TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Apertura en modo lectura:
    - Obteniendo atributos de un archivo.
    - Lectura completa del archivo.
    - Lectura de una línea.
- Apertura en modalidad escritura:
    - Escribiendo en el archivo.
    - Modificando el nombre.
    - Cerrando archivo.

APERTURA EN MODO LECTURA

Python ofrece una función con la que se puede crear un objeto de archivo con varios modos, uno de ellos, es el modo de lectura.

El modo de lectura en Python permite devolver el contenido del archivo. De esta manera, el programa mostrará el contenido del archivo una vez que se ejecute el código relacionado con el modo de lectura.

La función de lectura lee todo el archivo a la vez, pero esto se puede cambiar. El programador Python puede especificar un tamaño de byte con los parámetros de implementación del modo lectura.

Por último, es importante destacar también que Python incorpora funciones integradas para leer un archivo línea a línea. Esto simplifica el proceso, y evita que tengan que ser los programadores los que implementen el mecanismo de lectura de archivos.

OBTENIENDO ATRIBUTOS DE UN ARCHIVO

Una vez que un archivo está abierto, y tienes un objeto **file**, puedes obtener mucha información relacionada con él.

Se pueden acceder a los siguientes atributos del objeto **file**:

- **closed**: retorna *True* si el archivo se ha cerrado. De lo contrario, *False*.
- **mode**: retorna el modo de apertura.
- **name**: retorna el nombre del archivo.
- **encoding**: retorna la codificación de caracteres de un archivo de texto.

```
>>> archivo = open("remeras.txt", "r+")
>>> contenido = archivo.read()
>>> nombre = archivo.name
>>> modo = archivo.mode
>>> encoding = archivo.encoding
>>> archivo.close()

>>> if archivo.closed:
...     print "El archivo se ha cerrado correctamente"
... else:
...     print "El archivo permanece abierto"
...
El archivo se ha cerrado correctamente

>>> nombre
'remeras.txt'

>>> modo
'r+'

>>> encoding
None
```

LECTURA COMPLETA DEL ARCHIVO

El método `read()` leerá todo el contenido de un archivo, y lo devolverá como una cadena de texto.

En el siguiente ejemplo, se usará el método `read()` para imprimir una lista de compras del archivo lista_compras.txt:

```
archivo = open("lista_compras.txt")
print(archivo.read())
```

Este método acepta un parámetro adicional donde podemos especificar el número de caracteres a leer.

Modificando el ejemplo anterior, podremos imprimir solo la primera palabra, añadiendo el número 5 como argumento de **`read()`**.

```
archivo = open("lista_compras.txt")
print(archivo.read(5))
```

Si omitimos el argumento adicional, o el número es negativo, el archivo completo será leído.

LECTURA DE UNA LÍNEA

Si queremos leer un archivo línea por línea (en lugar de sacar todo el contenido del archivo de una sola vez) entonces podemos usar la función **`readline()`**. Existen muchas razones por las cuales pudiéramos querer solo una línea de un archivo de texto, tal vez solo necesitamos la primera o la última línea, o una mitad del archivo.

```
fichero = open('ejemplo.txt')
print(fichero.readline())
print(fichero.readline())
# Contenido de la primera línea
# Contenido de la segunda línea
```

Pero, ¿si quisiéramos todas las líneas que contiene el archivo, pero separadas? Podemos usar la misma función solo que en una nueva forma. Esta es la función **`readlines()`**.

```
fichero = open('ejemplo.txt')
lineas = fichero.readlines()
print(lineas)
#['Contenido de la primera línea\n', 'Contenido de la segunda
línea\n',
#'Contenido de la tercera línea\n', 'Contenido de la cuarta línea']
```

De manera muy sencilla podemos iterar las líneas, e imprimirlas por pantalla.

```
fichero = open('ejemplo.txt')
lineas = fichero.readlines()
for linea in lineas:
```

```
    print(linea)
#Contenido de la primera línea
#Contenido de la segunda línea
#Contenido de la tercera línea
#Contenido de la cuarta línea
```

APERTURA EN MODALIDAD ESCRITURA

El modo o la función de escritura en Python puede habilitarse al mismo tiempo que se ejecuta la de lectura, o se puede optar por ejecutar ambas de manera independiente, según las necesidades de cada proyecto y programador.

Con la función de escritura se define el contenido del archivo de texto usando la sintaxis de cadenas de Python.

En el caso de que haya que agregar nuevo contenido a un archivo ya programado mediante la función de escritura, Python permite, en el manejo de archivos, añadir nuevos contenidos a archivos de texto.

En este sentido, aplicar las funciones de lectura y escritura a todo el contenido de nuevo es una opción que consume muchos recursos, por eso existe la función de adición. En ella se puede añadir los nuevos contenidos para la escritura del archivo.

ESCRIBIENDO EN EL ARCHIVO

Método write()

Se puede añadir contenido al archivo de la siguiente manera:

```
fichero = open("datos_guardados.txt", 'w')
fichero.write("Contenido a escribir")
fichero.close()
```

Por lo tanto, si ahora abrimos el fichero **datos_guardados.txt**, veremos como efectivamente contiene una línea con contenido a escribir.

Es muy importante el uso de `close()`, ya que si se deja el fichero abierto, podríamos llegar a tener un comportamiento inesperado que queremos evitar. Por lo tanto, siempre que se abre un fichero, es necesario cerrarlo cuando hayamos acabado.

Para guardar una lista de elementos en el fichero, donde cada elemento de la lista se almacenará en una línea distinta.

```
# Abrimos el fichero
fichero = open("datos_guardados.txt", 'w')
# Tenemos unos datos que queremos guardar
lista = ["Manzana", "Pera", "Plátano"]
# Guardamos la lista en el fichero
for linea in lista:
    fichero.write(linea + "\n")
# Cerramos el fichero
fichero.close()
```

Se está almacenando la línea más \n. Es importante añadir el salto de línea, pues por defecto no se añade, y si queremos que cada elemento de la lista se almacene en una distinta, será necesario su uso.

Método writelines()

También se puede usar el método `writelines()` y pasarle una lista. Dicho método se encargará de guardar todos los elementos de la lista en el fichero.

```
fichero = open("datos_guardados.txt", 'w')
lista = ["Manzana", "Pera", "Plátano"]
fichero.writelines(lista)
fichero.close()
# Se guarda
# ManzanaPeraPlátano
```

Es importante resaltar lo que se guarda es ManzanaPeraPlátano, todo junto. Si queremos que cada elemento se almacene en una línea distinta, deberíamos añadir el salto de línea en cada elemento de la lista, tal como se muestra a continuación.

```
fichero = open("datos_guardados.txt", 'w')
lista = ["Manzana\n", "Pera\n", "Plátano\n"]
fichero.writelines(lista)
fichero.close()
# Se guarda
# Manzana
# Pera
# Plátano
```

MODIFICANDO EL NOMBRE

Usando la función os.rename()

Una solución simple para cambiar el nombre de un archivo en Python es usar la función: `os.rename()`.

```
import os
os.rename('filename.txt', 'new_filename.txt')
```

Si el archivo no está presente en el directorio de trabajo, debe especificar su ruta completa.

```
import os
os.rename('/path/to/dir/filename.txt',
'/path/to/dir/new_filename.txt')
```

Si el directorio de origen y el de destino son los mismos, es decir, puede hacer lo siguiente:

```
import os
 dir = '/path/to/dir'
old_file = os.path.join(dir, 'filename.txt')
new_file = os.path.join(dir, 'new_filename.txt')
 os.rename(old_file, new_file)
```

Usando la función shutil.move()

Como alternativa, puede utilizar el ***shutil*** módulo, que ofrece varias operaciones de alto nivel en archivos. Puedes usar la función `shutil.move()` para renombrar o mover un archivo.

```
import shutil
shutil.move('/path/to/dir/filename.txt',
'/path/to/dir/new_filename.txt')
```

CERRANDO ARCHIVO

Python cierra automáticamente un archivo si el objeto de referencia del archivo se asigna a otro, es una práctica estándar cerrar un archivo abierto, ya que en un archivo cerrado se reduce el riesgo de ser modificado o leído de forma injustificada.

Python tiene el método `close()` para cerrar un archivo. El método `close()` se puede llamar más de una vez, y si se realiza alguna operación en un archivo cerrado, genera un **ValueError**.

El siguiente código muestra un uso simple del método `close()` para cerrar un archivo abierto.

```
file = open("sample.txt")
print(file.read())
file.close()
```

Ahora, si intentamos realizar cualquier operación en un archivo cerrado como se muestra a continuación, genera un **ValueError**:

```
file = open("sample.txt")
print(file.read())
file.close()
file.write(" Attempt to write on a closed file !")
Salida:
ValueError: operación de E / S en archivo cerrado.
```

Uso del with

Se puede ahorrar una línea de código si hacemos uso de lo siguiente. En este caso nos podemos ahorrar la llamada al `close()`, ya que se realiza automáticamente. Se podría escribir de la siguiente manera:

```
lista = ["Manzana\n", "Pera\n", "Plátano\n"]
with open("datos_guardados.txt", 'w') as fichero:
     fichero.writelines(lista)
```