Clase 6

Repaso de lo que vimos en la **#clase5**

CLASES
OBJETOS
POO

UNIDAD VIII
ARCHIVOS, MÓDULOS Y PAQUETES

Archivo

Un archivo es una *secuencia de datos almacenados en un medio persistente que están disponibles para ser utilizados por un programa*. Todos los archivos tienen un nombre y una ubicación dentro del sistema de archivos del sistema operativo.

Un programa no puede manipular los datos de un archivo directamente. Para hacerlo, debe abrir el archivo y asignarlo a una variable, que llamaremos el archivo lógico.

PROCESO

Si no la tienen incluido, necesitarán importar la librería io:

```
import io
```

Si el archivo no está previamente creado, se crea al abrirse, esto quiere decir que en el proceso creación y apertura pueden ser una sola etapa.

Cuando el archivo se crea y abre, se debe especificar para qué se abre: lectura, escritura, agregar contenido al final o lectura + escritura.

```
mi_archivo = open("archivo-prueba.txt", "r")
mi_archivo2 = open("archivo-prueba2.txt", "w")
mi_archivo3 = open("archivo-prueba3.txt", "a")
mi_archivo4 = open("archivo-prueba4.txt", "r+")
```

Modos de abrir un archivo

por el tipo de archivo

- 't' se trata de un archivo de texto.
- 'b' permite escritura en modo binario
- 'U' define saltos de línea universales para el modo de lectura.

por el tipo de acceso

- 'r' es el modo de lectura.
- 'w' es un modo de escritura. En caso de existir un archivo, éste será sobreescrito.
- 'a' es un modo de escritura. En caso de existir un archivo, comienza a escribir al final del mismo.
- 'x' es un modo de escritura para crear un nuevo archivo. En caso de que el archivo exista se emitirá un error de tipo FileExistsError.
- 'r+' es un modo de escritura/lectura, también puede ser ‘w+’.

SIEMPRE QUE SE ABRA UN ARCHIVO HAY QUE CERRARLO

```
mi_archivo = open("archivo-prueba.txt", "r")
mi_archivo2 = open("archivo-prueba2.txt", "w")
mi_archivo3 = open("archivo-prueba3.txt", "a")
mi_archivo4 = open("archivo-prueba4.txt", "r+")

mi_archivo.close()
mi_archivo2.close()
mi_archivo3.close()
mi_archivo4.close()
```

Crear un archivo

Hay dos formas de crear un archivo, recomiendo se acostumbren a la cláusula with porque es la que más utilizan los devs, es como “más profesional”.

```
# Crear archivo

mi_archivo = open("archivo1.txt", "w+")# yo elijo casi siempre w+, salvo casos especiales
mi_archivo.write("Creando mi primer archivo")
mi_archivo.close()# no olvidar cerrar el archivo cuando dejamos de trabajar en él

with open("archivo2.txt", "w+") as mi_archivo2:
    mi_archivo2.write("Creando mi segundo archivo...")
    mi_archivo2.read()
    mi_archivo2.close()
```

Si el archivo ya está creado, con open() solo se lo agrega a la variable lógica pasándole la ruta de acceso en nuestra computadora como parámetro y desde allí se lo manipula.

```
with open(r"C:\Users\fravega\Desktop\archivo-prueba.txt", "w+") as mi_archivo3:
    mi_archivo3.write("Agregando este contenido a mi tercer archivo...")
    mi_archivo3.close()
```

Métodos de manipulación

- write(): escribe desde la primera línea del archivo, si éste ya tiene contenido, lo que se pase como parámetro lo sobreescribe. Importante, si el archivo se abre en “a”, cuando se utiliza el método write() éste agrega contenido al final del archivo, y agrega ese contenido tantas veces como se le dé run al programa.
- read(): lee el contenido del archivo, pero no lo imprime, para poder imprimirlo hay que almacenarlo en una variable e imprimirla.
- seek(): ubica el cursor en el índice que se le pase por parámetro.

- readlines(): convierte el contenido del archivo en una lista, el separador de cada elemento es el salto de línea, entonces cada elemento es una línea del archivo.
- readline(): lee la primera línea del archivo y como límite los caracteres que se pasen por parámetro.
- writelines(): escribe el archivo al final y agrega el contenido del parámetro.
- writable(): devuelve True si el archivo está abierto en modo escritura.
- readable(): devuelve True si el archivo está abierto en modo lectura.
- seekable(): devuelve True si es posible desplazarse dentro del archivo.
- tell(): devuelve la posición en la que se encuentra el puntero dentro del archivo.
- close(): cierra el archivo.

```
data = """
éste es el contenido
de mi primer archivo"""

with open("archivo.txt", "w+") as f:
    f.write("Escribiendo mi primer archivo...\n")
    f.write(data)
    f.seek(0)
    print(f.read())
    f.seek(0)
    for line in f.readlines():
        print(line)
    print(f.tell())
    print()
    f.seek(25)
    print(f.read())
    f.seek(15)
    print(f.read())
    f.seek(0)
    print(len(f.read()))
    f.writelines("hola")
    print(f.read())
    f.close()
```

Módulos

Un módulo *es un tipo de archivo .py o .pyc que contiene funciones, clases, objetos, variables e incluso otros módulos para ser utilizados en otros archivos de Python.*

Se utiliza para que el código sea más ordenado, limpio, legible y reutilizable.

Para poder usar el contenido de un módulo es necesario importarlo en el archivo en el que haremos uso del mismo.

formas de importar módulos

```python
# Siempre se importan los módulos y paquetes al inicio

import funciones_mate

# Utilizo el contenido:

suma = funciones_mate.sumar(1, 2)
```

```python
# Siempre se importan los módulos y paquetes al inicio

import funciones_mate as fm

# Utilizo el contenido:

suma = fm.sumar(1, 2)
```

```python
# Siempre se importan los módulos y paquetes al inicio

from funciones_mate import sumar, restar

suma = sumar(1, 2)
resta = restar(3, 4)
```

```python
# Siempre se importan los módulos y paquetes al inicio

from funciones_mate import *

suma = sumar(1, 3)
resta = restar(4, 5)
multiplicacion = producto(6, 7)
division = dividir(8, 4)
potenciacion = potencia(3, 5)
```

Paquetes

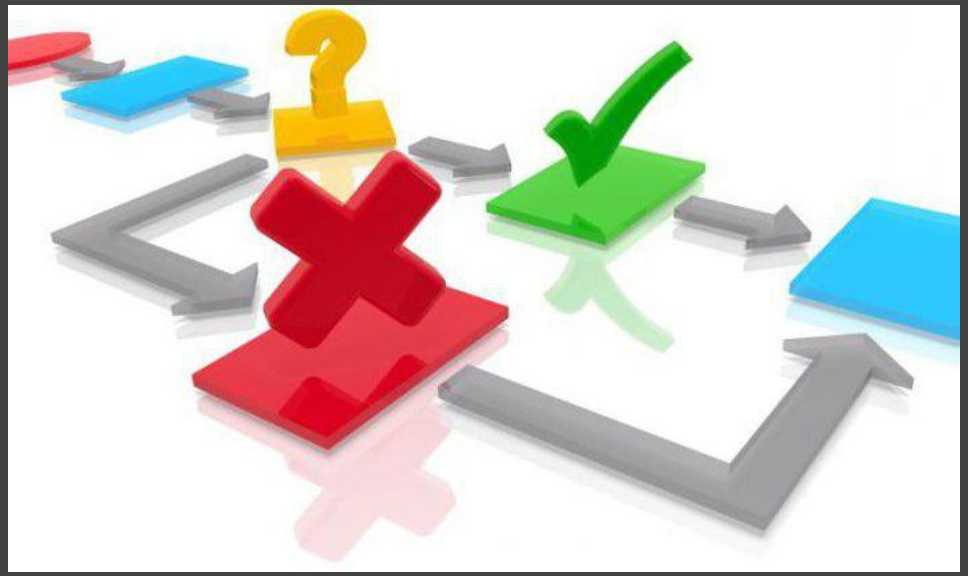
Un paquete *es un directorio o carpeta donde se almacenan diferentes módulos que están relacionados entre sí*.
Se crean con una carpeta que tenga sí o sí un archivo (file) llamado *__init__.py*. Dentro de un paquete puede haber otros paquetes, es importante que cada paquete interno tenga su propio file __init__.py.

