Archivos e información de la web con `requests`

Fuente: Vivaldi browser (https://vivaldi.com/wp-content/uploads/Quickly-search-for-information-online.png)

El insumo de todo sistema informático son los datos. Transformar los datos (es decir, indicadores cuantitativos o cualitativos) en información (es decir, elementos para la toma de decisiones) es el resultado de procesar los datos or un algoritmo. El corazon de este curso es el análisis de los datos y debemos partir de las fuentes: en este caso, los archivos con información útil y donde encontrarlos: la Web.

Luis A. Muñoz

Tipos de archivos

De distinguen dos tipos de archivos:

- Archivos de texto
- Archivos binarios

Los primeros son una colección de datos con un formato específico: caracteres codificados según un estándar (actualmente, UTF-8 para soportar todos los caracteres internacionales). Los últmos son datos bianrios con un formato propietario. Por ejemplo, el código Python es un archivo de texto, mientras que un documento en Word es un archivo binario.

En los usuarios de Windows suele haber mucha confusión al momento de entender que un archivo de texto es un archivo con información en forma de caracteres de texto y nada más, sin ninguna relación con la extensión que tenga el archivo. Un archivo que tenga la extensión `.txt` no es un archivo de texto: es un archivo que Windows lo intentará abrir con el Bloc de Notas y si es un archivo de texto se abrirá sin inconvenientes.

En Linux, los archivos no tienen extensiones y es más fácil evitar estas confusiones...

Por lo tanto, la diferencia entre ambos tipos de archivos es una cuestión de formato. En Python, esto se especifica según el modo de lectura:

- Archivos de texto: mode='r' (lectura)
- Archivos de datos: mode='rb' (lectura binaria)

Filesystem

El *filesystem* es la organización de los datos en el medio de almacenamiento por parte del sistema operativo. Para trabajar con soltura con archivos y hay manejarse tambien así con el *filesystem*. En Windows, Linux y OS X, los archivos se organizan en un árbol de directorios que cuelgan de un nodo principal o archivo *root* (*C:* en Windows en caso del disco con esta etiqueta, o */.* en el caso de OSX o Linux). Todos los archivos estan ubicados en un sitio especifico dentro del *filesystem* y la forma de especificar esta dirección es con una ruta o *path*.

Por ejemplo, el archivo *proyecto1.py* puede estar alojado en la ruta *C:\Usuarios\elvio\Documentos*

NOTA: Los ejemplos de este documento estan hechos para ser ejecutdos en un equipo con Windows. Si tiene un sistema operativo diferente, reemplace "C:\" por "/"

La librería `os`

Una complicación con los sistemas operativos es que para especificar el path Windows utiliza "\" y Linux y OS X utilizan "/". Afortunadamente, la librería `os` viene a solucionarnos este y otros problemas:

In [2]:
```
import os

os.path.join("Usuarios", "elvio", "Documentos")
```

```
'Usuarios\\elvio\\Documentos'
```

En diferentes SO el resultado será compatible con el sistema (note que Windows utiliza "\\" para especificar que el caracter "\" no es parte de un caracter de escape como "\n"). Esto resulta util para crear rutas sobre los archivos en los que se quiere trabajar:

In [3]:
```
files = ['proyecto1.py', 'proyecto2.py', 'proyecto3.py']

for filename in files:
    print(os.path.join("C:\\Users\\elvio\\Documents", filename))
```

```
C:\Users\elvio\Documents\proyecto1.py
C:\Users\elvio\Documents\proyecto2.py
C:\Users\elvio\Documents\proyecto3.py
```

Otra información útil es saber en que parte del filesystem nos encontramos. Esto es, cual es el directorio actual de trabajo (o *current working directory*) con `getcwd` :

In [111]:
```
ruta_home = os.getcwd()
print(ruta_home)
```

```
C:\Users\lmuno\OneDrive - Universidad Peruana de Ciencias\EL184 - Prog Avanzada de Computadoras
```

Podemos movernos a otra ubicación en el filesystem con `chdir` :

In [18]:
```
os.chdir("C:\\Windows\\System32")
print(os.getcwd())
```

```
C:\Windows\System32
```

Existen dos formas de especificar una ruta: *ruta relativa* y *ruta absoluta*. La primera especifica la ubicación de un archivo respecto a la ubicación actual, mientras que la última especifica la ubicación de un archivo respecto del directorio *root*.

- Ruta relativa: *.\proyecto1.py*
- Ruta absoluta: *C:\Usuarios\elvio\Documentos\proyecto1.py"*

Por otro lado, tambien se tienen los archivos "." y ".." en un directorio. Estas son etiquetas del directorio actual y del directorio padre. Por ejemplo:

In [8]:
```
os.chdir(".")
print(os.getcwd())
```

```
C:\Windows\System32
```

In [9]:
```
os.chdir("..")
print(os.getcwd())
```

```
C:\Windows
```

Creando directorios

Vamos a alejarnos de todo lo que diga "Windows" si no queremos borrar nada... vamos a crear un directorio de prueba en C: con `os.makedirs` . Este método puede crear todos los directorios de una ruta específica. Vamos a utilizar `os.path.exists` para validar si el directorio existe o no antes de crear uno nuevo

In [30]:
```
for i in range(12):
    if not os.path.exists(f"C:\\test\\test{i+1}"):
        os.makedirs(f"C:\\test\\test{i+1}")
```

Ubiquemonos en el directorio `test`

In [31]:
```
os.chdir("C:\\test")
print(os.getcwd())
```

```
C:\test
```

Veamos el contenido del directorio "C:\test":

In [35]:
```
print(os.listdir("."))
```

```
['test1', 'test10', 'test11', 'test12', 'test2', 'test3', 'test4', 'test5', 'test6', 'test7', 'test8', 'test9', 'test_txt.txt']
```

Archivos de texto

La forma más sencilla de crear un archivo de texto es abriendo un archivo con el Context Manager `with` :

In [1]:
```
with open("readme.txt", mode='w', encoding='utf-8') as file:
    file.write("INSTRUCCIONES\n")
    file.write("=============\n\n")
    file.write("En los directorios se tienen diferentes test a ser distribuidos\n")
    file.write("entre los diferentes alumnos")
```

Esto debe de generar un archivo de texto. Haciendo un listado del direcorio se debe de observar el resultado:

In [50]:

```
for item in os.listdir("."):
    # Si la ruta es un direcotrio se muestra el nombre
    if os.path.isdir(item):
        print(item)
    #Si la ruta es un archivo muestra su nombre y tu tamaño en bytes
    elif os.path.isfile(item):
        print(item, "    ", os.path.getsize(item), "bytes")
```

```
readme.txt      125 bytes
test1
test10
test11
test12
test2
test3
test4
test5
test6
test7
test8
test9
```

Para leer un archivo de texto, lo más conveniente es abrir un archivo en un bloque `with` y tratar el archivo como un iterable:

In [51]:

```
with open("readme.txt", mode='r', encoding='utf-8') as file:
    for line in file:
        print(line.strip())    # No olvidar strip() para eliminar "\n"
```

```
INSTRUCCIONES
=============

En los directorios se tienen diferentes test a ser distribuidos
entre los diferentes alumnos
```

Los archivos de texto no resultan ser muy utiles porque no tienen un formato que pueda ser interpretado por una computadora. Es basicamente información legible para una persona, por lo que su uso será la de guardar información simple o tabulada. Sin embargo, si podemos hacer scripts que generen documentos a partir de información. Por ejemplo, generemos unos tests para unos doce alumnos que guardaremos en las diferentes carpetas que hemos creado de forma automática:

In [77]:

```
# GENERADOR DE ARCHIVOS DE PREGUNTAS PARA UN CUESTIONARIO
import random

capitales = {"Amazonas": "Chachapoyas",
             "Áncash": "Huaraz",
             "Apurímac": "Abancay",
             "Arequipa": "Arequipa",
             "Ayacucho": "Ayacucho",
             "Cajamarca": "Cajamarca",
             "Cusco": "Cusco",
             "Huancavelica": "Huancavelica",
             "Huánuco": "Huánuco",
             "Ica": "Ica",
             "Junín": "Huancayo",
             "La Libertad": "Trujillo",
             "Lambayeque": "Chiclayo",
             "Lima": "Lima",
             "Loreto": "Iquitos",
             "Madre de Dios": "Puerto Maldonado",
             "Moquegua": "Moquegua",
             "Pasco": "Cerro de Pasco",
             "Piura": "Piura",
             "Puno": "Puno",
             "San Martín": "Moyobamba",
             "Tacna": "Tacna",
```

```
            "Tumbes": "Tumbes",
            "Ucayali": "Pucallpa"}

# Se generan los 12 archivos de examenes
for quizNum in range(1, 13):
    # Se generan los nombres de los arvhivos de test en cada directorio y el de respuestas correctas
    quizFileName = os.path.join(f"C:\\test\\test{quizNum}", f"text{quizNum}.txt")
    answerKeyFileName = os.path.join(f"C:\\test\\test{quizNum}", f"test{quizNum}_answers.txt")

    # Se abren ambos archivos para su generación
    with open(quizFileName, mode='w', encoding='utf-8') as quizFile:
        with open(answerKeyFileName, mode='w', encoding='utf-8') as answerKeyFile:
            # Se crea un encabezado en el archivo de test
            quizFile.write("Nombre:\n\nFecha:\n\nPeriodo:\n\n")
            quizFile.write(' ' * 20 + f"Capitales por departamento - Test {quizNum}\n\n")

            # Se obtiene una lista con las capitales por departamento
            # y se desordena la muestra
            departamentos = list(capitales.keys())
            random.shuffle(departamentos)

            # Para cada uno de los departamentos...
            for idx, departamento in enumerate(departamentos, start=1):
                # Se guarda la respuesta correcta (capital del departamento)
                correctAnswer = capitales.get(departamento)
                # Se guarda una lista con todas las capitales y se elimina la correcta
                wrongAnswers = list(capitales.values())
                del wrongAnswers[wrongAnswers.index(correctAnswer)]
                # Se obtiene una muestra aleatoria de 3 respuestas aleatorias + la correcta
                wrongAnswers = random.sample(wrongAnswers, 3)
                answerOptions = wrongAnswers + [correctAnswer]
                # Se desordenan las opciones de respuesta
                random.shuffle(answerOptions)

                # Escribimos la pregunta en el test
                quizFile.write(f"{idx}. ¿Cuál es la capital de {departamento}?:\n")

                for letter, answer in zip('ABCD', answerOptions):
                    quizFile.write(f"\t{letter}. {answer}\n")
                else:
                    quizFile.write('\n')

                # Escribirmos la respuesta en el archivo de respuestas
                answerKeyFile.write(f"{idx}. {'ABCD'[answerOptions.index(correctAnswer)]}\n")
```

Tip: librería `shelve`

Si lo que se quiere es guardar las variables de una programa de forma rápida existe una forma de hacerlo sin tener que generar archivos con todas las complejidades de saber la ruta y el formato: la librería `shelve` .

In [1]:

```
import shelve

# Guardando datos en un archivo shelve
datos = [1, 2, 3, 4, 5]

with shelve.open('mis_datos') as shelve_file:
    shelve_file['datos'] = datos
```

Para guardar datos utilizando `shelve` se abre un archivo al que llamaremos `mis_datos` en un bloque `with` con la instrucción `shelve.open` . Una vez abierto, se trata este archivo como un diccionario, donde la llave será un `str` con el nombre de la variable a almacenar y el valor será la variable misma. Si revisa le ruta actual, vera que hay unos archivos con el nombre `mis_datos` con extensión `bak` , `dat` y `dir` . Intente abrirlos con un editor... no va a encontrar registros de la información. Pero podemos traerla de vuelta.

In [90]:
```
# Borremos la variable datos...
del datos

# ...y la recuperamos del archivo
with shelve.open("mis_datos") as shelve_file:
    datos = shelve_file['datos']

print(datos)
```

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

¡Facil, sencillo y seguro!

Archivos CSV

Un archivo CSV es un archivo de texto con un formato estándar. En este, los valores se guardan como valores de texto, separados por algun separador, usualmente una coma (",") de donde viene el nombre Comma Separated Value, aunque puede ser un espacio en blanco, un tabulador ("\t") o un punto y coma (";").

Los archivos CSV son reconocidos por las Hojas de Cálculo como Excel y ordena los datos por columnas utilizando el caracter de separación. Hay que tomar en consideración que en los países donde se utiliza la "," como separador de miles, se debe de utilizar el ";" para un archivo CSV que sea reconocido por Excel. Al final, un archivo CSV es una Hoja de Cálculo simplificada, sin pestañas ni fórmulas.

Hay otro detalle a considerar: un CSV no se puede manipular como un archivo de texto al que lo podemos seprar utilizando `split(',')` ya que no todas las comas son separadores. Un CSV también tiene sus propios caracteres de escape, lo que permite que una coma pueda ser parte de los valores (como en el caso de un número escrito con la forma 1,200). Esa es la razón por la que siempre hay que usar la librería `csv` .

