



Gobierno Bolivariano
de Venezuela

Ministerio del Poder Popular
para la Educación Universitaria

Universidad Politécnica Territorial
del Estado Trujillo
"Mario Briceño Irigorry"



REPÚBLICA BOLIVARIANA DE VENEZUELA
MINISTERIO DEL PODER POPULAR
PARA LA EDUCACIÓN UNIVERSITARIA CIENCIA Y TECNOLOGÍA
UPT TRUJILLO "MARIO BRICEÑO IRAGORRY"
NÚCLEO "BARBARITA DE LA TORRE"
TRUJILLO EDO. TRUJILLO
PNF EN INFORMATICA

MANEJO DE ARCHIVO EN PYTHON

Autores:

Márquez Vásquez Rafael José C.I: 31.865.848

Moreno Montilla Jhonaiker Leonel C.I: 31.367.380

Lozada Joslymar C.I: 31.735.124

Profesor: Ing. Carlos Bravo

Unidad Curricular: Programación

Trujillo, Noviembre de 2024



MANEJO DE ARCHIVOS EN PYTHON

Manipular archivos es una actividad importante en cada aplicación web. Los tipos de actividades que se pueden realizar en el archivo abierto están controlados por los Modos de Acceso. Estos describen cómo se utilizará el archivo después de haber sido abierto.

Estos Modos de Acceso también especifican dónde debe ubicarse el controlador de archivo dentro del archivo. Similar a un puntero, un controlador de archivo indica dónde se deben leer o colocar los datos en el archivo.

En Python, hay seis métodos o modos de acceso, que son:

1. Solo lectura ('r'): Este modo abre los archivos de texto solo para lectura. El inicio del archivo es donde se encuentra el controlador. Si el archivo no existe, se produce un error de I/O. Este es el modo predeterminado para abrir archivos.
2. Leer y escribir ('r+'): Este método abre el archivo tanto para lectura como para escritura. El inicio del archivo es donde se encuentra el controlador. Si el archivo no existe, se produce un error de I/O.
3. Solo escritura ('w'): Este modo abre el archivo solo para escritura. Los datos en los archivos existentes se modifican y sobrescriben. El inicio del archivo es donde se encuentra el controlador. Si el archivo aún no existe en la carpeta, se crea uno nuevo.
4. Escribir y leer ('w+'): Este modo abre el archivo tanto para lectura como para escritura. El texto se sobrescribe y se elimina de un archivo existente. El inicio del archivo es donde se encuentra el controlador.



5. Solo agregar ('a'): Este modo permite abrir el archivo para escritura. Si el archivo aún no existe, se crea uno nuevo. El controlador se establece al final del archivo. Los datos recién escritos se agregarán al final, siguiendo los datos escritos anteriormente.
6. Agregar y leer ('a+'): Usando este método, puedes leer y escribir en el archivo. Si el archivo aún no existe, se crea uno nuevo. El controlador se establece al final del archivo. El texto recién escrito se agregará al final, siguiendo los datos escritos anteriormente.

A continuación se muestra el código necesario para crear, escribir y leer archivos de texto utilizando los métodos o modos de acceso de manejo de archivos de Python.

Cómo crear archivos en Pitón

En Python, se utiliza la función `open ()` con una de las siguientes opciones:

– "x" o "w" – para crear un archivo nuevo:

