



Proyecto 01

Semestre 2021-1

Prof. José Galaviz Casas

Ayud. María Ximena Lezama

Modelado y programación

Kevin Ariel Merino Peña¹ Armando Abraham Aquino Chapa²

11 de octubre de 2020



1. Definición del problema

Un aspecto indispensable antes de comenzar a programar es entender el problema a través de todos los datos que se nos fueron proporcionados. Esto lo logramos gracias a las siguientes preguntas:

- ¿Qué es lo que queremos obtener?: Tenemos que obtener el clima de la ciudad de salida y la ciudad de llegada dado ciertos tickets de avión que nos puso a disposición el aeropuerto de la Ciudad de México.
- ¿Qué datos tenemos para obtenerlo?: Cómo se mencionó anteriormente, tenemos una cantidad fija de tickets, todos ellos agrupados en dos archivos *.csv* (diferentes entre sí) que contienen información que puede ser de utilidad e información que aparentemente puede estar de más.
- ¿Los datos son suficientes? Para el objetivo del proyecto, que es solamente obtener el clima, en algunos casos sí son suficientes. Pero es importante recalcar que habría que hacer adecuaciones para que todos los datos proporcionados funcionen
- ¿Qué operaciones o construcciones se deben obtener para llegar a la solución?: Debemos buscar la manera de normalizar la información proporcionada, luego encontrar algún recurso que nos permita consultar el clima con la información con la que contamos, finalmente construir una forma de presentar los datos.

2. Análisis del problema y selección de la mejor alternativa.

Una vez definido el proyecto, veamos como fue nuestro proceso para el análisis del problema. Para ello podemos dividir el análisis en los siguientes subproblemas/aspectos:

1. *Web Services*:

- Notemos que para saber el clima de los vuelos que se nos fueron proporcionados, tendríamos que usar *Web Services* para poder consultar el clima. Nuestra elección fue: *OpenWeatherMap*.
- Al no tener conocimientos sobre *OpenWeatherMap* tuvimos que documentarnos para aprender como realizar peticiones y obtener la información necesaria.
- Tendríamos que solventar errores con respecto al exceso de peticiones por un cierto tiempo para no tener ningún tipo de problema con la API
- Relacionar la información que tenemos disponible en los archivos *.csv* como las coordenadas, nombre de las ciudades, etc; con la API, para así obtener el clima.

2. Archivos *.csv*

- Leer estos archivos para poder solicitar la peticiones a la API.
- Descartar la información redundante que no es de utilidad para nuestro objetivo.
- Para generar un programa eficiente, nos percatamos que hay muchos vuelos que se repiten, por lo que tendríamos que preprocesar la información de estos vuelos repetidos, es decir, que no haya información repetida. De esta forma también podríamos ahorrarnos mucho tiempo de ejecución, ya que en *OpenWeatherMap* sólo podemos realizar un cierto número de peticiones por minuto.

3. Manejar errores generales, y otros aspectos particulares:

- Modelar un caché que nos permitiría agilizar nuestro problema y así no realizar peticiones redundantes.

¹Número de cuenta 317031326

²Número de cuenta 317058163

- Personalmente, al no tener un conocimiento avanzado en Python, tener que informarme sobre sintaxis, diccionarios y otros aspectos relacionados a este lenguaje de programación.
- Conocer que estructuras de datos nos conviene más utilizar.
- Dar un formato correcto y legible a la salida del programa.
- Verificar la consistencia de los archivos que nos dan como parámetro, para así poder validarlos si cumplen con la información necesaria, o rechazarlos y mandar una advertencia de que no cumplen con nuestros requerimientos.

Lenguaje de programación

Seleccionamos como lenguaje de programación a Python, y lo hemos elegido debido a que de manera muy breve y concisa se pueden hacer proyectos tan sólidos y robustos como sea necesario. Personalmente hemos tomado la iniciativa de usar este lenguaje porque muchos de los consejos de profesoras, profesores y colegas de la carrera es aprender este lenguaje de programación porque buena parte de la industria trabaja con él.

Caché

Para modelar esta funcionalidad empleamos una librería llamada *functools*, de donde extraímos *lru_cache* para crear un caché de las peticiones hechas, usamos esta herramienta porque no tiene un límite de tiempo como sí ocurre en otras librerías y es muy fácil de usar.

Preprocesamiento de datos

Pensamos que sería más fácil leer el archivo csv y separar las coordenadas repetidas, para que sólo hagamos las peticiones necesarias y no más. Decidimos emplear un principio que estamos aprendiendo a usar "*Pide el máximo y ofrece el mínimo*", bajo este principio es que decidimos sólo trabajar con un tipo de archivo csv muy acotado, con las columnas con tal nombre y el tal orden.

Nos pareció importante hacerlo así para poder venderle más "mejoras" a nuestra clientela.

Una vez visto como analizamos el problema principal en subproblemas, hemos determinado que lo más adecuado para resolverlo era obtener las siguientes clases:

- **Clase Weather:**
 - Esta clase sería la encargada de realizar las distintas peticiones al servidor para obtener todos los datos correspondientes al clima. Aquí también modelaríamos nuestro caché.
 - Manejar los posibles errores, como el exceso de número de peticiones en un lapso de tiempo, para no obtener ningún problema con el servidor, y manejar los errores donde no es posible consultar la información
 - Otorgar de un formato correcto y legible a la salida del programa
- **Clase CSVReader**
 - Su función principal sería leer los archivos *csv* otorgados por el aeropuerto de la Ciudad de México.
 - Preprocesar cierto tipo de información que si es esencial de la que es redundante o no tiene utilidad para nuestros objetivos.
 - Ordenar información (de los archivos *csv* que se nos fueron proveídos) para que de esta forma no obtengamos elementos repetidos, y así evitar realizar peticiones de más.
- **Clase main.**
 - Cómo su nombre lo indica, en esta clase se concretarían las peticiones y formatos de respuestas a nuestros clientes.
 - Procesar las peticiones no repetidas para posteriormente enviarlas al servidor.
 - Asimismo, verificaríamos la consistencia del archivo enviado como parámetro para evitar cualquier tipo de inconveniente y que nuestro programa no colapse.

Por último respondamos la siguiente pregunta: ¿La forma en qué hemos modelado el problema es la mejor alternativa?. Si, es una de las mejores alternativas, ya que pueden asegurar que el problema se resuelve de una forma: eficiente, es decir, sin consumir la mínima cantidad de recursos, pero tampoco abusando de ellos. El programa es amigable con el usuario, porque a pesar de no ser interactivo, si tiene un muy buen formato el cual permite que el contenido sea legible y la información esté ordenada correctamente. También, el programa es tolerable a fallos. Aunque no es totalmente robusto, si tomamos en cuenta los errores más comunes y naturales a la hora de manejar el programa. Más adelante mencionaremos cuales son los planes a futuro que tenemos para este proyecto, ya sea para una mejora o para mantenimiento.

3. Pseudocódigo

CSVReader.py

Función 1: read_no_repeated_coordinates

Entrada: Nombre de un archivo (ruta)

Salida : Lista de coordenadas no repetidas

```

1 while Archivo(nombre dado) esté abierto do
2   foreach renglones ← documento do
3     if (latitud , longitud ) no están en la lista then
4       Agreagrlas

```

Función 2: read_csv_file

Entrada: Nombre de un archivo (ruta)

Salida : Lista de diccionarios con vuelos

```

1 try:
2   Abrir ruta
3   for linea ← archivo do
4     linea ← lista
5 catch FileNotFoundError:
6   muestra Error, escribe una ruta válida
7   exit
8 catch FileExistsError:
9   muestra Error, archivo válido
10  exit

```

Función 3: read_headers

Entrada: Nombre de un archivo (ruta)

Salida : Lista de cabeceras

```

1 with:
2   lector ← Leer primera linea( ruta)

