ARCHIVOS

Algoritmos y estructuras de datos I

ARCHIVOS

Guardar datos a un disco o recuperar los datos previamente guardados son procesos fundamentales en cualquier programa informático.

La importancia del almacenamiento de contenidos es obvia: envío de información a otros usuarios, posponer el trabajo varios días o semanas sin tener que introducir manualmente los datos de nuevo, acceso a información almacenada en sistemas remotos, etc. Incluso para desarrollos de software de relativamente corta longitud resulta relevante la gestión de datos, por ahorrar una cantidad de tiempo considerable.

Podemos pensar en los archivos de forma análoga a lo que sería un archivo físico: un lugar donde hay información almacenada.

TIPOS DE ACCESO

En el manejo de archivos en programación, es fundamental comprender las diferentes formas de acceso que existen para interactuar con la información almacenada.

El acceso a un archivo puede realizarse de distintas maneras, cada una con sus propias características y aplicaciones específicas. En este contexto, se destacan tres métodos principales: acceso secuencial, acceso aleatorio y acceso binario.

Cada uno de estos métodos ofrece ventajas y limitaciones particulares, lo que los hace adecuados para diferentes situaciones y requisitos de programación.

Secuencial

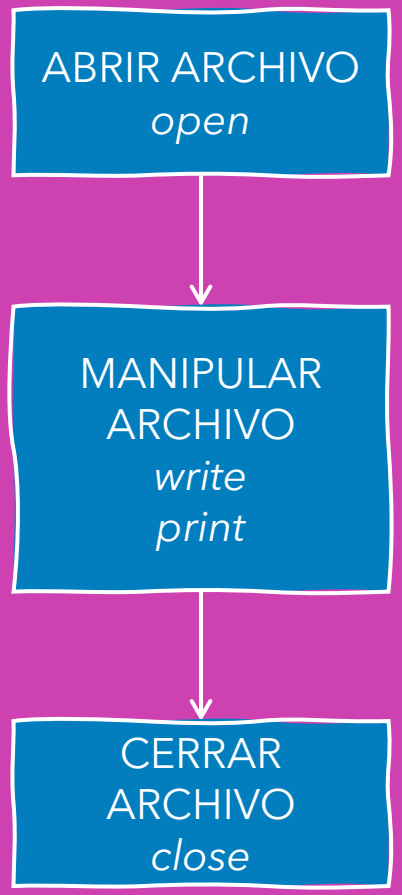
Se leen y escriben los datos de modo que, si se quiere acceder a un dato que está hacia la mitad de un archivo, habrá que pasar primero por todos los datos anteriores. Los archivos de texto son de acceso secuencial.

Aleatorio

Permiten acceder directamente a un dato sin tener que pasar por todos los demás, y pueden acceder a la información en cualquier orden. Tienen la limitación de que los datos están almacenados en bloques que se llaman registros, y que todos los registros que se almacenan en un fichero deben ser del mismo tamaño.

Binario

Son como los de acceso aleatorio, pero el acceso no se hace por registros sino por bytes.



ABRIR Y CERRAR ARCHIVOS SECUENCIALES

Vamos a estudiar únicamente ciertas formas de extraer y guardar datos de un archivo secuencial, a través de algunas funciones disponibles en el lenguaje para este fin. Esto será suficiente para poder crear programas que leen y guardan datos desde archivos de forma simple.

Para manipular información de un archivo, por ejemplo, guardar datos en él o leer datos desde él, lo primero que hemos de hacer es abrir el archivo.

A su vez, cada vez que accedemos a un archivo hemos de indicar bajo qué modalidad vamos a utilizar ese fichero: si es para leer, escribir o añadir los datos.

CREAR Y LEER UN ARCHIVO

La función open(), nos permite abrir un archivo existente. Sin embargo, de no existir, lo creará inmediatamente. Cabe resaltar que, al abrir un archivo, lo estamos preparando para su lectura, de modo que podremos acceder a su contenido (si lo tiene) e incluso, una vez abierto, podríamos escribir en él. Su uso es el siguiente:

contenido = open (*ubicación del archivo, modo*)

La variable *contenido* tendrá todas las líneas que compongan al archivo y las podremos recorrer con un ciclo for de forma individual. La función open() recibe el nombre del archivo como primer parámetro y el modo que se usará para abrir ese archivo.

