



PROCESAMIENTO DE DATOS

**CURSO PYTHON
CILSA**

Procesamiento de datos

El procesamiento de datos se produce cuando se recaban datos y se traducen a información utilizable.

Datos vs Información: Los datos consisten en nada más que hechos (organizados o no organizados) que luego pueden ser manipulados en otras formas para que sean útiles y comprensibles, convirtiendo los **datos** en **información**.

Etapas de procesamiento de datos

- Recopilación de datos
- Preparación de datos
- Entrada de datos
- **Procesamiento**
- Interpretación de los datos
- Almacenamiento de datos

Continuando con la escritura

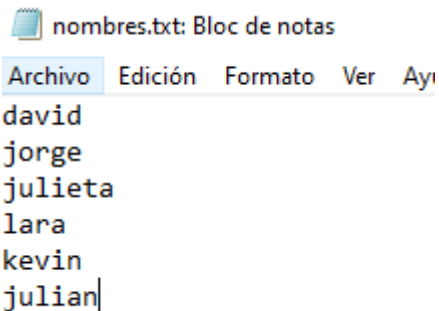
Una nueva opción para escribir en un archivo es utilizando la función PRINT.

Por ejemplo, cuando tengo el archivo destino abierto con una variable llamada **archivo**, y tengo una lista de elementos llamada **lista_nombres**, y quiero que esos datos sean guardados en el documento.

En esta situación, podría realizar lo siguiente para iterar sobre la lista y al mismo tiempo volcarlo sobre el archivo:

```
for nombre in lista_nombres:  
    print(nombre, file=archivo)
```

Para los siguientes ejemplos, utilizaremos un breve archivo de texto como el siguiente:



nombres.txt: Bloc de notas

Archivo Edición Formato Ver Ayu

david
jorge
julieta
lara
kevin
julian|

Cierre de un archivo

Vimos previamente como creábamos un archivo y como guardábamos datos dentro del mismo... utilizando open y close.

También podemos utilizar la palabra clave **with** para cerrar archivos en Python, junto con la palabra **as**. La siguiente tabla muestra el ejemplo que es equivalente a lo visto anteriormente:

<pre>with open("alumnos.txt") as archivo: print(archivo.read())</pre>	<pre>archivo = open("alumnos.txt") print(archivo.read()) archivo.close()</pre>
---	--

Estoy mostrando en pantalla cada línea del contenido de mi archivo.

¿Qué es read? A continuación, veremos algunos **métodos** sobre archivos que permiten gestionar la lectura de su contenido.

Lectura del contenido

Read: leerá todo el contenido de un archivo y lo devolverá como una cadena de texto.

```
archivo = open("alumnos.txt")
```

```
print(archivo.read()) # devuelve todo el contenido del archivo, línea por línea.
```

Readline: Este método leerá **una línea** del archivo y la devolverá.

```
with open("nombres.txt") as archivo:
```

```
    print(archivo.readline()) # devuelve: david
```

Readlines: de la misma manera que el anterior, pero en este caso devolverá una lista de todas las líneas de texto del archivo, va a incluir el salto de línea, por ejemplo: ['david\n', 'jorge\n', ...]

For: podemos utilizar el bucle FOR para iterar sobre el elemento archivo:

```
with open("nombres.txt") as archivo:
```

```
    for linea in archivo:
```

```
        print(linea)
```

Archivos csv y encabezados

Podemos abrir archivos con extensión .csv, estos archivos son muy comunes a la hora de levantar datos de múltiples orígenes.

Utilizamos la operación de lectura del archivo, como se hace con un txt.

Ahora, nos puede suceder que existan encabezados sobre esa tabla que estamos leyendo.

Para esta situación, tenemos al menos 2 alternativas.

- Guardamos los datos en nuestras variables y luego eliminamos los datos de cada primer elemento correspondiente a la primera fila de la tabla.
- Utilizar función NEXT:

```
with open("datos.csv", "r") as f:  
    next(f)  
    for linea in f:  
        print(linea)
```

Método Split string

Podemos invocar el método Split que retorna una lista de substrings del String.

Este método separa el String en un listado con pequeñas partes indicándole cual es el separador.

Es decir que, si tengo un String y el separador será el carácter vacío, entonces obtendré como resultado una lista de varios elementos:

```
txt = "hola como estan?"  
x = txt.split()  
print(x)    # ['hola', 'como', 'estan?']
```

Si tuviera un string con comas, podría separar los elementos por ese carácter utilizando `txt.split(",")`

Método find string

Hay dos opciones para encontrar un subtring dentro de una string en Python, find() y rfind().

Cada uno retorna la posición en la que se encuentre la substring. La diferencia entre los dos, es que find() retorna la posición de la primera similitud de la substring y rfind() retorna la última posición de la similitud de la substring.

Ejemplo:

```
frase= " Quien tiene un amigo tiene un tesoro."
```

```
resulFind = frase.find("tiene")
```

```
resulRfind = frase.rfind("tiene")
```

```
print(resulFind, resulRfind) # 7 y 22
```


Método string join

El método `str.join(iterable)` es usado para unir todos los elementos de un **iterable** (lista, diccionario, strings, entre otros.) con un **string** específico. Si, el iterable no contiene ningún valor en los strings, sucederá un error.

Ejemplos:

Concatena la lista de strings con ":"

```
print(":".join(["usuario1", "1234"]))      #    usuario1:1234
```