



UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

Documentacion

Modelado y Programacion

Victor Federico Torres Trejo
Diego Castro Rendon

October 4, 2022

Contents

1	Definicion del problema	4
1.1	Entender el problema	4
1.2	Arsenal	4
1.3	Requisitos funcionales	4
1.4	Requisitos no funcionales	4
2	Analisis del problema	5
2.1	Proceso de Solucion	5
2.2	Modelo de Datos	6
2.3	Seleccion de la mejor alternativa	7
3	Diagrama de Flujo	9
4	Mantenimiento	12
5	Pruebas	12
5.1	Aeropuerto	13
5.2	Prueba de Archivos	14
5.2.1	revisarCSV	14
5.2.2	escribirDestinos	15
5.2.3	leerDestinos	16
5.3	Cache Clima	17
5.3.1	refrescar	17
5.3.2	buscar aeropuerto	19
5.3.3	obtener clima y realizar peticion	20
5.4	Historial	22
5.4.1	Obtener numero de vuelo	22
5.4.2	Convertir a vuelo	23
5.5	Lista de Aeropuertos	24
5.5.1	Procesar Vuelo	24
5.5.2	Buscar Aeropuerto de Origen	25
5.5.3	Insertar aeropuerto de origen	26
5.5.4	Buscar aeropuerto de destino	27
5.5.5	Revisar archivos	29
5.5.6	Obtener nombres	29

6	Referencias	29
6.1	Web services	29
6.2	Json	30
6.3	Tiempo	30
6.4	Archivos	31
6.5	Interfaz	31
6.6	Setup.py	32
6.7	Asserts	32
6.8	Documentacion	32
7	Costos	33

1 Definicion del problema

Dia a dia miles de personas van y vienen de aeropuertos a aeropuertos, cambiando drasticamente de zona horaria y clima. El aeropuerto de la Ciudad de Mexico te contrata para una tarea, la cual es entregar el informe del clima de la ciudad de salida y la ciudad de llegada para 3 mil tickets que salen el mismo dia que se corre el algoritmo. El programa no busca que sea interactivo, ya que el mercado a quien va dirigido son sobrecargos y pilotos que desconocen de programacion, por lo que solo nos interesara el clima.

1.1 Entender el problema

Se quiere obtener el clima del aeropuerto de origen como el de destino, dada su longitud y latitud de cada aeropuerto ademas de una llave API para poder hacer uso de servicios web y obtener dichos datos del clima. Estos datos suficientes siempre y cuando la llave que se tenga es valida, ya que en el caso contrario no podemos solicitar el clima de ninguna ubicacion. El resultado es una solucion si la informacion que contiene es el clima de dos aeropuertos dados. Para llegar a dicho resultado nosotros al hacer la peticion tenemos que filtrar el resultado parcial para poder seleccionar la informacion que mas nos interesa y proporcionar la informacion solicitada al usuario.

1.2 Arsenal

- Paradigma: orientado a objetos
- Lenguaje: Python
- Herramientas: API, JSON, CSV, Tkinter, DateTime

1.3 Requisitos funcionales

Obtener el clima de los aeropuertos de origen y destino mediante el uso de servicios web (API) dada la ubicacion de los mismos y una llave API valida..

1.4 Requisitos no funcionales

Garantizar la eficacia, seguridad al procesar los datos y la resistencia a fallos del programa. El programa debe ser amigable para cualquier persona ajena

a los conceptos de computacion y ajena al manejo de una terminal. Debe minimizar la entrada proporcionada por el usuario.

2 Analisis del problema

2.1 Proceso de Solucion

Se va a usar tanto enfoque procedural como orientado a objetos, en cuanto al **enfoque procedural** tenemos la base de datos como entrada en formato de archivo CSV que contiene el nombre de cada ciudad, latitud y longitud del aeropuerto de destino y de origen. Leemos todos los vuelos de la base de datos y retornamos una serie de archivos JSON por cada aeropuerto de origen, que contiene el aeropuerto de origen como todos los destinos que tiene. Esto despliega todos los aeropuertos de origen disponible para que el usuario seleccione que vuelo quiere conocer, una vez que selecciona el vuelo se da como entrada el nombre del aeropuerto y despliegan todos los aeropuertos de destino dado el JSON ademas que se realiza la peticion de la informacion del aeropuerto origen.

Y despues que se despliegan los destinos el usuario selecciona uno y se le da el boton de consultar, el cual realiza la peticion del aeropuerto destino y transforma los datos JSON en datos legibles. Tambien cada consulta registramos un vuelo en el historial.

En cuanto al **enfoque orientado a objetos** se crea una clase aeropuerto que encapsula los datos de cada aeropuerto y nos permite calcular su funcion hash, esto se explicara mas adelante.

Una clase que nos permite guardar los datos de los climas para evitar realizar peticiones innecesarias, esto es cada 5 minutos pedir el clima cuando no ha cambiado en ese intervalo, tiene como atributos la lista de climas, el tamaño de dicha lista ademas de el valor de la llave API y su comportamiento es evitar esas peticiones innecesarias ademas de realizar la peticion cuando si se requiera y proporcionar el clima, para esto hace uso de la clase datosClima, la cual filtra y proporciona de una manera mas legible los datos de los clima, ya que estos al realizar una peticion nos dan en formato JSON.

Ademas tenemos una clase que nos permite hacer la filtracion o el proceso que dada la base de datos, obtener un arreglo de listas donde cada lista es un aeropuerto origen y sus aeropuertos de destino, sus atributos igual son una lista de listas y su tamaño de la lista de listas, en cuanto a comportamiento

procesa cada linea del csv denominada vuelo, permite buscar si un aeropuerto de origen o destino ya existe ademas de la insercion de los de origen y escribe un archivo JSON por cada aeropuerto de origen, revisa la base de datos antes de procesarla tambien nos permite conocer los nombres de los aeropuertos de origen, esto para mostrarla en la interfaz.

Adicional tenemos una clase que codifica la clase aeropuerto para su escritura en un archivo JSON.

Y por ultimo tenemos la clase de la interfaz la cual nos permite una mejor experiencia y amigabilidad con el usuario, ya que restringe la entrada a solo validos aeropuertos y es muy intuitiva de usar, su comportamiento se define por elegir aeropuerto de origen y muestra todos los de destino, muestra todos los aeropuertos de origen, muestra los datos del clima en pantalla, permite editar la llave API, ademas de mostrar los ultimos 3 vuelos realizados

2.2 Modelo de Datos

Los datos se representaran principalmente a traves de tablas hash o de dispersion o diccionarios, como queremos almacenar la informacion de muchos aeropuertos se considero que un arreglo es la estructura mas adecuada para almacenar los distintos datos, esto determina el desempeño ya que el acceso a un arreglo es constante, ahora se usa una tabla de dispersion porque al tener una llave hash relacionada a los datos del aeropuerto el acceder a los valores almacenados conociendo su llave hash es tiempo constante, lo cual beneficia mucho el desempeño. Se ajusta perfecto al esbozo de una solucion ya que nos permite almacenar los datos dado un tamaño y una tabla hash, nos permite conocer y manipular la informacion de manera constante o casi constante, en el caso de lista de aeropuertos donde tenemos lista de listas, conocer la ubicacion de la lista del aeropuerto de origen es constante y generalmente no son muchos los aeropuertos de destino, por lo que acceder a ellos es muy rapido, en el caso del cache es una tabla hash la cual accedemos al clima de manera constante. Se manejan las colisiones de manera lineal, es decir, si la casilla correspondiente esta ocupada, se busca la siguiente disponible y el tamaño se selecciono tratando de minimizar las colisiones y el tamaño. No puede haber algo mejor en cuanto a tiempo porque el acceder a los datos es constante y si hay peores porque en tanto los arreglos (no hay relacion aeropuerto-indice), lista, cola, pila el acceder a los elementos es complejidad lineal.

2.3 Seleccion de la mejor alternativa

Ya se expuso el porque la estructura de datos tabla hash es la mejor para el proyecto y el proceso de solucion, del diseño de las clases asi como su comportamiento y atributos y el como estas interactuan entre si vistas desde un enfoque procedural, asi que solo queda como vamos a seleccionar la mejor alternativa de diseño de la interfaz, lo que se busca es minimizar la interaccion con el usuario de manera que este tenga que ingresar lo minimo indispensable para evitar errores de entrada, ya que la aplicacion va dirigida a personas ajenas a la computacion, para esto usamos el procesamiento de la base de datos para obtener los aeropuertos de origen, de esta manera podemos hacer que el usuario seleccione el aeropuerto requerido de una lista restringida, una vez seleccionada esta opcion se despliega los aeropuertos de destino de este aeropuerto de origen, al usuario seleccionar la opcion, se tiene control sobre la entrada del sistema, se evitan errores. Para poder desplegar las opciones de destino se usa tablas hash y escritura sobre archivos formato JSON, para las peticiones se sabe que no podemos hacer mas de 60 por minuto, pero tambien sabemos que el usuario no es lo suficientemente veloz como para seleccionar y consultar 60 peticiones por minuto, ademas que hay un retraso de unos segundos por la lectura del archivo JSON correspondiente para el despliegue de aeropuertos destino, por lo que eso no sera un problema, lo que si es que hay que evitar requests innecesarias por ejemplo la misma peticion con un intervalo de 10 minutos ya que el clima no ha variado significativamente, por lo que establecemos un intervalo de 30 minutos y con tablas hash simulando una memoria cache guardamos la informacion json ademas de la hora en que se ha realizado la peticion. Por ultimo para registros previos podemos visualizar los ultimos 3 vuelos registrados. Entiendase registrar por un vuelo consultado. Cabe mencionar que se va a hacer uso de OpenWeatherAPI para la request de los datos del clima de cada aeropuerto.

Para poder garantizar la minima interaccion se diseño un boceto el cual solo tiene 3 entradas: la llave API, la seleccion de aeropuerto de origen y destino la cual como ya mencionamos se acota, asi que al diseñar el boceto de la aplicacion quedo de la siguiente manera:

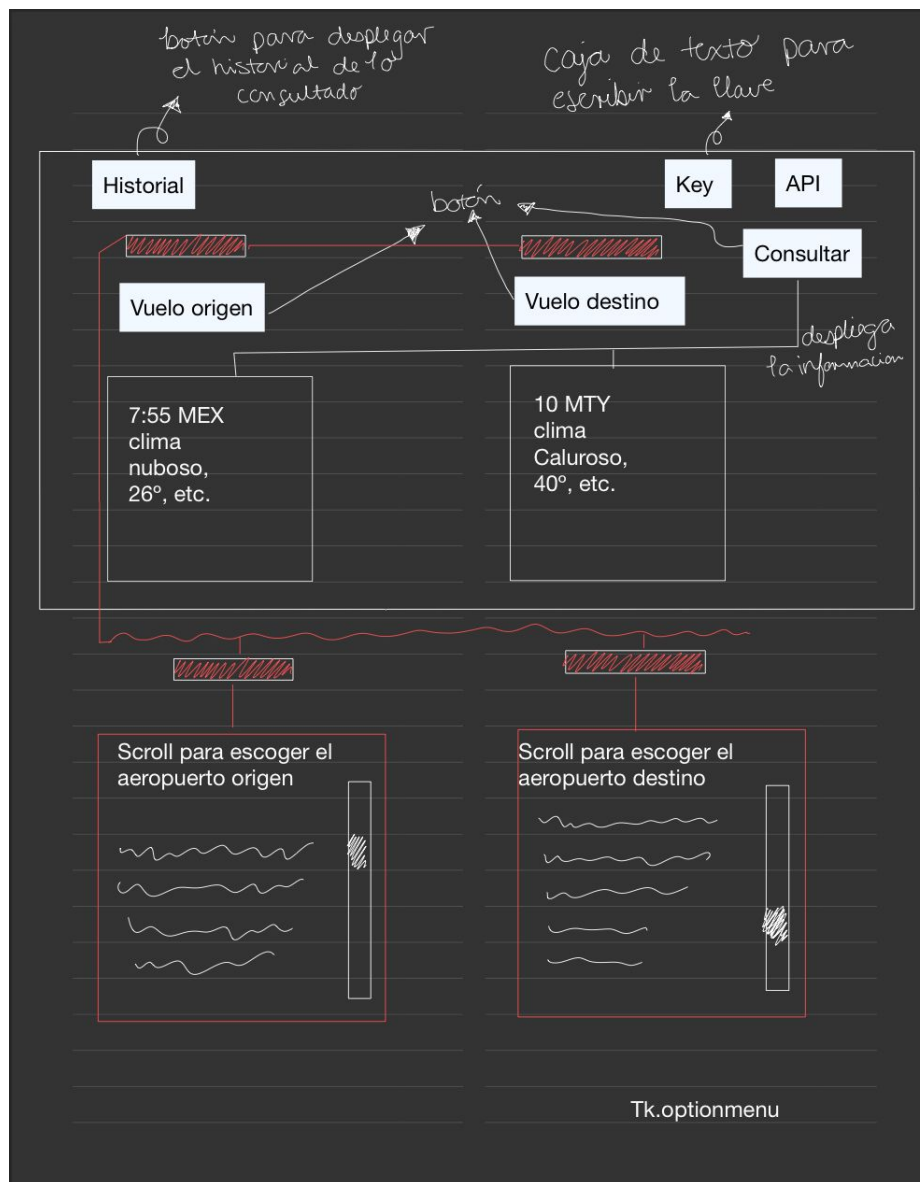
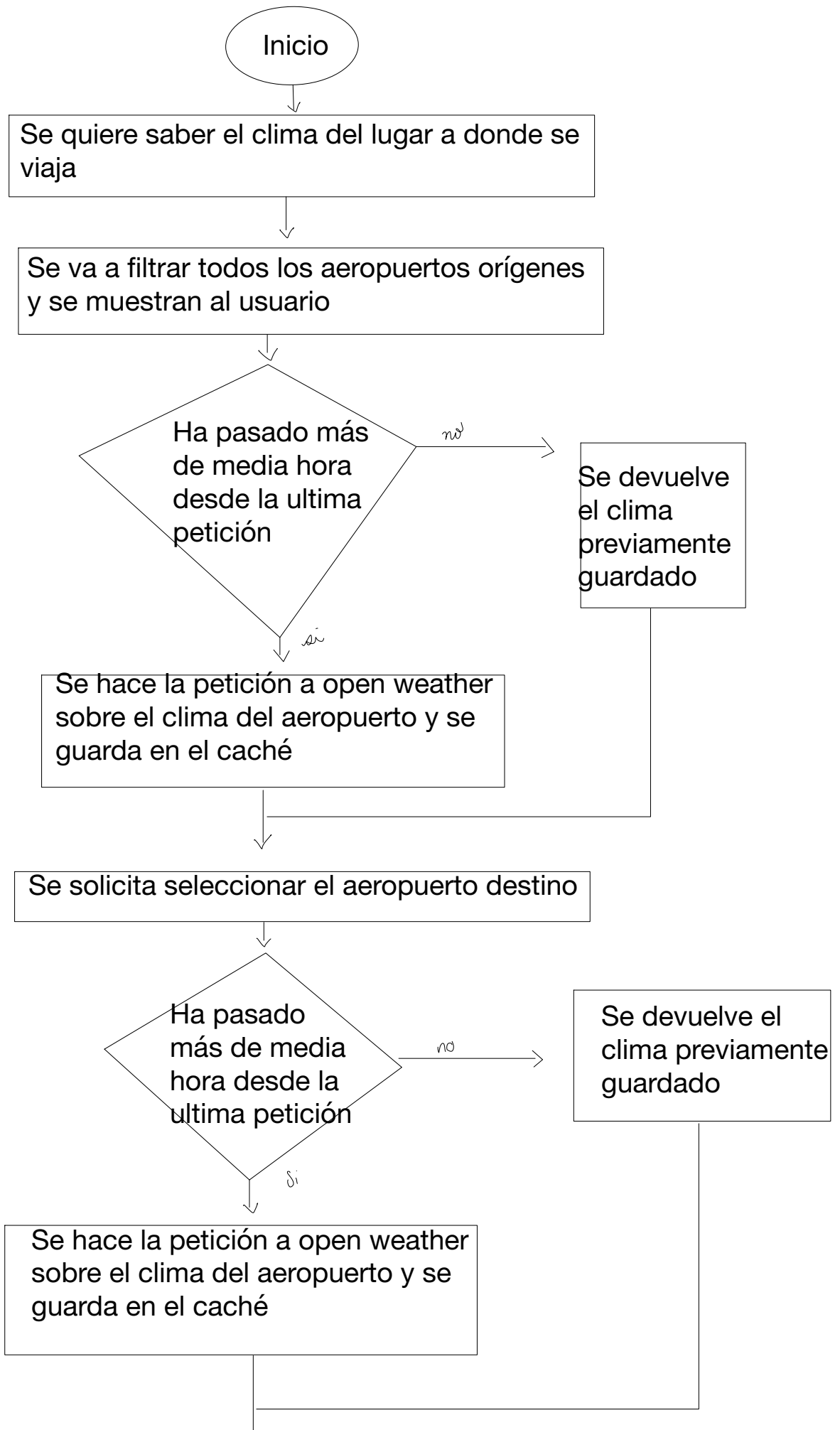
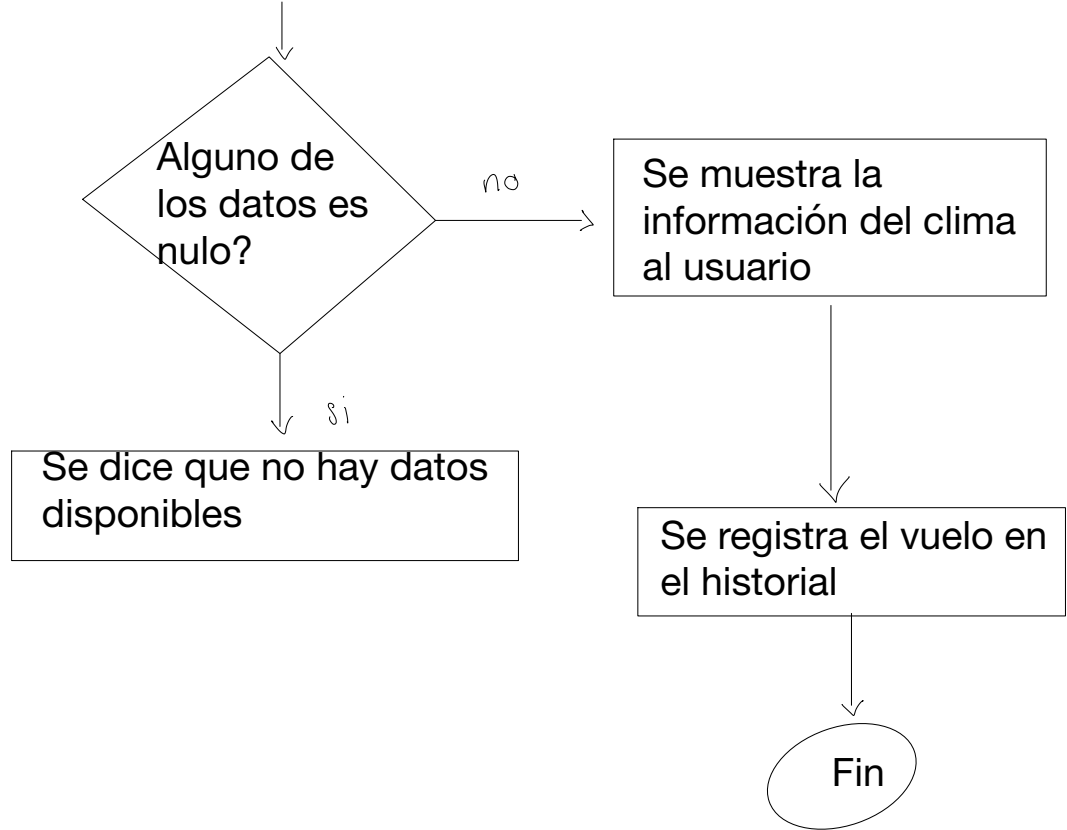


Figure 1:

3 Diagrama de Flujo





4 Mantenimiento

El sistema se dividió en 9 archivos el cual cada uno representa una clase o un conjunto de funciones para que el sistema pueda funcionar de manera correctamente, estos son:

- aeropuerto.py el cual encapsula los datos del aeropuerto
- procesarArchivos.py el cual contiene las funciones para procesar la base de datos, escribir las salidas parciales en archivos JSON y leer esos archivos
- cacheClima.py contiene la clase para simular la cache del sistema y las peticiones con la API
- codificadorJsonAeropuerto.py la cual nos ayuda a poder transformar el aeropuerto en una salida formato JSON
- datosClima.py la cual nos ayuda a filtrar los datos JSON en algo mas legible y facil de entender.
- historialVuelo.py la cual nos ayuda a mantener un registro de los vuelos consultados
- interfaz.py la cual contiene todo lo relacionado con la interfaz
- listaDeAeropuertos.py la cual nos ayuda a procesar la base de datos usando tablas hash y guardar la informacion de cada aeropuerto de origen.
- main.py contiene la funcion principal que es la que corre el sistema

Tambien cabe mencionar que si las librerias cambian o el mismo lenguaje recibe una actualizacion importante es importante considerar mantenimiento.

5 Pruebas

Como se dividió el archivo en 9 programas se haran las pruebas pertinentes a los archivos que tengan funcionalidad, en cuanto al archivo main.py es meramente declarativo, es decir solo manda a inicializar la interfaz por lo que no es necesario hacer pruebas sobre ese archivo, sobre datosClima.py como

solo es utilizado para encapsular datos no tiene ningun metodo tampoco se haran pruebas sobre el, en cuanto a la interfaz sus metodos tienen entradas definidas por la misma interfaz, es decir, leer la opcion seleccionada ademas de metodos que son para mostrar informacion en pantalla o eliminarla, por lo tanto no se haran pruebas sobre interfaz.py


5.1 Aeropuerto

En esta clase solo tenemos el metodo de funcion hash, por lo que probaremos con un aeropuerto cualquiera y un objeto nulo

Tenemos para el siguiente codigo un aeropuerto con la siguiente entrada tenemos que el valor hash del aeropuertoPrueba es $(77 + 69 + 88) + (10) + (100) = 341\%11 = 3$

```
aeropuertoPrueba = aeropuerto.Aeropuerto('MEX', 10, 100)
assert(aeropuertoPrueba.funcionHash(11) == 3)
```

Por lo que al correr la siguiente salida es:



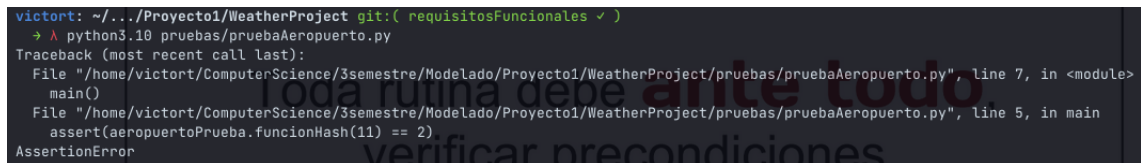
```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales ✓ )
→ python3.10 pruebas/pruebaAeropuerto.py
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales ✓ )
→
```

Figure 2: Salida

Si ahora probamos con un resultado distinto a 3, sea 2

```
aeropuertoPrueba = aeropuerto.Aeropuerto('MEX', 10, 100)
assert(aeropuertoPrueba.funcionHash(11) == 2)
```

Se obtiene la siguiente salida:



```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales ✓ )
→ python3.10 pruebas/pruebaAeropuerto.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaAeropuerto.py", line 7, in <module>
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaAeropuerto.py", line 5, in main
    assert(aeropuertoPrueba.funcionHash(11) == 2)
AssertionError
```

Figure 3: Salida

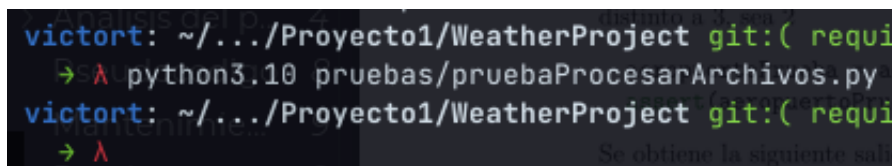
5.2 Prueba de Archivos

| Para el prueba de archivos la funcion de revisarFormatoVuelo es meramente auxiliar para probar que el formato de vuelo sea de la forma: (string,string,float,float, float, float) para poder validar que el formato de vuelo es valido, pero esta funcion es llamada desde revisarCSV por lo que es redundante hacer una prueba de esto ya que se usa directamente al revisar la base de datos dada.

5.2.1 revisarCSV

- Si revisamos dataset1.csv como base de datos, la cual es una base de datos valida, tendremos la siguiente salida

```
assert(revisarCSV('dataset1.csv')), "Base de datos invalida"
```

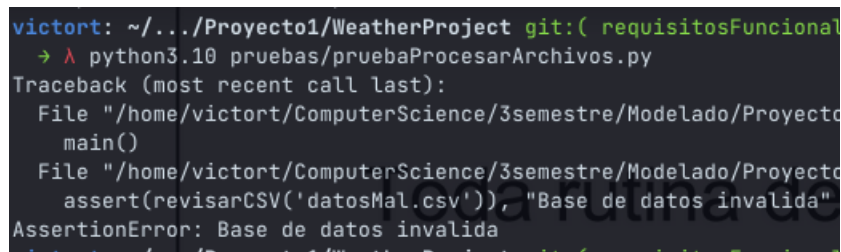


A terminal window showing a user named victort in a directory ~/.../Proyecto1/WeatherProject. They run the command `python3.10 pruebas/pruebaProcesarArchivos.py`. The output shows the assertion passing, indicating the dataset is valid. A watermark 'Se obtiene la siguiente salida' is visible at the bottom right.

Figure 4: Salida

- En el caso de que ingresemos datos.csv la cual es una base de datos no valida, tendremos la siguiente salida

```
assert(revisarCSV('datosMal.csv')), "Base de datos invalida"
```

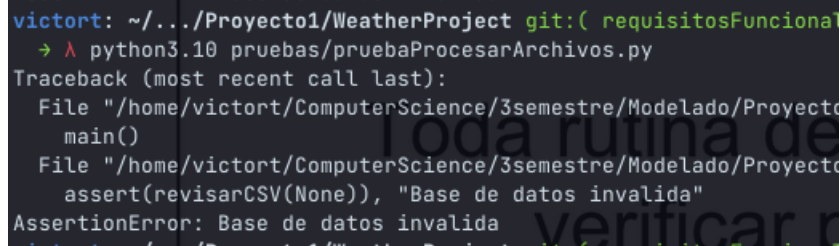


A terminal window showing a user named victort in a directory ~/.../Proyecto1/WeatherProject. They run the command `python3.10 pruebas/pruebaProcesarArchivos.py`. The output shows a traceback with an `AssertionError: Base de datos invalida` message, indicating the dataset is invalid. A watermark 'Se obtiene la siguiente salida' is visible at the bottom right.

Figure 5: Salida

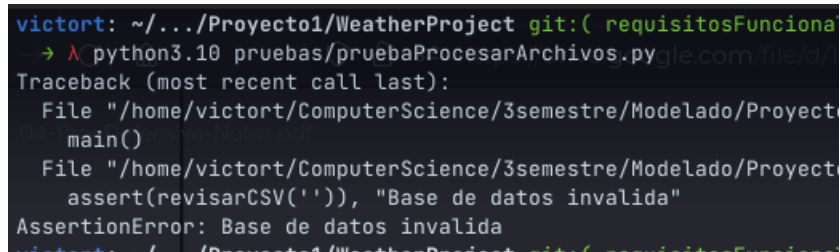
- Si no ingresamos ningun archivo como base de datos, tendremos la siguiente salida

```
assert(revisarCSV(None)), "Base de datos invalida"
assert(revisarCSV('')), "Base de datos invalida"
```



```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFunciona
→ ^ python3.10 pruebas/pruebaProcesarArchivos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto
    assert(revisarCSV(None)), "Base de datos invalida"
AssertionError: Base de datos invalida
```

Figure 6: Salida



```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFunciona
→ ^ python3.10 pruebas/pruebaProcesarArchivos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto
    assert(revisarCSV('')), "Base de datos invalida"
AssertionError: Base de datos invalida
```

Figure 7: Salida

5.2.2 escribirDestinos

- Para escribir los destinos revisaremos si le damos el archivo correcto de una base de datos recortada para poder calcular la salida, para poder comparar bien creamos una funcion auxiliar que compara dos listas

```
lista = escribirDestinos("../pruebas/datosPrueba/baseDeDatos.csv", 2)
assert(compararLista(lista.lista, [[], [['TLC', 19.3371, -99.566]\
, ['MTY', 25.7785, -100.107]], [])), "Destinos invalidos"
```

```
victort: ~/.../Proyecto1/WeatherProject git:(requisitosFuncionales U:11 ?:1 x)
→ python3.10 pruebas/pruebaProcesarArchivos.py
victort: ~/.../Proyecto1/WeatherProject git:(requisitosFuncionales U:11 ?:1 x)
→
```

Figure 8: Salida

- El resultado si le pasamos un archivo nulo

```
lista = escribirDestinos("None", 2)
assert(compararLista(lista.lista, [[], [['TLC', 19.3371, -99.566],\
, ['MTY', 25.7785, -100.107]],[])), "Destinos invalidos"
```

```
victort: ~/.../Proyecto1/WeatherProject git:(requisitosFuncionales U:11 ?:1 x)
→ python3.10 pruebas/pruebaProcesarArchivos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaProcesarArchivos.py", line 1, in <module>
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaProcesarArchivos.py", line 1, in main
    assert(compararLista(lista.lista, [[], [['TLC', 19.3371, -99.566], ['MTY', 25.7785, -100.107]],[])), "Destinos invalidos")
AttributeError: 'NoneType' object has no attribute 'lista'
```

Figure 9: Salida

5.2.3 leerDestinos

- Esta funcion lee los destinos de un aeropuerto de origen, asi que se lee los destinos de Acapulco

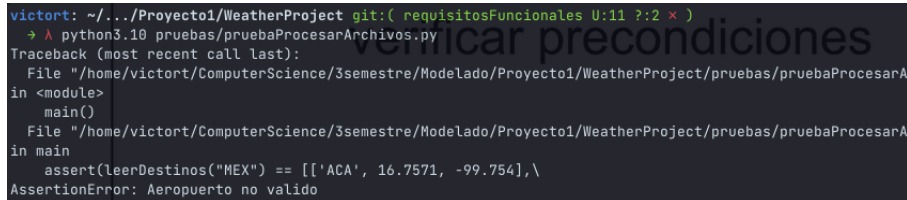
```
assert(leerDestinos("ACA") == [['ACA', 16.7571, -99.754],\
['MEX', 19.4363, -99.0721]])
```

```
victort: ~/.../Proyecto1/WeatherProject git:(requisitosFuncionales U:11 ?:1 x)
→ python3.10 pruebas/pruebaProcesarArchivos.py
victort: ~/.../Proyecto1/WeatherProject git:(requisitosFuncionales U:11 ?:1 x)
→
```

Figure 10: Salida

- Se lee los destinos de un aeropuerto inexistente

```
assert(leerDestinos("MEX") == [['ACA', 16.7571, -99.754],\
                                ['MEX', 19.4363, -99.0721]])
```

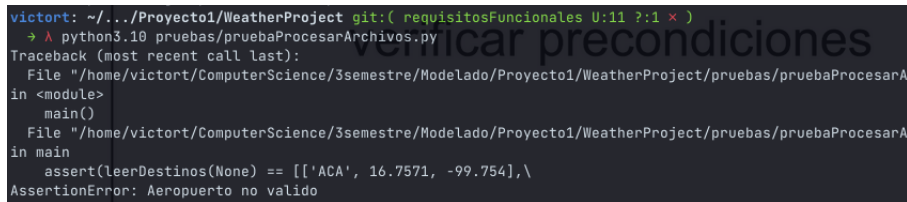


```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:11 ? :2 × )
→ A python3.10 pruebas/pruebaProcesarArchivos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaProcesarA.py", line 1, in <module>
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaProcesarA.py", line 1, in main
    assert(leerDestinos("MEX") == [['ACA', 16.7571, -99.754],\
AssertionError: Aeropuerto no valido
```

Figure 11: Salida

- Se lee los destinos de un aeropuerto nulo

```
assert(leerDestinos(None) == [['ACA', 16.7571, -99.754],\
                              ['MEX', 19.4363, -99.0721]])
```



```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:11 ? :1 × )
→ A python3.10 pruebas/pruebaProcesarArchivos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaProcesarA.py", line 1, in <module>
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaProcesarA.py", line 1, in main
    assert(leerDestinos(None) == [['ACA', 16.7571, -99.754],\
AssertionError: Aeropuerto no valido
```

Figure 12: Salida

5.3 Cache Clima

Creamos un cache de prueba, el metodo de actualizaAPI no tiene sentido probarlo dado que solo establece un atributo, pero si lo usaremos en refrescar que realiza la peticion del servicio web

5.3.1 refrescar

- Se refresca con el aeropuerto de mexico

```
cacheEjemplo.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
, "No se actualizaron los datos"
```



```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:1 × )
→ python3.10 pruebas/pruebaCacheClima.py
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:2 × )
→
```

Figure 13: Salida

- Se refresca con un aeropuerto nulo

```
cacheEjemplo.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
aeropuerto1 = None
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
, "No se actualizaron los datos"
```

```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:12 ? :2 × )
→ python3.10 pruebas/pruebaCacheClima.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py", line 14, in <module>
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py", line 10, in main
    assert(cacheEjemplo.refrescarClima(aeropuerto1)), "No se actualizaron los datos"
AssertionError: No se actualizaron los datos
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:12 ? :2 × )
```

Figure 14: Salida

- No se establece una llave API por lo que es errónea la llave vacía y se solicita la información del aeropuerto de México

```
aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
, "No se actualizaron los datos"
```

```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:12 ? :2 × )
→ python3.10 pruebas/pruebaCacheClima.py
HTTP Error 401: Unauthorized
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py", line 14, in <module>
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py", line 10, in main
    assert(cacheEjemplo.refrescarClima(aeropuerto1)), "No se actualizaron los datos"
AssertionError: No se actualizaron los datos
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:12 ? :2 × )
```

Figure 15: Salida

5.3.2 buscar aeropuerto

- Con el aeropuerto de mexico insertado, se busca el aeropuerto de mexico

```
aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
cacheEjemplo.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
    , "No se actualizaron los datos"
assert(cacheEjemplo.buscarAeropuerto(aeropuerto1) != -1)\
    , "Aeropuerto no encontrado"
```



Figure 16: Salida

- Se busca el aeropuerto de acapulco el cual no se encuentra en el cache

```
aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
aeropuerto2 = Aeropuerto('ACA', 16.7571, -99.754)
cacheEjemplo.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
    , "No se actualizaron los datos"
assert(cacheEjemplo.buscarAeropuerto(aeropuerto1) != -1)\
    , "Aeropuerto no encontrado"
```

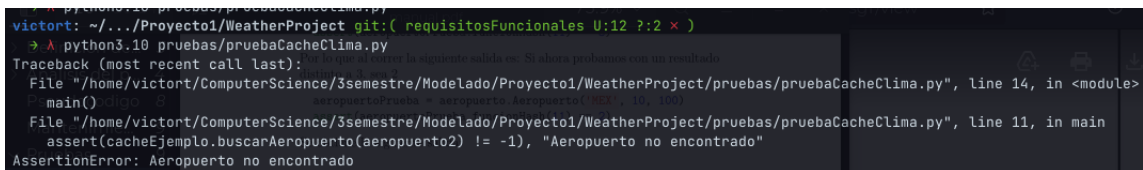
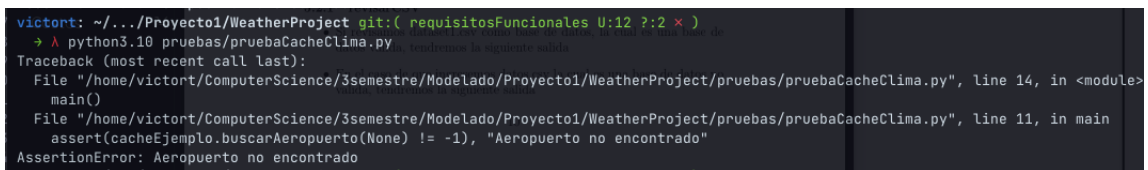


Figure 17: Salida

- Se busca un aeropuerto nulo

```
aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
cacheEjemplo.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
    , "No se actualizaron los datos"
assert(cacheEjemplo.buscarAeropuerto(None) != -1)\
    , "Aeropuerto no encontrado"
```



```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:12 ?;2 x )
→ python3.10 pruebas/pruebaCacheClima.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py", line 14, in <module>
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py", line 11, in main
    assert(cacheEjemplo.buscarAeropuerto(None) != -1), "Aeropuerto no encontrado"
AssertionError: Aeropuerto no encontrado
```

Figure 18: Salida

5.3.3 obtener clima y realizar peticion

Se compara con realizar peticion ya que los datos que hay en el cache son formato JSON que son los datos que devuelve realizar peticion pero realizar peticion lo hace directamente.

- Se obtiene el clima de un aeropuerto que ya se refresco el aeropuerto

```
aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
cacheEjemplo.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
    , "No se actualizaron los datos"
assert(cacheEjemplo.obtenerClima(aeropuerto1)\
    .__eq__(DatosClima(cacheEjemplo\
    .realizarPeticion(aeropuerto1)))) , "Datos JSON distintos"
```

```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:1 × )
→ ^ python3.10 pruebas/pruebaCacheClima.py - google.com/file/d/13dFvn35PMU
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:2 × )
→ ^
```

Figure 19: Salida

- Se busca el clima de un aeropuerto nulo

```
aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
cacheEjemplo.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
    , "No se actualizaron los datos"
assert(cacheEjemplo.obtenerClima(None)\
    .__eq__(DatosClima(cacheEjemplo\
    .realizarPetición(aeropuerto1)))) , "Datos JSON distintos"
```

```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:14 ? :2 × )
→ ^ python3.10 pruebas/pruebaCacheClima.py
/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py:13: DeprecationWarning: Not
d in a boolean context
    assert(cacheEjemplo.obtenerClima(None).__eq__(DatosClima(cacheEjemplo.realizarPetición(aeropuerto1)))) , "Datos JSON distintos"
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:14 ? :2 × )
→ ^
```

Figure 20: Salida

- Se busca el clima de un aeropuerto que no esta en el cache

```
aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
cacheEjemplo.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
    , "No se actualizaron los datos"
assert(cacheEjemplo.obtenerClima(aeropuerto2)\
    .__eq__(DatosClima(cacheEjemplo\
    .realizarPetición(aeropuerto1)))) , "Datos JSON distintos"
```

```

victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:14 ? :2 × )
→ python3.10 pruebas/pruebaCacheClima.py
/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py:13: DeprecationWarning: No
ed in a boolean context
assert(cacheEjemplo.obtenerClima(aeropuerto2).__eq__(DatosClima(cacheEjemplo.realizarPetición(aeropuerto1)))), "Datos JSON dis
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:14 ? :2 × )
→

```

Figure 21: Salida

- Por ultimo se prueba realizar peticion para un aeropuerto nulo

```

aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
cacheEjemplo.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
assert(cacheEjemplo.refrescarClima(aeropuerto1))\
    , "No se actualizaron los datos"
assert(cacheEjemplo.realizarPetición(None) != None)\
    , "Datos JSON nulos"

```

```

victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:14 ? :2 × )
→ python3.10 pruebas/pruebaCacheClima.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py", line 15, in <module>
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaCacheClima.py", line 14, in main
    assert(cacheEjemplo.realizarPetición(None) != None), "Datos JSON nulos"
AssertionError: Datos JSON nulos
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:14 ? :2 × )
→

```

Figure 22: Salida

5.4 Historial

5.4.1 Obtener numero de vuelo

Se busca que devuelva un dato de tipo entero

```

assert(type(obtenerNumeroDeVuelo()) == int)\
    , "Tipo de dato incorrecto"

```

Solo se realizo una prueba porque la funcion es sin parametros, salida:

```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:4 x )
→ python3.10 pruebas/pruebaHistorialVuelos.py
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:4 x )
→ python3.10 pruebas/pruebaHistorialVuelos.py
Por lo que al correr la siguiente salida es: Si ahora probamos con un
distinto a 3, sea 2
aeropuertoPrueba = aeropuerto.Aeropuerto('MEX', 10,
assert(aeropuertoPrueba.funcionHash(11) == 2)
```

Figure 23: Salida

5.4.2 Convertir a vuelo

Usaremos un objeto de la clase clima cache para poder acceder a la funcion de realizar peticion, ya que esta funcion toma datos JSON y los convierte en texto legible para vuelos

- Revisamos que lo que nos devuelva es una string ya que quiere decir que proceso bien los datos JSON de entrada.

```
aeropuerto1 = Aeropuerto('MEX', 19.4363, -99.0721)
cacheClima = CacheClima(11)
cacheClima.actualizarAPI("9d92b9e2262e46e5b34601d6f706cf43")
cacheClima.refrescarClima(aeropuerto1)
assert(type (convertirAVuelo(\
    cacheClima.realizarPeticion(aeropuerto1),\
    cacheClima.realizarPeticion(aeropuerto1))) == str)
```

```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:4 x )
→ python3.10 pruebas/pruebaHistorialVuelos.py
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:4 x )
→ python3.10 pruebas/pruebaHistorialVuelos.py
Por lo que al correr la siguiente salida es: Si ahora probamos con un
distinto a 3, sea 2
aeropuertoPrueba = aeropuerto.Aeropuerto('MEX', 10,
assert(aeropuertoPrueba.funcionHash(11) == 2)
```

Figure 24: Salida

- Se le manda dos datos nulos como ambos parametros

```
assert(type (convertirAVuelo(cacheClima.obtenerClima(None),\
                             cacheClima.obtenerClima(None))) == str)
```

```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:4 x )
→ python3.10 pruebas/pruebaHistorialVuelos.py
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:4 x )
→ python3.10 pruebas/pruebaHistorialVuelos.py
Por lo que al correr la siguiente salida es: Si ahora probamos con un
distinto a 3, sea 2
aeropuertoPrueba = aeropuerto.Aeropuerto('MEX', 10,
assert(aeropuertoPrueba.funcionHash(11) == 2)
```

Figure 25: Salida

En ambos caso tenemos la misma salida, una string que representa el vuelo

5.5 Lista de Aeropuertos

5.5.1 Procesar Vuelo

- Se revisa que si se procesa un vuelo de acapulco a mexico, se inserta de manera correcta en la lista de listas.

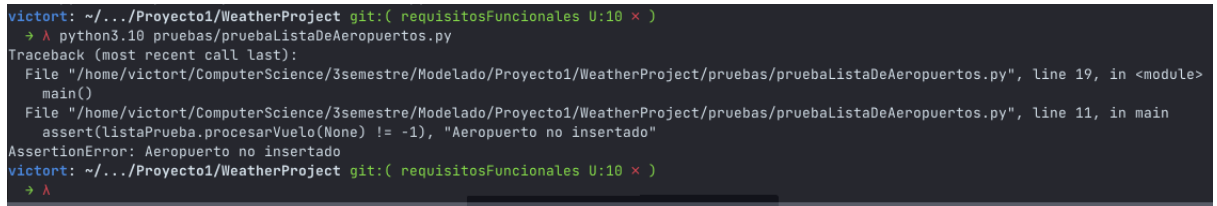
```
vuelo = ['ACA', 'MEX', 16.7571, -99.754, 19.4363, -99.0721]
assert(listaPrueba.procesarVuelo(vuelo) != -1)
```

```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:1 ? :2 x )
→ python3.10 pruebas/pruebaListaDeAeropuertos.py
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:2 x )
→ python3.10 pruebas/pruebaListaDeAeropuertos.py
auxiliar para probar que el formato de vuelo sea de la forma: (string,string,
float, float) para poder validar que el formato de vuelo es valido, pero
```

Figure 26: Salida

- Se revisa si se procesa un vuelo nulo

```
vuelo = None
assert(listaPrueba.procesarVuelo(vuelo) != -1)
```



```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:10 x )
→ A python3.10 pruebas/pruebaListaDeAeropuertos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaListaDeAeropuertos.py", line 19, in <module>
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProject/pruebas/pruebaListaDeAeropuertos.py", line 11, in main
    assert(listaPrueba.procesarVuelo(None) != -1), "Aeropuerto no insertado"
AssertionError: Aeropuerto no insertado
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:10 x )
→ A
```

Figure 27: Salida

5.5.2 Buscar Aeropuerto de Origen

Se toma el vuelo registrado de acapulco a mexico

- Si se proceso el vuelo de acapulco a mexico, se busca a acapulco dentro de la lista

```
aeropuerto1 = Aeropuerto('ACA', 16.7571, -99.754)
assert(listaPrueba.buscarAeropuertoOrigen(aeropuerto1) != -1)
```



```
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:1 ? :2 x )
→ A python3.10 pruebas/pruebaListaDeAeropuertos.pyivos
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:2 x )
→ A
```

Figure 28: Salida

- Se busca un aeropuerto nulo

```
aeropuerto1 = Aeropuerto('ACA', 16.7571, -99.754)
assert(listaPrueba.buscarAeropuertoOrigen(None) != -1)
```



```

victort: ~/.../Proyecto1/WeatherProject git:(requisitosFuncionales U:10 x )
→ ^ python3.10 pruebas/pruebaListaDeAeropuertos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProje
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProje
    assert(listaPrueba.buscarAeropuertoOrigen(None) != -1), "Aeropuerto inexis
AssertionError: Aeropuerto inexistente
victort: ~/.../Proyecto1/WeatherProject git:(requisitosFuncionales U:10 x )

```

Figure 29: Salida

- Se busca un aeropuerto de un vuelo que no hemos procesado

```

aeropuertoPrueba = Aeropuerto('MTY', 0 ,0)
assert(listaPrueba.buscarAeropuertoOrigen(aeropuertoPrueba) != -1)

```

```

victort: ~/.../Proyecto1/WeatherProject git:(requisitosFuncionales U:10 x )
→ ^ python3.10 pruebas/pruebaListaDeAeropuertos.py
Traceback (most recent call last):5.2 Prueba de Archivos
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProj
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProj
    assert(listaPrueba.buscarAeropuertoOrigen(aeropuertoPrueba) != -1), "Aerop
AssertionError: Aeropuerto inexistente
victort: ~/.../Proyecto1/WeatherProject git:(requisitosFuncionales U:10 x )

```

Figure 30: Salida

5.5.3 Insertar aeropuerto de origen

- Dado un aeropuerto de origen se busca que se haya insertado en un índice valido

```

assert(listaPrueba.insertarAeropuertoOrigen(aeropuertoPrueba) != -1)\
, "No se inserto el aeropuerto"

```

```

victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:1 ? :2 × )
→ ^ python3.10 pruebas/pruebaListaDeAeropuertos.py vos
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:2 × )
→ ^

```

Figure 31: Salida

- Se inserta un aeropuerto nulo

```

assert(listaPrueba.insertarAeropuertoOrigen(None) != -1)\
, "No se inserto el aeropuerto"

```

```

victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:10 × )
→ ^ python3.10 pruebas/pruebaListaDeAeropuertos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherPro
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherPro
    assert(listaPrueba.insertarAeropuertoOrigen(None) != -1), "No se inserto e
AssertionError: No se inserto el aeropuerto
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:10 × )

```

Figure 32: Salida

5.5.4 Buscar aeropuerto de destino

Se usa la misma lista con el vuelo de acapulco a mexico

- Dado el vuelo anterior de acapulco a mexico se busca a mexico dentro de la lista correspondiente a acapulco

```

assert(listaPrueba.buscarAeropuertoDestino(\
    listaPrueba.lista[aeropuerto1.funcionHash(11)]\
    , aeropuerto2.nombre)), "Aeropuerto inexistente"

```

```

victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:1 ? :2 x )
→ ^ python3.10 pruebas/pruebaListaDeAeropuertos.py vos
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:2 x )
→ ^

```

Figure 33: Salida

- Se busca un aeropuerto nulo en la lista de acapulco

```

assert(listaPrueba.buscarAeropuertoDestino(\
        listaPrueba.lista[aeropuerto1.funcionHash(11)]\
        , None)), "Aeropuerto inexistente"

```

```

victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:10 x )
→ ^ python3.10 pruebas/pruebaListaDeAeropuertos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProje
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProje
    assert(listaPrueba.buscarAeropuertoOrigen(None)!= -1), "Aeropuerto inexist
AssertionError: Aeropuerto inexistente
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:10 x )

```

Figure 34: Salida

- Se busca un aeropuerto en una lista vacia

```

assert(listaPrueba.buscarAeropuertoDestino([], aeropuerto2.nombre))\
        , "Aeropuerto inexistente"

```

```

victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:10 x )
→ ^ python3.10 pruebas/pruebaListaDeAeropuertos.py
Traceback (most recent call last):
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProje
    main()
  File "/home/victort/ComputerScience/3semestre/Modelado/Proyecto1/WeatherProje
    assert(listaPrueba.buscarAeropuertoDestino([], aeropuerto2.nombre)), "Aero
AssertionError: Aeropuerto inexistente
victort: ~/.../Proyecto1/WeatherProject git:( requisitosFuncionales U:10 x )

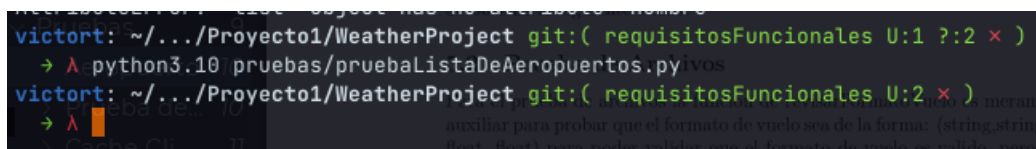
```

Figure 35: Salida

5.5.5 Revisar archivos

Se revisa que se hayan escrito los aeropuertos de manera correcta de acuerdo a los nombres de aeropuerto

```
assert(listaPrueba.revisarArchivosJSON()), "JSONs no escritos"
```



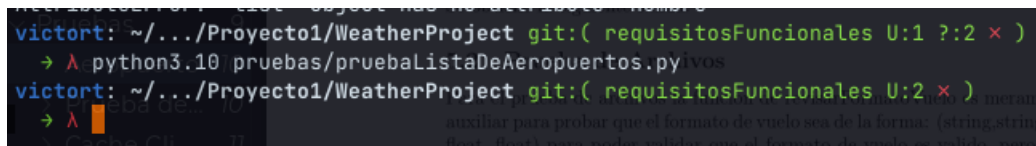
A terminal window showing a test run. The prompt is 'victort: ~/.../Proyecto1/WeatherProject'. The command is 'python3.10 pruebas/pruebaListaDeAeropuertos.py'. The output shows 'git:(requisitosFuncionales U:1 ? :2 x)' and a red 'x' indicating a failure. The test name is 'pruebaListaDeAeropuertos.py'.

Figure 36: Salida

5.5.6 Obtener nombres

Si solo se han insertado acapulco y monterrey como aeropuertos de origen se busca que se devuelvan solo los nombres de esos dos aeropuertos

```
assert(listaPrueba.obtenerNombres() == ['ACA', 'MTY'])
```



A terminal window showing a test run. The prompt is 'victort: ~/.../Proyecto1/WeatherProject'. The command is 'python3.10 pruebas/pruebaListaDeAeropuertos.py'. The output shows 'git:(requisitosFuncionales U:1 ? :2 x)' and a red 'x' indicating a failure. The test name is 'pruebaListaDeAeropuertos.py'.

Figure 37: Salida

6 Referencias

6.1 Web services

Nos apoyamos de <https://unprogramador.com/consumir-webservices-en-python-get-tutori> para poder hacer la request de los datos, urlopen que esta en urllib.request lo que hace es dado una url nos devuelve un archivo, hace la peticion al

servicio web, nosotros especificamos la URL que es una combinacion de longitud latitud y la API, agregamos lenguaje tambien, despues de que recibimos el archivo usamos read para leerlo y con loads lo que hacemos es transformarlo en una cadena formato json usando la libreria json, que es lo que devolvemos, manejamos los errores de peticiones con HTTPError y lo imprimimos, para saber que datos seleccionar de los multiples datos que estan contenidos en la cadena formato json, leimos la documentacion de <https://openweathermap.org/current> que es como convertimos datos en formato json a datos en formato string usando datos clima que basicamente es el concepto de arreglos.

6.2 Json

Para poder escribir un aeropuerto en un archivo JSON hacia falta codificarlo a JSON, por lo que nos basamos en <https://stackoverflow.com/questions/46005944/python-json-encoder-default-method> que lo que hace es crear una clase la cual devuelve los atributos de la clase en forma de lista, para poder devolverlos en formato json y poderlos escribir asi, si es instancia de aeropuerto devuelve la lista y sino la clase base devuelve el error que no son de la misma clase. Para poder ver como escribir en un archivo json nos basamos en el siguiente link <https://stackoverflow.com/questions/3768895/how-to-make-a-class-json-serializable> donde se especifica que se debe indicar la clase que codifica al aeropuerto y para ver la diferencia entre leer y escribir un archivo json nos apoyamos del siguiente archivo: <https://www.freecodecamp.org/espanol/news/python-leer-archivo-json-como-ca> donde nos dice que dump escribe sobre un archivo json y para que sirva la identacion y que load solo carga el contenido y con loads la string,

6.3 Tiempo

Para el manejo del tiempo nos basamos en <https://www.geeksforgeeks.org/python-datetime-timedelta-function/> para poder establecer la diferencia entre dos fechas que basicamente es restar dos .now() y con timedelta establecemos valor de la diferencia requerida, y con <https://www.programiz.com/python-programming/datetime/current-datetime> para poder ver como visualizar el tiempo en un formato conocido que es con strftime y estableciendo que queremos horas minutos y segundos.

6.4 Archivos

Usamos el <https://www.geeksforgeeks.org/python-os-path-isfile-method/> para saber como conocer si un archivo existe o no dado un a ruta determinada usando la libreria os y devuelve un booleano

Para leer el csv usamos el <https://www.programiz.com/python-programming/reading-csv-files> para conocer todo el contenido del csv que solo es llamar a reader de parte de la biblioteca csv para que iteremos sobre el contenido, por ello usamos next ya que es un iterador

Para el manejo de lectura escritura en archivos txt usamos el <https://programminghistorian.org/es/lecciones/trabajar-con-archivos-de-texto> donde tambien se nos dice como usar append, igual que todos los archivos solo hay que especificar bien la ruta y lo demas es parecido a los demas lenguajes de programacion.

6.5 Interfaz

La interfaz tal vez fue la que mas ayuda necesitamos de codigos de internet ya que a veces se vuelve muy especifica o asi y tkinter es una libreria muy extensa, para el displayMenu se uso <https://www.geeksforgeeks.org/tkinter-optionmenu-widget/> que entendimos porque se establece una stringvar que va a ser la variable de seleccion de la barra de menu, y luego en el caso de destino que cada vez que seleccionamos un aeropuerto de origen se redefine destino, se apoyo en <https://www.plus2net.com/python/tkinter-OptionMenu-proj1.php> para ver como limpiar todas las opciones y redefinir la lista de opciones con addcommand y con las etiquetas correspondientes, reiniciando igual la variable de seleccion a cadena vacia, es por ello que consultar se le asigna tambien el realizar la peticion de destino ya que al redefinir se pierde el comando relacionado, en el caso de leer textbox como lo requerimos en la API nos basamos en <https://recursospython.com/guias-y-manuales/caja-de-texto-entry-tkinter/> y en el caso del historial queremos mostrar una ventana emergente asi que utilizamos <https://www.geeksforgeeks.org/python-tkinter-messagebox-widget/> donde nos indica como crear un message box, pasandole el titulo y la informacion y para mostrar el historial usamos <https://es.acervolima.com/python-lectura-de-las-ultimas> ya que mostrar todo se nos complico mientras mas grande se hacia, lo que hace es con un iterador negativo lee las lineas desde el final del archivo y es 3*3 porque queremos los ultimos 3 vuelos y cada vuelo contiene 3 lineas, por

lo que son las ultimas 9 lineas. Para poder inicializar exitosamente la interfaz nos hicimos de ayuda de la documentacion: <https://docs.python.org/es/3/library/tkinter.html>, grid y place nos ayudan a colocar los botones y etiquetas a lo largo de toda la interfaz grafica, en init se inicializa la ventana con la ventana principal. MinSize es para un tamaño minimo decente.

6.6 Setup.py

Tamien cabe destacar que como usamos paquetes y en especial tenemos el directorio pruebas y el directorio src en donde en los archivos de pruebas usamos los paquetes de src, nos fue difícil como encontrar exportarlos hasta que vimos que con setup.py e init.py podíamos exportar y usar paquetes, setup.py y el porque lo encontramos aquí <https://betterprogramming.pub/sharing-code-using-a-setup-py-b6a596646532> donde se nos explica el que hace setup que le da un nombre al paquete y nos dice si queremos excluir alguno, en este caso no, y la importancia de init.py se encuentra aquí <https://stackoverflow.com/questions/448271/what-is-init-py-for> y básicamente es que con init.py nos damos cuenta que es un paquete el directorio y con setup.py establecemos cuales son paquetes y buscamos todos los archivos menos los especificados, en este caso ninguno y instalamos setup.py para que se añada a la librería de python y lo podemos usar. También nos apoyamos de: <https://www.educative.io/answers/what-is-setuppy> para ver como instalarlo y que hacia setup.py

6.7 Asserts

Nuestras primeras pruebas fueron muy rústicas, ahora usamos asserts que entendimos que son en base a lo explicado por esta página: <https://ellibrodepython.com/assert-python> donde nos dice que básicamente son funciones pruebas que buscan si lo que dado una función y su parametro devuelve es lo esperado que uno especifica, sino rompe el programa para indicar que no se cumplió lo esperado.

6.8 Documentacion

Para generar la documentacion en python se baso en la documentacion de la página: <https://docs.python.org/3/library/pydoc.html> que es la que

indica que banderas poner al correr la linea para generar la documentacion pertinente, para esto tienen que esta bien documentadas todas las clases

7 Costos

El costo de desarrollo por la beta seria de \$1000, el de desarrollo final seria un costo total ya de \$2000 y por darle mantenimiento habria un costo de \$500.