RESET

PYTHON

TALLER DE PROGRAMACIÓN

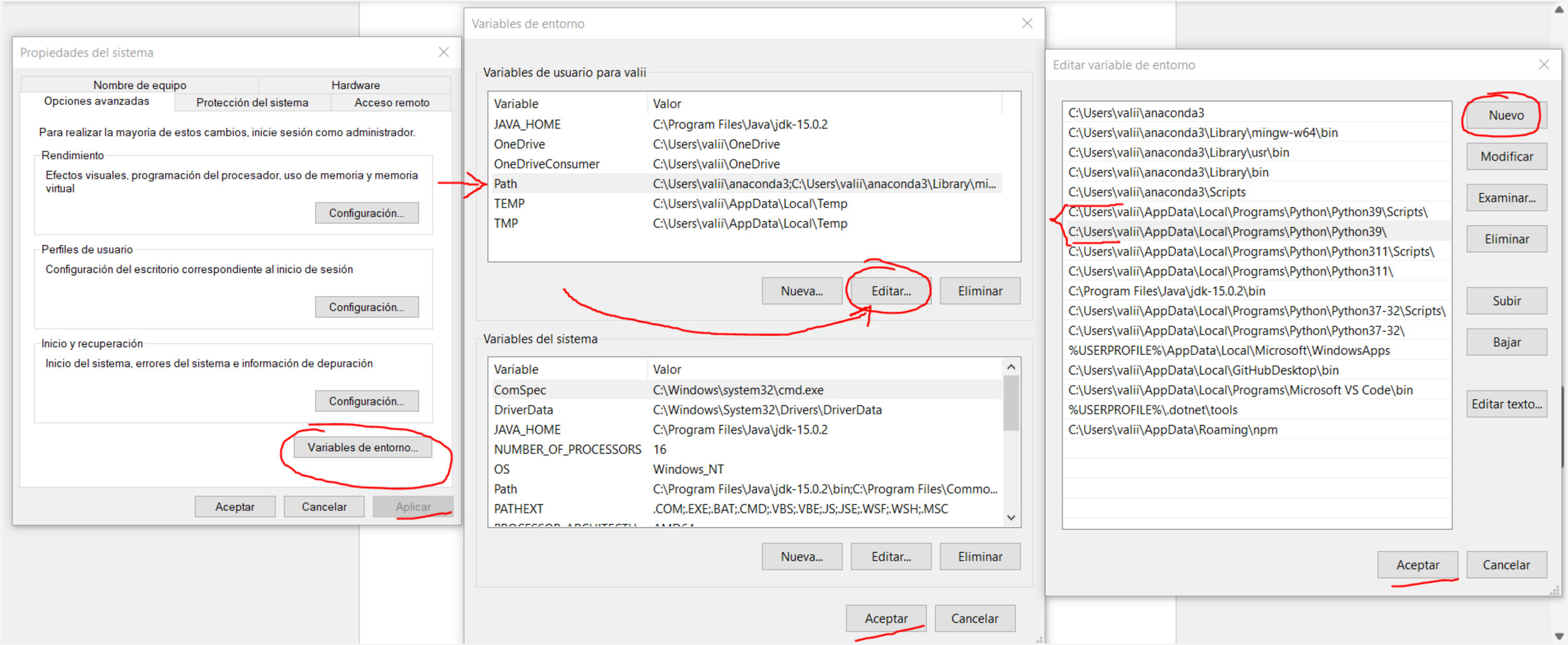
VALENTIN MUSAT

PARTE 1: INSTALACIÓN

PYTHON 3.X --> WWW.PYTHON.ORG/DOWNLOADS

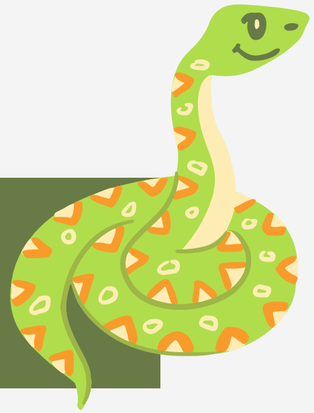
O directamente...

