

CIENCIA DE DATOS CON PYTHON: RECOLECCIÓN, ALMACENAMIENTO Y PROCESO



Fuente: adobestock/505413960



Autor:
Amaury Giovanni Méndez Aguirre

ÍNDICE

Introducción.	3
Lectura de Ficheros (archivos).	4
Archivos de texto simples	5
Archivos CSV	10
Archivos de Excel.	14
Conclusiones	17
Introducción.	18
Lectura de Archivos y APIs.	19
JSON	20
XML	22
Recolección mediante APIs.	24
Serialización de Objetos (joblib)	30
Conclusiones	31
Procesos - pasos	32
Bibliografía.	36

INTRODUCCIÓN

Cuando se registran datos de forma digital, suele hacerse en diferentes software y formatos como hojas de cálculo o archivos de texto con separadores especiales. Es necesario algunas veces "limpiar" los datos para extraer solo aquellos que nos son útiles para un determinado problema y es aquí donde en Python, por medio de librerías especializadas, puede lograrse esto de una forma más cómoda. Su importancia radica en la sencillez y efectividad con la cual podemos recolectar datos desde diferentes fuentes y formatos.



Visitar página

Los archivos de datos para los ejemplos desarrollados en este módulo los puedes encontrar en el siguiente enlace:
<https://github.com/amaury84/aamodulo2.git>



Lectura de Ficheros (archivos)

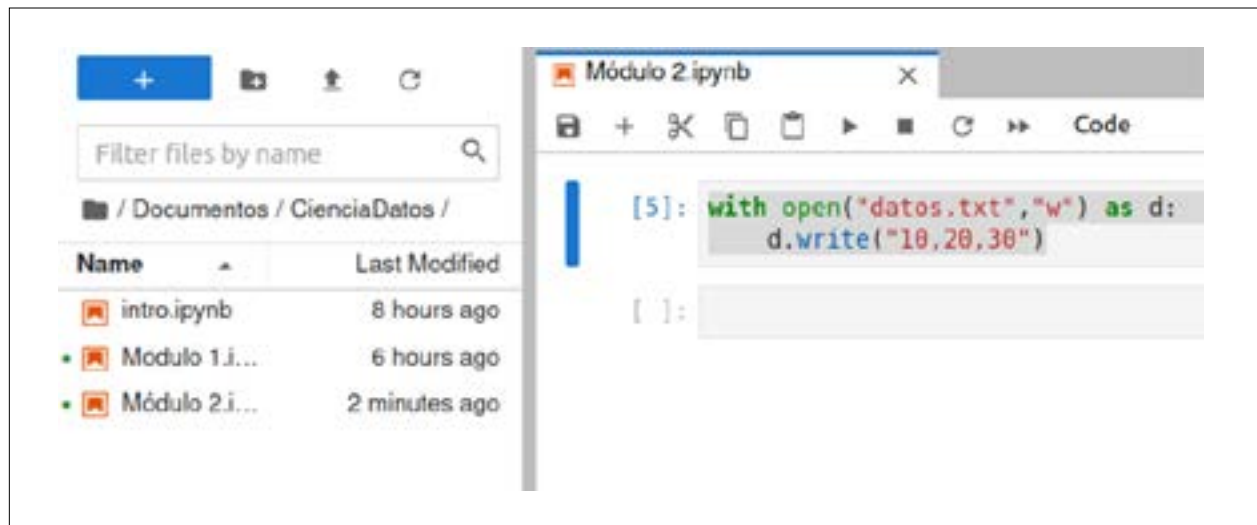


Archivos de texto simples

Para comenzar, necesitaremos tener las fuentes de datos a emplear. Para nuestro primer ejemplo, construiremos nuestra propia fuente y luego generamos la lectura apropiada de cada uno, así que empecemos generando nuestro primer archivo:

```
with open("datos.txt", "w") as d:  
    d.write("10,20,30")
```

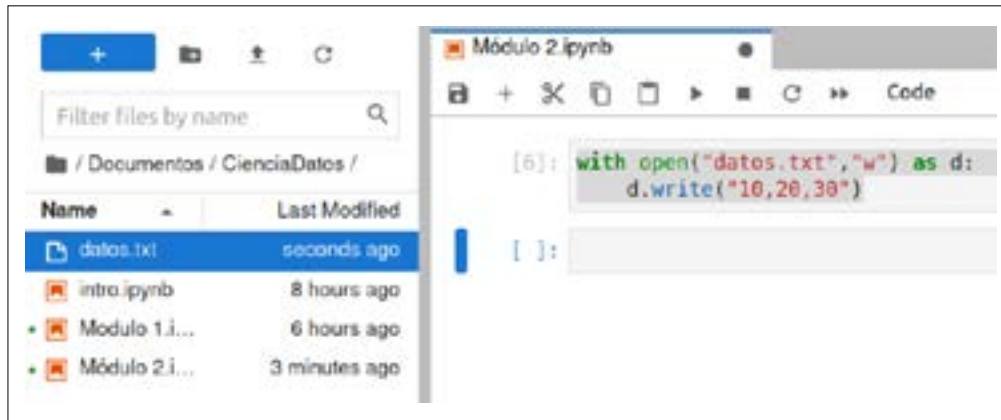
Figura 1



Fuente: propiaw

Vemos que antes de ejecutar el código, el archivo datos.txt no existe. Aquí estamos usando instrucciones puras de Python para crear y luego leer los archivos que creemos.

Figura 2



Fuente: propia

La instrucción **with** nos asegura que el proceso de crear el objeto con la clase **open**, se cerrará correctamente. Sin esta instrucción, el código sería el siguiente:

```
d = open("datos.txt", "w")
d.write("10,20,30")
d.close()
```



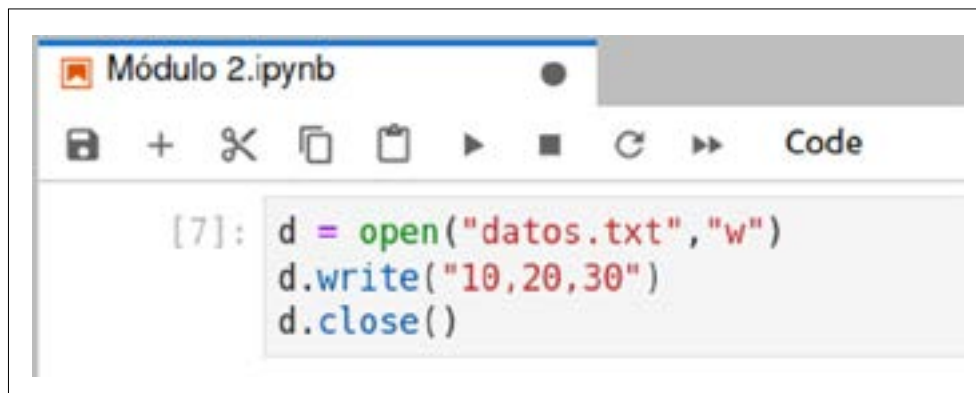
with:

Palabra reservada en Python para crear un bloque de código que se encargará de cerrar correctamente un archivo que se esté trabajando por medio de la instrucción **open**

open:

Palabra reservada en Python como clase que trabaja con la apertura de archivos en forma de lectura o escritura

Figura 3



Fuente: propia

Observa que se crea un objeto por medio de la clase **open** y recibe como argumentos dos datos, el primero es el nombre del archivo y el segundo el modo en el cual será abierto el archivo. Por lo general se usa la letra "w" para decirle al programa que abra el archivo en forma de escritura, y la letra "r" se usará para decirle que lo abra en forma de lectura. Otro modificador es la letra "a" que se usa para agregar datos al archivo sin borrar los ya existentes. Luego se implementan en este ejercicio los métodos **.write()** para escribir información en ese archivo y el método **.close()** para cerrar el archivo correctamente. Las dos formas de crear un archivo son correctas y será el programador quien decida la mejor opción según lo desee. Es más usual encontrar códigos implementando la instrucción **with**.



write:

Método de la clase open que permite la escritura de un archivo

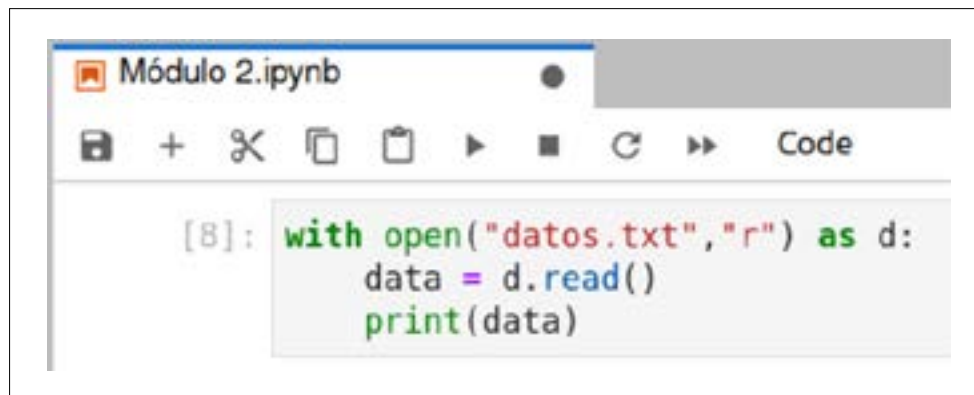
close:

Método de la clase open que permite el cierre de un archivo

Ahora, si queremos recuperar su contenido en una variable, lo podemos realizar de la siguiente manera:

```
with open("datos.txt","r") as d:  
    data = d.read()  
    print(data)
```

Figura 4



Fuente: propia

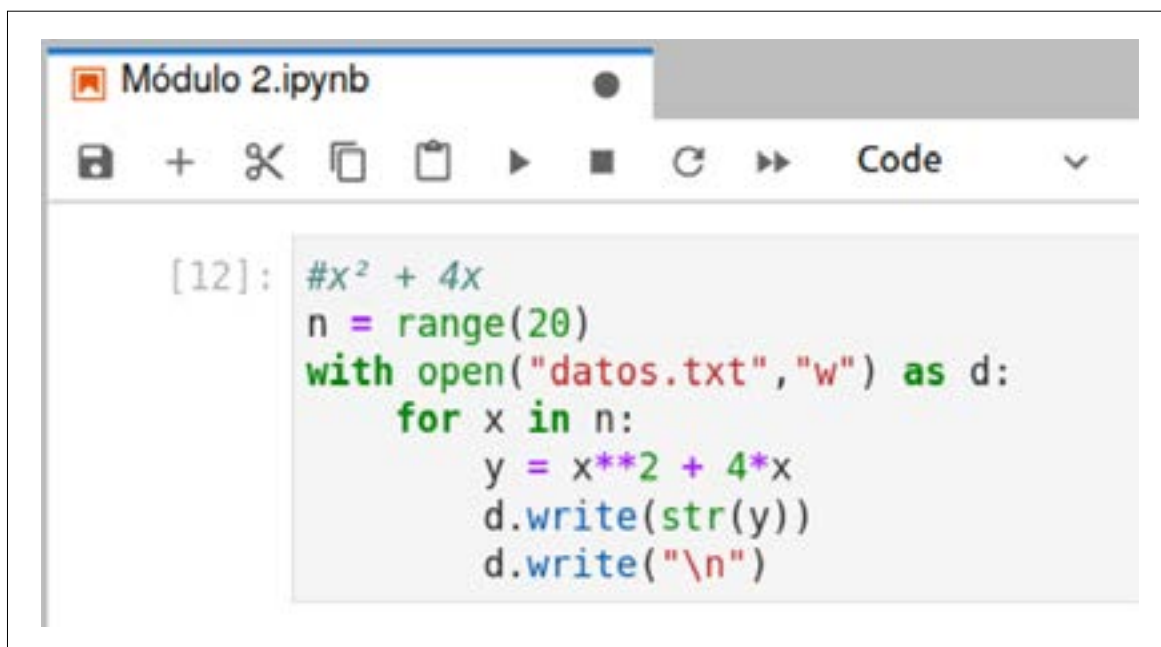
Aquí, al usar el método `.read()` se cargarán en la variable **data** todos los datos contenidos en un archivo. Si se desea leer varias líneas en un archivo, se implementaría el método `.readline()` en un ciclo. Ahora que ya sabes cómo utilizar la clase `open`, es hora de que practiques creando y recuperando tus propios datos

Ejercicios:

1. Crea un archivo con los 20 primeros datos de la expresión matemática y luego recupera los datos en una lista
2. Crea un archivo con los 30 primeros datos de la expresión matemática $x + 10 - 4k$ donde $k = 10$

Como ayuda, veamos una posible solución del primer ejercicio.

Figura 5



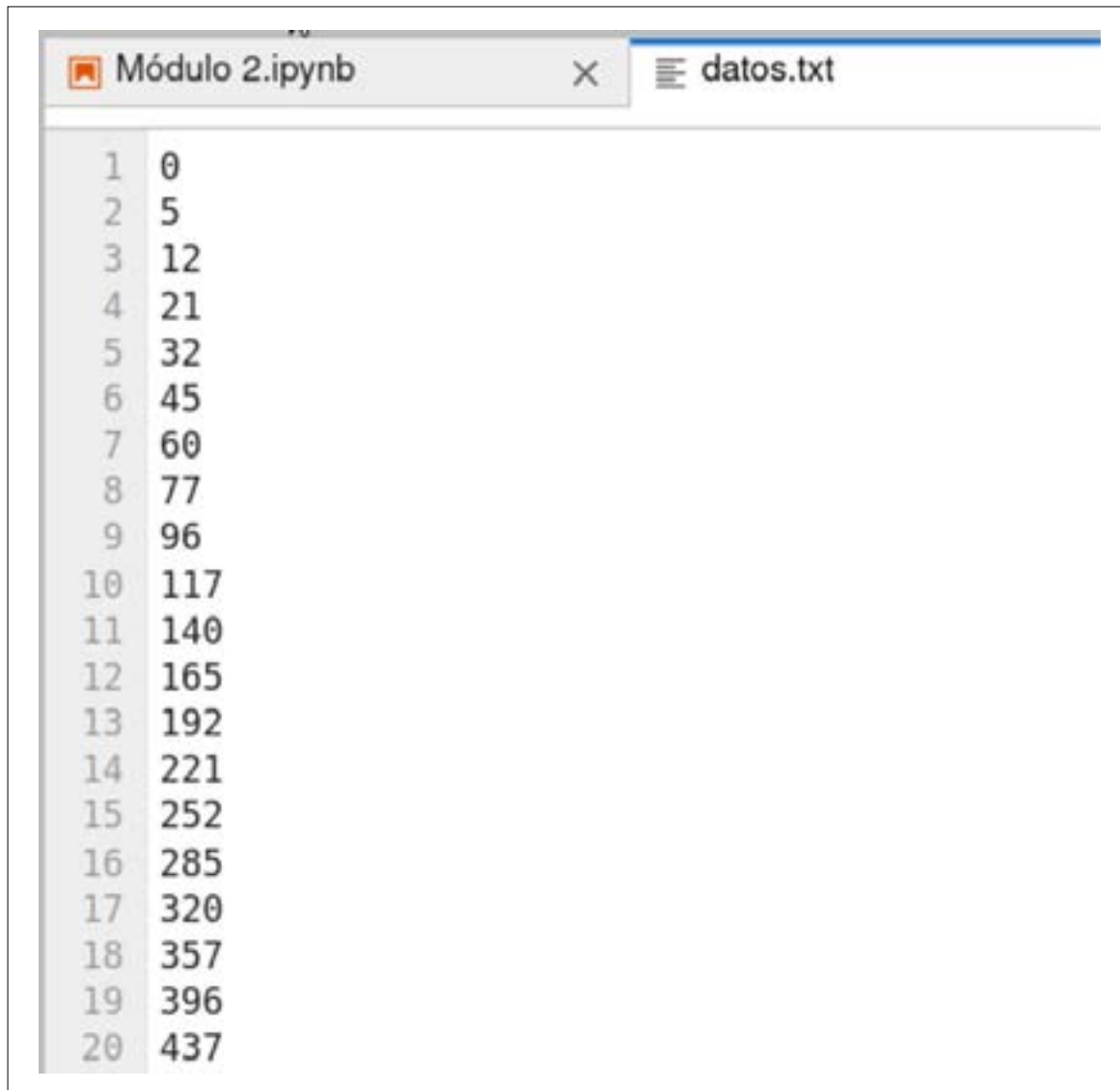
```
[12]: #x2 + 4x
n = range(20)
with open("datos.txt", "w") as d:
    for x in n:
        y = x**2 + 4*x
        d.write(str(y))
        d.write("\n")
```

Fuente: propia

Empezamos por crear nuestro rango de números y luego abrimos el archivo en forma de escritura. Recuerda que con el modificador "**w**" los datos nuevos siempre reemplazarán a los datos antiguos.

Dentro del bloque **with**, se ha implementado un bucle **for** para ir iterando cada operación matemática e ir escribiendo su resultado en el archivo. Observa como cada dato que pasa, debe ser convertido a texto usando la función **str()**. Finalmente para este ejercicio, se ha usado un modificador "**\n**" que crea un salto de línea; Es como dar Enter en cada paso (De no hacerlo, los datos quedarían pegados unos a otros). Esto creará una sola columna o campo con los datos de los primeros 20 resultados de dicha expresión matemática.

Figura 6



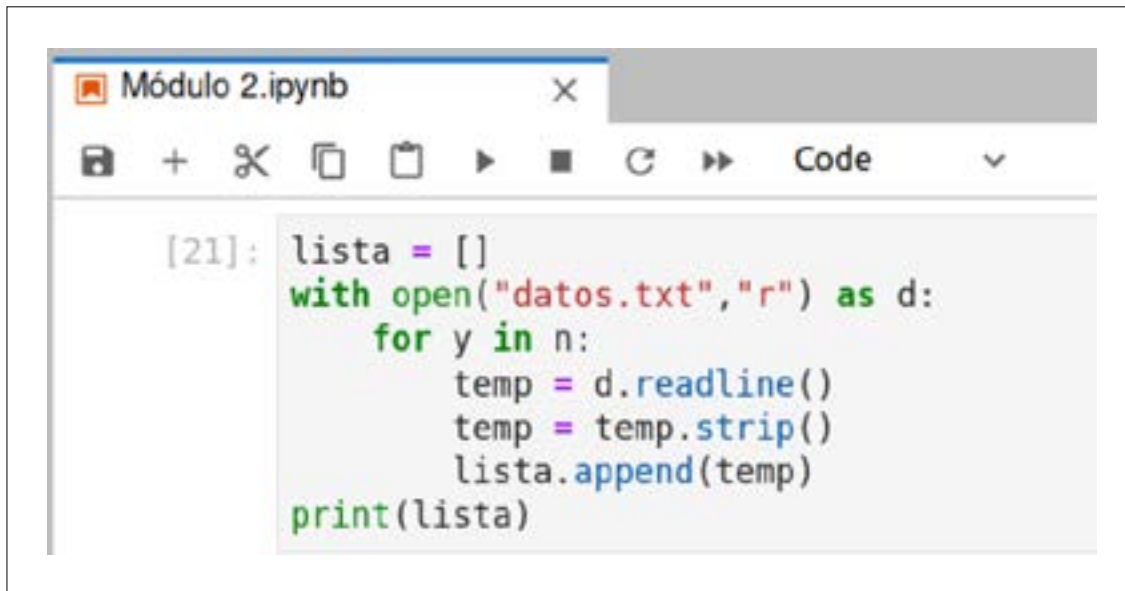
The image shows a Jupyter Notebook interface with two tabs: 'Módulo 2.ipynb' and 'datos.txt'. The 'datos.txt' tab is active, displaying a list of 20 numbers, each preceded by a line number from 1 to 20. The numbers are: 0, 5, 12, 21, 32, 45, 60, 77, 96, 117, 140, 165, 192, 221, 252, 285, 320, 357, 396, and 437.

1	0
2	5
3	12
4	21
5	32
6	45
7	60
8	77
9	96
10	117
11	140
12	165
13	192
14	221
15	252
16	285
17	320
18	357
19	396
20	437

Fuente: propia

Es hora de recuperar estos datos en Python, dentro de una lista, para dar solución al ejercicio. Un posible código sería el siguiente:

Figura 7



```
[21]: lista = []
      with open("datos.txt", "r") as d:
          for y in n:
              temp = d.readline()
              temp = temp.strip()
              lista.append(temp)
      print(lista)
```

Fuente: propia

Aquí, es importante notar que los datos deben limpiarse por el caracter de escape que cambia de línea "\n". El método **readline()** para este caso lee cada línea como si fuese la fila o registro del único campo que se tiene en este momento y el método **strip()** limpia el caracter de escape. De lo contrario, los datos aparecerán con dicho caracter.

Es hora de ver ejemplos que se ajustan al trabajo común de recolectar datos con librerías especializadas y a continuación empezaremos con archivos CSV

Archivos CSV

Un archivo CSV (Comma Separated Values) contiene normalmente varios datos como si se tratara de una tabla, lo cual conlleva a pensar que la primera línea es el encabezado y que a partir de la segunda línea es que se empieza a tener los datos. Como ejemplo considere el siguiente archivo CSV:

Figura 8



Delimitador: ,

	nombre	Física	Química	Artes
1	Francisco Alvarez	3,5	4,7	3,9
2	Juan Gonzalez	4,1	3,8	4,5
3	Mauricio Castro	3,2	3,4	4,7
4	Saray Lozano	4,5	4,3	4,8
5	Viviana Cáceres	3,9	4	3,6

Fuente: propia

Este archivo contiene encabezados, nombres con espacios y datos numéricos de tipo decimal con una coma como separador. En un bloc de notas el archivo debería verse de la siguiente manera:

Figura 9

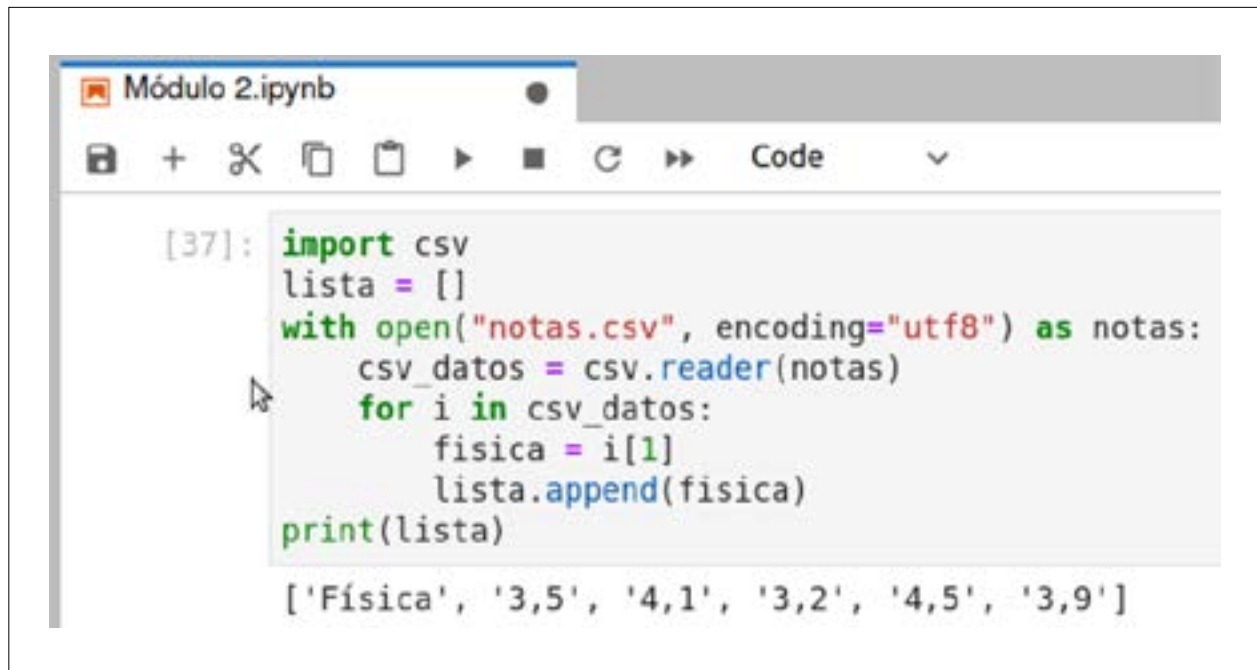
```

nombre,Física,Química,Artes
Francisco Alvarez,"3,5","4,7","3,9"
Juan Gonzalez,"4,1","3,8","4,5"
Mauricio Castro,"3,2","3,4","4,7"
Saray Lozano,"4,5","4,3","4,8"
Viviana Cáceres,"3,9",4,"3,6"
    
```

Fuente: propia

Es de notar que algunos datos están encerrados en comillas dobles y otros no, puesto que el delimitador es la coma. Contiene también datos de tipo texto con tildes y en ese orden de ideas, un posible código que puede cargar nuestros datos en una lista sería el siguiente:

Figura 10



```
[37]: import csv
      lista = []
      with open("notas.csv", encoding="utf8") as notas:
          csv_datos = csv.reader(notas)
          for i in csv_datos:
              fisica = i[1]
              lista.append(fisica)
      print(lista)

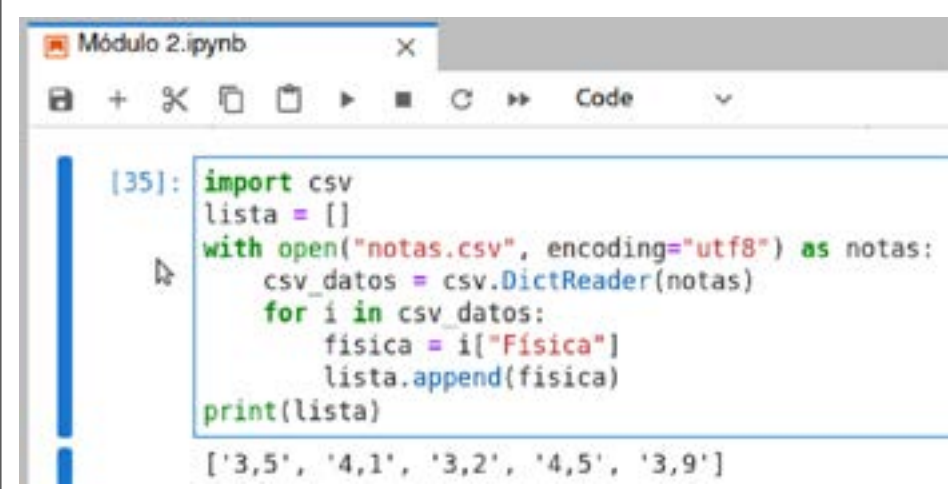
      ['Física', '3,5', '4,1', '3,2', '4,5', '3,9']
```

Fuente: propia

Hemos empleado el llamado de la clase **CSV** con el propósito de crear un objeto y emplear sus métodos para extraer los datos que necesitamos de dicho archivo. Se ha utilizado también un parámetro para que nuestro objeto lea correctamente los caracteres especiales como las tildes, por medio del modificador **encoding = "utf8"**.

Finalmente, estamos usando el método **.reader()** de la clase CSV para cargar todos los datos en el nuevo objeto **csv_datos**. Posteriormente empleamos un bucle **for** para obtener únicamente los datos pertenecientes a la columna o campo Física, pero es de notar que en esta lista de datos tendremos también dicha palabra. Una solución práctica nos la brinda esta clase por medio del método **.DictReader()** que reconoce el primer dato como un encabezado

Figura 11



```
Módulo 2.ipynb
[35]: import csv
      lista = []
      with open("notas.csv", encoding="utf8") as notas:
          csv_datos = csv.DictReader(notas)
          for i in csv_datos:
              fisica = i["Física"]
              lista.append(fisica)
      print(lista)

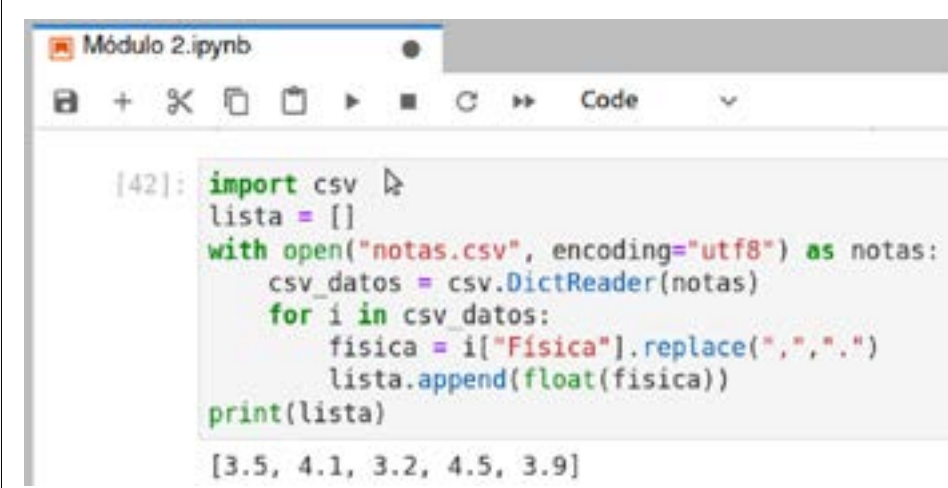
['3,5', '4,1', '3,2', '4,5', '3,9']
```

Fuente: propia

Mira que este método, como su nombre lo indica, crea un diccionario con los datos del archivo CSV y por ello resulta incluso más fácil obtener los datos por el nombre del campo o columna en cuestión, recuperando así solo los datos que necesitamos en este caso y pasándolos a una lista para su uso posterior.