Los marcadores de personal suelen generar archivos CSV diarios. Vamos a generar una simulación de esto:

In [117]:
```
empleados = [["6/6/2020 07:20", "Elvio Lado"],
             ["6/6/2020 07:22", "Elmer Curio"],
             ["6/6/2020 07:30", "Elba Lazo"],
             ["6/6/2020 07:36", "Susana Oria"],
             ["6/6/2020 07:49", "Armando Paredes"]]

# Regresemos a la ruta de este documento
os.chdir(ruta_home)
```

Para almacenar estos datos como un archivo CSV en Windows (y solo en Windows) hay un recordar establecer el parametro `newline=''` para evitar que se generen líneas en blanco entre los registros (esto por razones técnicas que estan detalladas aqui (https://docs.python.org/3/library/csv.html#id3)).

In [115]:
```
import csv

with open("entrada.csv", mode='w', newline='') as csv_file:
    writer = csv.writer(csv_file, delimiter=';')
    writer.writerow(["HORA", "EMPLEADO"])

    for registro in empleados:
        writer.writerow(registro)

print("Archivo generado:", os.path.abspath("entrada.csv"))
```

```
Archivo generado: C:\Users\lmuno\OneDrive - Universidad Peruana de Ciencias\EL184 - Prog Avanzada de Computadoras\entrada.csv
```

En el código anterior hay algunos detalles a considerar:

- Se utiliza el parametro `newline=''` por ser Windows. En otro sistema operativo esta opción no se coloca
- Se establece un objeto `csv.writer` sobre el archivo abierto para escribir sobre este.
- En el writer se define el tipo de separador como ";" para que sea compatible con Excel (por defecto es ",")
- Se esta utilizando el método `writerow(registro)` para escribir los registros. También se pudo haber llamado al método `writerows(empleados)`
- Se escribe una lista con los nombres de las columnas de los datos. Esto es el encabezado
- Se esta obteniendo la ruta absoluta del archivo generado con `os.path.abspath` para saber donde esta ubicado el archivo generado

Si todo esta bien, podrá abrir el archivo desde Excel.

JupyterLab tiene un visor de archivos CSV. Pruebe abrir el archivo desde el navegador de archivos a la izquierda para ver la información tabulada y podrá seleccionar el tipo de delimitador.

Ahora, leamos el archivo:

In [116]:
```
import csv

with open("C:\\test\\entrada.csv") as file:
    reader = csv.reader(file, delimiter=';')
    next(reader)      # Con esto pasamos a la siguiente linea: eliminamos el encabezado

    for line in reader:
        print(f"* Nombre: {line[1]:20} Hora de ingreso: {line[0]}")
```

```
* Nombre: Elmer Curio          Hora de ingreso: 6/6/2020 07:22
* Nombre: Elba Lazo            Hora de ingreso: 6/6/2020 07:30
* Nombre: Susana Oria          Hora de ingreso: 6/6/2020 07:36
* Nombre: Armando Paredes      Hora de ingreso: 6/6/2020 07:49
```

Detalles a considerar del código anterior:

- No es necesario especificar `newline=''` . Esto es solo para escirbir un archivo CSV
- No se especifica el modo al momento de abrir el archivo. El modo por defecto es lectura
- Se especifica un `csv.reader` para retornar una lista de datos a partir de cada línea de texto
- Se especifica el tipo de separador en el reader.
- Se estan utilizando los indices de la lista (en este caso, `line`) para mostrar los resultados.

Archivos JSON

Un archivo JSON (nadie sabe a ciencia cierta como se pronuncia eso pero esta generalizado utilizar "Jason") es un archivo con un formato más complejo. Es, al final, un diccionario almacenado. Para gestionar este tipo de archivos utilizaremos el módulo `json` .

Ampliemos el caso del registro de entrada con más datos, esta vez estructurados como un diccionario:

In [3]:
```
empleados = [{"ingreso": "6/6/2020 07:20", "salida": "6/6/2020 19:30", "nombre": "Elvio Lado"},
             {"ingreso": "6/6/2020 07:22", "salida": "6/6/2020 18:50", "nombre": "Elmer Curio"},
             {"ingreso": "6/6/2020 07:30", "salida": "6/6/2020 19:10", "nombre": "Elba Lazo"},
             {"ingreso": "6/6/2020 07:36", "salida": "6/6/2020 20:10", "nombre": "Susana Oria"},
             {"ingreso": "6/6/2020 07:49", "salida": "6/6/2020 17:59", "nombre": "Armando Paredes"}]
```

Esta vez tenemos una lista de registros, donde cada registro es un diccionario. Esta estructura la podemos almacenar tal cual en formato JSON. Para volcar los datos a un archivo JSON se utiliza el método `json.dump(data, file)` :

In [4]:
```
import json

with open("marca_dia.json", mode='w') as json_file:
    json.dump(empleados, json_file)
```

Si se abre un archvio JSON en un Bloc de Notas verá una lista con diccionarios escrita como un texto. Pero un visualizador de archvios JSON mostrará un resultado diferente. Pruebe abriendo el archivo en JupyterLab y verá la estructura de la información.

