Python

# Guía de ejercicios complementarios

# ¿Qué es?

La Guía fue creada para que puedas focalizar en reforzar y afianzar, potenciar y poner en práctica los saberes adquiridos en clase. No tiene corrección ni los ejercicios serán evaluados dentro de las entregas.

**¡A practicar!** 😉

Nota: Te recomendamos que descargues el archivo para que lo puedas editar

CODERHOUSE

# Sobre los ejercicios

Podemos identificar los elementos en:

**Ejercicios:** son propuestas de ejercitación práctica complementaria, basadas en problemáticas comunes del desarrollo de aplicaciones. Algunas consignas son específicas, pero otras son un poco más abiertas, con la intención de familiarizarnos con formatos narrativos en la solicitud de funcionalidades a programar.

**Notas:** hay ejercicios que presentan una nota, la cual nos brinda más información o tips para desarrollarlo

(nota: modo hardcore hacerlo sin leer las notas).

**Resoluciones propuestas:** encontrarás las resoluciones para que las compares con tus propias soluciones. Es propuesta porque **no es perfecta**, en desarrollo de software ninguna solución es infalible, y mucho depende del contexto, por lo cual si tu resolución presenta una funcionalidad similar o mejor a la propuesta ¡felicidades, resolviste el ejercicio!.

**Repositorio en GitHub:** Podrás encontrar las resoluciones alojadas en GitHub.

Guía complementaria – Parte 1

# Números, cadenas de caracteres, listas y tuplas

# Actividad 1
## Calculadora base

Escribe un programa que solicite al usuario dos números enteros. Luego, muestra por pantalla la suma, resta, multiplicación y división de esos dos números.

# Actividad 2
## Los últimos caracteres ríen mejor

Crea un programa que tome una cadena de texto como entrada del usuario. Luego, muestra por pantalla los primeros tres caracteres de la cadena, seguidos por los tres últimos caracteres. Además, concatena ambos subconjuntos y muestra el resultado.

# Actividad 3
## Cambio manual

Crea un programa que inicie con una lista de números enteros. Luego, solicita al usuario un número entero y un índice para reemplazar un elemento en la lista por el nuevo número ingresado en el índice ingresado.
Imprime la lista resultante.

**Notas actividad 3**
Aclaraciones

Corrobora que el índice que solicitas al usuario no exceda los índices que maneja la lista de números enteros.
La lista de números inicial es a elección del programador

CODERHOUSE

# Actividad 4
# ¿Cuánto y dónde?

Crea un programa que, teniendo en cuenta la siguiente tupla, muestre el valor del segundo elemento de la misma. Además, debe mostrar cuántas veces se repite este valor y cuál es el índice del valor repetido.

palabras_tupla = ("manzana", "pera", "uva", "naranja", "sandía", "manzana", "plátano", "kiwi", "pera", "fresa", "mango", "uva", "cereza", "manzana", "durazno")

**Notas actividad 4**
Aclaraciones

Ten en cuenta que la primera aparición sería la posición 1 de la tupla, por lo cual deberíamos buscar el índice de la siguiente aparición.

# Resoluciones

# Ejemplos de solución

## Actividad 1

```python
# Solicitar al usuario dos números enteros
num1 = int(input("Ingresa el primer número: "))
num2 = int(input("Ingresa el segundo número: "))

# Realizar operaciones y mostrar resultados
suma = num1 + num2
resta = num1 - num2
multiplicacion = num1 * num2
division = num1 / num2

print("Suma:", suma)
print("Resta:", resta)
print("Multiplicación:", multiplicacion)
print("División:", division)
```

## Actividad 2

```python
# Solicitar al usuario una cadena de texto
cadena = input("Ingresa una cadena de texto: ")

# Obtener los primeros tres caracteres
primeros_tres = cadena[:3]

# Obtener los tres últimos caracteres
ultimos_tres = cadena[-3:]

# Concatenar los subconjuntos y mostrar el resultado
concatenados = primeros_tres + ultimos_tres
print("Primeros tres caracteres:", primeros_tres)
print("Últimos tres caracteres:", ultimos_tres)
print("Concatenados:", concatenados)
```

# Actividad 3

```python
# Iniciar una lista de números enteros
numeros = [10, 20, 30, 40, 50]

# Solicitar al usuario un número y un índice
numero_nuevo = int(input("Ingresa un número nuevo: "))
indice = int(input("Ingresa un índice (0 al 4): "))

# Reemplazar el número en el índice dado
numeros[indice] = numero_nuevo

# Mostrar la lista resultante
print("Lista resultante:", numeros)
```

# Actividad 4

```python
# Definir la tupla de palabras
palabras_tupla = ("manzana", "pera", "uva", "naranja", "sandía", "manzana", "plátano",
                  "kiwi", "pera", "fresa", "mango", "uva", "cereza", "manzana", "durazno")

# Obtener el valor del segundo elemento de la tupla
segundo_elemento = palabras_tupla[1]

# Contar las repeticiones del segundo elemento
repeticiones = palabras_tupla.count(segundo_elemento)

# Encontrar los índices del segundo elemento
indice = palabras_tupla.index(segundo_elemento, 2)

# Mostrar resultados
print("El valor del segundo elemento es:", segundo_elemento)
print("El valor se repite", repeticiones, "veces.")
print("El índice de la segunda aparicion del valor es:", indice)
```

Guía complementaria – Parte 2

# Conjuntos, diccionarios y métodos de colecciones

CODERHOUSE

# Actividad 1
# Nombres únicos

Crea un programa que solicite al usuario ingresar nombres separados por comas. Luego, crea un conjunto con los nombres ingresados y muestra por pantalla la cantidad de nombres únicos en el conjunto.

**Notas actividad 1**
Aclaraciones

Para separar los nombres recuerda que puedes utilizar split.

# Actividad 2
## Actualizando la tienda

Crea un programa que simule un inventario de productos en una tienda. Inicia con un diccionario que contenga algunos productos y sus cantidades. Luego, permite al usuario agregar un nuevo producto con su cantidad y actualizar la cantidad de un producto existente. Muestra el inventario actualizado.

Productos y cantidades:

Manzanas => 50

Bananas => 30

Peras => 40

# Actividad 3
## Python uno, Python dos, Python 3...

Crea un programa que tome una oración ingresada por el usuario y realice las siguientes operaciones: convierte la oración a mayúsculas, cuenta cuántas veces aparece la palabra "python", y muestra la oración sin espacios en blanco al inicio y al final.

# Actividad 4
## Intersectando y uniendo

Crea dos conjuntos con números ingresados por el usuario y separados por coma.
Luego, muestra cuáles elementos tienen en común los conjuntos y crea un nuevo conjunto que contenga la unión de ambos.

**Notas actividad 4**
Aclaraciones

Recuerda el uso de split para separar valores de un string.

# Resoluciones

# Ejemplos de solución

## Actividad 1

```python
# Solicitar al usuario ingresar nombres separados por comas
nombres_ingresados = input("Ingresa nombres separados por comas: ")

# Crear un conjunto a partir de los nombres
conjunto_nombres = set(nombres_ingresados.split(","))

# Mostrar la cantidad de nombres únicos en el conjunto
cantidad_nombres_unicos = len(conjunto_nombres)
print("Cantidad de nombres únicos:", cantidad_nombres_unicos)
```

## Actividad 2

```python
# Inicializar el diccionario de inventario de productos
inventario = {
    "Manzanas": 50,
    "Bananas": 30,
    "Peras": 40
}

# Agregar un nuevo producto al inventario
nuevo_producto = input("Ingresa el nombre de un nuevo producto: ")
cantidad_nuevo_producto = int(input(f"Ingresa la cantidad de {nuevo_producto}: "))
inventario[nuevo_producto] = cantidad_nuevo_producto

# Actualizar la cantidad de un producto existente
producto_existente = input("Ingresa el nombre de un producto existente: ")
nueva_cantidad_existente = int(input(f"Ingresa la nueva cantidad de {producto_existente}: "))
inventario[producto_existente] = nueva_cantidad_existente

# Mostrar el inventario actualizado
print("Inventario actualizado:", inventario)
```

# Actividad 3

```python
# Solicitar al usuario una oración
oracion = input("Ingresa una oración: ")

# Convertir la oración a mayúsculas
oracion_mayusculas = oracion.upper()

# Contar cuántas veces aparece "python" en la oración
contador_python = oracion.count("python")

# Mostrar la oración sin espacios en blanco al inicio y al final
oracion_sin_espacios = oracion.strip()

print("Oración en mayúsculas:", oracion_mayusculas)
print("Cantidad de veces 'python' aparece:", contador_python)
print("Oración sin espacios en blanco:", oracion_sin_espacios)
```

# Actividad 4

```python
# Solicitar al usuario ingresar números separados por comas para cada conjunto
numeros1 = set(input("Ingresa números para el primer conjunto: ").split(","))
numeros2 = set(input("Ingresa números para el segundo conjunto: ").split(","))

# Verificar si los conjuntos tienen elementos en común
tienen_elementos_comunes = numeros1.isdisjoint(numeros2)

# Crear un conjunto con la unión de ambos conjuntos
union_conjuntos = numeros1.union(numeros2)

print("¿Que elementos tienen en común?", tienen_elementos_comunes)
print("Unión de conjuntos:", union_conjuntos)
```

Guía complementaria – Parte 3

# Operadores básicos, expresiones anidadas y controladores de flujo

CODERHOUSE

# Actividad 1
## Paresitos

Crea un programa que solicite dos números enteros al usuario y determine si ambos son números pares.

.

CODERHOUSE

# Actividad 2
## x 3, x 5

Escribe un programa que tome un número entero ingresado por el usuario y determine si es un número par, divisible por 3 y 5 al mismo tiempo, o ninguno de los anteriores.

.

# Actividad 3
## ¡Positivo! ¿Par sí o par no?

Crea un programa que solicite un número entero al usuario y determine si es un número negativo, positivo o igual a cero. En caso de ser positivo, verifica si es par o impar.

# Actividad 4
## Alineando la condicional

Escribe un programa que tome un número entero ingresado por el usuario y muestre "Es positivo" si el número es mayor que cero, de lo contrario, muestra "No es positivo".

# Resoluciones

# Ejemplos de solución

## Actividad 1

```python
numero1 = int(input("Ingresa el primer número: "))
numero2 = int(input("Ingresa el segundo número: "))

if numero1 % 2 == 0 and numero2 % 2 == 0:
    print("Ambos números son pares.")
else:
    print("Al menos uno de los números no es par.")
```

## Actividad 2

```python
numero = int(input("Ingresa un número: "))

if numero % 2 == 0:
    if numero % 3 == 0 and numero % 5 == 0:
        print("El número es par y divisible por 3 y 5.")
    else:
        print("El número es par pero no es divisible por 3 y 5.")
else:
    print("El número no es par.")
```

# Actividad 3

```python
numero = int(input("Ingresa un número: "))

if numero > 0:
    if numero % 2 == 0:
        print("El número es positivo y par.")
    else:
        print("El número es positivo e impar.")
elif numero < 0:
    print("El número es negativo.")
else:
    print("El número es igual a cero.")
```

# Actividad 4

```python
numero = int(input("Ingresa un número: "))
resultado = "Es positivo" if numero > 0 else "No es positivo"
print(resultado)
```

Guía complementaria – Parte 4

# Controladores de flujo II

# Actividad 1
## Hay tabla

Escribe un programa que tome un número entero positivo ingresado por el usuario y muestre la tabla de multiplicar de ese número. Repite la solicitud al usuario de ingresar un nuevo número hasta que ingrese un cero.

**Notas actividad 1**
Aclaraciones

En este ejercicio deberás utilizar 2 bucles anidados. Uno para todo y el otro para la generación de la tabla.

# Actividad 2
## Ubica la palabra

Escribe un programa que tome una lista de nombres ingresados por el usuario separados por un espacio y los liste mostrando su ubicacion.

**Notas actividad 2**
Aclaraciones

Recuerda usar el split para separar un string. También ten presente la funcionalidad enumerate.

# Actividad 3
# ¿Vocales perdidas?

Crea un programa que solicite al usuario ingresar una palabra. Luego, recorre la palabra y cuenta cuántas vocales contiene. Al final, si no se encontraron vocales, muestra un mensaje comunicando que la palabra ingresada solo contiene consonantes.

**Notas actividad 3**
Aclaraciones

Podemos utilizar el operador ‘in’ para corroborar si tiene vocales o no 😉

# Actividad 4
## Salteando palabras

Escribe un programa que tome una lista de palabras ingresadas por el usuario. Luego, que muestre cada palabra una por una. Si la palabra es "salir", finaliza el programa. Si la palabra es "omitir", se pasa a la siguiente iteración. Al final, si ninguna palabra fue "salir", muestra un mensaje avisando la situación.

**Notas actividad 4**
Aclaraciones

Recuerda usar slice para cortar un string.

# Actividad 5
# Inventario

Imagina que estás administrando un pequeño almacén y deseas realizar un seguimiento de los productos en tu inventario. Escribe un programa que te permita ingresar el nombre y la cantidad de varios productos. Utiliza un bucle para recorrer los productos y mostrar su nombre y cantidad. Al final, muestra el total de productos en el inventario.

# Resoluciones

# Ejemplos de solución

## Actividad 1

```
while True:
    numero = int(input("Ingresa un número positivo (o un número negativo para salir): "))
    if numero == 0:
        break
    for i in range(1, 11):
        resultado = numero * i
        print(f"{numero} x {i} = {resultado}")
```

## Actividad 2

```
nombres = input("Ingresa una lista de nombres separados por espacios: ").split()

for indice, nombre in enumerate(nombres):
    print(f"Nombre {indice + 1}: {nombre}")
```

# Actividad 3

```python
palabra = input("Ingresa una palabra: ")
vocales = "aeiouAEIOU"
contador_vocales = 0

for letra in palabra:
    if letra in vocales:
        contador_vocales += 1
else:
    if contador_vocales == 0:
        print("La palabra no contiene vocales.")
    else:
        print(f"La palabra contiene {contador_vocales} vocales.")
```

# Actividad 4

```python
palabras = input(
    "Ingresa una lista de palabras separadas por espacios: "
  ).split()

for palabra in palabras:
    if palabra == "salir":
        break
    elif palabra == "omitir":
        continue
    else:
        print(palabra)
else:
    print("No se encontró la palabra 'salir'.")
```

CODERHOUSE

# Actividad 5

```python
inventario = {}

num_productos = int(input("Ingresa la cantidad de productos en el inventario a cargar: "))

# el _ se suele utilizar par adefinir variables que no se van a utilizar como
# en este caso que no se utiliza su valor dentro del for
for _ in range(num_productos):
    nombre = input("Ingresa el nombre del producto: ")
    cantidad = int(input(f"Ingresa la cantidad de {nombre}: "))
    inventario[nombre] = cantidad

print("\nInventario:")
total_productos = 0

for producto, cantidad in inventario.items():
    print(f"{producto}: {cantidad}")
    total_productos += cantidad

print("\nTotal de productos en el inventario:", total_productos)
```

Guía complementaria - Parte 5

# Funciones

# Actividad 1
## Gestión de tareas pendientes

Crea un programa que permita a un usuario llevar un registro de tareas pendientes. El programa debe:

- Permitir al usuario agregar tareas.
- Marcar tareas como completadas. agregámdole un tilde o algo que identifique que se completó al principio de la tarea.
- Listar las tareas pendientes.

**Notas actividad 1**
Aclaraciones

Utiliza una lista y funciones separadas para gestionar las tareas.

# Actividad 2
# Contactame

Desarrolla un programa que permita a un usuario registrar información de contactos (nombre, número de teléfono y correo electrónico). El programa debe almacenar estos contactos y permitir al usuario buscar contactos por nombre o número de teléfono.

**Notas actividad 2**
Aclaraciones

Utiliza un diccionario para gestionar los contactos y funciones separadas para realizar estas acciones.

CODERHOUSE

# Actividad 3
## Red de computadora

Crea un programa que permita a un usuario configurar la red de una computadora. El programa debe aceptar argumentos clave para configurar la dirección IP, la máscara de subred y la puerta de enlace. Luego, muestra la configuración de red completa.

**Notas actividad 3**
Aclaraciones

Utiliza kwargs como parámetro de la función permitiendote pasar cualquier llave-valor a la función.

CODERHOUSE

# Actividad 4
# Tu promedio

Crea un programa que permita calcular el promedio de un número variable de notas ingresadas por el usuario. La función calcular_promedio puede recibir un número variable de notas.
Luego, muestra el promedio de las notas ingresadas.

**Notas actividad 4**
Aclaraciones

Deberán utilizar *args como parametro para que se permita el número variado de notas.

CODERHOUSE

# Resoluciones

# Ejemplos de solución

## Actividad 1

```python
def agregar_tarea(tareas, tarea):
    # Agrega una nueva tarea a la lista "tareas".
    tareas.append(tarea)

def marcar_completada(tareas, indice):
    # Marca una tarea como completada agregando "✓" al principio.
    if 0 <= indice < len(tareas):
        tareas[indice] = "✓ " + tareas[indice]

def listar_tareas(tareas):
    # Muestra las tareas pendientes numeradas.
    for i, tarea in enumerate(tareas):
        print(f"{i + 1}. {tarea}")

tareas_pendientes = []  # Lista para almacenar tareas pendientes.
```

```python
while True:
    print("\n1. Agregar tarea")
    print("2. Marcar tarea como completada")
    print("3. Listar tareas pendientes")
    print("4. Salir")
    opcion = input("Selecciona una opción: ")

    if opcion == '1':
        tarea = input("Ingresa la tarea pendiente: ")
        agregar_tarea(tareas_pendientes, tarea)
        print("Tarea agregada.")
    elif opcion == '2':
        listar_tareas(tareas_pendientes)
        indice = int(input("Ingresa el número de la tarea completada: ")) - 1
        marcar_completada(tareas_pendientes, indice)
        print("Tarea marcada como completada.")
    elif opcion == '3':
        listar_tareas(tareas_pendientes)
    elif opcion == '4':
        break
```

# Actividad 2

```python
def agregar_contacto(agenda, nombre, telefono, correo):
    # Agrega un nuevo contacto al diccionario "agenda".
    agenda[nombre] = {'Teléfono': telefono, 'Correo': correo}

def buscar_contacto(agenda, criterio):
    # Busca y muestra contactos por nombre o número de teléfono.
    for nombre, info in agenda.items():
        if criterio in nombre or criterio == info['Teléfono']:
            print(f"Nombre: {nombre}")
            print(f"Teléfono: {info['Teléfono']}")
            print(f"Correo: {info['Correo']}")

agenda = {}  # Diccionario para almacenar contactos.

while True:
    print("\n1. Agregar contacto")
    print("2. Buscar contacto")
    print("3. Salir")
    opcion = input("Selecciona una opción: ")

    if opcion == '1':
        nombre = input("Ingresa el nombre del contacto: ")
        telefono = input("Ingresa el número de teléfono: ")
        correo = input("Ingresa el correo electrónico: ")
        agregar_contacto(agenda, nombre, telefono, correo)
        print("Contacto agregado.")
    elif opcion == '2':
        criterio = input("Ingresa el nombre o número de teléfono a buscar: ")
        buscar_contacto(agenda, criterio)
    elif opcion == '3':
        break
```

# Actividad 3

```python
def configurar_red(**kwargs):
    print("\nConfiguración de Red:")
    for clave, valor in kwargs.items():
        print(f"{clave}: {valor}")

while True:
    print("\n1. Configurar Red")
    print("2. Salir")
    opcion = input("Selecciona una opción: ")

    if opcion == '1':
        ip = input("Ingresa la dirección IP: ")
        mascara = input("Ingresa la máscara de subred: ")
        puerta_enlace = input("Ingresa la puerta de enlace: ")
        configurar_red(IP=ip, Máscara=mascara, Puerta_enlace=puerta_enlace)
    elif opcion == '2':
        break
```

CODERHOUSE

# Actividad 4

```python
def calcular_promedio(*notas):
    total = sum(notas)
    promedio = total / len(notas)
    return promedio

print("Calculadora de Promedio de Notas")
notas = []

while True:
    nota = float(input("Ingresa una nota (o 0 para calcular el promedio): "))
    if nota == 0:
        break
    notas.append(nota)

if notas:
    promedio = calcular_promedio(*notas)
    print(f"El promedio de las notas ingresadas es: {promedio:.2f}")
else:
    print("No se ingresaron notas.")
```

Guía complementaria - Parte 6

# Excepciones y POO

# Actividad 1
## Búsqueda del error

Desarrolla un programa que permita al usuario buscar información sobre ciudades. Tendrás un diccionario llamado ciudades_info que contiene información sobre algunas ciudades, como su país, población y puntos de interés. El programa debe permitir al usuario ingresar el nombre de una ciudad y mostrar la información correspondiente. El programa debe poder manejar el caso en el que la ciudad no existe en el diccionario y mostrando un mensaje avisando del error.

```
ciudades_info = {
    'Paris': {
        'Pais': 'Francia',
        'Poblacion': 2187526,
        'Puntos_de_Interes': ['Torre Eiffel',
'Louvre', 'Catedral de Notre-Dame']
    },
    'Nueva York': {
        'Pais': 'Estados Unidos',
        'Poblacion': 8398748,
        'Puntos_de_Interes': ['Estatua de la
Libertad', 'Central Park', 'Times Square']
    },
    'Tokio': {
        'Pais': 'Japón',
        'Poblacion': 13929286,
        'Puntos_de_Interes': ['Torre de Tokio',
'Templo Senso-ji', 'Palacio Imperial']
    }
}
```

# Actividad 2
# Edad correcta

Escribe un programa que permita al usuario ingresar su edad. El programa debe validar si la edad ingresada está dentro del rango de 18 a 65 años, y mostrar un mensaje correspondiente. Utiliza un bloque try-except con múltiples bloques except para manejar posibles errores relacionados con la entrada del usuario, como una entrada no numérica o una edad fuera del rango válido.
Tip: puedes usar la funcionalidad 'raise' para que se genere una excepción.

**Notas actividad 2**
Aclaraciones

La función raise se utiliza para generar manualmente una excepción en Python. Puedes especificar el tipo de excepción que deseas generar y opcionalmente proporcionar un mensaje de error personalizado. Por ejemplo:

```
# Generar una excepción ValueError con un mensaje personalizado
raise ValueError("Este es un mensaje de error personalizado.")
```

En el ejemplo anterior, se genera una excepción ValueError con el mensaje "Este es un mensaje de error personalizado." Esta funcionalidad puede ser útil cuando deseas controlar el flujo de tu programa y generar excepciones específicas en función de condiciones personalizadas.

# Actividad 3
## Estamos de oferta

Escribe un programa que permita al usuario ingresar el precio de un producto y un código de descuento. El programa debe validar si el precio es un número positivo y si el código de descuento es válido. Los errores posibles incluyen entradas no numéricas, números negativos y códigos de descuento no válidos.

Los codigos de descuento son:
descuentos_validos = ["DESC10", "DESC20", "DESC30"]

**Notas actividad 3**
Aclaraciones

Utiliza un bloque try-except con varios bloques except para manejar estos errores y mostrar mensajes de error específicos en cada caso. Ten en cuenta el uso de la instrucción raise comentada en la actividad anterior de ser necesario su uso.

# Resoluciones

# Ejemplos de solución

## Actividad 1

```python
ciudades_info = {
    'Paris': {
        'Pais': 'Francia',
        'Poblacion': 2187526,
        'Puntos_de_Interes': ['Torre Eiffel', 'Louvre', 'Catedral de Notre-Dame']
    },
    'Nueva York': {
        'Pais': 'Estados Unidos',
        'Poblacion': 8398748,
        'Puntos_de_Interes': ['Estatua de la Libertad', 'Central Park', 'Times Square']
    },
    'Tokio': {
        'Pais': 'Japón',
        'Poblacion': 13929286,
        'Puntos_de_Interes': ['Torre de Tokio', 'Templo Senso-ji', 'Palacio Imperial']
    }
}

ciudad = input("Ingresa el nombre de una ciudad: ")

try:
    informacion = ciudades_info[ciudad]
    print(f"Información sobre {ciudad}:")
    print(f"País: {informacion['Pais']}")
    print(f"Población: {informacion['Poblacion']}")
    print(f"Puntos de Interés: {', '.join(informacion['Puntos_de_Interes'])}")
except KeyError:
    print(f"Error: La ciudad '{ciudad}' no está en la base de datos.")
```

## Actividad 2

```python
try:
    numero = float(input("Ingresa un número positivo: "))

    if numero <= 0:
        raise ValueError()
    else:
        print(f"Has ingresado el número {numero}.")

except TypeError:
    print("Error: La entrada no es un número.")
except ValueError:
    print("Error: Ingresa un número válido.")
except:
    print("Error desconocido.")
```

CODERHOUSE

# Actividad 3

```python
try:
    precio = float(input("Ingresa el precio del producto: "))
    codigo_descuento = input("Ingresa el código de descuento: ")

    if precio <= 0:
        raise ValueError("El precio debe ser un número positivo.")

    descuentos_validos = ["DESC10", "DESC20", "DESC30"]

    if codigo_descuento not in descuentos_validos:
        raise ValueError("Código de descuento no válido.")

    if codigo_descuento == "DESC10":
        precio_descuento = precio * 0.9
    elif codigo_descuento == "DESC20":
        precio_descuento = precio * 0.8
    else:
        precio_descuento = precio * 0.7

    print(f"El precio con descuento es: {precio_descuento}")

except ValueError as ve:
    print(f"Error de Valor: {ve}")
except TypeError:
    print("Error: La entrada no es válida.")
except Exception as e:
    print(f"Error desconocido: {e}")
```

CODERHOUSE

Guía complementaria – Parte 7

# POO II y Herencia

CODERHOUSE

# Actividad 1
## Figuras

Define una clase Figura con un método de instancia area que devuelve el área de la figura. Luego, crea clases hijas como Circulo y Rectangulo que hereden de Figura y proporcionen implementaciones diferentes del método area.

# Actividad 2
## Calculadora POO

Crea una clase Calculadora con un método de clase llamado suma que acepte dos números como argumentos y devuelva la suma de ellos. Luego, utiliza este método para realizar algunas operaciones de suma.

# Actividad 3
## Gestión de empleados

Crea una clase Empleado que tenga los siguientes atributos de instancia: nombre, apellido, edad, salario. Luego, crea una clase Gerente que herede de Empleado y tenga un atributo adicional departamento. Define métodos de instancia para ambas clases, como mostrar_informacion para mostrar los detalles de un empleado y aumentar_salario para aumentar el salario de un empleado en un porcentaje dado. Crea instancias de ambas clases y realiza algunas operaciones.

**Notas actividad 3**
Aclaraciones

Recuerda utilizar el super para ejecutar el init de una clase padre.

# Actividad 4
# La biblioteca

Crea una clase Libro con atributos de instancia como titulo, autor, año_publicacion, y disponible (un valor booleano que indica si el libro está disponible o no). Luego, crea una clase Biblioteca que tenga una lista de libros y métodos de instancia para prestar un libro, devolver un libro y mostrar todos los libros disponibles. Utiliza atributos de clase para registrar la cantidad total de libros en la biblioteca. Crea instancias de ambas clases y realiza operaciones de préstamo y devolución de libros.

**Notas actividad 4**
Aclaraciones

Para modificar un atributo de clase podes hacer lo siguiente:

Casa.atributo_de_clase = 'prueba'

En este ejemplo se toma Casa como la clase y el atributo atributo_de_clase, donde se guarda un string (esto puede hacerse desde los metodos.)

# Actividad 5
# Tienda en línea

Crea una clase Producto con atributos de instancia como nombre, precio, cantidad_disponible y codigo_producto. Luego, crea una clase CarritoCompras que permita a los clientes agregar productos, eliminar productos y calcular el total de la compra. Utiliza un atributo de clase para rastrear la cantidad total de productos en el carrito de compras de todos los clientes. Crea instancias de ambas clases y simula operaciones de compra.

# Resoluciones

# Ejemplos de solución

## Actividad 1

```python
class Figura:
    def area(self):
        pass

class Circulo(Figura):
    def __init__(self, radio):
        self.radio = radio

    def area(self):
        return 3.1416 * self.radio * self.radio

class Rectangulo(Figura):
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def area(self):
        return self.base * self.altura

# Ejemplo de uso:
circulo = Circulo(5)
rectangulo = Rectangulo(4, 6)
print("Área del círculo:", circulo.area())
print("Área del rectángulo:", rectangulo.area())
```

## Actividad 2

```python
class Calculadora:
    @classmethod
    def suma(cls, num1, num2):
        return num1 + num2

# Ejemplo de uso:
resultado = Calculadora.suma(5, 3)
print("Resultado de la suma:", resultado)
```

# Actividad 3

```python
class Empleado:
    def __init__(self, nombre, apellido, edad, salario):
        self.nombre = nombre
        self.apellido = apellido
        self.edad = edad
        self.salario = salario

    def mostrar_informacion(self):
        return f"Nombre: {self.nombre} {self.apellido}, Edad: {self.edad}, Salario: ${self.salario}"

    def aumentar_salario(self, porcentaje):
        self.salario += self.salario * (porcentaje / 100)

class Gerente(Empleado):
    def __init__(self, nombre, apellido, edad, salario, departamento):
        super().__init__(nombre, apellido, edad, salario)
        self.departamento = departamento

    def mostrar_informacion(self):
        return super().mostrar_informacion() + f", Departamento: {self.departamento}"

# Ejemplo de uso:
empleado1 = Empleado("Juan", "Pérez", 30, 40000)
gerente1 = Gerente("Ana", "Gómez", 35, 60000, "Ventas")

print(empleado1.mostrar_informacion())
print(gerente1.mostrar_informacion())

empleado1.aumentar_salario(10)
print(f"Nuevo salario de {empleado1.nombre}: ${empleado1.salario}")
```

# Actividad 4

```python
class Libro:
    def __init__(self, titulo, autor, año_publicacion):
        self.titulo = titulo
        self.autor = autor
        self.año_publicacion = año_publicacion
        self.disponible = True

class Biblioteca:
    cantidad_total_libros = 0

    def __init__(self):
        self.libros = []

    def agregar_libro(self, libro):
        self.libros.append(libro)
        Biblioteca.cantidad_total_libros += 1

    def prestar_libro(self, titulo):
        for libro in self.libros:
            if libro.titulo == titulo and libro.disponible:
                libro.disponible = False
                return f"El libro '{titulo}' ha sido prestado."
        return f"El libro '{titulo}' no está disponible."

    def devolver_libro(self, titulo):
        for libro in self.libros:
            if libro.titulo == titulo and not libro.disponible:
                libro.disponible = True
                return f"El libro '{titulo}' ha sido devuelto."
        return f"El libro '{titulo}' no puede ser devuelto."

    def mostrar_libros_disponibles(self):
        disponibles = [libro.titulo for libro in self.libros if libro.disponible]
        return f"Libros disponibles: {', '.join(disponibles)}"
```

```python
# Ejemplo de uso:
libro1 = Libro("El Gran Gatsby", "F. Scott Fitzgerald", 1925)
libro2 = Libro("Cien Años de Soledad", "Gabriel García Márquez", 1967)

biblioteca = Biblioteca()
biblioteca.agregar_libro(libro1)
biblioteca.agregar_libro(libro2)

print(biblioteca.prestar_libro("El Gran Gatsby"))
print(biblioteca.devolver_libro("El Gran Gatsby"))
print(biblioteca.mostrar_libros_disponibles())
```

# Actividad 5

```python
class Producto:
    def __init__(self, nombre, precio, cantidad_disponible, codigo_producto):
        self.nombre = nombre
        self.precio = precio
        self.cantidad_disponible = cantidad_disponible
        self.codigo_producto = codigo_producto

class CarritoCompras:
    cantidad_total_productos = 0

    def __init__(self):
        self.productos = []

    def agregar_producto(self, producto):
        self.productos.append(producto)
        CarritoCompras.cantidad_total_productos += 1

    def eliminar_producto(self, codigo_producto):
        for producto in self.productos:
            if producto.codigo_producto == codigo_producto:
                self.productos.remove(producto)
                CarritoCompras.cantidad_total_productos -= 1
                return f"Producto '{producto.nombre}' eliminado del carrito."
        return f"Producto con código '{codigo_producto}' no encontrado en el carrito."

    def calcular_total_compra(self):
        total = sum([producto.precio for producto in self.productos])
        return f"Total de compra: ${total}"
```

```python
# Ejemplo de uso:
producto1 = Producto("Laptop", 800, 5, "LP001")
producto2 = Producto("Teléfono", 300, 10, "PH002")

carrito = CarritoCompras()
carrito.agregar_producto(producto1)
carrito.agregar_producto(producto2)

print(carrito.eliminar_producto("PH002"))
print(carrito.calcular_total_compra())
```

CODERHOUSE

Guía complementaria – Parte 8

# Scripts, módulos, paquetes y manejo de archivos

CODERHOUSE

# Actividad 1
## Registrando gastos

Escribe un programa en Python que permita a un usuario registrar sus gastos diarios en un archivo de texto llamado "gastos.txt". El programa debe permitir al usuario ingresar la descripción del gasto y la cantidad gastada. Luego, debe guardar estos datos en el archivo en el siguiente formato:

"Fecha: {fecha} - Descripción: {descripción} - Cantidad: {cantidad}"

Donde fecha es la fecha actual y descripción y cantidad son los datos ingresados por el usuario.

# Actividad 2
## Gestión de tareas pendientes (Con persistencia de datos)

Basándonos en el ejercicio 1 de funciones, edita el código para que
el programa guarde el listado de tareas en un json al terminar la ejecución del programa y
lo recupere al iniciarse el mismo.

**Notas actividad 2**
Aclaraciones

Ten en cuenta que el modo read del open genera un error al querer abrir un archivo que no existe.
Y, a diferencia del read en los txt, el json.load también genera un error si el archivo no tiene nada.

# Actividad 3
## Registro de calificaciones

Escribe un programa que permita a un profesor registrar las calificaciones de sus estudiantes en un archivo json llamado "calificaciones.json".
El programa debe permitir al profesor ingresar nombres de estudiantes y sus calificaciones. Luego, debe guardar estos datos en el archivo cuando termine el programa para
persistir estos datos para futuras ejecuciones del programa. Utilizar los nombres de los alumnos como claves y las notas como valores.

**Notas actividad 3**
Aclaraciones

Ten en cuenta que el modo read del open genera un error al querer abrir un archivo que no existe.
Y, a diferencia del read en los txt, el json.load también genera un error si el archivo no tiene nada.

# Actividad 4
# Metiendo a la bolsa

Con las funciones de los ejercicios anteriores, sepáralas en 2 o más módulos y
genera un paquete que incluya estos módulos.

**Notas actividad 4**
Aclaraciones

Recuerda que en los módulos deben ir solo código que sea de definición y, en caso de
querer agregar código de ejecución en alguno de los módulos, debe agregarse dentro de la siguiente condicional

```
if __name__ == '__main__':
```

# Resoluciones

# Ejemplos de solución

## Actividad 1

```python
import datetime

# Función para registrar gastos
def registrar_gasto():
    fecha = datetime.date.today()
    descripcion = input("Ingrese la descripción del gasto: ")
    cantidad = input("Ingrese la cantidad gastada: ")

    with open("gastos.txt", "a") as archivo:
        archivo.write(f"Fecha: {fecha} - Descripción: {descripcion} - Cantidad: {cantidad}\n")

# Llamada a la función para registrar un gasto
registrar_gasto()
```

# Actividad 2

```python
import json

def agregar_tarea(tareas, tarea):
    # Agrega una nueva tarea a la lista "tareas".
    tareas.append(tarea)

def marcar_completada(tareas, indice):
    # Marca una tarea como completada agregando "✓" al principio.
    if 0 ≤ indice < len(tareas):
        tareas[indice] = "✓ " + tareas[indice]

def listar_tareas(tareas):
    # Muestra las tareas pendientes numeradas.
    for i, tarea in enumerate(tareas):
        print(f"{i + 1}. {tarea}")

try:
    with open('tareas_pendientes.json', 'r') as archivo:
        tareas_pendientes = json.load(archivo)
except Exception as err:
    tareas_pendientes = []  # Lista para almacenar tareas pendientes.
```

```python
while True:
    print("\n1. Agregar tarea")
    print("2. Marcar tarea como completada")
    print("3. Listar tareas pendientes")
    print("4. Salir")
    opcion = input("Selecciona una opción: ")

    if opcion == '1':
        tarea = input("Ingresa la tarea pendiente: ")
        agregar_tarea(tareas_pendientes, tarea)
        print("Tarea agregada.")
    elif opcion == '2':
        listar_tareas(tareas_pendientes)
        indice = int(input("Ingresa el número de la tarea completada: ")) - 1
        marcar_completada(tareas_pendientes, indice)
        print("Tarea marcada como completada.")
    elif opcion == '3':
        listar_tareas(tareas_pendientes)
    elif opcion == '4':
        break

with open('tareas_pendientes.json', 'w') as archivo:
    json.dump(tareas_pendientes, archivo, indent=4)
```

# Actividad 3

```python
import json

# Función para registrar calificaciones
def registrar_calificacion(calificaciones):
    nombre_estudiante = input("Ingrese el nombre del estudiante: ")
    calificacion = input("Ingrese la calificación del estudiante: ")
    calificaciones[nombre_estudiante] = calificacion

try:
    with open('calificaciones.json', 'r') as archivo:
        calificaciones = json.load(archivo)
except Exception as err:
    calificaciones = {}


while True:
    print("\n1. Agregar alumno y calificacion")
    print("2. Salir")
    opcion = input("Selecciona una opción: ")      You, last week • Se

    if opcion == '1':
        registrar_calificacion(calificaciones)
        print("Calificacion agregada.")
    elif opcion == '2':
        break

with open('calificaciones.json', 'w') as archivo:
    json.dump(calificaciones, archivo, indent=4)
```

# Actividad 4

```python
# Pasos:
#     1. generar los modulos con las funciones que queremos que contengan
#     2. crear una carpeta que sera nuestro paquete
#     3. crear el archivo __init__.py
#     4. en el mismo lugar donde esta el paquete crear un archivo setup.py
#     5. completar el setup.py con los datos vistos en las clases
#     6. ejecutar con python desde la consola el archivo setup.py y pasarle como argumento sdist
#     7. Luego de estos pasos deberia haberse generado una carpeta dist en la cual se va a encontrar el paquete que creamos
```

CODERHOUSE

Guía complementaria – Parte 9

# Git, GitHub y Django

CODERHOUSE

# Actividad 1
## Repasemos Git

Sigamos las siguientes instrucciones para practicar un poco el uso de git...

**1.** Crea una carpeta nueva y abrirla para trabajar en ella.
**2.** Inicializa un repositorio de git.
**3.** Crea una rama por cada parte de la guía, hasta la parte 8 inclusive, de ejercicios en esta guía (parte1, parte2, parte3, etc).

**4.** Por cada rama de las partes, seguir los siguientes pasos:

A. Crear una carpeta con el nombre de la rama.
B. Dentro de la carpeta crear para la actividad 1 un archivo que contenga la resolución y luego hacer un commit con el mensaje “Actividad 1”.
C. Repetir el punto anterior para las actividades restantes.

**5.** Luego, volver a la rama principal (master o main, depende como lo tengan configurado) y hacer un merge por cada una de las ramas de las partes.

**6.** Por último, hacer un push a un repositorio creado en github.

Con esto dejaremos todo lo que tenía cada rama (cada carpeta de actividades) en la rama principal.

PARA RECORDAR

# Contenido destacado

A partir de la siguiente actividad, cada parte tiene su propia rama con las actividades resueltas. Esto se maneja de esta forma, debido a que haremos un proyecto de django repasando todo lo visto en clases y con el manejo de ramas se puede ir viendo el progreso escalonado delimitado, por lo que van solicitando las actividades.

# Actividad 2
## Repaso creación de proyecto

Crea un proyecto de Django siguiendo los pasos vistos en clase.

Este proyecto deberá contar con 2 vistas:

**1.** La primera deberá enviar por HttpResponse un string que indique en qué año aproximadamente nació una persona si le pasamos por la url la edad (para revisar si un string es solo numérico se puede usar el método .isnumeric() de los strings que devuelve True en caso de que sean solo números).

**2.** La segunda solo requerirá que en lugar de pasarle un string como argumento al HttpResponse se le pase un template en el cual se mostrará un mensaje que diga "Bienvenidos".
La url de acceso deberá ser 'bienvenida/'.

# Resoluciones

# Ejemplos de solución

## Actividad 1

Esta actividad puede ser guiada revisando los commits de la rama master o pasándose a las diferentes ramas de las partes específicas. Esto es debido a que se siguió los mismos pasos para armar este repositorio así lo tienen como guía.

## Actividad 2

Punto 1 : Disponible aquí

Punto 2 : Disponible aquí

Guía complementaria - Parte 10

# Portfolio y Playground intermedio parte I

CODERHOUSE

# Actividad 1
## Trabajemos con apps

A partir de este punto las actividades siguientes se centrarán en la modificación y mejora del proyecto que se arrancó en la actividad 2 de la ejercitación de Git, GitHub y Django.

Teniendo esto en cuenta, realiza las siguientes modificaciones:

**1.** Crea una app 'inicio' a la cual le trasladaremos las vistas creadas en la actividad pasada. Recuerda modificar el setting.py para que nuestro proyecto reconozca la app creada. (una vez trasladadas las vistas eliminar el archivo views.py que quedo vacío)

**2.** Genera el archivo urls.py para nuestra app y agrega el código correspondiente en el urls.py que se encuentra con el settings.py para que se conecte al urls.py de nuestra app (uso de la funcionalidad "include" ).

**3.** Cambia la carga actual de templates en las vistas por el uso de render (el shortcut). Cambia la ubicación de los templates para que estén dentro de la app que le corresponda.

# Actividad 2
## Usemos un template predefinido

Busca en **starbootstrap** un template que te guste, descárgalo e impleméntalo (con los cambios que sientas necesario en el HTML) en el proyecto en una vista llamada inicio dentro de la app inicio.
Para esta vista define el path de la url vacío.

# Actividad 3
## Guardo y muestro

**1.** Crear un modelo que cuente con 3 atributos: nombre(charfield), edad(integerfield) y fecha(datefield). Para este modelo agrega una vista que tome por url 2 parámetros, uno para el charfield, otro para el integerfield y el datefield lo rellenaremos con la fecha del momento cuando se crea. Teniendo estos datos en la vista, genera un objeto para guardarlo en la base de datos y, también, para pasarlo por contexto a un template que crearemos.
**2.** El template del punto anterior deberá mostrar nombre y edad. Además, en caso que el dia del campo fecha sea mayor a 15 se deberá mostrar en un listado cada letra del nombre.

**Recordatorio**: Una vez creado un modelo ejecutar el comando makemigrations y luego migrate para que se plasme la creación del modelo en la bd.

# Resoluciones

# Ejemplos de solución

## Actividad 1

- Parte 1: Disponible aquí
- Parte 2: Disponible aquí
- Parte 3: Disponible aquí

## Actividad 2

Disponible aquí

## Actividad 3

- Parte 1: Disponible aquí
- Parte 2: Disponible aquí

Guía complementaria - Parte 11

# Playground Intermedio parte II

CODERHOUSE

# Actividad 1
## Figuras formularios y listado

Crea un nuevo modelo Paleta con 4 atributos: marca(charfield), modelo(charfield), anio(integerfield), nueva(booleanfield). El modelo deberá tener una vista para el formulario de creación y otra para el listado de paletas creadas (esta última deberá incluir un formulario de búsqueda).

**Aviso**: Los formularios de creación y búsqueda también deben crearse en esta actividad.

# Actividad 2
## Mejora de templates y panel de admin

Acomoda los templates para que implementen herencia, mitigando la repetición de código. Además, acomoda en la barra de navegación el acceso a las vistas de 'inicio', 'crear_paleta' y 'buscar_paleta'.

CODERHOUSE

# Actividad 3
## Apartado admin

Registra los modelos en el apartado de admin. Luego accede al mismo y prueba crear, modificar, ver, eliminar paletas.
Agrega en los modelos registrados el metodo magico `__str__` para que el listado del admin sea más legible.

# Ejemplos de solución

## Actividad 1

- Disponible aquí

## Actividad 2

- Disponible aquí

## Actividad 3

- Disponible aquí

Guía complementaria – Parte 12

# Playground avanzado parte I

# Actividad 1
## Edita y borra

Al modelo paleta agrégale una vista para la actualizar datos y otra para eliminar paletas. (No utilizar Clases Basadas en Vistas)
En este punto, modifica el nav de la app para que a la vista de creación se acceda desde la vista de búsqueda/listado y el acceso a actualizar o borrar estén en botones junto a cada paleta del listado.

**IMPORTANTE**: Para no requerir validar en la vista de qué tipo tiene que ser el parámetro que se pasa por la url podemos en la url cuando definimos el nombre del parámetro, ejemplo `paletas/editar/<paleta_id>/`, indícale de qué tipo queremos que nuestra app detecte que va a ser, ejemplo `paletas/editar/<int:paleta_id>/`.

# Actividad 2
## CBV

Crea un nuevo modelo PelucheAnimal con 4 atributos: animal(charfield), altura(floatfield), fecha(datefield). El modelo deberá contar con una CBV para cada una de las siguientes funcionalidades: crear, listar (incluir el buscador), editar, eliminar y mostrar más detalles del mismo.
Recuerda registrar el modelo en el apartado de admin.

**IMPORTANTE**: En la CBV del listado se deben encontrar los accesos a todas las demás vistas, esta debe ser la única accesible desde la barra de navegación de la pagina.

**EXTRA**: Puedes crear otra app llamada peluche y aca contener todo lo relacionado a PelucheAnimal (y tal vez a futuro sobre peluches en general).

# Actividad 3
# Autenticación

Agrégale al proyecto una nueva app llamada cuentas y luego:

**1.** Agrega a cuentas un vista para el login (usar un formulario custom que pida el mail además de el usuario y la contraseña) y una vista para el logout.
**2.** Ahora, una vista para que un usuario pueda registrarse en nuestra app.
**3.** Agrega a las vistas de edición y borrado de los modelos creados hasta el momento (Paleta y PelucheAnimal) el decorador o mixin, según corresponda, para limitar el acceso estas funcionalidades a personas que no están logueadas.

# Resoluciones

# Ejemplos de solución

## Actividad 1

Disponible aquí

## Actividad 2

Disponible aquí

## Actividad 3

- Parte 1: Disponible aquí
- Parte 2: Disponible aquí
- Parte 3: Disponible aquí

Guía complementaria - Parte 13

# Playground avanzado parte II y próximos pasos en el mundo de la programación

CODERHOUSE

# Actividad 1
## Más datos

Teniendo en cuenta que pocos son los datos que podemos guardar de un usuario, crea un modelo que esté relacionado con User y permita guardar un avatar y algún dato extra sobre el usuario (ej, pagina, biografía, fecha de nacimiento, etc).

# Actividad 2
## El perfil

Agrega un apartado donde el usuario pueda ver su información (nombre, apellido, avatar, etc) y que tenga un acceso a un apartado para modificar dicha información (también que se pueda modificar la contraseña).

# Actividad 3
## Ver y describir el peluche

Agrégale al peluche animal un atributo para cargarle una imagen y una descripción, para esta última permitir que sea con formato de texto enriquecido. Ambos campos deben ser incluidos en todas las vistas relacionadas a peluche animal.

# Resoluciones

# Ejemplos de solución

## Actividad 1

- Disponible aquí

## Actividad 2

- Disponible aquí

## Actividad 3

- Disponible aquí