Observa que los datos siguen teniendo comillas, lo cual indica que todo dato que se lea desde un fichero o archivo CSV, es texto. Para convertir estos datos a un tipo de dato numérico, debemos emplear el método **replace()** del objeto **str** y la función **float()**, puesto que estos números tienen parte decimal y están separados por comas en este ejemplo, pero para Python, el separador de los decimales es el símbolo punto.

Figura 12



```
Módulo 2.ipynb
[42]: import csv
      lista = []
      with open("notas.csv", encoding="utf8") as notas:
          csv_datos = csv.DictReader(notas)
          for i in csv_datos:
              fisica = i["Física"].replace(",",".")
              lista.append(float(fisica))
      print(lista)

[3.5, 4.1, 3.2, 4.5, 3.9]
```

Fuente: propia

De esta forma, ya tendríamos nuestros datos listos para realizar las operaciones matemáticas que necesitemos.

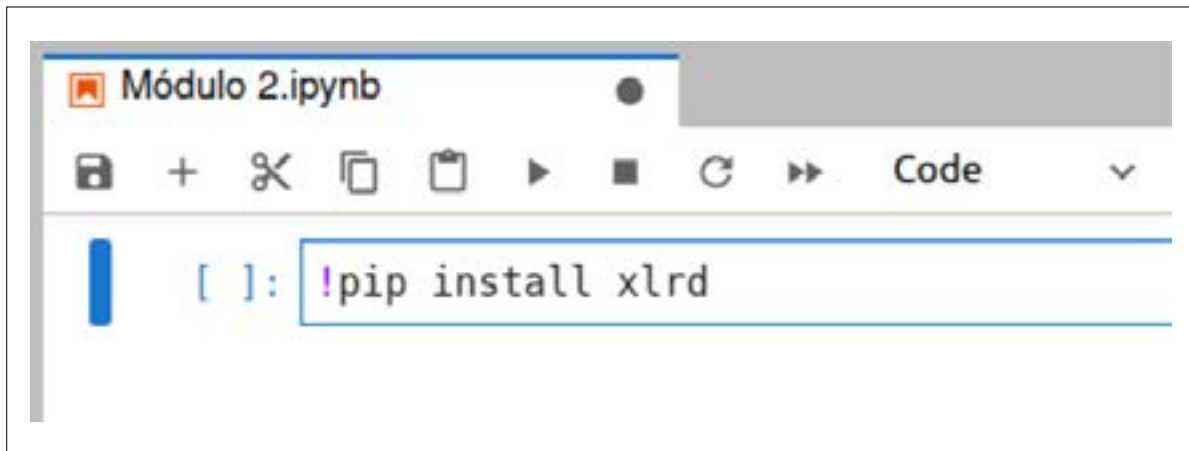
Archivos de Excel

Un archivo de excel se caracteriza por una estructura de libro y hojas de cálculo, que suelen tener algunas veces fórmulas, cadenas de texto, números y fechas entre otros datos. Cada hoja es un conjunto de celdas y aunque Python no tiene una biblioteca estándar para manipular archivos de Excel, si tiene dos bibliotecas popularmente usadas que son **xlrd** para leer archivos **XLS (formato 97 - 2003)** y **xlwt** para crearlos. (Caballero, 2019)

Guardando el archivo CSV ahora como un archivo de Excel, vamos a mostrar cómo recuperar información usando la biblioteca **xlrd**.

Primero se debe instalar la biblioteca xlrd en Python ejecutando la siguiente instrucción: `!pip install xlrd`

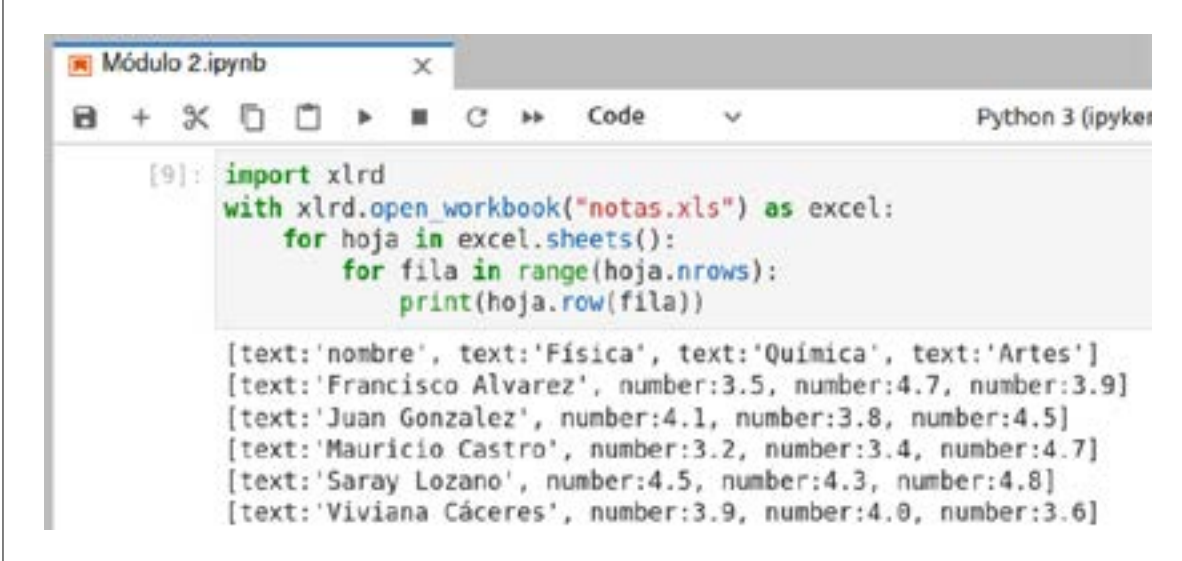
Figura 13



Fuente: propia

Una vez instalada la biblioteca, podremos ejecutar el siguiente código:

Figura 14



```

Módulo 2.ipynb
[9]: import xlrd
    with xlrd.open_workbook("notas.xls") as excel:
        for hoja in excel.sheets():
            for fila in range(hoja.nrows):
                print(hoja.row(fila))

[text:'nombre', text:'Física', text:'Química', text:'Artes']
[text:'Francisco Alvarez', number:3.5, number:4.7, number:3.9]
[text:'Juan Gonzalez', number:4.1, number:3.8, number:4.5]
[text:'Mauricio Castro', number:3.2, number:3.4, number:4.7]
[text:'Saray Lozano', number:4.5, number:4.3, number:4.8]
[text:'Viviana Cáceres', number:3.9, number:4.0, number:3.6]
    
```

Fuente: propia

En este código, se ha empleado el método **.open_workbook()** para acceder a la ruta donde está el archivo de excel. El método **.sheets()** devuelve una lista con todos los nombres de las hojas dentro del libro de excel. El atributo **nrows** devuelve el número de filas que contienen datos y finalmente el método **.row()** que recupera la información de cada fila en una lista.

Si deseamos recuperar solo los datos de la columna Artes, un posible código sería el siguiente:

Figura 15



```

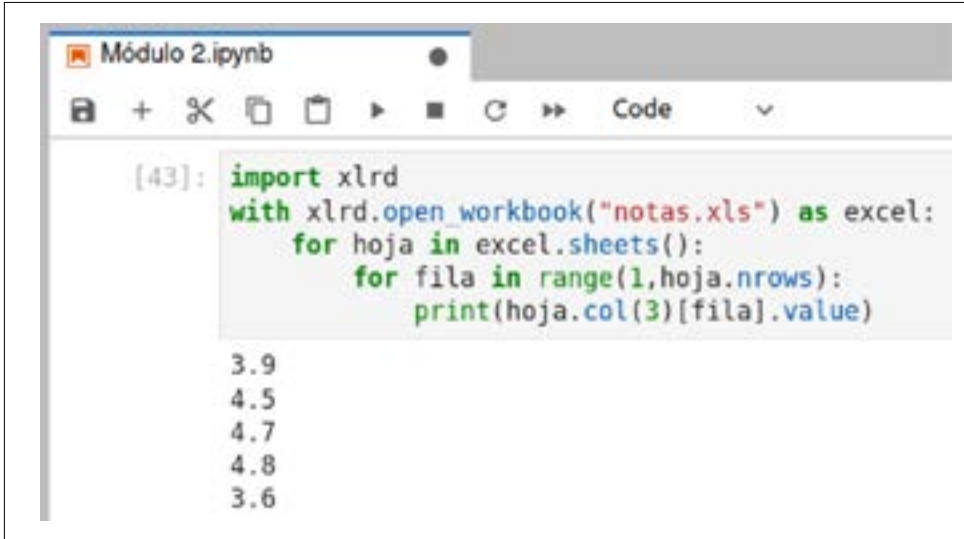
Módulo 2.pynb
[30]: import xlrd
    with xlrd.open_workbook("notas.xls") as excel:
        for hoja in excel.sheets():
            print(hoja.col(3))

[text:'Artes', number:3.9, number:4.5, number:4.7, number:4.8, number:3.6]
    
```

