

C++ y Python: Una Breve Comparación

Colaborador 1: Jairo Jefferson Ortega Vega

Código: 20220089E

Escuela profesional de Ing. Física

Universidad Nacional

de Ingeniería

Lima, Perú

jairo.ortega.v@uni.pe

Colaborador 2: Ricardo Berrios Moya

Código: 20190543E

Escuela profesional de Física

Universidad Nacional

de Ingeniería

Lima, Perú

ricardo.berrios.m@uni.pe

Colaborador 3: Misael Urbano Cochachin

Código: 20221520A

Escuela profesional de Física

Universidad Nacional de Ingeniería

Lima, Perú

msael.urbano.c@uni.pe

Resumen

Este informe presenta una breve comparación entre Python y C++, destacando sus ventajas, desventajas y casos de uso. Ambos lenguajes son ampliamente utilizados en diversas áreas, pero ofrecen características muy diferentes que los hacen más adecuados para ciertos propósitos.

I. INTRODUCCIÓN

Python y C++ son dos lenguajes de programación populares, cada uno con fortalezas particulares. Python es conocido por su simplicidad y legibilidad, mientras que C++ destaca por su eficiencia y control sobre los recursos del sistema. Esta comparación tiene como objetivo proporcionar una visión general para ayudar a los desarrolladores a elegir el lenguaje adecuado según sus necesidades.

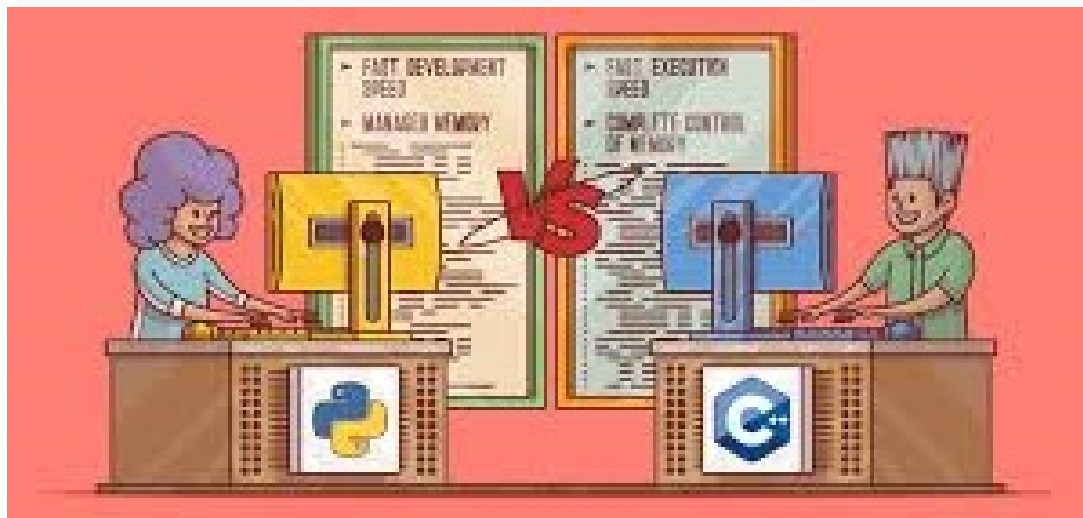


Figura 1. Imagen ilustrativa

II. VENTAJAS DE PYTHON

II-A. Simplicidad y legibilidad

Python está diseñado para ser fácil de leer y escribir. Su sintaxis simple reduce la curva de aprendizaje, lo que lo hace ideal para principiantes.

II-B. Amplia biblioteca estándar

Cuenta con una extensa colección de bibliotecas para tareas comunes, como manejo de datos, aprendizaje automático y desarrollo web.

II-C. Portabilidad

Python es multiplataforma, lo que significa que el código puede ejecutarse en varios sistemas operativos con pocas modificaciones.

II-D. Productividad y desarrollo rápido

La escritura de código en Python suele ser más rápida debido a su sintaxis concisa y al manejo automático de memoria.

III. VENTAJAS DE C++

III-A. Eficiencia y rendimiento

C++ permite un control detallado sobre el hardware, lo que lo hace extremadamente rápido y eficiente en términos de uso de recursos.

III-B. Programación de bajo nivel

Ofrece acceso directo a memoria y hardware, siendo ideal para sistemas embebidos y controladores.

III-C. Flexibilidad

Admite múltiples paradigmas de programación, como orientación a objetos, programación genérica y funcional.

III-D. Estándares y soporte de larga data

C++ ha estado en uso por décadas, lo que asegura estabilidad y un ecosistema maduro.

IV. COMPARATIVA

Característica	Python	C++
Facilidad de uso	Alta	Media
Rendimiento	Medio	Alto
Bibliotecas estándar	Amplias	Amplias
Control sobre hardware	Bajo	Alto
Adecuado para principiantes	Sí	No
Programación multiplataforma	Sí	Sí

Cuadro I
COMPARATIVA ENTRE PYTHON Y C++

V. CASOS DE USO

V-A. Python

Python es ideal para:

- Desarrollo de aplicaciones web.
- Análisis de datos y aprendizaje automático.
- Scripts de automatización.

V-B. C++

C++ es más adecuado para:

- Desarrollo de sistemas operativos.
- Motores de videojuegos.
- Aplicaciones en tiempo real y sistemas embebidos.

VI. MANEJO DE ARCHIVOS EN PYTHON VS. C++

El manejo de archivos es una funcionalidad común en muchos lenguajes de programación, y Python ofrece una interfaz sencilla y potente para trabajar con archivos, comparable a las operaciones en C++. En el siguiente análisis, describiremos las características principales del manejo de archivos en Python a través del código proporcionado y lo contrastaremos con la sintaxis de C++.

VI-A. Apertura y Cierre de Archivos

En Python, la función `open(nombre_archivo, modo)` permite abrir un archivo en diferentes modos: lectura ('r'), escritura ('w'), y anexar ('a'). Cada modo define cómo se accederá al archivo, de manera similar al flujo de archivos (`fstream`) en C++. En C++, se usan objetos de clases como `ifstream` (para lectura), `ofstream` (para escritura), y `fstream` (para lectura y escritura combinadas), donde también es importante especificar un modo al abrir el archivo.

Ejemplo en Python:

```
archivo = open("archivo.txt", 'r')
```

Ejemplo en C++:

```
std::ifstream archivo("archivo.txt");
```

VI-B. Lectura de Archivos

Python utiliza métodos como `read()` para leer todo el contenido de un archivo como una cadena, o `readline()` y `readlines()` para leer línea por línea. En C++, esto se logra típicamente con el operador de extracción (`>>`) o el método `getline()`.

Ejemplo en Python:

```
contenido = archivo.read()
```

Ejemplo en C++:

```
std::string contenido;  
std::getline(archivo, contenido);
```

VI-C. Escritura de Archivos

Para escribir en un archivo, Python usa el método `write()`, mientras que en C++, el operador de inserción (`<<`) y el método `write()` en flujos de salida (`ofstream`) permiten realizar esta tarea.

»Ejemplo en Python:

```
»archivo.write("Texto a escribir.\n")
```

»Ejemplo en C++:

```
»archivo << "Texto a escribir.\n";
```

»VI-D. Anexar Datos al Archivo

»Python ofrece el modo 'a' (anexar) para agregar contenido al final de un archivo existente, sin sobrescribir su contenido previo. En C++, esto se logra abriendo el archivo en modo `std::ios::app`.

»Ejemplo en Python:

```
»archivo = open("archivo.txt", 'a')  
»archivo.write("Contenido adicional.\n")
```

»Ejemplo en C++:

```
»std::ofstream archivo("archivo.txt", std::ios::app);  
»archivo << "Contenido adicional.\n";
```

»VI-E. Cierre de Archivos

»En ambos lenguajes, es crucial cerrar los archivos después de usarlos para liberar recursos. Python utiliza el método `close()`, mientras que en C++ el método `close()` de las clases de flujo cumple esta función. En Python, también se pueden usar bloques `with` para manejar automáticamente la apertura y el cierre de archivos.

»Ejemplo en Python:

```
»archivo.close()
```

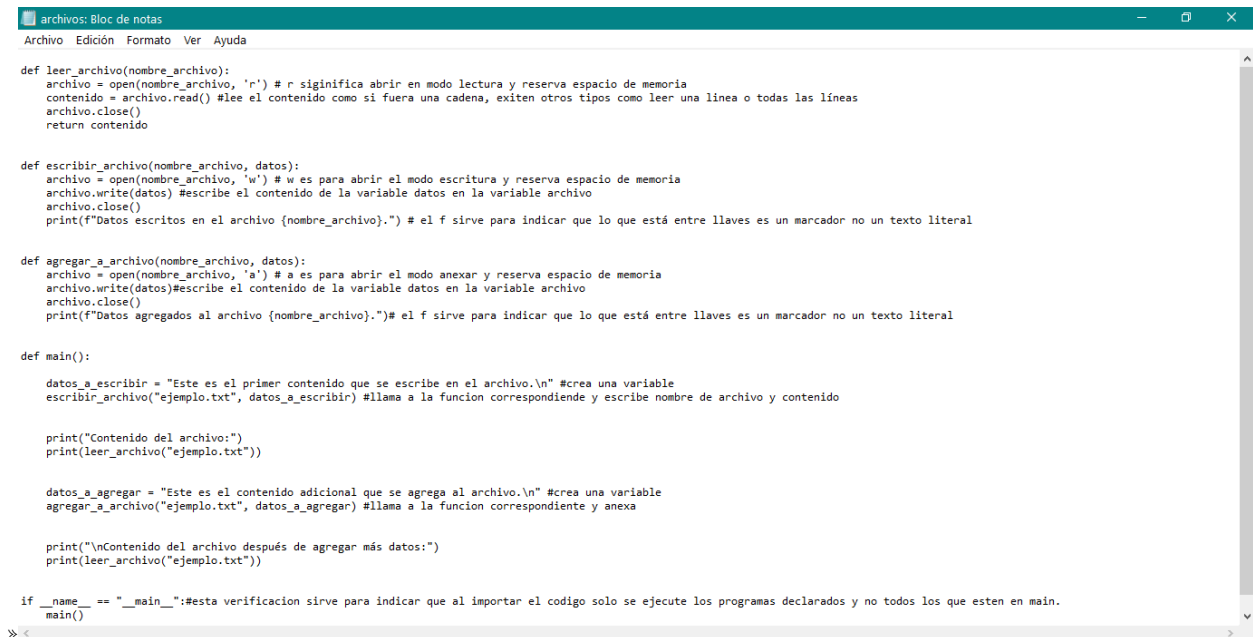
»Ejemplo en C++:

```
»archivo.close();
```

»VI-F. Estructura del Código y Modularidad

»En el código de Python, las funciones `leer_archivo`, `escribir_archivo`, y `agregar_a_archivo` están bien definidas y modularizadas, lo que facilita su reutilización. El programa incluye un bloque `if __name__ == "__main__":` que asegura que las funciones principales solo se ejecuten cuando el archivo se ejecuta directamente, sin interferir al importarlo en otro lugar. En C++, una estructura modular similar puede lograrse mediante funciones o clases, y no existe una necesidad directa de una verificación como `__main__`, ya que el punto de entrada es siempre la función `main()`.

»Este ejemplo destaca cómo Python simplifica las operaciones de manejo de archivos con su sintaxis compacta y métodos directos. En C++, si bien las operaciones requieren un enfoque más explícito, ofrecen un control más granular sobre las operaciones de archivo y el manejo de datos binarios.



```
archivos: Bloc de notas
Archivo Edición Formato Ver Ayuda

def leer_archivo(nombre_archivo):
    archivo = open(nombre_archivo, 'r') # r significa abrir en modo lectura y reserva espacio de memoria
    contenido = archivo.read() # lee el contenido como si fuera una cadena, existen otros tipos como leer una línea o todas las líneas
    archivo.close()
    return contenido

def escribir_archivo(nombre_archivo, datos):
    archivo = open(nombre_archivo, 'w') # w es para abrir el modo escritura y reserva espacio de memoria
    archivo.write(datos) # escribe el contenido de la variable datos en la variable archivo
    archivo.close()
    print(f"Datos escritos en el archivo {nombre_archivo}.") # el f sirve para indicar que lo que está entre llaves es un marcador no un texto literal

def agregar_a_archivo(nombre_archivo, datos):
    archivo = open(nombre_archivo, 'a') # a es para abrir el modo anexar y reserva espacio de memoria
    archivo.write(datos) # escribe el contenido de la variable datos en la variable archivo
    archivo.close()
    print(f"Datos agregados al archivo {nombre_archivo}.") # el f sirve para indicar que lo que está entre llaves es un marcador no un texto literal

def main():
    datos_a_escribir = "Este es el primer contenido que se escribe en el archivo.\n" # crea una variable
    escribir_archivo("ejemplo.txt", datos_a_escribir) # llama a la función correspondiente y escribe nombre de archivo y contenido

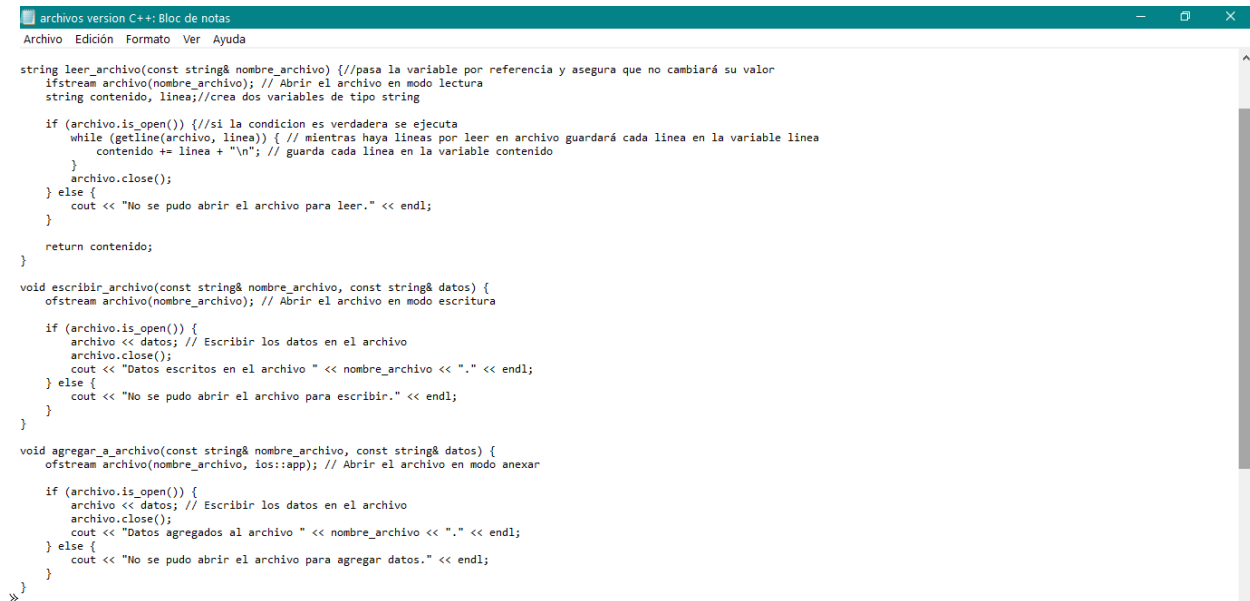
    print("Contenido del archivo:")
    print(leer_archivo("ejemplo.txt"))

    datos_a_agregar = "Este es el contenido adicional que se agrega al archivo.\n" # crea una variable
    agregar_a_archivo("ejemplo.txt", datos_a_agregar) # llama a la función correspondiente y anexa

    print("\nContenido del archivo después de agregar más datos:")
    print(leer_archivo("ejemplo.txt"))

if __name__ == "__main__": # esta verificación sirve para indicar que al importar el código solo se ejecute los programas declarados y no todos los que estén en main.
    main()
```

Figura 2. Código en lenguaje Python



```
string leer_archivo(const string& nombre_archivo) { //pasa la variable por referencia y asegura que no cambiará su valor
    ifstream archivo(nombre_archivo); // Abrir el archivo en modo lectura
    string contenido, linea; //crea dos variables de tipo string

    if (archivo.is_open()) { //si la condicion es verdadera se ejecuta
        while (getline(archivo, linea)) { // mientras haya lineas por leer en archivo guardará cada línea en la variable linea
            contenido += linea + "\n"; // guarda cada línea en la variable contenido
        }
        archivo.close();
    } else {
        cout << "No se pudo abrir el archivo para leer." << endl;
    }

    return contenido;
}

void escribir_archivo(const string& nombre_archivo, const string& datos) {
    ofstream archivo(nombre_archivo); // Abrir el archivo en modo escritura

    if (archivo.is_open()) {
        archivo << datos; // Escribir los datos en el archivo
        archivo.close();
        cout << "Datos escritos en el archivo " << nombre_archivo << "." << endl;
    } else {
        cout << "No se pudo abrir el archivo para escribir." << endl;
    }
}

void agregar_a_archivo(const string& nombre_archivo, const string& datos) {
    ofstream archivo(nombre_archivo, ios::app); // Abrir el archivo en modo anexar

    if (archivo.is_open()) {
        archivo << datos; // Escribir los datos en el archivo
        archivo.close();
        cout << "Datos agregados al archivo " << nombre_archivo << "." << endl;
    } else {
        cout << "No se pudo abrir el archivo para agregar datos." << endl;
    }
}
```

Figura 3. Código en lenguaje C++

»VII. SUMA DE ELEMENTOS EN UN ARREGLO EN PYTHON VS. C++

»En este ejemplo, se define un programa en Python que calcula la suma de los elementos de un arreglo ingresado por el usuario. A continuación, se describen las operaciones principales y se comparan con su equivalente en C++.

»VII-A. Definición de Funciones

»En Python, se define una función llamada `suma_elementos`, que toma un arreglo como argumento y devuelve la suma de sus elementos. Esta es una operación simple, donde la función recorre cada elemento del arreglo y lo agrega a una variable suma. En C++, una función equivalente puede ser definida de manera similar.

»Ejemplo en Python:

```
»def suma_elementos(arreglo):
»    suma = 0
»    for elemento in arreglo:
»        suma += elemento
»    return suma
```

»Ejemplo en C++:

```
»int suma_elementos(std::vector<int>& arreglo) {
»    int suma = 0;
»    for (int elemento : arreglo) {
»        suma += elemento;
»    }
»    return suma;
»}
```

»VII-B. Entrada de Datos

»En Python, el tamaño del arreglo y sus elementos son ingresados por el usuario a través de la función `input()`. En este caso, el tamaño se convierte en un número entero y luego se solicita al usuario que ingrese cada elemento del arreglo, los cuales se agregan a la lista utilizando el método `append()`.

»Ejemplo en Python:

```
»tamaño = int(input("Ingresa el tamaño del arreglo: "))
```

```

»arreglo = []
»for i in range(tamaño):
»    elemento = int(input())
»    arreglo.append(elemento)

```

»En C++, se puede hacer de la siguiente manera, utilizando cin para obtener la entrada del usuario. En lugar de una lista, se usaría un vector, que es dinámico en tamaño, similar a las listas en Python.

»Ejemplo en C++:

```

»int tamaño;
»std::cout << "Ingresa el tamaño del arreglo: ";
»std::cin >> tamaño;
»std::vector<int> arreglo;
»for (int i = 0; i < tamaño; i++) {
»    int elemento;
»    std::cin >> elemento;
»    arreglo.push_back(elemento);
»}

```

»VII-C. *Cálculo y Salida de la Suma*

»En Python, después de ingresar los elementos, se llama a la función suma_elementos y se imprime el resultado utilizando print().

»Ejemplo en Python:

```

»suma = suma_elementos(arreglo)
»print("La suma de los elementos es:", suma)

```

»En C++, se usaría std::cout para mostrar la salida.

»Ejemplo en C++:

```

»int suma = suma_elementos(arreglo);
»std::cout << "La suma de los elementos es: " << suma << std::endl;

```

»VII-D. *Estructura del Código y Modularidad*

»Al igual que en el código de manejo de archivos, Python utiliza la verificación if __name__ == "__main__": para ejecutar la función principal solo cuando el archivo es ejecutado directamente. Este concepto no tiene un equivalente exacto en C++, ya que la ejecución siempre comienza desde la función main().

»Ejemplo en Python:

```

»if __name__ == "__main__":
»    main()

```

»En C++, la función main() siempre es el punto de entrada:

»Ejemplo en C++:

```

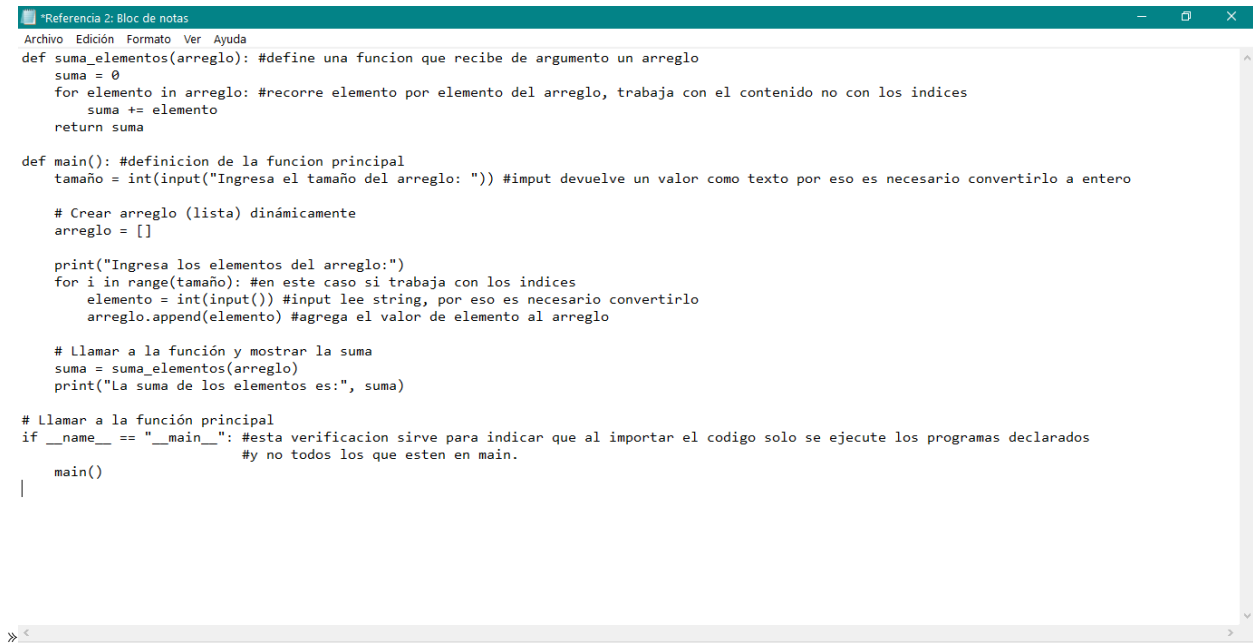
»int main() {
»    // Código principal aquí
»    return 0;
»}

```

»VII-E. Resumen de Comparación

»Este ejemplo muestra cómo tanto Python como C++ permiten definir funciones para sumar elementos de un arreglo de manera eficiente. Aunque las sintaxis varían entre ambos lenguajes, el concepto fundamental es el mismo. Python ofrece una manera más concisa de manejar listas y la entrada del usuario, mientras que C++ requiere más manejo explícito de estructuras como `std::vector` y un control más detallado de tipos de datos.

»El uso de la verificación `__name__ == "__main__"`: en Python proporciona una forma conveniente de modularizar el código para evitar que se ejecute al importar el archivo en otros scripts, mientras que en C++ esto no es necesario debido a la estructura predeterminada de la función `main()`.



```
def suma_elementos(arreglo): #define una funcion que recibe de argumento un arreglo
    suma = 0
    for elemento in arreglo: #recorre elemento por elemento del arreglo, trabaja con el contenido no con los indices
        suma += elemento
    return suma

def main(): #definicion de la funcion principal
    tamaño = int(input("Ingresa el tamaño del arreglo: ")) #input devuelve un valor como texto por eso es necesario convertirlo a entero

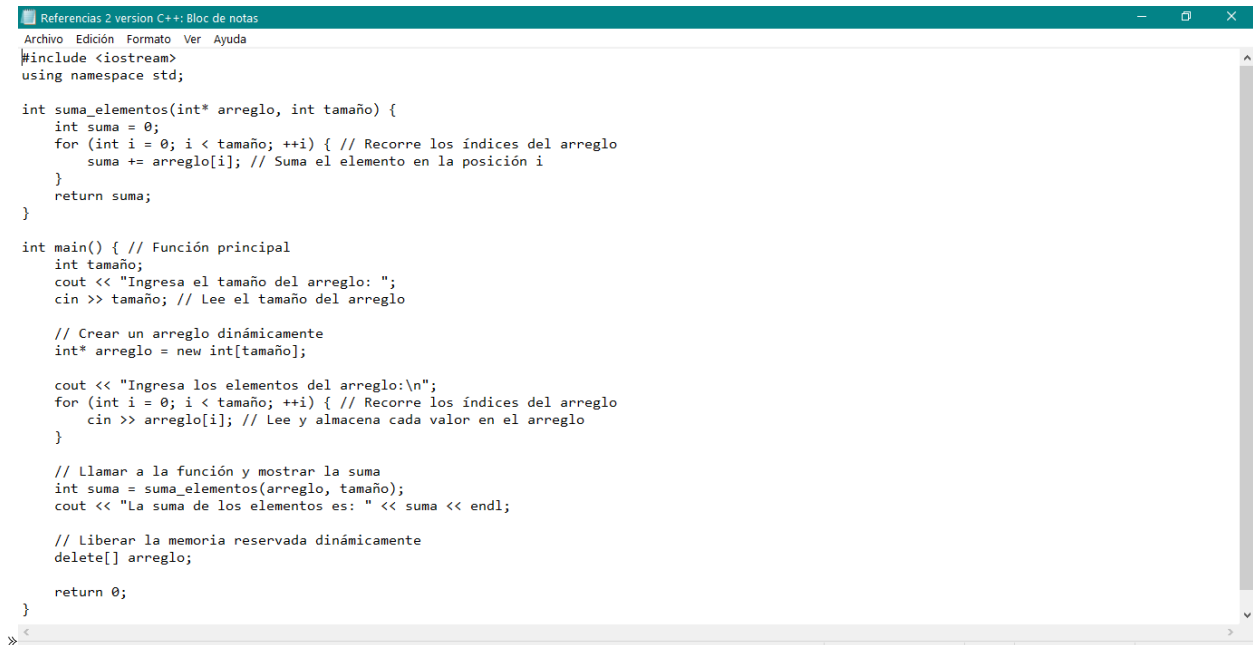
    # Crear arreglo (lista) dinámicamente
    arreglo = []

    print("Ingresa los elementos del arreglo:")
    for i in range(tamaño): #en este caso si trabaja con los indices
        elemento = int(input()) #input lee string, por eso es necesario convertirlo
        arreglo.append(elemento) #agrega el valor de elemento al arreglo

    # llamar a la función y mostrar la suma
    suma = suma_elementos(arreglo)
    print("La suma de los elementos es:", suma)

# llamar a la función principal
if __name__ == "__main__": #esta verificacion sirve para indicar que al importar el codigo solo se ejecute los programas declarados
    #y no todos los que esten en main.
    main()
```

Figura 4. Código en lenguaje Python



```
Referencias 2 version C++: Bloc de notas
Archivo Edición Formato Ver Ayuda
#include <iostream>
using namespace std;

int suma_elementos(int* arreglo, int tamaño) {
    int suma = 0;
    for (int i = 0; i < tamaño; ++i) { // Recorre los índices del arreglo
        suma += arreglo[i]; // Suma el elemento en la posición i
    }
    return suma;
}

int main() { // Función principal
    int tamaño;
    cout << "Ingresa el tamaño del arreglo: ";
    cin >> tamaño; // Lee el tamaño del arreglo

    // Crear un arreglo dinámicamente
    int* arreglo = new int[tamaño];

    cout << "Ingresa los elementos del arreglo:\n";
    for (int i = 0; i < tamaño; ++i) { // Recorre los índices del arreglo
        cin >> arreglo[i]; // Lee y almacena cada valor en el arreglo
    }

    // Llamar a la función y mostrar la suma
    int suma = suma_elementos(arreglo, tamaño);
    cout << "La suma de los elementos es: " << suma << endl;

    // Liberar la memoria reservada dinámicamente
    delete[] arreglo;

    return 0;
}
```

Figura 5. Código en lenguaje C++

»VIII. CADENAS

»VIII-A. Definición de Funciones

»En Python, se define una función palindromo que toma una cadena como argumento, limpia los caracteres no alfabéticos y verifica manualmente si es un palíndromo.

»Ejemplo en Python:

```
»def palindromo(cadena):
»    array = []
»    for caracter in cadena:
»        if 'a' <= caracter.lower() <= 'z':
»            array.append(caracter.lower())
»    inicio = 0
»    fin = len(array) - 1
»    while inicio < fin:
»        if array[inicio] != array[fin]:
»            return False
»        inicio += 1
»        fin -= 1
»    return True
```

»En C++, la lógica es similar. Se define una función es_palindromo que utiliza un arreglo para almacenar los caracteres válidos y comprueba si forman un palíndromo.

»Ejemplo en C++:

```
»#include <iostream>
»#include <cctype>
»#include <cstring>
»
»bool es_palindromo(const char* cadena) {
»    char array[100];
»    int longitud = 0;
»    for (int i = 0; cadena[i] != '\0'; ++i) {
»        if (isalpha(cadena[i])) {
»            array[longitud++] = tolower(cadena[i]);
»        }
»    }
```



```

»         }
»     }
»     int inicio = 0, fin = longitud - 1;
»     while (inicio < fin) {
»         if (array[inicio] != array[fin]) {
»             return false;
»         }
»         ++inicio;
»         --fin;
»     }
»     return true;
» }

```

»VIII-B. Entrada de Datos

»En Python, la entrada del usuario se obtiene con la función `input()` y se almacena en una cadena.

»Ejemplo en Python:

```

»palabra = input("Ingresa una cadena: ")
»if palindromo(palabra):
»    print("La cadena es un palíndromo.")
»else:
»    print("La cadena NO es un palíndromo.")

```

»En C++, se utiliza `std::cin` para leer la entrada del usuario y almacenarla en un arreglo de caracteres.

»Ejemplo en C++:

```

»int main() {
»    char palabra[100];
»    std::cout << "Ingresa una cadena: ";
»    std::cin.getline(palabra, 100);
»    if (es_palindromo(palabra)) {
»        std::cout << "La cadena es un palíndromo.\n";
»    } else {
»        std::cout << "La cadena NO es un palíndromo.\n";
»    }
»    return 0;
»}

```

»VIII-C. Resumen de Comparación

»Ambos lenguajes permiten verificar si una cadena es un palíndromo utilizando los mismos pasos conceptuales:

- »■ Limpiar la cadena de caracteres no alfabéticos.
- »■ Convertir los caracteres a minúsculas.
- »■ Comparar los extremos de la cadena procesada.

»En Python, el manejo de cadenas es más conciso, mientras que en C++ es necesario un control explícito del almacenamiento y los tipos de datos.

```
Cadenas: Bloc de notas
Archivo Edición Formato Ver Ayuda
def palindromo(cadena):
    # Creamos un arreglo vacío para almacenar los caracteres de la cadena
    array = []

    # Paso 1: Convertir la cadena a minúsculas y eliminar caracteres no alfabéticos
    for caracter in cadena: # Recorrer cada carácter en la cadena original
        if 'a' <= caracter.lower() <= 'z': # Verificar si es una letra alfabética
            array.append(caracter.lower()) # Convertir a minúscula y agregar a la lista

    # Paso 2: Verificar si es un palíndromo manualmente
    inicio = 0
    fin = len(array) - 1
    while inicio < fin: # Compara los caracteres desde los extremos hacia el centro
        if array[inicio] != array[fin]:
            return False # Si hay una diferencia, no es un palíndromo
        inicio += 1
        fin -= 1

    return True # Si se recorren todos los caracteres sin diferencias, es un palíndromo

palabra = input("Ingresa una cadena: ")

if palindromo(palabra):
    print("La cadena es un palíndromo.")
else:
    print("La cadena NO es un palíndromo.")
```

Figura 6. Código en lenguaje Python

```
main.cpp
5 // Prototipo de la función
6 bool Palindromo(const string& cadena);
7
8 int main() {
9     string palabra;
10
11     cout << "Ingresa una cadena: ";
12     getline(cin, palabra);
13
14     if (Palindromo(palabra)) {
15         cout << "La cadena es un palíndromo." << endl;
16     } else {
17         cout << "La cadena NO es un palíndromo." << endl;
18     }
19
20     return 0;
21 }
22
23 bool Palindromo(const string& cadena) {
24     // Creamos un arreglo vacío para almacenar los caracteres de la cadena
25     string array; // Para almacenar los caracteres alfabéticos en minúsculas
26
27     // En este bucle convertiremos la cadena a minúsculas y eliminaremos caracteres no alfabéticos
28     for (int i = 0; i < cadena.size(); i++) {
29         char c = cadena[i];
30         // Verificar manualmente si es una letra alfabética (mayúscula o minúscula)
31         if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) {
32             // Convertir mayúsculas a minúsculas manualmente
33             if (c >= 'A' && c <= 'Z') {
34                 c = c - 'A' + 'a'; // Convertir a minúscula
35             }
36             array += c;
37         }
38     }
39
40     // Verificar si es un palíndromo manualmente
41     int inicio = 0;
42     int fin = array.size() - 1;
43     while (inicio < fin) {
44         if (array[inicio] != array[fin])
45             return false;
46         inicio++;
47         fin--;
48     }
49     return true;
50 }
51
52 // Fin del programa
53
54 input
Ingresa una cadena: yeah
La cadena NO es un palíndromo.
...Program finished with exit code 0
Press ENTER to exit console.
```

Figura 7. Código en lenguaje C++

»IX. FUNCIONES: PROBLEMA DE LA TORRE DE HANÓI

»El problema de la Torre de Hanói es un ejercicio clásico de recursión que consiste en mover un número de discos (n) de una torre de origen a una torre de destino utilizando una torre auxiliar, siguiendo las reglas:

- »■ Solo se puede mover un disco a la vez.
- »■ Un disco más grande no puede estar sobre uno más pequeño.

»Ejemplo en Python:

```
»# Inicializamos el contador en cero
»cont = 0
»
»def torre_de_hanoi(n, origen, destino, auxiliar):
»    global cont # Variable global para contar movimientos
»    if n == 1:
»        print(f"Mover disco 1 de {origen} a {destino}")
»        cont += 1
»        return
»    torre_de_hanoi(n - 1, origen, auxiliar, destino)
»    print(f"Mover disco {n} de {origen} a {destino}")
»    cont += 1
»    torre_de_hanoi(n - 1, auxiliar, destino, origen)
»
»# Ejemplo
»n_discos = 4
»torre_de_hanoi(n_discos, 'O', 'D', 'A')
»print(f"Numero de movimientos: {cont}")
```

»**Ejemplo en C++:** En C++, se implementa de manera similar utilizando recursión, pero con manejo explícito de las variables globales.

```
»#include <iostream>
»using namespace std;
»
»int cont = 0; // Variable global para contar movimientos
»
»void torre_de_hanoi(int n, char origen, char destino, char auxiliar) {
»    if (n == 1) {
»        cout << "Mover disco 1 de " << origen << " a " << destino << endl;
»        cont++;
»        return;
»    }
»    torre_de_hanoi(n - 1, origen, auxiliar, destino);
»    cout << "Mover disco " << n << " de " << origen << " a " << destino << endl;
»    cont++;
»    torre_de_hanoi(n - 1, auxiliar, destino, origen);
»}
»
»int main() {
»    int n_discos = 4;
»    torre_de_hanoi(n_discos, 'O', 'D', 'A');
»    cout << "Numero de movimientos: " << cont << endl;
»    return 0;
»}
```

»IX-A. Explicación del Algoritmo

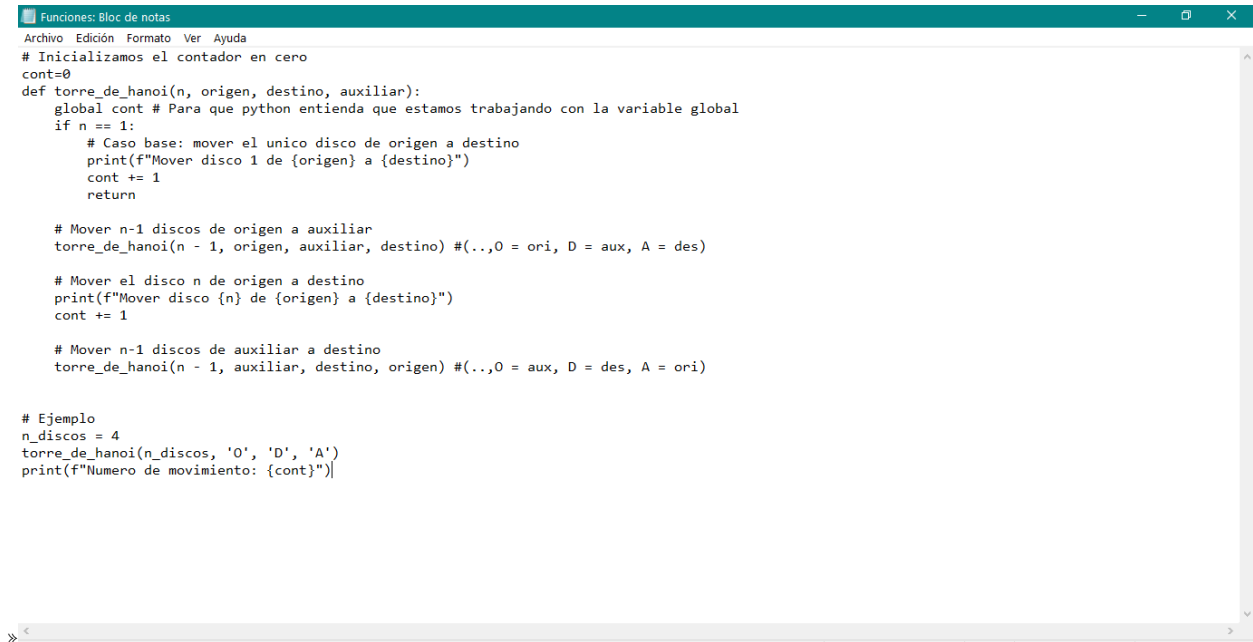
»El algoritmo de la Torre de Hanói sigue los siguientes pasos recursivos:

- »1. Mover los n-1 discos de la torre de origen a la torre auxiliar.

- »2. Mover el disco más grande (n) de la torre de origen a la torre de destino.
- »3. Mover los n-1 discos de la torre auxiliar a la torre de destino.

»IX-B. Resumen de Comparación

»En Python, la implementación utiliza variables globales y cadenas para nombrar las torres. En C++, el enfoque es similar, pero se utiliza una función main() como punto de entrada y se maneja explícitamente la salida con std::cout. Ambos lenguajes manejan la recursión de manera natural, permitiendo resolver el problema de forma directa.



```

Funciones: Bloc de notas
Archivo Edición Formato Ver Ayuda
# Inicializamos el contador en cero
cont=0
def torre_de_hanoi(n, origen, destino, auxiliar):
    global cont # Para que python entienda que estamos trabajando con la variable global
    if n == 1:
        # Caso base: mover el unico disco de origen a destino
        print(f"Mover disco 1 de {origen} a {destino}")
        cont += 1
        return

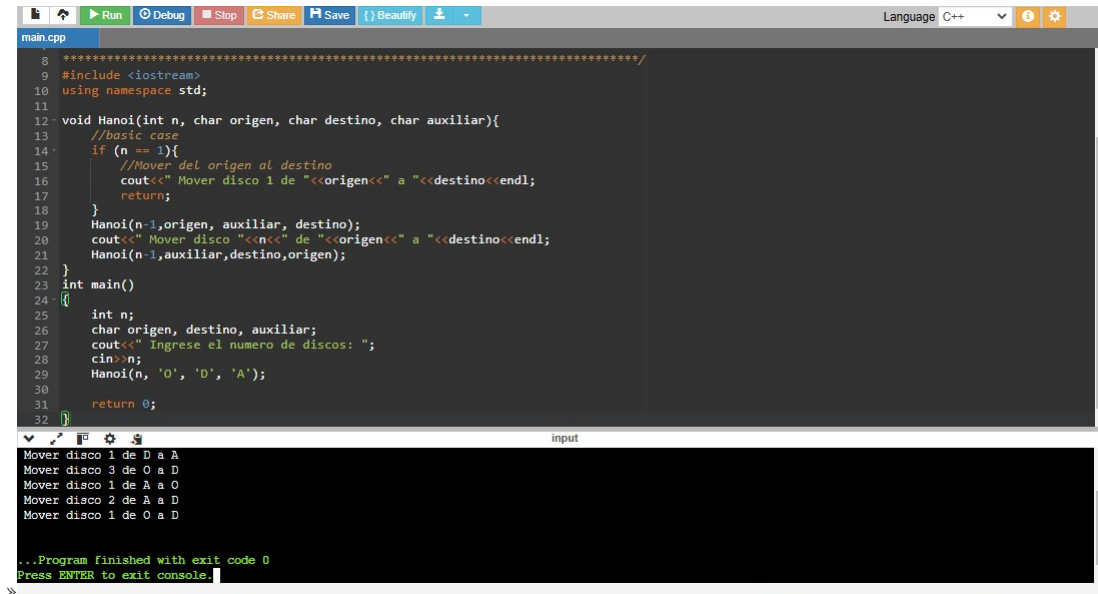
    # Mover n-1 discos de origen a auxiliar
    torre_de_hanoi(n - 1, origen, auxiliar, destino) #(..,0 = ori, D = aux, A = des)

    # Mover el disco n de origen a destino
    print(f"Mover disco {n} de {origen} a {destino}")
    cont += 1

    # Mover n-1 discos de auxiliar a destino
    torre_de_hanoi(n - 1, auxiliar, destino, origen) #(..,0 = aux, D = des, A = ori)

# Ejemplo
n_discos = 4
torre_de_hanoi(n_discos, 'O', 'D', 'A')
print(f"Numero de movimiento: {cont}")
  
```

Figura 8. Código en lenguaje Python



```

main.cpp
8 *****
9 #include <iostream>
10 using namespace std;
11
12 void Hanoi(int n, char origen, char destino, char auxiliar){
13     //basic case
14     if (n == 1){
15         //Mover del origen al destino
16         cout<<" Mover disco 1 de "<<origen<<" a "<<destino<<endl;
17         return;
18     }
19     Hanoi(n-1,origen, auxiliar, destino);
20     cout<<" Mover disco "<<n<<" de "<<origen<<" a "<<destino<<endl;
21     Hanoi(n-1,auxiliar,destino,origen);
22 }
23
24 int main()
25 {
26     int n;
27     char origen, destino, auxiliar;
28     cout<<" Ingrese el numero de discos: ";
29     cin>>n;
30     Hanoi(n, 'O', 'D', 'A');
31     return 0;
32 }
  
```

input

```

Mover disco 1 de D a A
Mover disco 3 de O a D
Mover disco 1 de A a O
Mover disco 2 de A a D
Mover disco 1 de O a D
...Program finished with exit code 0
Press ENTER to exit console.
  
```

Figura 9. Código en lenguaje C++

»X. CLASES: CÁLCULO DE DISTANCIA ENTRE PUNTOS

»El cálculo de la distancia entre dos puntos en un plano cartesiano es una operación fundamental en matemáticas y programación. Este ejemplo utiliza una clase Punto en Python para representar puntos y calcular la distancia entre ellos utilizando el método de Newton-Raphson para obtener raíces cuadradas.

»X-A. Definición del Problema

»El problema se resuelve utilizando dos métodos principales dentro de la clase Punto:

- »■ `raiz_cuadrada`: Calcula la raíz cuadrada de un número no negativo utilizando el método de Newton-Raphson.
- »■ `calcular_distancia`: Calcula la distancia euclidiana entre el punto actual y otro punto dado.

»Ejemplo en Python:

```
»class Punto:
»    def __init__(self, x=0, y=0):
»        self.x = x
»        self.y = y
»
»    def raiz_cuadrada(self, numero):
»        if numero < 0: # Calcula la raíz cuadrada usando el método de Newton-Raphson.
»            raise ValueError("El número debe ser positivo o cero.")
»
»        tolerancia = 1e-10
»        aproximacion = numero / 2.0
»
»        while abs(aproximacion**2 - numero) > tolerancia:
»            aproximacion = (aproximacion + numero / aproximacion) / 2.0
»
»        return aproximacion
»
»    def calcular_distancia(self, otro_punto): # Calcula la distancia.
»        dx = otro_punto.x - self.x
»        dy = otro_punto.y - self.y
»        return self.raiz_cuadrada(dx**2 + dy**2)
»
»# Ejemplo
»p1 = Punto(3.0, 4.0)
»p2 = Punto(7.0, 1.0)
»distancia = p1.calcular_distancia(p2)
»print(f"La distancia entre los puntos ({p1.x}, {p1.y}) y ({p2.x}, {p2.y}) es: {distancia}")
```

»X-B. Explicación del Algoritmo

»El cálculo de la distancia entre dos puntos sigue estos pasos:

- »1. Se calculan las diferencias en las coordenadas x y y entre los dos puntos.
- »2. Se calcula la suma de los cuadrados de estas diferencias.
- »3. Se aplica el método de Newton-Raphson para obtener la raíz cuadrada de la suma, representando la distancia euclidiana.

»X-C. Resumen de Comparación

»El uso de clases permite encapsular el comportamiento de los puntos en un solo lugar, facilitando su reutilización y extensión. Este enfoque hace que el código sea modular, fácil de entender y adecuado para aplicaciones que requieran cálculos geométricos frecuentes. La implementación en Python utiliza un estilo de programación orientado a objetos para simplificar el manejo de datos y operaciones asociadas a los puntos.

```
Estructuras 1 Python: Bloc de notas
Archivo Edición Formato Ver Ayuda

class Punto:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def raiz_cuadrada(self, numero):

        if numero < 0: #Calcula la raíz cuadrada usando el método de Newton-Raphson.
            raise ValueError("El número debe ser positivo o cero.")

        tolerancia = 1e-10
        aproximacion = numero / 2.0

        while abs(aproximacion**2 - numero) > tolerancia:
            aproximacion = (aproximacion + numero / aproximacion) / 2.0

        return aproximacion

    def calcular_distancia(self, otro_punto): #Calcula la distancia.
        dx = otro_punto.x - self.x
        dy = otro_punto.y - self.y
        return self.raiz_cuadrada(dx**2 + dy**2)

#Ejemplo
p1 = Punto(3.0, 4.0)
p2 = Punto(7.0, 1.0)
distancia = p1.calcular_distancia(p2)
print(f"La distancia entre los puntos ({p1.x}, {p1.y}) y ({p2.x}, {p2.y}) es: {distancia}")
```

Figura 10. Código en lenguaje Python

```
Estructuras 1: Bloc de notas
Archivo Edición Formato Ver Ayuda

struct Punto { // Estructura para representar un punto en el plano
    float x;
    float y;
};

float valorAbsoluto(float num) { // Función para calcular el valor absoluto
    return (num < 0) ? -num : num;
}

float calcularDistancia(const Punto& p1, const Punto& p2) { // Función para calcular la distancia entre dos puntos

    float dx = p2.x - p1.x;
    float dy = p2.y - p1.y;

    float distanciaCuadrada = dx * dx + dy * dy;

    float raiz = distanciaCuadrada / 2.0f; // Aproximación inicial
    float tolerancia = 1e-10; // Precisión de la raíz cuadrada

    while (valorAbsoluto(raiz * raiz - distanciaCuadrada) > tolerancia) { // Método de Newton-Raphson para calcular la raíz cuadrada
        raiz = (raiz + distanciaCuadrada / raiz) / 2.0f;
    }

    return raiz;
}

int main() {
    Punto p1, p2;

    cout << "Ingrese las coordenadas del primer punto (x1, y1): ";
    cin >> p1.x >> p1.y;

    cout << "Ingrese las coordenadas del segundo punto (x2, y2): ";
    cin >> p2.x >> p2.y;

    float distancia = calcularDistancia(p1, p2);
    cout << "La distancia entre los puntos es: " << distancia << endl;

    return 0;
}
```

Figura 11. Código en lenguaje C++

»XI. CLASES: SIMULACIÓN DE UNA CUENTA BANCARIA

»El manejo de una cuenta bancaria es una tarea común que puede modelarse de manera efectiva utilizando clases en programación orientada a objetos. Este ejemplo muestra cómo implementar una clase CuentaBancaria en Python para simular las operaciones básicas de una cuenta bancaria.

»XI-A. Definición del Problema

»El problema se resuelve utilizando los siguientes métodos dentro de la clase CuentaBancaria:

- »■ depositar: Permite agregar dinero al saldo de la cuenta si la cantidad es válida.
- »■ retirar: Resta dinero del saldo de la cuenta si la cantidad es válida y hay suficiente saldo disponible.
- »■ consultar_saldo: Muestra el saldo actual de la cuenta.

»Adicionalmente, el programa interactivo permite al usuario realizar estas operaciones repetidamente hasta que decida salir.

»Ejemplo en Python:

```
»class CuentaBancaria:
»    def __init__(self, saldo_inicial=0.0):
»        self.saldo = saldo_inicial # Inicializa el saldo actual
»
»    def depositar(self, cantidad):
»        if cantidad > 0: # Agrega dinero a la cuenta.
»            self.saldo += cantidad
»            print(f"Has depositado ${cantidad:.2f}. Saldo actual: ${self.saldo:.2f}")
»        else:
»            print("Cantidad inválida para depositar.")
»
»    def retirar(self, cantidad): # Resta el dinero si hay saldo suficiente.
»        if 0 < cantidad <= self.saldo:
»            self.saldo -= cantidad
»            print(f"Has retirado ${cantidad:.2f}. Saldo actual: ${self.saldo:.2f}")
»        else:
»            print("Cantidad inválida o saldo insuficiente.")
»
»    def consultar_saldo(self):
»        print(f"Saldo actual: ${self.saldo:.2f}") # Muestra el saldo actual.
»
»# Función interactiva
»print("Cuenta Bancaria")
»saldo_inicial = float(input("Ingrese su saldo inicial: "))
»cuenta = CuentaBancaria(saldo_inicial)
»
»while True:
»    print("\nSeleccione una opción:")
»    print("1. Depositar dinero")
»    print("2. Retirar dinero")
»    print("3. Consultar saldo")
»    print("4. Salir")
»
»    opcion = input("Opción: ")
»
»    if opcion == "1":
»        cantidad = float(input("Ingrese la cantidad a depositar: "))
»        cuenta.depositar(cantidad)
»    elif opcion == "2":
»        cantidad = float(input("Ingrese la cantidad a retirar: "))
»        cuenta.retirar(cantidad)
```

```

» elif opcion == "3":
»     cuenta.consultar_saldo()
» elif opcion == "4":
»     print("¡Gracias por usar nuestra aplicación bancaria!")
»     break
» else:
»     print("Opción inválida. Intente nuevamente.")

```

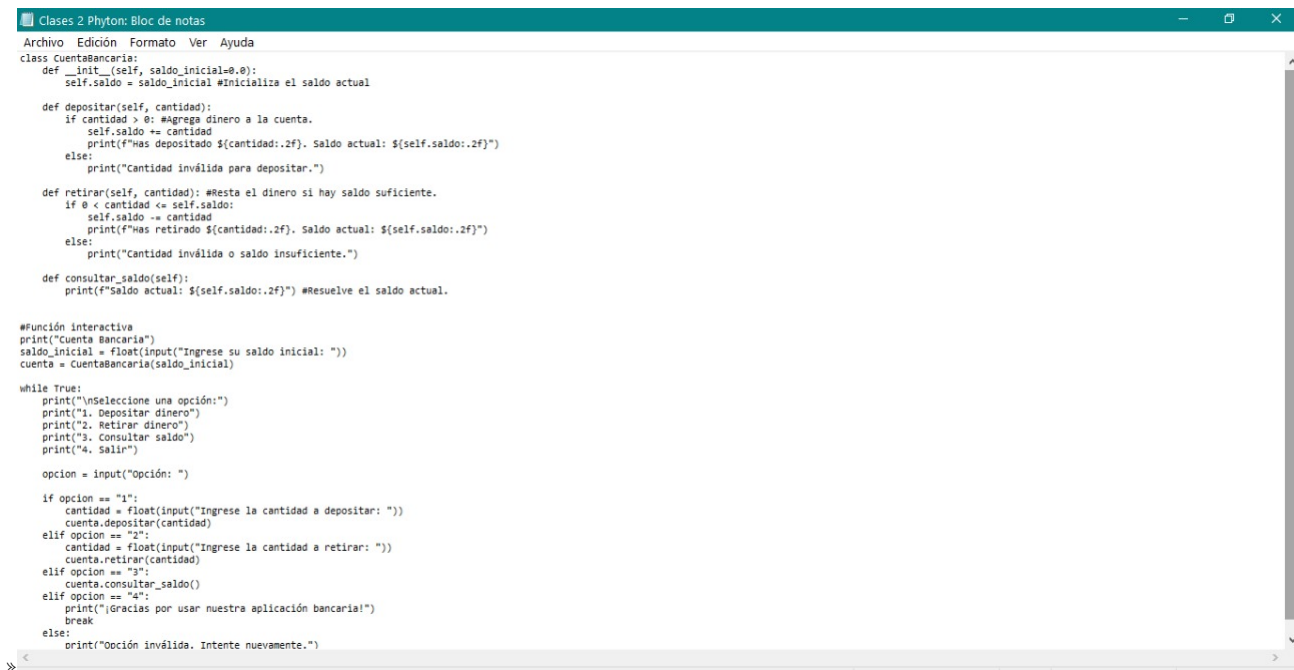
»XI-B. Explicación del Algoritmo

»El algoritmo para simular una cuenta bancaria incluye los siguientes pasos:

- »1. Inicializar el saldo con un valor proporcionado por el usuario.
- »2. Presentar un menú interactivo que permita elegir entre las opciones de depósito, retiro, consulta de saldo o salir.
- »3. Validar las cantidades ingresadas para depósito y retiro, asegurándose de que sean positivas y que no excedan el saldo disponible.
- »4. Actualizar el saldo después de cada operación válida.
- »5. Repetir el proceso hasta que el usuario elija salir.

»XI-C. Resumen de Comparación

»Este programa demuestra cómo encapsular la lógica y los datos relacionados con una cuenta bancaria dentro de una clase. El uso de métodos específicos para cada operación asegura que el código sea modular y fácil de mantener. La interacción con el usuario hace que el programa sea intuitivo y práctico para simular operaciones bancarias en un entorno controlado.



```

Clases 2 Phytton: Bloc de notas
Archivo Edición Formato Ver Ayuda
class CuentaBancaria:
    def __init__(self, saldo_inicial=0.0):
        self.saldo = saldo_inicial #inicializa el saldo actual

    def depositar(self, cantidad):
        if cantidad > 0: #Agrega dinero a la cuenta.
            self.saldo += cantidad
            print(f"Mas depositado ${(cantidad:.2f)}. Saldo actual: ${self.saldo:.2f}")
        else:
            print("Cantidad inválida para depositar.")

    def retirar(self, cantidad): #Resta el dinero si hay saldo suficiente.
        if 0 < cantidad <= self.saldo:
            self.saldo -= cantidad
            print(f"Mas retirado ${(cantidad:.2f)}. Saldo actual: ${self.saldo:.2f}")
        else:
            print("Cantidad inválida o saldo insuficiente.")

    def consultar_saldo(self):
        print(f"Saldo actual: ${self.saldo:.2f}") #Resuelve el saldo actual.

#Función interactiva
print("Cuenta Bancaria")
saldo_inicial = float(input("Ingrese su saldo inicial: "))
cuenta = CuentaBancaria(saldo_inicial)

while True:
    print("\nSeleccione una opción:")
    print("1. Depositar dinero")
    print("2. Retirar dinero")
    print("3. Consultar saldo")
    print("4. Salir")

    opcion = input("Opción: ")

    if opcion == "1":
        cantidad = float(input("Ingrese la cantidad a depositar: "))
        cuenta.depositar(cantidad)
    elif opcion == "2":
        cantidad = float(input("Ingrese la cantidad a retirar: "))
        cuenta.retirar(cantidad)
    elif opcion == "3":
        cuenta.consultar_saldo()
    elif opcion == "4":
        print("¡Gracias por usar nuestra aplicación bancaria!")
        break
    else:
        print("Opción inválida. Intente nuevamente.")

```

Figura 12. Código en lenguaje Phytton


```

1 #include <iostream>
2 using namespace std;
3
4 class CuentaBancaria { // Clase para representar una cuenta bancaria
5 private:
6     float saldo;
7
8 public:
9
10     CuentaBancaria() { //Constructor
11         saldo = 0.0f;
12     }
13
14     void depositar(float cantidad) { // Método para depositar dinero en la cuenta
15         if (cantidad > 0) {
16             saldo += cantidad;
17             cout << "Has depositado " << cantidad << ". Saldo actual: " << saldo << endl;
18         } else {
19             cout << "La cantidad a depositar debe ser mayor que 0." << endl;
20         }
21     }
22
23     void retirar(float cantidad) { // Método para retirar dinero de la cuenta
24         if (cantidad > 0) {
25             if (cantidad <= saldo) {

```

```

Cuenta Bancaria
. Depositar dinero
. Retirar dinero
. Ver saldo
. Salir
Elija una opción: 

```

Figura 13. Código en lenguaje C++

»XII. PROGRAMA PARA RESOLVER EL SUDOKU EN PHYTON

»La resolución de un Sudoku consiste en llenar un tablero 9×9 siguiendo las reglas:

- »■ Cada fila debe contener los números del 1 al 9 sin repetir.
- »■ Cada columna debe contener los números del 1 al 9 sin repetir.
- »■ Cada subcuadro 3×3 debe contener los números del 1 al 9 sin repetir.

»El siguiente código implementa una solución utilizando *backtracking*:

```

»# Función para buscar una celda libre
»def busca_celda_libre(sudoku):
»    for fila in range(9):
»        for columna in range(9):
»            if sudoku[fila][columna] == 0: # Compara cuál está libre
»                return (fila, columna) # Devuelve la posición de la celda libre
»    return None # Si no hay celdas vacías, retorna None
»
»# Función para verificar si un número es válido
»def numero_valido(sudoku, numero, fila, columna):
»    for columna_actual in range(9):
»        if sudoku[fila][columna_actual] == numero:
»            return False # Número ya está en la fila
»
»    for fila_actual in range(9):
»        if sudoku[fila_actual][columna] == numero:
»            return False # Número ya está en la columna
»
»    # Verificar si el número ya está en el subcuadro (3x3)
»    inicio_fila = (fila // 3) * 3
»    inicio_columna = (columna // 3) * 3
»    for fila_actual in range(inicio_fila, inicio_fila + 3):
»        for columna_actual in range(inicio_columna, inicio_columna + 3):
»            if sudoku[fila_actual][columna_actual] == numero:
»                return False # Número ya está en el subcuadro
»    return True # Si no hay conflictos, el número es válido
»

```

```

»# Función para resolver el Sudoku usando backtracking
»def resolver(sudoku):
»    celda = busca_celda_libre(sudoku)
»    if not celda: # Si no hay más celdas vacías, está resuelto
»        return True
»    fila, columna = celda
»
»    for numero in range(1, 10): # Probar números del 1 al 9
»        if numero_valido(sudoku, numero, fila, columna):
»            sudoku[fila][columna] = numero # Colocar el número provisionalmente
»            if resolver(sudoku):
»                return True # Si se resuelve, finalizamos
»            sudoku[fila][columna] = 0 # Deshacer si no funcionó
»
»    return False # Si ningún número es válido, no hay solución
»
»# Función para imprimir el tablero de Sudoku
»def imprimir_solucion(sudoku):
»    for fila in range(9):
»        if fila % 3 == 0 and fila != 0: # Línea divisoria entre subcuadros
»            print("-" * 21)
»
»        for columna in range(9):
»            if columna % 3 == 0 and columna != 0:
»                print("| ", end="")
»            print(sudoku[fila][columna] if sudoku[fila][columna] != 0 else ".", end=" ")
»        print()
»
»# Resuelve el siguiente Sudoku
»sudoku = [
»    [5, 3, 0, 0, 7, 0, 0, 0, 0],
»    [6, 0, 0, 1, 9, 5, 0, 0, 0],
»    [0, 9, 8, 0, 0, 0, 0, 6, 0],
»    [8, 0, 0, 0, 6, 0, 0, 0, 3],
»    [4, 0, 0, 8, 0, 3, 0, 0, 1],
»    [7, 0, 0, 0, 2, 0, 0, 0, 6],
»    [0, 6, 0, 0, 0, 0, 2, 8, 0],
»    [0, 0, 0, 4, 1, 9, 0, 0, 5],
»    [0, 0, 0, 0, 8, 0, 0, 7, 9]
»]
»
»if resolver(sudoku):
»    print("Solución encontrada:")
»    imprimir_solucion(sudoku)
»else:
»    print("No hay solución para este tablero.")

```

»XII-A. Explicación del Algoritmo

»El algoritmo utiliza **backtracking** para probar todas las combinaciones posibles hasta encontrar una solución válida:

- »1. La función `busca_celda_libre` localiza una celda vacía (0).
- »2. La función `numero_valido` verifica si un número puede colocarse en una celda sin violar las reglas del Sudoku.
- »3. La función `resolver` implementa un enfoque recursivo para intentar colocar números en las celdas libres y retrocede si encuentra un conflicto.

»**Ejemplo de salida:**

```

»5 3 4 | 6 7 8 | 9 1 2
»6 7 2 | 1 9 5 | 3 4 8
»1 9 8 | 3 4 2 | 5 6 7
»-----+-----+-----
»8 5 9 | 7 6 1 | 4 2 3
»4 2 6 | 8 5 3 | 7 9 1
»7 1 3 | 9 2 4 | 8 5 6
»-----+-----+-----
»9 6 1 | 5 3 7 | 2 8 4
»2 8 7 | 4 1 9 | 6 3 5
»3 4 5 | 2 8 6 | 1 7 9

```

The screenshot shows a Python IDE with a dark theme. The top toolbar includes buttons for Run, Debug, Stop, Share, Save, Beautify, and a dropdown menu. The language is set to Python 3. The editor window shows a file named 'main.py' with the following code:

```

31
32 # Función para resolver el Sudoku usando backtracking
33 def resolver(sudoku):
34     # Buscar una celda vacía
35     celda = busca_celda_libre(sudoku)
36     if not celda: # Si no hay más celdas vacías, está resuelto
37         return True
38     fila, columna = celda
39

```

The console window shows the output of the program:

```

input
Solución encontrada:
5 3 4 | 6 7 8 | 9 1 2
6 7 2 | 1 9 5 | 3 4 8
1 9 8 | 3 4 2 | 5 6 7
-----
8 5 9 | 7 6 1 | 4 2 3
4 2 6 | 8 5 3 | 7 9 1
7 1 3 | 9 2 4 | 8 5 6
-----
9 6 1 | 5 3 7 | 2 8 4
2 8 7 | 4 1 9 | 6 3 5
3 4 5 | 2 8 6 | 1 7 9

...Program finished with exit code 0
Press ENTER to exit console.
»

```

Figura 14. Código en lenguaje Python

»XIII. CONCLUSIONES

»Python y C++ son lenguajes complementarios más que competidores. Python es excelente para el desarrollo rápido y tareas de alto nivel, mientras que C++ es indispensable cuando se necesita eficiencia y control del hardware. La elección entre ambos depende en gran medida del contexto y los requisitos del proyecto.