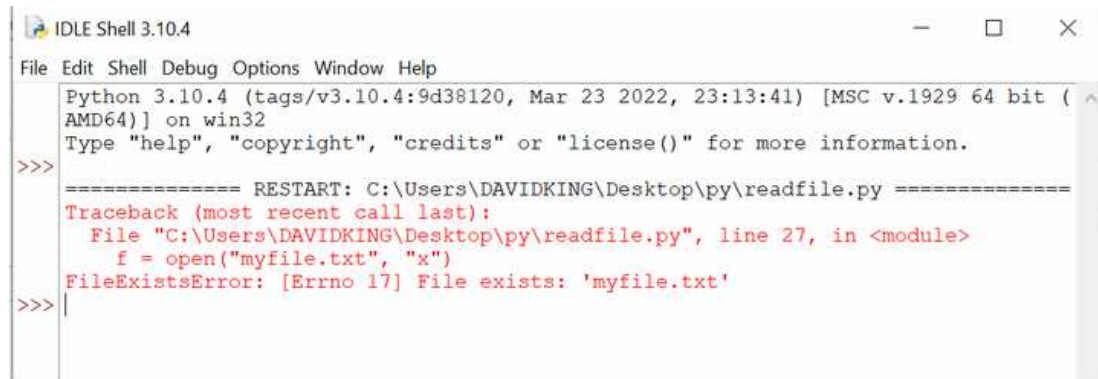
- **"x" – Crear:** este comando creará un archivo nuevo solo si no existe ningún archivo con ese nombre, de lo contrario, devolverá un error.

Ejemplo de cómo crear un archivo en Python usando el comando "x":

```
#crear un archivo de texto con la función de comando "x"  
  
f = open("myfile.txt", "x")
```

¡Hemos creado un nuevo archivo de texto vacío! Pero si vuelves a intentar el código anterior, por ejemplo, si intentas crear un nuevo archivo con el mismo nombre que usaste anteriormente (si deseas reutilizar el nombre de archivo

anterior), recibirás un error que te notificará que el archivo ya existe. Se verá como la imagen a continuación:



```
IDLE Shell 3.10.4
File Edit Shell Debug Options Window Help
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\DAVIDKING\Desktop\py\readfile.py =====
Traceback (most recent call last):
  File "C:\Users\DAVIDKING\Desktop\py\readfile.py", line 27, in <module>
    f = open("myfile.txt", "x")
FileExistsError: [Errno 17] File exists: 'myfile.txt'
>>>
```

- **"w" – Escribir:** Este comando creará un nuevo archivo de texto, sin importar si hay un archivo en la memoria con el nuevo nombre especificado. No devuelve un error si encuentra un archivo existente con el mismo nombre, en su lugar, sobrescribirá el archivo existente.

Ejemplo de como crear un archivo con el comando "w":

```
#creando un un texto con el el comando "w"
```

```
f = open("myfile.txt", "w")
```

#Este comando "w" también se puede utilizar para crear un archivo nuevo, pero a diferencia del comando "x", el comando "w" sobrescribirá cualquier archivo existente que se encuentre con el mismo nombre de archivo.

Con el código anterior, ya sea que el archivo exista o no exista en la memoria, aún puede continuar y usar ese código. Solo tenga en cuenta que sobrescribirá el archivo si encuentra un archivo existente con el mismo nombre.



CÓMO ESCRIBIR EN UN ARCHIVO EN PYTHON

Existen dos métodos para escribir en un archivo en Python, que son:

El método write():

Esta función inserta la cadena en el archivo de texto en una sola línea.

Según el archivo que hemos creado anteriormente, la siguiente línea de código insertará la cadena en el archivo de texto creado, que es "myfile.txt."

```
file.write("Hello There\n")
```

El método writelines():

Esta función inserta varias cadenas al mismo tiempo. Se crea una lista de elementos de cadena y luego cada cadena se agrega al archivo de texto.

Usando el archivo creado anteriormente arriba, la siguiente línea de código insertará la cadena en el archivo de texto creado, que es "myfile.txt."

```
f.writelines(["Hello World ", "You are welcome to  
Fcc\n"])
```

Ejemplo:

```
#Este programa muestra cómo escribir datos en un archivo  
de texto.This
```

```
file = open("myfile.txt","w")  
L = ["This is Lagos \n","This is Python \n","This is Fcc  
\n"]
```

```
#Asignando ["This is Lagos \n","This is Python \n","This  
is Fcc \n"] a
```



```
#la variable L, se puede usar cualquier letra o palabra  
de tu elección.  
#Las variables son contenedores en los que se pueden  
almacenar valores.  
#El \n se coloca para indicar el final de la línea.
```

```
file.write("Hello There \n")  
file.writelines(L)  
file.close()
```

```
# Usar close() para cambiar los modos de acceso a  
archivos
```

CÓMO LEER DESDE UN ARCHIVO DE TEXTO EN PYTHON

Hay tres métodos para leer datos de un archivo de texto en Python. Ellos son:

El método read ():

Esta función devuelve los bytes leídos como una cadena. Si no se especifica ninguna n, lee el archivo completo.

