Manejo de errores y archivos

Manejo de archivos con Python

{desafío} latam_

Utilizar sentencias de captura y generación de excepciones para el control del flujo de un programa acorde al lenguaje Python.

Codificar un programa que lee y escribe archivos utilizando el lenguaje Python para resolver un problema.

- Unidad 1:
  Introducción a la programación orientada a objetos con Python
- Unidad 2:
  Tipos de métodos e interacciones entre objetos
- Unidad 3:
  Herencia y polimorfismo
- Unidad 4:
  Manejo de errores y archivos

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Reconoce los comandos y operaciones básicas para el manejo de archivos para resolver un problema acorde al lenguaje Python.*
- *Crea un programa Python para lectura de datos de un archivo externo.*
- *Crea un programa Python para escritura de datos en un archivo externo.*

{desafío} latam_

¿Cómo se logra la entrada de datos en Python?

/* Manejo de archivos en Python */

{desafío}
latam_

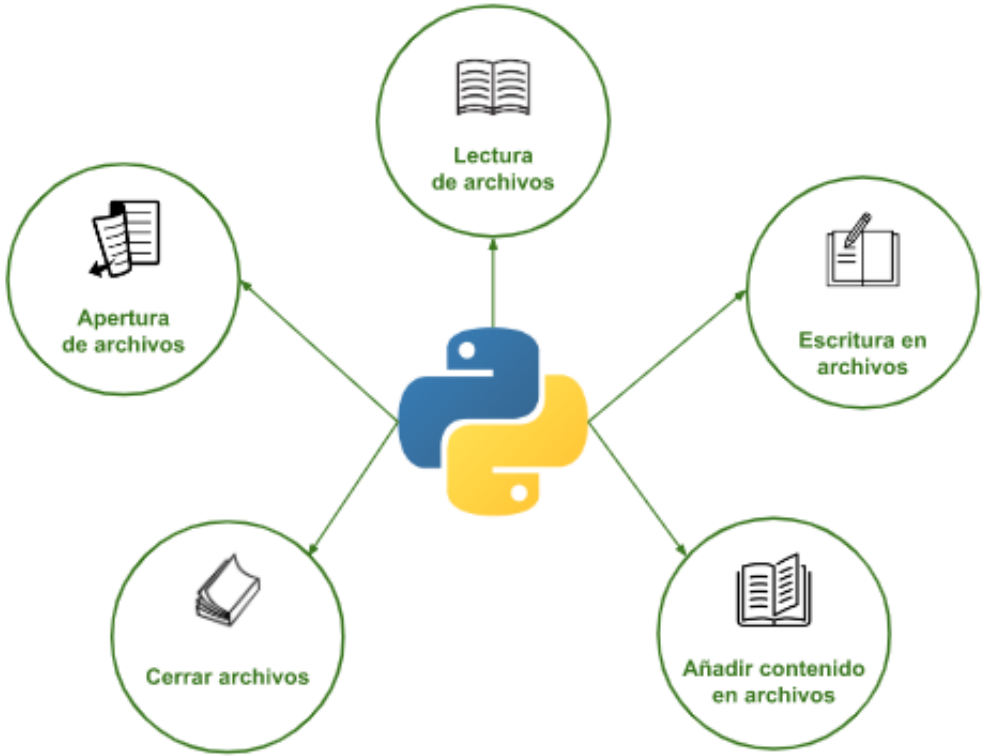
¿Por qué necesitamos manejar archivos?

Los archivos constituyen una fuente importante de entrada de datos, para resolver distintos problemas y algoritmos.

Por ejemplo:

- En ciencia de datos es muy utilizada la lectura de archivos csv como fuente de los datos de análisis.
- En el desarrollo web, a menudo se hace necesario el manejo de distintos archivos multimedia (fotos, videos, documentos, etc), que constituyen archivos que requieren ser manejados como bytes en lugar de texto.
- Otro uso común de salida de datos a un archivo es el registro de “logs” o historial de una aplicación.

Ilustración de manejo de lectura y escritura de archivos



¿Qué es un `FileDescriptor`?