GOOGLE COLAB --> COLAB.RESEARCH.GOOGLE.COM



INSTALACIÓN: CONFIGURACIÓN DEL PATH

WWW.JETBRAINS.COM/PYCHARM/DOWNLOAD



Versiones:

- **Community: Gratis**
- **Professional: Correo institucional**

Nuevo proyecto:

NOMBRE

ENTORNO

INTÉRPRETE

INSTALACIÓN: PYCHARM

PARTE 2: VARIABLES

EN PYTHON SON DE TIPO DINÁMICO = PUEDE CAMBIAR EL TIPO DE VARIABLE DURANTE LA EJECUCIÓN

```
entero = 34
flotante = 3.14
booleano = True
cadena = 'Hola123'
```

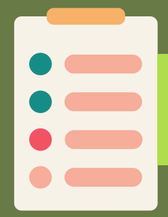
TIPOS

- **ENTEROS:** POSITIVOS Y NEGATIVOS
- **FLOTANTES:** CON DECIMALES
- **BOOLEANOS:** TRUE Y FALSE
- **CADENAS:** STRINGS DE CATACTERES

PERMITEN

- OPERAR ENTEROS Y FLOTANTES
- BOOLEANOS ACTÚAN NUMÉRICAMENTE
- CONCATENACIÓN DE CADENAS

VARIABLES: VARIABLES BÁSICAS



TEST1.PY

COLECCIONES

LISTA:
ORDENADA Y MODIFICABLE. ADMITE DUPLICADOS

TUPLA:
ORDENADA Y NO MODIFICABLE. ADMITE DUPLICADOS

SET:
NO ORDENADO Y ELEMENTOS NO MODIFICABLES. NO ADMITE DUPLICADOS

DICCIONARIO:
ORDENADO Y MODIFICABLE. NO ADMITE DUPICADOS

★ *LAS CADENAS TAMBIÉN SON LISTAS, DE CARACTERES

```
lista = [6, 2, 1, 4, 6, 3, 1]
lista_mixta = [2, 4.65, "Numero", True, 4 == 2]
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

tupla = (2, 1, 9, True, "Reset")
set_1 = {3, 6, "NotANumber", False, -1}
diccionario = {"Id": 34, "Nombre": "Roberto", "Vivo": True, 0: True}
```

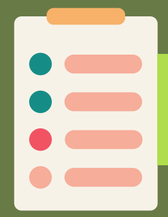
ORDENADA: LOS ELEMENTOS NO CAMBIAN SU ORDEN Y SON INDEXABLES

MODIFICABLE: AÑADIR, ELIMINAR O MODIFICAR ELEMENTOS UNA VEZ CREADA LA COLECCIÓN

DUPLICADOS: SI NO LOS ADMITE, SE SOBRESCRIBEN

VARIABLES: COLECCIONES

TODOS SUS MÉTODOS AQUÍ



TEST2.PY

CONDICIONALES

IF CONDICIONAL:

ACCIÓN A REALIZAR

ELIF CONDICIONAL:

ACCIÓN A REALIZAR

...

ELSE:

ACCIÓN POR DEFECTO

operador	comparación
==	es igual que
!=	es distinto de
<	es menor que
<=	es menor o igual que
>	es mayor que
>=	es mayor o igual que

★ FALSE = 0
TRUE != 0

Operador	Descripción	Uso
and	Devuelve True si ambos operandos son True	a and b
or	Devuelve True si alguno de los operandos es True	a or b
not	Devuelve True si alguno de los operandos False	not a
in	Devuelve True si un valor se encuentra en una secuencia	7 in [7,3,5] devuelve True
not in	Devuelve True si un valor no se encuentra en una secuencia	7 not in [7,3,5] devuelve False

CONDICIONALES

MATCH KEYWORD:

CASE CASO_1:

ACCIÓN A REALIZAR

CASE CASO_2:

ACCIÓN A REALIZAR

CASE _:

ACCIÓN POR DEFECTO

BUCLES

WHILE CONDICIONAL:

BUCLE

FOR VARIABLE **IN** ITERABLE:

BUCLE

FOR INDEX **IN RANGE**(0, MAX):

BUCLE CON ITERABLE[INDEX]

BREAK, CONTINUE, ELSE

TEST3.PY

PARTE 3: FUNCIONES

```python
def suma(a, b=4):
    resultado = a + b
    return resultado
```

```python
print(suma(3, 4))
print(suma(3))
print(suma(3, b=8))
print(suma(a=2, b=6))
```

PARÁMETROS

- POSICIONALES
- PALABRAS CLAVE
- AMBOS

DESEMPAQUETADO

```python
def imprimir_valores(*args, **kwargs):
    # Extraemos los parámetros posicionales
    for arg in args:
        print(arg)
    # Extraemos los parámetros de palabras clave
    for key, value in kwargs.items():
        print(f"{key}: {value}")
```

OTROS TIPOS

- LAMBDA

```python
cuadrado = lambda x: x ** 2
print(cuadrado(5))  # Imprime 25
```

- RECURSIVAS — FACTORIAL, FIBONACCI
- MÓDULOS — MATH, STRING, OS, RE

FUNCIONES: INTRODUCCIÓN

MODULOS EXTRA.PY

TEST4.PY

EJERCICIO 1

ES_PALINDROMO(PALABRA)

- PARÁMETRO: STRING
- SALIDA: BOOLEANO

COMPROBAR SI LOS CARACTERES QUE FORMAN LA CADENA SON UN PALÍNDROMO. NO HAY QUE TENER EN CUENTA MAYÚSCULAS NI ESPACIOS.
CONSEJOS: LAS FUNCIONES DE LOWER() Y REPLACE() DE STRING PUEDEN SERVIR PARA LIDIAR CON MAYÚSCULAS Y ESPACIOS. RECORRER UNA CADENA ES COMO RECORRER UNA LISTA (SLICING).

EJERCICIO 2

GENERAR_PRIMOS_HASTA(N)

- PARÁMETRO: ENTERO
- SALIDA: LISTA

GENERAR UNA LISTA CON TODOS LOS NÚMEROS PRIMOS (SIN CONTAR EL 1) HASTA EL NÚMERO QUE ENTRA COMO PARÁMETRO (NO INCLUIDO).
CONSEJOS: RECORDAD LA FORMA DE GENERAR LISTAS UTILIZANDO EL BUCLE **FOR.** SE PUEDEN UTILIZAR FUNCIONES AUXILIARES ENCAPSULADAS QUE DEFINAN UNA TAREA, COMO UNA CONDICIÓN.

EJERCICIO 3

ENCONTRAR_MAX_MIN(LISTA)

- PARÁMETRO: LISTA
- SALIDA: DOS ENTEROS

ENCONTRAR EL VALOR MÁXIMO Y MÍNIMO DE UNA LISTA DE NÚMEROS ENTEROS. SE PUEDEN DEVOLVER AMBOS EN UN MISMO RETURN.
CONSEJOS: LAS FUNCIONES NO DEFINIDAS POR EL USUARIO Y QUE NO DEPENDEN DE OTRAS CLASES O IMPORTS SE LLAMAN BUILTINS (COMO PRINT). PRUEBA A UTILIZAR MAX() Y MIN().

FUNCIONES: EJERCICIOS

EJERCICIOS FUNCIONES.PY

SOLUCIONES FUNCIONES.PY

PARTE 4: CLASES Y OBJETOS

PROGRAMACIÓN ORIENTADA A OBJETOS (POO) = EFICIENCIA, ENCAPSULACIÓN, ABSTRACCIÓN

NAMESPACE

- INDICAN LA **REFERENCIA** A UN OBJETO
- EL MÁS GENERAL ES EL **BUILTIN** (STRING)
- CADA MÓDULO (.PY) TIENE UN NAMESPACE **GLOBAL** CON LOS OBJETOS DEFINIDOS
- AL INVOCAR UNA FUNCIÓN, SE CREA UN NAMESPACE **LOCAL**, EL PRIMER ÁMBITO

CLASE

- "MOLDE" FUNDAMENTAL DE UN **OBJETO**
- DEFINE **ATRIBUTOS** (DATOS) Y **MÉTODOS** (FUNCIONES) QUE LOS USAN
- EL **CONSTRUCTOR** DE UNA CLASE PERMITE CREAR SUS DATOS
- LOS DATOS CREADOS DE UNA CLASE FORMAN OBJETOS. EL VALOR DE SUS ATRIBUTOS ES EL ESTADO DEL OBJETO. SUS MÉTODOS REALIZAN OPERACIONES (GETTERS O SETTERS)

```python
class Personaje:
    # Variable de clase
    contador_personajes = 0

    def __init__(self, nombre, salud, ataque):
        # Variables de instancia
        self.nombre = nombre
        self.salud = salud
        self.ataque = ataque
        Personaje.contador_personajes += 1

    def mostrar_estado(self):
        print(f"{self.nombre}: Salud={self.salud}, Ataque={self.ataque}")


# Crear una instancia de la clase Personaje
jugador1 = Personaje("Héroe", 100, 20)

# Acceder a los atributos y llamar a un método
print(jugador1.nombre)
jugador1.mostrar_estado()

# Acceder a una variable de clase
print(f"Personajes creados: {Personaje.contador_personajes}")
```

ATRIBUTO DE CLASE

CONSTRUCTOR

ATRIBUTO DE INSTANCIA

MÉTODO

INSTANCIA

ACCESO A ATRIBUTO Y MÉTODO

ACCESO A ATRIBUTO DE CLASE

CLASES Y OBJETOS: ESTRUCTURA DE UNA CLASE

```python
class Personaje:
    # Variable de clase
    contador_personajes = 0

    def __init__(self, nombre, salud, ataque):
        # Variables de instancia
        self.nombre = nombre
        self._salud = salud
        self._ataque = ataque
        Personaje.contador_personajes += 1

    def mostrar_estado(self):
        print(f"{self.nombre}: Salud={self._salud}, Ataque={self._ataque}")

    @property
    def salud(self):
        return self._salud

    @salud.setter
    def salud(self, value):
        self._salud = value


# Crear una instancia de la clase Personaje
jugador1 = Personaje("Héroe", 100, 20)

# Acceder a los atributos y llamar a un método
print(jugador1.nombre)
print(jugador1.salud)
print(jugador1._ataque)
jugador1.salud += 25
```

ATRIBUTOS DE INSTANCIA PRIVADOS CON _ SI NO SON PÚBLICOS

MÉTODO GETTER DEL ATRIBUTO. CON @ ES POSIBLE LLAMARLO COMO SI FUERA PROPIEDAD

MÉTODO SETTER DEL ATRIBUTO. CON @ SE PUEDE ASIGNAR COMO UN VALOR

ATAQUE ES PRIVADO Y NO TIENE MÉTODOS, NO SE PUEDE ACCEDER. SALUD SE PUEDE MOSTRAR Y MODIFICAR

CLASES Y OBJETOS: ATRIBUTOS PRIVADOS

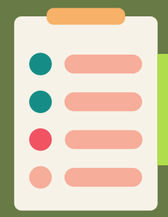
HAY SESIONES DE DYD EN KW

```python
class Guerrero(Personaje):
    def __init__(self, nombre, salud, ataque, arma):
        super().__init__(nombre, salud, ataque)
        self.arma = arma

    def mostrar_arma(self):
        print(f"{self.nombre} tiene un arma: {self.arma}")


# Crear una instancia de la clase derivada Guerrero
guerrero1 = Guerrero("Conquistador", 120, 25, "Espada")
guerrero1.mostrar_estado()
guerrero1.mostrar_arma()
```

