Status	Finished
Started	Wednesday, 27 November 2024, 6:30 PM
Completed	Wednesday, 27 November 2024, 6:32 PM
Duration	1 min 35 secs
Marks	7.00/7.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Archivos de texto

Las aplicaciones del mundo real procesan (leen o/y producen) diversos datos que se guardan en archivos externos. Muchos de ellos son textos. Ejemplos de archivos que tienen un formato de texto incluyen aquellos con extensión `.txt`, `.csv`, `.html` ... ¡y, desde luego, `.py`!

El contenido de un archivo de texto es una secuencia de **caracteres**[(a)], incluidos caracteres terminadores de línea como *newline* `'\n'`.

Podemos también considerar un archivo de texto como una secuencia de **líneas**, siendo cada línea una secuencia de caracteres terminada con `'\n'`. Un archivo de texto puede contener líneas **vacías**, esto es, que sólo tienen el carácter `'\n'`. [(b)]

Trabajar con archivos en general comprende **tres etapas**:

1. **Abrir** el archivo (se establece una conexión entre el programa y el archivo físico en el sistema de ficheros del ordenador).
2. **Leer** de, **o escribir** en, el archivo (se envían datos desde el archivo al programa, o desde el programa al archivo).
3. **Cerrar** el archivo (se libera la conexión entre el programa y el archivo físico en el sistema de ficheros; lo que pudiera requerir que previamente se completen operaciones de escritura aún pendientes).

(a) Todo archivo, del tipo que sea, hoy en día es una secuencia de octetos, o sea, de bloques de 8 bits, cuyas combinaciones de 0 y 1 codifican, de acuerdo con algún esquema preestablecido (que puede ser octeto a octeto o en grupos de octetos), algún tipo de dato. Así pues, cuando decimos que un archivo de texto contiene una secuencia de *caracteres*, lo que estamos queriendo decir es que cuando se construye el archivo los octetos que lo componen se escriben utilizando alguna codificación [*encoding*] de caracteres definida en un *estándar* o *norma*, por lo que, naturalmente, al leerlos posteriormente debieran ser interpretados de acuerdo con dicha codificación. Estándares de codificación de caracteres hay varios, no obstante, en nuestras prácticas escribiremos y leeremos usando una misma codificación, por lo que no nos afectará.

(b) Pudiéramos encontrarnos (aunque no sea recomendable según el caso) con que la última línea no esté vacía y no termine en `'\n'`. O incluso que el fichero esté vacío (contenga cero caracteres, ni siquiera un `'\n'`). Estrictamente, un fichero que contuviera un solo carácter que fuera un `'\n'` no estaría vacío, pues contendría una línea, eso sí, vacía.

Question 1

Complete

Mark 1.00 out of 1.00

Marque todas las respuestas correctas.

Select one or more:

- [x] Los archivos de texto pueden contener líneas vacías.
- [x] Los archivos de texto son secuencias de caracteres
- [ ] Los archivos de texto pueden contener líneas con 2 ocurrencias del caracter '\n'
- [x] Los archivos de texto son secuencias de líneas

Information

Apertura de un archivo

Para abrir un archivo de texto en Python llamamos a la función estándar **open()**:

```
f = open('data.txt', 'r')
```

La función open() tiene, generalmente, dos parámetros de tipo string:

- El nombre del archivo que se quiere abrir y
- La 'r' en caso de abrir archivo para su lectura (read).

Podemos especificar una ruta absoluta o relativa (en el sistema de ficheros) en la primera string. En el ejemplo hemos asumido que el archivo de entrada y la secuencia de comandos de Python (el programa) están en el mismo directorio.

La función open() devuelve una referencia al archivo abierto. En otras palabras: el flujo (de entrada, en este caso) se ha abierto y la variable de archivo, que hemos llamado f, a la izquierda de la asignación nos proporcionará la conexión, más adelante necesaria.

Si no hay en el disco un archivo con el nombre especificado, se genera un error (FileNotFoundError).

En un tercer argumento opcional podemos especificar la codificación a emplear en el archivo, por ejemplo:

```
f = open('data.txt', 'r', encoding = 'utf-8')
```

Question 2

Complete

Mark 1.00 out of 1.00

¿La siguiente operación produce un error?

```
f = open('data.txt', 'r', encoding='utf-8')
```

Select one:

- No se puede responder con la información disponible
- Sí, produce un FileNotFoundError
- No, funciona correctamente

Information

Tratamiento de un archivo de texto

Supongamos que el archivo `poem.txt` contiene el texto mostrado a continuación:

Twinkle, twinkle, little star,
how I wonder what you are.
Up above the world so high,
like a diamond in the sky.

Para leerlo y mostrarlo en pantalla escribiríamos lo siguiente:

```
f = open('poem.txt', 'r')
print(f.readline())        # 'Twinkle, twinkle, little star,\n'
```

Con el método **`readline()`**, leemos la primera línea del archivo. Este método lee y devuelve una línea, incluyendo el carácter de final de línea.

En la variable de archivo `f` se actualiza la posición real hasta la que se ha avanzado en un archivo después de cada lectura, por lo que la siguiente llamada devolverá la segunda línea:

```
print (f.readline())      # 'how I wonder what you are.\n'
```

Después de leer todas las líneas, la posición real llega al final del archivo y, en adelante, el método `readline()` devuelve `''` (una string vacía).

Para leer todas las líneas, a menudo usamos la instrucción `while`:

```
f = open('poem.txt', 'r')
line = f.readline()
```

```
while line != '':
    print(line, end = '') # la línea ya contiene un '\n' (salvo quizá la última)
    line = f.readline()
```

También es posible la siguiente versión (una string vacía se interpreta como `False` en una condición; recuérdese que una línea vacía no es una string vacía):

```
f = open('poem.txt', 'r')
line = f.readline()
```

```
while line:
    print(line, end = '') # la línea ya contiene un '\n' etc.
    line = f.readline()
```

Después de leer todos los datos necesarios, el archivo debe cerrarse inmediatamente:

```
f.close()
```

Después del cierre, se liberan todos los recursos del programa relacionados con el archivo, y el archivo queda libre para poder ser usado por otros programas.

Question 3

Complete

Mark 1.00 out of 1.00

¿Qué muestra el siguiente programa?

```
f = open('poem.txt', 'r')
line = f.readline()

while line != '':
    print(line[::-1], end='')
    line = f.readline()

f.close()
```

Select one:

- Cada línea del archivo se muesta en pantalla dada la vuelta (invertida)
- Se muestra en pantalla una línea con todos los caracteres del archivo en orden inverso
- Ocurre un error porque las líneas no pueden segmentarse

Information

Al leer líneas de un archivo, puede ser interesante ver su contenido real (todos los caracteres, incluido el final de las líneas). La función **`repr()`** es útil para este propósito, pues nos muestra los valores en un formato que podríamos usar para introducirlos como literales en nuestro código Python:

```
f = open('poem.txt','r')
line = f.readline()
```

```
while line != '':
    print(repr(line))
    line = f.readline()

f.close()
```

La salida (obsérvese que incluye los delimitadores ' de literal de string):

```
'Twinkle, twinkle, little star,\n'
'How I wonder what you are.\n'
'Up above the world so high,    \t\n'
'Like a diamond in the sky.\n'
```

Observe que la tercera línea contiene más espacios y un tabulador antes del \n final, que con un simple print no se podrían apreciar.

En Python, caracteres como el espacio, el tabulador y el final de línea se consideran **espaciadores** (*whitespaces*). A veces, debemos eliminar todos los espaciadores iniciales y finales antes de procesar una línea. Podemos usar el método **`strip()`** para esto, pues devuelve una copia de la string sobre la que sea aplica, pero sin los espaciadores al principio y al final que esta pudiera tener:

```
f = open('poem.txt','r')
line = f.readline()
```

```
while line != '':
    print(repr(line.strip()))
    line = f.readline()
```

```
f.close()
```

La salida:

```
'Twinkle, twinkle, little star,'
'How I wonder what you are.'
'Up above the world so high,'
'Like a diamond in the sky.'
```

Obsérvese que en la instrucción `print` no se usa en esta ocasión el parámetro `end = ''`, sino que se usa el terminador por omisión (\n), ya que el carácter \n ha sido eliminado por la operación `strip()`.

El método `strip()` puede tener un parámetro opcional: una string indicando el repertorio de caracteres que se quiere eliminar (no necesariamente espaciadores) de principio y final. También están disponibles las variantes **`lstrip()`**, que elimina los espaciadores, o caracteres especificados, del principio (***left***) de una string y **`rstrip()`** que hace lo mismo por el final (***right***). Repasar la semana 9 "Más sobre strings".

Question 4

Complete

Mark 1.00 out of 1.00

Relacione las siguientes funciones con los comentarios que explican algunas de sus características.

repr()	Ayuda a explorar el contenido real de una string
readline()	Devuelve una línea leída de un archivo
strip()	Elimina los espaciadores al principio y al final de una string
open()	Devuelve una referencia a un archivo abierto
close()	Cierra la conexión con un archivo abierto

Information

Tratamiento de un archivo usando la sentencia `for`

La sentencia `for` puede ser muy útil al procesar archivos de texto.

Si conociéramos de antemano la cantidad de líneas (algo inusual), en el ejemplo podríamos repetir la llamada `readline()` para imprimir las 4 líneas del poema:

```
f = open('poem.txt', 'r')
```

```
for i in range(4):
    print(f.readline(), end='')
```

```
f.close()
```

Pero sobre todo puede usarse, facilitando además el ciclo de lectura, en caso de un número desconocido de líneas (que es lo más usual):

```
f = open('poem.txt', 'r')
```

```
for line in f:
    print(line, end='')
```

```
f.close()
```

Question 5

Complete

Mark 3.00 out of 3.00

El archivo de entrada urls.txt contiene algunos nombres de dominio de webs favoritas, uno por línea, por ejemplo:

www.ukf.sk
www.bbc.com
www.ull.es
www.google.sk
www.ulpgc.es

Complete el siguiente programa correctamente para contar todas las direcciones de España:

```
file = open('urls.txt', 'r')
```

```
n = 0
for [ line ] in [ file ]:
    if line.strip().endswith('.es'):
        n += 1

print(n)
file.close()
```