Proyecto final Fundamentos

Bejarano Mayta Andrea
Escuela de Física
Universidad Nacional de Ingeniería
Lima, Perú
andrea.bejarano.m@uni.pe

Reyna Muñoz Paul
Escuela de Física
Universidad Nacional de Ingeniería
Lima, Perú
paul.reyna.m@uni.pe

Ruiz Elias Arom
Escuela de Física
Universidad Nacional de Ingeniería
Lima, Perú
arom.ruiz.e@uni.pe

***Resumen*—Este informe presenta una serie de problemas resueltos utilizando el lenguaje de programación Python. El documento completo cubre varios conceptos y técnicas fundamentales de Python, incluidas funciones, cadenas, archivos, diccionarios, clases, además de un pequeño uso de las librerías numpy, matplotlib y pandas para el manejo de datos. Cada pregunta está diseñada para aumentar la comprensión de los conceptos y va acompañada de una explicación detallada del método utilizado para obtener la solución. Además, cada solución incluye el código fuente completo, lo que proporciona una guía práctica para cualquiera que busque mejorar sus habilidades de programación en Python. Este informe está dirigido a estudiantes y profesionales que quieran profundizar en el desarrollo de soluciones algorítmicas de alto rendimiento utilizando Python.**

I. Introducción

En el presente informe se abordan una serie de ejercicios resueltos utilizando Python, enfocándose en diversas áreas clave de este lenguaje de programación. A lo largo del documento, se explorarán conceptos fundamentales como el uso de funciones, manipulación de cadenas, gestión de archivos, manejo de diccionarios, y la programación orientada a objetos con clases, además, se incluirán secciones dedicadas a la generación de números aleatorios y al análisis de datos utilizando la biblioteca pandas.

Python es conocido por su simplicidad y legibilidad, lo que lo convierte en una excelente opción tanto para principiantes como para desarrolladores experimentados. A diferencia de C++, Python es un lenguaje interpretado y de tipado dinámico, lo que significa que no requiere la declaración explícita de tipos de variables y permite una sintaxis más concisa y fácil de leer. En comparación, C++ es un lenguaje compilado y de tipado estático, lo que puede resultar en un código más complejo y detallado, pero también en una ejecución más rápida y eficiente en términos de rendimiento.

Las funciones en Python permiten la modularización y organización del código. Se definen con la palabra clave `def`, a diferencia de C++, donde se requiere una definición explícita del tipo de retorno y de los parámetros. La manipulación de cadenas en Python es simple gracias a sus métodos integrados, mientras que en C++ puede ser más complejo debido al uso de punteros y bibliotecas adicionales como `string.h`. En cuanto a la gestión de archivos, Python facilita las operaciones con la declaración `with` y el manejo automático de la apertura y cierre de archivos. En C++, se utilizan las clases `ifstream` y `ofstream` de la biblioteca estándar, que requieren un manejo más explícito.

El manejo de diccionarios en Python, que son equivalentes a los mapas en C++ (`std::map`), es más intuitivo y sencillo. Python soporta la programación orientada a objetos de manera accesible, con una sintaxis clara para la definición de clases y la implementación de herencia múltiple y métodos especiales. En C++, aunque también se soporta la orientación a objetos, la sintaxis puede ser más compleja y detallada. La generación de números aleatorios en Python es sencilla con el módulo `random`, mientras que en C++ se utiliza la biblioteca `<random>`, que puede ser más complicada de manejar. El análisis de datos con pandas en Python es extremadamente poderoso y flexible, y no tiene una biblioteca estándar equivalente en C++, lo que hace a Python la opción preferida para tareas de análisis de datos debido a su simplicidad y eficacia.

Este informe no solo busca proporcionar una comprensión teórica de los temas abordados, sino también ilustrar las diferencias prácticas y sintácticas entre Python y C++. A través de la combinación de teoría y práctica, se espera que el lector desarrolle una comprensión sólida y habilidades aplicables en el desarrollo de software y análisis de datos con Python, apreciando también las ventajas y desventajas de Python en comparación con C++ en distintos contextos.

II. Marco Teórico

II-A. Funciones en Python

Las funciones en Python son bloques de código reutilizables que permiten la modularización y la organización del programa. Se definen utilizando la palabra clave `def` seguida del nombre de la función y paréntesis que pueden incluir parámetros. Las funciones pueden devolver valores usando la palabra clave `return`. El uso de funciones mejora la legibilidad del código y facilita su mantenimiento y reutilización.

```
def funcion(parametro1, parametro2):
```

```
    resultado = parametro1 + parametro2
    return resultado
```

Además de las funciones definidas por el usuario, Python cuenta con una amplia gama de funciones integradas que permiten realizar operaciones comunes de forma eficiente, como `len()`, `sum()`, `max()`, entre otras.

II-B. Cadenas en Python

Las cadenas (o strings) son secuencias de caracteres y se representan utilizando comillas simples o dobles. Python ofrece una amplia variedad de métodos para manipular cadenas, como `split()`, que divide una cadena en una lista de subcadenas, `join()`, que une una lista de subcadenas en una sola cadena, y `replace()`, que permite reemplazar partes de la cadena con otra subcadena. La manipulación de cadenas es esencial en el procesamiento de textos y en la construcción de interfaces de usuario.

```
cadena = "Hola mundo"
print(cadena.upper())  # "HOLA MUNDO"
```

II-C. Archivos en Python

Python permite la manipulación de archivos mediante el uso de funciones integradas como `open()`. Los archivos pueden abrirse en diferentes modos, como lectura (`r`), escritura (`w`) y adición (`a`). La lectura de archivos permite extraer datos almacenados, mientras que la escritura y la adición permiten almacenar datos de manera persistente.

```
with open('archivo.txt', 'r') as archivo:
    contenido = archivo.read()
```

El uso de la declaración `with` asegura que el archivo se cierre correctamente después de que se complete la operación, lo que previene posibles errores y fugas de memoria.

II-D. Diccionarios en Python

Los diccionarios son colecciones desordenadas de pares clave-valor. Se definen utilizando llaves ({}) y permiten el acceso rápido a los valores mediante sus claves. Los diccionarios son extremadamente útiles para almacenar datos relacionados, como la información de un usuario o las configuraciones de una aplicación.

```
diccionario = {'clave1': 'valor1',
'clave2': 'valor2'}
print(diccionario['clave1'])  # "valor1"
```

Python ofrece varios métodos para manipular diccionarios, como `keys()`, `values()` y `items()`, que permiten acceder a las claves, valores y pares clave-valor del diccionario, respectivamente.

II-E. Clases en Python

Python es un lenguaje orientado a objetos y permite la creación de clases, que son plantillas para crear objetos. Las clases encapsulan datos y comportamientos en una sola estructura. Se definen utilizando la palabra clave `class`. El uso de clases facilita la creación de programas más complejos y estructurados.

```
class Clase:
    def __init__(self, atributo1):
        self.atributo1 = atributo1

    def metodo(self):
        return self.atributo1

objeto = Clase('valor')
print(objeto.metodo())  # "valor"
```

Las clases pueden tener métodos y atributos. Los métodos definen comportamientos y los atributos almacenan el estado de un objeto. Además, Python soporta herencia, lo que permite crear nuevas clases basadas en clases existentes, facilitando la reutilización y extensión del código.

II-F. Generación de Números Aleatorios en Python

La generación de números aleatorios en Python se logra utilizando el módulo `random`, que proporciona una variedad de funciones para generar números aleatorios. Estas funciones pueden ser utilizadas en aplicaciones que requieren elementos de aleatoriedad, como simulaciones, juegos y pruebas estadísticas.

```
import random

numero_aleatorio = random.randint(1, 10)
print(numero_aleatorio)  # Número entero
aleatorio entre 1 y 10
```

El módulo `random` también incluye funciones como `random()`, que genera un número de punto flotante aleatorio entre 0 y 1, y `choice()`, que selecciona un elemento aleatorio de una secuencia.

II-G. Análisis de Datos con pandas

Pandas es una biblioteca poderosa y flexible para el análisis de datos en Python. Proporciona estructuras de datos fáciles de usar y de alto rendimiento, como DataFrame, que permite la manipulación y el análisis de datos tabulares. Con pandas, es posible realizar operaciones como la limpieza de datos, la transformación de datos y el análisis exploratorio de datos.

```
import pandas as pd

datos = {'Nombre': ['Ana', 'Luis', 'Pedro'
], 'Edad': [28, 34, 29]}
df = pd.DataFrame(datos)
```

```
print(df)
```

Pandas ofrece una amplia gama de funciones y métodos para manipular datos, como `groupby()`, que permite agrupar datos, y `merge()`, que permite combinar DataFrames. Estas capacidades hacen de pandas una herramienta esencial para el análisis de datos en Python.

III. Desarrollo

A continuación el desarrollo

III-A. Parte 1-Funciones

En esta sección se presenta un código en Python que calcula el factorial de un número ingresado por el usuario utilizando una función recursiva. A continuación se describe y explica cada parte del código.

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Se define una función llamada `factorial` que toma un parámetro `n`. La función utiliza un enfoque recursivo para calcular el factorial del número. La condición base de la recursión es que si `n` es igual a 0, la función retorna 1. Esta condición se basa en el hecho de que el factorial de 0 es 1 ($0! = 1$). Si `n` no es 0, la función retorna el producto de `n` y la llamada recursiva a `factorial(n-1)`, lo que efectivamente descompone el problema en subproblemas más pequeños hasta alcanzar la condición base.

```
n = input("Enter a number: ")
n = int(n)
```

El programa solicita al usuario que ingrese un número utilizando la función `input()`. La entrada del usuario se convierte a un entero utilizando la función `int()`, ya que la función `input()` retorna una cadena de caracteres.

```
k = factorial(n)
print(k)
```

La variable `k` almacena el resultado de la llamada a la función `factorial()` con el número ingresado por el usuario. Finalmente, el programa imprime el valor de `k`, que es el factorial del número ingresado.

Este código es un ejemplo clásico de cómo se puede utilizar la recursión para resolver problemas en programación. La recursión es una técnica en la que una función se llama a sí misma para resolver subproblemas similares al problema original. En este caso, el cálculo del factorial de un número es una aplicación directa de la recursión. El uso de la recursión hace que el código sea más conciso y fácil de entender, aunque es importante tener en cuenta las limitaciones de recursión, como el límite de profundidad de recursión en Python, que puede ser un factor limitante para números muy grandes.

El código sigue un flujo lógico claro: solicita al usuario un número, calcula el factorial de ese número utilizando una función recursiva, y luego imprime el resultado. Esto ilustra cómo se puede combinar la entrada del usuario, el procesamiento recursivo y la salida de datos en un programa sencillo pero efectivo en Python.

III-B. Parte 1-Cadenas

En esta sección se presenta un código en Python que invierte una cadena de caracteres ingresada, utilizando una función iterativa. A continuación se describe y explica cada parte del código.

```
def invertir_cadena(cadena):
    cadena_invertida = ""
    for caracter in cadena:
        cadena_invertida = caracter +
        cadena_invertida
    return cadena_invertida
```

Se define una función llamada `invertir_cadena` que toma un parámetro `cadena`. Dentro de la función, se inicializa una variable `cadena_invertida` como una cadena vacía. Luego, se utiliza un bucle `for` para iterar a través de cada carácter en la cadena original. En cada iteración, el carácter actual (`caracter`) se antepone a `cadena_invertida`. De esta forma, se construye la cadena invertida carácter por carácter. Finalmente, la función retorna la cadena invertida.

```
# Ejemplo de uso
cadena_original = "Hola, mundo!"
cadena_invertida = invertir_cadena
(cadena_original)
print(cadena_invertida)
```

Se proporciona un ejemplo de uso de la función `invertir_cadena`. Primero, se define una variable `cadena_original` con el valor "Hola, mundo!". Luego, se llama a la función `invertir_cadena` con `cadena_original` como argumento y se asigna el resultado a la variable `cadena_invertida`. Finalmente, se imprime el valor de `cadena_invertida`.

El código ilustra cómo se puede invertir una cadena de caracteres utilizando un enfoque iterativo. La técnica empleada es simple y eficaz: iterar a través de cada carácter de la cadena original y anteponerlo a la cadena invertida. Este enfoque garantiza que el primer carácter de la cadena original se convierta en el último carácter de la cadena invertida, y así sucesivamente.

Este método es eficiente en términos de comprensión y es fácilmente adaptable para cualquier cadena de caracteres. La función `invertir_cadena` no depende de la longitud de la cadena, por lo que puede manejar cadenas de cualquier tamaño, limitadas solo por la memoria disponible en el sistema.

Además, este código destaca el uso de conceptos básicos de programación como la iteración y la manipulación de cadenas, proporcionando una solución clara y concisa para el problema de invertir una cadena. El enfoque iterativo utilizado aquí es particularmente útil en situaciones donde la recursión puede no ser la opción más adecuada debido a limitaciones de profundidad de recursión en algunos lenguajes de programación.

III-C. Parte 1-Diccionarios

En esta sección se presenta un código en Python que gestiona una agenda telefónica utilizando un diccionario. El programa permite agregar, eliminar, buscar y mostrar contactos con sus respectivos números de teléfono. A continuación se describe y explica cada parte del código.

```
agendaTelefonica = {}
```

Se inicializa un diccionario vacío llamado `agendaTelefonica`. Este diccionario almacenará los nombres de los contactos como claves y sus números de teléfono como valores.

```
def agregar(nombre, telefono):
    agendaTelefonica[nombre] = telefono
    print(f"{nombre} añadido, número de
    teléfono: {telefono}.")
```

Se define una función `agregar` que toma dos parámetros: `nombre` y `telefono`. La función añade un nuevo contacto a `agendaTelefonica` asignando el número de teléfono al nombre proporcionado y luego imprime un mensaje de confirmación.

```
def eliminar(nombre):
    if nombre in agendaTelefonica:
        del agendaTelefonica[nombre]
        print(f"Teléfono de {nombre}
        eliminado")
    else:
        print(f"No se encontró al contacto")
```

Se define una función `eliminar` que toma un parámetro `nombre`. La función verifica si el nombre existe en `agendaTelefonica`. Si existe, elimina el contacto y muestra un mensaje de confirmación; de lo contrario, informa que el contacto no se encontró.

```
def buscar(nombre):
    if nombre in agendaTelefonica:
        print(f"{nombre}: {agendaTelefonica
        [nombre]}")
    else:
        print(f"No se encontró al contacto")
```

Se define una función `buscar` que toma un parámetro `nombre`. La función verifica si el nombre existe en `agendaTelefonica`. Si existe, imprime el nombre y su número de teléfono asociado; de lo contrario, informa que el contacto no se encontró.

```
def mostrar():
    print("Agenda telefónica:")
    for nombre, telefono in
    agendaTelefonica.items():
        print(f"{nombre}: {telefono}")
```

Se define una función `mostrar` que no toma parámetros. La función imprime todos los contactos almacenados en `agendaTelefonica`. Utiliza un bucle `for` para iterar a través de los elementos del diccionario, imprimiendo cada nombre y su número de teléfono asociado.

```
agregar("Carmen", 941742631)
agregar("Theodoro", 946233559)
buscar("Theodoro")
mostrar()
eliminar("Carmen")
buscar("Carmen")
mostrar()
```

Se ejemplifica el uso de las funciones definidas anteriormente. Primero, se añaden dos contactos: Çarmençon el número 941742631 y ”Theodoroçon el número 946233559. Luego, se busca el contacto ”Theodoroz se muestra toda la agenda telefónica. A continuación, se elimina el contacto Çarmenz se intenta buscar nuevamente, mostrando que ya no está en la agenda. Finalmente, se muestra la agenda actualizada.

Este código ilustra cómo se puede gestionar una agenda telefónica utilizando un diccionario en Python. Los diccionarios son estructuras de datos muy útiles para almacenar y manipular pares clave-valor de manera eficiente. El código también demuestra el uso de funciones para organizar y modularizar las operaciones, haciendo que la gestión de los contactos sea clara y estructurada. El enfoque presentado permite realizar operaciones básicas de CRUD (Crear, Leer, Actualizar y Eliminar) de manera sencilla y efectiva.

III-D. Parte 1-Clases

En esta sección se presenta un código en Python que define una clase `Persona` y crea instancias de dicha clase. Cada instancia representa una persona con un nombre y una edad. Además, la clase incluye un método para mostrar la información de la persona. A continuación se describe y explica cada parte del código.

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

    def mostrar(self):
        print(f"Nombre: {self.nombre} -
        Edad: {self.edad}")
```

Se define una clase llamada `Persona`. Dentro de esta clase, se define el método `__init__`, que es el cons-

tructor de la clase. Este método toma dos parámetros adicionales a `self`: `nombre` y `edad`. El constructor inicializa los atributos `nombre` y `edad` de la instancia con los valores proporcionados.

También se define un método `mostrar` que imprime el nombre y la edad de la persona en el formato "Nombre: `self.nombre` - Edad: `self.edad`".

```
persona1 = Persona("Arom", 18)
persona2 = Persona("Paul", 20)
persona3 = Persona("Andrea", 20)
```

Se crean tres instancias de la clase `Persona` con los nombres y edades especificados. La variable `persona1` es una instancia de `Persona` con el nombre .Aromz la edad 18. De manera similar, `persona2` y `persona3` son instancias con los nombres "Paulz .Andrea", y ambas con la edad 20.

```
persona1.mostrar()
persona2.mostrar()
persona3.mostrar()
```

Se llama al método `mostrar` para cada una de las instancias creadas. Esto imprime la información de cada persona en la consola.

El uso de clases y objetos en Python permite organizar el código de manera modular y reutilizable. La definición de la clase `Persona` encapsula los datos y las funciones relacionadas en una sola entidad. Este enfoque es fundamental en la programación orientada a objetos (POO), donde los objetos son instancias de clases y representan entidades del mundo real o conceptos abstractos.

En este código, se demuestra cómo definir una clase con un constructor y métodos, cómo crear instancias de la clase, y cómo interactuar con esas instancias llamando a sus métodos. La POO es una herramienta poderosa en la programación que facilita la creación de programas más grandes y manejables mediante la encapsulación, la herencia y el polimorfismo.

El código presentado es un ejemplo básico pero completo de cómo utilizar la POO en Python, proporcionando una base sólida para aplicaciones más complejas que requieran la modelación de entidades y sus interacciones.

III-E. Parte 1-Archivos

En esta sección se presenta un código en Python que escribe varias líneas de texto en un archivo. El programa utiliza la estructura `with open` para manejar el archivo, asegurando una correcta apertura y cierre del mismo. A continuación se describe y explica cada parte del código.

```
with open('ejemplo.txt', 'w') as archivo:
    archivo.write('Este es el grupo\n')
    archivo.write('para la práctica 5 de\n')
    archivo.write('Fundamentos de
    Programación')
```

Se utiliza la estructura `with open` para abrir (o crear si no existe) un archivo llamado `ejemplo.txt` en modo de escritura ('w'). La estructura `with` asegura que el archivo se cierre correctamente después de que se ejecuten las operaciones dentro de su bloque, incluso si ocurre una excepción.

Dentro del bloque `with`, se llama al método `write` del objeto archivo para escribir texto en el archivo. Cada llamada a `write` agrega una línea de texto al archivo. La secuencia de operaciones es la siguiente:

1. `archivo.write('Este es el grupo')` escribe la línea .Este es el grupo"seguida de un salto de línea (). 2. `archivo.write('para la práctica 5 de')` escribe la línea "para la práctica 5 de"seguida de un salto de línea. 3. `archivo.write('Fundamentos de Programación')` escribe la línea "Fundamentos de Programación".

El uso del modo 'w' asegura que si `ejemplo.txt` ya existe, su contenido será sobrescrito. Si se quisiera añadir contenido sin sobrescribir el existente, se debería utilizar el modo 'a' (apéndice).

Este código demuestra cómo manejar archivos en Python de manera segura y eficiente. La estructura `with` es preferida sobre el uso de `open` y `close` debido a su capacidad para manejar automáticamente el cierre del archivo, lo cual es crucial para evitar fugas de recursos y asegurar que los datos se escriban correctamente.

Además, este ejemplo ilustra la simplicidad y legibilidad del manejo de archivos en Python, permitiendo al programador enfocarse en la lógica del programa sin preocuparse por detalles de bajo nivel relacionados con la gestión de recursos. La escritura en archivos es una operación fundamental en muchos programas, utilizada para persistir datos, generar reportes, y más.

III-F. Parte 2

El siguiente fragmento de código en Python implementa un sencillo juego de adivinanza de números. El programa genera un número aleatorio entre 1 y 100 y pide al usuario que lo adivine. A continuación, se describe y explica el funcionamiento de cada parte del código:

```
import random
```

La primera línea importa el módulo `random`, que permite generar números aleatorios. Este módulo es esencial para el funcionamiento del juego, ya que el número a adivinar será generado de forma aleatoria.

```
num = random.randint(1, 100)
numintentos = 0
```

Se genera un número aleatorio entre 1 y 100 utilizando la función `randint` del módulo `random`, y se asigna a la variable `num`. También se inicializa la variable `numintentos` a 0, que servirá para contar el número de intentos realizados por el usuario.

```
print("Adivine un número entero entre 1 y
100")
```

Se imprime un mensaje en la consola para informar al usuario sobre el objetivo del juego.

```
while True:
  try:
    intento = int(input("Ingrese el número:
    "))
    numintentos += 1
    if intento < num:
      print("El número es mayor que el
      ingresado")
    elif intento > num:
      print("El número es menor que el
      ingresado")
    else:
      print(f"Felicidades, adivinaste el
      número {num} en {numintentos}
      intentos.")
      break
  except ValueError:
    print("Entrada inválida, ingrese un
    número entero")
```

El núcleo del programa es un bucle `while` que continúa ejecutándose hasta que el usuario adivina correctamente el número. Dentro del bucle, se utiliza un bloque `try-except` para manejar posibles errores de entrada del usuario.

En el bloque `try`, se lee la entrada del usuario con la función `input` y se convierte a un entero con `int`. Si la conversión tiene éxito, el contador de intentos (`numintentos`) se incrementa en uno. Luego, el programa compara el número ingresado por el usuario (`intento`) con el número a adivinar (`num`). Si el número ingresado es menor que `num`, se informa al usuario que el número a adivinar es mayor. Si el número ingresado es mayor que `num`, se informa que el número a adivinar es menor. Si el usuario adivina el número correctamente, se imprime un mensaje de felicitación indicando el número de intentos realizados y se rompe el bucle con `break`.

En el bloque `except`, se captura una excepción de tipo `ValueError`, que ocurre si el usuario ingresa algo que no puede convertirse a un número entero. En ese caso, se imprime un mensaje de error indicando que la entrada no es válida y el bucle continúa.

Este código es un ejemplo sencillo de cómo se pueden utilizar las estructuras de control de flujo, manejo de excepciones y funciones integradas de Python para crear una aplicación interactiva. Además, ilustra el uso del módulo `random` para generar datos aleatorios y cómo manejar la interacción con el usuario de manera robusta.

III-G. Parte 3

En esta sección se presenta un código en Python que realiza un análisis de datos utilizando las bibliotecas `pandas`, `numpy`, y `matplotlib`. El programa carga un conjunto de datos desde un archivo CSV, calcula estadísticas básicas, y genera un gráfico de dispersión. A continuación se describe y explica cada parte del código.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

Se importan las bibliotecas necesarias: `pandas` para la manipulación de datos, `matplotlib.pyplot` para la visualización, y `numpy` para cálculos numéricos.

```
df = pd.read_csv('sample_data/mtcars.csv')
print(df.head())
```

Se carga un archivo CSV llamado mtcars.csv en un DataFrame de `pandas` utilizando `pd.read_csv`. El método `read_csv` lee el archivo y lo almacena en la variable `df`. Luego, se imprime las primeras cinco filas del DataFrame para inspeccionar los datos utilizando `df.head()`.

```
columna1 = 'hp'
media_hp = np.mean(df[columna1])
desviacion_hp = np.std(df[columna1])
print(f"La media de {columna1} es {media_hp}
")
print(f"La desviación estándar de {columna1}
es {desviacion_hp}")
```

Se define la variable `columna1` con el valor 'hp' que representa la columna de caballos de fuerza (horsepower). Se calculan la media y la desviación estándar de esta columna utilizando `np.mean` y `np.std`, respectivamente. Los resultados se imprimen en la consola.

```
columna2 = 'cyl'
media_cyl = np.mean(df[columna2])
desviacion_cyl = np.std(df[columna2])
print(f"La media de {columna2} es {media_cyl}
")
print(f"La desviación estándar de {columna2}
es {desviacion_cyl}")
```

Se define la variable `columna2` con el valor 'cyl' que representa la columna de cilindros. De manera similar, se calculan la media y la desviación estándar de esta columna y se imprimen los resultados en la consola.

```
plt.scatter(df[columna2], df[columna1], s=8,
color='blue')
plt.title(f'Gráfico de dispersión de
{columna2} por {columna1}')
plt.xlabel(columna2)
plt.ylabel(columna1)
plt.tight_layout()
```

`plt.show()`

Se genera un gráfico de dispersión utilizando `plt.scatter`. En este gráfico, `columna2` (cilindros) se coloca en el eje x y `columna1` (caballos de fuerza) en el eje y. Los puntos del gráfico se muestran en color azul y con un tamaño de 8 (`s=8`). Se añade un título al gráfico, así como etiquetas a los ejes x e y. Finalmente, `plt.tight_layout()` ajusta el espaciado del gráfico y `plt.show()` muestra el gráfico.

Este código demuestra cómo cargar datos desde un archivo CSV, realizar cálculos estadísticos básicos y generar visualizaciones con `pandas`, `numpy`, y `matplotlib`. La combinación de estas bibliotecas permite realizar un análisis de datos completo y eficiente en Python. El gráfico de dispersión generado proporciona una visualización clara de la relación entre dos variables, facilitando la interpretación de los datos y la identificación de posibles patrones o tendencias.

IV. Conclusiones

En este informe cubre varios aspectos fundamentales de la programación en Python, abarcando desde la manipulación de cadenas hasta el análisis de datos y la visualización. A lo largo del informe, se han presentado ejemplos prácticos que ilustran el uso de diversas funcionalidades y bibliotecas de Python para resolver problemas específicos. Las conclusiones generales de este trabajo son las siguientes:

En primer lugar, se ha demostrado cómo Python facilita la manipulación de cadenas de caracteres a través de ejemplos como la inversión de una cadena. La simplicidad y legibilidad del código en Python permiten implementar soluciones efectivas y comprensibles incluso para tareas que requieren una manipulación detallada de los datos.

En segundo lugar, se ha destacado la importancia de las estructuras de datos en Python, como los diccionarios, a través de la implementación de una agenda telefónica. Los diccionarios proporcionan una forma eficiente de almacenar y acceder a pares clave-valor, lo cual es esencial para muchos tipos de aplicaciones.

Asimismo, se ha mostrado el poder de la programación orientada a objetos (POO) en Python mediante la definición de clases y la creación de instancias. El ejemplo de la clase `Persona` ilustra cómo la POO permite modelar entidades del mundo real de manera intuitiva, organizando el código de forma modular y reutilizable.

Además, el uso de archivos en Python se ha ejemplificado con la escritura en un archivo de texto. La estructura `with open` facilita el manejo seguro de archivos, asegurando su correcta apertura y cierre, lo que es crucial para la integridad de los datos.

Finalmente, se ha explorado el análisis de datos y la visualización utilizando las bibliotecas `pandas`, `numpy`, y `matplotlib`. Estos ejemplos han demostrado cómo Python puede manejar grandes conjuntos de datos, calcular estadísticas básicas y generar visualizaciones que ayudan a interpretar y comunicar los resultados de manera efectiva.

En comparación con otros lenguajes de programación como C++, Python ofrece una sintaxis más sencilla y una biblioteca estándar más rica, lo que reduce significativamente el tiempo de desarrollo. Las capacidades de Python para la manipulación de datos y la visualización son particularmente destacables, facilitando el trabajo de los desarrolladores en áreas como el análisis de datos y la ciencia de datos.

En resumen, este informe ha proporcionado una visión integral de las capacidades de Python a través de ejemplos prácticos y concretos. La versatilidad de Python y su extensa biblioteca de módulos lo convierten en una herramienta poderosa para una amplia variedad de aplicaciones, desde la programación básica hasta el análisis avanzado de datos.