Fuente: propia

Como vemos, es una lista donde cada dato está separado por comas y contiene tanto el tipo de dato como su valor. Aquí se ha usado el método `.col()` para decirle a Python que cargue sólo la columna que queremos. Para obtener solo los valores y hacerlo sin la cabecera, el código podría ser el siguiente:

Figura 16



```
[43]: import xlrd
with xlrd.open_workbook("notas.xls") as excel:
    for hoja in excel.sheets():
        for fila in range(1, hoja.nrows):
            print(hoja.col(3)[fila].value)

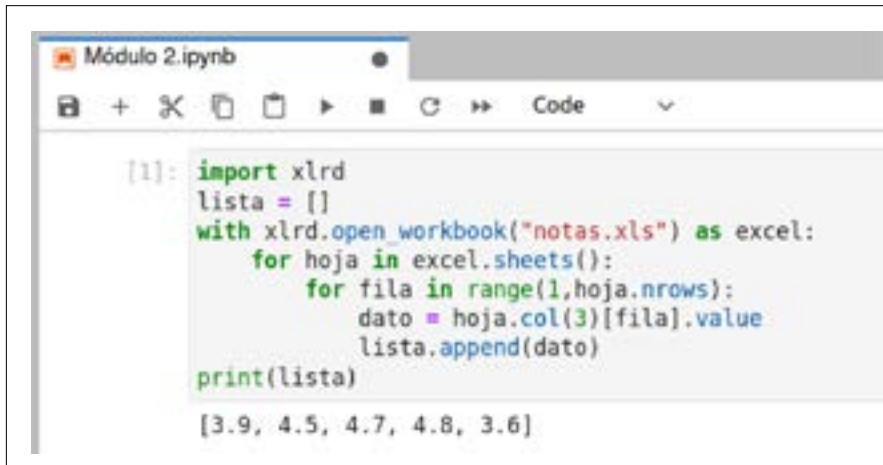
3.9
4.5
4.7
4.8
3.6
```

Fuente: propia

Observa que nuevamente se ha iterado en un **bucle for** para que lea los datos desde la fila con índice uno (1) puesto que la primera fila, la de índice cero (0), contiene los encabezados. Al momento de llamar los registros o datos de esa columna, se hace uso del atributo `.value` que se encarga de recuperar solo el contenido, los datos.

Puede probar recuperar estos datos en una lista de la siguiente forma:

Figura 17



```
[1]: import xlrd
lista = []
with xlrd.open_workbook("notas.xls") as excel:
    for hoja in excel.sheets():
        for fila in range(1, hoja.nrows):
            dato = hoja.col(3)[fila].value
            lista.append(dato)
print(lista)

[3.9, 4.5, 4.7, 4.8, 3.6]
```

Fuente: propia

Conclusiones

Al manejar archivos de texto o cualquier otro tipo de archivo como CSV o Excel, se recomienda hacerlo dentro de un bloque **with** para asegurar el cierre correcto del archivo después de haber leído o escrito datos. Los modificadores que contiene un archivo de tipo texto en su apertura son modo de lectura **r**, modo de sobrescritura **w**, el cual elimina el contenido anterior y lo reemplaza por el nuevo, y modo de escritura anidada **a**, el cual no elimina el contenido anterior, sino que anida o suma el nuevo contenido



Lectura recomendada

Te invito a realizar la siguiente lectura:

[Formatos de archivo](#)

Python

INTRODUCCIÓN

Existen formatos de datos que se desarrollaron para garantizar la compatibilidad entre aplicaciones sin importar el lenguaje en el cual se desarrollan, teniendo mucho auge en el desarrollo de aplicaciones web. Hoy en día, se emplean ciertos formatos como **JSON** y **XML** principalmente para compartir datos entre aplicaciones, pues éstas pueden comunicarse entre sí sin intervención de un usuario y son conocidas como API.



Lectura de Archivos y APIs



JSON

Este tipo de formato tiene mucha similitud con los diccionarios, porque son objetos de parejas de datos donde uno de ellos es la clave y el otro el valor. JSON (JavaScript Object Notation) es un estándar para compartir datos mediante APIS, lo cual veremos más adelante

Un archivo JSON tendría el siguiente formato

```
[  
  {"clave1":valor,"clave2":valor},  
  {"clave1":valor,"clave2":valor},  
  {"clave1":valor,"clave2":valor}  
]
```

Figura 18

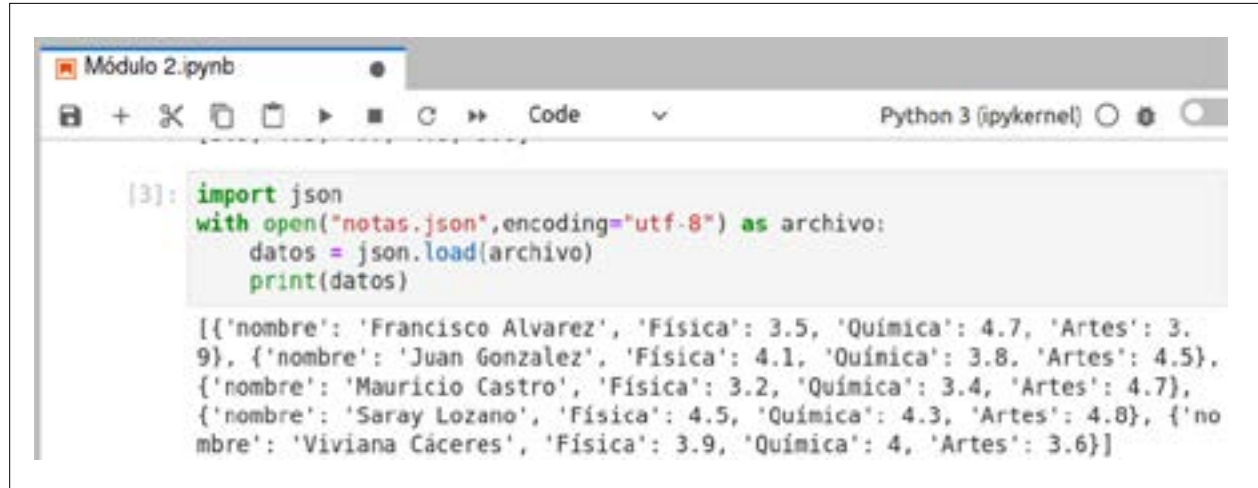
```
[  
  {"nombre":"Francisco Alvarez","Fisica":3.5,"Química":4.7,"Artes":3.9}  
  {"nombre":"Juan Gonzalez","Fisica":4.1,"Química":3.8,"Artes":4.5},  
  {"nombre":"Mauricio Castro","Fisica":3.2,"Química":3.4,"Artes":4.7},  
  {"nombre":"Saray Lozano","Fisica":4.5,"Química":4.3,"Artes":4.8},  
  {"nombre":"Viviana Cáceres","Fisica":3.9,"Química":4,"Artes":3.6}  
]
```

Fuente: propia

Vemos que en esencia puede considerarse una lista que contiene como elementos diccionarios. Los formatos JSON pueden ser en esencia Listas o Diccionarios.

Para realizar la recuperación de datos tal cual como lo hemos realizado en los ejemplos y ejercicios anteriores, una posible solución sería la siguiente:

Figura 19



```
Módulo 2.ipynb
Python 3 (ipykernel)

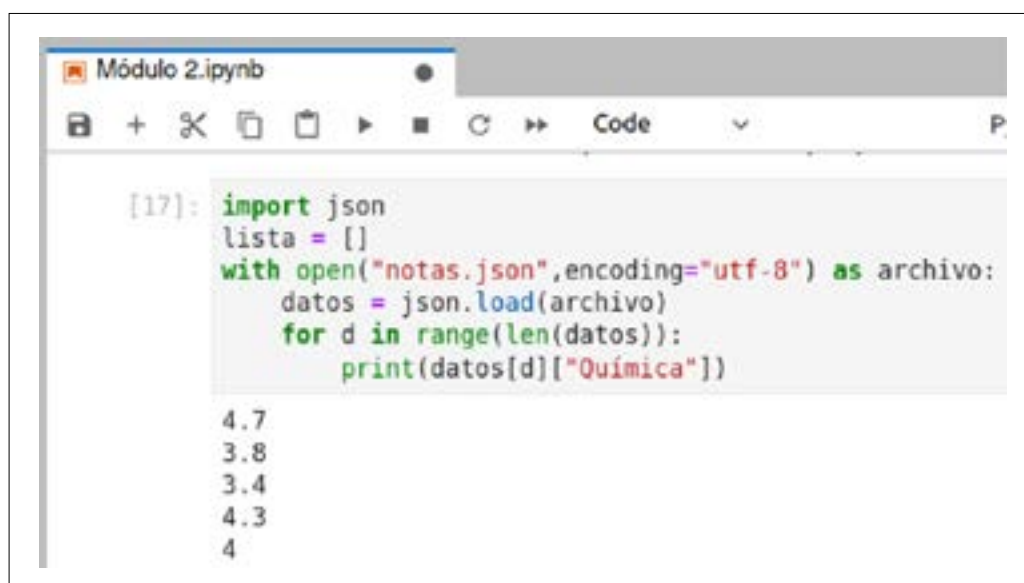
[3]: import json
with open("notas.json", encoding="utf-8") as archivo:
    datos = json.load(archivo)
    print(datos)

[{'nombre': 'Francisco Alvarez', 'Física': 3.5, 'Química': 4.7, 'Artes': 3.9}, {'nombre': 'Juan Gonzalez', 'Física': 4.1, 'Química': 3.8, 'Artes': 4.5}, {'nombre': 'Mauricio Castro', 'Física': 3.2, 'Química': 3.4, 'Artes': 4.7}, {'nombre': 'Saray Lozano', 'Física': 4.5, 'Química': 4.3, 'Artes': 4.8}, {'nombre': 'Viviana Cáceres', 'Física': 3.9, 'Química': 4, 'Artes': 3.6}]
```