La lectura de un archivo JSON utiliza el método `json.load(file)` :

In [5]:
```
with open("marca_dia.json") as json_file:
    data = json.load(json_file)

print(data)
```

```
[{'ingreso': '6/6/2020 07:20', 'salida': '6/6/2020 19:30', 'nombre': 'Elvio Lado'}, {'ingreso': '6/6/2020 07:22', 'salida': '6/6/2020 18:5
0', 'nombre': 'Elmer Curio'}, {'ingreso': '6/6/2020 07:30', 'salida': '6/6/2020 19:10', 'nombre': 'Elba Lazo'}, {'ingreso': '6/6/2020 07:3
6', 'salida': '6/6/2020 20:10', 'nombre': 'Susana Oria'}, {'ingreso': '6/6/2020 07:49', 'salida': '6/6/2020 17:59', 'nombre': 'Armando Par
edes'}]
```

Tip: Como imprimir un diccionario bien

El resultado anterior es, por decir algo, bastante feo. Hay una forma de imprimir un diccionario de forma estética valiéndose del módulo `json` , en este caso del método `json.dumps(data)` (la 's' en dump*s* es por 'string'), que toma un diccionario (o una estructura JSON que viene a ser lo mismo) y hace un volcado sobre un string que puede tener un fomato:

In [124]:
```
print(json.dumps(data, indent=4))    # indent es el número de espacios para la sangría de niveles
```

```
[
    {
        "ingreso": "6/6/2020 07:20",
        "salida": "6/6/2020 19:30",
        "nombre": "Elvio Lado"
    },
    {
        "ingreso": "6/6/2020 07:22",
        "salida": "6/6/2020 18:50",
        "nombre": "Elmer Curio"
    },
    {
        "ingreso": "6/6/2020 07:30",
        "salida": "6/6/2020 19:10",
        "nombre": "Elba Lazo"
    },
    {
        "ingreso": "6/6/2020 07:36",
        "salida": "6/6/2020 20:10",
        "nombre": "Susana Oria"
    },
    {
        "ingreso": "6/6/2020 07:49",
        "salida": "6/6/2020 17:59",
        "nombre": "Armando Paredes"
    }
]
```

Armando Paredes en quien llega último y se va primero...

`requests` o como obtener información de la web (JSON)

Python tiene una librería en la Biblioteca Estándar llamada `urllib` que pemite hacer consultas html que es mejor olvidar que existe. Es de esas cosas que se colocan debajo de la alformbra. Kenneth Reitz le hizo un favor a la comunidad de Python desarrollando la librería `requests` que es hoy la forma más sencilla de realizar requerimientos a un recursos web.

Esta libreria debe de instalarse por medio del gestor de paquetes pip, que descarga e instala packetes disponibles en el Python Packages Index (PyPI). Se puede utilizar el caracter "!" en una celda en un documento Jupyter para ejecutar comandos de consola.

In [126]:
```
!pip install requests
```

```
Requirement already satisfied: requests in c:\users\lmuno\anaconda3\lib\site-packages (2.24.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\lmuno\anaconda3\lib\site-packages (from requests) (2020.6.20)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\lmuno\anaconda3\lib\site-packages (from requests) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users\lmuno\anaconda3\lib\site-packages (from requests) (1.2
5.9)
Requirement already satisfied: idna<3,>=2.5 in c:\users\lmuno\anaconda3\lib\site-packages (from requests) (2.10)
```

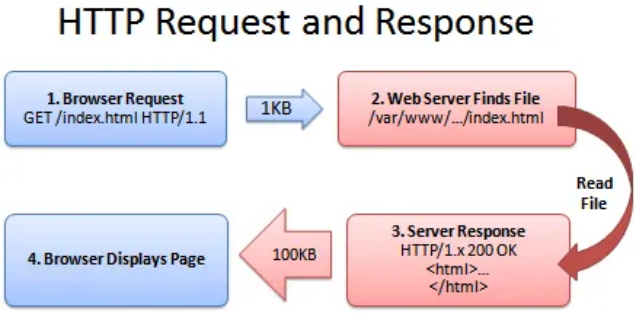
Utilicemos las instrucciones de prueba de la página oficinal en github del proyecto (https://github.com/psf/requests):