Existen varios modos para abrir los archivos, a continuación, se exponen los más comunes:

Modo	Acción
r	Lectura únicamente.
w	Escritura únicamente, reemplazando el contenido actual del archivo o bien creándolo si es inexistente.
a	Escritura únicamente, manteniendo el contenido actual y añadiendo los datos al final del archivo.

CREAR Y LEER UN ARCHIVO

Por lo mencionado en el recuadro anterior, si se quiere pasar contenido a un archivo sin eliminar el contenido original y además agregarle contenido, se utiliza el modo a. Si solo necesitas leer su contenido y nada más, entonces se debe usar *r*, de la siguiente manera:

```
contenido = open ("archivo.txt", "r")
```

Es importante notar, que la ruta hacia el archivo se calcula desde la raíz de tu programa Python. De ese modo, el archivo se creará o se leerá a partir de allí, a menos que le indiquemos una ruta diferente del tipo absoluta (tal como podría ser: *ruta/hacia/el/archivo.txt*), teniendo así varias subcarpetas donde se ubicaría tu archivo.

Se debe tener en cuenta que las subcarpetas que se hacen referencia dentro del programa deben existir antes de intentar usar el archivo, pues Python no las va a crear como haría con el archivo.

Si necesitamos abrir un archivo que mantenga las acentuaciones y caracteres especiales, podemos agregarle adicionalmente como parámetro la codificación de este:

```
open ("archivo.txt", "r", encoding="UTF-8")
```

LEER UN ARCHIVO Y MOSTRAR SU CONTENIDO

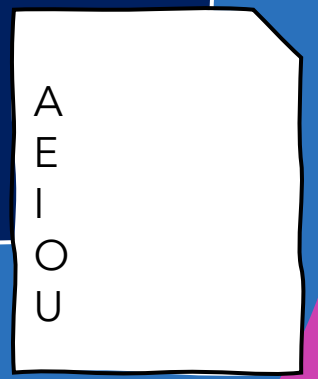
Veamos cómo usar *open()* para abrir un archivo y recorrer su contenido para mostrarlo en pantalla.

De esta manera, la variable *linea* irá almacenando distintas cadenas correspondientes a cada una de las líneas del archivo.

```
try:
    #Abrimos en modo solo lectura
    contenido = open("archivos/letras.txt", "r")

    #Recorremos y mostramos cada línea
    for linea in contenido:
        print(linea)

    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```



archivos/letras.txt

LEER UN ARCHIVO Y MOSTRAR SU CONTENIDO

Es posible, además, obtener todas las líneas del archivo utilizando una sola llamada con el método *read*, como se observa en el ejemplo.

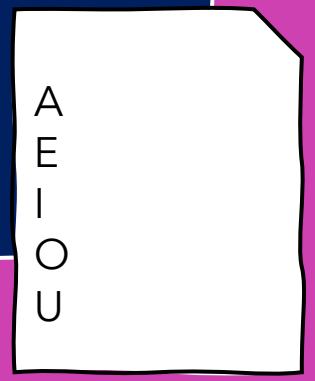
Siempre que una instrucción cargue un archivo completo en memoria debe tenerse el cuidado de utilizarla sólo con archivos pequeños, ya que podría agotarse la memoria de la computadora o causar problemas de rendimiento.