Fuente: propia

Observa que cada biblioteca, cada clase tiene sus propios métodos y no son compatibles entre sí. En este caso, estamos usando la instrucción **import json** para cargar esta biblioteca y emplear el método **.load()** que recupera correctamente todos los datos a nuestro programa. Si deseamos obtener, como hemos hecho antes, solo los datos numéricos de una columna en especial, lo podremos hacer usando el indexado correcto como presentamos a continuación:

Figura 20



```
Módulo 2.ipynb
Python 3 (ipykernel)

[17]: import json
lista = []
with open("notas.json", encoding="utf-8") as archivo:
    datos = json.load(archivo)
    for d in range(len(datos)):
        print(datos[d]["Química"])

4.7
3.8
3.4
4.3
4
```

Fuente: propia

Aquí simplemente se ha hecho uso del indexado por número y clave dentro de un **bucle for** para extraer solo los datos de la columna o campo Química. Recuerda que en este ejemplo, el archivo JSON primero es una lista y luego un diccionario.

La función **len ()** se ha implementado para conocer el tamaño de la lista y que el objeto **range ()** cree el rango correspondiente de elementos para este bucle

XML

(eXtensible Markup Language) es un formato que almacena datos en etiquetas, algo muy similar a una estructura HTML de una página web. Las etiquetas tienen una característica y es su apertura y cierre:

```
<xml>
  <nombre>Saray Lozano</nombre>
  <fisica>4.5</fisica>
</xml>
```

Un documento XML tiene entonces una estructura de etiquetas donde se cumple que una abre y otra cierra el contenido de un dato y puede entenderse como bloques o cajas que contienen a su vez otros bloques de etiquetas con datos.

Figura 21

```
<?xml version="1.0" encoding="UTF-8">
  <fila>
    <nombre>"Francisco Alvarez"</nombre>
    <Física>3.5</Física>
    <Química>4.7</Química>
    <Artes>3.9</Artes>
  </fila>
  <fila>
    <nombre>"Juan Gonzalez"</nombre>
    <Física>4.1</Física>
    <Química>3.8</Química>
    <Artes>4.5</Artes>
  </fila>
```

Fuente: propia

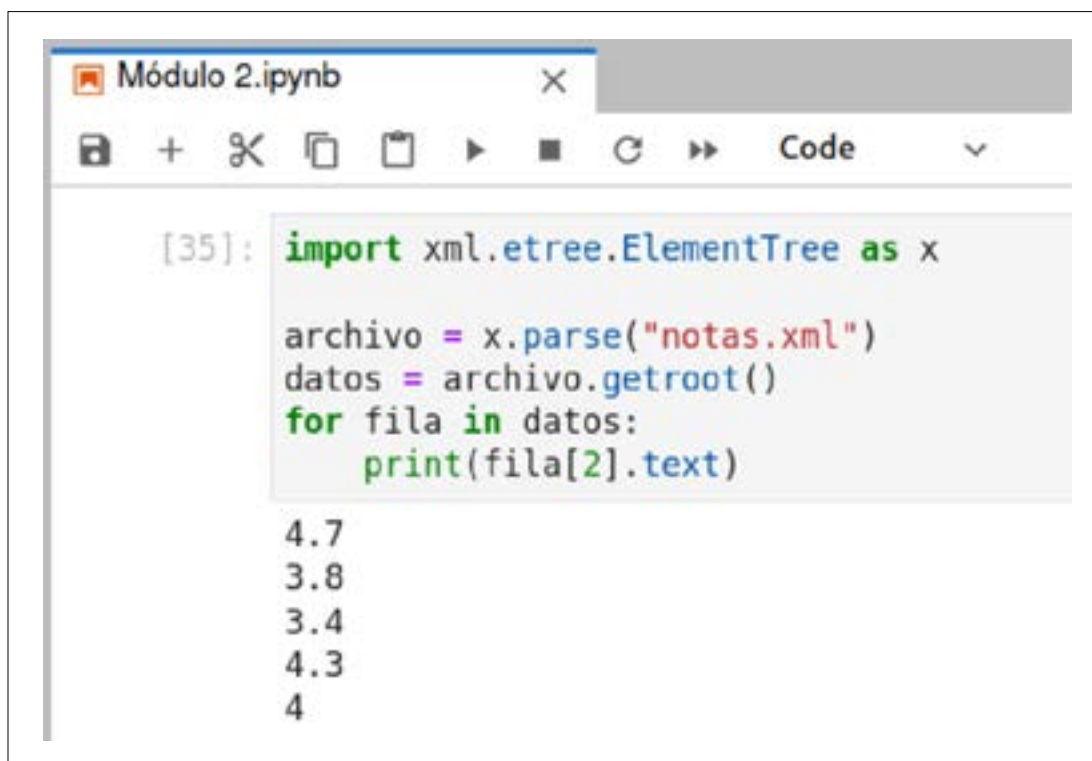
En Python, la biblioteca XML tiene cuatro paquetes de bibliotecas que son:

```
[ dom, parsers, etree, sax ]
```

Por facilidad, mostraremos el uso de `etree`.

Para leer y recuperar los datos de un archivo con extensión XML se hace lo siguiente:

Figura 22



```
[35]: import xml.etree.ElementTree as x

      archivo = x.parse("notas.xml")
      datos = archivo.getroot()
      for fila in datos:
          print(fila[2].text)

4.7
3.8
3.4
4.3
4
```

Fuente: propia

Se importa por completo el módulo `xml.etree.ElementTree` como `x`. El método `.parse()` crea un objeto de la clase `ElementTree` y el método `.getroot()` crea otro objeto de la clase `Element` correspondiente al objeto creado con `.parse()`

Finalmente, puede iterarse en un **bucle for** este último y obtener su contenido con el atributo `.text`. Para mostrar solo el contenido de una etiqueta, se indexa con índices numéricos, y en este caso, el número 2 corresponde con la etiqueta Química.

Recolección mediante APIs

Hoy en día, la mayoría de las aplicaciones comparten datos mediante APIs (Application Programming Interface) que son bibliotecas que ofrecen ciertas funciones para que otras aplicaciones puedan acceder a ellas.

Existe un protocolo denominado API-REST empleado para enviar y recibir datos particularmente en formato XML o JSON por medio del protocolo HTTP. Para esto, se hace uso de cuatro funciones que son POST, GET, PUT y DELETE. Veamos cómo se recupera información de una API

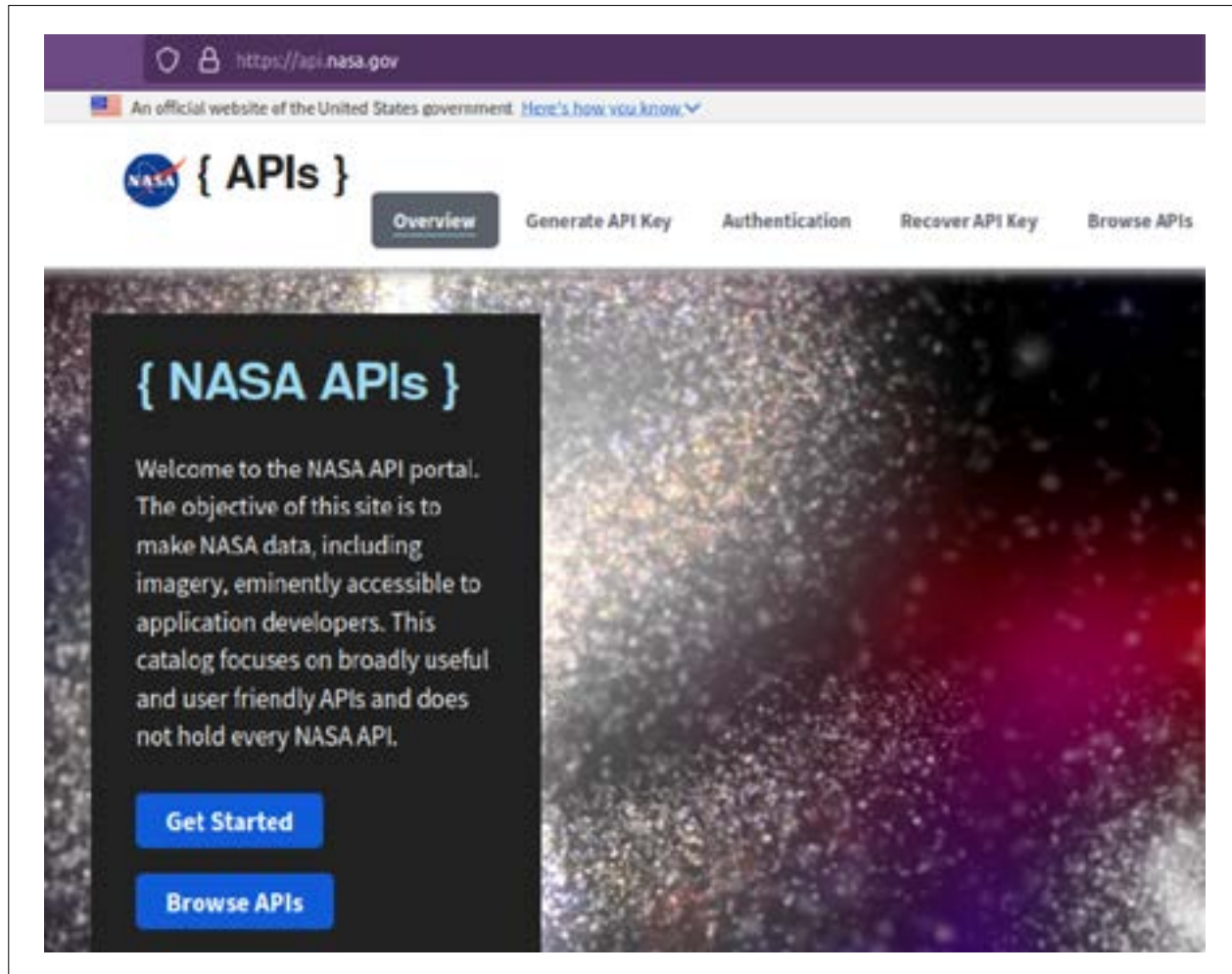
Algunas APIs son públicas y no necesitan de claves privadas o TOKENS especiales para su acceso. A modo de ejemplo, veremos cómo usar la API de la Nasa para obtener una imagen del día.



