PYTHON

Aprende Python con Zamy

Tabla de contenido

1. **Presentación** 4
2. **Sobre Python** 4
3. **Instalación de Python** 5
4. **Instalación del editor de código Visual Studio Code** 7
5. **Uso de las comillas** 9
6. **Uso de los comentarios** 10
7. **Como mostrar un texto por pantalla** 10
8. **Tipo de Datos** 11
9. **Operadores Matemáticos** 12
10. **Las variables** 14
11. **Ingreso de texto / Ingreso por teclado** 16
12. **Decisiones y validación de errores** 16

CONDICIONALES 16

MANEJO DE EXCEPCIONES 18

13. **Listas** 19
14. **Tuplas** 23
15. **Diccionarios** 25
16. **Bucles** 26
17. **Funciones** 27
18. **Uso de algunas librerías** 28
19. **Escritura de documentos** 35
20. **Importación de documentos** 37
21. **Programación Orientada a Objetos** 39
22. **Conexión a base de datos** 42
23. **Interfaz grafica** 44
24. **Empaquetado** 50

Ejercicios 52

Realiza Lista de 20 ejercicios básicos de Python 52

1. Hola, Mundo! 52

2. Operaciones Matemáticas 52

3. Cálculo del Área de un Triángulo 52

4. Conversión de Celsius a Fahrenheit 52

5. Determinar si un Número es Par o Impar 52

6. Calculadora Simple 53

7. Contador de Palabras 53

8. Tabla de Multiplicar 53

9. Mayor de Tres Números 53

10. Factorial de un Número 53

11. Generador de Números Primos 53

12. Cálculo de la Media y la Desviación Estándar 53

13. Adivina el Número 53
14. Conversión de Decimales a Binarios 53
15. Comprobador de Palíndromos 53
16. Cifrado César 53
17. Calculadora de Interés Compuesto 54
18. Generador de Secuencia Fibonacci 54
19. Juego de Piedra, Papel y Tijeras 54
20. Cronómetro 54
Lista de 20 ejercicios de complejidad media en Python: 54
1. Calculadora de Factorial Recursivo 54
2. Generador de Contraseñas Aleatorias 54
3. Simulador de Dados 54
4. Contador de Palabras en un Archivo de Texto 54
5. Calculadora de Distancia Euclidiana 54
6. Validador de Tarjetas de Crédito 54
7. Buscador de Archivos Duplicados 55
8. Generador de Tablas de Multiplicar 55
9. Juego del Ahorcado 55
10. Simulador de Banco 55
11. Análisis de Datos de Venta 55
12. Ordenamiento de Listas 55
13. Juego de Sudoku 55
14. Calculadora de Impuestos 55
15. Convertidor de Unidades 55
16. Generador de Gráficos de Barras 55
17. Simulador de Juego de Rol (RPG) 56
18. Calculadora de Tiempo de Viaje 56
19. Detección de Anagramas 56
20. Cifrado RSA Simple 56

1. Presentación

- ¡Hola a todos! Soy Zamy y les quiero dar la bienvenida a mi pequeño curso de Python.

- En este curso, encontrarás la sencillez con la que se aprende Python, de manera efectiva y práctica.

- Aprenderás las habilidades esenciales que te ayudarán a pasar tu materia o en tus proyectos personales, este curso es apto tanto para niños como para adultos.

- Cualquier duda o consulta comunícate con Zamy al WhatsApp +56 9 86638811, solo deja tu mensaje por temas personales no estoy contestando llamadas que no aparecen en mi registro.

2. Sobre Python

Historia de Python:

Python fue creado a finales de los 80 por Guido van Rossum en los Países Bajos. Su desarrollo se inició en diciembre de 1989 y la primera versión pública, Python 0.9.0, fue lanzada en febrero de 1991. Guido diseñó Python con un enfoque en la legibilidad del código y la simplicidad, lo que lo hace ideal para programadores principiantes y experimentados.

¿Qué es Python?:

Python es un lenguaje de programación de alto nivel, interpretado y multipropósito. Su filosofía se centra en la legibilidad y la claridad del código, lo que lo hace muy expresivo y fácil de aprender. Python es conocido por su sintaxis clara y su comunidad activa de desarrolladores que han creado una amplia gama de bibliotecas y módulos para diversas aplicaciones.

Usos Comunes de Python:

Python se utiliza en una variedad de aplicaciones, incluyendo:

- Desarrollo web con frameworks como Django y Flask.

- Ciencia de datos y análisis de datos con librerías como NumPy, Pandas y Matplotlib.

- Automatización de tareas y scripting.

- Desarrollo de aplicaciones de escritorio con herramientas como PyQt o Tkinter.

- Inteligencia artificial y aprendizaje automático con TensorFlow y PyTorch.

- Ciberseguridad, automatización de pruebas y análisis forense.

Implementación en Ciberseguridad:

Python se ha convertido en una herramienta esencial en ciberseguridad. Su simplicidad y versatilidad hacen que sea una elección popular para:

- Desarrollo de herramientas de seguridad y exploits.

- Análisis de vulnerabilidades y pruebas de penetración.

- Automatización de tareas de seguridad, como escaneo de puertos o análisis de logs.

- Desarrollo de scripts para detectar y responder a amenazas en tiempo real.

Conclusión Personal:

En resumen, Python es un lenguaje de programación versátil y poderoso que se destaca por su facilidad de uso y legibilidad. Su amplio alcance lo hace ideal tanto para principiantes como para expertos en programación. En el ámbito de la ciberseguridad, su capacidad de automatización y su comunidad activa de desarrolladores lo convierten en una herramienta valiosa para proteger sistemas y datos en un mundo cada vez más digital y conectado. El aprendizaje de Python abre un mundo de oportunidades en la programación y la seguridad informática. ¡Así que no dudes en comenzar tu viaje de aprendizaje en Python hoy mismo!

3. Instalación de Python

Windows:

1. Visita el sitio web oficial de Python en [python.org](https://www.python.org/).

2. Ve a la sección "Downloads" (Descargas) y haz clic en "Python x.x.x" (donde "x.x.x" representa la versión más reciente de Python).

3. Desplázate hacia abajo en la página y elige la versión de Python adecuada para tu sistema:

- Si tienes un sistema de 64 bits, elige la versión de 64 bits.

- Si tienes un sistema de 32 bits, elige la versión de 32 bits.

4. Haz clic en el archivo de instalación descargado para ejecutarlo.

5. En la primera ventana de instalación, asegúrate de marcar la casilla "Add Python x.x to PATH" (Agregar Python x.x al PATH) antes de hacer clic en "Install Now" (Instalar ahora).

6. El instalador de Python copiará los archivos necesarios y configurará Python en tu sistema.

7. Una vez completada la instalación, puedes verificar si Python se instaló correctamente abriendo el símbolo del sistema o PowerShell y escribiendo `python --version`. Deberías ver la versión de Python que acabas de instalar.

MacOS:

1. Abre una terminal en tu macOS. Puedes encontrar la Terminal en la carpeta "Utilidades" dentro de la carpeta "Aplicaciones" o usar Spotlight para buscarla.

2. Verifica si Python ya está instalado en tu sistema escribiendo `python3 --version`. Algunas versiones de macOS ya incluyen Python 3.

3. Si Python 3 no está instalado, puedes instalarlo utilizando Homebrew (un administrador de paquetes para macOS):

- Instala Homebrew si aún no lo tienes siguiendo las instrucciones en su sitio web.

- Después de instalar Homebrew, abre la terminal y ejecuta el siguiente comando para instalar Python 3:

```
brew install python
```

4. Verifica que Python 3 se haya instalado correctamente ejecutando `python3 --version`.

Linux (Ubuntu/Debian):

1. Abre una terminal en tu sistema Linux.

2. Verifica si Python ya está instalado en tu sistema escribiendo `python3 --version`. Muchas distribuciones de Linux ya incluyen Python 3.

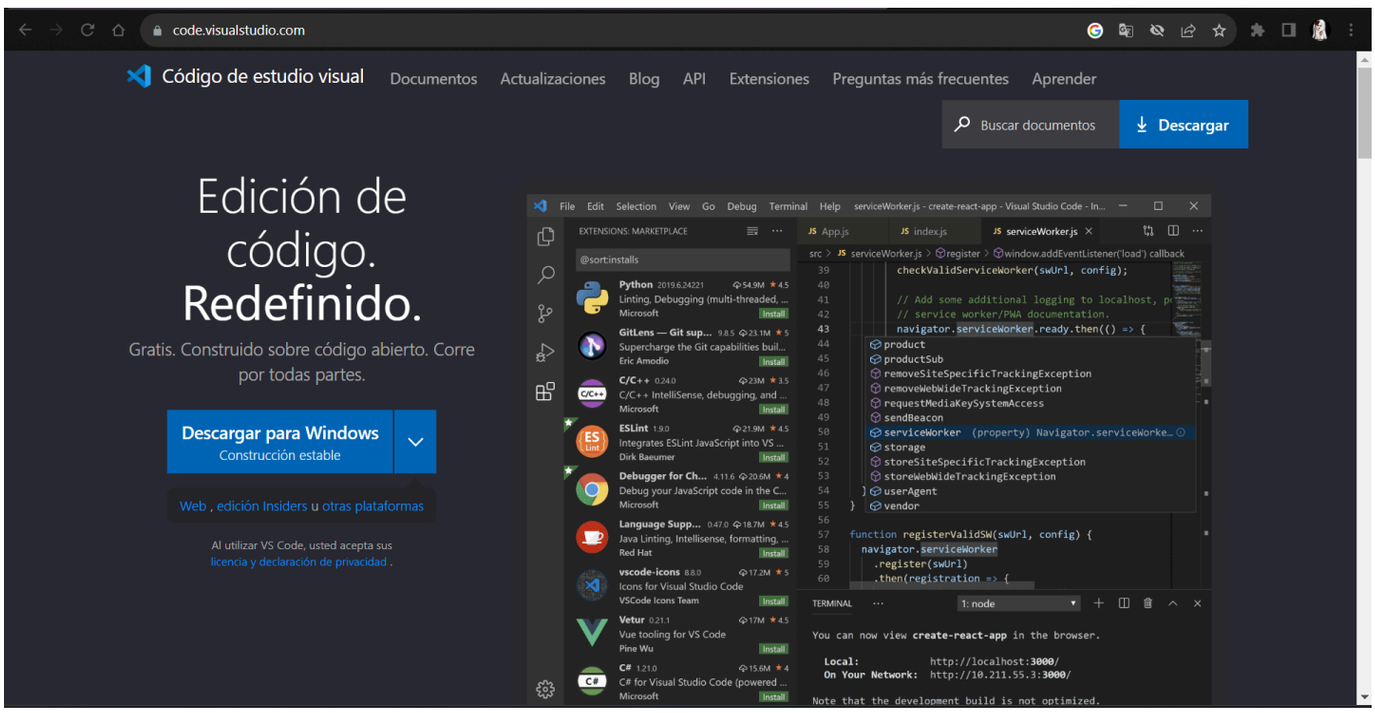
3. Si Python 3 no está instalado, puedes instalarlo ejecutando los siguientes comandos en la terminal:

```
sudo apt update
sudo apt install python3
```

4. Verifica que Python 3 se haya instalado correctamente ejecutando `python3 --version`.

Estas instrucciones te ayudarán a instalar Python en tu sistema, ya sea Windows, macOS o Linux. Una vez que Python esté instalado, estarás listo para empezar a escribir y ejecutar programas en Python.

4. Instalación del editor de código Visual Studio Code



Para Windows:

1. Visita el sitio web oficial de Visual Studio Code en https://code.visualstudio.com/.

2. Haz clic en el botón "Download for Windows" (Descargar para Windows) para descargar el instalador.

3. Una vez que se complete la descarga, haz doble clic en el archivo de instalación descargado (`VSCodeSetup-x64-x.xxx.xxx.exe`, donde "x.xxx.xxx" representa la versión más reciente).

4. Aparecerá una ventana de instalación. Sigue las instrucciones en pantalla, acepta los términos de la licencia y elige las opciones de instalación según tus preferencias.

5. Haz clic en "Install" (Instalar) para comenzar la instalación.

6. Después de la instalación, puedes abrir Visual Studio Code desde el menú Inicio o buscándolo en la barra de búsqueda.

Para MacOS:

1. Visita el sitio web oficial de Visual Studio Code en https://code.visualstudio.com/.

2. Haz clic en el botón "Download for Mac" (Descargar para Mac) para descargar el archivo .dmg.

3. Abre el archivo .dmg descargado.

4. Arrastra el icono de Visual Studio Code a la carpeta de Aplicaciones para instalarlo en tu sistema.

5. Una vez que se complete la instalación, puedes encontrar Visual Studio Code en tu carpeta de Aplicaciones y abrirlo desde allí.

Para Linux (Ubuntu/Debian):

1. Abre una terminal en tu sistema Linux.

2. Ejecuta los siguientes comandos uno por uno para agregar el repositorio oficial de Visual Studio Code y la clave GPG:

```
sudo apt update
sudo apt install software-properties-common apt-transport-https wget
wget -q https://packages.microsoft.com/keys/microsoft.asc -O- | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://packages.microsoft.com/repos/vscode stable main"
```

3. Después de agregar el repositorio, actualiza la lista de paquetes e instala Visual Studio Code:

sudo apt update

sudo apt install code

4. Una vez que se complete la instalación, puedes abrir Visual Studio Code desde el menú de aplicaciones o ejecutando `code` en la terminal.

Ahora que has instalado Visual Studio Code, puedes comenzar a usarlo como tu editor de código para programar en Python y muchas otras tecnologías.

5. Uso de las comillas

La elección del uso entre comillas generalmente se basa en preferencias personales, o en la necesidad de incluir comillas del tipo opuesto dentro de la cadena.

Forma de implementación entre comillas.

uso_comillas.py

```
'Comillas simples'
"Comillas dobles"

'Comillas simples  con "comillas dobles" '
"Comillas dobles con 'comillas  simples' "

'''Comillas  triple simples con "comillas dobles" '''
"""Comillas triple dobles con 'comillas  simples' """

```

Las comillas simples y las comillas dobles son cadenas de texto de una línea, las comillas triples son multilínea y puedes acomodar el texto como e parezca en cambio las simples y dobles generan un error al interrumpir la línea por ejemplo multilínea seria así.

```
'''Comillas  triple
        simples con
                "comillas dobles" '''


"""Comillas triple
        dobles con
    'comillas  simples' """

```

Las comillas simples y las comillas dobles al interrumpir su línea generan un error, no solo se puede ver el error en la línea 4 y 5 que fue donde se interrumpio la línea también se refleja el error de la línea de mas abajo en la línea 8.

```
'Comillas simples
        con "comillas dobles" '

"Comillas dobles con 'comillas  simples' "
```

6. Uso de los comentarios

Los comentarios se utilizan para proporcionar explicaciones o anotaciones dentro del código fuente, así como también comentar el mismo código para realizar pruebas

```

# Este es un comentario
```

7. Como mostrar un texto por pantalla

Mostrar una salida de texto por pantalla o salida de datos, implementamos print().

Se utiliza para mostrar información en la salida estándar, que generalmente es la consola o la terminal.

Puedes utilizar print() de diferente manera, para mostrar diferentes tipos de datos y mensajes.

Formas de mostrar texto con print().

```
31   print("comillas dobles" )
32   print('comillas simples' )
33   print("comillas dobles con 'comillas simples' " )
34   print('comillas simples con "comillas dobles" ' )
35
36   print(''' 3 comillas simples ''')
37   print(''' 3 comillas simples con "comillas dobles"''')
38
39   print(""" 3 comillas dobles """)
40   print(""" 3 comillas dobles con 'comillas simples' """)
41
42   # cuando van comillas diferentes adentro de otras comillas
43   # estas se muestran pero uando estas son todas iguales
44   # generan un error
```

Este sería el resultado seria

```
comillas dobles
comillas simples
comillas dobles con 'comillas simples'
comillas simples con "comillas dobles"
 3 comillas simples
 3 comillas simples con "comillas dobles"
 3 comillas dobles
 3 comillas dobles con 'comillas simples'
PS C:\Users\zamyk\Desktop\B\i>
```

A continuación, podemos ver algunos errores…

```
31    print("comillas
32          dobles" )
33
34    print('comillas
35          simples' )
36
37    print("comillas dobles con
38          'comillas simples' " )
39
40    print('comillas simples con
41          "comillas
42          dobles" ' )
43
44    print(''' 3 comillas simples ''')
45
46    print(''' 3 comillas
47                  simples con
48          "comillas
49                  dobles''')
50
```

Aquí se puede ver que cuando se parte el texto a la mitad en comillas simples y dobles me genera error pero no en donde están las comillas triples, esto es porque el print() solo muestra dicho contenido pero no interfiere con la función que implementa cada instrucción como por ejemplo la forma correcta e incorrecta de implementar las comillas.

Las comillas tripes se mantienen sin errores ya que estas no tienen problema alguno para que se le pueda dividir a modo de multilíneas.

El resultado del error seria el siguiente.

```
.exe c:/Users/zamyk/Desktop/B/i/tipo_dato.py
  File "c:\Users\zamyk\Desktop\B\i\tipo_dato.py", line 31
    print("comillas
          ^
SyntaxError: unterminated string literal (detected at line 31)
PS C:\Users\zamyk\Desktop\B\i>
```

Este me indica también en que línea comienza el error y me indica el tipo de error que tengo.

También podemos ver un error cuando ponemos un texto sin un entrecomillado y sin un valor(variables las cuales veremos más abajo).

```
48
49    print(cadena de texto sin comillas que genera un error )
50
```

El error generado se vería algo así.

```
.exe c:/Users/zamyk/Desktop/B/i/tipo_dato.py
  File "c:\Users\zamyk\Desktop\B\i\tipo_dato.py", line 49
    print(cadena de texto sin comillas que genera un error )
          ^^^^^^^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
PS C:\Users\zamyk\Desktop\B\i>
```

8. Tipo de Datos

Numericos:

Enteros (int) : Los números enteros son son valores sin parte desimal.(1, 2, 3, 4…)

Coma Flotante (float): Los números Coma Flotantes o también conocidos como decimal son aquellos números que los divide un punto.(1.5, 3.4, 16.50…)

Boleanos (Verdadero/ Falso): Los datos booleanos son un tipo de dato fundamental en programación que se utilizan para representar dos valores: Verdadero (True) y Falso (False).

1. Valores Booleanos:

- Verdadero (True): Representa un valor afirmativo o cierto.

- Falso (False): Representa un valor negativo o falso.

2. Uso de Datos Booleanos:

- Los datos booleanos se utilizan principalmente en expresiones condicionales para tomar decisiones en el código.

- Se usan en comparaciones, como igualdades (==), diferente que (!=), mayor que (>), menor que (<), mayor o igual que (>=), y menor o igual que (<=).

3. Valores Resultantes:

- Las expresiones condicionales devuelven un valor booleano (True o False) como resultado de la evaluación.

4. Operadores Lógicos:

- Se pueden combinar expresiones booleanas utilizando operadores lógicos como "and" (y), "or" (o) y "not" (no) para crear condiciones más complejas.

En resumen, los datos booleanos son esenciales para la toma de decisiones en programación, permitiéndonos controlar el flujo de ejecución del código según condiciones que pueden ser Verdaderas o Falsas. Estos valores son fundamentales en lenguajes de programación para implementar lógica y control de flujo en programas.

9. Operadores Matemáticos

Operadores matemáticos.

+ ⇔ Suma

- ⇔ Resta

*** ⇔ Multiplicación**

/ ⇔ División Decimal

% ⇔ Modulo, entrega el resto de un numero decimal

**** ⇔ Exponente**

// ⇔ División Entera

Operadores Comparativos:

== ⇔ Igual que

!= ⇔ Diferente que

> ⇔ **Mayor que**

>= ⇔ **Mayor o igual que**

< ⇔ **Menor que**

<= ⇔ **Menor o igual que**

Operadores Lógicos:

and ⇔ **Operador lógico "Y" (ambas condiciones deben ser verdaderas)**

or ⇔ **Operador lógico "O" (al menos una condición debe ser verdadera)**

not ⇔ **Operador lógico "NO" (invierte el valor de la condición)**

Operadores de Asignación:

= ⇔ **Igual (asigna un valor a una variable)**

+= ⇔ **Incremento (suma y asigna)**

-= ⇔ **Decremento (resta y asigna)**

***=** ⇔ **Multiplicación y asignación**

/= ⇔ **División y asignación**

%= ⇔ **Módulo y asignación**

****=** ⇔ **Exponente y asignación**

//= ⇔ **División entera y asignación**

Python sigue las reglas matemáticas estándar en la mayoría de las operaciones numéricas. A continuación, se presentan algunas de las reglas matemáticas clave que Python sigue:

1. Jerarquía de Operaciones: Python sigue la jerarquía de operaciones matemáticas estándar, que dicta el orden en el que se realizan las operaciones. Esto significa que primero se realizan las operaciones dentro de paréntesis, luego las multiplicaciones y divisiones, y finalmente las sumas y restas.

2. Operaciones Aritméticas: Python realiza operaciones aritméticas como suma (`+`), resta (`-`), multiplicación (`*`), división (`/`), módulo (`%`), exponente (`**`) y división entera (`//`) de acuerdo con las reglas matemáticas convencionales.

3. Paréntesis: Puedes utilizar paréntesis para forzar el orden de las operaciones. Las operaciones dentro de paréntesis se evalúan primero.

4. Asociatividad: Python sigue las reglas de asociatividad de las operaciones. Por ejemplo, en operaciones de suma y multiplicación, se realizan de izquierda a derecha.

5. Prioridad de Operadores: Los operadores tienen diferentes niveles de prioridad. Por ejemplo, la multiplicación y la división tienen prioridad sobre la suma y la resta.

6. Manejo de Enteros y Flotantes: Python realiza conversiones automáticas entre enteros y flotantes según sea necesario en las operaciones. Por ejemplo, si divides dos enteros y el resultado es un número no entero, Python devuelve un valor de tipo flotante.

7. Errores en la División: La división por cero (`0`) en Python genera una excepción (`ZeroDivisionError`), ya que no está permitida en matemáticas.

8. Precisión Numérica: Los números en coma flotante en Python tienen una precisión limitada debido a la representación binaria. Esto puede llevar a pequeños errores de redondeo en cálculos muy precisos.

9. Operadores de Comparación: Python utiliza operadores de comparación como `==` (igual), `!=` (no igual), `<` (menor que), `>` (mayor que), `<=` (menor o igual que) y `>=` (mayor o igual que) para comparar valores numéricos y devolver resultados booleanos (`True` o `False`).

10. Operaciones con Números Complejos: Python admite operaciones matemáticas con números complejos utilizando la notación `j` para la parte imaginaria.

Es importante recordar que Python sigue las reglas matemáticas estándar en la mayoría de las operaciones, lo que lo hace consistente con las expectativas matemáticas. Sin embargo, siempre es recomendable verificar y validar los resultados de tus cálculos, especialmente en casos críticos o donde la precisión es crucial.

10. Las variables

VARIABLES

- Son cajas que nos permiten almacenar variedad de valores que se pueden mantener o cambiar según se requiera, como por ejemplo números o cadenas de caracteres, se comienza con minúsculas o mayúsculas y que contiene únicamente letras, cifras y adicionalmente si es necesario el carácter subrayado es decir cuando añaden un guion bajo para remplazar espacios.

De esta manera creamos una variable y le asignamos un valor numérico entero

```
244    nombre = 5 # int numero entero
```

Ahora conoceremos type() que nos indica que tipo de datos o dato almacena una variable

```
244    nombre = 5 # int numero entero
245    print(type(nombre))
246    print("el valor es:", nombre, "y su typo de dato es", type(nombre))
247
248    nombre1 = 5.6 # float numero desimal
249    print(type(nombre1))
250    print("el valor es:", nombre1, "y su typo de dato es", type(nombre1))
251
252    nombre2 = "texto mundo frio" # string texto o cadena de texto
253    print(type(nombre2))
254    print("el valor es:", nombre2, "y su typo de dato es", type(nombre2))
255
```

```
258    # como mostrar las variables entre las comillas con un print
259    # con un valor numerico
260    nombre_variable = 6
261
262    print(''' mostrar ''',  nombre_variable,  ''' de esta manera''')
263    print(""" mostrar """,  nombre_variable,  """ de esta manera""")
264    print(' mostrar ',  nombre_variable,  ' de esta manera ')
265    print(" mostrar ",  nombre_variable,  " de esta manera")
266
```

```
268    # con un valor de texto
269    nombre_variable1 = "AbC"
270
271
272    print(''' mostrar ''', nombre_variable1, ''' de esta manera''')
273    print(""" mostrar """, nombre_variable1, """ de esta manera""")
274    print(' mostrar ', nombre_variable1, ' de esta manera ')
275    print(" mostrar ", nombre_variable1, " de esta manera")
276
```

Luego conocemos la función format()

La función format() en Python se usa para crear cadenas de texto con valores variables. Imagina que quieres imprimir un mensaje que incluye tu nombre y tu edad, pero no sabes qué valores tendrán. Puedes usar format() para hacerlo de la siguiente manera:

```
4   nombre = "Juan"
5   edad = 30
6
7   # Usando format() para crear una cadena con valores variables
8   mensaje = "Hola, mi nombre es {} y tengo {} años.".format(nombre, edad)
9
10  # Imprimimos el resultado
11  print(mensaje)
12
```

11. Ingreso de texto / Ingreso por teclado

Ingresar datos por teclado en Python se puede lograr utilizando la función `input()`. Esta función permite al usuario ingresar texto desde el teclado, que luego puede ser almacenado en una variable para su posterior procesamiento.

Con esta función ya logramos hacer que el usuario interactúe con nuestro sistema…

1. **Entrada de texto simple:**

```
5   nombre = input("Por favor, ingrese su nombre: ")
6   print("Hola, " + nombre + "!")
```

La respuesta seria

```
Por favor, ingrese su nombre: zamy
Hola, zamy!
PS C:\Users\zamyk\Desktop\B\i>
```

12. Decisiones y validación de errores

CONDICIONALES

Las condicionales son estructuras de control que permiten tomar decisiones en función de si una expresión es verdadera o falsa. Las principales condicionales en Python son:

1. if: Se utiliza para ejecutar un bloque de código si una expresión es verdadera.

```
if condicion:
    # Bloque de código si la condición es verdadera
```

2. else: Se utiliza junto con `if` para ejecutar un bloque de código cuando la condición no es verdadera.

```
if condicion:
    # Bloque de código si la condición es verdadera
else:
    # Bloque de código si la condición no es verdadera
```

3. elif: Se utiliza para evaluar múltiples condiciones en secuencia y ejecutar el bloque de código correspondiente al primer `elif` o al `else` si ninguna de las condiciones anteriores es verdadera.

```
if condicion1:
    # Bloque de código si la condición1 es verdadera
elif condicion2:
    # Bloque de código si la condición2 es verdadera
else:
    # Bloque de código si ninguna de las condiciones es verdadera
```

4. Operadores de comparación: Se utilizan en las condiciones para comparar valores. Algunos operadores comunes son

== (igual),

!= (no igual),

< (menor que),

> (mayor que),

<= (menor o igual que) y

>= (mayor o igual que).

```
if numero > 10:
    # Bloque de código si numero es mayor que 10
```

Estas estructuras condicionales permiten controlar el flujo de ejecución de un programa en función de las condiciones que se cumplan, lo que hace que Python sea muy versátil para tomar decisiones en la programación.

MANEJO DE EXCEPCIONES

En Python, el manejo de excepciones es una técnica que permite controlar y gestionar situaciones excepcionales que pueden ocurrir durante la ejecución de un programa. Las excepciones son errores que pueden interrumpir el flujo normal del programa.

Los bloques principales para manejar excepciones son:

1. try: Se utiliza para encerrar un bloque de código que puede generar una excepción.

```
try:
    # Bloque de código que puede generar una excepción
```

2. except: Se utiliza junto con `try` para manejar una excepción específica que puede ocurrir en el bloque `try`. Si se produce la excepción, se ejecuta el bloque `except`.

```
except TipoDeExcepcion as e:
    # Bloque de código para manejar la excepción
```

3. finally: Este bloque es opcional y se ejecutará siempre, ya sea que se produzca una excepción o no. Se utiliza para realizar tareas de limpieza o liberación de recursos.

```
finally:
    # Bloque de código que se ejecutará siempre
```

El manejo de excepciones es útil para prevenir que un programa se bloquee incluso cuando ocurren errores, y para tomar medidas adecuadas cuando se producen excepciones.

Ejemplo:

```
try:
    numero = int(input("Ingrese un número entero: "))
    resultado = 10 / numero
    print("El resultado es:", resultado)

except ZeroDivisionError as e:
    print("Error: No se puede dividir entre cero.")

except ValueError as e:
    print("Error: Ingrese un número entero válido.")

finally:
    print("Siempre se ejecutará este bloque, ya sea que haya excepciones o no.")
```

13. Listas

Las listas son una estructura de datos que te permite almacenar y organizar múltiples elementos en una sola variable. Puedes utilizar estas operaciones en listas para realizar diversas tareas:

1. Acceso a elementos: Las listas están indexadas, lo que significa que cada elemento de la lista tiene una posición única. Puedes acceder a elementos específicos utilizando índices

positivos o negativos. Los índices positivos comienzan desde 0 y avanzan hacia la derecha, mientras que los índices negativos comienzan desde -1 y avanzan hacia la izquierda.

2. Añadir elementos: Puedes agregar elementos a una lista de varias formas. Usar `.append()` agrega un elemento al final de la lista. `mi_lista.append('sandra')` agrega 'sandra' al final de la lista `mi_lista`. También puedes insertar elementos en posiciones específicas con `.insert()`, como en `mi_lista.insert(2, 'sandy')`.

3. Extender la lista: Si deseas agregar múltiples elementos a una lista al final, puedes usar `.extend()`. Por ejemplo, `mi_lista.extend(['lalita', 'carlitos', 'pupis'])` agrega estos elementos al final de la lista.

4. Insertar una lista dentro de otra lista: Puedes insertar una lista dentro de otra lista. Por ejemplo, `mi_lista.insert(0, ['lalita', 'carlitos', 'pupis'])` inserta la lista `[lalita', 'carlitos', 'pupis']` en la posición 0 de `mi_lista`.

5. Encontrar el índice de un elemento: Si deseas saber en qué posición se encuentra un elemento en la lista, puedes usar `.index()`. `mi_lista.index('pupis')` devuelve el índice de 'pupis' en la lista.

6. Comprobar la existencia de un elemento: Puedes verificar si un elemento está presente en una lista con la expresión `elemento in lista`. Por ejemplo, `'maria' in mi_lista` devolverá `True` si 'maria' está en `mi_lista`.

7. Eliminar elementos: Puedes eliminar elementos de una lista utilizando `.remove()`, que elimina el elemento especificado. `mi_lista.remove('camil')` eliminará 'camil' de la lista. También puedes usar `.pop()` para eliminar el último elemento de la lista, como en `mi_lista.pop()`.

8. Unir listas: Puedes combinar dos o más listas concatenándolas con el operador `+`. `listaA + listaB` crea una nueva lista que contiene todos los elementos de `listaA` seguidos de todos los elementos de `listaB`.

9. Crear una lista repetida: Puedes crear una lista que contenga elementos repetidos multiplicando la lista por un número entero. `['lalita', 'carlitos', 'pupis'] * 3` crea una nueva lista que contiene tres copias de la lista original.

Ejemplos:

listas.py > ...

```python
mi_lista = ['maria', 'pepe', 'camil' , 'gussi']
#               0       1       2           3   indice positivo
#              -4      -3      -2          -1   indice negativo



# PARA DAR LECTURA

# muestra la lista completa
print(mi_lista[:])
print(mi_lista)


# muestra la lista apartir del indice 1 y exculle del
# elemento 3 y los que le sigen
print(mi_lista[1:2])

# de esta manera le indico que tiene que mostrar todo
# apartir del elemento 2 y excluir lo demas
print(mi_lista[2:])
```

```python

# muestra solo el elemento 2 de la lista
print(mi_lista[2])

# con numeros negativos podemos acceder a la lista desde
# atras asta adelante
print(mi_lista[-1])

# muestra el primer elemento de una lista indicandose este
# como un inicio en cero
print(mi_lista[0])

# de esta manera se accese al primer elemento de la lista
# haciendo un recorrido desde el final al inicio
print(mi_lista[-3])

```

```python
# el \n nos proporciona un salto de linea y se implementa segun
# los espacios que requieras \n\n y el mismo print('') sin contenido
# genera tambien un espacio de ser requerido cuando no se pueda
# implementar \n
print('\n########################\n')
mi_lista = ['maria', 'pepe', 'camil' , 'gussi']
#               0       1       2           3   indice positivo
#              -4      -3      -2          -1   indice negativo


# PARA AÑADIR ELEMENTOS A CONTINUACION DEL ULTIMO DE
# LA LISTA ESTE TOM EL ULTIMO VLOR
mi_lista.append('sandra')

print(mi_lista[:])

print('\n########################\n')


```

listas.py > ...

```python

# AQUI SE INDICA EN QUE POSISION INSERTAR EL ELEMENTO
mi_lista.insert(2,'sandy')

print(mi_lista[:])


print('\n########################\n')

# ASI SE PUEDE EXTENDER LA LISTA CON MAS ELEMENTOS DE UNA SOLA VEZ
mi_lista.extend(['lalita', 'carlitos', 'pupis'])

print(mi_lista[:])


print('\n########################\n')


# Y DE ESTA MANERA LA PODEMOS INSERTAR EN UN LUGAR ESPECIFICO
mi_lista.insert(0,['lalita', 'carlitos', 'pupis'])
```

listas.py > ...

```
75
76  print(mi_lista[:])
77
78  print('\n#########################\n')
79
80  # ASI SE ENTREGA EL NUMERO DEL INDICE CORRESPONDIENTE AL QUE ESTE PERTENECE
81  print(mi_lista.index(('pupis')))
82
83  print('\n#########################\n')
84
85  # ASI COMPROBAMOS SI UN ELEMENTO SE ENCUENTRA O NO ADENTRO
86  # DE UNA mi_lista, SI UN ELEMENTO ESTA DEVUELVE TRU Y SI
87  # ESTE NO ESTA REGRESA FALSE
88  mi_lista = ['maria', 'pepe', 'camil' , 'gussi']
89  print('maria' in mi_lista)
90
91  print(mi_lista)
92
93
94  print('\n#########################\n')
```

listas.py > ...

```
95
96    # ELIMINAMOS UN ELEMENTO DE LA LISTA, CUANDO EL
97    # ELEMENTO ES ELIMINADO NOS MUESTRA UN MENSAJE DE NONE
98    print(mi_lista.remove(('camil')))
99    print(mi_lista)
100
101
102   print('\n############################\n')
103   # PARA ELIMINAR EL ULTIMO ELEMENTO DE UNA LISTA
104   mi_lista.pop()
105
106   print(mi_lista[:])
107
108
109   # ASI SE UNEN 2 O MAS LISTAS
110   listaA= ['lalita', 'carlitos', 'pupis']
111   listaB = ['maria', 'pepe', 'camil' , 'gussi']
112
113
```

```
114  # SE SEMAN AMBAS LISTAS => ['lalita', 'carlitos', 'pupis', 'maria', 'pepe', 'camil', 'gussi']
115  listaC= listaA + listaB
116  print (listaC )
117
118
119  # SE UNEN AMBAS LISTAS => (['lalita', 'carlitos', 'pupis'], ['maria', 'pepe', 'camil', 'gussi'])
120  listaD= listaA , listaB
121  print (listaD )
122
123  print('')
124
125
126  # MOSTRAR UNA LISTA REPETIDA ADENTRO DE UNA MISMA LISTA 2 VECES PARA ELLO SE MULTIPLICA
127  # EJ:
128  # Lista repetida:  ['lalita', 'carlitos', 'pupis', 'lalita', 'carlitos', 'pupis']
129  listaA= ['lalita', 'carlitos', 'pupis'] * 2
130
131  print ('Lista repetida: ', listaA, '\n')
132
133
```

14. Tuplas

Una tupla es algo inmutable, es algo que no se puede modificar

No se permiten:

 # append.

 # extend.

 # remove.

 # si puede comprobar si un elemento se encuentra en la tupla.

 # son mas rapidas.

 # menos espacio

 # formatean strings

 # puedan utilizarse para claves de un diccioario las listas no.

 # se puede buscar cual es el index

 # SINTAXIS DE UNA TUPLA

```python
tupl = ('elem1', 'elem2', 'elem3', 'elem4')

# BUSCAR UN ELEMENTO EN LA TUPLA
print(tupl[1])


# PASAR TUPLA A LISTA
# TUPLA
mitupla= ('julia', 13, 1, 13, 1995)

# NUEVA LISTA
lista_tupla=list(mitupla)
print(lista_tupla)

#  CONVERTIR LISTA A TUPLA
lista_a_tupla = tuple ( lista_tupla )
print(lista_a_tupla)

```

```python
# SABER SI UN ELEMENTO ESPESIFICO SE
# ENCUENTRA EN LA TUPLA
print('julia' in lista_a_tupla)

# CUENTA LOS ELEMENTOS DE UNA TUPLA COMPLETA
print("La cantidad de elementos con len:", len(lista_a_tupla))

# CUNTA LOS ELEMENTOS QUE SON IGUALES EJEMPLO 13 = 13
print("La cantidad de elementos con count:", lista_a_tupla.count(13))

```

```
47  print('')
48
49  # TUPLA UNITARIA ES UNA QUE SOLO
50  # TIENE UN UNICO ELEMENTO
51
52  # CUENTA LOS ELEMENTOS
53  unitaria = ('lenar',)
54  print('ELEMENTOS', len(unitaria))
55
56
57  # CUENTA LOS CARACTERES
58  unitaria = ('lenar')
59  print('CARACTERES', len(unitaria))
60
61
62  # DESEMPAQUETADO DE TUPLA
63  mitu= ('julia', 13, 1, 1995)
64
```

```
65  nombre, dia, mes, anio = mitu
66
67  print(nombre)
68  print(dia)
69  print(mes)
70  print(anio)
71
72
73  # EL INDEX SE IMPLEMENTA IGUAL QUE EN LAS LISTAS
74  print(mitu.index(mes))
```

15. Diccionarios

Estructura de datos que nos permite almacenar valores de diferentes tipos (numeros enteros, números decimales, cadenas de texto) la principal caracteristica de los diccionarios es que los datos se almacenan asociados a una clave de tal forma que se crea una asociacion de tipo clave:valor para cada elemento almacenado.

Los elementos almacenados no están ordenados, el orden es indiferente a la hora de almacenar información en un diccionario.

diccionario.py > ...

```python

# SINTAXIS DEL DICCIONARIO

mydiccionario={"Alemania":"Berlin",
               "Francia":"paris",
               "Reino Unido":"Londres" }

# DE LLAMA AL VALOR PARA MOSTRAR LA CLAVE
print(mydiccionario["Alemania"])
print(mydiccionario["Francia"])
print(mydiccionario["Reino Unido"])

print("\n################################\n")

# VER EL DICCIONARIO COMPLETO
print(mydiccionario)


print("\n################################\n")

```

diccionario.py > ...

```python

# AÑADIR ELEMENTOS
mydiccionario["Chile"]="Arica"
print(mydiccionario)


print("\n################################\n")

# MODIFICAR UN VALOR
mydiccionario["Chile"]="Santiago"
print(mydiccionario)


print("\n################################\n")

# ELIMINAR ELEMENTOS SE USA 'del'
del mydiccionario["Reino Unido"]
print(mydiccionario)

```

diccionario.py > ...

```python
# DICCIONARIO CON VALORES O CLAVES EN STRING

mydiccionario2={23:"Pepe",
                "Francisca":32,
                32:99 }

print(mydiccionario2)

print(mydiccionario2[23]) # sale error ver problema pendiente
print(mydiccionario2["Francisca"])
print(mydiccionario2[32])


print("\n##############################\n")

# MANEJAR O MEZCLAR DICCIONARIOS CON TUPLAS

tup1= ["España", "Francia", "Reino Unido", "Alemania"]

dic2= {tup1[0]:"Madrid", tup1[1]:"Paris" }

```

diccionario.py > ...

```python
# SE MUESTRA TODO EL CONTENIDO DE 'dic2'
print(dic2)

# ASI LLAMAMOS SOLO A UN ELEMENTO ESPECIFICO
# EN ESTE CASO UN VALOR ESPESIFICO
print(dic2["Francia"])

# PASAR MAS DE UN VALOR DE UN DICCIONARIO A TUPLA
# Diccionario dentro de otro diccionario
# con una tupla incluida

dic_dic= {23:"Jordan", "Nombre":"Michael", "Equipo":"Chicago", "anillos":{"temporadas":[1991, 1992, 1993, 1996, 1997, 1998]}}
print(dic_dic)
print(dic_dic["anillos"])

```

16. Bucles

Los bucles son estructuras de control que permiten

repetir un bloque de código múltiples veces. Los dos bucles

principales en Python son:

1. Bucle for: Se utiliza para iterar sobre una secuencia (como una lista, tupla, cadena de caracteres, etc.) o un objeto iterable.

for elemento in secuencia:

 # Bloque de código a ejecutar para cada elemento

2. Bucle while: Se utiliza para repetir un bloque de código mientras se cumple una condición.

```
while condicion:
    # Bloque de código a ejecutar mientras la condición sea verdadera
```

Dentro de los bucles, puedes utilizar sentencias como `break` para salir del bucle antes de que se cumpla la condición o `continue` para saltar la iteración actual y continuar con la siguiente.

Ejemplo de bucle for:

```
21  frutas = ["manzana", "banana", "cereza"]
22  for fruta in frutas:
23      print("Me gusta la", fruta)
```

Ejemplo de bucle while:

```
27  contador = 0
28  while contador < 5:
29      print("Contador:", contador)
30      contador += 1
```

17. Funciones

Una función es un bloque de código que realiza una tarea específica y puede ser llamado desde otras partes del programa, ósea reutilizar código que ya se implementó en otra área del archivo o en otro archivo.

```python
print('##########FORMA 2')

def em(mum1, num2):
    print(mum1 + num2)


em(1, 2)
em(5, 4)
em(6, 1)

```

```python
# FORMAS DE IMPLEMENTAR DEF
print('##########FORMA 1')

def om():
    print('hola')

om()

```

```python
print('##########FORMA 3')

def am(mum1, num2):
    resultado=mum1 + num2

    return resultado

print(am(1, 2))
print(am(5, 4))
print(am(6, 1))

```

```python
print('###########FORMA 4')

def am(mum1, num2):
    resultado=mum1 + num2

    return resultado

almacena= am(1, 2)
print(almacena)

```

```python
print('###########FORMA 5')

def am(mum1, num2):
    resultado=mum1 + num2

    return resultado

almacena= am(1, 2), am(5, 2)
print(almacena)
```

18. Uso de algunas librerías

Las librerías en el contexto de la programación, también conocidas como bibliotecas, son conjuntos de código predefinido que contienen funciones, clases y recursos adicionales para realizar tareas específicas en un lenguaje de programación.

En términos generales:

- Las librerías son como herramientas o cajas de herramientas que los programadores utilizan para realizar tareas comunes de manera más eficiente. Estas tareas pueden incluir cálculos matemáticos, manipulación de archivos, comunicación en red, procesamiento de datos y muchas otras.

- Cada librería se enfoca en un conjunto específico de tareas o problemas, y proporciona funciones y objetos que simplifican la implementación de esas tareas.

- Las librerías permiten a los programadores reutilizar el código que otras personas han escrito, lo que acelera el desarrollo de aplicaciones y reduce la necesidad de crear todo desde cero.

- En Python, muchas librerías están disponibles en la librería estándar, lo que significa que se incluyen automáticamente con la instalación de Python y se pueden utilizar sin necesidad de una instalación adicional.

Por lo tanto, cuando se hace referencia a librerías en el contexto de programación, se está hablando de estos conjuntos de código que extienden las capacidades de Python y facilitan la implementación de diversas tareas.

Como instalar librerías de Python:

- Abre la línea de comandos (CMD) en tu computadora, o anda a la parte inferior de tu computadora y escribe CMD, ábrelo ejecutar como administrador.

- En la ventana de la línea de comandos, copia y pega el siguiente comando y presiona Enter:

pip install nombre_de_la_libreria_aqui

Esto descargará e instalará la librería

Recuerda que debes tener Python instalado previamente en tu computadora. Si aún no tienes Python instalado, regresa al comienzo del libro e instala lo que nesecites.

1. Librería math:

- Utilidad: Ofrece funciones y constantes matemáticas para realizar cálculos avanzados.

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import math`

- Descripción: Útil para cálculos matemáticos precisos, como trigonometría, exponenciación, logaritmos y raíces cuadradas.

- [Documentación de `math`](https://docs.python.org/3/library/math.html)

2. Librería random:

- Utilidad: Genera números aleatorios y facilita operaciones relacionadas con la aleatoriedad.

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import random`

- Descripción: Importante para tareas que requieren introducir un componente aleatorio, como juegos, simulaciones y pruebas.

- [Documentación de `random`](https://docs.python.org/3/library/random.html)

3. Librería os:

- Utilidad: Permite interactuar con el sistema operativo para tareas como navegación por directorios y manipulación de archivos.

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import os`

- Descripción: Esencial para tareas relacionadas con la gestión de archivos y directorios en el sistema operativo.

- [Documentación de `os`](https://docs.python.org/3/library/os.html)

4. Librería datetime:

- Utilidad: Facilita el manejo de fechas y horas en Python.

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import datetime`

- Descripción: Útil para realizar tareas de programación relacionadas con fechas y horas, como cálculos de tiempo.

- [Documentación de `datetime`](https://docs.python.org/3/library/datetime.html)

5. Librería json:

- Utilidad: Permite trabajar con datos en formato JSON (JavaScript Object Notation).

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import json`

- Descripción: Importante para el intercambio de datos estructurados entre aplicaciones y el almacenamiento de configuraciones.

- [Documentación de `json`](https://docs.python.org/3/library/json.html)

6. Librería csv:

- Utilidad: Simplifica la lectura y escritura de archivos CSV (Comma-Separated Values).

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import csv`

- Descripción: Útil para procesar datos tabulares en hojas de cálculo y archivos CSV.

- [Documentación de `csv`](https://docs.python.org/3/library/csv.html)

7. Librería re (expresiones regulares):

- Utilidad: Permite trabajar con patrones de búsqueda y manipulación de cadenas de texto utilizando expresiones regulares.

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import re`

- Descripción: Importante para buscar y manipular texto en situaciones donde se requiere flexibilidad en los patrones de búsqueda.

- [Documentación de `re`](https://docs.python.org/3/library/re.html)

8. Librería urllib.request:

- Utilidad: Facilita la interacción con recursos en línea a través de URLs, como la descarga de contenido web.

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import urllib.request`

- Descripción: Útil para acceder a contenido en línea y realizar solicitudes HTTP.

- [Documentación de `urllib.request`](https://docs.python.org/3/library/urllib.request.html)

9. Librería collections:

- Utilidad: Ofrece contenedores y estructuras de datos adicionales, como diccionarios con orden, colas y pilas.

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import collections`

- Descripción: Útil para tareas de programación que requieren estructuras de datos más avanzadas.

- [Documentación de `collections`](https://docs.python.org/3/library/collections.html)

10. Librería sqlite3:

- Utilidad: Permite la interacción con bases de datos SQLite desde Python.

- Instalación: No se requiere instalación adicional, ya que es parte de la librería estándar de Python.

- Importación: `import sqlite3`

- Descripción: Esencial para trabajar con bases de datos SQLite en aplicaciones Python.

- [Documentación de `sqlite3`](https://docs.python.org/3/library/sqlite3.html)

11. Librería mysql-connector-python:

- Utilidad: La librería `mysql-connector-python` permite a los desarrolladores de Python conectarse y comunicarse con bases de datos MySQL. Es esencial para realizar operaciones de lectura y escritura en bases de datos MySQL desde una aplicación Python.

- Instalación: pip install mysql-connector-python

- Importación: import mysql.connector

- Descripción: mysql-connector-python proporciona una interfaz eficiente para conectarse a servidores MySQL y realizar diversas operaciones de bases de datos, como consultas SELECT, inserciones de datos, actualizaciones y eliminaciones. Facilita la creación de conexiones, la gestión de cursores y la recuperación de resultados. Es una herramienta esencial para aplicaciones que requieren persistencia de datos utilizando MySQL como sistema de gestión de bases de datos.

La librería mysql-connector-python es valiosa para desarrolladores que trabajan con bases de datos MySQL en aplicaciones Python. Permite interactuar de manera efectiva con las bases de datos, lo que es esencial para aplicaciones web, sistemas de gestión, análisis de datos y muchas otras aplicaciones que requieren almacenamiento y recuperación de datos.

- Documentación: Si deseas obtener más información detallada sobre `mysql-connector-python` y cómo utilizarla, puedes consultar la [documentación oficial de mysql-connector-python](https://pypi.org/project/mysql-connector-python/).

12. Librería requests:

- Utilidad: Permite realizar solicitudes HTTP, lo que facilita la comunicación con servidores web y la obtención de datos.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install requests`.

- Importación: `import requests`

- Descripción: Útil para acceder a recursos en línea, interactuar con APIs web y descargar contenido desde la web.

- [Documentación de `requests`](https://docs.python-requests.org/en/master/)

13. Librería numpy:

- Utilidad: Ofrece soporte para arrays y matrices multidimensionales, junto con funciones para realizar cálculos numéricos.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install numpy`.

- Importación: `import numpy`

- Descripción: Ampliamente utilizado en ciencia de datos y cálculos científicos para realizar operaciones numéricas eficientes.

- [Documentación de `numpy`](https://numpy.org/doc/stable/)

14. Librería pandas:

- Utilidad: Proporciona estructuras de datos y herramientas para el análisis y manipulación de datos tabulares.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install pandas`.

- Importación: `import pandas`

- Descripción: Ampliamente utilizado en análisis de datos y procesamiento de conjuntos de datos.

- [Documentación de `pandas`](https://pandas.pydata.org/docs/)

15. Librería matplotlib:

- Utilidad: Facilita la creación de gráficos y visualizaciones de datos.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install matplotlib`.

- Importación: `import matplotlib.pyplot as plt`

- Descripción: Importante para la visualización de datos y la representación gráfica de resultados.

- [Documentación de `matplotlib`](https://matplotlib.org/stable/contents.html)

16. Librería tensorflow:

- Utilidad: Ofrece herramientas para la creación y entrenamiento de modelos de aprendizaje automático, especialmente redes neuronales.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install tensorflow`.

- Importación: `import tensorflow as tf`

- Descripción: Ampliamente utilizado en aprendizaje automático y desarrollo de modelos de inteligencia artificial.

- [Documentación de `tensorflow`](https://www.tensorflow.org/api_docs/python/tf)

17. Librería flask:

- Utilidad: Facilita la creación de aplicaciones web y API RESTful.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install Flask`.

- Importación: `from flask import Flask`

- Descripción: Importante para el desarrollo de aplicaciones web y servicios web.

- [Documentación de `Flask`](https://flask.palletsprojects.com/en/2.1.x/)

18. Librería sqlalchemy:

- Utilidad: Proporciona una interfaz de alto nivel para interactuar con bases de datos relacionales.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install sqlalchemy`.

- Importación: `from sqlalchemy import create_engine`

- Descripción: Importante para la interacción con bases de datos y el mapeo objeto-relacional.

- [Documentación de `sqlalchemy`](https://docs.sqlalchemy.org/en/20/)

19. Librería beautifulsoup4:

- Utilidad: Facilita la extracción de datos de páginas web, en particular, el análisis y la manipulación de documentos HTML y XML.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install beautifulsoup4`.

- Importación: `from bs4 import BeautifulSoup`

- Descripción: Importante para la web scraping y la extracción de información de sitios web.

- [Documentación de `beautifulsoup4`](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)

20. Librería pygame:

- Utilidad: Facilita el desarrollo de juegos y aplicaciones multimedia.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install pygame`.

- Importación: `import pygame`

- Descripción: Ampliamente utilizado en el desarrollo de juegos y aplicaciones interactivas.

- [Documentación de `pygame`](https://www.pygame.org/wiki/GettingStarted)

21. Librería selenium:

- Utilidad: Permite la automatización de navegadores web y la interacción con páginas web.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install selenium`.

- Importación: `from selenium import webdriver`

- Descripción: Importante para la automatización de pruebas web y tareas repetitivas en navegadores.

- [Documentación de `selenium`](https://selenium-python.readthedocs.io/)

22. Librería openpyxl:

- Utilidad: Facilita la creación y manipulación de archivos de hojas de cálculo de Excel.

- Instalación: Puedes instalarlo utilizando `pip` con el siguiente comando: `pip install openpyxl`.

- Importación: `from openpyxl import Workbook`

- Descripción: Útil para la automatización de tareas relacionadas con hojas de cálculo de Excel.

- [Documentación de `openpyxl`](https://openpyxl.readthedocs.io/en/3.0/)

19. Escritura de documentos

Entendido. A continuación, proporcionaré una lista de los modos de apertura de archivos comunes y luego generaré ejemplos completos para leer, escribir, modificar, agregar y eliminar datos en archivos CSV.

Modos de apertura de archivos:

r: Lectura (Read). Abre el archivo en modo lectura.

W: Escritura (Write). Abre el archivo en modo escritura. Si el archivo existe, lo sobrescribe; de lo contrario, crea uno nuevo.

a: Agregación (Append). Abre el archivo en modo de agregación. Si el archivo existe, agrega datos al final; de lo contrario, crea uno nuevo.

x: Exclusivo (Exclusive). Abre el archivo en modo exclusivo para escritura, creando uno nuevo si no existe.

b: Modo binario (Binary). Abre el archivo en modo binario, por ejemplo, 'rb' para lectura binaria o 'wb' para escritura binaria.

Ahora, generemos ejemplos completos para cada uno de estos modos:

```python
# Ejemplo 1: Leer un archivo CSV (Modo 'r')
import csv

with open('archivo.csv', 'r') as archivo_csv:
    lector_csv = csv.reader(archivo_csv)
    for fila in lector_csv:
        print(fila)

```

```python
# Ejemplo 2: Escribir datos en un archivo CSV (Modo 'w')
import csv

datos = [
    ['Nombre', 'Edad', 'Ciudad'],
    ['Juan', 30, 'Madrid'],
    ['María', 25, 'Barcelona'],
]

with open('nuevo_archivo.csv', 'w', newline='') as archivo_csv:
    escritor_csv = csv.writer(archivo_csv)
    for fila in datos:
        escritor_csv.writerow(fila)
```

```python
# Ejemplo 3: Agregar datos a un archivo CSV (Modo 'a')
import csv

nuevos_datos = ['Pedro', 35, 'Sevilla']

with open('archivo.csv', 'a', newline='') as archivo_csv:
    escritor_csv = csv.writer(archivo_csv)
    escritor_csv.writerow(nuevos_datos)
```

```python
# Ejemplo 4: Modificar datos en un archivo CSV (Lectura, Modificación, Escritura)
import csv

# Leer datos existentes
datos_existente = []
with open('archivo.csv', 'r') as archivo_csv:
    lector_csv = csv.reader(archivo_csv)
    for fila in lector_csv:
        datos_existente.append(fila)

# Modificar datos (por ejemplo, cambiar la edad de María)
for fila in datos_existente:
    if fila[0] == 'María':
        fila[1] = 26  # Cambiamos la edad

# Escribir datos modificados en un nuevo archivo CSV
with open('archivo_modificado.csv', 'w', newline='') as archivo_csv:
    escritor_csv = csv.writer(archivo_csv)
    for fila in datos_existente:
        escritor_csv.writerow(fila)
```

```python
# Ejemplo 5: Eliminar una fila específica en un archivo CSV (Lectura, Filtro, Escritura)
import csv

# Leer datos existentes y filtrar las filas que deseas conservar
datos_existente = []
fila_a_eliminar = 'Juan'  # Nombre de la fila que se eliminará

with open('archivo.csv', 'r') as archivo_csv:
    lector_csv = csv.reader(archivo_csv)
    for fila in lector_csv:
        if fila[0] != fila_a_eliminar:
            datos_existente.append(fila)

# Escribir los datos filtrados en un nuevo archivo CSV
with open('archivo_sin_juan.csv', 'w', newline='') as archivo_csv:
    escritor_csv = csv.writer(archivo_csv)
    for fila in datos_existente:
        escritor_csv.writerow(fila)
```

Para profundizar mas visita: https://realpython.com/python-csv/

20. Importación de documentos

La importación de documentos se refiere a la capacidad de Python para cargar y utilizar módulos o archivos externos en tu programa. Esto es especialmente útil cuando deseas utilizar funciones, clases o datos definidos en otros archivos Python o módulos preexistentes. Para importar documentos en Python, utilizas la palabra clave import.

Ejemplo de Importación de un Documento:

Supongamos que tienes un archivo llamado operaciones.py que contiene funciones matemáticas como suma y resta:

operaciones.py

```python
# operaciones.py

def suma(a, b):
    return a + b

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

```

Puedes importar y utilizar estas funciones en otro archivo Python de la siguiente manera:

operaciones.py | llamar_operaciones.py

```python
# Importar el archivo operaciones.py
import operaciones

# Usar las funciones del archivo importado
resultado_suma = operaciones.suma(5, 3)
resultado_resta = operaciones.resta(10, 4)

print(f"Resultado de la suma: {resultado_suma}")
print(f"Resultado de la resta: {resultado_resta}")

```

En este ejemplo, el archivo llamar_operaciones.py importa el archivo operaciones.py utilizando import operaciones. Luego, puede acceder a las funciones suma y restas definidas en operaciones.py y utilizarlas en el programa principal.

La importación de documentos es una técnica esencial para organizar y reutilizar código en Python, ya que te permite dividir tu código en módulos más pequeños y fáciles de mantener. Puedes importar tanto módulos estándar de Python como tus propios archivos personalizados.

21. Programación Orientada a Objetos

Introducción a la Programación Orientada a Objetos (POO):

- La POO es un paradigma de programación que se basa en la idea de tratar los datos y las acciones que se realizan sobre ellos como "objetos".

- Los objetos son instancias de "clases", que son como plantillas o moldes para crear objetos.

- En Python, todo es un objeto, desde números y cadenas hasta estructuras de datos más complejas.

Clases en Python:

- Una clase es una definición de un objeto. Define sus atributos (datos) y métodos (funciones) que pueden realizar operaciones en esos datos.

- Se define una clase utilizando la palabra clave `class` seguida del nombre de la clase.

```
28  class MiClase:
29      # Atributos
30      atributo1 = "Hola"
31      atributo2 = 42
32
33      # Métodos
34      def metodo1(self):
35          return "Método 1 ejecutado"
36
```

Objetos en Python:

- Un objeto es una instancia de una clase. Puedes crear múltiples objetos a partir de la misma clase.

```
39    objeto1 = MiClase()
40    objeto2 = MiClase()
```

Atributos de Clase y de Instancia:

- Los atributos pueden ser de clase (compartidos por todos los objetos de la clase) o de instancia (propiedad específica de cada objeto).

```
44    class MiClase:
45        atributo_de_clase = "Soy un atributo de clase"
46
47        def __init__(self, valor):
48            self.atributo_de_instancia = valor
49
```

Métodos en Clases:

- Los métodos son funciones definidas dentro de una clase y operan en los atributos de la clase.

- El método `__init__` es el constructor y se llama automáticamente cuando se crea un objeto de la clase.

```
52    class MiClase:
53        def __init__(self, valor):
54            self.atributo = valor
55
56        def mostrar_atributo(self):
57            print(self.atributo)
58
59    objeto = MiClase("Hola, soy un objeto")
60    objeto.mostrar_atributo()  # Muestra "Hola, soy un objeto"
61
```

Encapsulación:

- En Python, no hay un control estricto sobre el acceso a los atributos, pero se usa una convención para indicar que un atributo es privado agregando un guion bajo antes del nombre (ejemplo: `_atributo_privado`).

Herencia:

- La herencia permite crear una nueva clase que hereda atributos y métodos de una clase existente.

```
66  class ClaseBase:
67      def metodo_base(self):
68          print("Método base")
69
70  class ClaseHija(ClaseBase):
71      def metodo_hijo(self):
72          print("Método hijo")
73
74  objeto_hijo = ClaseHija()
75  objeto_hijo.metodo_base()  # Puede acceder al método de la clase base
```

Polimorfismo:

- El polimorfismo permite que diferentes clases implementen métodos con el mismo nombre, pero que funcionen de manera diferente.

```python
class Perro:
    def sonido(self):
        print("Ladrar")

class Gato:
    def sonido(self):
        print("Maullar")

def hacer_sonar(animal):
    animal.sonido()

perro = Perro()
gato = Gato()
hacer_sonar(perro)  # Ladrar
hacer_sonar(gato)   # Maullar

```

Estos son los conceptos clave de la programación orientada a objetos en Python. Con una comprensión sólida de estos conceptos, puedes crear programas más organizados y reutilizables mediante la creación de clases y objetos.

22. Conexión a base de datos

Para esta parte del libro necesitas ya tener conocimientos en base de datos.

Python con MySQL y SQLite

- Importación de las librerías de bases de datos.

- Establecimiento de una conexión a la base de datos.

- Creación de una tabla.

- Inserción de datos.

- Consulta de datos.

- Actualización de datos.

- Eliminación de datos.

- Cierre de la conexión.

MySQL y SQLite:

- Ambas bases de datos admiten SQL estándar, pero difieren en términos de escalabilidad y configuración.

- MySQL es una base de datos de servidor que se utiliza en aplicaciones más grandes y requiere un servidor MySQL en ejecución.

- SQLite es una base de datos de un solo archivo que es ideal para aplicaciones más pequeñas y no requiere un servidor.

1. MySQL (conector mysql-connector-python)

- Comando de instalación: `pip install mysql-connector-python`

- [Documentación de MySQL Connector/Python](https://dev.mysql.com/doc/connector-python/en/)

- Ejemplo de conexión MySQL:

```
26  import mysql.connector
27
28  # Conexión a la base de datos MySQL
29  conexion = mysql.connector.connect(
30    host="localhost",  # Dirección del servidor de la base de datos (cambia esto si es diferente)
31    user="root",  # Nombre de usuario de la base de datos (cambia esto si es diferente)
32    password="",  # Contraseña del usuario (deja en blanco si no tienes contraseña)
33    database="nombre_basedatos"# Nombre de la base de datos a la que te quieres conectar (cambia esto)
34  )
35  cursor = conexion.cursor()
```

Se crea un objeto cursor. En el contexto de bases de datos, un cursor es un objeto que te permite interactuar con la base de datos, enviar consultas y recuperar resultados.

Explicación detallada:

1. conexión es el objeto que representa la conexión a la base de datos que se estableció previamente.

2. `cursor()` es un método de este objeto `conexion`. Cuando se llama a conexion.cursor(), se crea un nuevo objeto cursor que está asociado con la conexión a la base de datos. Este cursor actúa como un puntero o marcador de posición que te permite ejecutar consultas SQL y recuperar resultados de esas consultas.

Por lo tanto, después de crear este objeto cursor, puedes usarlo para realizar varias operaciones en la base de datos, como ejecutar consultas SELECT, INSERT, UPDATE o DELETE, y recuperar los resultados de esas operaciones.

Si deseas interactuar con la base de datos, generalmente primero creas un cursor, ejecutas tus consultas o comandos SQL usando ese cursor y luego cierras el cursor y la conexión cuando hayas terminado.

Ejemplo de conexión SQLite:

```
47  import sqlite3
48
49  # Conexión a la base de datos SQLite (o creación de un nuevo archivo si no existe)
50  conexion = sqlite3.connect("basedatos.db")
51  cursor = conexion.cursor()
52
```

Operaciones CRUD de ejemplo en ambas bases de datos:(debajo de cada conexión viene el Crud)

```
 90  # Creación de una tabla
 91  cursor.execute("""CREATE TABLE IF NOT EXISTS nombre_tabla
 92                    (id INT AUTO_INCREMENT PRIMARY KEY, nombre VARCHAR(255), edad INT)""")
 93
 94  # Inserción de datos en la tabla
 95  insert_query = "INSERT INTO nombre_tabla (nombre, edad) VALUES (%s, %s)"
 96  datos = ("Ejemplo", 30)
 97  cursor.execute(insert_query, datos)
 98  conexion.commit()
 99
100  # Consulta de datos en la tabla
101  select_query = "SELECT * FROM nombre_tabla"
102  cursor.execute(select_query)
103  resultados = cursor.fetchall()
104  for fila in resultados:
105      print(f"ID: {fila[0]}, Nombre: {fila[1]}, Edad: {fila[2]}")
```

```python
# Actualización de datos en la tabla
update_query = "UPDATE nombre_tabla SET edad = %s WHERE nombre = %s"
datos = (25, "Ejemplo")
cursor.execute(update_query, datos)
conexion.commit()

# Eliminación de datos en la tabla
delete_query = "DELETE FROM nombre_tabla WHERE nombre = %s"
datos = ("Ejemplo",)
cursor.execute(delete_query, datos)
conexion.commit()

# Cierre de la conexión
cursor.close()
conexion.close()
```

23. Interfaz grafica

Tkinter: La Biblioteca de Interfaz Gráfica de Usuario (GUI) de Python

Tkinter es una biblioteca estándar de Python que se utiliza para crear interfaces gráficas de usuario (GUI). Con Tkinter, puedes crear ventanas, etiquetas, botones, cuadros de texto, y muchos otros componentes de GUI para interactuar con tus programas de Python.

Características Clave de Tkinter:

1. Facilidad de Uso: Tkinter es conocido por su simplicidad y facilidad de uso. Es una excelente elección para principiantes en programación GUI.

2. Portabilidad: Las aplicaciones creadas con Tkinter funcionan en múltiples sistemas operativos, como Windows, macOS y Linux.

3. Ampliamente Utilizado: Tkinter es una de las bibliotecas GUI más utilizadas en Python y es compatible con muchas aplicaciones y proyectos de código abierto.

Pasos Básicos para Crear una Aplicación Tkinter:

1. Importar Tkinter: Debes importar el módulo Tkinter con from tkinter import *. (con * indicamos que llamamos a todo o que abarca esta librería para no ir poniendo los nombres de los elementos de uno en uno.

2. Crear una Ventana Principal: Utiliza Tk() para crear una ventana principal que actuará como el contenedor de tus componentes de GUI.

3. Agregar Componentes: Puedes agregar etiquetas, botones, cuadros de texto, etc., a la ventana principal.

4. Definir Funciones (def): Debes definir funciones que se ejecutarán cuando los usuarios interactúen con los componentes (por ejemplo, al hacer clic en un botón).

5. Ejecutar la Aplicación: Llama a ventana.mainloop(), para iniciar la aplicación y permitir que los usuarios interactúen con la GUI.(El nombre de la función no necesariamente tiene que ser ventana esa es tu elección).

Tkinter es una herramienta poderosa para crear aplicaciones de escritorio con interfaces de usuario intuitivas y visualmente atractivas. A medida que profundices en Tkinter, descubrirás que puedes personalizar y extender tus aplicaciones GUI de muchas maneras, lo que te permite crear aplicaciones útiles y atractivas en Python.

Componentes de Tkinter:

1. Ventana (‘Tk’):

- Propósito: La ventana principal de la aplicación. Es el contenedor principal para todos los otros componentes.

2. Etiqueta (‘Label’):

- Propósito: Muestra texto o imágenes.

3. Cuadro de Texto de Entrada (‘Entry’):

- Propósito: Permite al usuario introducir texto o datos.

4. Botón (‘Button’):

- Propósito: Activa una acción cuando se hace clic.

5. Cuadro de Texto (‘Text’):

- Propósito: Permite la entrada o visualización de texto de varias líneas.

6. Área de Desplazamiento (‘Scrollbar’):

- Propósito: Proporciona barras de desplazamiento para componentes como cuadros de texto.

7. Lista (‘Listbox’):

- Propósito: Muestra una lista de elementos para que el usuario seleccione.

8. Cuadro de Verificación (‘Checkbutton’):

- Propósito: Permite al usuario seleccionar una o más opciones de una lista.

9. Botón de Radio (‘Radiobutton’):

- Propósito: Permite al usuario seleccionar una sola opción de un conjunto de opciones mutuamente excluyentes.

10. Cuadro de Opción (‘OptionMenu’):

- Propósito: Proporciona una lista desplegable de opciones para que el usuario seleccione una.

11. Marco (‘Frame’):

- Propósito: Un contenedor que agrupa otros componentes para facilitar la organización y disposición.

12. Ventana emergente (‘Toplevel’):

- Propósito: Crea ventanas secundarias o emergentes que son independientes de la ventana principal.

13. Imagen (‘PhotoImage’):

- Propósito: Permite la incorporación de imágenes en tu aplicación Tkinter.

14. Canvas (‘Canvas’):

- Propósito: Proporciona un área en blanco donde puedes dibujar gráficos, figuras y más.

15. Menús (‘Menu’):

- Propósito: Agrega barras de menú y menús desplegables a tu aplicación.

16. Barra de progreso (‘ProgressBar’):

- Propósito: Muestra el progreso de una tarea o proceso en la aplicación.

17. Cuadro de Diálogo (‘DialogBox’):

- Propósito: Crea ventanas de diálogo para interactuar con el usuario.

Los métodos de posicionamiento en Tkinter: pack(), grid(), y place().

1. Método pack():

- Descripción: El método pack() se utiliza para organizar los componentes de manera que se ajusten automáticamente en una ventana principal de Tkinter. Coloca los componentes uno debajo del otro en una dirección (vertical u horizontal) y ajusta automáticamente el espacio entre ellos.

- Uso Típico: pack() se utiliza comúnmente cuando deseas que los componentes se alineen en una única dirección, ya sea vertical o horizontal, y permites que Tkinter ajuste automáticamente el espacio entre ellos.

- Ejemplo:

```
etiqueta1 = tk.Label(ventana, text="Etiqueta 1")
etiqueta2 = tk.Label(ventana, text="Etiqueta 2")

etiqueta1.pack()
etiqueta2.pack()
```

2. Método grid():

- Descripción: El método grid() se utiliza para organizar los componentes en una cuadrícula (filas y columnas) en una ventana principal de Tkinter. Esto te permite un control preciso sobre la ubicación y alineación de los componentes en relación con otros componentes.

- Uso Típico: grid() se utiliza cuando necesitas una disposición en forma de cuadrícula y deseas controlar la ubicación exacta de los componentes en filas y columnas.

- Ejemplo:

```
etiqueta1 = tk.Label(ventana, text="Etiqueta 1")
etiqueta2 = tk.Label(ventana, text="Etiqueta 2")

etiqueta1.grid(row=0, column=0)
etiqueta2.grid(row=0, column=1)
```

3. Método place():

- Descripción: El método place() se utiliza para posicionar los componentes de manera precisa en una ventana principal de Tkinter mediante coordenadas X e Y. Esto te permite un control preciso sobre la ubicación de los componentes en la ventana.

- Uso Típico: place() se utiliza cuando necesitas un control detallado y específico sobre la ubicación de los componentes en la ventana y deseas posicionarlos en coordenadas exactas.

- Ejemplo:

```
etiqueta1 = tk.Label(ventana, text="Etiqueta 1")
etiqueta2 = tk.Label(ventana, text="Etiqueta 2")

etiqueta1.place(x=50, y=30)
etiqueta2.place(x=100, y=60)
```

En resumen, la elección de pack(), grid(), o place() depende de tus necesidades de diseño. pack() es útil para diseños simples y automáticos, grid() proporciona un control detallado en una cuadrícula, y place() es útil para un posicionamiento preciso en coordenadas específicas. Puedes usar estos métodos según tus necesidades específicas para diseñar interfaces de usuario personalizadas en Tkinter.

Propiedades de Diseño de Componentes:

1. Fondo (‘bg’): Define el color de fondo del componente.

2. Tamaño (‘width’, ‘height’): Especifica el tamaño del componente en píxeles.

3. Texto (‘text’): Define el texto que se mostrará en el componente (por ejemplo, en una etiqueta o botón).

4. Fuente (‘font’): Permite especificar el tipo de fuente, tamaño y estilo del texto.

5. Color de Texto (‘fg’): Establece el color del texto del componente.

6. Borde (‘borderwidth’): Define el ancho del borde del componente.

7. Cursor (‘cursor’): Especifica el cursor del mouse que se mostrará cuando el puntero esté sobre el componente.

8. Imagen (‘image’): Permite asignar una imagen a un componente.

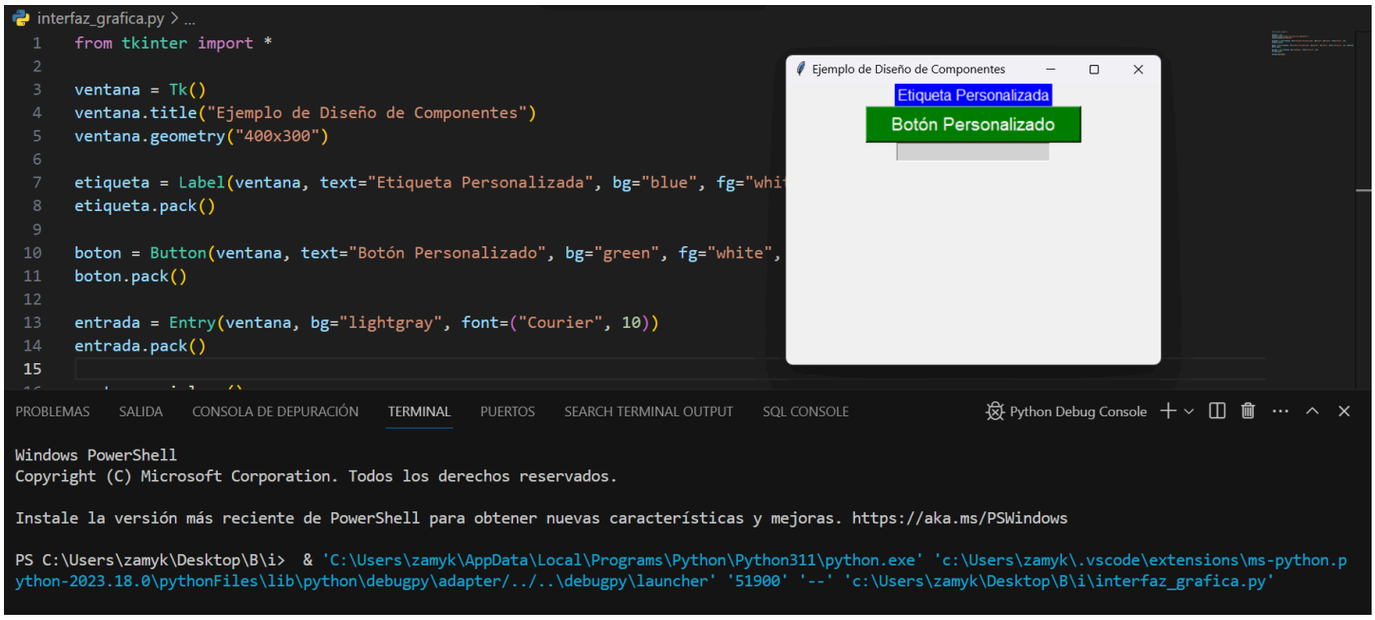
9. Variedad de Opciones (‘options’): Cada componente puede tener opciones específicas adicionales para personalizar su comportamiento y diseño.

Ejemplo de Uso de Propiedades de Diseño:

interfaz_grafica.py > ...

```python
from tkinter import *

ventana = Tk()
ventana.title("Ejemplo de Diseño de Componentes")
ventana.geometry("400x300")

etiqueta = Label(ventana, text="Etiqueta Personalizada", bg="blue", fg="white", font=("Arial", 12))
etiqueta.pack()

boton = Button(ventana, text="Botón Personalizado", bg="green", fg="white", font=("Helvetica", 14), width=20)
boton.pack()

entrada = Entry(ventana, bg="lightgray", font=("Courier", 10))
entrada.pack()

ventana.mainloop()
```

Resultado



Este ejemplo muestra cómo personalizar el diseño de una etiqueta, un botón y un cuadro de texto de entrada utilizando las propiedades mencionadas. Puedes ajustar estas propiedades según tus preferencias y necesidades de diseño específicas que tu requieras.

Documentación oficial de Tkinter en Python](https://docs.python.org/3/library/tkinter.html)

La documentación oficial proporciona una descripción detallada de los componentes de Tkinter, métodos, ejemplos y más. Puedes explorarla para obtener información completa y detallada sobre cómo trabajar con Tkinter en tus aplicaciones de GUI de Python.

24. Empaquetado

1. PyInstaller:

- Utilidad: PyInstaller es una herramienta que permite convertir tus scripts de Python en ejecutables independientes para Windows, macOS y Linux.

- Comando de instalación: `pip install pyinstaller`

- [Página oficial de PyInstaller](http://www.pyinstaller.org)

- Ejemplo de conversión: pyinstaller --onefile script.py

2. cx_Freeze:

- Utilidad: cx_Freeze es una herramienta que permite crear ejecutables a partir de tus scripts de Python. Ofrece la posibilidad de empaquetar tus propias bibliotecas para una distribución más flexible.

- Comando de instalación: `pip install cx_Freeze`

- [Página oficial de cx_Freeze](https://anthony-tuininga.github.io/cx_Freeze/)

- Ejemplo de conversión:

```
import cx_Freeze

executables = [cx_Freeze.Executable("script.py")]

cx_Freeze.setup(
    name="MiPrograma",
    options={"build_exe": {"packages": ["tu_libreria"]}},
    executables=executables
)
```

3. py2exe:

- Utilidad: py2exe es una herramienta específica para Windows que te permite convertir tus scripts de Python en ejecutables de Windows (archivos .exe).

- Comando de instalación: `pip install py2exe`

- [Página oficial de py2exe](http://www.py2exe.org)

- Ejemplo de conversión (configuración en el archivo `setup.py`):

```
from distutils.core import setup
import py2exe

setup(console=["script.py"])
```

4. Py2App:

- Utilidad: Py2App es una herramienta destinada a usuarios de macOS. Permite crear aplicaciones macOS a partir de scripts de Python.

- Comando de instalación: No se instala mediante pip; se utiliza específicamente en macOS.

- [Página oficial de Py2App](https://py2app.readthedocs.io/en/latest/)

- Ejemplo de conversión (configuración en un archivo `setup.py`):

```
from setuptools import setup

APP = ['script.py']
DATA_FILES = []
OPTIONS = {'argv_emulation': True}

setup(
    app=APP,
    data_files=DATA_FILES,
    options={'py2app': OPTIONS},
    setup_requires=['py2app'],
)
```

5. auto-py-to-exe:

- Utilidad: auto-py-to-exe es una interfaz gráfica que simplifica la conversión de scripts de Python en ejecutables para Windows.

- Comando de instalación: `pip install auto-py-to-exe`

- [Página oficial de auto-py-to-exe](https://github.com/beeware/auto-py-to-exe)

6. Inno Setup:

- Utilidad: Inno Setup es una herramienta de creación de instaladores de aplicaciones de Windows. Aunque no convierte scripts de Python en ejecutables, se utiliza para empaquetar aplicaciones de Windows en un instalador.

- Comando de instalación: No se instala mediante pip; es una herramienta de creación de instaladores para Windows.

- [Página oficial de Inno Setup](http://www.jrsoftware.org/isinfo.php)

Ejercicios

Pon en práctica tus conocimientos

Realiza Lista de 20 ejercicios básicos de Python

1. Hola, Mundo!
- Imprime "Hola, Mundo!" en la pantalla.

2. Operaciones Matemáticas
- Realiza operaciones matemáticas simples como suma, resta, multiplicación y división.

3. Cálculo del Área de un Triángulo
- Escribe un programa que calcule el área de un triángulo utilizando la fórmula: (base * altura) / 2.

4. Conversión de Celsius a Fahrenheit
- Convierte una temperatura en grados Celsius a grados Fahrenheit usando la fórmula: F = (C * 9/5) + 32.

5. Determinar si un Número es Par o Impar
- Pide al usuario un número y determina si es par o impar.

6. Calculadora Simple
- Crea una calculadora que realice operaciones básicas como suma, resta, multiplicación y división.

7. Contador de Palabras
- Pide al usuario una frase y cuenta cuántas palabras contiene.

8. Tabla de Multiplicar
- Genera la tabla de multiplicar de un número dado por el usuario.

9. Mayor de Tres Números
- Solicita tres números y determina cuál es el mayor de ellos.

10. Factorial de un Número
- Calcula el factorial de un número ingresado por el usuario.

11. Generador de Números Primos
- **Crea un programa que genere números primos hasta un límite especificado.**

12. Cálculo de la Media y la Desviación Estándar
- **Pide al usuario una serie de números y calcula su media y desviación estándar.**

13. Adivina el Número
- **Genera un número aleatorio y permite al usuario adivinarlo.**

14. Conversión de Decimales a Binarios
- **Convierte un número decimal en su representación binaria.**

15. Comprobador de Palíndromos
- **Verifica si una palabra o frase es un palíndromo (se lee igual de izquierda a derecha y viceversa).**

16. Cifrado César
- **Implementa el cifrado César para cifrar y descifrar mensajes.**

17. Calculadora de Interés Compuesto
- **Calcula el interés compuesto a partir de la inversión inicial, la tasa de interés y el tiempo.**

18. Generador de Secuencia Fibonacci
- **Genera los primeros n términos de la secuencia Fibonacci.**

19. Juego de Piedra, Papel y Tijeras
- **Crea un juego simple de Piedra, Papel y Tijeras que permita al usuario jugar contra la computadora.**

20. Cronómetro
- **Crea un cronómetro simple que permita al usuario medir el tiempo transcurrido.**

Lista de 20 ejercicios de complejidad media en Python:

1. Calculadora de Factorial Recursivo
- Escribe una función que calcule el factorial de un número de forma recursiva.

2. Generador de Contraseñas Aleatorias
- Crea un programa que genere contraseñas aleatorias con letras, números y símbolos.

3. Simulador de Dados
- Crea un programa que simule el lanzamiento de dados y calcule la suma de los valores.

4. Contador de Palabras en un Archivo de Texto
- Lee un archivo de texto y cuenta cuántas veces aparece cada palabra.

5. Calculadora de Distancia Euclidiana
- Escribe una función que calcule la distancia euclidiana entre dos puntos en un plano.

6. Validador de Tarjetas de Crédito
- Implementa un programa que valide números de tarjetas de crédito utilizando el algoritmo de Luhn.

7. Buscador de Archivos Duplicados
- Escanea un directorio en busca de archivos duplicados y muestra una lista de duplicados.

8. Generador de Tablas de Multiplicar
- Crea un programa que genere tablas de multiplicar personalizadas.

9. Juego del Ahorcado
- Implementa el juego del ahorcado, donde el jugador debe adivinar una palabra oculta.

10. Simulador de Banco
- Crea un sistema simple de simulación bancaria que permita realizar depósitos y retiros.

11. Análisis de Datos de Venta
- **Lee un archivo CSV de datos de ventas y calcula el total de ventas, promedio, máximo y mínimo.**

12. Ordenamiento de Listas
- **Escribe un algoritmo de ordenamiento (por ejemplo, burbuja o selección) para ordenar una lista de números.**

13. Juego de Sudoku
- **Crea un juego de Sudoku con generación automática de tableros y verificación de soluciones.**

14. Calculadora de Impuestos
- **Crea una calculadora que determine el impuesto a pagar según un ingreso y una tasa de impuesto.**

15. Convertidor de Unidades
- **Implementa un convertidor que convierta entre diferentes unidades de medida (por ejemplo, de metros a pies).**

16. Generador de Gráficos de Barras
- **Crea un programa que genere un gráfico de barras a partir de datos ingresados por el usuario.**

17. Simulador de Juego de Rol (RPG)
- **Diseña un juego de rol simple con personajes, combates y misiones.**

18. Calculadora de Tiempo de Viaje
- **Calcula el tiempo de viaje entre dos ciudades conociendo la distancia y la velocidad promedio.**

19. Detección de Anagramas
- **Escribe una función que detecte si dos palabras son anagramas entre sí.**

20. Cifrado RSA Simple

- Implementa un cifrado RSA básico para encriptar y desencriptar mensajes.