Desde PyCharm, cuando dan click en New hay una opción que es Python Package y ya les crea un directorio con su file __init__.py.

Para importar es similar a los módulos

```python
from paquete import funciones_mate

from paquete import *

from paquete.funciones_mate import sumar, restar

from paquete.funciones_mate import *

from paquete import funciones_mate as fm
```

UNIDAD IX
EXCEPCIONES Y EXPRESIONES REGULARES



Excepciones

Una excepción es un *error que ocurre durante la ejecución de un programa pero que no está en la sintaxis del código*.

El ejemplo más común es la división por 0, si bien la sintaxis es correcta, el programa lanza un error porque, como sabemos, no se puede dividir por 0.

```python
# Funciones matemáticas
def sumar(a, b):
    return a + b

def restar(a, b):
    return a - b

def multiplicar(a, b):
    return a * b

def divifir(a, b):
    return a / b


num1 = 5
num2 = 0

print(sumar(num1, num2))
print(restar(num1, num2))
print(multiplicar(num1, num2))
print(divifir(num1, num2))
```

```
5
5
0
Traceback (most recent call last):
  File "excepciones.py", line 23, in <module>
    print(divifir(num1, num2))
  File "excepciones.py", line 14, in divifir
    return a / b
ZeroDivisionError: division by zero
```

Estos errores se pueden incluir en nuestra sintaxis, de esta manera se evita que el programa se rompa y pueda seguir su curso normalmente.

¿Cómo?

- ➔ Primero debemos saber el nombre del error, en este caso ZeroDivisionError.
- ➔ Segundo, debemos *incluir la sentencia try-except en nuestro código*, para poder contemplar este error.

```python
def dividir(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return f"No se puede dividir por cero. Operación fallida"
```

Aplicado try-except, lo que hacemos es indicar que, si está todo ok se efectúa el cálculo, y si aparece el error de ZeroDivisionError se devuelve el string de aviso.

Al ser ejecutada la función dividir(), se intentará realizar excepto que el parámetro b sea 0, y devolverá ese mensaje.

```
5
5
0
No se puede dividir por cero. Operación fallida
```

También se puede agregar la sentencia finally que no es obligatoria, pero siempre que esté se ejecutará, haya o no haya error.

```python
def dividir(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return f"No se puede dividir por cero. Operación fallida"
    finally:
        return "El cálculo ha finalizado"
```

Algunos ejemplos de las más comunes:

- *ValueError*: cuando se espera un tipo de dato y se recibe otro.
- *NameError*: cuando se recibe el nombre de una variable, función, clase u objeto pero no fue previamente definido.
- *TypeError*: cuando el tipo de dato que se quiere manipular debería ser otro.

Expresiones Regulares (RegEx)

Una expresión regular es una *secuencia de caracteres que forma un patrón que sirve para realizar búsquedas y filtrar datos*.

Para poder utilizar las regex, hay que importar la librería re:

```
import re
```

Hay muchísimas regex, vamos a ver las más utilizadas, pero para saber el listado total y cómo trabajar con cada una pueden ingresar a este link https://docs.python.org/3/library/re.html

(la maga de Python,)

```python
import re

texto = "Mi nombre es Magalí y tengo 28 años"
buscar = "nombre"

if re.search(buscar, texto):
    print(f"La cadena contiene {buscar} entre sus caracteres")
else:
    print(f"La cadena no contiene {buscar} entre sus caracteres")

texto_encontrado = re.search(buscar, texto)
print(texto_encontrado.start())  # me devolverá la posición del primer caracter de la cadena donde se encuentra
print(texto_encontrado.end())  # me devovlerá la posición del caracter siguiente al último de donde se encuentra
print(texto_encontrado.span())  # devuelve una tupla de dos números, el primero es start() y el segundo end()

cadena = "Me gusta la programación, estudié programación, trabajo en programación"
buscar = "programación"

print(re.findall(buscar, cadena))  # devolverá una lista con la cantidad de veces que aparece el patrón en la cadena
print(len(re.findall(buscar, cadena)))  # devolverá la cantidad de veces que aparece el patrón en la cadena
```

Como pueden observar, la sintaxis es:

```
re.método(patrón, variable)
```

Metacaracteres

Se conoce como metacaracteres a *aquellos que, dependiendo del contexto, tienen un significado especial para las expresiones regulares.*

Algunos autores los llaman “caracteres comodín”

Tipos

- Anclas
- Clases de caracteres
- Rangos
- Universal
- Cuantificadores

ANCLAS

Indican que lo que queremos encontrar se encuentra al principio o al final de la cadena. Combinándolas, podemos buscar algo que represente a la cadena entera.

- ➔ `^patrón`: **coincide con cualquier cadena que comience con patrón.**
- ➔ `patrón$`: **coincide con cualquier cadena que termine con patrón.**

CLASES DE CARACTERES

Se utilizan cuando se quiere buscar un caracter dentro de varias posibles opciones. Una clase se delimita entre corchetes y lista posibles opciones para el caracter que representa.

- `[abc]`: coincide con a, b, o c
- `[387ab]`: coincide con 3, 8, 7, a o b
- `niñ[oa]s`: coincide con niños o niñas.
- Para evitar errores, en caso de que queramos crear una clase de caracteres que contenga un corchete, debemos escribir una barra \ delante, para que el motor de expresiones regulares lo considere un caracter normal: la clase [ab\[] coincide con a, b y [.

RANGOS

Si queremos encontrar un número, podemos usar una clase como [0123456789], o podemos utilizar un rango. Un rango es una clase de caracteres abreviada que se crea escribiendo el primer caracter del rango, un guión y el último caracter del rango. Múltiples rangos pueden definirse en la misma clase de caracteres.

- ➔ `[a-c]`: **equivale a [abc]**
- ➔ `[0-9]`: **equivale a [0123456789]**
- ➔ `[a-d5-8]`: **equivale a [abcd5678]**
- ➔ **Es importante aclarar que si se quiere buscar un guión debe colocarse al principio o al final de la clase:**

  `[a4-] [-a4] [a\-4]`

RANGOS NEGADOS

Así como podemos listar los caracteres posibles en cierta posición de la cadena, también podemos listar caracteres que no deben aparecer.

➔ `[^abc]`: **coincide con cualquier caracter distinto a a, b y c**

UNIVERSAL

Coincide con cualquier caracter.

➔ `\".\"`

CUANTIFICADORES

Son caracteres que multiplican el patrón que les precede.

- `**?**`: coincide con cero o una ocurrencia del patrón. Dicho de otra forma, hace que el patrón sea opcional
- `**+**`: coincide con una o más ocurrencias del patrón
- `*****`: coincide con cero o más ocurrencias del patrón.
- `**{x}**`: coincide con exactamente x ocurrencias del patrón
- `**{x, y}**`: coincide con al menos x y no más de y ocurrencias. Si se omite x, el mínimo es cero, y si se omite y, no hay máximo. Esto permite especificar a los otros como casos particulares: ? es {0,1}, + es {1,} y * es {,} o {0,}.

```python
lista = ["Maria Lopez",
         "Lucia Perez",
         "Orlando Gomez",
         "Violeta Gutierrez",
]

for elemento in lista:
    if re.findall("^Maria", elemento):
        print(elemento)
    elif re.findall("Perez$", elemento):
        print(elemento)
    elif re.findall("[o-t]", elemento):
        print(elemento)
    elif re.findall("^[a-l]", elemento):
        print(elemento)
    elif re.findall("[abcd]", elemento):
        print(elemento)
```

```python
vuelos = [
    "4M7901",
    "4M7905",
    "AR1529",
    "OB709"
]

for i in vuelos:
    if re.findall("^4M", i):
        print(f"El vuelo {i} es de LATAM")
    elif re.findall("^AR", i):
        print(f"El vuelo {i} es de Aerolíneas Argentinas")
    elif re.findall("^OB", i):
        print(f"El vuelo {i} es de Boliviana de Aviación")
```

```python
vuelos_ar = [
    "AR1133",
    "AR1305",
    "AR1293",
    "AR1371",
    "AR1303",
    "AR2709",
    "AR2701",
    "AR2602"
]

for i in vuelos_ar:
    if re.findall("AR[1100-1800]", i):
        print(f"El vuelo {i} es un vuelo internacional")
    else:
        print(f"El vuelo {i} es un vuelo de cabotaje")
```