PROGRAMACIÓN II - UNIDAD 4

Ing. Gastón Weingand (gaston.weingand@uai.edu.ar)

Agenda

1. Entrada y salida estándar.
2. Archivos: Lectura y escritura.
3. Cierre y tratamiento de errores en la gestión de archivos.
4. Expresiones regulares, patrones. Módulo re.

Agenda

1. Entrada y salida estándar.
2. Archivos: Lectura y escritura.
3. Cierre y tratamiento de errores en la gestión de archivos.
4. Expresiones regulares, patrones. Módulo re.

1- Entrada estándar

Además del uso de input el programador Python cuenta con otros métodos para obtener datos del usuario. Uno de ellos es el uso de parámetros a la hora de llamar al programa en la línea de comandos. Por ejemplo:

```
python editor.py hola.txt
```

En este caso hola.txt sería el parámetro, al que se puede acceder a través de la lista sys.argv, aunque, como es de suponer, antes de poder utilizar dicha variable debemos importar el módulo sys. sys.argv[0] contiene siempre el nombre del programa tal como lo ha ejecutado el usuario, sys.argv[1], si existe, sería el primer parámetro; sys.argv[2] el segundo, y así sucesivamente.

1- Entrada estándar

```
import sys

if(len(sys.argv) > 1):
    print ("Abriendo " + sys.argv[1])
else:
    print ("Debes indicar el nombre del archivo")
```

1- Salida estándar

La sentencia print, o más bien las cadenas que imprime, permiten también utilizar técnicas avanzadas de formateo, de forma similar al sprintf de C. Veamos un ejemplo bastante simple:

```
print ("Hola %s" % "mundo")
print ("%s %s" % ("Hola", "mundo"))
```

1- Salida estándar

Los especificadores más sencillos están formados por el símbolo % seguido de una letra que indica el tipo con el que formatear el valor, Por ejemplo:

Especificador	Formato
%s	Cadena
%d	Entero
%o	Octal
%x	Hexadecimal
%f	Real

1- Salida estándar

En el caso de los reales es posible indicar la precisión a utilizar precediendo la f de un punto seguido del número de decimales que queremos mostrar:

```
from math import pi
print ("%.4f" % pi)
```

La misma sintaxis se puede utilizar para indicar el número de caracteres de la cadena que queremos mostrar

```
print ("%.4s" % "hola mundo")
```

Agenda

1. Entrada y salida estándar.
2. Archivos: Lectura y escritura.
3. Cierre y tratamiento de errores en la gestión de archivos.
4. Expresiones regulares, patrones. Módulo re.

2- Archivos

Los archivos en Python son objetos de tipo file creados mediante la función open (Abrir). Esta función toma como parámetros una cadena con la ruta al fichero a abrir, que puede ser relativa o absoluta; una cadena opcional indicando el modo de acceso (Si no se especifica se accede en modo lectura) y, por último, un entero opcional para especificar un tamaño de buffer distinto del utilizado por defecto

2- Archivos

El modo de acceso puede ser cualquier combinación lógica de los siguientes modos:

- ‘r’: read, lectura. Abre el archivo en modo lectura. El archivo tiene que existir previamente, en caso contrario se lanzará una excepción de tipo IOError.
- ‘w’: write, escritura. Abre el archivo en modo escritura. Si el archivo no existe se crea. Si existe, sobreescribe el contenido.
- ‘a’: append, añadir. Abre el archivo en modo escritura. Se diferencia del modo ‘w’ en que en este caso no se sobreescribe el contenido del archivo, sino que se comienza a escribir al final del archivo.

2- Archivos

El modo de acceso puede ser cualquier combinación lógica de los siguientes modos:

- ‘b’: binary, binario.
- ‘+’: permite lectura y escritura simultáneas.
- ‘U’: universal newline, saltos de línea universales. Permite trabajar con archivos que tengan un formato para los saltos de línea que no coincide con el de la plataforma actual (en Windows se utiliza el caracter CR LF, en Unix LF y en Mac OS CR).

2- Archivos

Para la lectura de archivos se utilizan los métodos read, readline y realines.

```
f = open("archivo.txt", "r") #Abro el archivo para la lectura

completo = f.read() #Leo el texto completo
print(completo)
f.seek(0) #Vuelvo a posicionarme en el primer lugar del archivo...
completo = f.readlines()
print(completo)

print(f.tell()) #Indica la posición del puntero en el archivo
f.seek(0)

#parte = f2.read(512) #Si quisiera leer n cantidad de caracteres

while True:
    linea = f.readline() #Línea a línea
    if not linea: break
    print (linea)
```

2- Archivos

Para la escritura de archivos se utilizan los método write y writelines. Mientras el primero funciona escribiendo en el archivo una cadena de texto que toma como parámetro, el segundo toma como parámetro una lista de cadenas de texto indicando las líneas que queremos escribir en el fichero

```
f = open("archivo.txt", "a") #Abro el archivo para agregar información
f.write("\nquinta linea")
f.close()
```

Agenda

1. Entrada y salida estándar.
2. Archivos: Lectura y escritura.
3. Cierre y tratamiento de errores en la gestión de archivos.
4. Expresiones regulares, patrones. Módulo re.

3- Tratamiento de errores en archivos

Al igual que en el tratamiento de errores con try y except de cualquier línea de código, los archivos pueden ser gestionados de igual manera. Veamos:

```
archivo = "archivo2.txt"
while True: #Creamos un bucle infinito
    try: #Intentamos abrirlo
        with open (archivo, "r+") as f: #Abrimos utilizando Reescribir (r+) #with evita el close [using de C#]
            contenido = f.read()  #<-Lo abrimos utilizando el método read
            print (contenido)
            break
    except:
        print("Error al intentar abrir")
        print ("No se encuentra el archivo",archivo, "Especifique su nombre correctamente:")
        archivo = (input("Nombre de archivo:"))
```