```

Función 4: validate_file

Entrada: Nombre de un archivo (ruta) pasados como argumento al programa

```
1 if longitud del argumento no es 2 then
2   muestra: Error
3   Debe indicar la ruta a un archivo csv
4   salir
5 if no coincide la extensión .csv then
6   muestra: Error, sólo admito archivos csv
7   salir
8 cabecera ← nombres de listas admitidas
9 entrada_cabecera ← read_headers(argumento[1])
10 if longitud(entrada_cabecera) no es igual a longitud(cabecera) then
11   muestra: ERROR El archivo csv debe tener los siguientes encabezados: origin, destination, origin_latitude,
12     origin_longitude, destination_latitude, destination_longitude
13   salir
14 foreach cabeza in entrada_cabecera do
15   if cabeza no está en cabecera then
16     muestra: ERROR El archivo csv debe tener los siguientes encabezados: origin, destination,
17       origin_latitude, origin_longitude, destination_latitude, destination_longitude
18   salir
```

Función 5: run

Entrada: Nombre de un archivo (ruta)

```
1 validate_file(argumentos al correr el programa);
2 entradas ← read_csv_file(argumento al correr el programa)
3 solicitudes_no_repetidas ← read_no_repeated_coordinates(argumentos al iniciar)
4 foreach solicitud en solicitudes_no_repetidas do
5   peticion ← make_api_request_by_coordinates(solicitud[0], solicitud[1])
6   peticiones ← setdefault(solicitud, peticion)
7 foreach entrada en entradas do
8   muestra: Datos del clima ;) con formato bonito
```

Función 6: make_api_request_by_coordinates

Entrada: latitud, longitud

Salida : llamada a función parse_weather_info

```
1 if contador > 59 then
2   contador ← 0
3   esperar 1 minuto para continuar
4 get(url + latitud y longitud dadas)
5 contador ← contador + 1
```

Función 7: formato_de_horas

Entrada: Número de fecha y hora (unix)

Salida : cadena de texto con hora en formato 12 hrs

```
1 convierte_fotante: numero dado
2 local_timezone ← get_localzone()()
3 local_time ← fromtimestamp(flotante, local_timezone)
4 regresar: local_time con formato de 12 horas, (CODIGO DEL TIEMPO)
```

Función 8: `parse_weather_info`

Entrada: respuesta en formato json

Salida : llamada a función `parse_weather_info`

```
1 try :  
2     | extraer información del archivo json con las llaves proporcionadas por la documentación de la API  
3 catch KeyError:  
4     | regresar: Error, no se pudo consultar la información  
5 regresar: El pronóstico del clima es: X , humedad: x  
6     Temperatura actual: X°C, mínima: X°C, máxima: X°C Amanecer: X Puesta del sol: X
```

main.py

Weather.py

4. Mejora a futuro

Podemos hablar de varias mejoras a este proyecto, pues está elaborado justo para ofrecer posteriormente nuevas funcionalidades a la cliente, entre ellas podemos enunciar

- Aceptar csv con otro tipo de información que no sean coordenadas
Ya que por el momento sólo acepta un formato muy acotado de archivo, donde los parámetros deben estar bien definidos y en cierto orden
- Una interfaz gráfica
En especial si se va a emplear directamente en aeropuertos donde las clientas estén consultando la información
- Conexión con su sistema
Es casi seguro que si se trata de un *aeropuerto* ya cuentan con un sistema integrado, donde podríamos anexar este nuevo componente
- Exportación de datos
Ofrecer que la información pueda persistir en caso de que requieran consultar datos pasados o climas históricos

4.1. Cobro por el programa

Para ponderar el cobro por este pequeño programa hemos decidido seguir la sugerencia de buscar el salario promedio en latinoamérica para un programador, así que, según *talent.com*³ el salario promedio es de

\$92 por hora

eso es súper poquito, nos tardamos aproximadamente 10 horas en hacerlo, idear, modelar y plantear la solución fue sencillo, sólo que hubo que revisar sintaxis en python porque no habíamos trabajado en él y eso causó retrasos. cobraríamos \$1,900.00, así cada uno recibiría \$950 por sus horas invertidas y por el mantenimiento que habrá que darle si presenta algún problema adicional.

³<https://mx.talent.com/salary?job=Programador#>