynb
 '-- 12 - Diccionarios.ipynb
 '-- 13 - Escritura y lectura de archivos.ipynb
 '-- codeSemana7.py
 '-- Semana 7 - Banco de preguntas (tuplas, diccionarios, archivos).txt
 '-- TextFiles

Directorio de trabajo: C:\Users\lmuno\Desktop\Programacion de Computadoras 2017-1\Semana 07\TextFiles
```

Asi como se puede crear un directorio, tambien se pueden eliminar:

In [22]:

```
# Se crea un directorio 'NuevoDir' dentro de 'TextFiles'
if not (os.path.exists('NuevoDir')):
    os.mkdir('NuevoDir')

# Verificar que se ha creado este directorio
print("Directorios dentro de 'TextFiles':", os.listdir())

# Se elimina el directorio recientemente creado
os.rmdir('NuevoDir')

# Verificar que se ha eliminado el directorio
print("Directorios dentro de 'TextFiles':", os.listdir())
```

```
Directorios dentro de 'TextFiles': ['NuevoDir']
Directorios dentro de 'TextFiles': []
```

¡Hay que ser muy ordenado para manipular los archvivos! Por ejemplo, si se crea un directorio hay que salir de este para eliminarlo. Una forma de moverse hacia atras en un directorio es utilizar ".." para retroceder un directorio hijo a un directorio padre:

```
# Si el sistema esta en la ruta C:\Dir1\Dir2\Dir3....
chdir('..')     # Esto hara que la ruta sea C:\Dir1\Dir2
chdir('..')     # Esto hara que la ruta sea C:\Dir1
```

Manipulación de archivos de texto

Acceder a un archivo de texto se utiliza la instrucción:

```
open('archivo.txt')
```

Esta instrucción abrirá un arhivo de texto de nombre 'archivo.txt', el modo *lectura* e interpretará los caraceteres del archivo segun el formato 'utf-8'.

Si el archivo no existe, se generá un error (excepción FileNotFoundError).

In [74]:
```
print("Accediendo al archivo 'texto.txt':")
try:
    open('texto.txt')
except FileNotFoundError:
    print("El archivo 'texto.txt' no existe")
```

```
Accediendo al archivo 'texto.txt':
```

Si el archivo existe y ha sido abierto, este se queda "secuestado" por el sistema. Esta es la forma normal de operación para mantener la integridad del sistema operativo (solo un elemento puede afectar un archivo). Es necesario cerrar un archivo abierto:

```
close('texto.txt')
```

In [28]:
```
try:
    close('texto.txt')
except NameError:
    print("El archivo 'texto.txt' no se encuentra abierto")
```

```
El archivo 'texto.txt' no se encuentra abierto
```

Esto puede representar un problema, ya que es posible que se produzca un error en el programa y este se cierre sin que se halla cerrado un archivo abierto. Es por esta razón que la forma correcta de abrir un archivo es utilizando laa instrucción **with**. Esta instrucción permite abrir un archivo y cuando el programa se cierra, también se cerrará el archivo. Esta vez vamos a abrir un archivo en modo *escritura* de forma tal que si el arvhivo no existe este se creeara.

Esto requiere especificar el campo mode='w' (por defecto, mode='r'). También existen los modos 'a' para agregar datos a los ya existentes ('w' crea un arhivo nuevo y si existe lo reescribe). Hay varios modos adicionales que se pueden consultar en la ayuda de Python.

Por otro lado, por defecto, la información de codificación es enconding='utf-8'. Sin embargo, los caracteres 'á','é','í','ó','ú','ñ' y sus formas mayúsculas no son soportadas bajo este esquema de codificación. Para nuestro idioma debemos de utilizar encoding='latin-1'

Una vez que el archivo se encuentre abierto, podemos escribir información en el archivo de texto con la instrucción archivo.write(). **Aca hay que tener cuidado, ya que funciona diferente a print(): es necesario incluír \n si se quiere pasar a la linea inferior**

In [4]:
```
with open ('textos.txt', mode='w', encoding='latin-1') as file:
    for num in range(11):
        # Se guarda una secuencia de numeros como strings uno debajo del otro
        file.write(str(num) +'\n')
```

Podemos abrir el archivo utilizando **with** para evitar la complicación de verificar si esta abierto o cerrado y verificar su contenido (no será necesario especificar el modo como *lecutura* porque es el modo por defecto). La lectura se hara con la instrucción *archivo.read()*

In [3]:

```
with open('texto.txt', encoding='latin-1') as file:
    print(file.read())
```

```
0
1
2
3
4
5
6
7
8
9
10
```

Volvamos a abrir el archivo en mode='w' para sobreescribirlo pero sin utilizar los \n:

In [54]:

```
with open ('texto.txt', mode='w', encoding='latin-1') as file:
    for num in range(11):
        # Se guarda una secuencia de numeros como strings uno debajo del otro
        file.write(str(num) +', ')

with open('texto.txt', encoding='latin-1') as file:
    print(file.read())
```

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

Ahora vamos a abrir el archivo en mode='a' para agregar información en un archivo ya existente:

In [51]:

```
with open ('texto.txt', mode='a', encoding='latin-1') as file:
    for num in range(11, 21):
        # Se guarda una secuencia de numeros como strings uno debajo del otro
        file.write(str(num) +', ')

with open('texto.txt', encoding='latin-1') as file:
    print(file.read())
```

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
```

Hay tres formas de leer los datos de un arhivo:

```
arvhivo.read()       # Lee todo el contenido de un arvhivo de texto.
avhivo.readline()    # Lee cada linea de un archivo y luego apunta a la linea inferior. Incluy
e \n
archivo.readlines()  # Retorna todas las lineas en una lista que incluirá el caracter \n al fi
nal
```

In [5]:

```
with open ('texto.txt', mode='w', encoding='latin-1') as file:
    for num in range(11):
        # Se guarda una secuencia de numeros como strings uno debajo del otro
        file.write(str(num) +'\n')

# Lectura con read()
print("Lectura con .read()")
with open('texto.txt', encoding='latin-1') as file:
    print(file.read())

# Lectura con readline()
print("Lectura con .readline()")
with open('texto.txt', encoding='latin-1') as file:
    while True:
        line = file.readline()
        if line:
            print(file.readline())
        else:
            break

# Lectura con readlines()
print("Lectura con .readlines()")
with open('texto.txt', encoding='latin-1') as file:
    print(file.readlines())
```

```
Lectura con .read()
0
1
2
3
4
5
6
7
8
9
10

Lectura con .readline()
1

3

5

7

9


Lectura con .readlines()
['0\n', '1\n', '2\n', '3\n', '4\n', '5\n', '6\n', '7\n', '8\n', '9\n', '10\n']
```

Otra forma de leer los datos es utilizando el archivo como un iterable:

In [73]:

```
with open ('texto.txt', mode='w', encoding='latin-1') as file:
    for num in range(11):
        # Se guarda una secuencia de numeros como strings uno debajo del otro
        file.write(str(num) +'\n')

# Leyendo el contenido de un arhivo utilizado file como iterable
with open('texto.txt', encoding='latin-1') as file:
    for line in file:
        print(line, end='')     # Cada linea ya incluye \n
```

```
0
1
2
3
4
5
6
7
8
9
10
```

Manipulacion de arvhvo csv

Un archivo .csv es un archivo con valores separados por comas ("comma value separated"). Este tipo de archivos es tipico en el intercambio de datos cuya fuente de origen son hojas de calculo (Excel, Google Spreadsheats, etc.) o equipos de medición (GPS, tacometros,etc.)

En estos archivos, los valores estan separados por ',' y esto hace más sencillo poder obtener no las lineas de un archivo de texto, sino los valores almacenados. Creemos un arvhivo csv:

In [80]:

```
with open ('datos.csv', mode='w', encoding='latin-1') as file:
    for num in range(11):
        # Se guarda un listado de numeros, sus cuadrados y cubos
        line = "{},{},{}\n".format(num, num**2, num**3)
        file.write(line)

# Leyendo el contenido de un arhivo csv
with open('datos.csv', encoding='latin-1') as file:
    for line in file:
        print(line, end='')     # Cada linea ya incluye \n
```

```
0,0,0
1,1,1
2,4,8
3,9,27
4,16,64
5,25,125
6,36,216
7,49,343
8,64,512
9,81,729
10,100,1000
```

Como es un arvhivo en formato csv, utilizaremos una función especial para leer los campos de este archivo. Esta función es parte del modulo csv que será necesario importarlo. La función *reader = csv.reader(archivo_csv)* retornará un *puntero* que apunta a la primera linea. *reader* es un iterable por lo que se podrá utilizar dentro de un lazo.

Se puede utilizar una lista para trasladar la información del archivo a un estructura que podrá ser consultada mas adelante:

In [81]:

```
import csv

data = []
with open('datos.csv') as csv_file:
    reader = csv.reader(csv_file)

    for line in reader:
        data.append([line[0], line[1], line[2]])

for items in data:
    print(items)
```

```
['0', '0', '0']
['1', '1', '1']
['2', '4', '8']
['3', '9', '27']
['4', '16', '64']
['5', '25', '125']
['6', '36', '216']
['7', '49', '343']
['8', '64', '512']
['9', '81', '729']
['10', '100', '1000']
```

Se puede observar que *csv.reader(file)* retorna una linea con los datos separados y a los que se puede acceder de forma independiente utilzando indices. Es comun que los archivos csv contengan un encabezado de datos:

In [85]:

```
with open ('datos.csv', mode='w', encoding='latin-1') as file:
    file.write("{},{},{}\n".format('NUM','CUAD','CUBO'))
    for num in range(11):
        # Se guarda un listado de numeros, sus cuadrados y cubos
        line = "{},{},{}\n".format(num, num**2, num**3)
        file.write(line)

# Leyendo el contenido de un arhivo csv
with open('datos.csv', encoding='latin-1') as file:
    for line in file:
        print(line, end='')     # Cada linea ya incluye \n
```

```
NUM,CUAD,CUBO
0,0,0
1,1,1
2,4,8
3,9,27
4,16,64
5,25,125
6,36,216
7,49,343
8,64,512
9,81,729
10,100,1000
```

Para evitar el encabezado al momento de leer el archivo (y en general, de alguna linea), se utiliza la funcion *next(reader)* para saltar una linea. Vamos a repetir la lectura del archivo csv evitando el encabezado e imprimiendo los resultados:

In [86]:

```
import csv

data = []
with open('datos.csv') as csv_file:
    reader = csv.reader(csv_file)
    # Se pasa a la siguiente linea para evitar el encabezado
    next(reader)

    for line in reader:
        print("Numero: {:3} Cuadrado: {:4} Cubo: {:6}".format(line[0], line[1], line[2]))
```

```
Numero: 0   Cuadrado: 0    Cubo: 0
Numero: 1   Cuadrado: 1    Cubo: 1
Numero: 2   Cuadrado: 4    Cubo: 8
Numero: 3   Cuadrado: 9    Cubo: 27
Numero: 4   Cuadrado: 16   Cubo: 64
Numero: 5   Cuadrado: 25   Cubo: 125
Numero: 6   Cuadrado: 36   Cubo: 216
Numero: 7   Cuadrado: 49   Cubo: 343
Numero: 8   Cuadrado: 64   Cubo: 512
Numero: 9   Cuadrado: 81   Cubo: 729
Numero: 10  Cuadrado: 100  Cubo: 1000
```

El acceso a un archivo binario requiere la manipulación de bytes de información, lo que escapa el alcance de este curso.