Proyecto 1 - MYP

Javier Alejandro Mancera Quiroz
No. de cuenta: 319274831

Eduardo Hernández Floriano
No. de cuenta: 319121498

Definición del problema

El problema nos pide obtener los climas de ciertas ciudades de origen y destino de vuelos en una aerolínea. De inicio lo que se tiene como entrada son solo una lista de vuelos con ciudad de origen y destino con sus debidas coordenadas, de las cuales debemos devolver su clima correspondiente de cada ciudad.

Ahora bien, lo que se tiene pensado es que al mostrar los datos del clima de cada ciudad el usuario pueda conocer estos datos para su propio beneficio y tenga un viaje cómodo sin limitantes. Con los datos del clima de la ciudad de origen puede conocer la situación en el que esta si tiene que considerar un lapso de tiempo para que llegue a tiempo a su despegue, si vive algo retirado del aeropuerto cuales son las mejores condiciones en las que tiene que ir al aeropuerto. Por otro lado, con los datos del clima de la ciudad de destino sabrá cual ha sido el estado ambiental de la ciudad si tiene temperaturas más bajas o por el contrario más calurosas para que así el usuario contemple cualquier situación y pueda llevar lo que necesite.

Además creemos que los datos sí son suficientes para la entrega de los resultados necesarios ya que con las coordenadas, podemos consultar el clima para devolverlo. Bien, lo que se hará para obtener los datos requeridos es hacer una petición a una API, es esta ocasión a la llamada Open Weather; esto nos resuelve el problema, ya que podremos consultar el clima del momento de la mayor parte de lugares posible, nosotros acomodaremos la información de tal manera que al desplegarse, se pueda observar el clima de los lugares deseados, así teniendo el dato requerido, como se pensó hasta ahora, fue que despleguemos una lista ordenada alfabéticamente con base en la clave de origen con los vuelos distinguibles del aeropuerto, para que de esa manera el usuario trate de buscar el suyo con base al lugar de origen y destino y pueda seleccionarlo para que de esa manera, se despliegue en la pantalla un mensaje con los datos del clima de origen y de destino.

Requisitos funcionales y no funcionales

Como se mencionó anteriormente, dada una entrada de datos sobre la región específica y devolverá el clima de la región especificada en la entrada, esta salida se desplegará como un string con la información deseada.
Para que esto se lleve a cabo, lo que se hará será llamar a una API que nos entregue esta información para desplegarla al momento de la consulta. La información de los lugares se obtendrá de un archivo csv que lo acomodaremos en un tipo diccionario para facilitar su consulta.

Análisis del proyecto

¿Cuáles son los datos que tenemos?

Lo que recibimos es un archivo "dataset1.csv"que contiene los siguientes datos:

- Ciudad de origen (iata_code_org).
    * Latitud de origen.
    * Longitud de origen.
- Ciudad de destino (iata_code_des)
    * Latitud de destino.
    * Longitud de destino.

Cada uno representa a un aeropuerto.

¿Qué es lo que queremos obtener con dichos datos?

Lo que se quiere obtener es el clima de las ciudades (origen y destino) en tiempo real.

¿Comó obtenemos los resultados esperados con los datos que contamos?

El usuario tendrá la oportunidad de escoger un viaje de una lista de viajes disponibles con ciudad de origen y ciudad de destino que de acuerdo a su selección obtendremos la latitud y la longitud de tales ciudades. Con los datos obtenidos las utilizaremos en un web service para obtener el clima de las ciudades que seleccionó el usuario. Al obtener el clima de las ciudades mostrará el programa el clima de esas ciudades, desplegandolas en la pantalla para que el usuario las pueda visualizar los datos.

Selección de la mejor alternativa

Para este proyecto, nosotros optamos por el lenguaje python, ya que creemos que es un poco fácil de ocupar y además como enfocaremos nuestro proyecto en los objetos, creemos que este lenguaje queda acorde a nuestras necesidades.

Pruebas

En el caso de las pruebas que nosotros realizamos, primero quisimos averiguar cómo hacer pruebas de nuestro programa, ya que a pesar de que antes habíamos hecho pruebas muy básicas, no nos habíamos metido a ver cómo es que realmente se debe hacer una prueba, por lo que estuvimos en búsqueda de una alternativa sencilla y que pudieramos entender su funcionamiento en el menor tiempo posible, y con los tips de la clases en donde se nos mencionó los *asserts*, entonces investigamos cómo funciona un assert en python y vimos que solo nos manda una excepción en donde le podemos mandar un mensaje de nuestro error, pero también encontramos una biblioteca llamada *unittest* que nos ayuda bastante a hacer pruebas en nuestro código, ya que además de poder hacer uso de asserts, también lo que nos puede decir es si pasó la prueba y cuanto tiempo se tardó, eso nos llamó la atención bastante para poder ejecutarlo y ver qué pasa con los tiempos de las funciones de nuestro programa.

Interfaz

Para la interfaz, optamos por ese diseño, ya que así nos lo imaginamos, que tuviera una lista con las claves iata de origen y destino de cada vuelo ordenados alfabéticamente basandonos en el origen, en la cual el usuario tenga que buscar la que corresponda a su boleto para consultar el clima de ambas partes.

Solicitud

Para la solicitud, nosotros optamos por hacer uso de las bibliotecas json y urlopen para poder manejar los datos obtenidos de la API de mejor manera, y de ahí, construir las cosas que se mostrarían al usuario y la información que se deba guardar en el cache.

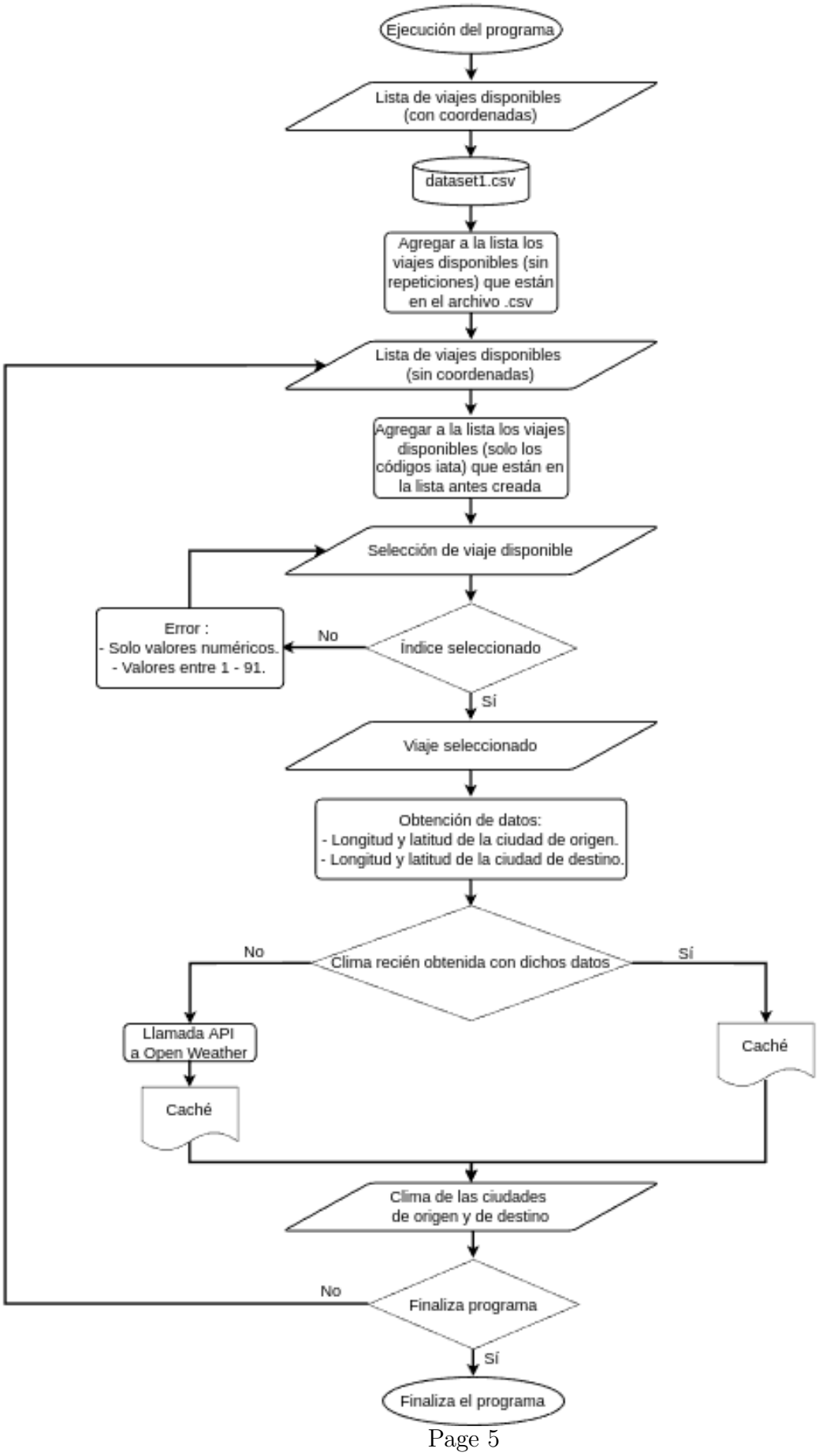
Clave API

Aquí, nosotros optamos por usar una biblioteca llamada *cryptography* que se tiene que descargar por medio de un comando especificado en el README, esto con la finalidad de nosotros poder esconder nuestra clave en git y que no se viera literalmente, sino como una clave encrptada, fue una de las soluciones que pensamos ya que, encriptar la llave se modifica la forma de esta y no es la misma que la que se requiere para hacer la llamada, por eso hicimos un método para después desencriptar y hacer la petición. Al final no supimos si fue la mejor opción en cuestiones de seguridad, ya que teníamos otra solución que era no subir el archivo donde tenía la llave y sólo tenerlo localmente en una carpeta específica que nos pasaríamos solo entre los colaboradores para colocarlo y que sirviera. Así, que pues al final quedó el encriptado.

Pseudocódigo

1. **INICIO** de programa
2. **OBTENER** dataset1.csv
3. **MIENTRAS** que se lee dataset1.csv
4. **AGREGAR** los datos a una lista sin repetidos
5. **FIN** de mientras
6. **MIENTRAS** que se lee la lista sin repetidos
7. **AGREGAR** los datos iata_code a una lista de viajes disponibles
8. **FIN** de mientras
9. **MOSTRAR** los elementos de viajes disponibles
10. **RECIBIR** viaje seleccionado
11. **OBTENER** los datos de longitud y latitud de la ciudad de origen y la ciudad de destino
12. **SI** los datos se encuentran en el caché **ENTONCES**
13. **OBTENER** el clima del caché
14. **SINO**
15. **SOLICITAR** API a Open Weather
16. **GUARDAR** datos al caché
17. **MOSTRAR** clima de la ciudad de origen y la ciudad de destino
18. **FIN** de programa

Diagrama de flujo (vista desde la terminal)



Planes futuros

Posiblemente, lo que se nos ocurrió que podríamos llevar a cabo a futuro sería crear un sistema en la nube en la que podamos de alguna manera interactuar con el programa desde cualquier lugar con tan solo conectarse a internet. Otra posible mejora que se nos ocurrió podría ser hacer el programa para que se pudiera ejecutar en un celular y de esa manera hacerlo más fácil de consultar. Otra opción podría ser que el programa mostrara una recomendación o una advertencia acerca del destino: mostrando vestimentas que podría utilizar, herramientas que le pudieran servir, referencias a páginas web donde muestre lugares que puede visitar de esa ciudad, que muestre recientes noticias que debería de conocer el usuario y platillos que no debería de perderse probarlos.

Costo del programa

Por último, para el costo del programa realizamos un breve análisis de cuanto tiempo nos tomó en realizar el programa y nuestra experiencia en el área de la programación para así obtener un precio adecuado para el usuario y de acuerdo a nuestros servicios en hacerlo y necesidades. Por lo cual, de acuerdo a nuestra experiencia estuvimos de acuerdo en tener un precio fijo por hora de $50,00 que lo multplicamos de acuerdo a las necesidades requeridas tanto del usuario como para el cliente que solicita el programa y las horas que nos llevó hacer cada rubro, siendo las siguientes:

Para cliente	
Requisitos del programa	Hora(s) de trabajo
Requisitos funcionales	
- Utilizar web services.	1 hr
- Manejo de errores y pruebas realizadas	21 hrs
- Modelar un caché	10 hrs
Pertinencia del diseño	
- Explicar el procedimiento de tu programa.	7 hrs
Pulcritud de la implementación	
- Información extraída y código documentado.	10 hrs
Eficiencia	
- Desempeño	1 hr
Total	50 hrs

Para usuario	
Requisitos del programa	Hora(s) de trabajo
- Interfaz gráfica.	48 hrs
- Datos esperados.	3 hrs
- Opciones disponibles.	4 hrs
- Ejecución y rapidez del programa.	1 hr
- Instrucciones e indicadores claras.	1 hr
- Actualización de datos.	7 hrs
Total	64 hrs

Siendo un total de 114 hrs de trabajo dedicados al programa, que al multiplicarlos por 50 nos da como resultado $\$5,700$, siendo esté el costo de nuestro programa.

Reporte final

El resultado final de acuerdo a nosotros fue el esperado, optamos por crear una interfaz gráfica que visualizar los datos en la terminal ya que queríamos experimentar un poco el desarrollo de un programa con una interfaz con la que podamos visualizar más comodamente, aparte se veia algo "sucio"como se veia en la terminal y deja mucho que desear porque su interacción con el usuario era algo tedioso y muy repetitivo, hasta pudimos eliminar líneas de código para que se viera más limpio nuestro código.
Para entender cual fue nuestro criterio para realizar el programa lo explicaremos en forma detallada por pasos:

1. Para poder obtener los datos del archivo .csv se tenia que leer linea por linea pero notamos que varios datos se repetían así que pusimos un ciclo que revisara linea por linea para tenerlos en una estructura que nos facilitara el acceso de cada uno y no se tuviera ninguna repetición de algún dato.

2. Ahora lo que se requería era de tener los datos que necesitabamos mostrar, las ciudades de origen y la de destino, al usuario ya que no es necesario mostrar sus latitudes y longitudes de cada ciudad, sino simplemente sus nombres, aparte nos ayudaría esta estructura que formamos solo con los iata_code para poderlo tener en la interfaz gráfica y así el usuario pueda ver mejor las opciones.

3. Después seguiría la solicitud de la API, que en la cual, importamos los siguientes módulos:
   - json = Que nos ayudó a poder visualizar mejor los datos solicitados.
   - urlopen = Sirvió para poder hacer la solicitud a la web service .ºpenWeather".
   - cache = Nuestra caché con la que contamos para guardar la información obtenida.
   - entrada = Para poder manipular nuestra lista ya creada y obtener las coordenadas requeridas.

   Para poder utilizar la llave API la guardamos en un archivo que para tener mayor seguridad realizamos un método para encriptarlo que nos ofrece git-crypt para que así el usuario no sepa que contiene nuestra llave.

4. Al utilizar el caché nosotros pensamos que la mejor opción sería de que tuviera su propio módulo para que su accesso sea más sencillo y que no estuvieramos peleando con las funciones del archivo SolicitaAPI.py.

5. En la creación de la clase de la interfaz gráfica utilizamos:
   - tkinter = Para poder crear y personalizar nuestra ventana para el usuario.
   - SolicitaApi = obtener los resultados del clima.
   - timer = que nos ayuda a actualizar los datos del cache para que no obtengamos resultados incoherentes.

   Su diseño se basa en dos partes: una que es fija, da una cierta entrada del programa y para que interactue el usuario, y la otra es la parte que cambia de acuerdo a lo solicitado por el usuario. La utilización de los paquetes mencionados anteriormente, se deben al material disponible para su comprensión, es decir, nos basamos en los ejemplos que existen en donde se usan estas bibliotecas y decidimos proceder, creando un botón y una lista en donde se puedan visualizar los distintos viajes para elegir el correspondiente al boleto del cliente.

6. En el caso del acomodo de los archivos de la manera en la que están fue una solución a un inconveniente que tuvimos durante la organización de los ficheeros y la importación de los módulos para su convivencia y utilización entre ellos, ya que no podíamos correrlo porque no se reconocían, de hecho los archivos de las pruebas se dejaron a lado de principal.py debido a la misma razón, que es que si los poníamos en otra carpeta fuera o a lado de principal.py, no se ejecutaba.

Referencias

Para poder trabajar con un json:

- https://www.youtube.com/watch?v=9N6a-VLBa2I

Para realizar las pruebas:

- https://ellibrodepython.com/python-testing
- https://docs.python.org/3/library/unittest.htmlunittest.TestCase.debug
- https://www.youtube.com/watch?v=6tNS–WetLI
- https://recursospython.com/guias-y-manuales/unit-testing-doc-testing/

Para poder actualizar el caché:

- https://www.youtube.com/watch?v=5NJ9cc0dnCMt=12s

Documentación de tkinter (para la interfaz gráfica):

- https://www.tutorialspoint.com/python/python_gui_programming.htm

Para poder encriptar nuestro archivo que contiene la llave API:

- https://dev.to/heroku/how-to-manage-your-secrets-with-git-crypt-56ih

Para saber cuánto cobrar por nuestro programa:

- https://www.youtube.com/watch?v=8JMIDQOU8MMt=1995s