In [127]:
```
import requests

r = requests.get('https://api.github.com/repos/psf/requests')
r.json()["description"]
```

```
'A simple, yet elegant HTTP library.'
```

Para entender lo que sucede en estas líneas de código, hay que tener una idea del modelo HTTP y en concreto, que cosa es un request:



El protocolo que hace que la Web funcione es HTTP. En este se define que existen dos nodos: cliente y servidor. El cliente hace un pedido de información (un "requests" con una ruta) y el servidor retorna un código de respuesta (un "response") así como la información requerida (si la información existe, para lo que responderá con el código 200; en caso contrario responderá con un código 404). En el código anterior, todos los detalles de señalización y flujo de control de información estan resueltos en la librería `requests` .

El método `get` es la instrucción HTTP que se utiliza para hacer una petición en un mensaje que es formateado por la librería. En este caso, el requerimiento apunta a un archivo JSON y `requests` puede gestionar archivos JSON de forma nativa con el método `json` sobre la respuesta del requerimiento (el objeto `r`). En este caso, estamos viendo el valor de la llave `description` .

Veamos una estructura más compleja: La URL https://deperu.com/api/rest/noticias.json (https://deperu.com/api/rest/noticias.json) apunta a un servicio que retorna noticias en formato JSON. ¿Cómo podemos obtener un listado de los titulares? Guardemos este información en un archivo JSON para poder verlo en el visualizador:

In [137]:
```
url = "https://deperu.com/api/rest/noticias.json"
r = requests.get(url)
data = r.json()

with open("noticias.json", mode='w') as json_file:
    json.dump(data, json_file)
```

Revisando la estructura se observa que es una lista de diccionarios, donde las llaves 'fecha', 'titulo' y 'url' nos dan la información que queremos para presentar la información:

In [138]:

```
with open("noticias.json") as json_file:
    data_noticias = json.load(json_file)

print("LAS NOTICIAS DE LA HORA")
print("======================\n")
for noticia in data_noticias[:5]:           # [:5] => Las cinco noticias mas recientes
    print("  Fecha:", noticia['fecha'])
    print("  Titular:", noticia['titulo'])
    print("  Referencia:", noticia['url'])
    print()
```

```
LAS NOTICIAS DE LA HORA
======================

  Fecha: Mon, 20 Jul 2020 17:05:01 -0500
  Titular: Acreedores rechazan oferta de Argentina y hacen peligrar el canje de deuda
  Referencia: https://www.deperu.com/noticias/acreedores-rechazan-oferta-de-argentina-y-hacen-peligrar-el-canje-de-deuda-105214.html

  Fecha: Mon, 20 Jul 2020 16:59:57 -0500
  Titular: Los niños negros tienden a morir más que los blancos tras una operación en EEUU
  Referencia: https://www.deperu.com/noticias/los-ninos-negros-tienden-a-morir-mas-que-los-blancos-tras-una-operacion-en-eeuu-105213.html

  Fecha: Mon, 20 Jul 2020 15:14:03 -0500
  Titular: El dólar vuelve al supermercado en Cuba para impulsar la economía
  Referencia: https://www.deperu.com/noticias/el-dolar-vuelve-al-supermercado-en-cuba-para-impulsar-la-economia-105209.html

  Fecha: Mon, 20 Jul 2020 14:32:33 -0500
  Titular: Brasil inicia el martes fase final de pruebas de vacuna china contra el nuevo coronavirus
  Referencia: https://www.deperu.com/noticias/brasil-inicia-el-martes-fase-final-de-pruebas-de-vacuna-china-contra-el-nuevo-coronavirus-105211.html

  Fecha: Mon, 20 Jul 2020 14:05:46 -0500
  Titular: Piden donantes de plasma en Florida cuando escasean tratamientos y se saturan hospitales
  Referencia: https://www.deperu.com/noticias/piden-donantes-de-plasma-en-florida-cuando-escasean-tratamientos-y-se-saturan-hospitales-105207.html
```

¡Hemos construido un *newsletter* con muy pocas lineas de código!

In [ ]: