**DOCUMENTACIÓN**

**CODERS 155**

**INTEGRANTES**:

Tenorio Martínez Leonardo Daniel

Rodríguez Gómez Alan Gael

Bautista Gonzalez Hilda Julianna

Torres López Julio Axel

Arias Sandoval Carlos Max

import mysql.connector

from getpass import getpass

import os

import time

import math

from datetime import datetime

from werkzeug.security import generate\_password\_hash, check\_password\_hash

db = None

cursor = None

sesion = None

opc\_selected = None

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

class bcolors:

OK = '\033[92m' **#VERDE**

WARNING = '\033[93m' **#AMARILLO**

FAIL = '\033[91m' **#ROJO**

RESET = '\033[0m' **#REINICIAR COLOR**

**def limpiar\_pantalla():**

**"""**

**Limpia la pantalla de la consola.**

**Utiliza el comando 'cls' en Windows y 'clear' en Unix.**

"""

os.system('cls' if os.name == 'nt' else 'clear')

**def inicializar\_database():**

**"""**

**Establece la conexión con la base de datos y crea el cursor.**

**"""**

try:

global db, cursor

db = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="proyecto"

)

cursor = db.cursor()

if db.is\_connected():

print("CONEXION EXITOSA")

print("INICANDO...")

time.sleep(1.5)

limpiar\_pantalla()

except Exception as ex:

print(ex)

print("POR FAVOR CONTACTE CON UN ADMINISTRADOR")

time.sleep(4)

**def cerrar\_conex\_db():**

**"""**

**Cierra la conexión con la base de datos y el cursor.**

**"""**

if cursor:

cursor.close()

if db:

db.close()

**def register():**

**"""**

**Registra un nuevo usuario en la base de datos.**

**Solicita al usuario un nombre de usuario y contraseña, verifica que las contraseñas coincidan**

**y que el nombre de usuario no esté repetido. Luego, cifra la contraseña y la guarda en la base de datos.**

**"""**

while True:

while True:

limpiar\_pantalla()

print("---------------------------------------")

print("| Registro |")

print("---------------------------------------")

username = input("Nombre de usuario: ")

password = input("Contraseña: ")

password\_verify = input("Vuelve a introducir la contraseña: ")

if password == password\_verify:

print("Las contraseñas coinciden")

time.sleep(1)

break

else:

print("Las contraseñas no coinciden, vuelve a intentarlo")

time.sleep(1)

limpiar\_pantalla()

sql = "SELECT nombre\_usuario FROM usuarios WHERE nombre\_usuario = %s"

cursor.execute(sql,(username,)) # type: ignore

result = cursor.fetchone() # type: ignore

if result is None:

print("Registrando...")

time.sleep(3)

rol = 'user'

**##Se cifra la contraseña**

cripted\_password = generate\_password\_hash(password)

**##La contraseña se carga en la db**

sql = "INSERT INTO usuarios (nombre\_usuario, contraseña, rol, fecha\_hora) VALUES (%s, %s, %s, %s)"

values = (username, cripted\_password, rol, timestamp)

cursor.execute(sql, values) **# type: ignore**

db.commit() # **type: ignore**

**#Se indica si el registro fue exitoso**

limpiar\_pantalla()

print("Usuario registrado correctamente.")

time.sleep(1)

**# Registrar en el historial**

registrar\_historial(f"Registro de usuario", username)

break

else:

print("El usuario ya existe, vuelve a intentarlo")

time.sleep(1)

**def login():**

global role, username, password

**"""**

**Realiza el proceso de inicio de sesión comparando el nombre de usuario y contraseña ingresados**

**con los datos almacenados en la base de datos.**

**Returns:**

**bool: True si el inicio de sesión es exitoso, False si falla.**

**"""**

limpiar\_pantalla()

print("---------------------------------------")

print("| Login |")

print("---------------------------------------")

username = input("Nombre de usuario: ")

password = getpass("Contraseña: ")

sql = "SELECT contraseña, rol FROM usuarios WHERE nombre\_usuario = %s"

cursor.execute(sql, (username,)) **# type: ignore**

result = cursor.fetchone() **# type: ignore**

if result is None:

print("Nombre de usuario incorrecto. Vuelve a intentarlo")

time.sleep(1)

else:

hashed\_password = result[0]

role = result[1]

if check\_password\_hash(hashed\_password, password):

limpiar\_pantalla()

if role == "admin":

print("¡Bienvenido administrador!")

menu\_home() **# Se pasa el valor de 'role' a la función menu\_home()**

else:

menu\_home()

**# Registrar en el historial**

registrar\_historial(f"Inicio de sesión", username)

return True

else:

print("Contraseña incorrecta.")

time.sleep(1.5)

return False

**def change\_password():**

**"""**

**Permite al usuario cambiar su contraseña actual ingresando su nombre de usuario, contraseña actual y la nueva contraseña.**

**"""**

limpiar\_pantalla()

print("---------------------------------------")

print("| Cambiar contraseña |")

print("---------------------------------------")

username = input("Nombre de usuario: ")

current\_password = getpass("Contraseña actual: ")

new\_password = getpass("Nueva contraseña: ")

sql = "SELECT contraseña FROM usuarios WHERE nombre\_usuario = %s"

cursor.execute(sql, (username,)) **# type: ignore**

result = cursor.fetchone() **# type: ignore**

if result is None:

print("Nombre de usuario incorrecto.")

else:

hashed\_password = result[0]

if check\_password\_hash(hashed\_password, current\_password):

new\_hashed\_password = generate\_password\_hash(new\_password)

update\_sql = "UPDATE usuarios SET contraseña = %s WHERE nombre\_usuario = %s"

cursor.execute(update\_sql, (new\_hashed\_password, username)) # type: ignore

db.commit() **# type: ignore**

limpiar\_pantalla()

print("Contraseña actualizada correctamente.")

**# Registrar en el historial**

registrar\_historial(f"Cambio de contraseña", username)

else:

limpiar\_pantalla()

print("Contraseña incorrecta.")

**def opcion\_incorrecta():**

**"""**

**Maneja la situación en la que se selecciona una opción incorrecta en el menú.**

**Imprime un mensaje de error indicando que la opción seleccionada es incorrecta y**

**motiva al usuario a intentarlo nuevamente.**

**"""**

print("Error: Opción Incorrecta")

print("La opción seleccionada no es un número o no es una opción disponible")

print("VUELVE A INTENTARLO")

**def menu\_principal():**

**"""**

**Muestra el menú principal con opciones para registro, inicio de sesión, cambio de contraseña y salir.**

**"""**

limpiar\_pantalla()

print("---------------------------------------")

print("| MENU PRINCIPAL |")

print("|-------------------------------------|")

print("| 1.- Registro |")

print("| 2.- Login |")

print("| 3.- Cambiar contraseña |")

print("| 4.- Salir |")

print("|-------------------------------------|")

print("|Consult Date:", timestamp, " |")

print("---------------------------------------")

**def seleccion():**

**"""**

**Esta función permite al usuario seleccionar una opción del menú.**

**Solicita al usuario que ingrese la opción seleccionada y valida que sea un número entero.**

**Si el usuario ingresa un valor no válido, se llama a la función 'opcion\_incorrecta',**

**se realiza una pausa de 0.5 segundos y se limpia la pantalla de la consola.**

**Devuelve el valor de la opción seleccionada.**

**"""**

global opc\_selected

while True:

try:

opc\_selected = int(input("Escribe la opción a seleccionar: "))

return opc\_selected

except ValueError:

limpiar\_pantalla()

opcion\_incorrecta()

break

**def enviar\_mensaje():**

**"""Esta funcion nos permite ver el mensaje enviado del usuario"""**

limpiar\_pantalla()

print("---------------------------------------")

print("| Enviar Mensaje |")

print("---------------------------------------")

remitente = username

destinatario = input("Nombre de usuario destinatario: ")

mensaje = input("Mensaje: ")

sql = "SELECT \* FROM usuarios WHERE nombre\_usuario = %s"

cursor.execute(sql, (destinatario, )) **# type: ignore**

result = cursor.fetchone() **# type: ignore**

if result:

sql = "INSERT INTO mensajes (remite, destinatario, mensaje, fecha\_hora) VALUES (%s, %s, %s, %s)"

values = (remitente, destinatario, mensaje, timestamp)

cursor.execute(sql, values) **# type: ignore**

db.commit() **# type: ignore**

limpiar\_pantalla()

print("Mensaje enviado correctamente.")

**# Registrar en el historial**

registrar\_historial(f"Envío de mensaje a {destinatario}", remitente)

else:

print(bcolors.WARNING + "El usuario no existe. Ingresa un usuario correcto" + bcolors.RESET)

**def ver\_mensajes():**

**"""Nos permite ver el mensaje enviado"""**

sql = "SELECT \* FROM mensajes WHERE destinatario = %s"

cursor.execute(sql, (username,)) **# type: ignore Pasa el parámetro como una tupla**

result = cursor.fetchall() **# type: ignore**

if result:

limpiar\_pantalla()

print("---------------------------------------")

print(f" Mensajes recibidos ({len(result)}) ")

print("---------------------------------------")

for row in result:

print("Remitente:", row[1])

print("Mensaje:", row[3])

print("Fecha:", row[4])

print("----------------------------------------------------")

else:

print("Aún no has recibido mensajes")

**def mostrar\_historial():**

**"""**

**Esta función muestra el historial guardado en la base de datos.**

**Realiza una consulta SQL para obtener el historial del usuario específico.**

**Imprime el historial en la consola, mostrando el ID de referencia, la acción realizada,**

**la fecha y hora de la acción.**

**Si no hay registros en el historial, se imprime un mensaje indicando que no hay registros.**

**"""**

sql = "SELECT \* FROM historial WHERE usuario = %s ORDER BY fecha\_hora DESC" **# Añadir %s para incluir el valor de 'username'**

cursor.execute(sql, (username,)) **# type: ignore**

result = cursor.fetchall() **# type: ignore**

if result:

limpiar\_pantalla()

print("--------------------------------------------")

print("| Historial |")

print("--------------------------------------------")

for row in result:

print("ID de referencia:", row[0])

print(bcolors.OK + "Acción:", row[1] + bcolors.RESET)

print("Fecha y hora:", row[2])

print("--------------------------------------------")

else:

print("No hay registros en el historial.")

**def borrar\_historial():**

**"""**

**Esta función borra todo el historial almacenado en la base de datos.**

**Solicita confirmación al usuario antes de realizar la operación.**

**Si el usuario confirma, se ejecuta una consulta SQL para borrar todos los registros de la tabla 'historial'.**

**Se confirman los cambios en la base de datos y se imprime un mensaje de confirmación.**

**Si el usuario cancela la operación, se imprime un mensaje indicando que la operación fue cancelada.**

**"""**

limpiar\_pantalla()

confirmacion = input(bcolors.WARNING + "¿Estás seguro de que quieres borrar el historial? (s/n): " + bcolors.RESET)

if confirmacion.lower() == 's':

sql = "TRUNCATE TABLE historial"

cursor.execute(sql) **# type: ignore**

db.commit() **# type: ignore**

time.sleep(1)

print("Historial borrado correctamente.")

else:

print("Operación cancelada.")

time.sleep(1)

**def mostrar\_usuarios():**

**"""**

**Esta función muestra los usuarios registrados en la base de datos.**

**Realiza una consulta SQL para obtener todos los usuarios ordenados por fecha y hora.**

**Imprime los usuarios en la consola, mostrando el ID de referencia, la acción realizada,**

**la fecha y hora de la acción, y el nombre de usuario.**

**Si no hay registros de usuarios, se imprime un mensaje indicando que no hay registros.**

**"""**

limpiar\_pantalla()

sql = "SELECT \* FROM usuarios ORDER BY id DESC"

cursor.execute(sql) # type: ignore

result = cursor.fetchall() **# type: ignore**

if result:

print("------------------------------------------------")

print("| Uuarios registrados |")

print("------------------------------------------------")

limpiar\_pantalla()

for row in result:

print("ID del usuario:", row[0])

print("Usuario:", row[1])

print("Rol:", row[3])

print("Fecha y hora de registro :", row[4])

print("---------------------------------------")

else:

print("No hay registros en el historial de usuarios.")

**def eliminar\_usuario():**

**"""**

**Esta función permite eliminar a un usuario del sistema.**

**Solicita el nombre de usuario a eliminar y la contraseña de administrador.**

**Si la contraseña de administrador coincide, se solicita confirmación para eliminar al usuario.**

**Si la confirmación es 's', se ejecuta una consulta SQL para eliminar al usuario de la tabla 'usuarios',**

**se confirman los cambios en la base de datos y se imprime un mensaje de confirmación.**

**Si la confirmación es 'n' o el usuario cancela la operación, se imprime un mensaje indicando que la operación fue cancelada.**

**Si la contraseña de administrador no coincide, se imprime un mensaje indicando que la contraseña no es correcta.**

**"""**

limpiar\_pantalla()

user = input(bcolors.WARNING + "Ingresa el nombre de usuario que deseas eliminar del sistema: " + bcolors.RESET)

verify = getpass(bcolors.WARNING + "Ingresa tu contraseña administrador: " + bcolors.RESET)

**# Verificar si el usuario existe**

sql = "SELECT \* FROM usuarios WHERE nombre\_usuario = %s;"

cursor.execute(sql, (user,)) **# type: ignore**

result = cursor.fetchone() **#type: ignore # Obtener el primer resultado**

if result:

**# El usuario existe en el sistema**

if verify == password:

print(bcolors.OK + "\nLas contraseñas coinciden\n" + bcolors.RESET)

time.sleep(0.5)

confirmacion = input(bcolors.FAIL + "¿Estás seguro de que quieres eliminar al usuario? (s/n): " + bcolors.RESET)

if confirmacion.lower() == 's':

time.sleep(1)

print(bcolors.WARNING + "Eliminando usuario..." + bcolors.RESET)

sql = "DELETE FROM usuarios WHERE nombre\_usuario = %s;"

cursor.execute(sql, (user,)) **# type: ignore**

db.commit() **# type: ignore**

time.sleep(2)

print(bcolors.OK + "Usuario eliminado exitosamente" + bcolors.RESET)

**# Registrar en el historial**

registrar\_historial(f"Eliminación de usuario", user)

else:

print("Operación cancelada.")

time.sleep(1)

else:

print("La contraseña no es correcta.")

print("Cancelando operación...")

time.sleep(1)

else:

**# El usuario no existe en el sistema**

print(bcolors.WARNING + "El usuario no existe" + bcolors.RESET)

**def mostrar\_historial\_admin():**

**"""**

**Muestra el historial de acciones realizadas por el administrador.**

**Consulta la base de datos para obtener el historial ordenado por fecha y hora,**

**e imprime la información en la pantalla.**

**"""**

sql = "SELECT \* FROM historial ORDER BY fecha\_hora ASC"

cursor.execute(sql) **# type: ignore**

result = cursor.fetchall() **# type: ignore**

if result:

limpiar\_pantalla()

print("---------------------------------------")

print("| Historial de Usuarios |")

print("---------------------------------------")

for row in result:

print("ID de referencia:", row[0])

print(bcolors.OK + "Acción:", row[1] + bcolors.RESET)

print("Fecha y hora:", row[2])

print(bcolors.OK + "Usuario:", row[3] + bcolors.RESET)

print("---------------------------------------")

else:

print("No hay registros en el historial de usuarios.")

**def menu\_admin():**

**"""**

**Muestra el menú de opciones para el administrador.**

**El administrador puede elegir entre diferentes opciones, como mostrar el historial,**

**borrar el historial, eliminar un usuario o salir del programa.**

**"""**

while True:

print("---------------------------------------")

print("| MENÚ ADMINISTRADOR |")

print("|-------------------------------------|")

print("| 1.- Mostrar Historial |")

print("| 2.- Borrar Historial |")

print("| 3.- Eliminar usuario |")

print("| 4.- Mostrar usuarios |")

print("| 5.- Salir |")

print("---------------------------------------")

seleccion()

if opc\_selected == 1:

mostrar\_historial\_admin()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 2:

borrar\_historial()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 3:

eliminar\_usuario()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 4:

mostrar\_usuarios()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 5:

break

else:

time.sleep(1)

limpiar\_pantalla()

**def menu\_figuras():**

**""" Muestra el menu de figuras**

**Nos pregunta que figura deseamos realizar utilizando numeros enteros"""**

while True:

print(" Generar figuras ")

print("-------------------------------------")

print("| ¿Que figura necesitas? |")

print("|-----------------------------------|")

print("| 1-Triángulos |")

print("| 2-Cuadrilateros |")

print("| 3-Poligonos |")

print("| 4-Circulos (Ovalos) |")

print("-------------------------------------")

seleccion()

if opc\_selected == 1:

limpiar\_pantalla()

menu\_triangulos()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 2:

limpiar\_pantalla()

menu\_cuadrilateros()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 3:

limpiar\_pantalla()

menu\_poligonos()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 4:

limpiar\_pantalla()

menu\_circulo()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

else:

opcion\_incorrecta()

time.sleep(2)

**def generar\_isoceles():**

**"""Funcion en la que se realiza el proceso de un triangulo isoseles"""**

limpiar\_pantalla()

la1= int(input("ing"))

la2=int(input("ing"))

la3=int(input("ing"))

if la1 + la2 > la3 and la1 + la3 > la2 and la2 + la3 > la1:

if la1 == la2 or la1 == la3 or la2 == la3:

if la1 == la2:

ladigu = la1

elif la1 == la3:

ladigu = la1

else:

ladigu = la2

for i in range(ladigu):

for j in range(ladigu - i - 1):

print(" ", end="")

for k in range(2 \* i + 1):

print("\*", end="")

print()

**def generar\_equilatero():**

**"""Funcion en la que se realiza el proceso para hacer un triangulo equilatero"""**

limpiar\_pantalla()

longitud = int(input("Ingrese la longitud de los lados del triángulo equilátero: "))

for i in range(longitud):

print(' ' \* (longitud - i - 1), end='')

print('\* ' \* (i + 1))

**def generar\_escaleno():**

**"""Funcion en la que se realiza el proceso para hacer un triangulo escaleno """**

lado1 = int(input("Ingresa la medida del primer lado"))

lado2 = int(input("Ingresa la medida del segundo lado"))

lado3 = int(input("Ingresa la medida del tercer lado"))

**# Verificar si los lados forman un triángulo válido**

if lado1 + lado2 <= lado3 or lado2 + lado3 <= lado1 or lado1 + lado3 <= lado2:

print("Los lados ingresados no forman un triángulo válido.")

return

**# Dibujar el triángulo**

for i in range(lado1):

print('\*' \* (i+1))

for i in range(lado2):

print('\*' \* (i+1))

for i in range(lado3):

print('\*' \* (lado3-i))

**# Solicitar las medidas de los lados al usuario**

lado1 = int(input("Ingrese la medida del primer lado: "))

lado2 = int(input("Ingrese la medida del segundo lado: "))

lado3 = int(input("Ingrese la medida del tercer lado: "))

**def generar\_tri\_rectangulo():**

**"""Funcion en la que se realiza el proceso para hacer un rectangulo"""**

**# Solicitar la altura al usuario**

altura = int(input("Ingrese la altura del triángulo rectángulo: "))

print("Dibujando triángulo rectángulo...")

print()

**# Dibujar el triángulo**

for i in range(1, altura + 1):

print('\*' \* i)

**def generar\_obtusangulo():**

**"""Funcion en la que se realiza un obtusangulo"""**

**# Solicitar la altura al usuario**

altura = int(input("Ingrese la altura del triángulo acutángulo: "))

for i in range(altura):

for j in range(altura - i):

print('\*', end=' ')

print()

**def generar\_cuadrado():**

**"""Funcion en la que se realiza el proceso de un cuadrado"""**

lado = int(input("Escribe la medida de los lados del cuadrado: ")) **# Solicita al usuario la medida del cuadrado y la guarda en la variable 'A'**

for i in range(lado): **# Inicia un bucle 'for' que itera desde 0 hasta 'A - 1'**

print(" \* " \* lado)  **# Imprime 'A' veces el carácter "\*", separado por espacios, formando una línea del cuadrado**

**def generar\_rombo():**

**"""Funcion en la que se realiza el proceso de un cuadrado"""**

lado = int(input("Escribe la medida de los lados del rombo"))

for i in range(lado):

for j in range(lado-i):

print(" ", end="")

for j in range(2\*i+1):

print("\*", end="")

print()

for i in range(lado-1):

for j in range(i+2):

print(" ", end="")

for j in range(2\*(lado-i-2)+1):

print("\*", end="")

print()

**##Declara la etiqueta "generar\_romboide" la cual sirve para generar un romboide**

**def generar\_romboide():**

**"""Esta funcion realiza un romboide"""**

base = int(input("Ingresa la medida de la base: "))

altura = int(input("Ingresa la medida de la altura: "))

for i in range(altura):

for j in range(base):

if i == 0 or i == altura - 1 or j == 0 or j == base - 1:

print("\*", end="")

else:

print(" ", end="")

print()

**##Declara la etiqueta "generar\_trapecio" la cual sirve para generar un romboide**

**def generar\_trapecio():**

**"""Esta funcion realiza un trapecio"""**

altura = int(input("Ingresa la medida de la altura: "))

base\_mayor = int(input("Ingresa la medida de la base mayor: "))

base\_menor = int(input("Ingresa la medida de la base menor: "))

for i in range(altura):

for j in range(base\_mayor + base\_menor):

if i == altura - 1 or j < base\_mayor - i or j >= base\_mayor + i + 1:

print(" ", end="")

else:

print("\*", end="")

print()

**##Declara la etiqueta "generar\_pentagono" la cual sirve para generar un pentagono**

**def generar\_pentagono():**

**"""Esta funcion realiza un pentagono"""**

altura = int(input("Escribe la Altura: "))

bama = int(input("Escribe la base menor: "))

espacios=0

for i in range (altura):

espacios= altura - i

print(" "espacios+" "\*(i+1))

for i in range (altura -3):

espacios = espacios + 1

print(" "espacios+' '\*bama)

bama=bama-1

**##Declara la etiqueta "generar\_hexagono" la cual sirve para generar un hexagono**

**def generar\_hexagono():**

**"""Esta funcion realiza un hexagono"""**

lado = int(input("Ingresa la medida del lado "))

espacios = lado - 1 **# Inicializa la variable 'espacios' con el valor del lado menos 1**

# Bucle para imprimir las líneas superiores del triángulo

for i in range(lado, 3 \* lado, 2):

print(" " \* espacios + " \*" \* i) # Imprime los espacios seguidos de los asteriscos

espacios -= 1  **# Reduce el número de espacios en cada iteración**

espacios = 1 **# Restablece el valor de 'espacios' a 1**

for i in range(3 \* lado - 4, lado - 2, -2):

print(" " \* espacios + " \*" \* i) **# Imprime los espacios seguidos de los asteriscos**

espacios += 1 **# Aumenta el número de espacios en cada iteración**

**##Declara la etiqueta "generar\_heptagono" la cual sirve para generar un heptagono**

**def generar\_heptagono():**

**"""Esta funcion realiza un heptagono"""**

lado = int(input("Ingresa la medida de los lados: "))

apotema = lado / (2 \* math.tan(math.pi / 7))

for i in range(7):

angulo = 2 \* math.pi \* i / 7

x = lado \* math.cos(angulo)

y = lado \* math.sin(angulo)

print(f"({x}, {y})")

**##Declara la etiqueta "generar\_octagono" la cual sirve para generar un octagono**

**def generar\_octagono():**

**"""Esta funcion realiza un octagono"""**

lado = int(input("Ingresa la media del lado: "))

for i in range(8):

angulo = 2 \* math.pi \* i / 8

x = lado \* math.cos(angulo)

y = lado \* math.sin(angulo)

print(f"({x}, {y})")

**##Declara la etiqueta "generar\_circulo" la cual sirve para generar un circulo**

**def generar\_circulo():**

**"""Esta funcion realiza un circulo"""**

radio = int(input("Ingresa el radio del círculo: "))

for y in range(-radio, radio + 1):

for x in range(-radio, radio + 1):

distancia = math.sqrt(x\*2 + y\*2)

if distancia <= radio + 0.5:

print("\*", end=" ")

else:

print(" ", end=" ")

print()

**# Solicitar al usuario el radio del círculo**

**##Declara la etiqueta "generar\_ovalo" la cual sirve para generar un ovalo**

**def generar\_ovalo():**

**"""Esta funcion realiza un ovalo"""**

ancho = int(input("Ingresa el ancho: "))

alto = int(input("Ingresa el alto: "))

centro\_x = ancho / 2

centro\_y = alto / 2

radio\_x = ancho / 2

radio\_y = alto / 2

for y in range(alto):

for x in range(ancho):

if ((x - centro\_x) / radio\_x) \* 2 + ((y - centro\_y) / radio\_y) \* 2 <= 1:

print("\*", end="")

else:

print(" ", end="")

print()

**##Declara la etiqueta "generar\_ovoide" la cual sirve para generar un ovoide**

**def generar\_ovoide():**

**"""Esta funcion realiza un ovoide"""**

x\_radius = int(input("Ingresa la media de radio x: "))

y\_radius = int(input("Ingresa la media de radio y: "))

x\_radius = abs(x\_radius)

y\_radius = abs(y\_radius)

for y in range(-y\_radius, y\_radius + 1):

for x in range(-x\_radius, x\_radius + 1):

if ((x / x\_radius) \* 2 + (y / y\_radius) \* 2) <= 1:

print("\*", end="")

else:

print(" ", end="")

print()

**##Declara la etiqueta "menu\_triangulos" la cual sirve para mostrar el menu de los triangulos donde da a elegir que tipo de triangulo necesitas**

**def menu\_triangulos():**

**"""Esta funcion imprime un menu"""**

while True:

print(" Generar figuras ")

print("-------------------------------------")

print("| Selección de Triángulo |")

print("|-----------------------------------|")

print("| 1-Equiltaero |")

print("| 2-Escaleno |")

print("| 3-Isoceles |")

print("| 4-Triangulo Rectángulo |")

print("| 5-Obtusángulo |")

print("-------------------------------------")

seleccion()

if opc\_selected == 1:

generar\_equilatero()

elif opc\_selected == 2:

generar\_escaleno()

elif opc\_selected == 3:

generar\_isoceles()

elif opc\_selected == 4:

generar\_tri\_rectangulo()

elif opc\_selected == 5:

generar\_obtusangulo()

else:

limpiar\_pantalla()

opcion\_incorrecta()

time.sleep(1)

limpiar\_pantalla()

**##Declara la etiqueta "menu\_cuadrilateros" la cual sirve para mostrar el menu de los cuadrilateros donde da a elegir que tipo de cuadrilatero necesitas**

**def menu\_cuadrilateros():**

**"""Esta funcion realiza diferentes tipos de cuadilateros"""**

while True:

print(" Generar figuras ")

print("-------------------------------------")

print("| Selección de Cuadrílatero |")

print("|-----------------------------------|")

print("| 1-Cuadrado |")

print("| 2-Rombo |")

print("| 3-Rombooide |")

print("| 4-Trapecio |")

print("-------------------------------------")

seleccion()

if opc\_selected == 1:

generar\_cuadrado()

elif opc\_selected == 2:

generar\_rombo()

elif opc\_selected == 3:

generar\_romboide()

elif opc\_selected == 4:

generar\_trapecio()

else:

limpiar\_pantalla()

opcion\_incorrecta()

time.sleep(1)

limpiar\_pantalla()

**##Declara la etiqueta "menu\_poligonos" la cual sirve para mostrar el menu de los poligonos donde da a elegir que tipo de poligono necesitas**

**def menu\_poligonos():**

**"""Esta funcion realiza diferentes tipos de poligonos"""**

while True:

print(" Generar figuras ")

print("-------------------------------------")

print("| Selección de poligono |")

print("|-----------------------------------|")

print("| 1-Pentagono |")

print("| 2-Hexagono |")

print("| 3-Heptagono |")

print("| 4-Octagono |")

print("-------------------------------------")

seleccion()

if opc\_selected == 1:

generar\_pentagono()

elif opc\_selected == 2:

generar\_hexagono()

elif opc\_selected == 3:

generar\_heptagono()

elif opc\_selected == 4:

generar\_octagono()

else:

limpiar\_pantalla()

opcion\_incorrecta()

time.sleep(1)

limpiar\_pantalla()

