INF-253 Lenguajes de Programación
Tarea 1: Python

Profesor: José Luis Martí Lara
Ayudante Cátedras: Lucio Fondón Rebolledo
Ayudante Tareas: Gabriel Carmona Tabja, Sebastián Campos Muñoz

7 de abril de 2021

1. Pido y entrego

Debido a la pandemia las personas deben quedarse en casa para evitar contagiarse de COVID-19, dentro de este contexto el desarrollo de aplicaciones a crecido exponencialmente. Por esta situación, en un pequeñísimo pueblo llamado San Vicente de Tagua Tagua, un empresario llamado Don Fondon concibió una idea de una app, pero al no saber programar porque no pasó el curso de programación y se fue a la camita, le pide ayuda a ustedes para que le realicen un programa que reciba peticiones ajax y las ejecuten. El futuro del pequeño emprendedor está en sus manos.

Se debe utilizar Python 3 y la librería RegEx para las expresiones regulares de la tarea, en caso de que alguna de éstas dos condiciones no se cumpla no se revisará la tarea.

2. Hoy presentamos: Ajax

2.1. Acerca de Ajax.

Ajax es una herramienta que permite realizar consultas HTTP: **POST**, **GET** y **DELETE**. Estas consultas permiten manejar datos en función de lo que necesite el usuario. Para motivos de ésta tarea, interpretaremos las consultas HTTP de la siguiente forma:

- **POST**: recibe parámetros y almacena los datos dentro de un archivo especificado.
- **GET**: recibe cero o más parámetros y la consulta responde con la información que calce con estos parámetros. Esta información se obtiene de un archivo especificado.
- **DELETE**: recibe cero o más parámetros y la consulta elimina todos la información que calce con estos parámetros. Esta información se borra de un archivo especificado.

El formato de una consulta Ajax tiene una forma de JSON. Esta estructura primero contiene un diccionario con la información de los parámetros, esta información está conformada por el Nombre del parámetro: Valor del parámetro. Además, la consulta Ajax contiene el tipo de consulta HTTP y el archivo relacionado a la consulta:

Code 1: Ejemplo

```
{
    data: {"parametro1": valor1, "parametro2": valor2},
    type: "GET",
    url: "archivo.txt"
}
```

DATOS importantes

Cómo podemos notar en el ejemplo, la consulta abre y cierra con llaves, luego viene el campo llamado **data** el cual posee los parámetros de la consulta. El nombre de los parámetros son strings y el valor de los parámetros se definirá más adelante, notar que cada parámetro : valor está separado por comas (exceptuando el último). Después del campo data, está el campo **type** el cual señala en un string el tipo de consulta y el campo **url** el cual también señala en un string el archivo relacionado a la consulta.

Para motivo de ésta tarea, la estructura de la consulta Ajax será siempre de la misma forma: una llave, un salto de línea, un tab, el campo data (junto con su valor), una coma, un salto de línea, un tab, el campo **type** (junto con su valor), una coma, un salto de línea, un tab, el campo **url** (junto con su valor), un salto de línea y la llave de cierre.

3. Consultas

3.1. POST

La consulta **POST** consiste en recibir una data y almacenarla al final del archivo objetivo. Esta data tiene parámetros donde estos pueden tener múltiples valores:

- números enteros positivos
- strings
- lista de números enteros positivos
- otro diccionario el cual puede contener todo lo de arriba y/u otro diccionario

Por ejemplo:

Code 2: Ejemplo de POST

```
{
    data: {"objeto": {"numero": 1, "nombre": "Mario Mario"}, "lista": [1, 3, 4, 6]},
    type: "POST",
    url: "ejemplo.txt"
}
```

Donde al ejecutar esto, se almacenará de la siguiente forma:

Code 3: Ejemplo de guardado de un POST

```
{"objeto": {"numero": 1, "nombre": "Mario Mario"}, "lista": [1, 3, 4, 6]}
```

3.2. GET

La consulta **GET** consiste en la obtención de datos de un archivo en específico que cumplan ciertas características. Por ejemplo, asumamos que un archivo tiene los siguientes datos:

Code 4: Ejemplo de información

```
{"nombre": "jaja", "objetorandom": {"nombre": "Tomc", "numerocualquiera": 2}}
{"nombre": "jaja", "lista": [1, 3, 7, 4, 6], "cosa": 3}
{"hacker": "jaja", "no": 3}
{"solito": 1}
{"objeto": {"numero": 1, "noNombre": "Mario Mario"}, "lista": [1, 3, 4, 6]}
{"notanlista": 2, "nombre": "jaja"}
```

Y se realiza la siguiente consulta **GET**:

Code 5: Ejemplo de GET

```
{
    data: {"nombre": "jaja"},
    type: "GET",
    url: "ejemplo.txt"
}
```

debe ser tal cual

Bajo éste contexto, los datos que se deberían obtener son los siguientes:

Code 6: Ejemplo de GET

```
{"nombre": "jaja", "objetorandom": {"nombre": "Tomc", "numerocualquiera": 2}}
{"nombre": "jaja", "lista": [1, 3, 7, 4, 6], "cosa": 3}
{"notanlista": 2, "nombre": "jaja"}
```

Otro ejemplo de consulta **GET** es el siguiente:

Code 7: Ejemplo de GET 2

```
{
    data: {"nombre": "jaja", "cosa": 3},
    type: "GET",
    url: "ejemplo.txt"
}
```

Con ésta consulta, el resultado a obtener debería ser el siguiente:

Code 8: Ejemplo de GET

```
{"nombre": "jaja", "lista": [1, 3, 7, 4, 6], "cosa": 3}
```

3.3. DELETE

La consulta **DELETE** consiste en recibir una data y un archivo objetivo, en el cual todos los registros que concuerdan con ciertas de las características de la data serán borrados.

Por ejemplo, si existe un archivo llamado **ejemplo.txt** y contiene la siguiente información.

*Eliminar lineas

Code 9: Ejemplo de archivo: ejemplo.txt

```
{"nombre": "jaja", "objetorandom": {"nombre": "Tomc", "numerocualquiera": 2}}
{"nombre": "jaja", "lista": [1, 3, 7, 4, 6], "cosa": 3}
{"hacker": "jaja", "no": 3}
{"solito": 1}
{"objeto": {"numero": 1, "noNombre": "Mario Mario"}, "lista": [1, 3, 4, 6]}
{"notanlista": 2, "nombre": "jaja"}
```

reconoce igual que GET, pero actua sobre el archivo.

Y se realiza la siguiente consulta:

Code 10: Ejemplo de DELETE

```
{
    data: {"nombre": "jaja"},
    type: "DELETE",
    url: "ejemplo.txt"
}
```

El archivo **ejemplo.txt** quedaría con el siguiente contenido:

Code 11: Ejemplo de archivo: ejemplo.txt

```
{"hacker": "jaja", "no": 3}
{"solito": 1}
{"objeto": {"numero": 1, "noNombre": "Mario Mario"}, "lista": [1, 3, 4, 6]}
```

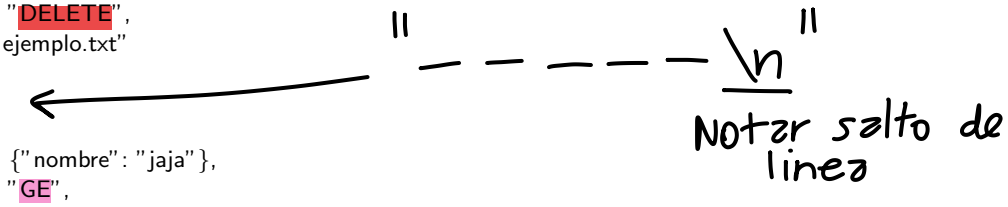
4. Su objetivo

Deben generar un programa el cual permita recibir una cantidad indefinida de consultas en un archivo llamado entrada.txt separadas por cinco guiones consecutivos y ejecutarlas adecuadamente. La respuesta de la consulta **GET** debe mostrar por pantalla los datos encontrados. Y en el caso del resto de las consultas deben mostrar un mensaje por consola el cual indique si la consulta se realizó con éxito o no. En el caso que la consulta sea incorrecta debe mostrar por pantalla que esta consulta es invalida y seguir con el programa.

Por ejemplo:

Code 12: Ejemplo de input

```
{
    data: {"nombre": "jaja"},
    type: "DELETE",
    url: "ejemplo.txt"
}
-----
{
    data: {"nombre": "jaja"},
    type: "GE",
    url: "ejemplo.txt"
}
-----
{
    data: {"nombre": "jaja"},
    type: "POST",
    url: "ejemplo.txt"
}
-----
{
    data: {"nombre": "jaja"},
    type: "GET",
    url: "ejemplo.txt"
}
```



En este caso, la segunda estaría mal escrita debido a que el type no pertenece a ninguno de los types definidos, dando como resultado estos mensajes mostrados por pantalla:

Code 13: Ejemplo de output

```
Consulta DELETE realizada!
Consulta mal escrita!
Consulta POST realizada!
{"nombre": "jaja"}
```

Para toda la detección de las consultas DEBE utilizar expresiones regulares, si se descubre que no se utilizaron expresiones regulares no se revisará la tarea

5. Datos de vital importancia

- Esta es una versión simplificada de las consultas Ajax realizada para ésta tarea, por lo que si ya conoce las consultas Ajax mantenga la estructura definida en este documento.
- Si el **GET** recibe una data vacía, o sea {}, entrega todo lo que este en el archivo.
- Si el data esta vació en un **DELETE**, o sea {}, entonces deberán borrar todo lo que hay DENTRO del archivo.
- Notar que tanto parámetro como valor deben ser iguales para poder borrarlos u obtenerlos.
- Se pueden realizar asunciones respecto a cosas que no estén escritas literalmente en la tarea que no rompan el sentido de ésta. (Si tiene dudas, puede preguntar a cualquiera de los dos ayudantes)
- Los 5 guiones para separar consultas SIEMPRE estarán bien escritos y SIEMPRE existirán.

6. Sobre Entrega

- Se debera entregar un programa llamado ajax.py.
- Las funciones implementadas deben ser comentadas de la siguiente forma. **SE HARÁN DESCUENTOS POR FUNCIÓN NO COMENTADA**

  ”’
  Nombre de la función

  Parametro 1 : Tipo
  Parametro 2 : Tipo
  Parametro 3 : Tipo
  .....

  Breve descripción de lo que realiza la función y lo que retorna ”’
- Se debe trabajar de forma individual obligatoriamente.
- **La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea1LP_RolIntegrante-1.tar.gz**
- **El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la correcta utilización de su programa.**
- **El no cumplir con las reglas de entrega conllevará un descuento máximo de 30 puntos en su tarea.**
- La entrega será vía aula y el plazo máximo de entrega es hasta el **Viernes 23 de abril a las 23:55 hora aula**.
- Por cada día de atraso se descontarán 20 puntos (10 puntos dentro la primera hora).
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.

7. Calificación

7.1. Entrega

- Orden (5 puntos)
- Uso correcto de expresiones regulares (25 puntos)
- Detección de errores (20 puntos)
- Ejecución de consultas:
  - POST (16 puntos)
  - GET (17 puntos)
  - DELETE (17 puntos)

7.2. Descuentos

- Falta de comentarios (-10 puntos c/u MAX 30)
- Falta de README (-20 puntos)
- No respeta formato reglas de entrega (-30 puntos)