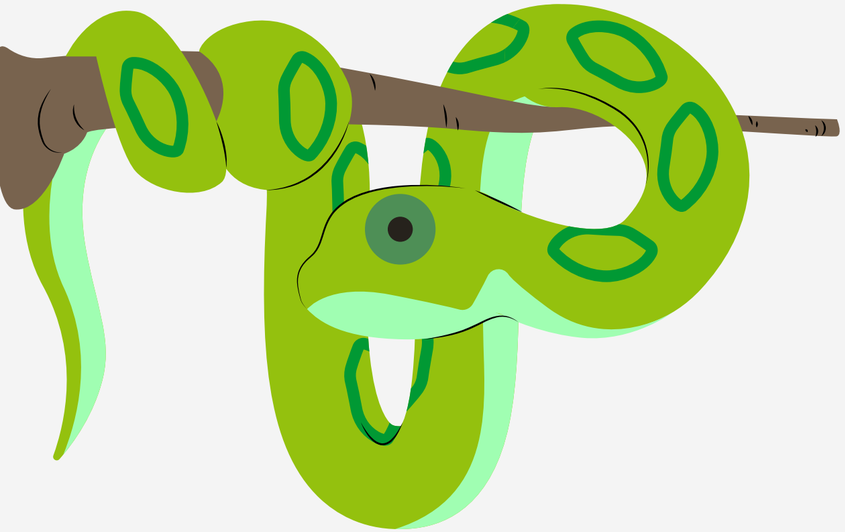
HERENCIA CON EL CONSTRUCTOR DE LA CLASE PROGENITOR.
TIENE ATRIBUTOS PROPIOS

MÉTODO PROPIO DE GUERRERO

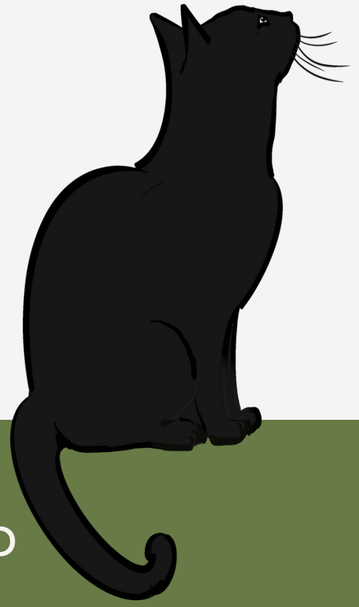
CONSTRUCTOR DE GUERRERO: ATRIBUTOS DEL PROGENITOR Y PROPIOS DE LA CLASE

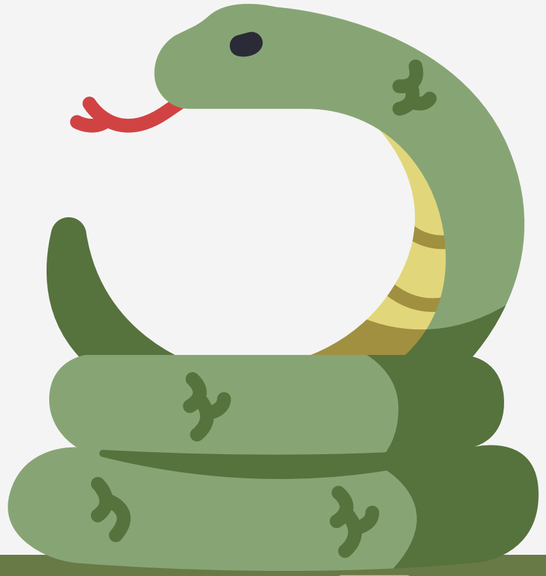


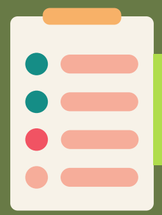
TEST5.PY



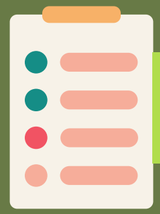
EJERCICIO: “EL ZOOLÓGICO”





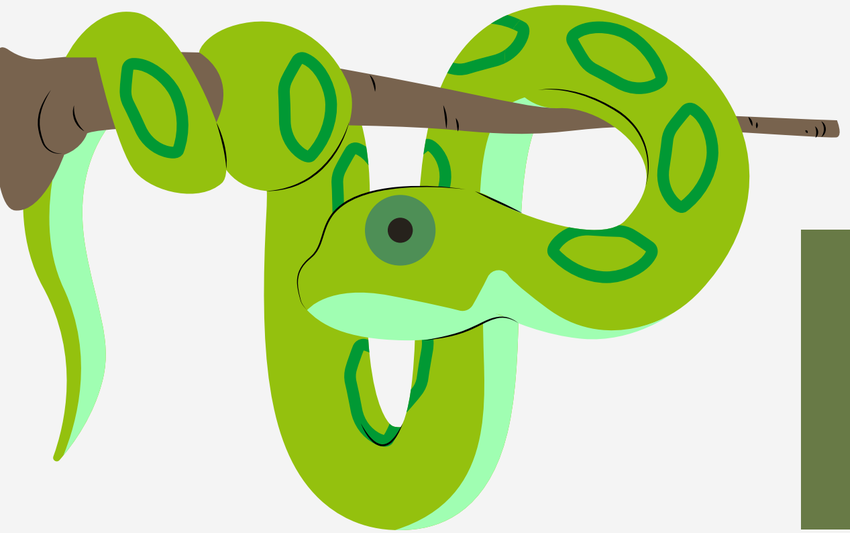


EJERCICIO OBJETOS.PY



SOLUCION OBJETOS.PY

PANDAS EXCEL.PY



¡¡MUCHAS GRACIAS!!