**##Declara la etiqueta "menu\_poligonos" la cual sirve para mostrar el menu de los poligonos donde da a elegir que tipo de poligono necesitas**

**def menu\_circulo():**

**"""Esta funcion realiza diferentes tipos de circulos"""**

while True:

print(" Generar figuras ")

print("-------------------------------------")

print("| Selección de circulos |")

print("|-----------------------------------|")

print("| 1-Círculo |")

print("| 2-Ovalo |")

print("| 3-Ovoide |")

print("-------------------------------------")

seleccion()

if opc\_selected == 1:

generar\_circulo()

elif opc\_selected == 2:

generar\_ovalo()

elif opc\_selected == 3:

generar\_ovoide()

else:

limpiar\_pantalla()

opcion\_incorrecta()

time.sleep(1)

limpiar\_pantalla()

**# Llamar a la función para dibujar el triángulo**

**##Declara la etiqueta "menu\_home" la cual sirve para mostrar el menu una vez que entras a tu cuenta**

**def menu\_home():**

**"""Esta funcion imprime un menu"""**

global sesion

if sesion:

while True:

print("--------------------------------")

print(f" ¿Qué deseas hacer @{username} ? ")

print("|------------------------------|")

print("| 1.- Generar figuras |")

if role == 'admin':

print("| 2.- Menu Administrador |")

print("| 3.- Enviar Mensaje |")

print("| 4.- Ver mensajes |")

print("| 5.- Mi historial |")

print("| 6.- Cambiar contraseña |")

print("| 7.- Salir |")

print("--------------------------------")

else:

print("| 2.- Enviar Mensajes |")

print("| 3.- Ver mensajes |")

print("| 4.- Mi historial |")

print("| 5.- Cambiar contraseña |")

print("| 6.- Salir |")

print("--------------------------------")

seleccion()

if opc\_selected == 1:

limpiar\_pantalla()

menu\_figuras()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 2:

if role == 'admin':

limpiar\_pantalla()

menu\_admin()

limpiar\_pantalla()

else:

limpiar\_pantalla()

enviar\_mensaje()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 3:

if role == 'admin':

limpiar\_pantalla()

enviar\_mensaje()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

else:

limpiar\_pantalla()

ver\_mensajes()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 4:

if role == 'admin':

limpiar\_pantalla()

ver\_mensajes()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

else:

limpiar\_pantalla()

mostrar\_historial()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 5:

if role == 'admin':

limpiar\_pantalla()

mostrar\_historial()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

else:

limpiar\_pantalla()

mostrar\_historial()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 6:

if role == 'admin':

limpiar\_pantalla()

change\_password()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

else:

sesion = False

break

elif opc\_selected == 7 and role == 'admin':

sesion = False

break

else:

limpiar\_pantalla()

opcion\_incorrecta()

time.sleep(1)

limpiar\_pantalla()

**##Declara la etiqueta "registrar\_historial" la cual sirve para cada accion que se haga se registre en el historial**

**def registrar\_historial(accion, username):**

**"""Esta funcion realiza un historial del usuario"""**

timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

sql = "INSERT INTO historial (accion, fecha\_hora, usuario) VALUES (%s, %s, %s)"

values = (accion, timestamp, username)

cursor.execute(sql, values) **# type: ignore**

db.commit() **# type: ignore**

try:

inicializar\_database()

while True:

menu\_principal()

seleccion()

if opc\_selected == 1:

register()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 2:

sesion = login()

if sesion:

while sesion:

menu\_home()

limpiar\_pantalla()

elif opc\_selected == 3:

change\_password()

input("Presiona Enter para continuar...")

limpiar\_pantalla()

elif opc\_selected == 4:

limpiar\_pantalla()

print("SALIENDO...")

time.sleep(1)

limpiar\_pantalla()

break

else:

time.sleep(1)

except Exception as ex:

print(ex)

finally:

cerrar\_conex\_db()