Documentación

Elaborado por:

- Sara Ivonne Sánchez Franco
- Jorge Uriel Pérez Romero

Resumen

El nombre del programa es staDb_sifs.py y su función es tomar dos archivos tipo ASCII con un determinado formato, se extrae la información y lo reescribe en un nuevo archivo cuyo nombre es staDb_sifs de extensión ASCII con un determinado formato.

Características

El programa fue codificado en python empleando el paradigma de programación funcional, de esta manera no es necesario reescribir nuevamente una parte del código, ni ser tan cuidadoso al nombrar las variables, además de que la codificación se vuelve más ordenada.

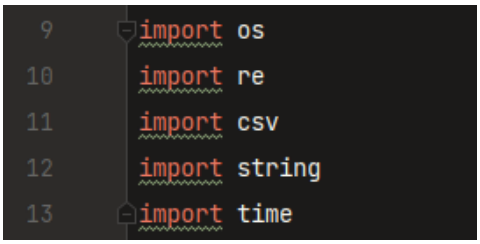
Descripción

El código esta comentado en su mayoría, lamentablemente sin acentos porque el teclado con el que se escribió se encuentra en inglés y cambiar de idioma se vuelve desesperante.

Código

Debemos aclarar que el programa debe estar dentro de la carpeta donde se tienen los archivos que se planea usar. También se recomienda que el lector use un editor de texto que muestre el número de línea porque se hace referencia tanto a los archivos ASCII así como al código.

Comenzamos por importar los módulos:



El módulo *os* nos permite ejecutar comandos del sistema operativo (SO) desde el script[1]. *re* es un módulo que proporciona operaciones de coincidencia de expresiones regulares. Tanto los patrones como las cadenas de texto a buscar pueden ser cadenas de Unicode (str) así como cadenas de 8 bits (bytes).[2]

[1] os — Interfaces misceláneas del sistema operativo — documentación de Python - 3.10.9
[2] re — Regular expression operations — Python 3.11.1 documentation

Nota: Los pies de página son las documentaciones de las bibliotecas utilizadas en este código y para acceder al link presione ctrl + clic izquierdo.

El módulo *csv* nos permite implementar clases para leer y escribir datos tabulare en formato CSV. Permite a los programadores decir "*escribe estos datos en el formato preferido por Excel*" o *"lee datos de este archivo que fue generado por Excel"* sin conocer los detalles precisos del formato CSV usado por Excel. Los programadores también pueden describir los formatos CSV entendidos por otras aplicaciones o definir sus propios formatos CSV para fines particulares[3].

El módulo *string* permite hacer operaciones de cadenas de caracteres.[4]

El módulo *time* proporciona varias funciones relacionadas con el tiempo. Para la funcionalidad relacionada[5].

```
14  #------------------------------------------------------------
15  # Se inicia el tiempo de medida del arraque del programa
16  ini = time.time()
17  # Limpiando pantalla
18  # Cambia dependiendo del SO
19  os.system ("cls")
20  #------------------------------------------------------------
```

En la línea 16 se inicia a medir el tiempo de arranque del programa con el fin de que se cuente el tiempo de ejecución, no es relevante para los resultados esperados solo es algo que ayudara al usuario que el programa se ha ejecutado satisfactoriamente.

La línea 19 nos permite limpiar la pantalla del centro de comando, en la imagen se muestra el comando para el SO Windows, si se quiere ejecutar en Linux; *"cls"* debe ser cambiado por "*clear*".

```
27  def EliminarAst(Nam_Neue_Arch, Nam_Alt_Arch):
28      file1 = open(Nam_Neue_Arch, 'r')
29      file2 = open(Nam_Alt_Arch, 'w')
30      for line in file1.readlines():
31          x = re.findall("\*", line)
32          if not x:
33              file2.write(line)
34      file1.close()
35      file2.close()
```

[3] csv — Lectura y escritura de archivos CSV — documentación de Python - 3.8.14
[4] string — Operaciones comunes de cadena de caracteres — documentación de Python - 3.11.1
[5] time — Acceso a tiempo y conversiones — documentación de Python - 3.11.1

Comenzando por la primera función (EliminarAst) cuyos parámetros son dos nombres de dos archivos. *Nam_Neue_Arch* es únicamente el nombre del archivo de salida y *Nam-_Alt_Arch* es el archivo existente en el directorio del usuario. Las líneas 28 y 29 permite leer (*'r'*), para crear y escribir sobre un nuevo fichero se usa el argumento (*'w'*).

