UTN FACULTAD REGIONAL CÓRDOBA
INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Desarrollo de Aplicaciones con Objetos

Plantel Docente

- ✓ Diego Javier Serrano. dserrano@frc.utn.edu.ar
- ✓ Martin Gustavo Cassatti. mcasatti@frc.utn.edu.ar
- ✓ Fernando Mario Sanabria. fsanabria@frc.utn.edu.ar
- ✓ Oscar Ernesto Botta. obotta@frc.utn.edu.ar

Contenidos Semana 2

Temas

- ✓ Uso de archivos plano
- ✓ Lectura y manipulacion de archivos txt y csv
- ✓ Lectura y manipulacion de archivos json
- ✓ Conjuntos
- ✓ Diccionarios

Material de la semana

- ✓ Aula Virtual
  https://uv.frc.utn.edu.ar/
  DAO 2025

- ✓ Repositorio
  Github.com

- ✓ Material Conceptual
  https://uv.frc.utn.edu.ar/pluginfile.php/918257/mod_resource/content/4/unidad1.pdf

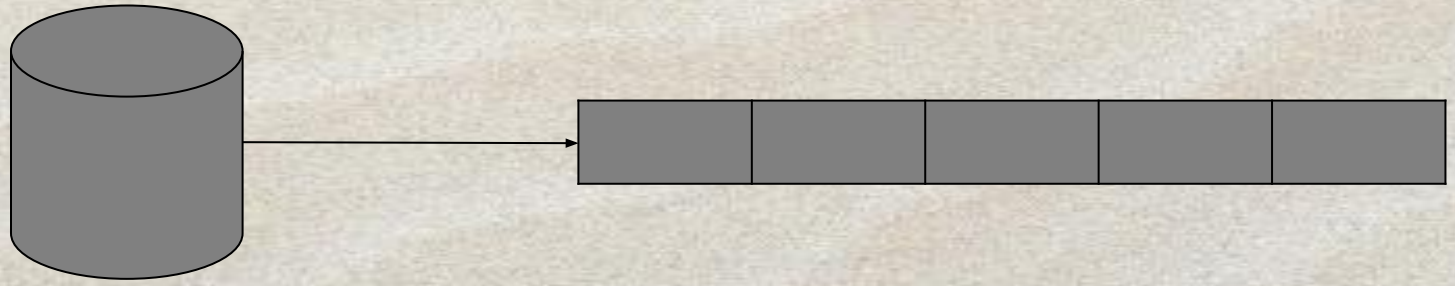
- ✓ Tutorial Visual Studio Code
  https://uv.frc.utn.edu.ar/pluginfile.php/918408/mod_resource/content/2/vsc.pdf

Material de la semana (II)

- ✓ Videos – Tutoriales
- ✓ Playlist https://www.youtube.com/@DAO-UTN
- ✓ Semana 3 - Segunda parte: https://www.youtube.com/playlist?list=PLCYPq4p46CyT5-w-_jeuTvgdk81-P8XzZ
  - Lectura de Archivos
    - https://www.youtube.com/watch?v=im8BCHSyrFE
  - Lectura de Archivos TXT
    - https://www.youtube.com/watch?v=Fxvo18vB0SU
  - Lectura de Archivos CSV
    - https://www.youtube.com/watch?v=zRlxad7d5KY

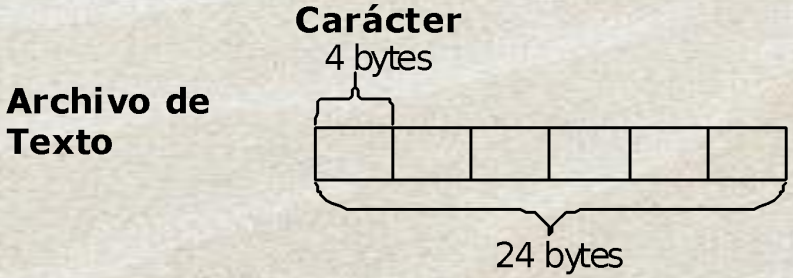
Archivos

- Es una estructura de datos lineal almacenada en dispositivos de almacenamiento externos (como discos o memorias flash).
- Los datos que se graban en un archivo se representan en sistema binario y por cada dato se utilizan tantos bytes como sea necesario para representar ese dato.
- El *tamaño de un archivo* es la *cantidad total de bytes* que contiene el archivo.

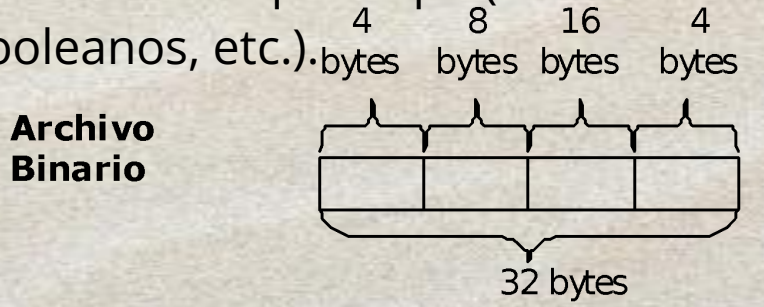


Tipos de archivo

Archivos de texto: Todos los bytes del archivo son interpretados como caracteres (en base, por ejemplo, a la tabla ASCII).



Archivos binarios: Las secuencias o bloques de bytes que el archivo contiene representan información de cualquier tipo (números en formato binario, caracteres, valores booleanos, etc.).



Archivos de texto: Formato

- El lenguaje Python emplea por default el estándar Unicode bajo la forma de codificación UTF-8 (Unicode Trasformation Format o Formato de Transformación Unicode) para interpretación de texto.
- Como el estándar ASCII está incluido como un subconjunto de Unicode, cualquier archivo de texto con formato ASCII puro será leído y gestionado sin problemas.
- Cuando se abre un archivo de texto en Python, se pueden leer y grabar cadenas de caracteres en ese archivo con relativa sencillez.

Archivos de texto: Apertura

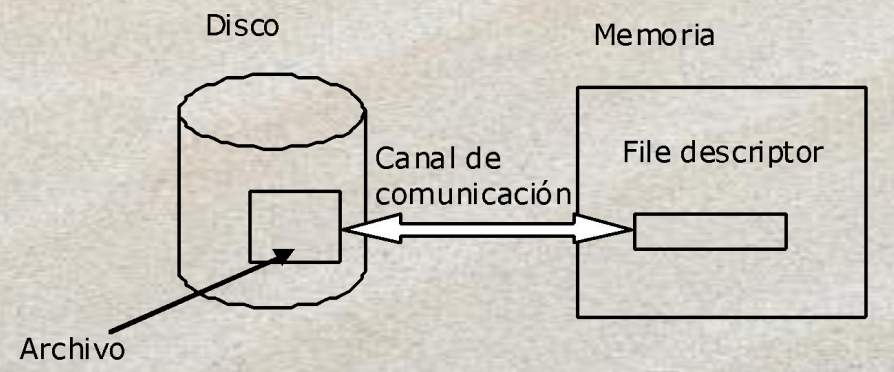
La función open() es la encargada de abrir el canal de comunicación entre el dispositivo que contiene al archivo y la memoria principal.

Definición general:

open (nombre físico o file descriptor, modo de apertura)

Ejemplos:

```
m = open("datos.txt", "w")
m = open("c:\\documents\\datos.txt", "w")
```



Archivos de texto: Cierre

- La función close() cierra el canal de comunicación, cierra la conexión y libera cualquier recurso que estuviese asociado al archivo.
- Luego de invocar a close(), la variable u objeto que representaba al archivo (el manejador del archivo) queda indefinida.
- En Python los archivos son cerrados automáticamente cuando la variable para accederlos sale del ámbito en que fue definida.
- El programador debe estar seguro que el archivo que estuvo manejando se cierre oportunamente, ya sea en forma manual o en forma automática.

Archivos de texto: Leer contenidos y bytes

- read(): Permite leer el contenido completo del archivo, retornándolo como una sola y única cadena de caracteres.
- read(n): Permite leer y retornar n bytes desde el archivo.
- Los caracteres '\n' que el archivo pudiera tener se preservan y se copian en la cadena retornada:

```
def leer_todo():
    m = open('datos.txt', 'r')
    todo = m.read()
    print('Contenido completo:')
    print(todo)
    m.close()
```

Archivos de texto: Leer líneas

- readline(): Permite leer una línea simple del archivo, retornándola como una cadena de caracteres.
- La lectura inicia donde esté el file pointer y terminará en el primer salto de línea que encuentre o al llegar al final del archivo. Mantiene al final de la cadena retornada el carácter '\n‘.
- Cuando llega al final del archivo, el método readline() retorna una cadena vacía ('').

```
def leer_lineas():

    m = open('datos.txt‘, ‘r‘ )

    linea1 = m.readline()

    linea2 = m.readline()

    print('\nContenido linea por linea:')

    print(linea1, end='')

    print(linea2)

    m.close()
```

Archivos de texto: Leer líneas

- Para leer el archivo completo, línea por línea, se puede usar un ciclo for que itere sobre el contenido del file object:

```
def leer_con_iterador():
    m = open('datos.txt‘, ‘r’)
    print('\nContenido con iterador:')
    for line in m:
        print(line, end='')
    m.close()
```

Archivos de texto: Leer líneas

- readlines(): Retorna una lista (un arreglo) que contiene a todas las líneas del archivo (incluidos los '\n' al final de cada una).
- readlines(n) Intentará recuperar hasta n bytes del archivo. En este caso, solo incluirá en la lista retornada las cadenas completas que haya logrado leer:

```
def list_lines():

    m = open('datos.txt', 'r')

    lista = m.readlines()

    print('\nLista de lineas contenidas:')

    print(lista)

    m.close()
```

Uso de With:

```
with open("mi_archivo.txt", "w") as archivo:
    archivo.write("Esta línea se escribe usando el bloque with.\n")
```

Archivos de texto: Escribir líneas

- Ejemplo Escribir
- # Abre (o crea) un archivo en modo de escritura
- archivo = open("mi_archivo.txt", "w")
- # Escribe una línea de texto en el archivo
- archivo.write("¡Hola, este es el contenido de mi archivo!\n")
- # Escribe otra línea de texto en el archivo
- archivo.write("Aquí va otra línea de texto.\n")
- # Cierra el archivo
- archivo.close()

- Ejemplos Agregar
- # Abre el archivo en modo de anexado (append)
- archivo = open("mi_archivo.txt", "a")
- # Escribe otra línea de texto al final del archivo
- archivo.write("Esta línea se añadió al final del archivo.\n")
- # Cierra el archivo
- archivo.close()

Archivos de texto: Métodos comunes

Método	**Aplicación**
closed()	Retorna *True* si el archivo está cerrado.
flush()	Vuelca al archivo los buffers de grabación, si corresponde. Aplicable sólo archivos abiertos para grabar (no hace nada en caso contrario).
readable()	Retorna True si el archivo está disponible para ser leído.
truncate(size=None)	Trunca el contenido del archivo a una cantidad igual a *size* bytes. El archivo puede *aumentar* o *disminuir* su tamaño.
writable()	Retorna *True* si el archivo está disponible para ser grabado.
writelines(lines)	Graba el contenido de la lista *lines* en el archivo, por defecto ***sin*** incluir separadores de línea (por lo cual, el programador debe indicar ese separador al final de cada elemento de la lista *lines* si quiere los separadores).

Archivos JSON

- JSON (JavaScript Object Notation) es un formato de intercambio de datos que se utiliza para representar información estructurada de manera legible tanto para humanos como para máquinas.
- Diseñado originalmente como un subconjunto del lenguaje JavaScript, actualmente es ampliamente utilizado en una variedad de lenguajes de programación.
- Los archivos JSON están compuestos por una colección de pares clave-valor, donde las claves son cadenas de texto y los valores pueden ser cadenas, números, booleanos, objetos, matrices u otros valores JSON.

Archivos JSON

```
{
  "nombre": "Juan",
  "edad": 30,
  "soltero": true,
  "pasatiempos": ["fútbol", "lectura", "viajes"]
}
```

En este ejemplo, el objeto JSON tiene cuatro pares clave-valor: "**nombre**", "**edad**", "**soltero**" y "**pasatiempos**". Los valores pueden ser de diferentes tipos, como una **cadena de texto** ("Juan"), un **número** (30), un **booleano** (true) y un **vector de cadenas** ("fútbol", "lectura", "viajes")

Archivos JSON: Lectura

En Python, se puede utilizar la librería incorporada llamada json para leer e interpretar archivos JSON de manera sencilla.

Supongamos que hay un archivo llamado "datos.json" con el siguiente contenido:

```
{
  "nombre": "Juan",
  "edad": 30,
  "soltero": true,
  "pasatiempos": ["fútbol", "lectura", "viajes"]
}
```

Archivos JSON: Lectura

```
import json

# Abre el archivo JSON en modo lectura
with open('datos.json', 'r') as archivo:
    datos = json.load(archivo)

# Ahora, 'datos' es un diccionario que contiene la información del archivo JSON
print(datos['nombre'])  # Imprime: Juan
print(datos['edad'])    # Imprime: 30
print(datos['pasatiempos'])  # Imprime: ['fútbol', 'lectura', 'viajes']
```

Archivos JSON: Escritura

```
import json
datos= {
    'nombre':'Juan Perez',
    'edad':30,
    'profesion':'estudiante'
    }
# Abre el archivo JSON en modo escritura
nombre_archivo = 'datos_simples.json'
# Grabar el archivo JSON
with open(nombre_archivo, 'w') as archivo_json:
    json.dump(datos, archivo_json, indent=4)

print(f'Datos grabados en {nombre_archivo}')
```

Archivos JSON: Archivos complejos


```
{
     "ResultSet": {
     "totalResultsAvailable": "1827221",
     "totalResultsReturned": 2,
     "firstResultPosition": 1,
     "Result": [
          {
               "Title": "potato jpg",
               "Summary": "Kentang Si bungsu dari …",
               "Url": "http://www.aprenderaprogramar.com/spaw/uploads /images/potato.jpg",
               "ClickUrl": "http://www. aprenderaprogramar.com/spaw/uploads/images/potato.jpg",
               "RefererUrl": "http://www.mediaindonesia.com/mediaperempuan/index.php?ar_id=Nzkw",
               "FileSize": 22630,
               "FileFormat": "jpeg",
               "Height": "362",
               "Width": "532",
               "Thumbnail": {
               "Url": "http://thm-a01.yimg.com/nimage/557094559c18f16a",
               "Height": "98",
               "Width": "145"
               }
          },{...},{...}
     ]
}
```

Colecciones avanzadas: Conjuntos

Los conjuntos son una estructura de datos proporcionada por el lenguaje que permite almacenar y manipular colecciones de elementos.

A diferencia de las listas o las tuplas, los conjuntos tienen características distintivas que los hacen únicos:

- Elementos únicos
- No ordenados
- Mutables (pueden cambiar luego de creados)
- Admiten operaciones de conjuntos:
  - union()
  - intersection()
  - difference()
- Heterogéneos (elementos de distintos tipos)

Colecciones avanzadas: Conjuntos

- Los conjuntos se crean con la función set() o simplemente colocando llaves {}
- La sintaxis {}, puede ser utilizada con o sin elementos.
- Hay que tener en cuenta que si se crea un conjunto con elementos duplicados, los mismos se ignoran:
  - a = {1,2,5,2,7,3,1}
  - print(a) # 1,2,5,7,3