



Proyecto 01
Semestre 2021-1
Prof. José Galaviz Casas
Ayud. María Ximena Lezama
Modelado y programación

Kevin Ariel Merino Peña¹ Armando Abraham Aquino Chapa²
9 de octubre de 2020



1. Definición del problema

2. Análisis del problema

Análisis del problema y selección de la mejor alternativa. Al ir analizando los diversos rubros que puede abarcar el problema, se determinó 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.
- Manejar los posibles errores, como el exceso de numero 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.
-

3. Selección de la mejor alternativa

¹Número de cuenta 317031326

²Número de cuenta n

4. 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 FileNotFoundException:
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 cabezaera ← nombres de listas admitidas
9 entrada_cabecera ← read_headers(argumento[1])
10 if longitud(entrada_cabecera) no es igual a longitud(cabezera) then
11   muestra: ERROR El archivo csv debe tener los siguientes encabezados: origin, destination, origin_latitude,
    origin_longitude, destination_latitude, destination_longitude
12   salir
13 foreach cabeza in entrada_cabecera do
14   if cabeza no está en cabezera then
15     muestra: ERROR El archivo csv debe tener los siguientes encabezados: origin, destination,
    origin_latitude, origin_longitude, destination_latitude, destination_longitude
16   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

```

Weather.py

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 localtimezone ← get_localzone()()
3 local_time ← fromtimestamp(flotante, localtimezone)
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
```

5. 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

5.1. 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 unx programadorx, 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, así cada uno recibiría \$950 por sus horas invertidas.

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