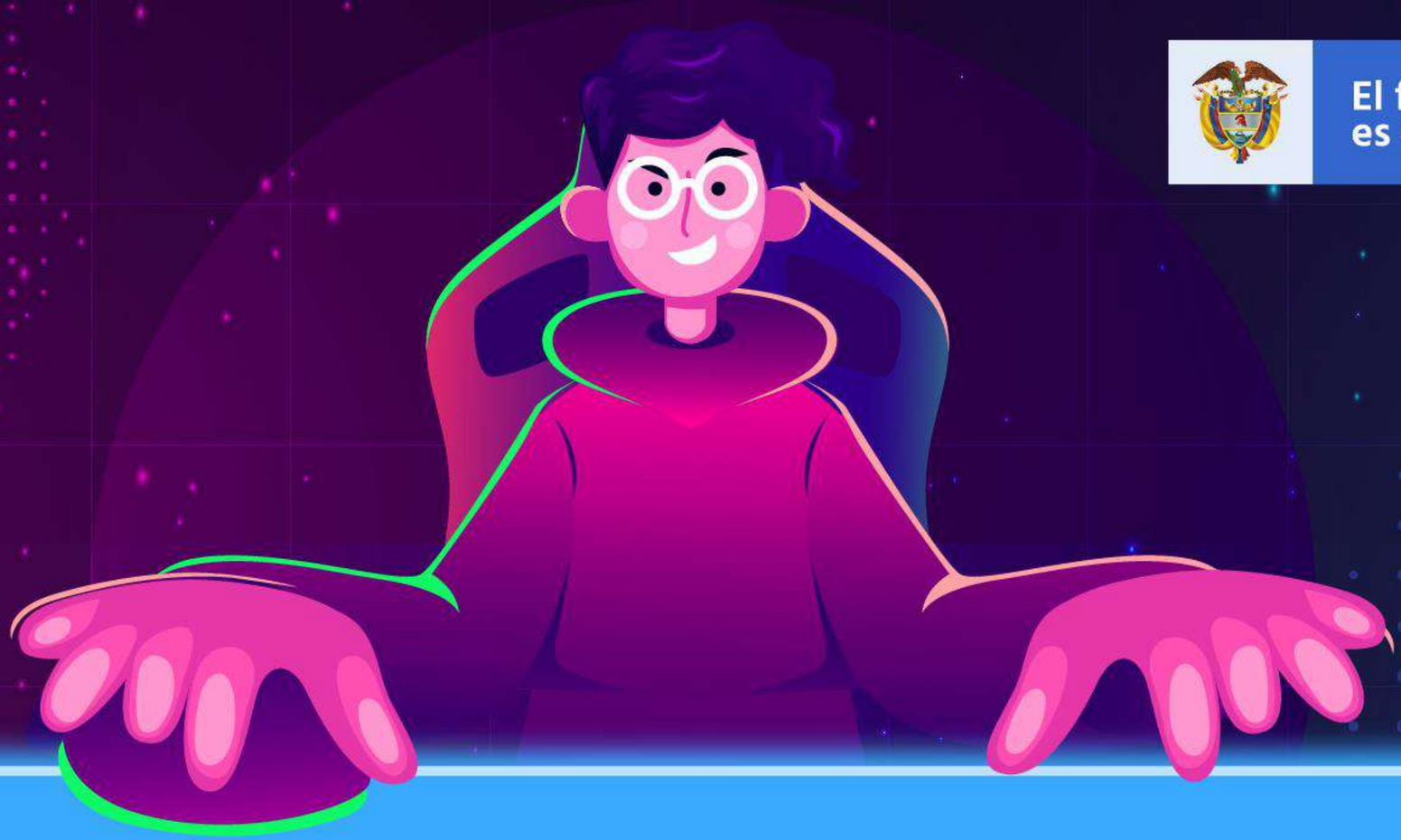

El futuro digital es de todos

MinTIC

Manejo de archivos .CSV

OPERADO POR:

Universidad Tecnológica de Pereira

Mision TIC 2022

RUTA DE APRENDIZAJE 1

El método Where:

De forma semejante a las series, el método de los dataframes where filtra los valores contenidos en el dataframe de forma que solo los que cumplan cierta condición se mantengan. El resto de valores son sustituidos por un valor que, por defecto, es NaN.
Por ejemplo, partimos del siguiente dataframe:

```
In [214]: df = pd.DataFrame(np.arange(12).reshape(-1, 3), columns=["A", "B", "C"])
          df
```

Out[214]:

	A	B	C
0	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11

Si ahora queremos filtrar los valores múltiplos de 2, por ejemplo, podemos hacerlo de la siguiente forma:

```
In [215]: df.where(df % 2 == 0)
```

Out[215]:

	A	B	C
0	0.0	NaN	2.0
1	NaN	4.0	NaN
2	6.0	NaN	8.0
3	NaN	10.0	NaN

Todos aquellos valores que no son múltiplo de 2 son sustituidos por NaN. Si, por ejemplo, quisiéramos cambiar de signo a los valores que no cumplen la condición impuesta, lo haríamos así:

```
In [216]: df.where(df % 2 == 0, -df)
```

Out[216]:

	A	B	C
0	0	-1	2
1	-3	4	-5
2	6	-7	8
3	-9	10	-11

Eliminación de elementos:

El método pandas.DataFrame.drop elimina las filas o columnas indicadas y devuelve el resultado, permitiéndose diferentes criterios para especificarlas.
El primer criterio consiste en indicar la lista de etiquetas a eliminar y el eje al que pertenecen. Partamos del siguiente dataframe:

```
In [217]: df = pd.DataFrame(np.arange(16).reshape([4, 4]),
                            index = ["a", "b", "c", "d"],
                            columns = ["A", "B", "C", "D"])
          df
```

Out[217]:

	A	B	C	D
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

Podemos eliminar, por ejemplo, las filas cuyas etiquetas son "a" y "c" con el siguiente código:

```
In [218]: df.drop(["a", "c"], axis = 0)
```

Out[218]:

	A	B	C	D
b	4	5	6	7
d	12	13	14	15

Unión de series y dataframes:

Frecuentemente nos encontramos con que los datos a analizar están repartidos entre dos o más bloques de datos, lo que nos obliga a unirlos, bien concatenándolos, o bien realizando un "join" entre las estructuras (uniones del mismo tipo que las realizadas en bases de datos). Revisemos las funciones asociadas.

Unión de series:

Comencemos revisando las opciones disponibles para las series pandas

La función concat:

Un caso con el que nos encontramos con relativa frecuencia es aquel en el que queremos unir una serie a otra. Por ejemplo:

```
In [224]: s = pd.Series([1, 2, 3, 4, 5,], index = ["a", "b", "c", "d", "e"])
          r = pd.Series([10, 11, 12], index = ["f", "g", "h"])
```

Si deseamos unir r y s en una nueva serie, podemos usar la función pandas.concat. Esta función permite especificar el eje a lo largo del cual unir los diferentes objetos (pueden ser series o dataframes). Por defecto, la concatenación se realiza a lo largo del eje 0:

```
In [225]: t = pd.concat([s, r])
          print(type(t))
          t

          <class 'pandas.core.series.Series'>

Out[225]: a     1
          b     2
          c     3
          d     4
          e     5
          f    10
          g    11
          h    12
          dtype: int64
```

Podemos ver en el ejemplo anterior que el resultado es una serie pandas.
Si especificamos como eje de concatenación el eje 1, pandas alineará los valores con idénticas etiquetas. En el siguiente ejemplo, las series a y b tienen algunas etiquetas comunes (y otras no). El resultado incluye todas las etiquetas asignando el valor NaN ("Not a Number") a aquellos valores desconocidos:

In [226]:
```
a = pd.Series([1, 2, 3, 4, 5,], index = ["a", "b", "c", "d", "e"])
b = pd.Series([10, 11, 12], index = ["a", "b", "f"])
pd.concat([a, b], axis = 1, sort = True)
```

Out[226]:

	0	1
a	1.0	10.0
b	2.0	11.0
c	3.0	NaN
d	4.0	NaN
e	5.0	NaN
f	NaN	12.0

(se ha utilizado el argumento sort = True para ocultar cierto aviso al respecto de un cambio en la funcionalidad de esta función en versiones futuras de la librería pandas)
Como puede observarse, el resultado es un dataframe:

```
In [227]: type(pd.concat([a, b], axis = 1, sort = True))

Out[227]: pandas.core.frame.DataFrame
```

Por otro lado, ya sabemos que las etiquetas del índice no tienen por qué ser diferentes, de forma que si estuviésemos concatenando series con etiquetas comunes en sus índices, el resultado sería equivalente a los vistos hasta ahora:

```
In [228]: s = pd.Series([1, 2, 3, 4], index = ["a", "b", "c", "d"])
          r = pd.Series([10, 11, 12], index = ["a", "c", "e"])
          pd.concat([s, r])

Out[228]: a     1
          b     2
          c     3
          d     4
          a    10
          c    11
          e    12
          dtype: int64
```

El método append

Como se ha comentado, el método pandas.DataFrame.append es un atajo de la función concat que ofrece funcionalidad semejante pero limitada: no permite especificar el eje de concatenación (siempre es el eje 0) ni el tipo de "join" (siempre es tipo "Outer").

Si seguimos con los mismos **dataframes** de la sección anterior:

```
In [229]: a = pd.Series([1, 2, 3, 4, 5], index = ["a", "b", "c", "d", "e"])
          b = pd.Series([10, 11, 12], index = ["f", "g", "h"])

In [230]: c = a.append(b)
          c

Out[230]: a     1
          b     2
          c     3
          d     4
          e     5
          f    10
          g    11
          h    12
          dtype: int64
```

Si el argumento ignore_index toma el valor True, se ignoran las etiquetas de las series:

```
In [231]: c = a.append(b, ignore_index = True)
          c

Out[231]: 0     1
          1     2
          2     3
          3     4
          4     5
          5    10
          6    11
          7    12
          dtype: int64
```

Es necesario mencionar que el argumento será siempre interpretado como etiqueta, aun cuando pueda estar representando un índice válido:

```
In [93]: df2 = pd.DataFrame(np.arange(12).reshape([4, 3]),
                            index = ["a", "b", "c", "e"],
                            columns = ["B", "C", "D"])
         df2
```

Out[93]:

	B	C	D
a	0	1	2
b	3	4	5
c	6	7	8
e	9	10	11

Podemos ver cuál es el resultado de aplicar este método a df1:

Manejo de archivos .CSV

Uno de los formatos más utilizados en la actualidad para intercambio de datos es CSV (“Comma Separated Values”).

Estas son básicamente archivos de texto en los que cada línea contiene una fila de datos con múltiples registros delimitados por un separador.

Tradicionalmente el separador suele ser la coma, de ahí el nombre del formato. Aunque también se pueden utilizan otros caracteres que no suelen estar contenidos en los datos. Por ejemplo, espacios, tabuladores y puntos y coma. Lo que los hace muy fáciles de procesar y son soportados por cualquier aplicación. Incluso son fáciles de leer por personas con editores de texto. Por eso es clave saber guardar y leer archivos CVS con Python.

In [4]:
```
import pandas as pd

data = {'first_name': ['Sigrid', 'Joe', 'Theodoric','Kennedy', 'Beatrix', 'Olimpia', 'Grange', 'Sallee'],
        'last_name': ['Mannock', 'Hinners', 'Rivers', 'Donnell', 'Parlett', 'Guenther', 'Douce', 'Johnstone'],
        'age': [27, 31, 36, 53, 48, 36, 40, 34],
        'amount_1': [7.17, 1.90, 1.11, 1.41, 6.69, 4.62, 1.01, 4.88],
        'amount_2': [8.06,  "?", 5.90,  "?",  "?", 7.48, 4.37,  "?"]}

datosDataFrame = pd.DataFrame(data)

print(datosDataFrame)

#Acaba la implementacion

datosDataFrame.to_csv('example.csv')
```

	first_name	last_name	age	amount_1	amount_2
0	Sigrid	Mannock	27	7.17	8.06
1	Joe	Hinners	31	1.90	?
2	Theodoric	Rivers	36	1.11	5.9
3	Kennedy	Donnell	53	1.41	?
4	Beatrix	Parlett	48	6.69	?
5	Olimpia	Guenther	36	4.62	7.48
6	Grange	Douce	40	1.01	4.37
7	Sallee	Johnstone	34	4.88	?

En caso de que se desee cambiar el delimitador se puede indicar con la propiedad sep. Para que este sea punto y coma simplemente se ha de escribir:

```
In [3]: datosDataFrame.to_csv('nombre_salida.csv', sep=';')
```

Lectura de archivos CSV:

La lectura de los archivos se realiza con el método read_csv de pandas. Solamente se la ha de indicar la ruta al archivo.

```
In [4]: df = pd.read_csv('example.csv')
        df
```

Out[4]:

	Unnamed: 0	first_name	last_name	age	amount_1	amount_2
0	0	Sigrid	Mannock	27	7.17	8.06
1	1	Joe	Hinners	31	1.90	?
2	2	Theodoric	Rivers	36	1.11	5.9
3	3	Kennedy	Donnell	53	1.41	?
4	4	Beatrix	Parlett	48	6.69	?
5	5	Olimpia	Guenther	36	4.62	7.48
6	6	Grange	Douce	40	1.01	4.37
7	7	Sallee	Johnstone	34	4.88	?

Por defecto se utiliza la primera línea del fichero como cabecera para asignar los nombres a las columnas. En el caso de que el archivo no disponga de cabecera se puede evitar esto asignando None a la propiedad head.

```
In [9]: df = pd.read_csv('nombre_salida.csv', header=None)
        df
```

Out[9]:

	0
0	;first_name;last_name;age;amount_1;amount_2
1	0;Sigrid;Mannock;27;7.17;8.06
2	1;Joe;Hinners;31;1.9;?
3	2;Theodoric;Rivers;36;1.11;5.9
4	3;Kennedy;Donnell;53;1.41;?
5	4;Beatrix;Parlett;48;6.69;?
6	5;Olimpia;Guenther;36;4.62;7.48
7	6;Grange;Douce;40;1.01;4.37
8	7;Sallee;Johnstone;34;4.88;?

El archivo que se ha utilizado en esta ocasión tiene cabecera, por lo que esta se ha cargado como la primera fila. En caso de que se desee ignorar una o más filas se le puede indicar medítate la propiedad skiprows.

In [10]:
```
df = pd.read_csv('nombre_salida.csv')
df
```

Out[10]:

	;first_name;last_name;age;amount_1;amount_2
0	0;Sigrid;Mannock;27;7.17;8.06
1	1;Joe;Hinners;31;1.9;?
2	2;Theodoric;Rivers;36;1.11;5.9
3	3;Kennedy;Donnell;53;1.41;?
4	4;Beatrix;Parlett;48;6.69;?
5	5;Olimpia;Guenther;36;4.62;7.48
6	6;Grange;Douce;40;1.01;4.37
7	7;Sallee;Johnstone;34;4.88;?

Los nombres de las columnas del dataframe se pueden indicar mediante la propiedad names.

```
In [11]: df = pd.read_csv('example.csv',
                          skiprows = 1,
                          names=['UID', 'First Name', 'Last Name', 'Age', 'Sales #1', 'Sales #2'])
         print(type(df['Sales #2'][2]))
```

```
<class 'str'>
```

Es posible que los archivos contengan valores nulos. En el ejemplo se puede ver que este es ?. La propiedad que permite que se asigne un valor nulo cuando se encuentra un valor dado es na_values=['.']

```
In [12]: df = pd.read_csv('example.csv',
                          skiprows=1,
                          names=['UID', 'First Name', 'Last Name', 'Age', 'Sales #1', 'Sales #2'],
                          na_values=['?'])
         df
```

Out[12]:

	UID	First Name	Last Name	Age	Sales #1	Sales #2
0	0	Sigrid	Mannock	27	7.17	8.06
1	1	Joe	Hinners	31	1.90	NaN
2	2	Theodoric	Rivers	36	1.11	5.90
3	3	Kennedy	Donnell	53	1.41	NaN
4	4	Beatrix	Parlett	48	6.69	NaN
5	5	Olimpia	Guenther	36	4.62	7.48
6	6	Grange	Douce	40	1.01	4.37
7	7	Sallee	Johnstone	34	4.88	NaN

Finalmente se puede asignar un índice a los datos

```
In [13]: df = pd.read_csv('example.csv',
                          skiprows=1,
                          names=['UID', 'First Name', 'Last Name', 'Age', 'Sales #1', 'Sales #2'],
                          na_values=['?'],
                          index_col='UID')
         df
```

Out[13]:

UID	First Name	Last Name	Age	Sales #1	Sales #2
0	Sigrid	Mannock	27	7.17	8.06
1	Joe	Hinners	31	1.90	NaN
2	Theodoric	Rivers	36	1.11	5.90
3	Kennedy	Donnell	53	1.41	NaN
4	Beatrix	Parlett	48	6.69	NaN
5	Olimpia	Guenther	36	4.62	7.48
6	Grange	Douce	40	1.01	4.37
7	Sallee	Johnstone	34	4.88	NaN

O más de uno

```
In [14]: df = pd.read_csv('example.csv',
                          skiprows=1,
                          names=['UID', 'First Name', 'Last Name', 'Age', 'Sales #1', 'Sales #2'],
                          na_values=['?'],
                          index_col=['First Name', 'Last Name'])
         print(df)
```

```
                       UID  Age  Sales #1  Sales #2
First Name Last Name
Sigrid     Mannock       0   27      7.17      8.06
Joe        Hinners       1   31      1.90       NaN
Theodoric  Rivers        2   36      1.11      5.90
Kennedy    Donnell       3   53      1.41       NaN
Beatrix    Parlett       4   48      6.69       NaN
Olimpia    Guenther      5   36      4.62      7.48
Grange     Douce         6   40      1.01      4.37
Sallee     Johnstone     7   34      4.88       NaN
```

Manejo de archivos Excel:

Antes de poder guardar un archivo Excel desde Python es necesario disponer de un dataframe. Por lo que se puede crear uno de ejemplo, como se hizo al hablar de los archivos CVS:

```
In [15]: import pandas as pd

data = {'first_name': ['Sigrid', 'Joe', 'Theodoric','Kennedy', 'Beatrix', 'Olimpia', 'Grange', 'Sallee'],
        'last_name': ['Mannock', 'Hinners', 'Rivers', 'Donnell', 'Parlett', 'Guenther', 'Douce', 'Johnstone'],
        'age': [27, 31, 36, 53, 48, 36, 40, 34],
        'amount_1': [7.17, 1.90, 1.11, 1.41, 6.69, 4.62, 1.01, 4.88],
        'amount_2': [8.06,  "?", 5.90,  "?",  "?", 7.48, 4.37,  "?"]}

df = pd.DataFrame(data, columns = ['first_name', 'last_name', 'age', 'amount_1', 'amount_2'])
```

Ahora para exportar los datos en formato Excel simplemente se ha utilizar el método to_excel del dataframe. En esta ocasión se ha de indicar el archivo en el que se desea guardar los datos mediante una cadena de texto. Opcionalmente se puede indicar también el nombre de la hoja del libro Excel mediante la propiedad sheet_name. El contenido del archivo generado el siguiente código se muestra en la figura.

```
In [16]: df.to_excel('example.xlsx', sheet_name='example')
```

Lectura de archivos Excel en Python:

El proceso de lectura se realiza con el método read_excel de pandas. En el caso de que el libro contenga más de una hoja se puede indicar el nombre de la que se desea importar mediante el método sheet_name. Cuando no se indique una cargara el contenido de la primera hoja del libro.

```
In [18]: df = pd.read_excel('example.xlsx', sheet_name='example')
```

Por defecto el método utiliza la primera línea del fichero como cabecera para asignar los nombres a las columnas. En el caso de que el archivo no disponga de cabecera se puede evitar este comportamiento asignando None a la propiedad head.

```
In [19]: df = pd.read_excel('example.xlsx', header=None)
```

El archivo que se ha utilizado en esta ocasión tiene cabecera, por lo que esta se ha importado como la primera fila. En caso de que se desee ignorar una o más filas se le puede indicar mediante la propiedad skiprows.

```
In [20]: df = pd.read_excel('example.xlsx', header=None, skiprows=1)
```

En el caso de que se desee indicar un nombre concreto para cada una de las columnas, diferente al de la hoja, se puede indicar mediatne la propiedad names.

```
In [21]: df = pd.read_excel('example.xlsx',
                            skiprows = 1,
                            names=['UID', 'First Name', 'Last Name', 'Age', 'Sales #1', 'Sales #2'])
```

Manejo de archivos de texto (.txt)

Abrir un archivo para leer o escribir en Python:

Antes de leer o escribir archivos con Python es necesario es necesario abrir una conexión. Lo que se puede hacer con el comando open(), al que se le ha de indicar el nombre del archivo.
La documentación se encuentra en: https://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-fileshttp://

Por defecto la conexión se abre en modo lectura, con lo que no es posible escribir en el archivo. Para poder escribir es necesario utilizar la opción "w" con la que se eliminará cualquier archivo existente y creará uno nuevo. Otra opción que se puede utilizar es "a", con la que se añadirá nuevo contenido al archivo existente. Las opciones se pueden ver en el siguiente código.

```
In [27]: # Abre el archivo para escribir y elimina los archivos anteriores si existen
         fic = open("Archivos/text.txt", "w")
```

```
In [28]: # Abre el archivo para agregar contenido
         fic = open("Archivos/text.txt", "a")
```

```
In [29]: # Abre el archivo en modo lectura
         fic = open("Archivos/text.txt", "r")
```

```
In [30]: fic.close()
```

En todos los casos, una vez finalizado las operaciones de lectura y escritura con los archivos, una buena práctica es cerrar el acceso. Para lo que se debe utilizar el método close().

Como un resumen:

La **r** indica el modo lectura. Si se intentara utilizar la función write para escribir algo, se lanzaría la excepción IOError. A continuación los distintos modos.

- **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.
- **w+**, **r+** o **a+** – Lectura y escritura.

El signo **'+'** permite realizar ambas operaciones. La diferencia entre **w+** y **r+** consiste en que la primera opción borra el contenido anterior antes de escribir nuevos datos, y crea el archivo en caso de ser inexistente. **a+** se comporta de manera similar, aunque añade los datos al final del archivo.

Todas las opciones anteriores pueden combinarse con una 'b' (de binary), que consiste en leer o escribir datos binarios. Esta opción es válida únicamente en sistemas Microsoft Windows, que hacen una distinción entre archivos de texto y binarios. En el resto de las plataformas, es simplemente ignorada. Ejemplos: rb, wb, ab+, rb+, wb+.

Escribir archivos de texto en Python:

Antes de guardar un archivo es necesario disponer de un vector con las cadenas de texto que se desean guardar. Para ello se puede crear un vector al que se le puede llamar data.

```
In [31]: data = ["Línea 1", "Línea 2", "Línea 3", "Línea 4", "Línea 5"]
```

Para escribir el contenido de este vector en un archivo se puede hacer de dos maneras: línea a línea o de una sola vez.