Visitar página

Primero se debe ir al sitio web de la API de la Nasa:
<https://api.nasa.gov/>

Figura 23

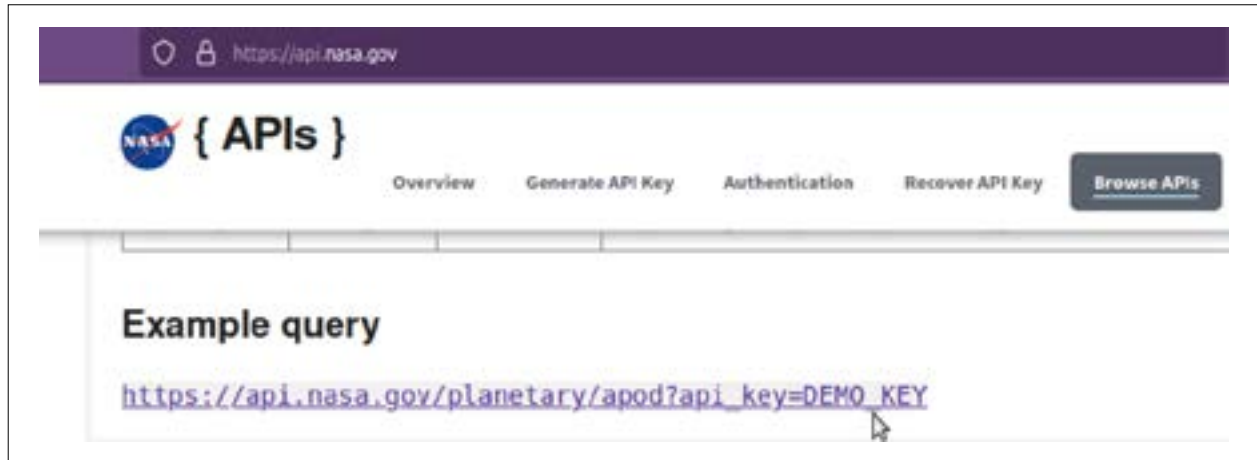


Fuente: propia

Una vez allí, en el enlace o botón Browse APIs vamos a encontrar una forma de acceder con un TOKEN DEMO, es decir, sin la necesidad de crear una cuenta, puesto que la mayoría de las APIs que permiten capturar grandes volúmenes de datos, requieren crear una cuenta para usar un TOKEN personal (Tweeter, Telegram, Webex de Cisco, entre otras)

Ahora damos clic en el primer enlace, APOD (Astronomy Picture of de Day).

Figura 24

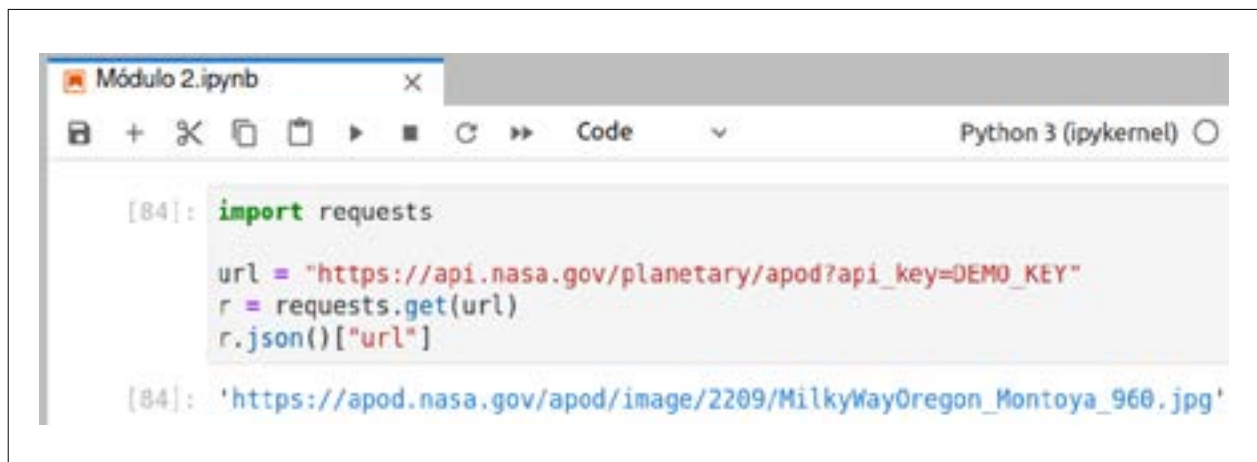


Fuente: propia

Al navegar un poco, vamos a encontrar el subtítulo Example query con el TOKEN DEMO. Simplemente lo tendremos en cuenta para desarrollar el siguiente código, el cuál nos dará acceso a la imagen del día usando la biblioteca **requests** y el método **.get()**.

```
import requests
url = "https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY"
r = requests.get(url)
r.json()["url"]
```

Figura 25



Fuente: propia

Se obtiene como resultado un enlace hacia la imagen. Aquí notamos algunas cosas:

1. El uso de **`import requests`** para llamar a la biblioteca que nos permite hacer peticiones a páginas web
2. La creación del objeto **`r`** usando **`requests.get()`**
3. El uso del método **`.json()`** de requests para cargar los datos JSON
4. El uso del índice **`["url"]`**

Para poder conocer estos aspectos, cada API tiene su propia documentación y es muy importante leerla, pues cada una tendrá sus propios índices y también métodos y atributos para dar información específica. En este caso, si se quisiera conocer cuáles índices se tienen disponibles para usar, se emplearía el siguiente código usando el método **`.keys()`**:

Figura 26

```
import requests

url = "https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY"
r = requests.get(url)
r.json().keys()

dict_keys(['copyright', 'date', 'explanation', 'hdurl', 'media_type', 'service_version', 'title', 'url'])
```

Fuente: propia

Otras APIs como Google Maps, Twitter, piden que se cree una cuenta de facturación o un inicio de sesión para otorgar una API-KEY o TOKEN de acceso para poder usar sus APIs. Por esta razón, nos centramos en APIs que tengan TOKENs gratuitos o que simplemente no se requiera por el momento, como en este momento, la API de la NASA.



Visitar página

En el sitio web <https://datosabiertos.bogota.gov.co/> podemos encontrar por ejemplo, una API para ver los casos confirmados de covid en Bogotá a la fecha. Navegando por el sitio web encontramos directamente la siguiente página donde podremos usar los beneficios de un API-REST o también descargar todos los datos en formato CSV: <https://datosabiertos.bogota.gov.co/dataset/numero-de-casos-confirmados-por-el-laboratorio-de-covid-19-bogota-d-c/resource/b64ba3c4-9e41-41b8-b3fd-2da21d627558>

Usando Python obtendremos los datos de la siguiente forma:

Figura 27

```
Módulo 2.ipynb
[127]: import requests
url = "https://datosabiertos.bogota.gov.co/api/3/"

r = requests.get(url)
datos = r.json()["result"]["records"]
print(json.dumps(datos, indent=4))

[
  {
    "EDAD": "20",
    "FECHA_DIAGNOSTICO": "2021-06-18",
    "CASO": "1147919",
    "FUENTE_O_TIPO_DE_CONTAGIO": "En estudio"
  }
]
```

Fuente: propia

Aquí hemos usado los índices clave **result** y **records** obtenidos previamente al analizar la variable **datos** con el método **.keys()** que contiene el JSON que nos entregan desde el sitio web y retorna en la variable **datos** una lista con diccionarios de los datos de contagios de covid-19 en Bogotá. Siguiendo esta lógica, se podrá iterar en bucles y por ejemplo obtener los datos filtrados por localidad, edad, fuente o tipo de contagio, entre otros índices clave que tienen estos datos. Para su visualización en un formato un poco más organizado, se ha empleado el método **json.dumps()** con un argumento **indent = 4**

Si quisiéramos obtener los datos de quienes se han recuperado en la localidad de Suba, un posible código, luego de analizar el formato JSON y ver sus claves, sería el siguiente:

Figura 28

```
import requests
url = "https://datosabiertos.bogota.gov.co/api/3/action/datastore

r = requests.get(url)
datos = r.json()["result"]["records"]
for i in datos:
    if i["LOCALIDAD_ASIS"]=="Suba" and i["ESTADO"]=="Recuperado":
        print(json.dumps(i,indent=4))
#print(json.dumps(datos,indent=4))

{
  "EDAD": "85",
  "FECHA_DIAGNOSTICO": "2021-06-18",
  "CASO": "1147934",
  "FUENTE_O_TIPO_DE_CONTAGIO": "En estudio",
  "SEXO": "F",
  "CIUDAD": "Bogot\u00e1",
  "FECHA_DE_INICIO_DE_SINTOMAS": null,
  "UBICACION": "Casa",
  "_id": 1147934,
  "ESTADO": "Recuperado",
```

Fuente: propia

Serialización de Objetos (joblib)

Algunas veces, en el desarrollo de algoritmos para Ciencias de Datos, necesitamos guardar y recuperar ciertos modelos como una secuencia de bytes que compacta los mismos. Se usaba el módulo Pickler para esta tarea pero en este momento es considerado inseguro y solo se recomienda su uso de deserialización desde fuentes confiables, pues por este método se podría filtrar archivos maliciosos. La forma implementada más acertada para serializar y deserializar objetos es la siguiente:

#Serializar datos

```
import joblib
datos = ["nombre", "asignatura", "calificación"]
joblib.dump(datos, "notas.pkl")
```

#Recuperar datos serializados

```
carga = joblib.load("notas.pkl")
```

Figura 29

```
[10]: import joblib
      datos = ["nombre", "asignatura", "calificación"]
      joblib.dump(datos, "notas.pkl")

[10]: ['notas.pkl']

[11]: carga = joblib.load("notas.pkl")

[12]: carga

[12]: ['nombre', 'asignatura', 'calificación']
```

Fuente: propia

El método ***dump()*** se encarga de serializar los datos y guardarlos

El método ***load()*** se encarga de recuperar los datos para su nuevo uso

Esta forma de almacenar y recuperar datos localmente se utilizará en el módulo 4 cuando se necesite usar la persistencia de modelos al entrenar los clasificadores como SVM o KNN (temas que se desarrollarán más adelante)

Conclusiones

De forma práctica se recomienda implementar el uso del formato JSON al momento de compartir datos con otros programas, aplicaciones o APIs, tanto por su sencillez como por su compatibilidad con el tipo de datos ***dict*** (diccionario) de Python. Recuerde que en este tipo de dato los datos están encerrados entre corchetes y pueden almacenar en su valor todo tipo de dato como listas, tuplas, textos, números y otros diccionarios.

Es importante que al manejar APIs, se cuente con el TOKEN que permite acceder a la aplicación sin la necesidad de entregar usuarios y contraseñas en los códigos. Algunos de estos TOKENs tienen caducidad de tiempo, lo que hace que su implementación sea muy viable en términos de seguridad.



Lectura recomendada

Te invito a que realices las siguientes lecturas:

Persistencia de datos

Python

JSON

Python

CSV

<https://docs.python.org/es/3.8/library/csv.html>



Procesos - pasos



Los datos desde diferentes archivos ya no serán un problema para Saray, pues puede implementar bucles para leer diferentes archivos o simplemente pedir que por política institucional, los nombres de los archivos estén en un formato de nombre específico como el nombre de la asignatura, para que al cargar el archivo, el programa no muestre ningún error.

```
import csv

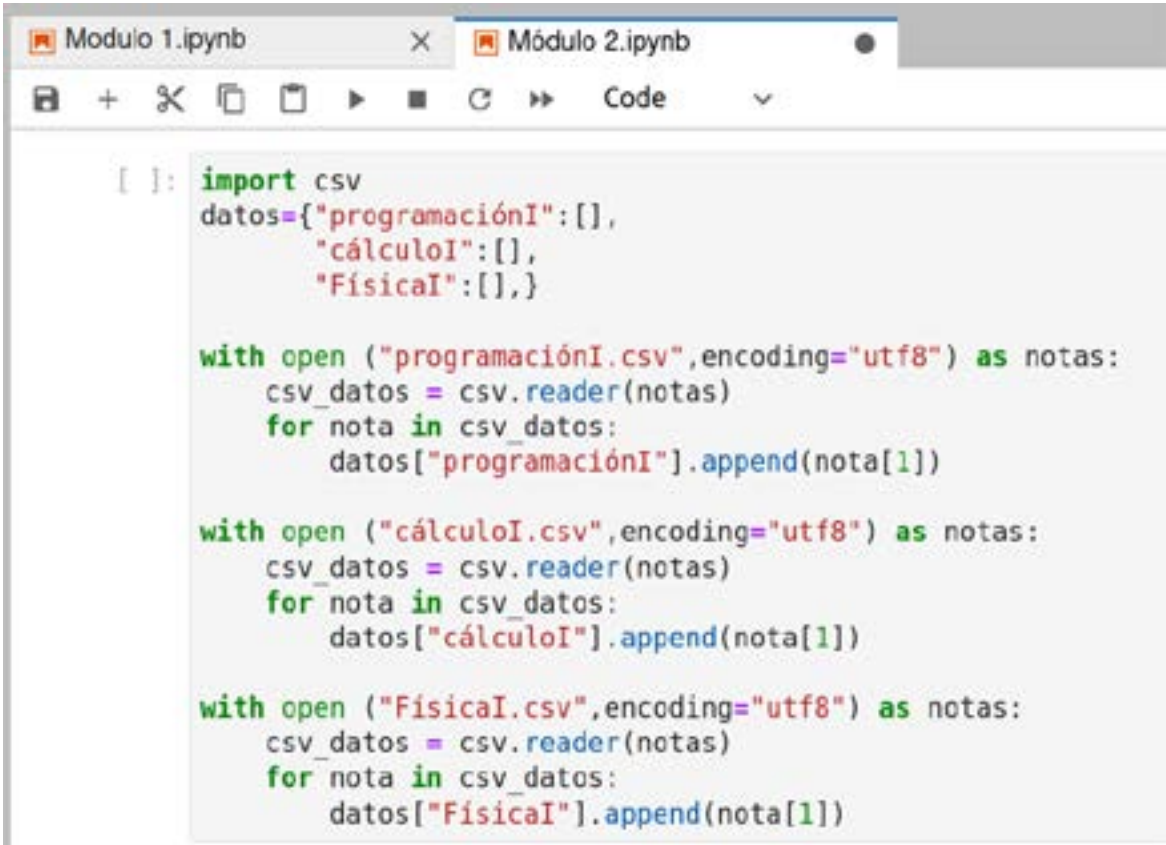
datos={"programaciónI":[],
      "cálculoI":[],
      "FísicaI":[]}

with open ("programaciónI.csv",encoding="utf8") as notas:
    csv_datos = csv.reader(notas)
    for nota in csv_datos:
        datos["programaciónI"].append(nota[1])

with open ("cálculoI.csv",encoding="utf8") as notas:
    csv_datos = csv.reader(notas)
    for nota in csv_datos:
        datos["cálculoI"].append(nota[1])

with open ("FísicaI.csv",encoding="utf8") as notas:
    csv_datos = csv.reader(notas)
    for nota in csv_datos:
        datos["FísicaI"].append(nota[1])
```

Figura 1



```
[ ]: import csv
datos={"programaciónI":[],
      "cálculoI":[],
      "FísicaI":[],}

with open ("programaciónI.csv",encoding="utf8") as notas:
    csv_datos = csv.reader(notas)
    for nota in csv_datos:
        datos["programaciónI"].append(nota[1])

with open ("cálculoI.csv",encoding="utf8") as notas:
    csv_datos = csv.reader(notas)
    for nota in csv_datos:
        datos["cálculoI"].append(nota[1])

with open ("FísicaI.csv",encoding="utf8") as notas:
    csv_datos = csv.reader(notas)
    for nota in csv_datos:
        datos["FísicaI"].append(nota[1])
```

Teniendo este código como ejemplo, ¿De qué manera deberán estar los datos organizados en los archivos csv para que el código de Saray funcione correctamente? Recuerda que el propósito es realizar la carga de todos los datos para posteriormente procesarlos y encontrar los mejores resultados, los mejores promedios.

Asume por ahora que todos los posibles estudiantes se encuentran cursando todas las asignaturas.

¿Recomendarías alguna modificación para el código que está diseñando Saray?

Por ejemplo, convertir el diccionario datos en una lista o un nuevo diccionario que tenga como clave el semestre

```
""" versión de mejora 1 """  
  
import csv  
  
datos=[  
    {"programaciónI":[],  
     "cálculoI":[],  
     "FísicaI":[]}  
]  
  
with open ("programaciónI.csv",encoding="utf8") as notas:  
    csv_datos = csv.reader(notas)  
    for nota in csv_datos:  
        datos[0]["programaciónI"].append(nota[1])  
  
""" versión de mejora 2 """  
  
import csv  
  
datos={"semestreI":  
    {"programaciónI":[],  
     "cálculoI":[],  
     "FísicaI":[]}  
}  
  
with open ("programaciónI.csv",encoding="utf8") as notas:  
    csv_datos = csv.reader(notas)  
    for nota in csv_datos:  
        datos["semestreI"]["programaciónI"].append(nota[1])
```

BIBLIOGRAFÍA

García, J. (2018). Ciencia De Datos. Técnicas Analíticas Y Aprendizaje Estadístico. Un Enfoque Práctico - Jesús García.pdf [6ng22yvvd2lv]. Idoc. pub. Recuperado de <https://idoc.pub/documents/idocpub-6ng22yvvd2lv>.

Python Tutorial. W3schools.com. (2022). Retrieved 31 August 2022, from <https://www.w3schools.com/python/default.asp>.