



Universidad Simón Bolívar  
Departamento de Computación y Tecnología de la información  
CI-2691- Laboratorio de algoritmos I

## Laboratorio 7

El objetivo de este laboratorio es conocer el manejo de entrada y salida de datos usando archivos.

Contenido: Lectura y escritura en archivos. Operaciones con archivos.

### Lectura y escritura en archivos

En Python se pueden manejar archivos de texto, es decir, su contenido se interpreta como caracteres por lo que se leen y escriben strings desde y hacia el archivo. Las principales operaciones son:

- **OPEN**

*open()* devuelve un objeto tipo archivo, y es comúnmente usado con dos argumentos, con la siguiente sintaxis:

```
open(nombre_de_archivo, modo)
```

El primer argumento `nombre_de_archivo` es un string que contiene el nombre/dirección del archivo. El segundo argumento es otro string que contiene un par de caracteres que describen la forma ó `modo` en que se utilizará el archivo. El `modo` puede ser `'r'` cuando sólo se necesita leer el archivo, `'w'` para sólo escritura (si existe un archivo con el mismo nombre será borrado o reescrito), y `'a'` que abre el fichero para añadir nuevos caracteres; por lo que todos los datos escritos en el archivo se añaden automáticamente al final. También, está el modo `'r+'` que abre el archivo para lectura y escritura. El argumento de modo es opcional por lo que se supondrá `'r'` si se omite. Por ejemplo,

```
f = open('workfile1', 'w')
```

abre el archivo para escritura.

Al hacer la apertura de un archivo se genera un objeto de tipo archivo (`f`) que permite realizar ciertas operaciones las cuales se explicarán más adelante.

En el modo de texto, el valor por defecto en la lectura para fin de línea depende de la plataforma específica (`\n` en Unix, `\r\n` en Windows) convirtiéndolas simplemente en `\n`. En

la escritura, el valor por defecto para fin de línea se convierte en el caracter específico según la plataforma.

- **REMOVE**

En caso de querer eliminar un archivo, puede utilizar la función `os.remove(string)`, siendo `string` la dirección del archivo. Previo al uso de estas funciones, debe importar la librería `OS`, usando la instrucción: `import os`.

## Operaciones con Archivos

Como se vio en la sección anterior, al abrir un archivo se crea un objeto de tipo archivo al cual denominamos `f`. Si se quiere leer el contenido de un archivo, se utiliza la función `f.read(tamaño)`, la cual lee la cantidad de datos indicado en `tamaño`. El argumento `tamaño` es opcional, y en caso de omitirse, la función devolverá el archivo completo como un string. Si la función llega a fin del archivo, devuelve una cadena vacía.

También se puede utilizar la función `f.readline()` para leer una línea del archivo. Si el `readline` devuelve una cadena vacía significa que se ha llegado al fin de archivo. Se puede leer líneas de un archivo de forma iterativa usando un `for`, de una manera sencilla como se muestra a continuación.

```
for line in f:
    print(line, end='')
```

Si se desean leer todas las líneas de un archivo y colocarlas en una lista, puede utilizar `list(f)` o `f.readlines()`. Por ejemplo, imaginemos que se tiene un archivo de texto, con el nombre `"input.txt"`. El archivo contiene el siguiente texto:

```
Linea 1
Linea 2
```

Si se desea leerlo con la operación `f.readlines()`, se puede hacer la siguiente secuencia de instrucciones:

```
f=open("input.txt")
lineas=f.readlines()
```

El resultado es una lista formada por las líneas del archivo:

```
['Linea 1\n', 'Linea 2\n']
```

La variable “lineas” será una lista con los dos strings: “Linea 1\n”, y “Linea 2\n”.

Para escribir contenido en un archivo, se puede utilizar la operación `f.write(frase)`, siendo `frase` la cadena a escribir. La operación devuelve la cantidad de caracteres escritos. Por ejemplo:

```
>>> f.write('Hello World')
11
```

Para escribir en el archivo, algo diferente a un string, primero debe convertirse a string. Veamos un ejemplo:

```
>>> valor = [2, 3, 4]
>>> s = str(valor)
>>> f.write(s)
9
```

Luego de haber leído el archivo, es necesario llamar a la operación `f.close()`. Esto cerrará y liberará los recursos utilizados para mantener el archivo abierto. También se tiene la función `f.closed` que dice si el objeto archivo `f` está cerrado.

Una buena práctica al trabajar con archivos, es utilizar un bloque de instrucciones que comienza con la palabra clave `with`. Dicho bloque termina con la operación `f.closed`. Esto garantiza que el archivo sea cerrado correctamente a pesar de haber ocurrido algún error en el camino. A continuación se coloca un ejemplo:

```
>>> with open('nombreDeArchivo', 'r') as f:
...     read_data = f.read()
>>> f.closed
```

En este caso, en el bloque dentro del `with`, se mantendrá el archivo abierto. Al salir del bloque, existe la garantía de que estará cerrado. En las 3 líneas de código del ejemplo anterior, se abrió el archivo, y se leyeron todos los datos con la operación `f.read()`. Al final, la operación `f.closed` confirma que está cerrado. En la variable `read_data` se almacenan todos los datos leídos del archivo, pues es la acción por defecto, cuando no se especifica el tamaño a leer.

Como ilustración de todos los comandos de entrada y salida anteriores, ejecute en python el ejemplo 1 del prelaboratorio. Para esto descargue de su espacio de aula virtual los archivos **Prelab7Ejemplo1.py**, y los dos archivos **workfile1.txt** y **workfile2.txt** en un mismo directorio: se encuentran comprimidos en el archivo **Prelab7Ejemplo.rar**.

## Ejercicio:

Un programador ha sido contratado por un laboratorio de genética para trabajar en el área de codificación y decodificación de ADN (Ácido Desoxirribonucleico). Dado que el equipo de trabajo en el laboratorio es bastante grande, al programador solo le corresponde hacer una pequeña parte del proyecto. Ésta debe ser realizada en Python, y naturalmente, cumpliendo con todas las buenas prácticas aprendidas hasta ahora, tales como: buen estilo de programación, precondiciones y postcondiciones, invariantes y cotas, análisis descendente del curso CI-2691 Laboratorio de Algoritmos I.

Como la cantidad de datos guardados por el laboratorio es tan grande, los datos han sido factorizados para ahorrar espacio. La factorización simplemente consiste en colocar un número entero y un string por línea. El número entero indica la cantidad de veces que el string debe copiarse para representar la cadena real de ADN. Por ejemplo, una línea que sea “3 ACG”, representa a “ACGACGACG”. La cadena solo debe contener los caracteres válidos de las bases nitrogenadas (A, T, C y G).

El programa a desarrollar debe leer un archivo con el formato explicado anteriormente de una cadena ADN factorizada, cuyo nombre del archivo es “**input.txt**”. El programa debe escribir en un archivo, el resultado del ADN no-factorizado. El archivo a crear tendrá el nombre “**output.txt**”.

Como a veces ocurre cuando se almacena información en medios no confiables, los datos son susceptibles a errores. Para simplificar el problema, se puede asumir como precondición que los números serán no negativos y las cadenas factorizadas de ADN serán no mayor a 50 caracteres. Todo lo leído que no cumpla con estas características puede ser considerado como un error. En el caso de leer una línea y no posea error, simplemente debe imprimir el ADN no-factorizado. Además, el programa debe ser capaz de identificar las líneas con los datos erróneos, e imprimir en el resultado el mensaje “Línea errónea”.

Por ejemplo, si el archivo de entrada contiene:

```
1 A
5 ATCA
0 CCC
-1 A
10 SA
51 ATCG
2 T
```

La salida correspondiente a dicha entrada es:

```
A
ATCAATCAATCAATCAATCA
```

Linea erronea  
Linea erronea  
Linea erronea  
TT

Sugerencia: Consulte la referencia [2] sobre string, especialmente la funciones *split* y *join*.

### Condiciones de la entrega

Cree un archivo comprimido “tgz” llamado **PreLab7-X.tgz**, donde X es su número de carné y súbalo al aula virtual de cada integrante del equipo antes del martes 01 de noviembre del 2016 a las 8:00 am. El archivo PreLab7-X.tgz deberá contener los archivos **PreLab7Ejercicio1.py**, **input.txt** y **output.txt**.

### Referencias

[1] Input and Output. Tutorial de Python. Disponible en la Web.  
<https://docs.python.org/3.3/tutorial/inputoutput.html>

[2] Strings, commons strings operations. Disponible en la Web.  
<https://docs.python.org/3.3/library/string.html>