```
try:
    #Abrimos en modo solo lectura
    contenido = open("archivos/letras.txt", "r")

    lineas = contenido.read()

    print(lineas)

    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```



archivos/letras.txt

DELIMITADOR DE LÍNEAS

Cada línea o renglón dentro del archivo, constituye un registro. La longitud es variable y para reconocer el fin de cada una de las líneas se utiliza un delimitador *(\n)* que representa un salto de línea:

A	\n	E	\n	I	\n	O	\n	U	\n

En definitiva, un archivo de texto se compone de una cadena de caracteres con la particularidad de que tiene un salto de línea como delimitador.

```
try:
    #Abrimos en modo solo lectura
    contenido = open("archivos/letras.txt", "r")

    lineas = contenido.read()

    print(lineas)

    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```

```
A\n
E\n
I\n
O\n
U\n
```

archivos/letras.txt

```
try:
    contenido = open("archivos/nombres.txt", "r")

    caracteres = contenido.read()

    listaCaracteres = []
    for caracter in caracteres:
        listaCaracteres.append(caracter)

    #['O', 'l', 'i', 'v', 'e', 'r', '\n', 'G', 'a', 'b', 'r', 'i', 'e', 'l', 'a']
    print(listaCaracteres)

    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```

Oliver
Gabriela

archivos/nombres.txt

LEER UN ARCHIVO Y MOSTRAR SU CONTENIDO

A continuación, se muestra un ejemplo de lectura de archivos y el procesamiento del contenido utilizando *read*, donde podremos recorrer de carácter a carácter.

LEER UN ARCHIVO Y MOSTRAR SU CONTENIDO

Si quisiéramos obtener cada uno de los renglones (líneas) del archivo, podemos utilizar el método *split*.

Este código intenta abrir un archivo de texto *(nombres.txt)* en modo de solo lectura y leer su contenido. Se carga todo el contenido del archivo en una variable utilizando *read()*, y luego se divide en líneas usando *split("\n")*, generando una lista de nombres.

La lista de renglones se imprime en pantalla, mostrando todos los nombres leídos del archivo.

```
try:
    #Abrimos en modo solo lectura
    contenido = open("archivos/nombres.txt", "r")

    caracteres = contenido.read()

    renglones = caracteres.split("\n")

    #['Oliver', 'Gabriela']
    print(renglones)

    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```

archivos/nombres.txt

LEER UN ARCHIVO Y MOSTRAR SU CONTENIDO

```
try:
    #Abrimos en modo solo lectura
    contenido = open("archivos/frases.txt", "r")

    caracteres = contenido.read()

    renglones = caracteres.split("\n")

    print(renglones)

    for renglon in renglones:
        palabrasPorRenglon = renglon.split()
        print(palabrasPorRenglon)

    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```

"Como no sabían que era imposible, ellos lo hicieron"\n
Anónimo

archivos/frases.txt

En el siguiente ejemplo, obtenemos una lista de los renglones y, por cada uno de ellos, las distintas palabras que los contiene.

Este código intenta abrir un archivo de texto (frases.txt) en modo de solo lectura, leer su contenido y procesar cada línea.

Primero, el contenido se carga en una variable usando *read()* y luego se divide en líneas con *split("\n")*. El programa imprime la lista de líneas y, para cada línea, separa las palabras usando *split()* y las imprime individualmente.

LEER UN ARCHIVO Y MOSTRAR SU CONTENIDO

También, podríamos procesar un documento con un formato preestablecido, tal como un CSV (valores separados por coma), tal como se muestra en el ejemplo.

Este código intenta abrir un archivo CSV en modo lectura, leer su contenido y procesarlo línea por línea, donde cada línea contiene un registro con *"nombre,nota"*.

Primero, carga todo el contenido del archivo, lo divide en líneas y luego separa cada línea por comas para extraer el nombre y la nota, imprimiéndolos en pantalla.

```
try:
    #Abrimos en modo solo lectura
    contenido = open("archivos/notas.csv", "r")

    caracteres = contenido.read()
    notas = caracteres.split("\n")

    #Recorremos y mostramos cada línea
    for nota in notas:
        nombre, nota = nota.split(",")
        print(f"Nombre: {nombre} - Nota: {nota}")

    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```

Oliver,10\n
Gabriela,7\n
Irina,2\n

archivos/notas.csv

Lic. Yaps David P.

```python
nombres = ["Gabriela", "Oliver", "Irina"]

try:
    #Abrimos en modo escritura
    archivo = open("archivos/personas.txt", "w")

    for nombre in nombres:
        try:
            archivo.write(nombre + "\n")
        except Exception as e:
            print("Error al escribir en el archivo:", e)

    archivo.close()

except FileNotFoundError:
    print("No se pudo encontrar el archivo.")

except PermissionError:
    print("No tiene permiso para escribir en el
archivo.")

except Exception as e:
    print("Se produjo un error:", e)
```

ESCRIBIR EN UN ARCHIVO CON WRITE

El método *write()*, lo podemos utilizar para escribir en un archivo ya abierto. La misma recibe solo el texto que vayamos a agregar, como se muestra en el ejemplo.

Para cerrar el archivo debemos utilizar la función *close()*. Eso permitirá que el contenido sea finalmente escrito en el archivo. La función *write()* no agrega saltos de línea, así que debemos ponerlos con *"\n"* para cada línea que deseamos escribir en el archivo.

Lic. Yaps David P.

```
try:
    nombres = ["Gabriela", "Oliver", "Matías"]

    #Abrimos en modo escritura
    archivo = open("archivos/personas.txt", "w")

    try:
        for nombre in nombres:
            print(nombre, file=archivo)

    except Exception as e:
        print("Error al escribir en el archivo:", e)

    finally:
        archivo.close()

except FileNotFoundError:
    print("No se pudo encontrar el archivo.")

except PermissionError:
    print("No tiene permiso para escribir en el archivo.")

except Exception as e:
    print("Se produjo un error:", e)
```

ESCRIBIR EN UN ARCHIVO CON PRINT

Otro modo de escribir en un archivo usando la función *print()*. Podemos utilizarla para imprimir valores en un archivo con el parámetro file y sacando ventaja de su funcionamiento habitual (saltos de línea, espacios y demás).

```
try:
    archivo = open("archivos/personas.txt", "a")

    nombre = " "
    while(nombre != ""):
        nombre = input("Ingrese un nombre: ")
        if (nombre != ""):
            try:
                print(nombre, file=archivo)
            except Exception as e:
                print("Error al escribir en el archivo:", e)

    archivo.close()

except FileNotFoundError:
    print("No se pudo encontrar el archivo.")

except PermissionError:
    print("No tiene permiso para escribir en el archivo.")

except Exception as e:
    print("Se produjo un error:", e)
```

AGREGAR CONTENIDO A UN ARCHIVO

Modificando el modo de escritura (segundo parámetro con la letra a), agregaremos nuevo contenido hacia el final del archivo que estamos haciendo referencia.

Se debe tener en cuenta que, en caso de que el archivo no exista, lo crea.

Lic. Yaps David P.

```python
def registrarExcepcion(e):
    try:
        archivo = open('errores.log', 'a')
        try:
            error = f"Tipo: {type(e)} - Mensaje: {str(e)}\n"
            print(f"Ocurrió un error: {error}")
            archivo.write(error)
        finally:
            archivo.close()
    except Exception as logError:
        print(f"Error al escribir en el log: {logError}")


dato = input("Introduce un número: ")

try:
    numero = int(dato)
    print(f"El número introducido es: {numero}")
except Exception as e:
    registrarExcepcion(e)
```

PERSISTENCIA DE ERRORES EN ARCHIVOS

Este código permite persistir errores en archivos mediante la captura y registro de excepciones.

Su finalidad es proporcionar un método robusto para registrar detalles cruciales sobre cualquier error que ocurra durante la ejecución del programa.

Al emplear esta técnica, se facilita la depuración y el análisis posterior de problemas, asegurando una gestión eficiente de excepciones que contribuye a mejorar la estabilidad y el mantenimiento del software desarrollado.

Lic. Yaps David P.

AUTOEVALUACIÓN

La autoevaluación es crucial para identificar mejoras y fortalezas de forma objetiva, facilitando la fijación de metas y la toma de acciones para alcanzarlas.