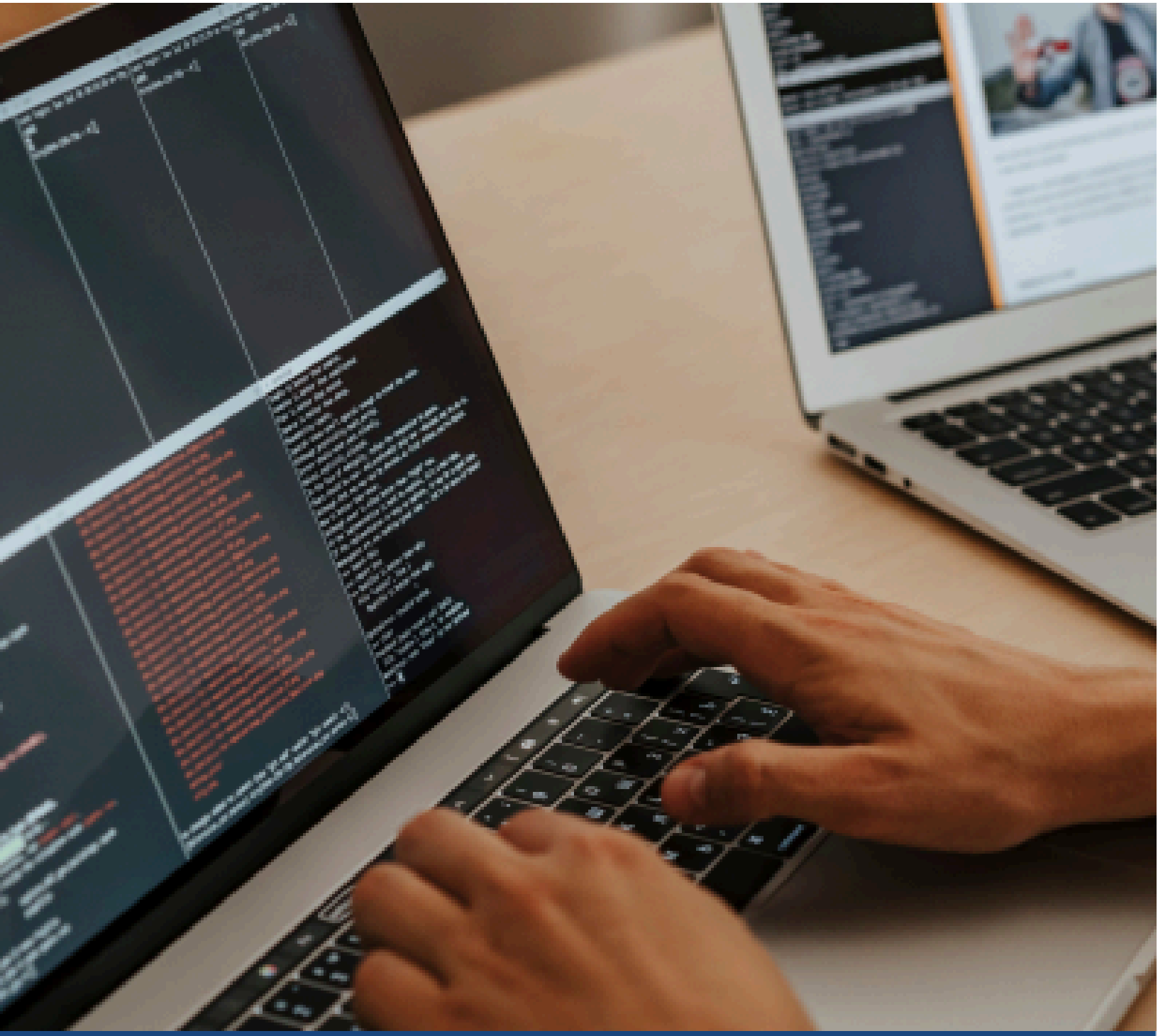
TIC

Lección 1: Manejo de archivos



TALENTO TECH

UTP Universidad Tecnológica de Pereira | faceIT

Tiempo de ejecución: 4 horas
Planteamiento de la sesión

Manejo de Ficheros

El manejo de archivos es una parte muy importante de cualquier aplicación, ya sea para la escritura de logs como bitácora de acciones o errores, o para el almacenamiento de datos.

En python existen un par de maneras de hacer, la primera y que tiene integrada entre sus funciones propias es el método open() El método cuenta con las opciones que se presentan a continuación, aunque no todas son obligatorias

open(file, mode='r', buffering=- 1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

- **file** es la ruta hacía el fichero que se quiere abrir, puede ser relativa o absoluta.
- **mode** define la forma en la cual se abrirá el archivo, por defecto se abre en modo 'r' que es solo lectura pero cuenta con más opciones

Opción	Significado
'r'	abierto para lectura (por defecto)
'w'	abierto para escritura
'x'	abierto para creación en exclusiva, falla si el fichero ya existe
'a'	abierto para escritura, añadiendo al final del fichero si este existe
'b'	modo binario
't'	modo texto (por defecto)
'+'	abierto para actualizar (lectura y escritura)

- **buffering** es opcional y. permite configurar la política del buffer

- **encoding** es el nombre de la codificación empleada con el fichero. Esto solo debe ser usado en el modo texto, por ejemplo UTF-8

- **errors** es una cadena opcional que especifica como deben manejarse los errores de codificación y descodificación – esto no puede ser usado en modo binario. Están disponibles varios gestores de error. Cuenta con opciones como:

    - 'strict' para lanzar una excepción ValueError si hay un error de codificación. El valor por defecto, None, produce el mismo efecto.

    - 'ignore' ignora los errores. Nótese que ignorar errores de codificación puede conllevar la pérdida de datos.

    - 'replace' provoca que se inserte un marcador de reemplazo (como '?') en aquellos sitios donde hay datos malformados.

    - 'surrogateescape' representará cualquier bytes incorrectos como unidades de código sustituto bajo que van desde U+DC80 a U+DCFF. Estas unidades de código sustituto volverán a convertirse en los mismos bytes cuando el gestor de errores surrogateescape sea usado al escribir datos. Esto es útil para el procesado de ficheros con una codificación desconocida.

    - 'xmlcharrefreplace' está soportado solamente cuando se escribe a un fichero. Los caracteres que no estén soportados por la codificación son reemplazados por la referencia al carácter XML apropiado &#nnn;.

    - 'backslashreplace' reemplaza datos malformados con las secuencias de escapes de barra invertida de Python.

    - 'namereplace' reemplaza caracteres no soportados con secuencias de escape \N{...} (y también está sólo soportado en escritura).

- **newline** determina cómo analizar los caracteres de nueva línea de la secuencia. Puede ser None, '','\n','\r', y '\r\n'.

Los archivos, como buena practica, es importante cerrarlos después de operar, para ello se utiliza el comando close()

```
f = open('/content/sample_data/anscombe.json', 'rt')

print(f.read(40)) #Puede leer todo el archivo con solo read() o indicar un número de carácteres

#o readline para leer una línea completa
print(f.readline())
print(f.readline())
print(f.readline())

# O ciclar por todo el archivo
for line in f:
print(line)

# Para escribir en un archivo, según la tabla anterior utilizamos la opción
#'a' para añadir al final o 'w' para sobrescribir el archivo
f = open("demofile2.txt", "a") #Crear el archivo si no existe en la ruta indicada
f.write('Mis primeras líneas en el fichero \n')
f.close()
f = open("demofile2.txt","r")
print(f.read())

f = open("demofile2.txt","w")
f.write("Uy, qué pasó?\n")
f.close()
#Ahora se ha sobrescrito el contenido anterior:
f = open("demofile2.txt","r")
print(f.read())

# con la opción 'x' se crea el archivo solo si no existe, caso contrario da un error
f = open("demofile2.txt","x")
f.write("Como ya existe, esto no va a funcionar\n")
f.close()
#Ahora se ha sobrescrito el contenido anterior:
f = open("demofile2.txt","r")
print(f.read())
```

Procesamiento de archivos de texto

Si tus archivos de texto contienen algunos caracteres nacionales no cubiertos por el juego de caracteres ASCII estándar, es posible que necesites un paso adicional. La invocación de tu función open() puede requerir un argumento que denote una codificación específica del texto.

Por ejemplo, si estás utilizando un sistema operativo Unix/Linux configurado para usar UTF-8 como una configuración de todo el sistema, la función open() puede verse de la siguiente manera:

```
stream = open('file.txt','rt', encoding='utf-8')
```

La lectura del contenido de un archivo de texto se puede realizar utilizando diferentes métodos; ninguno de ellos es mejor o peor que otro. Depende de ti cual de ellos prefieres y te gusta.
El más básico de estos métodos es el que ofrece la función read().

read()

Si se aplica a un archivo de texto, la función es capaz de:

- Leer un número determinado de caracteres (incluso solo uno) del archivo y devolverlos como una cadena.
- Leer todo el contenido del archivo y devolverlo como una cadena.
- Si no hay nada más que leer (el cabezal de lectura virtual llega al final del archivo), la función devuelve una cadena vacía.

Comenzaremos con la variante más simple y usaremos un archivo llamado text.txt. El archivo contiene lo siguiente:

```
Lo hermoso es mejor que lo feo.
Explícito es mejor que implícito.
Simple es mejor que complejo.
Complejo es mejor que complicado.
```

Ahora el código para leerlo:

```
from os import strerror

try:
counter = 0
stream = open('text.txt',"rt")
char = stream.read(1)
while char !='':
        print(char, end='')
        counter += 1
        char = stream.read(1)
stream.close()
print("\n\nCaracteres en el archivo:", counter)
except IOError as e:
print("Se produjo un error de E/S: ", strerror(e.errno))
```

- Se usa el mecanismo try-except y se abre el archivo con el nombre (text.txt en este caso).
- Intenta leer el primer carácter del archivo (char = stream.read(1)).
- Si tienes éxito (esto se demuestra por el resultado positivo de la condiciónwhile), se muestra el carácter (nota el argumento end=, ¡es importante! ¡No querrás saltar a una nueva línea después de cada carácter!).
- También, se actualiza el contador (counter).
- Intenta leer el siguiente carácter y el proceso se repite.

Si estás absolutamente seguro de que la longitud del archivo es segura y puedes leer todo el archivo en la memoria de una vez, puedes hacerlo: la función read(), invocada sin ningún argumento o con un argumento que se evalúa a None, hará el trabajo por ti.
Recuerda: **el leer un archivo muy grande (en terabytes) usando este método puede dañar tu sistema operativo.**

readline()

Si deseas manejar el contenido del archivo como un conjunto de líneas, no como un montón de caracteres, el método readline() te ayudará con eso.
El método intenta **leer una línea completa de texto del archivo,** y la devuelve como una cadena en caso de éxito. De lo contrario, devuelve una cadena vacía.
Esto abre nuevas oportunidades: ahora también puedes contar líneas fácilmente, no solo caracteres.

```
from os import strerror

try:
character_counter = line_counter = 0
stream = open('text.txt','rt')
line = stream.readline()
while line !='':
    line_counter += 1
    for char in line:
        print(char, end='')
        character_counter += 1
    line = stream.readline()
stream.close()
print("\n\nCaracteres en el archivo:", character_counter)
print("Líneas en el archivo: ", line_counter)
except IOError as e:
print("Se produjo un error de E/S:", strerror(e.errno))
```

readlines()

Otro método, que maneja el archivo de texto como un conjunto de líneas, no como caracteres, es readlines().
Cuando el método readlines(), se invoca sin argumentos, intenta **leer todo el contenido del archivo y devuelve una lista de cadenas, un elemento por línea del archivo.**

Si no estás seguro de si el tamaño del archivo es lo suficientemente pequeño y no deseas probar el sistema operativo, puedes convencer al método readlines() de leer no más de un número especificado de bytes a la vez (el valor de retorno sigue siendo el mismo, es una lista de una cadena).

```
stream = open("text.txt")
print(stream.readlines(20))
print(stream.readlines(20))
print(stream.readlines(20))
print(stream.readlines(20))
stream.close()
```

Puedes esperar que readlines() procese el contenido del archivo de manera más efectiva que readline(), ya que puede ser invocado menos veces.

Nota: cuando no hay nada que leer del archivo, el método devuelve una lista vacía. Úsalo para detectar el final del archivo.

```
from os import strerror

try:
    ccnt = lcnt = 0
    for line in open('text.txt','rt'):
      lcnt += 1
    for ch in line:
      print(ch, end='')
      ccnt += 1
    print("\n\nCaracteres en el archivo:", ccnt)
    print("Líneas en el archivo: ", lcnt)

except IOError as e:
    print("Se produjo un error de E/S: ", strerror(e.errno))
```

Manejo de archivos de texto: write()

Escribir archivos de texto parece ser más simple, ya que hay un método que puede usarse para realizar dicha tarea.
El método se llama write() y espera solo un argumento: una cadena que se transferirá a un archivo abierto (no lo olvides), el modo de apertura debe reflejar la forma en que se transfieren los datos, **escribir en un archivo abierto en modo de lectura no tendrá éxito**).
No se agrega carácter de nueva línea al argumento de write(), por lo que debes agregarlo tu mismo si deseas que el archivo se complete con varias líneas.
El ejemplo en el editor muestra un código muy simple que crea un archivo llamado newtext.txt (nota: el modo de apertura w asegura que **el archivo se creará desde cero,** incluso si existe y contiene datos) y luego coloca diez líneas en él.

```
from os import strerror

try:
  file = open('newtext.txt','wt')
  for i in range(10):
      file.write("línea #" + str(i+1) + "\n")
  file.close()
except IOError as e:
  print("Se produjo un error de E/S: ", strerror(e.errno))
```

El código debe dar como resultado un archivo llamado newtext.txt con el contenido:

línea #1
línea #2
línea #3
línea #4
línea #5
línea #6
línea #7
línea #8
línea #9
línea #10

Ejercitación

Tu tarea es escribir un programa que:

- Pida al usuario el nombre del archivo de entrada.
- Lea el archivo (si es posible) y cuente todas las letras latinas (las letras mayúsculas y minúsculas se tratan como iguales).
- Imprima un histograma simple en orden alfabético (solo se deben presentar recuentos distintos de cero).

Crea un archivo de prueba para tu código y verifica si tu histograma contiene resultados válidos.
Suponiendo que el archivo de prueba contiene solo una línea con:
#samplefile.txt
aBc

El resultado esperado debería verse de la siguiente manera:
a -> 1
b -> 1
c -> 1

Consejo: Creemos que un diccionario es un medio perfecto de recopilación de datos para almacenar los recuentos. Las letras pueden ser las claves mientras que los contadores pueden ser los valores.

Solución

```
from os import strerror

# Inicializa 26 contadores para cada letra latina.
# Nota: hemos usado una comprensión para esto.
counters = {chr(ch): 0 for ch in range(ord('a'), ord('z') + 1)}
file_name = input("Ingresa el nombre del archivo a analizar: ")
try:
  file = open(file_name,"rt")
  for line in file:
      for char in line:
          # Si es una letra...
          if char.isalpha():
              # ... lo trataremos en minúsculas y actualizaremos el contador
              apropiado.
              counters[char.lower()] += 1

file.close()
# Imprimamos los contadores.
for char in counters.keys():
    c = counters[char]
    if c > 0:
       print(char,'->', c)
except IOError as e:
  print("Se produjo un error de E/S: ", strerror(e.errno))
```

Ejercitación

El profesor Jekyll dirige clases con estudiantes y regularmente toma notas en un archivo de texto. Cada línea del archivo contiene 3 elementos: el nombre del alumno, el apellido del alumno y el número de puntos que el alumno recibió durante ciertas clases.
Los elementos están separados con espacios en blanco. Cada estudiante puede aparecer más de una vez dentro del archivo del profesor Jekyll.
El archivo puede tener el siguiente aspecto:

John Smith 5
Anna Boleyn 4.5
John Smith 2
Anna Boleyn 11
Andrew Cox 1.5
Tu tarea es escribir un programa que:

- Pida al usuario el nombre del archivo del profesor Jekyll.
- Lea el contenido del archivo y cuenta la suma de los puntos recibidos porcada estudiante.
- Imprima un informe simple (pero ordenado), como este:

Andrew Cox 1.5
Anna Boleyn 15.5
John Smith 7.0

Consejo: Emplea un diccionario para almacenar los datos de los estudiantes.

Solución

```
# Una clase de la excepción base para nuestro código:
class StudentsDataException(Exception):
pass

# Una excepción para líneas erróneas:
class WrongLine(StudentsDataException):
def __init__(self, line_number, line_string):
    super().__init__(self)
    self.line_number = line_number
    self.line_string = line_string

# Una excepción para un archivo vacío.
class FileEmpty(StudentsDataException):
def __init__(self):
    super().__init__(self)

from os import strerror

# Un diccionario para los datos de los estudiantes:
data = { }
file_name = input("Ingresa el nombre del archivo de datos del estudiante: ")
line_number = 1
try:
```

```
f = open(file_name,"rt")
# Leer el archivo completo en la lista.
lines = f.readlines()
f.close()
# ¿El archivo está vacío?
if len(lines) == 0:
  raise FileEmpty()
# Escanee el archivo línea por línea.
for i in range(len(lines)):
    # Obtener la línea n.
    line = lines[i]
    # Divídirlo en columnas.
    columns = line.split()
    # Debe haber 3 columnas, ¿están ahí?
    if len(columns) != 3:
      raise WrongLine(i + 1, line)
    # Construye una clave a partir del nombre y apellido del estudiante.
    student = columns[0] + ' ' + columns[1]
    # Obtener puntos.
    try:
        points = float(columns[2])
    except ValueError:
        raise WrongLine(i + 1, line)
    # Actualizar el diccionario.
    try:
       data[student] += points
    except KeyError:
       data[student] = points
# Imprimir resultados.
for student in sorted(data.keys()):
    print(student,'\t', data[student])

except IOError as e:
 print("Se produjo un error de E/S: ", strerror(e.errno))
except WrongLine as e:
 print("Línea incorrecta #" + str(e.line_number) + " en el archivo fuente:"
 +e.line_string)
except FileEmpty:
 print("Archivo fuente vacío")
```

Resumen

1. Para leer el contenido de un archivo, se pueden utilizar los siguientes métodos>

- read(number): lee el número de carácteres/bytes del archivo y los retorna como una cadena, es capaz de leer todo el archivo a la vez.
- readline() lee una sola línea del archivo de texto.
- readlines(número) lee el número de líneas del archivo de texto; es capaz de leer todas las líneas a la vez.
- readinto(bytearray): lee los bytes del archivo y llena el bytearray con ellos.

2. Para escribir contenido nuevo en un archivo, se pueden utilizar los siguientes métodos:

- write(string): escribe una cadena a un archivo de texto.
- write(bytearray): escribe todos los bytes de un bytearray a un archivo.

3. El método open() devuelve un objeto iterable que se puede usar para recorrer todas las líneas del archivo dentro de un bucle for. Por ejemplo:

```
for line in open("file","rt"):
 print(line, end='')
```

El código copia el contenido del archivo a la consola, línea por línea. **Nota:** el stream se cierra **automáticamente** cuando llega al final del archivo.