Escribir el archivo línea a línea:

El método más fácil directo para volcar el vector en un archivo es escribir el contenido línea a línea. Para ello se puede iterar sobre el archivo y utilizar el método write de archivo. Este proceso es lo que se muestra en el siguiente ejemplo.

```
In [34]: fic = open("text_1.txt", "w")

         for line in data:
             fic.write(line)
             fic.write("\n")

         fic.close()
```

Nótese que los elementos de vector no finalizan con el carácter salto de línea. Por lo tanto, es necesario añadir este después de escribir cada línea. Ya que, de lo contrario, todos los elementos se escribirían en una única línea en el archivo de salida.

Una forma de escribir el archivo línea a línea sin que sea necesario incluir el salto de línea es con la función print. Para esto es necesario incluir la opción "file" con la conexión al archivo. Esta opción se puede ver en el siguiente ejemplo.

```
In [35]: fic = open("text_2.txt", "w")

         for line in data:
             print(line, file=fic)

         fic.close()
```

Escribir el archivo de una vez:

Finalmente, en el caso de que los datos se encuentren en un objeto iterable se puede utilizar el método writelines para volcar este de golpe. Aunque es necesario tener en cuenta que este método no agrega el salto de línea, por lo que puede ser necesario agregarlo con antelación.

```
In [36]: fic = open("text_3.txt", "w")
         fic.writelines("%s\n" % s for s in data)
         fic.close()
```

Leer archivos de texto en Python:

La lectura de los archivos, al igual que la escritura, se puede hacer de dos maneras: línea a línea o de una sola vez.

Leer el archivo de una vez:

El procedimiento para leer los archivos de texto más sencillo es hacerlo de una vez con el método readlines. Una vez abierto el archivo solamente se ha de llamar a este método para obtener el contenido. Por ejemplo, se puede usar el siguiente código.

```
In [37]: fic = open('text_1.txt', "r")
         lines = fic.readlines()
         fic.close()

In [38]: lines

Out[38]: ['Línea 1\n', 'Línea 2\n', 'Línea 3\n', 'Línea 4\n', 'Línea 5\n']
```

En esta ocasión lines es un vector en el que cada elemento es una línea del archivo. Alternativamente, en lugar del método readlines se puede usar la función list para leer los datos.

```
In [39]: fic = open('text_1.txt', "r")
         lines = list(fic)
         fic.close()

In [40]: lines
Out[40]: ['Línea 1\n', 'Línea 2\n', 'Línea 3\n', 'Línea 4\n', 'Línea 5\n']
```

Leer el archivo línea a línea:

En otras ocasiones puede ser necesario leer el archivo línea a línea. Esto se puede hacer simplemente iterando sobre el fichero una vez abierto. En cada iteración se podrá hacer con cada línea cualquier operación que sea necesaria. En el siguiente ejemplo cada una de las líneas se agrega a un vector.

```
In [41]: fic = open('text_1.txt', "r")
         lines = []

         for line in fic:
             lines.append(line)

         fic.close()

In [42]: lines
Out[42]: ['Línea 1\n', 'Línea 2\n', 'Línea 3\n', 'Línea 4\n', 'Línea 5\n']
```

Eliminar los saltos de línea en el archivo importado:

Los tres métodos que se han visto para leer los archivos importan el salto de línea. Por lo que puede ser necesario eliminarlo antes de trabajar con los datos. Esto se puede conseguir de forma sencilla con el método rstrip de las cadenas de texto de Python. Lo que se puede hacer iterando sobre el vector.

```
In [43]: lines1 = [s.rstrip('\n') for s in lines]

In [44]: lines1
Out[44]: ['Línea 1', 'Línea 2', 'Línea 3', 'Línea 4', 'Línea 5']
```

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022