Ejemplo:

```
f = open("myfiles.txt", "r")  
#('r') opens the text files for reading only  
print(f.read())  
#El "f.read" imprime los datos en el archivo de texto en  
la consola cuando se ejecuta.
```

El método readline():

Esta función lee una línea de un archivo y la devuelve como una cadena. Lee como máximo n bytes para el n especificado. Pero incluso si n es mayor que la longitud de la línea, no lee más de una línea.



```
f = open("myfiles.txt", "r")  
print(f.readline())
```

El método readlines():

Esta función lee todas las líneas y las devuelve como elementos de cadena en una lista, uno para cada línea.

Puedes leer las dos primeras líneas llamando `readline()` dos veces, leyendo las dos primeras líneas del archivo:

```
f = open("myfiles.txt", "r")  
print(f.readline())  
print(f.readline())
```

CÓMO CERRAR UN ARCHIVO DE TEXTO EN PYTHON

Es una buena práctica cerrar siempre el archivo cuando se ha terminado con él.

Ejemplo para cerrar un archivo de texto:

Esta función cierra el archivo de texto cuando haya terminado de modificarlo:

```
f = open("myfiles.txt", "r")  
print(f.readline())  
f.close()
```

La función `close ()` al final del código le dice a Python, ya terminé con esta sección de creación o lectura, es como decir Fin.



Ejemplo:

El siguiente programa muestra más ejemplos de formas de leer y escribir datos en un archivo de texto. Cada línea de código tiene comentarios para ayudar a comprender lo que está sucediendo:

```
# Programa para mostrar varias formas de leer y
# escribir datos en un archivo de texto.

file = open("myfile.txt","w")
L = ["This is Lagos \n","This is Python \n","This is Fcc
\n"]

# Asignar ["This is Lagos \n","This is Python \n","This
is Fcc \n"]
# a la variable L

#El \n se coloca para indicar fin de línea

file.write("Hello There \n")
file.writelines(L)
file.close()
# Usar close() para cambiar los modos de acceso a
archivos

file = open("myfile.txt","r+")
print("Output of the Read function is ")
print(file.read())
print()

# La función seek(n) Mueve el puntero hacia el byte
indicado,
# el byte desde el principio.
file.seek(0)

print( "The output of the Readline function is ")
print(file.readline())
print()

file.seek(0)
```




```
# Mostrar la diferencia entre lectura y línea de lectura

print("Output of Read(12) function is ")
print(file.read(12))
print()

file.seek(0)

print("Output of Readline(8) function is ")
print(file.readline(8))

file.seek(0)
# Función de lectura de líneas
print("Output of Readlines function is ")
print(file.readlines())
print()
file.close()
```

Este es el resultado del código anterior cuando se ejecuta en la consola. Se asignó "This is Lagos", "This is Python", y "This is Fcc" a "L" y luego se pidió que imprima, usando la función "file.read".

El código anterior muestra que la función "readline()" devuelve la letra según el número especificado, mientras que la función "readlines()" devuelve cada cadena asignada a "L", incluido \n. Es decir, la función "readlines()" imprimirá todos los datos del archivo.



```

IDLE Shell 3.10.4
File Edit Shell Debug Options Window Help
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\DAVIDKING\Desktop\file.py =====
Output of the Read function is
Hello There
This is Lagos
This is Python
This is Fcc

The output of the Readline function is
Hello There

Output of Read(12) function is
Hello There

Output of Readline(8) function is
Hello Th
Output of Readlines function is
['Hello There \n', 'This is Lagos \n', 'This is Python \n', 'This is Fcc \n']
>>>

```

EJEMPLO DE MANEJO DE ARCHIVOS EN PYTHON

Paso 1: Crear y escribir en un archivo

```
nombre_archivo = 'ejemplo.txt'
```

Abrir el archivo en modo escritura ('w'), esto creará el archivo si no existe

```
with open(nombre_archivo, 'w') as archivo:
```

```
    archivo.write('Hola, este es un ejemplo de manejo de archivos en Python.\n')
```

```
    archivo.write('Aquí agregamos una segunda línea de texto.\n')
```

```
    archivo.write(';Hasta luego!\n')
```

Paso 2: Leer desde el archivo

```
print("Contenido del archivo:")
```

```
with open(nombre_archivo, 'r') as archivo:
```

```
    contenido = archivo.read()
```

```
    print(contenido)
```



```
# Paso 3: Agregar más datos al archivo
with open(nombre_archivo, 'a') as archivo: # abrir en
modo añadir ('a')
    archivo.write('Esta línea se agrega al final del
archivo.\n')

# Paso 4: Volver a leer el archivo para ver el contenido
actualizado
print("Contenido actualizado del archivo:")
with open(nombre_archivo, 'r') as archivo:
    contenido_actualizado = archivo.read()
    print(contenido_actualizado)
```

Descripción del código:

Crear y escribir en un archivo:

Abrimos un archivo llamado ejemplo.txt en modo escritura ('w').
Escribimos varias líneas de texto en el archivo.
Usamos la instrucción with que asegura que el archivo se cierre automáticamente al finalizar el bloque.
Leer desde el archivo:

Abrimos el archivo en modo lectura ('r') y leemos su contenido completo.
Imprimimos el contenido en la consola.
Agregar más datos al archivo:

Abrimos el archivo en modo añadir ('a'), lo que permite agregar texto sin borrar el contenido existente.
Escribimos una nueva línea.
Leer el archivo nuevamente:

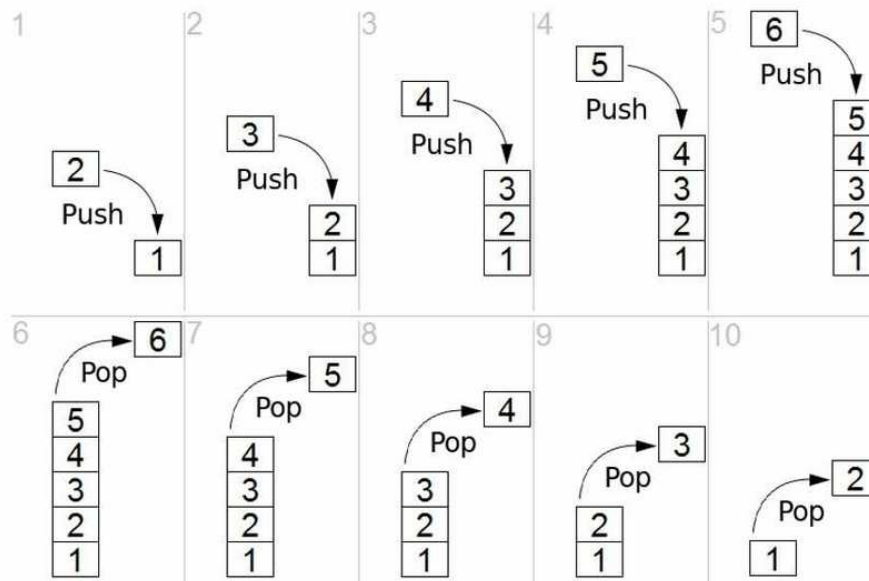
Volvemos a abrir el archivo en modo lectura para imprimir el contenido actualizado.

ESTRUCTURAS DE DATOS EN PYTHON: PILAS, COLAS

Pilas en detalle

Una pila es una colección ordenada de elementos donde la adición de un nuevo elemento y la eliminación de los existentes siempre ocurren desde el mismo extremo.

