Proyecto de fundamentos de la programación

ROTTGER EMMANUEL PARRAGUEZ LEANDRO
Escuela de Ciencias de la computación
UNI
20230250C

LUIS ANTONIO HEREDIA BRINGAS
Escuela de Fisica
UNI
20230452E

VALENTINO JAIR PEREZ MELO
Escuela de Ingenieria Fisica.
UNI
20234126E

2024-06-26

1. Funciones en Python

Las funciones en Python son bloques de código que realizan tareas específicas y pueden ser reutilizadas en diferentes partes de un programa. Se definen con la palabra clave **def** seguida del nombre de la función y paréntesis **()**. Pueden tener parámetros para recibir datos de entrada y pueden devolver resultados utilizando la palabra clave **return**. Las variables definidas dentro de una función son locales, mientras que las variables definidas fuera de una función son globales. Es una buena práctica documentar las funciones utilizando docstrings para describir su propósito, parámetros y valores de retorno. Las funciones en Python son ciudadanos de primera clase, lo que significa que pueden ser tratadas como cualquier otro objeto, permitiendo su uso en expresiones, asignaciones y como argumentos de otras funciones. Además, Python admite la recursión, lo que permite a una función llamarse a sí misma, lo que es útil para resolver problemas que pueden dividirse en casos más pequeños. En resumen, las funciones son componentes esenciales para organizar y estructurar el código de manera modular, promoviendo la reutilización y la claridad del código.

Ejercicio: Crear una función que calcule el factorial de un número.

```python
# Definición de la función factorial
def factorial(numero):
    # Caso base: si el número es 0, el factorial es 1
    if numero == 0:
        return 1
    else:
        # Llamada recursiva: calcula el factorial del número restando 1
        # y multiplicándolo por el factorial del número anterior
        return numero * factorial(numero - 1)

# Solicitar al usuario que ingrese un número para calcular su factorial
numero = int(input("Digite un numero para hallar su factorial: "))

# Calcular el factorial del número ingresado llamando a la función factorial
resultado = factorial(numero)

# Imprimir el resultado del cálculo del factorial
print("El factorial de", numero, "es:", resultado)
```

Codigo:

Ejecución:

```
Digite un numero para hallar su factorial: 4
El factorial de 4 es: 24
```

2. Cadenas en Python

una cadena (o string) es una secuencia de caracteres. Los caracteres pueden ser letras, números, símbolos de puntuación o incluso espacios en blanco. Las cadenas en Python se pueden definir utilizando comillas simples (') o dobles (").

Ejercicio: Escribir una función que invierta una cadena.

```python
# Definición de la función para invertir una cadena
def invertirCadena(cadena):
    # cadena[::-1] devuelve la cadena desde el último carácter hasta el primero, invirtiéndola.
    cadena_invertida = cadena[::-1]
    # Devolvemos la cadena invertida
    return cadena_invertida

# Solicitar al usuario que ingrese una cadena de texto
cadena = input("Ingrese la cadena a invertir: ")

# Llamar a la función para invertir la cadena
cadena_invertida = invertirCadena(cadena)

# Imprimir la cadena original y la cadena invertida
print("la cadena original: ", cadena)
print("la cadena invertida: ", cadena_invertida)
```

Codigo:

Ejecución:

```
Ingrese la cadena a invertir: Hola Mundo!
la cadena original:  Hola Mundo!
la cadena invertida:  !odnuM aloH

***Repl Closed***
```

3. Referencia y asignación dinámica en Python

las referencias y la asignación dinámica son conceptos fundamentales. Las referencias implican que una variable no almacena directamente un valor, sino una referencia a un objeto en memoria, lo que permite que múltiples variables apunten al mismo objeto. La asignación dinámica significa que las variables pueden contener diferentes tipos de valores en diferentes momentos sin necesidad de especificar un tipo de datos. Esto proporciona flexibilidad y conveniencia durante el desarrollo, ya que las variables pueden cambiar de tipo según sea necesario.

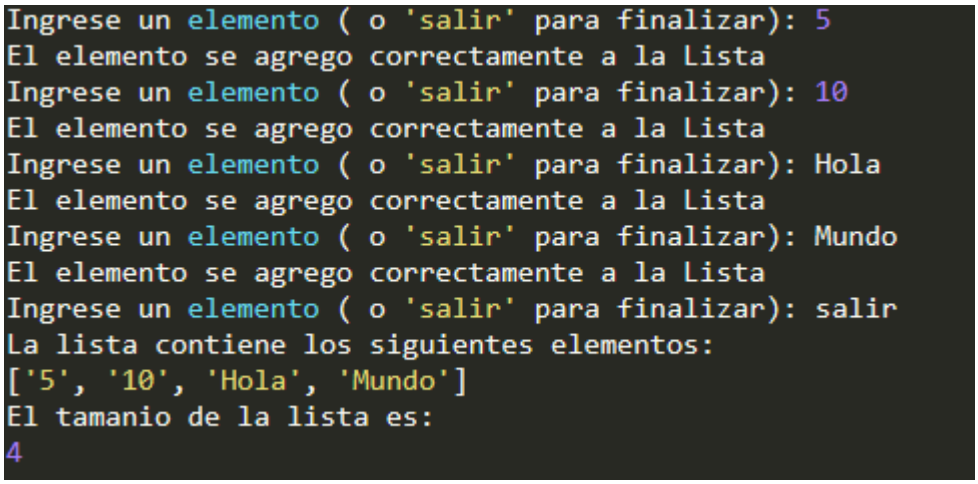
Listas: Una lista en Python es una estructura de datos que se utiliza para almacenar una colección ordenada de elementos. Es una de las estructuras de datos más versátiles y utilizadas en Python debido a su flexibilidad y capacidad para manejar diferentes tipos de datos.

Ejercicio: Crear una lista dinámica (usando listas en Python) y añadir elementos de forma interactiva.

```python
# Creamos una lista vacía para almacenar elementos
lista = []
# Iniciamos un bucle while que se ejecutará indefinidamente hasta que se rompa explícitamente
while True:
    # Solicitamos al usuario que ingrese un elemento
    elemento = input("Ingrese un elemento (o 'salir' para finalizar): ")
    # Verificamos si el usuario ingresó 'salir', en cuyo caso rompemos el bucle
    if elemento.lower() == 'salir':
        break
    # Agregamos el elemento ingresado por el usuario a la lista
    lista.append(elemento)
    # Imprimimos un mensaje para confirmar que el elemento se ha agregado correctamente a la lista
    print("El elemento se agrego correctamente a la Lista")
# Una vez que el usuario haya ingresado 'salir' y el bucle se haya roto, imprimimos los elementos de la lista
print("La lista contiene los siguientes elementos: ")
print(lista)
# Calculamos la cantidad de elementos en la lista utilizando la función len()
cantidad = len(lista)
# Imprimimos el tamaño de la lista
print("El tamanio de la lista es: ")
print(cantidad)
```

Codigo:

Ejecución:



4. Diccionarios en Python

un diccionario es una estructura de datos mutable y no ordenada que permite

almacenar pares clave-valor. Las claves en un diccionario son únicas y se utilizan para acceder a los valores asociados. Los diccionarios son heterogéneos, lo que significa que los valores pueden ser de cualquier tipo de datos, mientras que las claves suelen ser cadenas o enteros. Los diccionarios proporcionan operaciones y métodos útiles para agregar, eliminar y modificar elementos, así como para

acceder a las claves y valores. Son ampliamente utilizados en Python debido a su flexibilidad y eficiencia para asociar información relacionada y manipular datos de manera eficiente.

Ejercicio: Crear una agenda telefónica utilizando diccionarios.

```python
# Crear el diccionario para almacenar los contactos
agenda = {}

# Función para agregar un contacto
def agregar_contacto(nombre, telefono):
    agenda[nombre] = telefono
    print("Contacto", nombre, "agregado con exito.")

# Función para buscar un contacto
def buscar_contacto(nombre):
    if nombre in agenda:
        print(nombre + ":", agenda[nombre])
    else:
        print("El contacto", nombre, "no existe.")

# Función para eliminar un contacto
def eliminar_contacto(nombre):
    if nombre in agenda:
        del agenda[nombre]
        print("Contacto", nombre, "eliminado con exito.")
    else:
        print("El contacto", nombre, "no existe.")

# Función para mostrar todos los contactos
def mostrar_contactos():
    if agenda:
        print("Agenda telefonica:")
        for nombre, telefono in agenda.items():
            print(nombre + ":", telefono)
    else:
        print("La agenda esta vacia.")

# Función principal para el menú de la agenda
def menu():
    while True:
        print("\nAgenda Telefonica")
        print("1. Agregar contacto")
        print("2. Buscar contacto")
        print("3. Eliminar contacto")
        print("4. Mostrar todos los contactos")
        print("5. Salir")
        opcion = input("Seleccione una opcion: ")
        if opcion == '1':
            nombre = input("Ingrese el nombre del contacto: ")
            telefono = input("Ingrese el telefono del contacto: ")
            agregar_contacto(nombre, telefono)
        elif opcion == '2':
            nombre = input("Ingrese el nombre del contacto a buscar: ")
            buscar_contacto(nombre)
        elif opcion == '3':
            nombre = input("Ingrese el nombre del contacto a eliminar: ")
            eliminar_contacto(nombre)
        elif opcion == '4':
            mostrar_contactos()
        elif opcion == '5':
            print("Saliendo de la agenda telefonica.")
            break
        else:
            print("Opcion no valida, por favor seleccione una opcion del 1 al 5.")

# Ejecutar el menú
menu()
```

Codigo:

Ejecución:

```
Agenda Telefónica
1. Agregar contacto
2. Buscar contacto
3. Eliminar contacto
4. Mostrar todos los contactos
5. Salir
Seleccione una opción: 1
Ingrese el nombre del contacto: Luis
Ingrese el teléfono del contacto: 961836166
Contacto Luis agregado con éxito.

Agenda Telefónica
1. Agregar contacto
2. Buscar contacto
3. Eliminar contacto
4. Mostrar todos los contactos
5. Salir
Seleccione una opción: 2
Ingrese el nombre del contacto a buscar: Luis
Luis: 961836166

Agenda Telefónica
1. Agregar contacto
2. Buscar contacto
3. Eliminar contacto
4. Mostrar todos los contactos
5. Salir
Seleccione una opción: 3
Ingrese el nombre del contacto a eliminar: Luis
Contacto Luis eliminado con éxito.

Agenda Telefónica
1. Agregar contacto
2. Buscar contacto
3. Eliminar contacto
4. Mostrar todos los contactos
5. Salir
Seleccione una opción: 4
La agenda está vacía.

Agenda Telefónica
1. Agregar contacto
2. Buscar contacto
3. Eliminar contacto
4. Mostrar todos los contactos
5. Salir
Seleccione una opción: 5
Saliendo de la agenda telefónica.
```

5. Archivos en Python

Los archivos en Python son recursos utilizados para almacenar y recuperar datos de manera persistente en el sistema de archivos de la computadora. Permiten conservar datos más allá de la duración de un programa en ejecución. Los archivos pueden ser de varios tipos, como archivos de texto sin formato, archivos CSV, archivos JSON, archivos XML, archivos binarios, entre otros. Al trabajar con archivos, se especifica un modo de apertura que indica cómo se utilizará el archivo, como lectura, escritura o agregación. Las operaciones básicas incluyen abrir un archivo, leer o escribir datos en él y cerrarlo cuando haya terminado. Es importante manejar los errores y excepciones que pueden ocurrir al trabajar con archivos. En

resumen, los archivos en Python son esenciales para muchas aplicaciones que requieren el almacenamiento y procesamiento de datos de manera persistente.

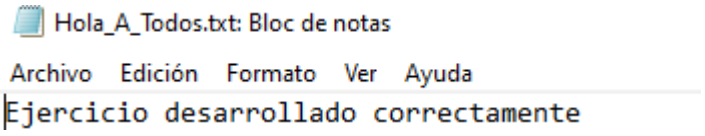
Ejercicio: Escribir y leer datos desde un archivo de texto.

```python
# Definición de la función para escribir en un archivo
def Escribir_archivo():
    # Solicitar al usuario que ingrese el nombre del archivo a escribir
    nombre_archivo = input("Ingrese el nombre del archivo a escribir: ")
    nombre_archivo = nombre_archivo + ".txt"  # Agregar extensión .txt al nombre del archivo

    # Abrir el archivo en modo escritura ('w') usando 'with' para garantizar el cierre automático del archivo
    with open(nombre_archivo, 'w') as archivo:
        # Solicitar al usuario que ingrese el texto a escribir en el archivo
        texto = input("Digite un texto para escribir en el archivo: ")
        # Escribir el texto en el archivo, agregando un salto de línea al final
        archivo.write(texto + "\n")

    # Imprimir un mensaje para confirmar que el texto se ha escrito correctamente en el archivo
    print("Texto escrito en el archivo correctamente")

# Definición de la función para leer un archivo
def Leer_archivo():
    # Solicitar al usuario que ingrese el nombre del archivo a leer
    nombre_archivo = input("Digite el nombre del archivo a leer: ")
    nombre_archivo += ".txt"  # Agregar extensión .txt al nombre del archivo

    try:
        # Intentar abrir el archivo en modo lectura ('r') usando 'with' para garantizar el cierre automático del archivo
        with open(nombre_archivo, 'r') as archivo:
            # Leer todo el contenido del archivo
            contenido = archivo.read()
            # Imprimir el contenido del archivo
            print("El contenido del archivo:")
            print(contenido)
    except FileNotFoundError:
        # Manejar el caso en el que el archivo no se pueda encontrar
        print("No se pudo encontrar el archivo")

# Llamar a la función para escribir en un archivo
Escribir_archivo()

# Llamar a la función para leer un archivo
Leer_archivo()
```

Codigo:

Ejecución:

```
Ingrese el nombre del archivo a escribir: Hola_A_Todos
Digite un texto para escribir en el archivo: Ejercicio desarrollado correctamente
Texto escrito en el archivo correctamente
Digite el nombre del archivo a leer: Hola_A_Todos
El contenido del archivo:
Ejercicio desarrollado correctamente
```



6. Clases en Python

una clase es una plantilla que se utiliza para crear objetos. Las clases encapsulan datos y funcionalidad relacionados en una sola entidad, lo que facilita la organización y el mantenimiento del código. Las clases pueden tener atributos que representan datos asociados con los objetos y métodos que operan en esos datos. Además, las clases admiten la herencia, lo que permite crear una jerarquía de clases donde las subclases pueden extender o modificar el comportamiento de las superclases. Las clases también admiten el polimorfismo, lo que significa que diferentes objetos pueden responder al mismo mensaje de manera diferente. En resumen, las clases en Python son una parte fundamental de la programación orientada a objetos y proporcionan una forma eficaz de modelar entidades del mundo real y estructurar el código de manera modular y reutilizable.

Ejercicio : Definir una clase Persona con atributos nombre y edad, y un método para mostrar estos datos.

```python
# Definición de la clase Persona
class Persona:
    # Método especial __init__ para inicializar objetos de la clase con atributos nombre y edad
    def __init__(self, nombre, edad):
        self.nombre = nombre  # Asignación del nombre pasado como parámetro al atributo nombre del objeto
        self.edad = edad  # Asignación de la edad pasada como parámetro al atributo edad del objeto

    # Método para mostrar los datos de la persona
    def mostrar_datos(self):
        print("Nombre:", self.nombre, "Edad:", self.edad)

# Función para crear una instancia de Persona con datos ingresados por el usuario
def crear_persona():
    nombre = input("Ingrese el nombre de la persona: ")  # Solicitar al usuario que ingrese el nombre
    edad = int(input("Ingrese la edad de la persona: "))  # Solicitar al usuario que ingrese la edad
    return Persona(nombre, edad)  # Devolver una nueva instancia de Persona con los datos ingresados

# Ejemplo de uso:
print("Crear una persona:")
persona1 = crear_persona()  # Llamar a la función crear_persona para crear una nueva instancia de Persona
persona1.mostrar_datos()  # Llamar al método mostrar_datos para mostrar los datos de la persona creada
```

Codigo:
Ejecución:

```
Crear una persona:
Ingrese el nombre de la persona: Luis
Ingrese la edad de la persona: 18
Nombre: Luis Edad: 18
```

Proyecto de Fundamentos de la programación parte 2

Introducción:

Continuando con la segunda parte del proyecto, sean desarrollado cuatro problemas básicos que abarcan diferentes aspectos fundamentales de la programación en python. Estos problemas son:

- Crear una calculadora que pueda realizar operaciones básicas (suma, resta, multiplicación, división).
- un juego de adivinanza de números
- un conversor de monedas
- una aplicación de gestión de tareas

El objetivo de este informe es presentar una descripción detallada de cada proyecto, su funcionalidad y el proceso de desarrollo.

1. Calculadora de Operaciones Básicas:

La calculadora es un programa que permite realizar operaciones aritméticas básicas como suma, resta, multiplicación y división. Este problema haremos uso de funciones y de la validación de entradas del usuario para el uso optimo.

Codigo:

```python
def suma(a, b): # Funcion para sumar dos numeros
    return a + b
def resta(a, b): # Funcion para restar dos numeros
    return a - b
def multiplicacion(a, b): # Funcion para multiplicar dos numeros
    return a * b
def division(a, b): # Funcion para dividir dos numeros
    if b != 0:
        return a / b
    else:
        return "Indeterminada"
def main():
    opcion = 0
    while opcion <= 0 or opcion > 5:
        print("\t\tBienvenido a la calculadora: ") # Menu interactivo para saber que funcion desea el usuario ingresar
        print("1. Suma")
        print("2. Resta")
        print("3. Multiplicacion")
        print("4. Division")
        print("5. Salir")
        print("Digite una operacion: ")
        opcion = int(input())
    if opcion == 5:  # Aqui por si no desea usar la calculadora
        print("Gracias por usar la calculadora")
        return
    num1 = float(input("Digite el primer numero: "))# Digitamos los 2 numeros para usar las respectivas funciones
    num2 = float(input("Digite el segundo numero: "))
    # Aqui se ejecuta las funciones segun corresponda
    if opcion == 1:
        print(f"La suma de los numeros es: {suma(num1, num2)}")
    elif opcion == 2:
        print(f"La resta de los numeros es: {resta(num1, num2)}")
    elif opcion == 3:
        print(f"La multiplicacion de los numeros es: {multiplicacion(num1, num2)}")
    elif opcion == 4:
        resultado_division = division(num1, num2)
        print(f"La division de los numeros es: {resultado_division}")
# Se verifica si se está ejecutando como el programa principal
if __name__ == "__main__":
    main()
```

Funcionalidad:

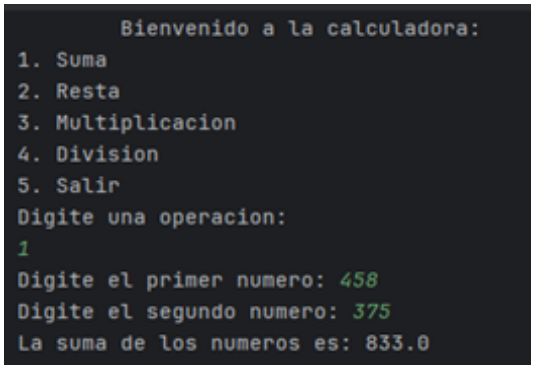
- **Operaciones:** La calculadora puede realizar las operaciones de suma, resta, multiplicación y división.
- **Validación:** El programa valida las entradas para asegurarse de que sean números y maneja errores como la división por cero.
- **Interfaz de Usuario:** La calculadora presenta una interfaz simple donde el usuario puede introducir dos números y seleccionar la operación deseada.

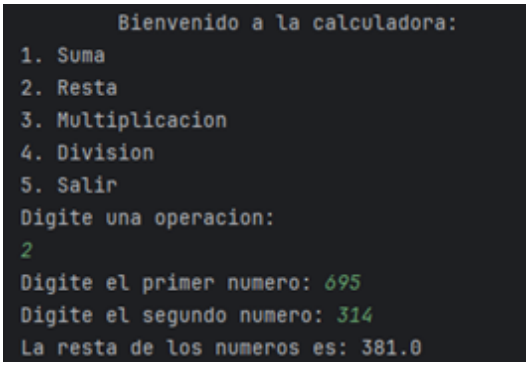
Proceso de Desarrollo:

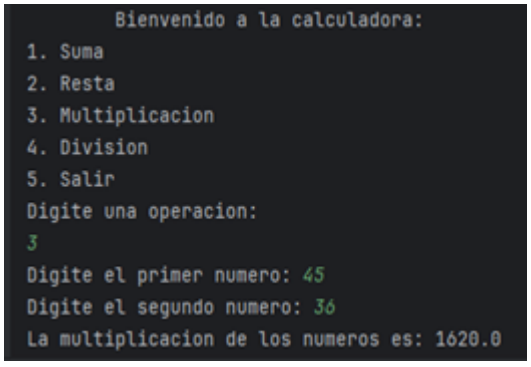
Para el desarrollo de este codigo tuvimos en cuenta estos puntos clave

- Diseño de la interfaz de usuario.
- Implementación de funciones para cada operación.
- Adición de validaciones y manejo de errores

Consola:







```
        Bienvenido a la calculadora:
1. Suma
2. Resta
3. Multiplicacion
4. Division
5. Salir
Digite una operacion:
4
Digite el primer numero: 765
Digite el segundo numero: 45
La division de los numeros es: 17.00
```

```
       Bienvenido a la calculadora:
1. Suma
2. Resta
3. Multiplicacion
4. Division
5. Salir
Digite una operacion:
4
Digite el primer numero: 78
Digite el segundo numero: 0
La division de los numeros es: Indeterminada
```

```
        Bienvenido a la calculadora:
1. Suma
2. Resta
3. Multiplicacion
4. Division
5. Salir
Digite una operacion:
5
Gracias por usar la calculadora
```

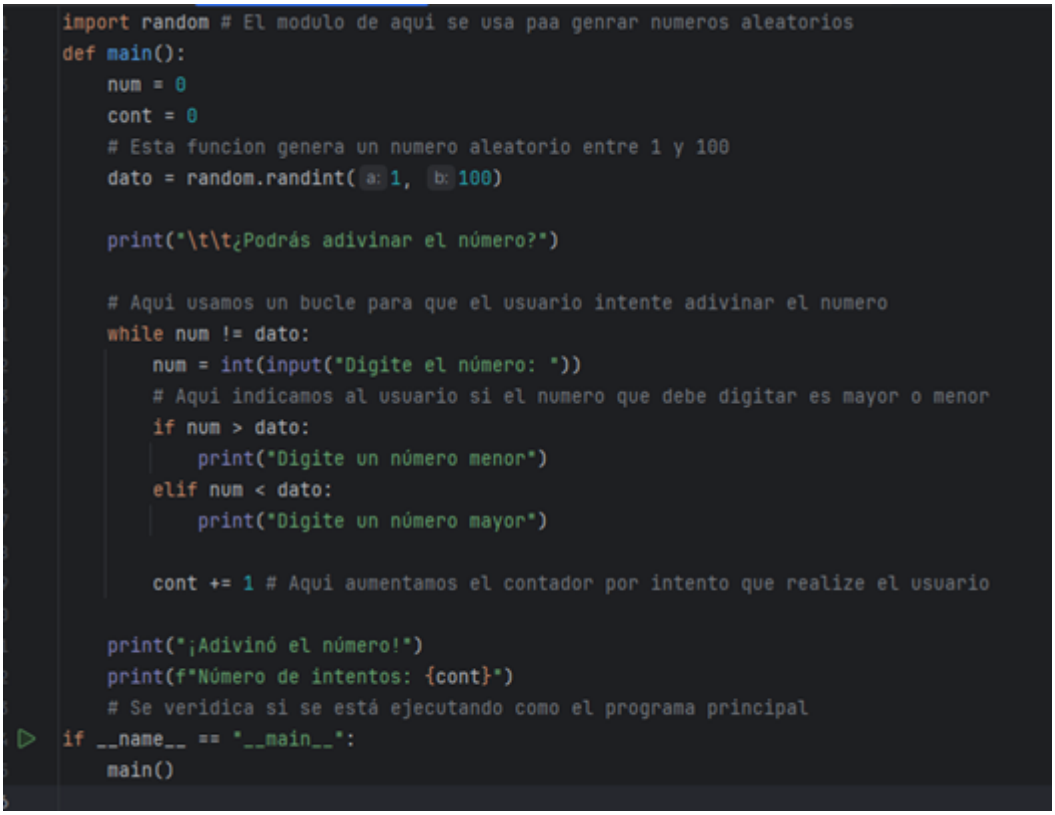
2. Juego de Adivinanza de Números:

Este juego consiste en que el programa elige un número aleatorio y el usuario intenta adivinarlo. El programa proporciona pistas indicando si el número adivinado es mayor o menor que el número ha encontrar.

Observacion:

Puse el paramtro de 1 a 100 para tener un rango a encontrar

Codigo:



```python
import random # El modulo de aquí se usa paa genrar numeros aleatorios
def main():
    num = 0
    cont = 0
    # Esta funcion genera un numero aleatorio entre 1 y 100
    dato = random.randint( a: 1,  b: 100)

    print("\t\t¿Podrás adivinar el número?")

    # Aquí usamos un bucle para que el usuario intente adivinar el numero
    while num != dato:
        num = int(input("Digite el número: "))
        # Aqui indicamos al usuario si el numero que debe digitar es mayor o menor
        if num > dato:
            print("Digite un número menor")
        elif num < dato:
            print("Digite un número mayor")

        cont += 1 # Aqui aumentamos el contador por intento que realize el usuario

    print("¡Adivinó el número!")
    print(f"Número de intentos: {cont}")
    # Se veridica si se está ejecutando como el programa principal
if __name__ == "__main__":
    main()
```

Funcionalidad:

- **Generación de Números:** El programa genera un número aleatorio dentro de un rango predefinido.
- **Interacción con el Usuario:** El usuario introduce su adivinanza y el programa le indica si el número es mayor o menor que el número objetivo.
- **Intentos:** El programa puede limitar el número de intentos o permitir intentos ilimitados.

Proceso de Desarrollo:

Para el desarrollo de este codigo tuvimos en cuenta estos puntos clave

- **Implementación de la generación de números aleatorios.**
- **Creación de la lógica para comparar el número adivinado con el número objetivo.**
- **Diseño de la interacción con el usuario.**

Consola:

```
        ¿Podrás adivinar el número?
Digite el número: 78
Digite un número menor
Digite el número: 42
Digite un número mayor
Digite el número: 64
Digite un número mayor
Digite el número: 70
Digite un número mayor
Digite el número: 75
Digite un número menor
Digite el número: 73
¡Adivinó el número!
Número de intentos: 6
```

```
        ¿Podrás adivinar el número?
Digite el número: 50
Digite un número mayor
Digite el número: 70
Digite un número mayor
Digite el número: 85
Digite un número mayor
Digite el número: 90
Digite un número menor
Digite el número: 87
Digite un número mayor
Digite el número: 89
¡Adivinó el número!
Número de intentos: 6
```

3. Conversor de Monedas:

Este programa convierte entre diferentes monedas utilizando tasas de cambio predefinidas. El usuario puede seleccionar las monedas de origen y destino, y el programa calcula el equivalente en la moneda deseada.

Codigo:

```python
def dolares(a):
    tasa_dolares = 0.26295 # Tasa de cambio  de soles a dólares
    cant_dolares = a * tasa_dolares # Convertir la cantidad de soles a dólares
    return cant_dolares
def euros(a):
    tasa_euros = 0.24488 # Tasa de cambio de soles a euros
    cant_euros = a * tasa_euros # Convertir la cantidad de soles a euros
    return cant_euros
def yenes(a):
    tasa_yenes = 41.91467 # Tasa de cambio de soles a yenes
    cant_yenes = a * tasa_yenes # Convertir la cantidad de soles a yenes
    return cant_yenes
def peso_mexicano(a):
    tasa_pesos = 4.71028 # Tasa de cambio de soles a pesos mexicanos
    cant_pesos = a * tasa_pesos # Convertir la cantidad de soles a pesos mexicanos
    return cant_pesos
def main():
    moneda = 0
    # Aqui usamos un bucle para que ingrese una cantidad de dinero real
    while True:
        moneda = float(input("Digite su cantidad de dinero en soles: "))
        if moneda >= 0:
            break
        # Aqui imprimimos el resultado de convertir nuestra cantidad de soles
        # a otras divisas precisando los 2 primeros decimales
    print("Las cantidades en otras divisas son:")
    print(f"En dólares: {dolares(moneda):.2f}")
    print(f"En euros: {euros(moneda):.2f}")
    print(f"En yenes: {yenes(moneda):.2f}")
    print(f"En pesos mexicanos: {peso_mexicano(moneda):.2f}")

# Se veridica si se está ejecutando como el programa principal
if __name__ == "__main__":
    main()
```

Funcionalidad:

- Selección de Monedas: El usuario puede seleccionar la moneda de origen y la moneda de destino.
- Tasas de Cambio: El programa utiliza tasas de cambio predefinidas para realizar la conversión en este caso usamos 4.
- Cálculo: El programa calcula el valor convertido basado en la tasa de cambio seleccionada.

Proceso de Desarrollo:

El desarrollo del conversor de monedas incluyó:

- **Definición de las tasas de cambio.**
- **Implementación de la lógica para realizar la conversión.**
- **Creación de la interfaz de usuario para seleccionar las monedas y mostrar el resultado.**

Consola:

```
Digite su cantidad de dinero en soles: 750
Las cantidades en otras divisas son:
En dólares: 197.21
En euros: 183.66
En yenes: 31436.00
En pesos mexicanos: 3532.71
```

```
Digite su cantidad de dinero en soles: 2140
Las cantidades en otras divisas son:
En dólares: 562.71
En euros: 524.04
En yenes: 89697.39
En pesos mexicanos: 10080.00
```

4. Aplicación de Gestión de Tareas:

Esta aplicación permite a los usuarios agregar, eliminar y marcar tareas como completadas. Es una herramienta útil para la gestión personal de tareas y pendientes.

Codigo:

```python
class Tarea:
    def __init__(self, descripcion):
        # Inicializamos la tarea con una descripcion y un estado incompleto
        self.descripcion = descripcion
        self.completada = False

def mostrar_menu():
    # Mostramos el menu principal y las opciones para el usuario
    print("\nMenú de tareas:")
    print("1. Agregar tarea")
    print("2. Eliminar tarea")
    print("3. Marcar tarea como completada")
    print("4. Mostrar tareas")
    print("5. Salir")
    return int(input("Seleccione una opción: "))

def agregar_tarea(tareas):
    # Agregamos una tarea a la lista
    descripcion = input("Ingrese la descripción de la tarea: ")
    tareas.append(Tarea(descripcion))
    print("¡Tarea agregada!")

def eliminar_tarea(tareas):
    # Eliminamos una tarea de la lista
    mostrar_tareas(tareas)
    indice = int(input("Ingrese el índice de la tarea a eliminar: "))
    if 0 <= indice < len(tareas):
        tareas.pop(indice)
        print("¡Tarea eliminada!")
    else:
        print("¡Índice no válido!")
```

```python
def marcar_tarea_completada(tareas):
    # Marcamos la tarea como completada
    mostrar_tareas(tareas)
    indice = int(input("Ingrese el índice de la tarea a marcar como completada: "))
    if 0 <= indice < len(tareas):
        tareas[indice].completada = True
        print("¡Tarea marcada como completada!")
    else:
        print("¡Índice no válido!")

def mostrar_tareas(tareas):
    # Mostramos todas las tareas con su estado (completada o pendiente)
    print("\nLista de tareas:")
    for i, tarea in enumerate(tareas):
        estado = "Completada" if tarea.completada else "Pendiente"
        print(f"{i}. {tarea.descripcion} [{estado}]")

def main():
    tareas = []
    while True:
        # Aquí muestra el menú y obtiene la opción seleccionada por el usuario
        opcion = mostrar_menu()
        if opcion == 1:
            agregar_tarea(tareas)
        elif opcion == 2:
            eliminar_tarea(tareas)
        elif opcion == 3:
            marcar_tarea_completada(tareas)
        elif opcion == 4:
            mostrar_tareas(tareas)
        elif opcion == 5:
            print("Saliendo del programa...")
            break
        else:
            print("¡Opción no válida!")

if __name__ == "__main__":
    # Se verifica si se está ejecutando como el programa principal
    main()
```

Funcionalidad:

- **Agregar Tareas:** Los usuarios pueden agregar nuevas tareas a la lista.
- **Eliminar Tareas:** Los usuarios pueden eliminar tareas de la lista.
- **Marcar como Completadas:** Los usuarios pueden marcar tareas como completadas.

Proceso de Desarrollo:

El desarrollo de la aplicación de gestión de tareas incluyó:

- **Implementación de la funcionalidad para agregar, eliminar y marcar tareas.**
- **Diseño de la interfaz de usuario para mostrar y gestionar las tareas.**
- **Manejo del almacenamiento de tareas, ya sea en memoria o en un archivo.**

Consola:

```
Menú de tareas:
1. Agregar tarea
2. Eliminar tarea
3. Marcar tarea como completada
4. Mostrar tareas
5. Salir
Seleccione una opción: 1
Ingrese la descripción de la tarea: terminar el proyecto
¡Tarea agregada!

Menú de tareas:
1. Agregar tarea
2. Eliminar tarea
3. Marcar tarea como completada
4. Mostrar tareas
5. Salir
Seleccione una opción: 1
Ingrese la descripción de la tarea: estudiar para el examen
¡Tarea agregada!

Menú de tareas:
1. Agregar tarea
2. Eliminar tarea
3. Marcar tarea como completada
4. Mostrar tareas
5. Salir
Seleccione una opción: 3

Lista de tareas:
0. terminar el proyecto [Pendiente]
1. estudiar para el examen [Pendiente]
Ingrese el índice de la tarea a marcar como completada: 0
¡Tarea marcada como completada!
```

```
Menú de tareas:
1. Agregar tarea
2. Eliminar tarea
3. Marcar tarea como completada
4. Mostrar tareas
5. Salir
Seleccione una opción: 4

Lista de tareas:
0. terminar el proyecto [Completada]
1. estudiar para el examen [Pendiente]

Menú de tareas:
1. Agregar tarea
2. Eliminar tarea
3. Marcar tarea como completada
4. Mostrar tareas
5. Salir
Seleccione una opción: 2

Lista de tareas:
0. terminar el proyecto [Completada]
1. estudiar para el examen [Pendiente]
Ingrese el índice de la tarea a eliminar: 0
¡Tarea eliminada!

Menú de tareas:
1. Agregar tarea
2. Eliminar tarea
3. Marcar tarea como completada
4. Mostrar tareas
5. Salir
Seleccione una opción: 4
```

```
Lista de tareas:
0. estudiar para el examen [Pendiente]

Menú de tareas:
1. Agregar tarea
2. Eliminar tarea
3. Marcar tarea como completada
4. Mostrar tareas
5. Salir
Seleccione una opción: 5
Saliendo del programa...
```

Desarrollo Web con Flask/Django: Crear una aplicación web simple utilizando Flask o Django

Flask es un **framework**(un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software) **ligero y flexible**para el desarrollo de aplicaciones web en Python. Permite crear aplicaciones web rápidamente y con una **sintaxis sencilla**, proporcionando herramientas para el manejo de peticiones HTTP, manejo de sesiones, autenticación de usuarios, entre otros. Flask es ideal para crear aplicaciones web simples y rápidas, pero también es lo suficientemente potente y extensible para desarrollar aplicaciones web más complejas. Es una herramienta muy popular en la comunidad de desarrollo web en Python.

Para alguien experimentado, cuando piensa en Python, el framework de facto que llega a su mente es el framework Django. Pero desde una perspectiva de principiantes (en este caso nosotros) de Python, Flask es más fácil para comenzar que en comparación con Django.

PASOS:

Primero debemos **instalar Python**, de preferencia versiones +**3.7**; luego usando el comando **pip**en el terminal procedemos a **instalar el Flask**:

```
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Windows\System32>pip install Flask
Requirement already satisfied: Flask in c:\users\familia perez\appdata\local\programs\python\python312\lib\sit
e-packages (3.0.3)
Requirement already satisfied: Werkzeug>=3.0.0 in c:\users\familia perez\appdata\local\programs\python\python3
12\lib\site-packages (from Flask) (3.0.3)
Requirement already satisfied: Jinja2>=3.1.2 in c:\users\familia perez\appdata\local\programs\python\python312
\lib\site-packages (from Flask) (3.1.4)
Requirement already satisfied: itsdangerous>=2.1.2 in c:\users\familia perez\appdata\local\programs\python\pyt
hon312\lib\site-packages (from Flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in c:\users\familia perez\appdata\local\programs\python\python312\
lib\site-packages (from Flask) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in c:\users\familia perez\appdata\local\programs\python\python31
2\lib\site-packages (from Flask) (1.8.2)
Requirement already satisfied: colorama in c:\users\familia perez\appdata\local\programs\python\python312\lib\
site-packages (from click>=8.1.3->Flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\familia perez\appdata\local\programs\python\python3
12\lib\site-packages (from Jinja2>=3.1.2->Flask) (2.1.5)

C:\Windows\System32>Flask --version
Python 3.12.4
Flask 3.0.3
Werkzeug 3.0.3

C:\Windows\System32>
```

Procedemos a crea un archivo de Python en nuestro editor de texto favorito (en este caso elegimos **VS Code**) y lo nombramos *app1.py*. En este archivo, comenzamos importando Flask y creando una instancia de la aplicación:

src > static > css > app1.py > ...

```python
#de la libreria flask importamos Flask
from flask import Flask, render_template
app = Flask(__name__) #designamos un objeto 'app'

@app.route("/") #creamos una ruta para la página principal
def home(): #definimos una funcion
    return render_template("home.html") #tomamoslo de un archivo html

@app.route('/about')
def about():
    return render_template('about.html') #tomamoslo de un archivo html

if __name__ == "__main__":
    app.run(debug=True) #Autoreiniciar en cada cambio
```

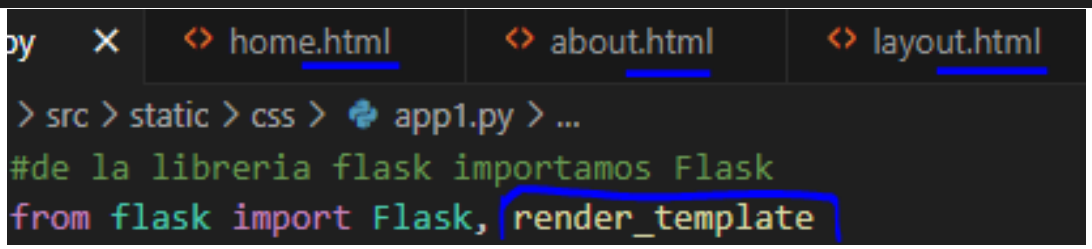
Guardamos el archivo y lo ejecutamos en la terminal utilizando el comando **python app1.py**. Flask iniciará un servidor local y podremos acceder a la aplicación web en nuestro navegador ingresando la URL `http://localhost:5000/`. (utilizamos Chrome)

```
PS C:\Users\Familia Perez\Desktop\Facultad de Ciencias (FC-UNI)\3er Ciclo\CC112 Fundamentos de porgramación\proyecto> python app
1.py
```

Nosotros en realidad no queremos estar enviando tan solo texto, queremos enviar HTML's, ya que cuando creamos un sitio web tenemos que crear páginas web con interfaces y así saber sobre qué trata este sitio web.

En este caso creamos una carpeta llamada 'templates' que contenga archivos HTML's y en cada ruta retornaremos un archivo HTML, pero para ello primero importamos desde Flask 'render_template'

```python
@app.route("/") #creamos una ruta para la página principal
def home(): #definimos una funcion
    return render_template("home.html") #tomamoslo de un archivo html

@app.route('/about')
def about():
    return render_template('about.html') #tomamoslo de un archivo html
```



Utilizamos una plantilla **HTML:5**que viene por default en VS Code y nos facilita bastante, usamos esto en primera instancia en **home.html**

Como nuestro programa estará en un proceso de prueba, y lo estamos modificando y probando cada instante, para no estar borrando y volviendo a ejecutar, queremos que se auto reinicie en cada cambio

```python
if __name__ == "__main__":
    app.run(debug=True) #Auto-reiniciar en cada cambio
```

Para estilizar mejor nuestra página creamos una carpeta '**static'**, dentro una carpeta '**css'**y dentro un archivo '**main.css'**; luego enlazamos el archivo css con el HTML.

Para mejorar nuestro diseño haremos uso de algún freamwork de css, en este caso usaremos **Bootstrap**que es bien conocido; es un freamwork que nos proporciona estilos de css ya creados debido al poco conocimiento de css que tenemos, además que nos ahorra tiempo en lugar de escribir código css puro.

Copiamos en el HTML el **CDN**para el css que aparece en la página de Bootstrap, que no es más que una dirección de internet, en el **home.html**.

```html
<!--BOOTSTRAP 5-->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin=
    "anonymous">
<!--CUSTOM CSS-->
```

Dentro de la página de Bootstrap, en la pestaña de **Navbar**copiamos el código de estilo de navegación más preparada, que contenga un **título y pestañas**que nos conduzca a cada ruta. Modificamos el título y el nombre de 2 pestañas que en este caso serán las únicas que usaremos.

```html
<nav class="navbar navbar-expand-lg bg-body-tertiary">
    <div class="container-fluid">
        <a class="navbar-brand" href="/">Python WebApp</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
            data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false"
            aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item">
                    <a class="nav-link active" aria-current="page" href="/">Home Page</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="{{ url_for('about') }}">About Page</a>
                </li>
            </ul>
    </div>
</nav>
```

Creamos ahora un tercer archivo HTML, llamado **layout**, que nos servirá para colocar todo el código que podamos *reutilizar*que se encontraba en **home.html**; en este caso reutilizaremos lo que se encontraba en el head que siempre se repite, y la parte de la navegación, debajo de esta última será lo que va a cambiar. Colocaremos un **'block content'**y **'endblock',**lo que significa que justo en estas 2 líneas de código vamos a poder colocar todo el código de otras páginas capaces de reutilizar el contenido del **'head'** y la **navegación**

```
38        {% block content %}
39        {% endblock%}
```

Todo el código de **home.html**y **about.html**va a *extender*de un archivo llamado layout.html, en los que únicamente escribimos lo que no va a repetirse, ya que el código equivalente a ambos está en el layout; el código va *entre 'block content' y 'endblock'*, *reutilizando*así una porción de código en diversos archivos, navegando fácilmente en la página.

home

```
{% extends "layout.html" %}

{% block content %}

<div class="jumbotron">
    <h1 class="display-4"><b><u>Qué tal, profesor Américo</u>!</b></h1>
    <p class="lead">Esta es básicamente nuestra aplicación/página web como resultado del uso de <i><b>Flask</b></i>
      (framework sencillo y flexible para los usuarios) y <i><b>Python</b></i>, espero le agrade.</p>
    <hr class="my-4">
    <p>Muchas gracias por el conocimiento brindado a nuestras personas en las prácticas del curso
      <i><b>Fundamentos de programación.</b></i></p>
    <a class="btn btn-primary btn-lg" href="#" role="button"><b>Nos veremos pronto</b></a>
  </div>

{% endblock%}
```

about

```
{% extends "layout.html" %}

{% block content %}

<h1>Para hacer una aplicación web sencilla utilizando Flask y Python, seguimos básicamente
    los pasos siguientes:

    <br><br>- Primero, debemos asegurarnos de tener <b>Python</b> instalado en la computadora. Luego,
    instalar <b>Flask</b> utilizando el comando pip en la terminal.

    <br>- Procedemos luego a crear un archivo de Python en VS Code y nombrarlo (Nombre_cualquiera).py.
    En este archivo, comenzamos importando Flask y creando una instancia de la aplicación:

    <br>- Finalmente guardamos el archivo y lo ejecutamos en la terminal utilizando el comando
    python (Nombre_cualquiera).py. Flask iniciará un servidor local y podremos acceder a la
    aplicación web en nuestro navegador ingresando la URL <u>http://localhost:5000/</u>.
    ¡Listo! Ahora poseemos una aplicación web sencilla utilizando Flask y Python.
    Claro que podemos añadir <b>más rutas, templates HTML, estilos CSS y funcionalidades adicionales</b>
    para personalizar y mejorar la aplicación web. (que es lo que hicimos justamente)

    <br><br><b>(Mayores detalles en el proyecto en látex y el código subido a GitHub)</b>
    </h1>

{% endblock%}
```

En la página de **Bootstrap**, ventana de **Jumbotrom**, copiamos la plantilla de código en el **home.html**que nos proporciona un buen diseño de nuestra página principal (de bienvenida), y lo colocamos en un *container* en el **layout**para que esté centrado.

```
<div class="jumbotron">
    <h1 class="display-4"><b><u>Qué tal, profesor Américo</u>!</b></h1>
    <p class="lead">Esta es básicamente nuestra aplicación/página web como resultado del uso de <i><b>Flask</b></i>
      (framework sencillo y flexible para los usuarios) y <i><b>Python</b></i>, espero le agrade.</p>
    <hr class="my-4">
    <p>Muchas gracias por el conocimiento brindado a nuestras personas en las prácticas del curso
      <i><b>Fundamentos de programación.</b></i></p>
    <a class="btn btn-primary btn-lg" href="#" role="button"><b>Nos veremos pronto</b></a>
```

```
37    <div class="container p-4">
38        {% block content %}
39        {% endblock%}
40    </div>
```

Finalmente le aplicaremos un color de fondo a nuestra aplicación, para ello ingresamos a una página del navegador llamada **uigradients**que nos *proporciona degradados en código css*, elegimos el color que más nos guste, copiamos su código y lo pegamos en el **main.css**.

Luego de todos estos pasos, estaría completada nuestra aplicación web sencilla; cabe mencionar que posteriormente usando **Flask**y **mysqldb**podemos hacer *formularios*, *tomardatos*y demás. ¡Es un mundo muy grande por incursionar!

Agradecemos a los canales: ***Reolcode***, ***Fazt***, ***Condenautas***, ***DariDeveloper***, ***ikerdev UskoKruM2010***, ***tavodev***y ***Frank Andrade***, *por la información brindada en este campo en el que no estábamos completamente familiarizados.*