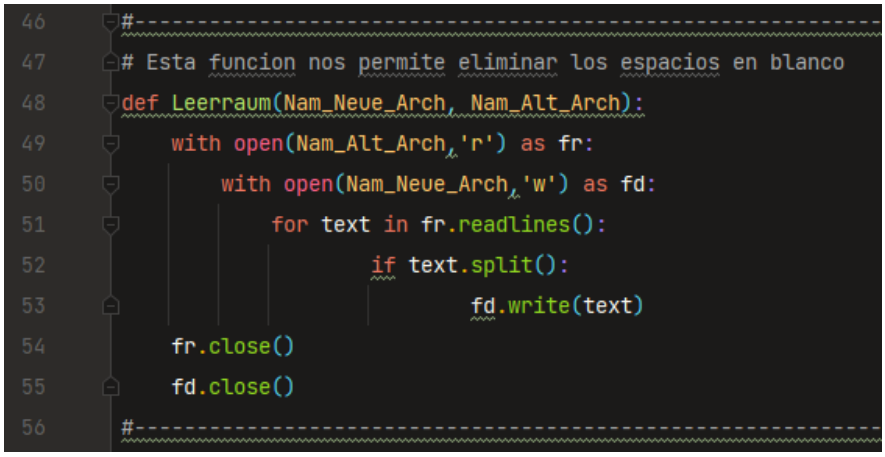
Las líneas 30 a la 33 permite leer el archivo ASCII existente en la carpeta del usuario línea por línea y lo escribe en el nuevo archivo ASCII, sin embargo, gracias al módulo *re* si una línea contiene un asterisco (*) lo salta. Finalmente, en las líneas 34 y 35 se cierran ambos archivos; el creado y el abierto. Cabe destacar que cerrar los ficheros es muy importante porque de lo contrario cuando el código se ejecute marcará errores de compilación.

```
36  #-----------------------------------------------
37  def EliminarGato(Nam_Neue_Arch, Nam_Alt_Arch):
38      file1 = open(Nam_Neue_Arch, 'r')
39      file2 = open(Nam_Alt_Arch, 'w')
40      for line in file1.readlines():
41          x = re.findall("#", line)
42          if not x:
43              file2.write(line)
44      file1.close()
45      file2.close()
46  #-----------------------------------------------
```

La función EliminarGato funciona de la misma manera que EliminarAst, sin embargo, elimina las líneas que contiene un #, por eso es importante que los comentarios se coloquen en una línea diferente y no sobre la que contiene información, como es el caso del archivo sitevecs en las líneas 1329 y 1330 (si se modifica esto del archivo sitevecs el programa aún funciona).

Se debe aclarar que se han hecho dos funciones distintas para eliminar # y * porque si se agrega *y = re.findall("#", line)* antes de la línea 32 y *and* en la condición *if* es insuficiente debido a que el módulo *re* no era capaz de reconocer dos caracteres simultáneamente.

La siguiente función es Leerraum que nos permite leer un archivo ASCII y escribir en otro, pero saltando las filas que se encuentran vacías.



```
46  #-------------------------------------------------------------------
47  # Esta funcion nos permite eliminar los espacios en blanco
48  def Leerraum(Nam_Neue_Arch, Nam_Alt_Arch):
49      with open(Nam_Alt_Arch,'r') as fr:
50          with open(Nam_Neue_Arch,'w') as fd:
51              for text in fr.readlines():
52                      if text.split():
53                              fd.write(text)
54      fr.close()
55      fd.close()
56  #-------------------------------------------------------------------
```

Las líneas 49 y 50 permite leer un fichero (“r”) o si se desea crear y escribir sobre el mismo se usa (“w”). Se lee cada fila del archivero y si se encuentra vacío lo salta y escribe a continuación la siguiente fila que contenga caracteres. Nuevamente los ficheros se cierran.

La siguiente función se llama ASCIIzuCSV, y como su nombre lo indica transforma un archivo ASCII a un CSV separado por comas, incluso si se abre en algún programa como Excel, pareciera tener el formato de un xlsx.

```
72  #-------------------------------------------------------------------------------------------------------------
73  ''' Esta función solo transforma un archivo ASCII a CSV
74  Esta funcion ha sido elaborada para que sea mas facil extraer determinados datos'''
75  def ASCIIzuCSV(Alte_Archive, Archive_csv):
76      # Esta parte ayuda a que cada carte separada por espacios quede dentro de una celda
77      out = open(Archive_csv, 'w', newline = '')
78      csv_writer = csv.writer(out, dialect='excel')
79      f = open(Alte_Archive, 'r')
80      for line in f.readlines():
81          line = line.replace ('\ t', ',') # Remplazando cada espacio por una coma
82          lista = line.split () #Convierta la cadena en una lista, para que pueda escribir csv por celda
83          csv_writer.writerow(lista)
84      f.close()
85      out.close()
86  #-------------------------------------------------------------------------------------------------------------
```

Se decidió transformar el ASCII a CSV porque es más fácil extraer la información por columnas (al menos en la experiencia de quién lo programo).

En las líneas 77 y 78 se crea un archivo de tipo csv y se indica que se deberá escribir en él. Con la línea 79 se abre el archivo ASCII introducido en los argumentos de la función, mismo que solamente debe leerse. De las líneas 80 a la 83; la función lee cada fila del archivo, remplaza los espacios por una coma para que cuando se transforme en una lista (línea 82) sea tomado como un elemento de una lista.

Finalmente, las listas se van escribiendo sobre el archivo csv creado anteriormente dentro de la misma función y se cierran ambos archivos cuando se haya completado las instrucciones descritas anteriormente.

De la función Spalte lee un archivo csv y toma una columna determinada transformándola en una lista. Se han hecho dos funciones, con la pequeña deferencia que Spalte_II agrega filas para que todas las columnas tengan el mismo tamaño.

Comenzando con Spalte de la línea 89 a la 92 se abre el archivo tipo csv y su contenido se guarda en una lista tomando en cuenta que cada espacio es una coma. De las líneas 96 a la 105 se lee cada elemento de la lista creada anteriormente y hasta cada salto de renglón se indica fin de una lista. En la línea 105 se escribe el valor del renglón que se desea extraer del archivo y lo retornamos dentro de una variable que guarda la columna como una lista. Nuevamente se cierra el archivo abierto.

Como se mencionaba anteriormente, se realizó otra función que permita seleccionar otros renglones porque Spalte no cuenta con la capacidad. Spalte_II agrega elementos vacíos para que todas las filas cuenten con el mismo número de columnas.

Para entender mejor el párrafo anterior el usuario debe dirigirse al archivo sitevecs, donde cada conjunto de caracteres es un elemento y que es distinto para cada fila, por ejemplo, si queremos la columna 4 nos marcara que estamos fuera de rango porque algunas filas solo cuentan con 3 elementos.

```
88  def Spalte(Archive, Position):
89      with open(Archive, newline = '') as f:
90          Stat = []
91          for line in f:
92              Stat.append(line)
93      # /-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/-/
94      # Convirtiendo cada renglon en una lista para despues seleccionar un determinado renglon
95      # Leyendo un renglon del archivo que hemos cargado
96      Neuen = []
97      N = []
98      for i in range(len(Stat)):
99          Neuen = 0
100         # Primero se lee cada elemento de la lista
101         NeueListe = Stat[i]
102         # Se transforma cada String en una Lista
103         Neuen = NeueListe.split(',')
104         # Se guarda la posicion especifica que se da en la funcion
105         N.append(Neuen[Position])
106     f.close()
107     return N
108 #----------------------------------------------------------------------------------------------------
```

```python
def Spalte_II(Archive, Saeule, n):
    le_archive = []
    # Esta parte nos permite abrir el archivo csv y leer cada linea como si fuera una lista
    with open(Archive) as File:
        reader = csv.reader(File, delimiter = ',', quotechar = ',', quoting = csv.QUOTE_MINIMAL)
        for row in reader:
            if len(row)<n: # n es la fila faltante para que sea del mismo tamano que el resto
                row.extend([' ']*6)
                le_archive.append(row)
            else:
                le_archive.append(row)
    ''' Finalemente la lista le_archive contiene el documento completo, los espacios vacios, como se mencionaba
     anteriormente fueron rellenados con espacios vacios para que asi pueda ser tomado cualquier columan de la lista
     y no marque error como la definicion spaten.
    A continuacion vamos a cdificar como elegir solo una columna teniendo como resultado una multilista que en el valor
    de return tenemos un resultado de tipo lista'''
    rubrik = []
    for i in range(len(le_archive)):
        rubrik.append(le_archive[i][Saeule])
    return rubrik, le_archive
```

De las líneas 114 a la 123 abrimos el archivo de tipo csv y cada fila del archivo se lee como una lista, sin embargo, se agregan más columnas (n: debe ser del tamaño de la fila más grande contenida en el archivo) que estarán vacías. Después se toma el valor de la columna que se desea extraer y se retorna las variables como se muestra en las líneas 129 a la 132.

```python
def Archive(Archive_Eingang, Archive_csv):
    Arch_A = 'stalocs_A'
    Arch_B = 'stalocs_B'
    Arch_C = 'stalocs_C'
    Leerraum(Arch_A, Archive_Eingang)
    EliminarAst(Arch_A, Arch_B)
    EliminarGato(Arch_B, Arch_C)
    ASCIIzuCSV(Arch_C, Archive_csv)
    # Este comando cambia dependiendo del SO
    os.remove(Arch_A)
    os.remove(Arch_B)
    os.remove(Arch_C)
```

En la función Achive se toman varias funciones descritas anteriormente. En esta función se introduce el nombre del archivo con el que contamos y el nombre del archivo csv que deseamos de salida. Los valores designados a las variables Arch_A, Arch_B, Arch_C son los nombres de los archivos que se van a crear y que no son visibles para el usuario puesto que se eliminan después de haberse utilizado.

En la línea 142 se ejecuta la función Leerrraum que nos permite quitar las filas vacías. Después se eliminan los asteriscos y los #, posteriormente, el mismo archivo se transforma en un csv que es con el que trabajaremos. Finamente, como se mencionaba anteriormente, se eliminan los archivos ASCII creados, dentro de la misma función.

Hasta aquí termina una parte del código y lo que se presenta a continuación es la extracción de la información por cada archivo porque cada uno cuenta con diferentes formatos e información, sin embargo, al ser funciones más grandes se ha divido en partes para que sea más digerible tanto para los autores como para los usuarios.

Para el archivo sitevecs se realizó una función con el mismo nombre al que sus argumentos se comprenden de los archivos sitevecs y sitelocs.

```
157 def SiteVecs(Arch_C, Arch_D):
158     # Se extraen las columnas de las estaciones
159     Sta_sts = Spalte(Arch_C, 0)
160     Sta_sis = Spalte(Arch_D, 0)
161     # Es el tamagno de las listas extraidas
162     Grosse_Sta_sts = len(Sta_sts)
163     Grosse_Sta_sis = len(Sta_sis)
```

Con la ayuda de la función Spalte extraemos la columna cero que, para ambos archivos, es la que contiene el nombre de las estaciones, las listas las guardamos en una variable al igual que el tamaño de las listas extraídas.

```
167     Gemeinsam = []
168     for i in range(Grosse_Sta_sts):
169         for j in range(Grosse_Sta_sis):
170             if Sta_sis[j] in Sta_sts[i]:
171                 Gemeinsam.append(Sta_sis[j])
```

Con las líneas 167 a la 171 se toman las dos listas que cuentan con la clave de las estaciones y se comparan, es decir, se busca que ambos archivos cuenten con las mismas estaciones y se guarden las que se tengan en común. Sin embargo, cuando los elementos de encuentran en ambas listas se duplican.

```
176     Zusammen = []
177     for item in Gemeinsam:
178         if item not in Zusammen:
179             Zusammen.append(item)
180     # ---------------------------------------
181     i = 0
182     together = []
183     for i in range(len(Sta_sis)):
184         if Sta_sis[i] in Zusammen:
185             together.append(Sta_sis[i])
```

Para evitar lo anterior mencionado en las líneas 177 a la 180 se eliminan los elementos repetidos y con las líneas 181 a 185 se toma la lista extraída del archivo sitevecs y se extraen las estaciones con las que ambos archivos cuentan.

```
191     i = 0
192     Stati = []
193     Buch = []
194     Datum = []
195     for i in range(len(Sta_sis)):
196         if Sta_sis[i] in together:
197             Stati.append(Sta_sis[i])
198             Buch.append(Sta_sis[i])
199         else:
200             Stati.append('*')
201             Datum.append(Sta_sis[i])
202     # Se calcula los idices de los valores que no corresponden a las estaciones para el resto de la informacion
203     j = 0
204     Skale = []
205     for j in range(len(Datum)):
206         if Datum[j] in Sta_sis:
207             Ska = Sta_sis.index(Datum[j])
208             Skale.append(Ska)
```

Las líneas 191 a la 201 se crean tres listas; Stati corresponde a la lista que se ha extraído del archivo sitevecs comparado con la lista de las estaciones que ambos archivos proporcionados contienen, Buch es la lista que contiene solamente el nombre de las estaciones contenidos en ambos archivos y Datum es la lista que solo contiene las fechas sin haber escrito el nombre de las estaciones.

Las líneas 203 a la 108 se calcula los índices en los que se encuentra las fechas dentro de la lista que se ha extraído directamente del archivo sitevecs porque las estaciones existentes en ambos archivos se encuentran en el archivo mencionado anteriormente.

```
216     Sta_sis_1 = Sta_sis[:249]
217     Sta_sis_2 = Sta_sis[249:]
218     Buch_1 = Buch[:78]
219     Buch_2 = Buch[78:]
```

Ocurrió un problema mientras se codificaba y es que un par de estaciones aparece repetido dos veces en líneas diferentes como si se trataran de dos estaciones diferentes (SLAC y SLCR) por lo tanto las listas que se ha extraído directamente del archivo sitevecs y Buch se han dividido en dos

Nota: Se divide porque conocemos la ubicación en la que se encuentran las estaciones repetidas.

```
224     i = 0
225     j = 0
226     St_1 = []
227     St_2 = []
228     for i in range(len(Buch_1)):
229         Sta_1 = Sta_sis_1.index(Buch_1[i])
230         St_1.append(Sta_1+1)
231     for j in range(len(Buch_2)):
232         Sta_2 = Sta_sis_2.index(Buch_2[j])
233         St_2.append(Sta_2 + St_1[-1] + 2)
```

En las líneas 224 y 233 se extraen los índices de las estaciones de las listas que se ha divido y se le suma una unidad, es decir se desplaza una posición y tomando en cuenta que la siguiente posición había información sobre las fechas por lo que se recomienda partir la lista una posición antes de que aparezca el nombre de una estación.

```
247     St_1.extend(St_2)
248     St_1.append(St_1[-1]+2)
```

En las líneas 247 y 248 se muestra cómo se unen ambas listas, sin embargo, a la segunda se le ha sumado el índice del último valor más uno de la primera lista (porque parece una fecha) para que haya continuidad en la segunda parte, y cuando se unan muestre la posición real recorrida en la que aparecen las estaciones.

Después de tener una lista con los índices en los que se encuentran las estaciones ahora se busca calcular cuantas veces se repite las estaciones, es decir, repetir las estaciones las veces necesarias respecto a sus fechas correspondientes.

```
254     i = 0
255     j = 0
256     Station = []
257     for i in range(len(St_1)-1):
258         Zahl = ((St_1[i+1] - St_1[i])-1)
259         Station.extend([Buch[i]] * Zahl)
```

Desde las líneas 254 a la 259 se calcula las veces que se debe repetir las estaciones y se multiplica el número de veces que debe aparecer la estación por su estación correspondiente.

Hasta esta parte ya tenemos la lista de las fechas y la lista de las estaciones que es, tal vez la parte más difícil de entender.

```
265     ANT_0 = Spalte_II(Arch_D, 5, 9)[0]
266     ANT_1 = Spalte_II(Arch_D, 6, 9)[0]
267     ANT_2 = Spalte_II(Arch_D, 7, 9)[0]
268     ANT_3 = Spalte_II(Arch_D, 8, 9)[0]
```

A continuación, se extrae las columnas que contienen las antenas empleando la función Spalte_II.

```
271     i = 0
272     ANT_Zussamen = []
273     for i in range(len(ANT_1)):
274         ANT_Wor = ANT_0[i]
275         ANT_Zussamen.append(ANT_Wor)
276     j = 0
277     ANT = []
278     for j in range(len(Skale)):
279         s = Skale[j]
280         ant = ANT_Zussamen[s]
281         ANT.append(ant)
```

Las líneas 271 a la 275 es para hacer la lista de las antenas, se hizo de esta manera porque puede que las antenas se encuentren en más de una columna por lo que en la línea 274 se pueden agregar más columnas y se guarda como un elemento de una lista en caso de ser necesario.

En las líneas 276 a la 281 se extraen los elementos en los que se encuentra las antenas. Debemos recordar que Skale es la lista de los índices en los que no hay una estación,

esto siempre se cumple por el formato del archivo que se ha extraído de manera indirecta cuando se extraía las listas de las fechas, las estaciones y los índices de las estaciones.

```
287     i = 0
288     ANT_Zus= []
289     for i in range(len(ANT_1)):
290         ANT_Zus.append(ANT_2[i] + ' ' + ANT_3[i])
291     j = 0
292     ANT_type = []
293     for j in range(len(Skale)):
294         s = Skale[j]
295         ant_type = ANT_Zus[s]
296         ANT_type.append(ant_type)
```

Se hace un proceso similar para extraer el tipo de antena porque cumple con el mismo formato que el nombre de la antena. Se puede hacer una función que cumpla con esta tarea, sin embargo, no se hizo en este caso debido que como se puede ver en la línea 290 se introducen dos listas que deben unirse, sin embargo, la forma en la que se extrae la información es la misma.

El mismo proceso se hace con para extraer la localización de las antenas y las horas, también se retornan las listas extraídas como se muestra en las líneas 299 a la 327.

```
299     E = Spalte_II(Arch_D, 2, 9)[0]
300     N = Spalte_II(Arch_D, 3, 9)[0]
301     V = Spalte_II(Arch_D, 4, 9)[0]
302     i = 0
303     s = 0
304     e = []
305     n = []
306     v = []
307     for i in range(len(Skale)):
308         s = Skale[i]
309         Standpunkt_e = E[s]
310         Standpunkt_n = N[s]
311         Standpunkt_v = V[s]
312         e.append(Standpunkt_e)
313         n.append(Standpunkt_n)
314         v.append(Standpunkt_v)
```

```
315     # ------------------------------------------------------------------
316     # Ahora se extrae el RX de la misma manera que los casos anteriores
317     rx = Spalte_II(Arch_D, 1, 9)[0]
318     i = 0
319     s = 0
320     RX = []
321     for i in range(len(Skale)):
322         s = Skale[i]
323         r = rx[s]
324         RX.append(r)
325     # ------------------------------------------------------------------
326     # Se extrae las listas con la informacion que se busca
327     return Station, Datum, ANT, e, n, v, RX, Skale, Buch, ANT_type
```

Para extraer las columnas del documento stalocs es similar a las anteriores, sin embargo, gracias a su formato y a la definición Spalte_II es más fácil extraer la información como se muestra en las líneas 333 a la 342:

```
333 def StaLocs(Arch_C):
334     # Extrayendo los monumentos
335     Denkmal = Spalte_II(Arch_C, 1, 4)[0]
336     # Extrayendo las estaciones
337     Station = Spalte_II(Arch_C, 0, 4)[0]
338     # Extrayendo la posicion de las antenas
339     x = Spalte_II(Arch_C, 3, 4)[0]
340     y = Spalte_II(Arch_C, 4, 4)[0]
341     z = Spalte_II(Arch_C, 5, 9)[0]
342     return Station, Denkmal, x, y, z,
```

Cabe mencionar que dentro del archivo stalocs en la línea 198 no se cuenta con la información de las coordenadas de la estación correspondiente.

Posteriormente, se hicieron las definiciones con cada palabra clave que se requiere para hacer el archivo staDb_sifs. Se ha codificado de esta manera para tener un mejor orden (de acuerdo con el autor).

Las siguientes funciones permiten dar formato y guárdalo en forma de lista para después escribirlo en un archivero. Cada palabra clave se pide ha sido escrito en funciones también para tener más orden y poder modificar el código (si es necesario, lo cual no lo es si los archivos otorgados tienen el mismo formato).

La primera función es la de los comentarios (KEYBOARDS) en el que no hay argumentos de entrada solo es una lista cuyos elementos son de tipo string.

```
350     def KEYBOARDS():
351         key = ['KEYBOARDS: ' + 'ID' + ' STATE' + ' ANT' + ' RX']
352         return key
```

Debe tenerse cuidado porque es el orden en que van a aparecer las palabras clave a lo largo de todo el archivo de salida.

La siguiente función es para la palabra clave ID en el que se llama la función SiteVecs para extraer las columnas del archivo csv. Una vez cumplida la tarea anterior se da el formato deseado como puede verse en las líneas 358 a la 362.

```
354 def ID(Arch_C, Arch_D):
355     # Se extraen las estaciones del archivo Stalocs y se ordenan alfabeticamente para que no haya problema en acomodar
356     # todo nuevamente
357     Buch = SiteVecs(Arch_C, Arch_D)[8]
358     i = 0
359     station = []
360     for i in range(len(Buch)):
361         station.append(str(Buch[i] + ' ID Mexico'))
362     Ort = sorted(station)
363     return Ort
```

Cabe mencionar que se ha acomodado las estaciones en orden alfabético para que cada apartado tenga el mismo orden.

Después se hizo la función de STATE, que es la más difícil comparadas con el resto de las palabras clave.

```
365 def STATE (Arch_C, Arch_D):
366     # Se extrae la informacion del archivo StaLocs
367     station_L, monument, x, y, z = StaLocs(Arch_C)
368     station_V, date, ant, e, n, v, rx, skale, buch, ant_type = SiteVecs(Arch_C, Arch_D)
369     # ----------------------------------------------------------------------------------------------------------------
370     i, j = 0, 0
371     state = []
372     for i in range(len(station_L)):
373         for j in range(len(station_V)):
374             # Solo para las estaciones en las que se coicide ambos archivos se toma en cuenta
375             if sorted(station_V[j]) == sorted(station_L[i]):
376                 state.append(station_V[j] + ' STATE ' + date[j] + ' ' + rx[j] + ' ' + x[i] + ' ' + y[i] + ' ' + z[i] + ' 0')
377     State = sorted(state)
378     return State
379 # ----------------------------------------------------------------------------------------------------------------
```

En las líneas 367 y 368 se muestra como se extrae la información de ambos archivos proporcionados después de haberse transformado a un archivo csv. Se extrae de esta manera porque es una forma “*más eficiente*” y es muy útil cuando se quieren llamar todos

los valores de retorno. En las líneas 370 a la 378 se da el formato deseado de como se va a escribir la información en el archivo staDb_sifs.

Como podemos ver en las líneas 372 y 373 se ha realizado un doble bucle porque necesitamos información de ambos archivos, porque nos pide colocar las coordenadas de la estación que vienen en el archivo stalocs, y el tiempo que viene en el documento sitevecs. Cabe destacar que la velocidad es igual a cero en este caso debido a que no ninguno de los dos archivos cuenta con dicha información.

```
380 def ANT(Arch_C, Arch_D):
381     # Extraemos la informacion del archivo sitevecs
382     station, date, ant, e, n, v, rx, skale, buch, ant_type = SiteVecs(Arch_C, Arch_D)
383     # Se guarda toda la informacion en forma de diccionario
384     # Primero quitamos el signo de '+', '_' y de '-' de la informacion que se ha extraido de las antenas porque lo que sigue
385     #despues de esos simbolos son el radome
386     a = []
387     for i in range(len(ant)):
388         a.append(ant[i].replace("+", " "))
389     an = []
390     for j in range(len(a)):
391         an.append(a[j].replace("-", " "))
392     Ant = []
393     i = 0
394     for i in range(len(an)):
395         Ant.append(an[i].replace("_", " "))
396     i, j = 0, 0
397     ANT = []
398     for i in range(len(station)):
399         for j in range(len(buch)):
400             if sorted(station[i]) == sorted(buch[j]):
401                 ANT.append(station[i] + ' ANT ' + date[i] + ' ' + rx[i] + ' ' + Ant[i] + ' ' + e[i] + ' ' + n[i] + ' ' + v[i])
402     ant = sorted(ANT)
```

```
398     for i in range(len(station)):
399         for j in range(len(buch)):
400             if sorted(station[i]) == sorted(buch[j]):
401                 ANT.append(station[i] + ' ANT ' + date[i] + ' ' + rx[i] + ' ' + Ant[i] + ' ' + e[i] + ' ' + n[i] + ' ' + v[i])
402     ant = sorted(ANT)
403     return ant
```

La siguiente función es para escribir ANT. Se hace de manera similar cuya única diferencia es que a las columnas extraídas se le eliminan caracteres especiales como “+”, “_ “y “-“ porque la información que viene después de los caracteres mencionados anteriormente debe ir separados por un espacio de acuerdo con los ejemplos del documento StaDbFormat.dox. Para la palabra ANT nos pide, al igual que las demás, la clave de la estación, una palabra clave, también la fecha, hora, modelo de la antena y su posición. Finalmente retornamos la lista.

La siguiente función es RX que afortunadamente es sencilla de construir porque toda la información se encuentra en el archivo sivecs. Cabe destacar que su formación fue similar que la función ANT a excepción que no se elimina ningún carácter especial contenido en los elementos de la lista.

```
405 def RX(Arch_C, Arch_D):
406     # Extraemos la informacion del archivo sitevecs
407     station, date, ant, e, n, v, rx, skale, buch, ant_type = SiteVecs(Arch_C, Arch_D)
408     RX = []
409     for i in range(len(station)):
410         for j in range(len(buch)):
411             if sorted(station[i]) == sorted(buch[j]):
412                 RX.append(station[i] + ' RX ' + date[i] + ' ' + rx[i] + ' ' + ant_type[i])
413     Rx = sorted(RX)
414     return Rx
```

En resumen, lo que se ha hecho hasta ahora, es convertir el archivo ASCII y convertirlo en un csv, extraerse la información y dar un formato establecido, por lo que resta escribirlo en un fichero. Para cumplir dicha tarea se ha codificado otra función llamada Schreiben y sus argumentos de entrada son dos archivos csv.

```
417 def Schreiben(Arch_C, Arch_D):
418     key = KEYBOARDS()
419     id = ID(Arch_C, Arch_D)
420     state = STATE(Arch_C, Arch_D)
421     ant = ANT(Arch_C, Arch_D)
422     rx = RX(Arch_C, Arch_D)
423     # Abriendo el archivo
424     with open('staDb_sifs', 'w') as f:
425         for item in key:
426             f.write("%s\n" % item)
427         item = 0
428         for item in id:
429             f.write("%s\n" % item)
430         item = 0
431         for item in state:
432             f.write("%s\n" % item)
433         item = 0
434         for item in ant:
435             f.write("%s\n" % item)
436         item = 0
437         for item in rx:
438             f.write("%s\n" % item)
439     f.close()
```

En la función Schreiben se llaman las funciones cuyos nombres hacen referencia a cada sección del archivo que se va a construir y sus argumentos son los archivos de tipo csv. En la línea 424 se crea un fichero con el nombre staDb_sifs y sobre él se comienza a

escribir. En el código se puede ver varios ciclos, cuya tarea es escribir sobre el fichero cada elemento de una lista. Finalmente, en la línea 439 se cierra el fichero creado.

```
441  def staDb_sifs(Arch_A, Arch_B):
442      # Esto es por default
443      Arch_C = 'CSVstalocs.csv'
444      Arch_D = 'CSVsitevecs.csv'
445      Archive(Arch_A, Arch_C)
446      Archive(Arch_B, Arch_D)
447      Schreiben(Arch_C, Arch_D)
448      os.remove('CSVstalocs.csv')
449      os.remove('CSVsitevecs.csv')
```

En la función staDb_sifs los argumentos son los dos archivos de tipo ASCII, después en las líneas 443 y 444 vemos los nombres de los archivos csv, estos aun no existen, solo es el nombre que se ha proporcionado. Llamamos la función Archive que nos permite transformar un documento tipo ASCII a un csv y se llama la función Schreiben donde se extrae la información de los archivos csv y se le da un formato determinado. Finalmente se elimina ambos archivos csv para que el usuario solo vea el archivo de salida.

NOTA: dentro de la función Schreiben se emplean las funciones SiteVecs y SiteLocs.

Se ejecutan funciones dentro de las funciones con el fin de que el usuario solo modifique las líneas 460 y 461 que son los nombres de los archivos ubicados en su directorio donde también debe encuentrarse el código.

```
456  ##################################################################
457  ##################################################################
458  ### -----------   Es el progrma pincipal   -------------- ###
459  ###                                                            ###
460  Arch_A = 'stalocs'                                             ###
461  Arch_B = 'sitevecs'                                            ###
462  staDb_sifs(Arch_A, Arch_B)                                     ###
463  ###                                                            ###
464  #-------------------------------------------------------------###
465  ##################################################################
466  ##################################################################
467  #-----------------------------------------------------------------
468  fin = time.time()
469  print('Tiempo en ejecutar el programa: ', fin-ini, 'seg')
```

En la línea 468 se termina de contar el tiempo y la línea 469 se imprime el tiempo de ejecución del código para que el usuario sepa que el programa termino de compilar.

Cabe resaltar que el lector puede codificar todas las capturas de pantalla colocadas en este documento y obtendrá el mismo resultado que el autor.