**MANEJO DE ARCHIVOS EN PYTHON**

Para manejar archivos pueden utilizarse varios métodos:

Open() para abrir/crear un archivo

archivo = open('./Seccion 20/prueba.txt', 'w')

en este caso, se crea o se abre el archivo determinado en la ruta especificada y con ‘w’ se indica que se escribirá en el archivo.

Es recomendable escribir este código en un bloque try

except Exception as e:

    print(f'error {e}, {type(e)}')

finally:

    archivo.close()

finalmente, es recomendable cerrar el archivo.

**LEYENDO ARCHIVOS DESDE PYTHON**

Para esto, se utiliza el método open() con primer parámetro: ruta y segundo parámetro: ‘r’

El método que se utiliza es .read()

try:

    file = open('./Seccion 20/prueba.txt', 'r', encoding='utf8')

    print(file.read())

except Exception as e:

    print(f'error {e}, {type(e)}')

cuando se usa .read() a secas, se lee todo el documento, por tanto si luego de esta ejecución, se vuelve a usar .read() o .read(4) no habrá retorno ya que todos los caracteres fueron leidos.

.read() puede recibir como parámetro (int) la cantidad de caracteres a leer.

**MODOS DE OPERAR CON ARCHIVOS**

R – read. Open file for Reading, error if it does not exist

A -> open file for appending, create the files if it doesnot exist

W -> crea o abre para escritura

X -> crea, retorna error si ya existe

También se puede especificar el modo

T text

B ->binary (images)

**LEER LINEAS**

Con el método **.readline()** se lee una línea del achivo. Si se especifica la cantidad por parámetro hará lo mismo.

Con **.readlines()** lee todas las líneas y las devuelve con este formato de linea:

['Agregando información al archivo\n', '0\n', '1\n', '2\n', '3\n', '4\n', '5\n', '6\n', '7\n', '8\n', '9\n', '10\n', '11\n', '12\n',

'13\n', '14\n', '15\n', '16\n', '17\n', '18\n', '19\n', '20\n', '21\n', '22\n', '23\n', '24\n', '25\n', '26\n', '27\n', '28\n', '29\n', '30\n', '31\n', '32\n', '33\n', '34\n', '35\n', '36\n', '37\n', '38\n', '39\n', '40\n', '41\n', '42\n', '43\n', '44\n', '45\n', '46\n', '47\n', '48\n', '49\n', '50\n', '51\n', '52\n', '53\n', '54\n', '55\n', '56\n', '57\n', '58\n', '59\n', '60\n', '61\n', '62\n', '63\n', '64\n', '65\n', '66\n', '67\n', '68\n', '69\n', '70\n', '71\n', '72\n', '73\n', '74\n', '75\n', '76\n', '77\n', '78\n', '79\n', '80\n', '81\n', '82\n', '83\n', '84\n', '85\n', '86\n', '87\n', '88\n', '89\n', '90\n', '91\n', '92\n', '93\n', '94\n', '95\n', '96\n', '97\n', '98\n', '99\n', '100\n']

**CONTEXTO WITH DE ARCHIVOS**

Existe una sintaxis simplificada para abrir y cerrar el archivo que es la siguiente:

*with* open('./Seccion 21/with.txt', 'w', *encoding*='utf8') *as* with\_archivo:

dentro de este bloque de código se abre y cierra el archivo renombrado como **with\_archivo**

los métodos que se encargan de esto son \_\_enter\_\_() y \_\_exit\_\_() que pueden ser sobreescritos

**CREANDO UNA CLASE QUE ADMINISTRE RECURSOS**

Para manejar archivos de forma personalizada es necesario crear una clase. No es necesario que extienda desde otra, pero si que sobreescriba los métodos \_\_enter\_\_() y \_\_exit\_\_()

class ManejoArchivos:

    def \_\_init\_\_(*self*, *nombre*) -> None:

*self*.nombre = *nombre*

    def \_\_enter\_\_(*self*):

        print('estamos abriendo el archivo'.center(50, '-'))

*self*.nombre = open(*self*.nombre, 'r', *encoding*='utf8')

*return* *self*.nombre

    def \_\_exit\_\_(*self*, *execption\_type*, *exception\_value*, *trace\_back*):

        print('closing the fucking file'.center(50, '-'))

*if* *self*.nombre:

*self*.nombre.close()

el init solo recibe la ruta del archivo a abrir.

* \_\_enter\_\_() se ejecuta al recibir el archivo. Imprime una línea de código y ejecuta el método open sobre lo que esté guardado en self.nombre. A su vez, asgna ese valor al atributo nombre del objeto. Por ultimo retorna este mismo.
* El método exit, recibe el tipo de excepción, el valor y el mensaje. Imprime una línea de código y, en caso de que self.nombre (el lugar donde se guarda el archivo) “apunte a algun archivo”, se cierra. Este método se ejecuta al finalizar el código dentro del bloque with: