Ao de la unidad, la paz y el desarrollo

UNIVERSIDAD NACIONAL DE INGENIERA

FACULTAD DE CIENCIAS

Escuela profesional de Ciencias de la Computación

PROYECTO FINAL

Alumnos:

- Sebastian Octavio Figueroa Rueda 20231447E
- Pinto Salas Danna Ninel 20234190E
- Yupanqui Quispe Roy Mauricio 20232148A

Profesor:

- Chulluncuy Reynoso Americo Andres

Sección: CC112 - D

Índice

1. Introduccion

Este proyecto tiene como finalidad proporcionar a los estudiantes una experiencia prctica en el uso de Python. Mediante esta iniciativa, se busca que los alumnos apliquen los conceptos previamente adquiridos en C++ en un entorno diferente, descubriendo las ventajas de Python en trminos de simplicidad, eficiencia y aplicabilidad en mltiples reas tecnolgicas.

El proyecto tambin ofrecer la oportunidad de trabajar con aplicaciones reales que emplean Python, incentivando as el inters de los estudiantes por reas clave como la inteligencia artificial, el desarrollo web y la ciencia de datos. Adems, se pretende que los participantes amplen sus habilidades de programacin y adquieran una visin ms completa sobre la utilidad de distintos lenguajes de programacin en situaciones prcticas.

En esencia, este proyecto aspira a reforzar los conocimientos previos de los estudiantes, a la vez que fomenta su curiosidad y desarrollo profesional en sectores tecnolgicos emergentes y de gran relevancia.

2. Fundamento Teórico

2.1. Filosofia y diseo

2.1.1. El Zen de Python

El Zen de Python es una coleccin de aforismos que capturan la filosofa del diseo de Python. Se puede acceder a ellos escribiendo import this en el intrprete de Python. Aqu algunos de los aforismos ms destacados:

- Bello es mejor que feo.
- Explcito es mejor que implcito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado
- Espaciado es mejor que denso.

2.1.2. Diseo

- Legibilidad del cdigo Uno de los objetivos principales de Python es que el cdigo sea fcil de leer y entender. Esto se logra mediante una sintaxis limpia y el uso de sangras para definir bloques de cdigo en lugar de llaves o palabras clave.
- Simplicidad y consistencia Python sigue la regla de ünа forma obvia de hacer las cosas". Esto promueve la consistencia y facilita el aprendizaje y uso del lenguaje.
- Tipado dinmico pero fuerte Python es dinmicamente tipado, lo que significa que no se necesita declarar explcitamente los tipos de las variables, pero mantiene un tipado fuerte, evitando operaciones entre tipos incompatibles.
- Interpretado Python es un lenguaje interpretado, lo que significa que el cdigo se ejecuta lnea por lnea, permitiendo una rpida iteracin y prueba de cdigo.
- Multiplataforma Python es compatible con mltiples plataformas, lo que permite ejecutar el mismo cdigo en diferentes sistemas operativos sin necesidad de modificaciones significativas.
- Biblioteca estndar rica Python viene con una amplia biblioteca estndar que cubre muchas reas, desde manejo de archivos hasta protocolos de Internet, facilitando el desarrollo sin necesidad de dependencias externas.

En conclusion la filosofa y el diseo de Python estn orientados a crear un lenguaje que sea fcil de aprender y usar, promoviendo la escritura de cdigo claro, legible y mantenible. Estos principios han contribuido significativamente a su popularidad y adopcin en una amplia variedad de campos, desde desarrollo web hasta ciencia de datos y machine learning.

2.2. Caracteristicas del lenguaje

- Tipado dinmico y fuerte Python es un lenguaje de tipado dinmico, lo que significa que no es necesario declarar el tipo de una variable explcitamente; el tipo se determina automticamente en tiempo de ejecucin. Adems, Python es fuertemente tipado, lo que implica que no permite la combinacin implcita de tipos diferentes en una operacin, evitando as errores ambiguos.
- Gestin automtica de memoria Python maneja la memoria automticamente a travs de un recolector de basura. Utiliza el conteo de referencias y un recolector de basura generacional para gestionar la memoria de forma eficiente.
- Sintaxis clara y concisa Python utiliza una sintaxis clara y legible, con un fuerte nfasis en la legibilidad del cdigo. Los bloques de cdigo se definen por la indentacin, no por llaves o palabras clave.
- Estructuras de datos incorporadas Python incluye estructuras de datos de alto nivel que son fciles de usar y muy poderosas.
  - Listas: Colecciones ordenadas y mutables.
  - Diccionarios: Colecciones desordenadas de pares clave-valor.
  - Conjuntos: Colecciones desordenadas de elementos nicos.
  - Tuplas: Colecciones ordenadas e inmutables.
  - Programacin Orientada a Objetos
  - Python soporta la programacin orientada a objetos (POO), permitiendo la definicin de clases y la creacin de objetos.
- Funciones de primera clase Las funciones en Python son ciudadanos de primera clase, lo que significa que pueden ser pasadas como argumentos, retornadas desde otras funciones y asignadas a variables.
- Manejo de excepciones Python tiene un sistema robusto para el manejo de excepciones, lo que permite gestionar errores de manera elegante y mantener el flujo del programa.
- Biblioteca estndar amplia Python viene con una amplia biblioteca estndar que incluye mdulos para tareas comunes como manejo de archivos, manipulacin de cadenas, operaciones matemticas, y mucho ms.
- Multiplataforma Python es compatible con mltiples plataformas (Windows, macOS, Linux), lo que permite escribir cdigo que se puede ejecutar en diferentes sistemas operativos sin modificaciones significativas.
- Extensibilidad Python se puede extender con mdulos escritos en C o C++, lo que permite optimizar partes del cdigo que necesitan mayor rendimiento.
- Interpretado Python es un lenguaje interpretado, lo que significa que el cdigo se ejecuta lnea por lnea, facilitando la depuracin y la iteracin rpida durante el desarrollo.

  Apoyo a la programacin funcional Python soporta paradigmas de programacin funcional, incluyendo funciones lambda, map, filter y reduce.

En conclusin las caractersticas mencionadas hacen de Python un lenguaje poderoso y flexible, adecuado para una amplia gama de aplicaciones, desde scripts simples hasta complejos sistemas de inteligencia artificial. Su simplicidad, legibilidad y comunidad activa contribuyen a su continua popularidad y expansin en el mundo de la programacin.

2.3. Estructuras de Datos Incorporadas en Python

Python incluye una variedad de estructuras de datos de alto nivel que facilitan el manejo y la manipulacin de datos de manera eficiente y efectiva. A continuacin se detallan las principales estructuras de datos incorporadas:

- Listas : Las listas son colecciones ordenadas y mutables que permiten almacenar una secuencia de elementos. Las listas pueden contener elementos de cualquier tipo y soportan operaciones como aadir, eliminar y modificar elementos. Caractersticas:

- Ordenadas: Mantienen el orden de insercin de los elementos.
- Mutables: Se pueden modificar despus de su creacin.
- Indexadas: Se puede acceder a los elementos mediante ndices.

- Diccionarios : Los diccionarios son colecciones desordenadas de pares clave-valor, donde cada clave es nica y se utiliza para acceder a su valor correspondiente. Los diccionarios son tiles para almacenar datos asociados, como un mapa o un conjunto de propiedades.

  Caractersticas:

  - Desordenados : No mantienen el orden de insercin (hasta Python 3.7, donde se garantiz el orden de insercin).
  - Mutables: Se pueden modificar despus de su creacin.
  - Claves nicas: Cada clave en el diccionario debe ser nica.

- Conjuntos : Los conjuntos son colecciones desordenadas de elementos nicos, lo que significa que no permiten elementos duplicados. Los conjuntos son tiles para operaciones matemticas como la unin, interseccin y diferencia.

- Tuplas : Las tuplas son colecciones ordenadas e inmutables, lo que significa que no se pueden modificar despus de su creacin. Las tuplas son tiles para almacenar datos heterogneos y pueden ser utilizadas como claves en los diccionarios debido a su inmutabilidad.

  Caractersticas:

  - Ordenadas: Mantienen el orden de insercin de los elementos.
  - Inmutables: No se pueden modificar despus de su creacin.
  - Indexadas: Se puede acceder a los elementos mediante ndices.

2.4. Funciones y Manejo de Errores

2.4.1. Funciones

Las funciones en Python son bloques de cdigo reutilizables diseados para realizar tareas especficas. Estas permiten modularizar el cdigo, mejorar su legibilidad y facilitar su mantenimiento.

- Definicin de Funciones: Las funciones se definen utilizando la palabra clave def, seguida del nombre de la funcin y parntesis. Dentro de los parntesis se pueden incluir parmetros que la funcin recibir como entrada.

- Llamada a Funciones: Una vez definida, una funcin puede ser llamada en cualquier parte del cdigo pasando los argumentos necesarios.

- Parmetros y Argumentos: Las funciones pueden tener parmetros que permiten recibir informacin. Los argumentos son los valores que se pasan a la funcin cuando se llama.

- Valores por Defecto: Los parmetros pueden tener valores por defecto, permitiendo que la funcin se llame sin pasar todos los argumentos. Esto proporciona flexibilidad en la forma en que se pueden llamar las funciones.

- Funciones Lambda: Las funciones lambda son funciones annimas definidas con la palabra clave lambda. Son tiles para operaciones pequeas y rpidas que no requieren una funcin completa.

- Funciones de Primera Clase: En Python, las funciones son ciudadanos de primera clase, lo que significa que pueden ser asignadas a variables, pasadas como argumentos a otras funciones y retornadas desde otras funciones.

2.4.2. Manejo de errores

El manejo de errores en Python se realiza mediante un sistema robusto de excepciones, lo que permite gestionar errores de manera elegante y mantener el flujo del programa.

- Bloques try-except: Los bloques try y except se utilizan para capturar y manejar excepciones. El cdigo que puede generar una excepcin se coloca dentro del bloque try, y el cdigo para manejar la excepcin se coloca dentro del bloque except.
- Bloque finally: El bloque finally se ejecuta siempre, independientemente de si se lanz una excepcin o no. Es til para liberar recursos o realizar tareas de limpieza necesarias.
- Levantar Excepciones: Se puede utilizar la palabra clave raise para lanzar una excepcin manualmente, lo que permite gestionar situaciones especficas de error.
- Excepciones Personalizadas: Python permite la creacin de excepciones personalizadas definiendo nuevas clases que heredan de la clase base Exception. Esto es til para manejar errores especficos de la aplicacin de manera ms precisa.

2.5. Librerias y Herramientas

Python se distingue no solo por su sintaxis simple y legible, sino tambin por su extensa coleccin de libreras y herramientas que facilitan el desarrollo en diversos dominios. A continuacin, se describen algunas de las libreras y herramientas ms relevantes en el ecosistema de Python.

2.6. Biblioteca Estndar

La biblioteca estndar de Python es una coleccin de mdulos y paquetes que vienen incluidos con la instalacin de Python. Estos mdulos proporcionan soluciones para muchas tareas comunes, eliminando la necesidad de instalar libreras externas.

- os: Proporciona una forma de utilizar funcionalidades dependientes del sistema operativo, como manipulacin de archivos y directorios.
- sys: Permite acceder a variables y funciones especficas del intrprete de Python.
- datetime: Ofrece clases para manipular fechas y horas de manera sencilla.
- json: Permite trabajar con datos en formato JSON (JavaScript Object Notation).
- math: Proporciona acceso a funciones matemticas como trigonometra, logaritmos, y ms.
- urllib: Facilita el trabajo con URL, incluyendo la recuperacin de datos desde la web.

2.7. Libreras de Terceros

Adems de la biblioteca estndar, Python cuenta con una amplia gama de libreras de terceros que extienden su funcionalidad. Algunas de las ms populares incluyen:

- NumPy: Proporciona soporte para grandes matrices y matrices multidimensionales, junto con una coleccin de funciones matemticas de alto nivel para operar con estos arreglos.
- pandas: Ofrece estructuras de datos y herramientas de anlisis de datos de alto rendimiento, especialmente tiles para la manipulacin de datos tabulares.
- matplotlib: Permite crear visualizaciones estticas, animadas e interactivas en Python. requests: Simplifica el envo de solicitudes HTTP, siendo mucho ms intuitivo que los mdulos de la biblioteca estndar.
- scikit-learn: Proporciona herramientas simples y eficientes para anlisis de datos y minera de datos, incluyendo algoritmos de aprendizaje automtico.
- Flask: Un micro framework para desarrollo web, que permite crear aplicaciones web de manera sencilla y rpida.
- Django: Un framework de alto nivel para el desarrollo web, que fomenta el desarrollo rpido y el diseo limpio y pragmtico.

3. Desarrollo

3.1. Parte I

3.1.1. Ejercicio 1

```python
def factorial(n):
    # Caso base: factorial de 0 o 1 es 1
    if n == 0 or n == 1:
        return 1
    # Caso recursivo: n * factorial(n-1)
    else:
        return n * factorial(n - 1)

# Solicitar al usuario que ingrese un numero entero
while True:
    try:
        numero = int(input("Ingresa un numero entero para calcular su factorial: "))
        if numero < 0:
            print("El factorial no esta definido para n meros negativos.")
        else:
            resultado = factorial(numero)
            print(f"El factorial de {numero} es: {resultado}")
            break
    except ValueError:
        print("Por favor, ingresa un numero entero valido.")
```

3.1.2. Ejercicio 3

```python
def invertir_cadena(cadena):
    # Usamos rebanadas de cadena para invertir la cadena
    cadena_invertida = cadena[::-1]
    return cadena_invertida

# Solicitar al usuario que ingrese la cadena
cadena_original = input("Ingrese una cadena para invertirla: ")

# Llamar a la funcipn para invertir la cadena ingresada por el usuario
cadena_invertida = invertir_cadena(cadena_original)

# Mostrar resultados
print(f"Cadena original: {cadena_original}")
print(f"Cadena invertida: {cadena_invertida}")
```

3.1.3. Ejercicio 5

```python
# Crear una lista inicial
lista = [1, 2, 3, 4, 5]

# Mostrar la lista original
print("Lista original:", lista)

# Modificar un elemento especifico
lista[2] = 10

# Mostrar la lista despues de la modificacion
print("Lista modificada:", lista)

# A adir un nuevo elemento
lista.append(6)

# Mostrar la lista despues de a adir un elemento
print("Lista despues de a adir:", lista)

# Eliminar un elemento
del lista[1]

# Mostrar la lista despues de eliminar un elemento
print("Lista despu s de eliminar:", lista)
```

3.1.4. Ejercicio 7

```python
agenda_telefonica = {}

# Funcion para agregar un contacto a la agenda
def agregar_contacto(nombre, telefono):
    agenda_telefonica[nombre] = telefono
    print(f"Contacto '{nombre}' agregado correctamente con el numero '{telefono}'.")

# Funcion para buscar un contacto en la agenda
def buscar_contacto(nombre):
    if nombre in agenda_telefonica:
        print(f"Nombre: {nombre} - Telefono: {agenda_telefonica[nombre]}")
    else:
        print(f"El contacto '{nombre}' no se encuentra en la agenda.")

# Funcion para mostrar todos los contactos en la agenda
def mostrar_contactos():
    if agenda_telefonica:
        print("Lista de contactos en la agenda:")
        for nombre, telefono in agenda_telefonica.items():
            print(f"Nombre: {nombre} - Telefono: {telefono}")
    else:
        print("La agenda esta vacia.")

# Ejemplo de uso de la agenda telefonica
agregar_contacto("Juan", "123456789")
agregar_contacto("Maria", "987654321")
agregar_contacto("Pedro", "555123456")

# Mostrar todos los contactos
mostrar_contactos()

# Buscar un contacto
buscar_contacto("Maria")
buscar_contacto("Carlos")  # Ejemplo de contacto que no existe en la agenda
```

3.1.5. Ejercicio 11

```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def mostrar_datos(self):
        print(f'Nombre: {self.nombre}, Edad: {self.edad}')

# Ejemplo de uso de la clase Persona
if __name__ == "__main__":
    # Crear una instancia de Persona
    persona1 = Persona("Fabrizzio", 22)

    # Mostrar los datos de la persona1
    persona1.mostrar_datos()
```

3.2. Parte II

3.2.1. Ejercicio 1

```python
# Funcion para realizar la operacion de suma
def suma(a, b):
    return a + b

# Funcion para realizar la operacion de resta
def resta(a, b):
    return a - b

# Funcion para realizar la operacion de multiplicaci n
def multiplicacion(a, b):
    return a * b

# Funcion para realizar la operacion de divisi n
def division(a, b):
    # Manejo de divisi n entre cero
    if b == 0:
        return "Error: divisi n entre cero"
    else:
        return a / b

# Funcion principal que ejecuta la calculadora
def calculadora():
    while True:
        print("\nCalculadora b sica")
        print("Seleccione la operaci n:")
        print("1. Suma")
        print("2. Resta")
        print("3. Multiplicaci n")
        print("4. Divisi n")
        print("5. Salir")

        opcion = input("Ingrese la opcion (1/2/3/4/5): ")

        if opcion == '5':
            print(" Hasta luego!")
            break

        if opcion in ('1', '2', '3', '4'):
            num1 = float(input("Ingrese el primer numero: "))
            num2 = float(input("Ingrese el segundo numero: "))

            if opcion == '1':
                print(f"{num1} + {num2} = {suma(num1, num2)}")
            elif opcion == '2':
                print(f"{num1} - {num2} = {resta(num1, num2)}")
            elif opcion == '3':
                print(f"{num1} * {num2} = {multiplicacion(num1, num2)}")
            elif opcion == '4':
                print(f"{num1} / {num2} = {division(num1, num2)}")
        else:
            print("Opcion invalida. Por favor, ingrese una opcion valida (1/2/3/4/5).")

# Llamada a la funcion principal de la calculadora
calculadora()
```

3.2.2. Ejercicio 2

```python
import random

def jugar_adivina_numero():
    print(" Bienvenido al juego de adivinar el numero!")
    print("Estoy pensando en un numero entre 1 y 10")
    numero_secreto = random.randint(1, 10)
    intentos_realizados = 0

    while True:
        intento = int(input("Intenta adivinar el numero: "))
        intentos_realizados += 1

        if intento < numero_secreto:
            print("El numero secreto es mayor. Intenta de nuevo.")
        elif intento > numero_secreto:
            print("El numero secreto es menor. Intenta de nuevo.")
        else:
            print(f"Felicitaciones! Adivinaste el numero secreto {numero_secreto} en {intentos_realizados} intentos!")
            break

    volver_a_jugar = input(" Quieres jugar de nuevo? (s/n): ").lower()
    if volver_a_jugar == 's':
        jugar_adivina_numero()
    else:
        print("Gracias por jugar. Hasta luego!")

# Iniciar el juego
jugar_adivina_numero()
```

3.2.3. Ejercicio 3

```python
# Definimos las tasas de cambio con respecto al PEN (SOl PERUANO)
#Diccionario
cambio_dicc = {
    'PEN': 1.0,    # Sol peruano
    'EUR': 0.24,   # Euro
    'CAD': 0.36,  # Yen Japones
    'GBP': 0.20,   # Libra Esterlina
    'MXN': 4.77    # Peso Mexicano
}

def conversor(cantidad, moneda_origen, moneda_destino):
    """Convierte una cantidad de una moneda a otra usando las tasas de cambio predefinidas."""
    if moneda_origen not in cambio_dicc or moneda_destino not in cambio_dicc:
        raise ValueError("Moneda no soportada.")

    # Convertir la cantidad a PEN primero
    cantidad_pen = cantidad / cambio_dicc[moneda_origen]

    # Convertir de PEN a la moneda destino
    conversion_cantidad = cantidad_pen * cambio_dicc[moneda_destino]

    return conversion_cantidad

def main():
    print("Conversor de Monedas")
    print("Monedas soportadas: PEN, EUR, JPY, GBP, MXN")

    moneda_origen = input("Introduce la moneda de origen: ").upper()
    moneda_destino = input("Introduce la moneda de destino: ").upper()
#aqui verificamos que el monto sea un numero
    while True:
        cantidad_input = input("Introduce la cantidad a convertir: ")
        try:
            cantidad = float(cantidad_input)
            break
        except ValueError:
```

```
            print("Error: La cantidad debe ser un n m e r o valido.")
#
    try:
        resultado = conversor(cantidad, moneda_origen, moneda_destino)
        print(f"{cantidad} {moneda_origen} son {resultado:.2f} {moneda_destino}.")
    except ValueError as e:
        print(e)

#Esta ultima linea  asegura que la funcion main()
# se ejecute solo cuando el script se ejecuta directamente.
if __name__ == "__main__":
    main()
```

3.2.4. Ejercicio 4

```
class Tarea:
    def __init__(self, descripcion):

        #Inicializa una nueva tarea con la descripcion proporcionada y la marca como no
            completada.

        self.descripcion = descripcion
        self.completada = False

    def marcar_completada(self):

        #Marca la tarea como completada.

        self.completada = True

    def __str__(self):

        #Devuelve una representaci n en cadena de la tarea, indicando si esta completada
             o no.

        estado = "Hecha" if self.completada else "Pendiente"
        return f"{self.descripcion} - {estado}"


class GestorTareas:
    def __init__(self):

        #Inicializa el gestor de tareas con una lista vacia de tareas.

        self.tareas = []

    def agregar_tarea(self, descripcion):

        #Agrega una nueva tarea con la descripcion proporcionada a la lista de tareas.
        #Devuelve un mensaje de confirmacion.

        tarea = Tarea(descripcion)
        self.tareas.append(tarea)
        return f"Tarea agregada con exito: {descripcion}"

    def eliminar_tarea(self, indice):

        #Elimina la tarea en el indice proporcionado de la lista de tareas.
        #Devuelve un mensaje de confirmacion si la operacion es exitosa,
        #o un mensaje de error si el indice es invalido.

        if 0 <= indice < len(self.tareas):
            tarea_eliminada = self.tareas.pop(indice)
            return f"Tarea eliminada con exito: {tarea_eliminada.descripcion}"
        else:
            return "indice de tarea invalido"

    def marcar_tarea_completada(self, indice):

        #Marca la tarea en el indice proporcionado como completada.
        #Devuelve un mensaje de confirmacion si la operacion es exitosa,
        #o un mensaje de error si el indice es invalido.
```

```
56
57          if 0 <= indice < len(self.tareas):
58              self.tareas[indice].marcar_completada()
59              return f"Tarea marcada como completada: {self.tareas[indice].descripcion}"
60          else:
61              return " ndice de tarea invalido"
62
63      def listar_tareas(self):
64
65          #Devuelve una lista en cadena de todas las tareas y sus estados.
66          #Si no hay tareas, devuelve un mensaje indicando que no hay tareas disponibles.
67
68          if not self.tareas:
69              return "No hay tareas disponibles"
70          return "\n".join([f"{idx}. {tarea}" for idx, tarea in enumerate(self.tareas)])
71
72
73  def main():
74
75      #Funcion principal que maneja la interaccion del usuario con el gestor de tareas.
76      #Muestra un menu y permite agregar, eliminar, marcar como completadas y listar tareas
            .
77
78      gestor = GestorTareas()
79      while True:
80          # Muestra el menu de opciones
81          print("\nOpciones:")
82          print("1. Agregar tarea")
83          print("2. Eliminar tarea")
84          print("3. Marcar tarea como completada")
85          print("4. Listar tareas")
86          print("5. Salir")
87          opcion = input("Elige una opci n: ")
88
89          if opcion == '1':
90              # Agregar una nueva tarea
91              descripcion = input("Descripcion de la tarea: ")
92              mensaje = gestor.agregar_tarea(descripcion)
93              print(mensaje)
94          elif opcion == '2':
95              # Eliminar una tarea existente
96              indice = int(input("Indice de la tarea a eliminar: "))
97              mensaje = gestor.eliminar_tarea(indice)
98              print(mensaje)
99          elif opcion == '3':
100             # Marcar una tarea como completada
101             indice = int(input("Indice de la tarea a marcar como completada: "))
102             mensaje = gestor.marcar_tarea_completada(indice)
103             print(mensaje)
104         elif opcion == '4':
105             # Listar todas las tareas
106             mensaje = gestor.listar_tareas()
107             print(mensaje)
108         elif opcion == '5':
109             # Salir del programa
110             break
111         else:
112             # Manejo de opcion invalida
113             print("Opcion invalida")
114
115 if __name__ == "__main__":
116     main()
```

3.3. Parte III

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Crear un conjunto de datos simple
data = {
    'tenure': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'monthlycharges': [20, 30, 40, 50, 60, 70, 80, 90, 100, 110],
    'churn': [0, 0, 0, 1, 0, 1, 1, 0, 1, 1]
}

df = pd.DataFrame(data)

# Dividir el conjunto de datos en entrenamiento y prueba
X = df[['tenure', 'monthlycharges']]
y = df['churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Entrenar el modelo de regresi n  log stica
model = LogisticRegression()
model.fit(X_train, y_train)

# Hacer predicciones
y_pred = model.predict(X_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Mostrar las probabilidades predichas
print('Probabilidades predichas:', model.predict_proba(X_test))
```

4. Conclusiones

Python se ha consolidado como uno de los lenguajes de programacin ms verstiles y populares en la actualidad. Su diseo enfocado en la simplicidad y la legibilidad lo hace accesible tanto para principiantes como para programadores experimentados. Las caractersticas clave de Python, como el tipado dinmico, la gestin automtica de memoria y las estructuras de datos incorporadas, contribuyen a su facilidad de uso y eficiencia.

La extensa biblioteca estndar de Python y su ecosistema de libreras de terceros permiten a los desarrolladores abordar una amplia gama de aplicaciones, desde desarrollo web y automatizacin de tareas hasta anlisis de datos y aprendizaje automtico. Adems, las herramientas y entornos de desarrollo compatibles con Python facilitan la escritura, depuracin y mantenimiento del cdigo.

El manejo robusto de excepciones y la flexibilidad en la definicin de funciones permiten a los desarrolladores escribir cdigo robusto y reutilizable. La capacidad de Python para integrarse con otros lenguajes y sistemas, junto con su soporte multiplataforma, asegura su relevancia en diversos entornos y aplicaciones.

La comunidad activa y vibrante de Python, junto con su enfoque en la colaboracin y el cdigo abierto, ha impulsado el desarrollo continuo y la mejora del lenguaje. Esto se refleja en la cantidad de recursos educativos, proyectos open source y eventos comunitarios disponibles para los desarrolladores de Python.

En resumen, Python no solo es un lenguaje poderoso y flexible, sino tambin una herramienta esencial en el arsenal de cualquier desarrollador moderno. Su combinacin de simplicidad, versatilidad y una comunidad de apoyo lo convierte en una eleccin ideal para una amplia gama de proyectos y desafos tecnolgicos.