Cronograma

- Manejos de archivos
- Abrir-Cerrar un archivo
- Modos de apertura
- Escritura y Lectura de archivos.
- Directorios
- Pickle (guardar tipos de datos complejos)
- Ejercicios

¿Qué es un archivo?

- Conjunto contiguo de bytes.
- Almacena datos.
- Formato especifico:
  - Texto,Video, Foto, ejecutables, etc
- Compuesto por tres partes:
  - Encabezado
  - Datos
  - Fin de Archivo (EOF)

Manejo de archivos

- Para abrir un archivo simplemente indicamos la dirección (path) del archivo y le decimos que lo abra en cierto modo.
- En el ejemplo, “w”, es modo write (escritura)

  f = open("test.dat","w")
- Si hacemos un print de la variable: *print (f)*

  <open file ’test.dat’, mode ’w’ at fe820>

- La función ***open*** toma dos argumentos. El primero es el nombre del archivo y el segundo es el modo. La función retorna un objeto del tipo ***file***.
- Debemos cerrar siempre el archivo al terminar
  - f.close()

Manejo de archivos
De forma responsable

with

Cuando trabajamos con archivos hay dos maneras de asegurarnos que el archivo se cierre correctamente.

```
reader = open('mi_archivo.txt')
try:
    # Procesamos las lineas del archivo
finally:
    reader.close()
```

```
with open('mi_archivo.txt', 'r') as reader:
    # Procesamos las lineas del archivo.
```

Modos de apertura

- **r : lectura.** Abre el archivo en modo solo lectura. El archivo tiene que existir previamente, en caso contrario se lanzará una excepción de tipo IOError
- **w : escritura.** Abre el archivo en modo solo escritura. Si el archivo no existe se crea (por defecto, si no especificamos path, se crea en el workspace). OJO. No crea directorios.

  Si el archivo existe, sobrescribe el contenido.
- **a: añadir (append).** Abre el archivo en modo escritura, pero se posiciona al final del archivo. No sobrescribe lo que ya hay.

Lectura de archivos

- Para procesar un archivo podemos utilizar la función ***readline(size=-1)*** que lee linea a linea. Se suele usar combinada con un while.
- Hay otras funciones que se pueden utilizar para leer archivos:

  readlines() - Devuelve una lista con todas las líneas del archivo.

  read(size=-1) - devuelve una cadena (string) que contiene todo el contenido del archivo.

Lectura de archivos (2)

- Leemos todo el archivo

```
with open('nombres.txt', 'r') as archivo:
    # Leemos y mostramos el archivo entero
    print(archivo.read())
```

```
Rodrigo
Matias
Diego
Ximena
Alba
Carlos
Fernanda
Malena
Jose
Alvaro
Soledad
Gabriela
Pablo
```

- Leemos línea a línea con for

```
with open('nombres.txt', 'r') as archivo:
    # Leemos archivo recorriendo según lo necesitamos.
    for linea in archivo:
        print(linea)
```

- Leemos linea a linea

```
with open('nombres.txt', 'r') as archivo:
    # Leemos linea a linea
    print(archivo.readline())
    print(archivo.readline())
    print(archivo.readline())
```

```
Rodrigo

Matias

Diego
```

- Todo el archivo en una lista

```
with open('nombres.txt', 'r') as archivo:
    # Leemos archivo y retornamos una lista
    print(archivo.readlines())
```

```
['Rodrigo\n', 'Matias\n', 'Diego\n', 'Ximena\n', 'Alba\n',
aro\n', 'Soledad\n', 'Gabriela\n', 'Pablo\n']
```

Escritura de archivos

- Veremos dos formas distintas de escribir un archivo.

  archivo.write(cadena)

  archivo.writelines(lista_de_cadenas)

  Escribe un elemento de la lista atrás de otro, sin salto de línea

```
with open('nombres.txt', 'a') as archivo:
    # Abrimos en modo append.
    archivo.write("Martín\n")
    archivo.write("Gustavo")
```

```
with open('nombres.txt', 'a') as archivo:
    # Abrimos en modo append.
    mas_nombres = ['Natalia','Viviana','Liliana']
    archivo.writelines("\n".join(mas_nombres))
```

Directorios

- Cuando utilizamos la función ***open*** siempre se intenta abrir el archivo ubicado en el path en el que nos encontramos. En nuestro caso, es el workspace del proyecto.
- Si queremos abrir un archivo de cualquier otro sitio del disco, tenemos que especificar la **ruta** del archivo, por ejemplo:
  - *f = open("/usr/share/dict/words","r")*
- Las rutas pueden ser relativas o absolutas (debemos contar con los permisos necesarios).
- Siempre usamos barra (“/”) para separar directorios (aunque estemos en Win). Pero podemos abstraernos del separador utilizando **os.sep**. El módulo **os** provee muchas funciones para trabajar con archivos y directorios

Directorios (2)

- El módulo os abstrae ciertas particularidades del sistema operativo.
- Algunas funciones útiles que recomendamos investigar:
  - makedirs
  - chdir
  - path.isfile
  - path.isdir
  - access
  - path.join

Directorios (3) - *makedirs*

- Crear un directorio
  - os.makedirs(ruta)
- ruta ="test/test2" y lo va a crear en el espacio en que estemos trabajando con el IDE
- Movernos al directorio recientemente creado
  - os.chdir(ruta)
- Como se mencionó, podemos evitar el uso de la barra o contrabarra utilizada por el sistema operativo para separar directorios. El módulo os provee una constante para abstraer el separador:
  - os.makedirs("dir1" + os.sep + "dir2")

Directorios (4)

- Si se crea un directorio que ya existe obtenemos un error.
- Para comprobar la existencia de un archivo:

  La función is **isfile** devuelve un booleano

  print(os.path.isfile("dat.txt"))
- Para saber si existe un directorio se puede usar **os.path.isdir**
- Para comprobar si se puede leer o escribir un archivo se cuenta con la función booleana access:

  Para comprobar permisos de lectura:
- *os.access("foo.txt", os.R_OK)*

  Para comprobar permisos de escritura:

  os.access("foo.txt", os.W_OK)

Directorios (5)

- Otra manera útil para abstraer el separador de directorios es **os.path.join**

  os.path.join(ruta, ruta1[, ... rutaN]]))

  Une las rutas con el carácter de separación de directorios que le corresponda al sistema en uso.

- Ejemplo: Crea el directorio cat/dog/fish

  os.makedirs(os.path.join('cat','dog','fish'))

Módulo Pickle

- Las funciones **write** y **writelines** solamente escriben string en un archivo. ¿Qué pasa si queremos escribir algún otro tipo de dato? ¿Tuplas, enteros, diccionarios,objetos, etc?
- Existe un módulo llamado **Pickle** que toma cualquier tipo de datos de Python y lo convierte a una representación en cadena (lo serializa)
- Los objetos se guardan serializados para su posterior lectura.
- ¡Pero atención! El serializado realizado con **Pickle** no es legible, ya que se guarda en binario.

Serialización - picklear y despicklear

Para poder picklear y despicklear debemos abrir un archivo en modo escritura o lectura respectivamente, pero además, debemos agregar la “b” para indicar que lo que se guardará o leerá será binario

fichero = open("datos.dat", "wb")

Si tenemos un objeto x, la manera más simple de picklear el objeto:

pickle.dump(x, fichero)

Serialización - picklear y despicklear

Para despicklear el objeto:

x = pickle.load(f)

```
import pickle
sis_func = {'bin': "sis_bin", 'oct': "sis_oct", 'hex': "sis_hex"}
fichero = open("datos.dat", "wb")

pickle.dump(sis_func, fichero)
fichero.close()

fichero = open("datos.dat", "rb")
x = pickle.load(fichero)
print ("esto es desplickeado: {}".format(x))
```

Ejercicios

- **Ejercicio 1:**

  Escribir un programa, que reciba un *archivo* y un número *n* e imprima las primeras n líneas del archivo.

- **Ejercicio 2:**

  Escribir un programa, que copie todo el contenido de un archivo a otro, de modo que queden exactamente igual.

Ejercicios (2)

```
import datetime
print (datetime.datetime.now())
```

Ejercicio 3:

Utilizando el modulo datetime y la función datetime.datetime.now(), escribir una función que reciba dos parámetros:

- un nombre de archivo (con su ruta relativa incluida)
- un texto.

La función abre el archivo de log indicado por parámetro y guarda en el archivo una línea con un texto indicado por parámetro, incluyendo la hora actual.

Cierra el archivo de log.