Descriptor de archivos

- Recurso que permite manipular archivos, resolviendo operaciones de bajo nivel en el sistema operativo que permiten acciones de entrada y salida.
- Se representa por un número entero positivo. En Python normalmente se instancia como una variable de nombre **fd**. Para ello, se requiere hacer uso del método open del módulo **os**. Luego, la instancia **fd** es utilizada como argumento de los distintos métodos dentro del módulo **os** que permiten la manipulación del archivo (lectura, escritura y cierre).
- El uso de un descriptor de archivo está sugerido para acciones de bajo nivel de entrada y salida. Para uso normal, se sugiere utilizar la función nativa de Python open, que retorna un objeto de tipo **file**, el cual posee (entre otros) los métodos **read** y **write**, que permiten la lectura y escritura respectivamente.

{desafío} latam_

Si quieres conocer en mayor profundidad cómo trabajar con los descriptores de archivos en Python, puedes consultar la documentación oficial en este enlace.

/* Modos de apertura de archivos */

Función open

El primer paso para manipular un archivo en Python (leer, crear o modificar) es abrirlo.

- Primer argumento: la **ruta donde se encuentra** (o donde se desea crear) el archivo, o un int correspondiente a un descriptor de archivo. En caso de proporcionar una ruta, ésta puede ser absoluta o relativa, y debe incluir también el nombre del archivo.
- Segundo argumento: es el modo de apertura, el cual se representa por un pequeño texto, compuesto por letras que representan el modo de apertura. Es un argumento **opcional**, siendo su valor por defecto el modo de apertura de solo lectura.

El retorno de la función open es un objeto de tipo **file**, por defecto como archivo de texto.
En caso de que el archivo no se haya podido abrir, se lanzará una **excepción**.
Si se ha abierto el archivo en modo texto, y el archivo no es posible de convertir a texto, no se producirá un error en su apertura, pero quizás se produzca durante su lectura.

Función open

El primer paso para manipular un archivo en Python (leer, crear o modificar) es abrirlo.

En el siguiente código, se ha hecho uso además de la función abspath del módulo os, con el fin de obtener la ruta absoluta del archivo que se desea abrir. La ruta absoluta también puede escribirse manualmente, pero no es recomendado, ya que puede variar dependiendo del computador desde donde se ejecute el programa.

```
import os
log_file = open(os.path.abspath("logs/error.log"))
```

Si quieres conocer en mayor profundidad cómo funciona la función open, puedes revisar la documentación oficial de Python en este enlace.

Distintos modos de apertura

- El modo de apertura de un archivo (o ‘fichero’), está dado por el segundo argumento (parámetro mode) de la función open.

- Corresponde a un str que contiene el modo de apertura, representado por una o más letras, siendo su valor por defecto 'r', lo cual corresponde al modo de lectura de texto (sinónimo del modo 'rt').

- Adicionalmente, se puede añadir dentro del modo el símbolo “+”, que indica que el archivo se ha abierto para actualizar.

Distintos modos de apertura

Carácter	Significado
'r'	Abierto para lectura (por defecto)
'w'	Abierto para escritura , truncando primero el fichero (elimina su contenido)
'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)

Distintos modos de apertura

Solo lectura

- Solo se puede acceder al contenido de un archivo existente, sin posibilidad de modificarlo.
- Si la ruta especificada en open no existe, se lanzará una excepción de tipo **FileNotFoundError**.
- Si el archivo se abre correctamente, y se intenta hacer uso del método que permite escribir contenido en éste, se producirá una excepción de tipo **UnsupportedOperation**.

Distintos modos de apertura

Solo lectura

¿Cómo abrir los archivos en modo lectura?

```
# modo lectura (por defecto)
archivo = open("ejemplo.txt")

# modo lectura, espeficado por el segundo argumento 'r'
historial = open("log.txt", 'r')
```

{desafío} latam_

Distintos modos de apertura

Solo lectura

¿Cómo leer un archivo completo versus leerlo línea a línea?

Opción 1

```
pagina = open("index.html")

# Salida en modo interactivo:
# '<!DOCTYPE html>\n<html>\n    <head>\n        <title>¡Hola
Mundo!</title>\n    </head>\n    <body>\n        <p>¡Hola mundo!</p>\n
</body>\n</html>\n'
pagina.read()
```

{desafío} latam_

Distintos modos de apertura

Solo lectura

¿Cómo leer un archivo completo versus leerlo línea a línea?

Opción 2

```
pagina = open("index.html")
lineas = pagina.readlines()

# Salida:
# ['<!DOCTYPE html>\n', '<html>\n', '    <head>\n', '
<title>¡Hola Mundo!</title>\n', '    </head>\n', '    <body>\n', '
<p>¡Hola mundo!</p>\n', '    </body>\n', '</html>\n']
print(lineas)
```

Distintos modos de apertura

Solo lectura

¿Cómo leer un archivo completo versus leerlo línea a línea?

Opción 3

```
with open("index.html", "r") as archivo:
    for linea in archivo:
        print(linea.strip())
```

Distintos modos de apertura

Solo lectura

¿Cómo cerrar un archivo?

```
archivo = open("index.html")
archivo.close()
```

Distintos modos de apertura

Solo escritura

- Permite escribir datos en el archivo entregado como primer argumento de open.
- Si el archivo especificado no existe, será creado.
- Si el archivo existe, su contenido será eliminado en el momento de hacer uso de open.
- Si se intenta acceder al contenido de la instancia generada, ya sea mediante read, readlines, o iterando utilizando with, se generará una excepción de tipo **UnsupportedOperation**, indicando que la instancia del archivo no es posible de leer.

Distintos modos de apertura

Solo escritura

¿Cómo abrir un archivo en modo de solo escritura?

```
# Si el archivo no existe, será creado sin contenido
# Si el archivo existe, se eliminará su contenido
archivo = open("nuevo_log.log", 'w')
```

{desafío} latam_

Distintos modos de apertura

Solo escritura

¿Cómo escribir datos en un archivo?

```
import time
   try:
       edad = int(input("Ingrese su edad:\n"))
   except Exception as e:
       with open(f"{round(time.time())}.log", "w") as log:
           log.write(f"ERROR: {e}")
```

Distintos modos de apertura

Solo escritura

¿Cómo modificar el nombre de un archivo?

```
import os
antiguo = os.path.join("logs", "error.txt")
nuevo = os.path.join("logs", "error.log")
os.rename(antiguo, nuevo)
```

Distintos modos de apertura

Escritura y lectura

Para abrir un archivo que permita tanto su lectura como escritura (eliminando primero el contenido del archivo abierto), **se debe entregar como segundo argumento de la función open** (para el parámetro mode) **el valor "r+"**.

Este modo indica que el archivo se ha abierto para actualizar (permite lectura y escritura).

Si el archivo entregado como primer argumento de open no existe, se lanzará una excepción de tipo FileNotFoundError.

Distintos modos de apertura

Escritura y lectura

```
try:
    edad = int(input("Ingrese su edad:\n"))
except Exception as e:
    with open("ultimo_error.log", "r+") as log:
        log.write(f"ERROR: {e}")
```

{desafío} latam_

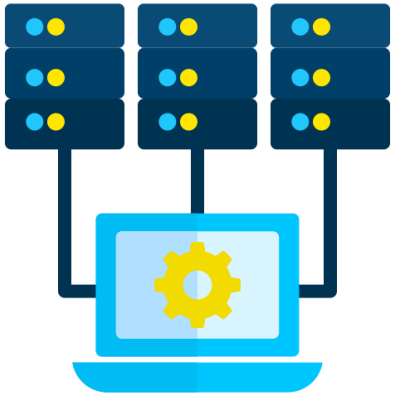
Distintos modos de apertura

Append

- Permite abrir un archivo en **modo de escritura** en el cual, al hacer uso de write (o writelines), el nuevo contenido será agregado al final del archivo abierto. En caso de que el archivo entregado como primer argumento no exista, éste será creado.

- Puede usarse en combinación con el modo +, resultando el valor "a+" como segundo argumento de open. De esta forma, el archivo abierto también se puede leer. Si el archivo ingresado como primer argumento no existe, se crea.

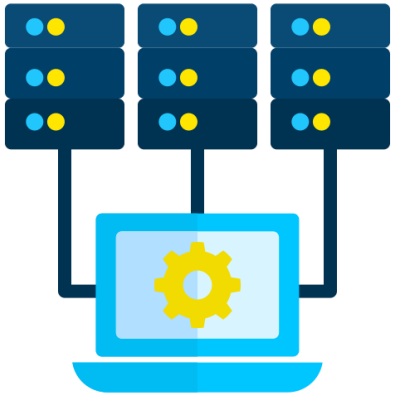
- Al abrir un archivo ya sea en modo "a" o "a+", éste se abre apuntando “al final del documento”. Esto se logra por medio del método **seek**, el cual recibe como argumento la posición (en bytes, un número entero) donde se desea posicionar dentro del archivo. Al ingresar como argumento el valor 0, se posicionaría al comienzo del contenido.



Distintos modos de apertura

Append

```
from datetime import datetime
try:
    edad = int(input("Ingrese su edad:\n"))
except Exception as e:
    with open("error.log", "a+") as log:
        log.seek(0)
        print(log.read())
        now = datetime.now()
        log.write(f"[{now.strftime('%Y-%m-%d %H:%M:%S')}] ERROR: {e}\n")
        log.seek(0)
        print(log.read())
```



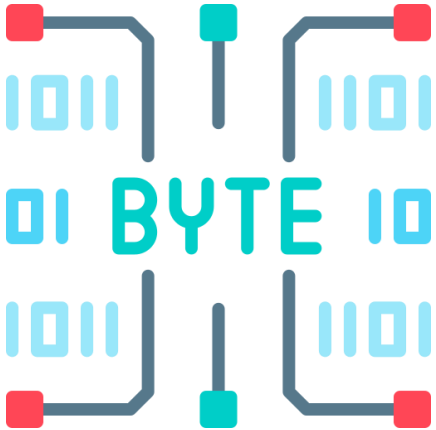
{desafío} latam_

Distintos modos de apertura

Lectura de bytes

- Por medio del método read también es posible leer una cantidad de bytes del archivo abierto.

- Para ello se debe pasar como argumento de read la cantidad de bytes (como número entero) que se desea leer.

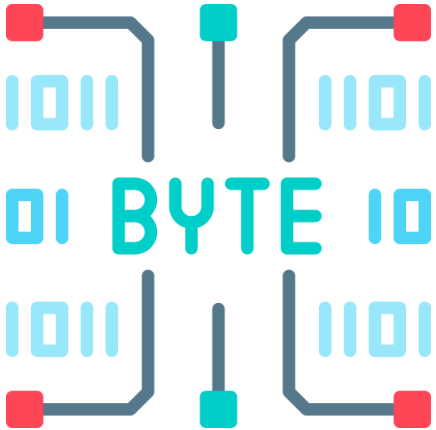


Distintos modos de apertura

Lectura de bytes

El **comienzo de la lectura** de los bytes indicados dependerá de la posición donde se encuentra abierto el archivo; si el archivo es abierto en modo de lectura, se lee (y cuenta la cantidad de bytes) desde el comienzo de su contenido.

```
with open("error.log") as log:
    log.seek(10)
    print(log.read(10))
# Salida: 3 20:17:05
```

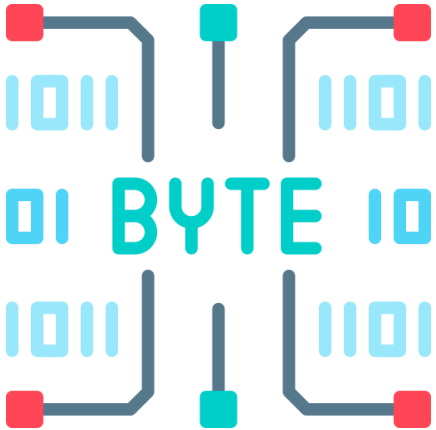


Distintos modos de apertura

Lectura de bytes

Al **terminar la lectura** de bytes indicadas, la posición de lectura del contenido se mueve al final de la cantidad de bytes indicada.

```
with open("error.log") as log:
    log.seek(10)
    print(log.read(10))
    print(log.read(10))
# Salida:
# 3 20:17:05
# ] ERROR: i
```



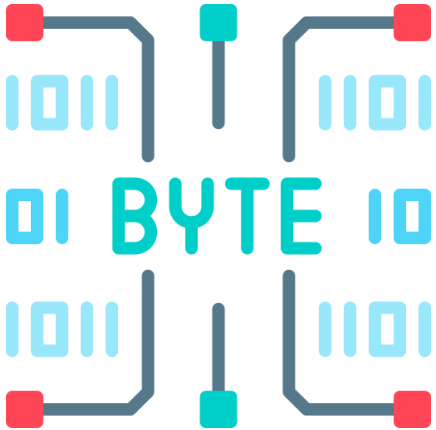
Distintos modos de apertura

Lectura de bytes

¿Cómo leer un archivo completo por chunks de bytes?

Existen ocasiones en que un archivo requiere ser manejado por bloques de lectura, a menudo debido a recursos limitados del computador frente a archivos de gran tamaño.

A su vez, muchas veces este tipo de manejo coincide con la lectura de datos en modo binario, el cual se obtiene al agregar "b" en el modo de apertura.

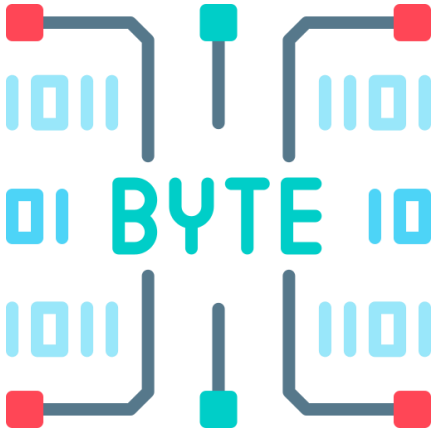


Distintos modos de apertura

Lectura de bytes

¿Cómo leer un archivo completo por chunks de bytes?

```
chunk_size = 70
with open("comprobante_de_pago.pdf", "rb") as archivo:
    chunk = archivo.read(chunk_size)
    while chunk:
        print(chunk)
        chunk = archivo.read(chunk_size)
```



Distintos modos de apertura

Resumen de los modos de apertura revisados y las acciones que se realizan en cada uno.

Modo	Lee	Escribe	Crea el archivo	Elimina contenido
'r'	✓	✗	✗	✗
'r+'	✓	✓	✗	✗
'w'	✗	✓	✓	✓
'w+'	✓	✓	✓	✓
'a'	✗	✓	✓	✗
'a+'	✓	✓	✓	✗

Buenas prácticas en el manejo de archivos

En general, al trabajar con archivos, lo más importante es manejar correctamente su apertura y cierre. Es por eso que se recomienda:

- Siempre abrir archivos utilizando el context manager with, ya que de esta forma el archivo se cerrará en caso de ocurrir alguna excepción, o al llegar al final de su contenido.
- En lugar de usar with, envolver la apertura del archivo en un bloque try/finally, de forma que siempre se cierre el archivo con close dentro de finally.

Ejercicio guiado

"Generación de objetos a partir de un archivo"

{desafío} latam_

Generación de objetos a partir de un archivo

Se le solicita crear instancias de Producto a partir de los datos contenidos en un archivo de texto. Cada línea del archivo entregado corresponde a un texto en estructura json, donde cada clave corresponde al nombre de uno de los atributos de Producto, y el valor asociado a dicha clave corresponde al valor que debe poseer el atributo en la instancia.

La estructura de la clase Producto es la siguiente:

```
class Producto():
    def __init__(self, nombre: str, precio: int) -> None:
        self.nombre = nombre
        self.precio = precio
```

Generación de objetos a partir de un archivo

Solución

Paso 1

En un script **productos.py**, crear la clase Producto entregada.

```
class Producto():
    def __init__(self, nombre: str, precio: int) -> None:
        self.nombre = nombre
        self.precio = precio
```

Paso 2

En el mismo archivo, importar el módulo json, y crear una variable contenedora de las instancias creadas, como una lista vacía.

```
import json
instancias = []
```

Generación de objetos a partir de un archivo

Solución

Paso 3

Abrir el archivo de texto que contiene los productos en modo lectura y utilizando with.

```
with open("productos.txt") as productos:
```

Paso 4

Dentro del bloque with, leer y almacenar una línea del texto.

```
linea = productos.readline()
```

Generación de objetos a partir de un archivo

Solución

Paso 5

Luego, iniciar un ciclo while que se ejecute mientras linea tenga un valor.

```
while linea:
```

Paso 6

Dentro del ciclo, usar linea como argumento de json.loads(), y almacenar retorno en una variable.

```
producto = json.loads(linea)
```

Generación de objetos a partir de un archivo

Solución

Paso 7

Luego, crear la instancia de Producto correspondiente y añadirla a la lista de instancias.

```
        instancias.append(
            Producto(producto.get("nombre"), producto.get("precio"))
        )
```

Paso 8

Finalmente, actualizar el contenido de la variable linea.

```
linea = productos.readline()
```

¿Por qué es importante el manejo de archivos en la resolución de un algoritmo?

¿Qué función nativa de Python permite la apertura de archivos?

¿Qué retorna el método readlines?

Próxima sesión...

- *Desafío evaluado.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales