Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Lenguajes Formales y de Programación

Laboratorio de Lenguajes Formales

PRÁCTICA ÚNICA

Ana Denisse Figueroa Marroquín

Carnet: 201801143

Fecha de Entrega: 17/02/2021

MANUAL TÉCNICO

El contenido de este manual tiene información acerca de cada uno de los eventos creados y utilizados, explicados con detalle. Se creó 6 eventos y una clase en total. Se utilizó un método de ordenamiento llamado SelectionSort entre otros.

1. Cargar Archivo:

En el evento **def cargar_archivo(),** se puede encontrar que se utilizó una función ask open file, la cual específicamente sirve para abrir un archivo por medio de una ventana emergente a través de consola. Se utilizó ask open file name para obtener la dirección y nombre del archivo, con esa información es con la que se trabaja y se obtiene el contenido del archivo txt a obtener.

```
def cargar_archivo():
    global file
    root = tk.Tk()
    root.withdraw()
    file = arch.askopenfilename()
```

2. Organizar el archivo:

La clase Organizer y el evento def file_reading(), se crearon juntos para poder controlar según el formato del archivo dato, el cual contiene filas de datos. En el evento se organiza la información que contiene nuestro archivo cargado.

Se abre el archivo y declarando (file,´r´) se lee y se carga en memoria lo que contiene el archivo. Para almacenar la información alterada se crearon dos diccionarios. Para luego crear un bucle for para recorrer nuestro archivo, como en el archivo existe cierta información que no sirve para la elaboración de las funciones, o sea no es data necesaria. Entonces sabemos que a partir de “=”, justo después de eso se encuentra la data que si se va a utilizar, por lo tanto usando la función. split(“=”) especificando el signo =, se quita y esa información lo guardamos con otro nombre en uno de los diccionarios al que se le llamará better_list.

Después se crean varios id and else para especificar si encuentra la palabra [“ORDENAR”], [“BUSCAR”] ó ambas, entonces vamos a almacenar en el diccionario options dichas palabras que son las que nos van a definir qué acción va a realizar con la data. Entonces se desarrolla una lógica en la que las opciones son un punto medio que van a definir que el lado izquierdo a partir de donde encuentra la función va a ser equivalente a la data que se ordenará y buscará, mientras que a partir del lado derecho de donde se encuentra la función será equivalente al número que se buscará en este caso que requiera buscar un número. A partir de ese punto nombramos las variables left_list y right_list.

Al final se introduce toda la información procesada en un diccionario y a ese diccionario se le reasigna la información dentro de una variable para poderla llamar en un return al final de la función.

```python
class Organizer:
    def file_reading(self,document):
        list_1 ={}
        for espace in document:
            options = {}
            name_options = espace.split("=")[0]
            better_list = espace.split("=")[1]

            if "ORDENAR" in better_list:
                options["ORDENAR"] = True

                if "BUSCAR" not in better_list:
                    options["BUSCAR"] = False
                    left_list = better_list.split("ORDENAR")[0]
                    options["data_list"] = "".join(left_list.split())
                else:
                    left_list = better_list.split("ORDENAR")[0]
                    if "BUSCAR" in left_list:
                        options["data_list"] = "".join(left_list.split("BUSCAR")[0].split())
                        options["BUSCAR"] = "".join(left_list.split("BUSCAR")[1].split())
                    else:
                        right_list = better_list.split("ORDENAR")[1]
                        options["data_list"] = "".join(left_list.split())
                        options["BUSCAR"] = "".join(right_list.split("BUSCAR")[1].split())

            else:
                options["ORDENAR"] = False
                if "BUSCAR" in better_list:
                    left_list = better_list.split("BUSCAR")[0]
                    right_list = better_list.split("BUSCAR")[1]
                    options["data_list"] = "".join(left_list.split())
                    options["BUSCAR"] = "".join(right_list.split())

            if options["BUSCAR"] != False:
                if options["BUSCAR"][(len(options["BUSCAR"]))-1] == ",":
                    options["BUSCAR"] = options["BUSCAR"][:-1]
```