Ejemplo:



Push & pop forman 2 operaciones principales, donde push = insert & pop = remove .

El código siguiente implementa Stacks desde cero en Python .



```
1  class Stack:
2      def __init__(self):
3          self.items = []
4
5      def isEmpty(self):
6          return self.items == []
7
8      def push(self, item):
9          self.items.append(item)
10
11     def pop(self):
12         return self.items.pop()
13
14     def top(self):
15         return self.items[len(self.items)-1]
16
17     def size(self):
18         return len(self.items)
19
```

- **Stack ()**: crea una nueva pila vacía.
- **push (elemento)** : agrega un nuevo elemento a la pila.
- **pop ()** : elimina el elemento superior de la pila. Esto devuelve el elemento pop.
- **top ()**: Devuelve el elemento más alto de la pila.
- **isEmpty ()**: comprueba si la pila está vacía.
- **size ()** : Devuelve el tamaño de la pila.

¡Veamos si esto funciona !

```
s = Stack()

print (s.isEmpty())
: True

s.push(100)
s.push(200)
```



```
s.pop()  
: 200
```

```
s.size()  
: 1
```

Eso funciona. ¡Uf !

EJEMPLO DE PILAS:

```
class Pila:  
    def __init__(self):  
        self.items = []  
  
    def esta_vacia(self):  
        return len(self.items) == 0  
  
    def apilar(self, item):  
        self.items.append(item)  
  
    def desapilar(self):  
        if not self.esta_vacia():  
            return self.items.pop()  
        else:  
            raise IndexError("Desapilando de una pila  
vacía")  
  
    def peek(self):  
        if not self.esta_vacia():  
            return self.items[-1]  
        else:  
            raise IndexError("La pila está vacía")  
  
    def size(self):  
        return len(self.items)  
  
# Ejemplo de uso de la pila  
if __name__ == "__main__":  
    pila = Pila()  
    pila.apilar(1)  
    pila.apilar(2)  
    pila.apilar(3)
```



```
print("Elemento en la parte superior de la pila:",  
pila.peek()) # Debe mostrar 3  
print("Desapilando:", pila.desapilar()) # Debe  
mostrar 3  
print("¿Está la pila vacía?", pila.esta_vacia()) #  
Debe mostrar False  
print("Tamaño de la pila:", pila.size()) # Debe  
mostrar 2
```

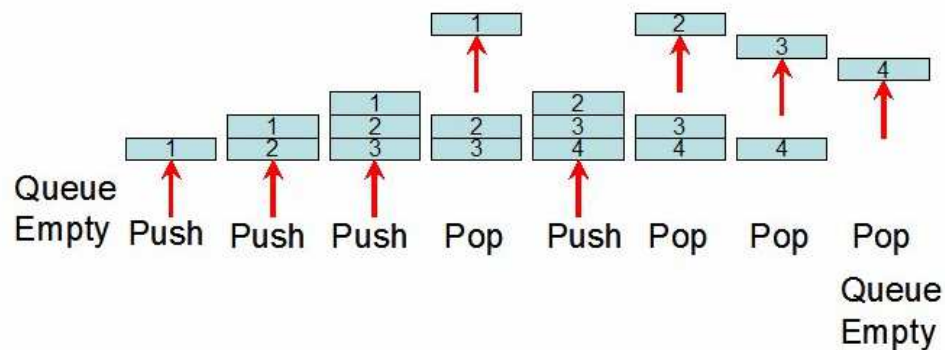
En este código:

Pila: es la clase que representa la estructura de la pila.
apilar: método para agregar un elemento a la pila.
desapilar: método para quitar el elemento superior de la pila.
peek: método para ver el elemento superior sin quitarlo.
esta_vacia: método para comprobar si la pila está vacía.
size: método que devuelve el número de elementos en la pila.

Colas en detalle

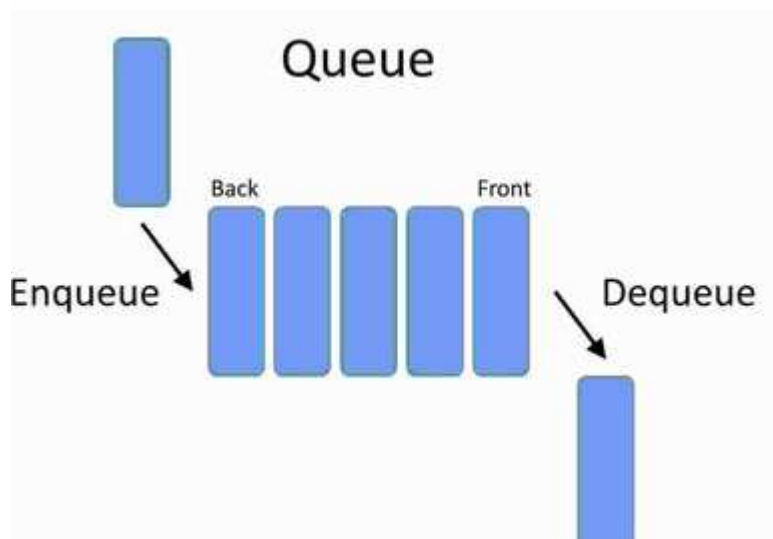
Una cola es una colección ordenada de elementos donde la adición de nuevos elementos ocurre en un extremo y la eliminación de elementos existentes ocurre en el otro extremo. Se denominan delante y detrás respectivamente.

Cuando se agrega un elemento a la cola, comienza en la parte posterior y avanza hacia el frente. Consulte las imágenes a continuación para obtener una imagen clara.



Cuando presionamos 1, comienza desde la parte trasera. Luego, cuando presionamos 2, 3, el número 1 comienza a moverse hacia el frente. Este proceso se llama poner en cola.

Imagina que queremos eliminar un elemento, esto sucede por orden de llegada. Por lo tanto, 1 aparecerá primero. Este proceso se llama sacar de cola.



El siguiente código implementa Colas desde cero en Python .



```
1  class Queue:
2      def __init__(self):
3          self.items = []
4
5      def isEmpty(self):
6          return self.items == []
7
8      def enqueue(self, item):
9          self.items.insert(0,item)
10
11     def dequeue(self):
12         return self.items.pop()
13
14     def size(self):
15         return len(self.items)
```

- **Queue ()**: crea una nueva cola vacía.
- **enqueue (elemento)**: agrega un nuevo elemento al final de la cola.
- **dequeue ()**: elimina el elemento frontal de la cola.
- **isEmpty ()**: comprueba si la cola está vacía.
- **size ()** : devuelve el tamaño de la cola.

Vamos a **ejecutar** el código!

```
q = Queue()

q.enqueue(2)
q.enqueue(10)

q.dequeue()
: 2

q.size()
```



: 1

¡Uf ! Esto también funcionó .

EJEMPLO DE COLAS:

```
from collections import deque

# Crear una cola
cola = deque()

# Agregar elementos a la cola
cola.append('Elemento 1')
cola.append('Elemento 2')
cola.append('Elemento 3')

print("Estado de la cola:", cola)

# Quitar un elemento de la cola
primer_elemento = cola.popleft()
print("Se ha quitado:", primer_elemento)

print("Estado de la cola después de quitar un elemento:",
cola)

# Agregar otro elemento
cola.append('Elemento 4')
print("Estado de la cola después de agregar otro
elemento:", cola)
```


Explicación del código:

1. **Importar `deque`:** Se importa desde `collections` para usarlo como una cola.
2. **Crear la cola:** Se inicializa una cola vacía.
3. **Agregar elementos:** Se usan `append()` para agregar elementos al final de la cola.
4. **Quitar elementos:** Se usa `popleft()` para quitar el primer elemento que se agregó.
5. **Mostrar el estado de la cola:** Se imprime el contenido de la cola en varias etapas.

```