3. Ordenar Listas:

En el evento listas_ordenadas(), se lee el archivo por medio de open_file = open(file, 'r'). Luego se usa un método de ordenamiento llamado SelectionSort el cual consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo. Luego se concatena el resultado con una variable vacía y lo almacenamos en dicha variable para al final llamarlo en un return.

```
def listas_ordenadas():
    open_file = open(file, 'r')
    lists_dictionary = Organizer().file_reading(open_file)
    data = ""
    for key,v in lists_dictionary.items():
        if v["ORDENAR"] == True:
            data_list = v["data_list"].split(",")
            for i in range(len(data_list)):
                least = i
                for k in range(i+1, len(data_list)):
                    if int(data_list[k]) < int(data_list[least]):
                        least = k

                ordenar_temp = data_list[least]
                data_list[least] = data_list[i]
                data_list[i] = ordenar_temp

            data += key + ": ORDENADOS=" + ",".join(data_list) + "\n"
    data = data[:-1]
    return data
```

4. **Buscar Listas:**

En el evento buscar_listas(), se lee el archivo por medio de una función open nuevamente para leer el archivo cargado y manejarlo nuevamente. Se crea otro diccionario en el que se manda a llamar la clase y evento donde se manejó la organización de la data. Se crea un for para recorrer la información del documento y se valúa la palabra [“BUSCAR”], que es la que será la palabra clave para que funcione este evento, seguido del dígito indicador de la búsqueda requerida. Entonces si existe la palabra buscar en la data se usará esta función. Se crea un contador para que recorra cada uno de los números que encuentre en la data almacenada en data_list y en esa parte ubicará el número y la posición donde coincida entonces si lo encuentra, guarda esa información en otro diccionario el cual se pasará a un string para poder concatenarlo y llamarlo en un return al final.

```
def buscar_listas():
    open_file = open(file, 'r')
    lists_dictionary = Organizer().file_reading(open_file)
    data2 = ""
    for key,v in lists_dictionary.items():
        if v["BUSCAR"] is not False:
            contador = 0
            a = ""
            data_list = v["data_list"].split(",")
            for n in data_list:
                if v["BUSCAR"] == n:
                    a += str(contador) + ","
                contador +=1
            if a != "":
                a = a[:-1]
            else:
                a = "NO ENCONTRADO"
            listado = ",".join(data_list)
            data2 += key+":"+listado+" BUSQUEDA POSICIONES = "+a+ "\n"
    data2 = data2[:-1]
    return data2
```

5. Concatenar resultados:

El evento found(), consiste en llamar los eventos de buscar y ordenar listas donde tenemos nuestra información ya lista, entonces se crean dos variables y se almacenan ahí, para luego concatenarlas y que ya se queden ambas en una variable para utilizar ese evento más adelante en nuestro html.

```
def found():
    ordenadas = listas_ordenadas()
    buscadas = buscar_listas()
    return ordenadas + "\n" + buscadas
```

6. Generar html:

El evento para generar el html se crea un for para recorrer nuestra información concatenada en el evento found. Se concatena también una variable vacía que es nueva, con la variable que recorre nuestro evento y la clase del html que se generará.

Escribimos el archivo html, el cual es muy simple, sólo contiene un backround y a mitad del html se interrumpe para introducir la información que queremos que aparezca en el archivo.

```
def html_file():
    complete = found()
    print(complete)
    fixi = ""
    for g in found().split("\n"):
        fixi += '<li class="gato">' + g + '</li>\n'

    f = open('archivo.html', 'w')
    message = """
    <!DOCTYPE html>
    <html>
    <head>
    <style>
    body {background-color: powderblue;}
    h1   {color: rgb(106, 158, 236);}
    p    {color: rgb(153, 0, 255);}
    </style>
        </head><h1></h1>
        <body>
        <ul class="miau">"""
    message += fixi
    message += """</ul>

        <p></p>

    </body>
    </html>
    """

    f.write(message)
    f.close()
    webbrowser.open_new_tab('archivo.html')
    return complete
```