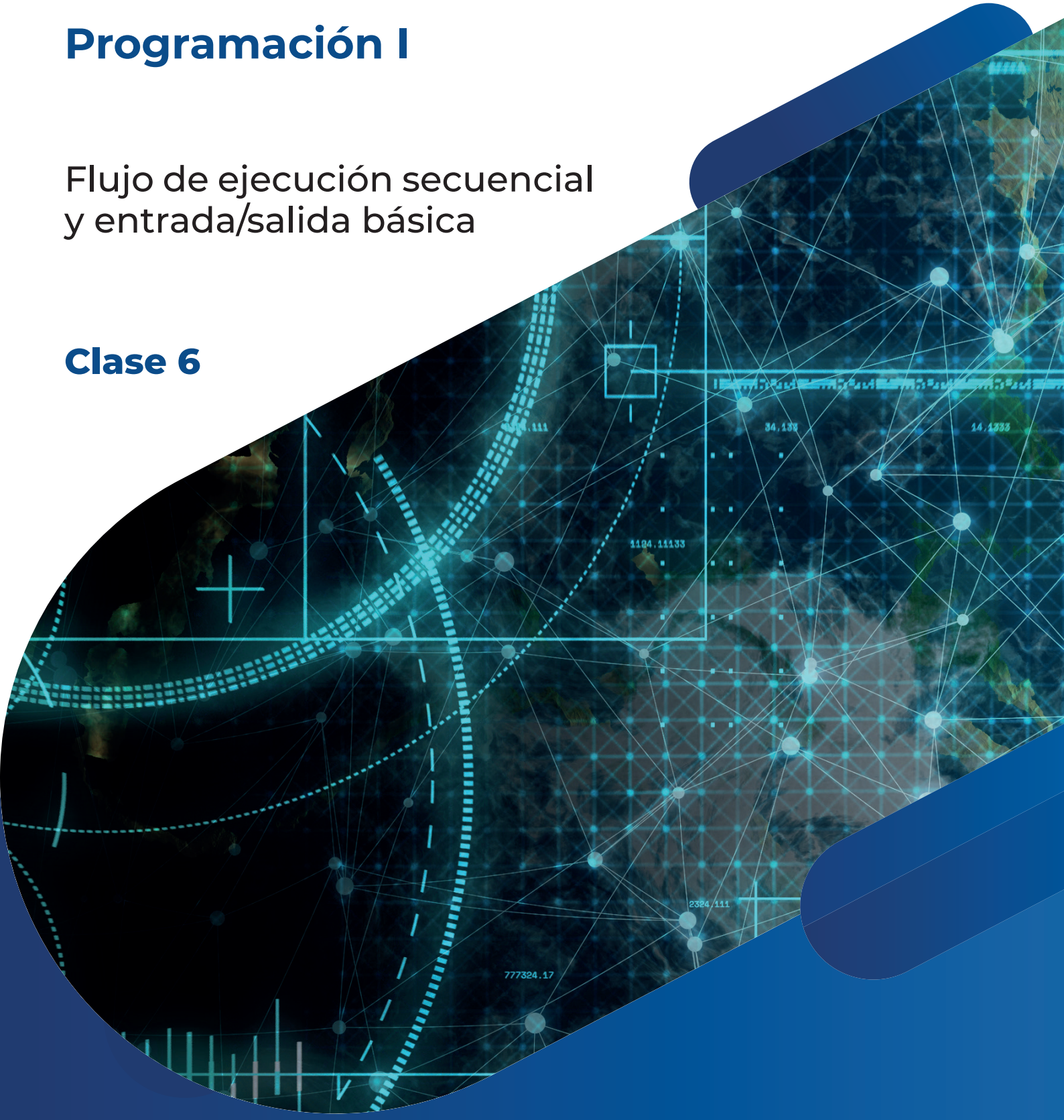
Programación I

Flujo de ejecución secuencial y entrada/salida básica

Clase 6



Ingeniería en ciberseguridad

La excelencia no se improvisa

1. INTRODUCCIÓN DE LA CLASE

¡Bienvenidos a la Clase 5! Hoy nos sumergiremos en los fundamentos del flujo de ejecución secuencial en Python y la forma básica de entrada y salida de datos. Imagina que, hasta este punto, hemos aprendido a definir variables, tipos de datos y operadores; ahora, veremos cómo estos elementos se ejecutan de manera lineal en un programa, sin saltos ni bifurcaciones. La secuencialidad es esencial para entender cómo el código avanza de la primera instrucción hasta la última, permitiéndonos crear algoritmos simples pero funcionales.

Además, conoceremos la función input(), que nos permitirá leer datos del usuario, haciendo nuestros programas más interactivos. Verás cómo se combina la estructura secuencial con la lectura de valores para procesarlos y presentar resultados de forma ordenada. Dominar la estructura secuencial y la función input() es el primer paso para desarrollar aplicaciones de mayor complejidad en el futuro.

Clase 5: Flujo de ejecución secuencial y entrada/salida básica (Parte 1)

Resultado o resultados de aprendizaje que será abordado con el contenido de la clase.

Reconocer las estructuras básicas de un lenguaje de programación estructurado, su sintaxis y su utilidad en la solución de problemas de programación.

Reto # 2

Contenido de la Clase:

5. **Flujo de ejecución secuencial y entrada/salida básica (Parte 1)**

En esta clase, profundizaremos en la forma más elemental de organizar las instrucciones en un programa: el flujo de ejecución secuencial. Bajo este modelo, cada línea de código se ejecuta de forma lineal, en el orden en que aparece, sin tomar decisiones lógicas ni saltos condicionales. Este tipo de flujo resulta ideal para tareas como la inicialización de variables, la realización de cálculos básicos o la presentación de resultados de manera progresiva. Al no haber bifurcaciones, es más sencillo comprender y depurar el comportamiento del programa, pues cada instrucción se procesa inmediatamente después de la anterior.

Para llevar la secuencialidad un paso más allá, aprenderemos cómo interactuar con el usuario a través de la función input(), que permite leer datos durante la ejecución. Esta capacidad otorga dinamismo al programa, al solicitar información que se procesará secuencialmente antes de mostrar una salida. Con ello, podemos desarrollar aplicaciones que realicen operaciones básicas (suma, resta, concatenaciones) basadas en los valores ingresados por el usuario, lo cual enriquece la experiencia y fomenta la interactividad.

Dividiremos el contenido en dos secciones principales:

1. Estructura de un programa secuencial: Explicaremos cómo se organizan las instrucciones de forma lineal y qué ventajas y limitaciones conlleva este modelo.

2. Función input() para lectura de datos en Python: Veremos cómo solicitar valores al usuario, convertirlos a distintos tipos y utilizarlos en cálculos o validaciones simples.

Esta aproximación te permitirá desarrollar programas que ejecuten pasos en un orden estricto, soliciten información al usuario y presenten resultados de manera ordenada. Asimismo, sentará las bases para la introducción de estructuras de control más complejas (condicionales y bucles) en clases posteriores, cuando sea necesario tomar decisiones o repetir instrucciones. De este modo, la secuencialidad servirá como cimiento

para la evolución hacia algoritmos más completos y potentes.

5.1. Estructura de un programa secuencial.

La estructura secuencial representa la forma más básica y directa para organizar y ejecutar instrucciones dentro de un programa. Se caracteriza principalmente por la ejecución lineal y continua de cada línea de código, donde cada instrucción se realiza justo después de haber finalizado la anterior, sin saltos ni interrupciones lógicas que modifiquen el orden establecido inicialmente. Esta manera de trabajar facilita notablemente el entendimiento y el seguimiento del flujo del programa, especialmente en etapas iniciales del aprendizaje de la programación.

Este modelo secuencial es ampliamente utilizado en situaciones donde las tareas son claras y no requieren condiciones particulares o repetición cíclica, resultando muy eficiente para realizar procesos simples o inicializaciones básicas. Por ejemplo, cuando se desea preparar un programa que calcule el salario de un trabajador a partir del número de horas laboradas y su tarifa por hora, simplemente se ingresan estos datos de forma consecutiva, se realiza la operación de multiplicación correspondiente, y finalmente se presenta el resultado al usuario. Este proceso no necesita evaluaciones condicionales ni estructuras repetitivas, por lo que una secuencia lineal es perfectamente adecuada.

Para comprender mejor cómo funciona, se puede dividir la estructura secuencial en cuatro etapas claramente diferenciadas:

1. Instrucciones en Orden: Esta primera etapa implica ejecutar las órdenes una por una, en el mismo orden en que aparecen escritas en el código. Cada instrucción finalizada da paso inmediato a la siguiente, lo que ofrece una predictibilidad que facilita su comprensión.
2. Declaraciones y Asignaciones: En esta fase inicial, el programador define las variables necesarias para almacenar información durante la ejecución del programa. Las variables se inicializan asignándoles valores iniciales, lo que garantiza que el programa tenga datos definidos claramente antes de proceder a realizar operaciones más avanzadas.
3. Operaciones y Cálculos: Posteriormente, se llevan a cabo las operaciones matemáticas o las manipulaciones básicas necesarias, tales como sumas, restas, multiplicaciones o divisiones, así como transformaciones sencillas de los datos. Es importante destacar que, en la estructura secuencial pura, estas operaciones no dependen de condiciones específicas o decisiones lógicas, sino que se realizan siempre en el orden establecido.
4. Salida de Resultados: Finalmente, los resultados obtenidos se presentan al usuario, ya sea mediante mensajes en pantalla, guardando la información en archivos o mediante otro medio de almacenamiento. Este paso concluye el proceso, entregando al usuario una respuesta clara y directa derivada de las operaciones realizadas previamente.

Entre las principales ventajas de utilizar una estructura secuencial destacan:

- Facilidad de Entendimiento: Gracias a su linealidad, cualquier persona puede seguir fácilmente el flujo del programa, lo cual simplifica notablemente tanto la enseñanza como el aprendizaje en etapas iniciales.
- Rapidez en la Depuración: Al no contar con bifurcaciones o decisiones complejas, identificar y corregir errores resulta más sencillo, permitiendo a los desarrolladores detectar rápidamente el punto exacto donde ocurre el problema.

Por otro lado, la estructura secuencial presenta limitaciones que es importante considerar:

- Limitada Capacidad de Adaptación: Debido a la ausencia de estructuras condicionales y repetitivas, no es posible adaptar el programa a situaciones cambiantes o inesperadas. Esta rigidez limita significativamente su aplicación en entornos dinámicos o complejos.
- Escalabilidad Limitada: Cuando los programas crecen en tamaño o complejidad, una estructura meramente secuencial se vuelve insuficiente, requiriendo inevitablemente complementarse con mecanis-

mos que permitan condiciones y repeticiones, como estructuras condicionales (if-else) o bucles (for, while).

Por lo tanto, aunque la estructura secuencial es el punto de partida ideal para aprender a programar y es eficiente en contextos específicos, es fundamental combinarla con otras estructuras de control a medida que la complejidad del proyecto aumenta. De esta forma, se asegura un desarrollo adecuado, eficiente y adaptativo que responda satisfactoriamente a las diversas necesidades del software actual.

La secuencialidad permite comprender cómo se ejecuta un programa paso a paso, siendo la base de la lógica computacional.

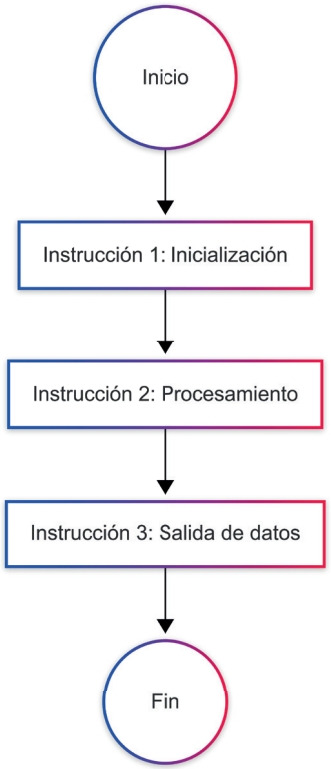


Imagen 1: Diagrama Esquemático de la Estructura Secuencial

Damián Nicolalde Rodríguez. (2024). Diagrama de la Estructura Secuencial.

La tabla 1 presenta un ejemplo sencillo de cómo se organiza un programa siguiendo la estructura secuencial en Python. Se muestran tres pasos fundamentales: en primer lugar, se declaran las variables y se les asignan valores. A continuación, se realizan operaciones simples, como un cálculo aritmético. Finalmente, se imprime el resultado o se guarda en un archivo, completando así el ciclo secuencial.

Tabla 1: Ejemplo de Estructura Secuencial en un Programa

Paso	Descripción	Ejemplo de Código
Declaración de Variables	Se definen variables y se les asignan valores iniciales.	nombre = “Carlos” edad = 25
Operaciones Simples	Se realizan cálculos aritméticos o lógicos básicos.	resultado = edad + 5

Presentación de Resultados	Se imprime el resultado o se guarda en un archivo, completando el ciclo secuencial.	print(“Nombre:”, nombre) print(“Nueva edad:”, resultado)

Damián Nicolalde Rodríguez. (2024).

```python
# Declaración de variable
x = 10

# Realización de un cálculo aritmético (se suma 5 a x)
resultado = x + 5

# Impresión del resultado
print("El resultado es:", resultado)
```

Imagen 2: Ejemplo de Programa Secuencial

Damián Nicolalde Rodríguez. (2024). Ejemplo de Programa Secuencial.

La tabla 2 expone diversos casos de uso y escenarios en los que la estructura secuencial resulta particularmente útil en Python. Cada fila describe una situación típica donde las instrucciones se ejecutan de forma lineal y ordenada, sin necesidad de condicionales ni bucles. Por ejemplo, en el escenario de Inicialización de Variables, el programa simplemente define y asigna valores base (contador, nombre, mensaje) al comienzo, mientras que, en el caso de Cálculos Matemáticos Simples, se efectúan operaciones lineales como sumar o restar sin ramificaciones lógicas. Del mismo modo, se ilustran situaciones donde se imprime información secuencialmente o se llevan a cabo rutinas de limpieza o configuración previas a la ejecución principal de un sistema. En conjunto, estos ejemplos refuerzan la idea de que la estructura secuencial es la base para organizar instrucciones que se ejecuten paso a paso, de manera clara y progresiva.

Tabla 2: Casos de Uso y Escenarios para Estructura Secuencial

Escenario	Descripción	Ejemplo Práctico
Inicialización de Variables	Se definen y asignan valores base antes de que el programa avance a pasos más complejos.	Declarar contador = 0, nombre = “Usuario” y mensaje = “Bienvenido” en las primeras líneas.
Cálculos Matemáticos Simples	Operaciones que no requieren bifurcación, como sumar, restar o multiplicar.	Un script que calcula el precio final de un producto sumando impuestos de forma lineal.
Presentación Secuencial de Datos	Mostrar mensajes o reportes en orden estricto, sin condicionales.	Imprimir un resumen de ventas diarias, línea tras línea, sin verificación de condiciones especiales.
Rutinas de Limpieza o Setup	Tareas que se realizan siempre en el mismo orden antes de iniciar la lógica principal de un sistema.	Cargar configuraciones básicas, abrir un archivo y leerlo, luego mostrar un menú inicial, todo sin bifurcaciones.

Damián Nicolalde Rodríguez. (2024).

Para complementar y profundizar, se sugieren acceder a este recurso:

- Título del enlace relacionado: Estructura secuencial en Python

- **Descripción del enlace relacionado:** El video “Estructura secuencial en Python” muestra cómo las instrucciones de un programa se ejecutan de manera lineal y ordenada, sin bifurcaciones. A través de ejemplos sencillos, se ilustran pasos como la declaración de variables, la realización de operaciones aritméticas básicas y la impresión de resultados, destacando la importancia de la secuencialidad como base para comprender y organizar el código antes de introducir estructuras de control más complejas.

- **Enlace:** Estructura Secuencial en Python (1/3)

5.2. **Función input() para lectura de datos (en Python).**

La función input() es la forma más sencilla de leer datos proporcionados por el usuario durante la ejecución de un programa.

La función input() es una de las herramientas más importantes en Python para la interacción directa con el usuario, permitiendo captar la información que este proporcione durante la ejecución del programa. Su sencillez radica en la facilidad con la que puede solicitar y recibir datos. Cuando un programa ejecuta esta función, aparece en pantalla un mensaje opcional dirigido al usuario, indicándole qué tipo de información se espera. A continuación, el programa detiene temporalmente su ejecución hasta que el usuario ingrese los datos solicitados y confirme su respuesta pulsando la tecla Enter. En ese momento, Python toma lo ingresado y lo devuelve al programa en forma de cadena de texto (tipo str), que puede ser posteriormente procesada según la necesidad específica del contexto.

El uso de la función input() constituye la base fundamental para programas interactivos, pues gracias a esta característica es posible que la ejecución de un software se ajuste a las diferentes necesidades o preferencias de quien lo utiliza en cada ocasión. Por ejemplo, un sistema de reservaciones puede solicitar información personalizada, como nombre, edad, fecha o preferencias específicas de cada usuario, adaptando sus respuestas o comportamiento dependiendo directamente de esos datos ingresados. Esto mejora enormemente la flexibilidad y utilidad práctica de cualquier aplicación.

Aunque input() ofrece una gran simplicidad en su uso, también requiere atención especial debido a ciertas particularidades relacionadas con el tipo de datos retornado. Es importante considerar que todo lo ingresado mediante esta función se obtiene inicialmente como texto, incluso si lo que se ingresa parece ser numérico. Esto implica que, para realizar cálculos matemáticos, comparaciones numéricas o cualquier otro procesamiento específico con valores numéricos, será indispensable convertir explícitamente estos datos a los tipos adecuados como int (entero) o float (decimal). De lo contrario, podrían generarse errores o resultados inesperados en la ejecución del programa.

Por ejemplo, si un programa necesita saber la edad del usuario y posteriormente compararla con un valor numérico específico, será necesario utilizar una conversión de datos explícita para manejar correctamente esta información. La forma más habitual es la siguiente:

```
edad = int(input(“Ingrese su edad: “))
```

En este ejemplo, la función input() primero pide al usuario que ingrese su edad mediante un mensaje claro. A continuación, convierte inmediatamente ese valor en un número entero mediante la función int(), asegurando que el programa maneje el dato como un número en lugar de texto. De esta forma, el valor ingresado puede ser usado de manera eficiente en operaciones numéricas posteriores, tales como validar la mayoría de edad o verificar que la edad ingresada esté dentro de un rango lógico.

La claridad en los mensajes mostrados al usuario también es fundamental cuando se utiliza input(). Un mensaje preciso y bien formulado ayuda significativamente a reducir posibles errores humanos, asegurando que los datos introducidos sean pertinentes y acordes al contexto esperado por el programa. Por ejemplo, si un programa requiere el ingreso de la temperatura en grados Celsius, un mensaje explícito como “Ingrese la temperatura actual en grados Celsius:” es más efectivo que una solicitud generalizada como “Ingrese un valor:”.

Aunque la función input() es sencilla y muy eficaz para conseguir interacción directa con los usuarios, exige del programador atención a la correcta conversión de datos y claridad en las instrucciones proporcionadas. Con ello, se mejora notablemente la experiencia de usuario y se evitan errores innecesarios durante la ejecución del programa, haciendo que las aplicaciones desarrolladas sean más robustas y amigables.

1. Uso Básico de input():

```
nombre = input(“Ingrese su nombre: “)

print(“Hola,”, nombre)
```

   Aquí, el programa solicita el nombre y lo almacena en la variable nombre. Luego, se imprime un saludo personalizado.

2. Conversión de Tipos: Como input() retorna siempre una cadena, para realizar operaciones numéricas debemos convertir el valor a int o float:

```
edad = int(input(“Ingrese su edad: “))

altura = float(input(“Ingrese su altura en metros: “))
```

   Esto posibilita el uso de operadores aritméticos en los valores ingresados.

3. Mensajes Claros: Es recomendable mostrar mensajes claros que indiquen al usuario qué tipo de dato debe ingresar (por ejemplo, “Ingrese un número entero”). Esto reduce errores y confusiones durante la ejecución.

Ventajas de input():

- Interactividad: Permite que el usuario ingrese valores en tiempo de ejecución, haciendo el programa dinámico y adaptable.
- Simplicidad: Es fácil de usar y suficiente para muchos programas introductorios que requieran entrada de datos.

Limitaciones de input():

- Solo Retorna Cadenas: Para trabajar con números o valores booleanos, se requiere conversión manual.
- Sin Validación Integrada: No evita automáticamente que el usuario ingrese texto en lugar de números; es necesario implementar verificación o manejo de excepciones para evitar errores.

La función input() brinda interactividad al programa, permitiendo al usuario ingresar datos que se procesan secuencialmente.

```
# Solicitar la edad del usuario y convertirla a entero
edad = int(input("Ingrese su edad: "))

# Imprimir mensaje de confirmación mostrando la edad ingresada
print("Tu edad es:", edad)
```

Imagen 3: Ejemplo de Input y Output en Python

Damián Nicolalde Rodríguez. (2024). Ejemplo de Input y Output.

La Tabla 3 ilustra diferentes casos de uso para la función input() en Python, enfatizando la necesidad de convertir los valores retornados (siempre cadenas) a tipos numéricos o lógicos según la finalidad del programa. En el primer caso, se muestra cómo solicitar datos numéricos que exigen la conversión a int o float para realizar operaciones matemáticas. El segundo caso aborda la lectura de texto con fines de validación (por ejemplo, contraseñas), mientras que el tercero combina la entrada de datos de distintos tipos (nombre, edad, salario) en un solo flujo secuencial. En conjunto, estos ejemplos subrayan la importancia de mensajes claros para el usuario y la conversión de tipo apropiada, con el fin de manipular correctamente la información que se ingresa.

Tabla 3: Casos de Uso de input() y su Conversión

Caso de Uso	Descripción	Ejemplo de Código
Ingreso de Datos Numéricos	Solicita valores para realizar operaciones matemáticas, exigiendo conversión a int o float.	num1 = int(input(“Ingrese un número entero: “)) num2 = float(input(“Ingrese un número decimal: “)
Lectura de Texto para Validaciones	Permite procesar cadenas para comparaciones lógicas (por ejemplo, contraseñas).	contraseña = input(“Ingrese su contraseña: “) if contraseña == “segura”: print(“Acceso permitido”)
Combinación de Entradas	Solicita múltiples datos de distintos tipos en un flujo secuencial.	nombre = input(“Nombre: “) edad = int(input(“Edad: “)) salario = float(input(“Salario: “))

Damián Nicolalde Rodríguez. (2024).

5.2.1. **try-except**

En Python, manejar errores durante la lectura de datos es fundamental para evitar que el programa se bloquee o arroje excepciones no controladas cuando el usuario ingresa información inesperada. Para ello, se utiliza el bloque try-except, que permite intentar ejecutar un código y, en caso de que se produzca un error, atraparlo y manejarlo de manera amigable. A continuación, se presenta un ejemplo que ilustra el manejo de un ValueError al convertir una cadena a un entero:

```
try:
    numero = int(input("Ingrese un número entero: "))
    print(f"El número ingresado es: {numero}")
except ValueError:
    print("Error: Debe ingresar un valor numérico válido.")
```

1. Bloque try:
    - Dentro del bloque try, se realiza la conversión de la cadena retornada por input() a un entero mediante int().
    - Si el usuario ingresa algo que no puede convertirse a número (por ejemplo, "hola" en lugar de "42"), Python genera un ValueError.
2. Bloque except ValueError:
    - Este bloque captura específicamente la excepción ValueError que ocurre cuando se intenta convertir a entero (o flotante) una cadena no numérica.
    - En lugar de que el programa se bloquee y muestre un mensaje de error técnico, se imprime un mensaje claro al usuario, indicando que debe ingresar un valor numérico válido.
3. Ventajas del try-except:
    - Mejora la Experiencia de Usuario: Evita que el programa se detenga abruptamente, ofreciendo un mensaje descriptivo sobre el error.
    - Control Fino de Excepciones: Puedes capturar distintos tipos de errores (por ejemplo, ValueError, ZeroDivisionError, etc.) y responder de manera diferente en cada caso.
    - Código Más Robusto: Permite que el programa continúe ejecutándose o solicite nuevamente la información, en lugar de cerrarse inesperadamente.
4. Posible Extensión – Reintento o Mensajes Más Detallados:
    - Podrías envolver este bloque en un bucle para solicitar nuevamente el dato hasta que el usuario ingrese un valor válido.
    - Podrías mostrar mensajes más específicos, como "El valor ingresado no es un número entero. Por favor, inténtelo de nuevo.", mejorando aún más la interacción.

El bloque try-except brinda un mecanismo para intentar ejecutar un código que podría fallar y capturar la excepción correspondiente, ofreciendo una forma de manejar el error sin interrumpir la ejecución del programa. Esto enriquece la experiencia de usuario y hace que el software sea más tolerante a fallos y sencillo de depurar.

Ejemplo de código en Python que envuelve la conversión de input() en un bloque try-except, manejando el error cuando el usuario ingresa texto en lugar de un número.

```python
# Manejo de Excepciones al Usar input()
try:
    # Se solicita la edad al usuario y se intenta convertir a entero
    edad = int(input("Ingrese su edad: "))
    print("Tu edad es:", edad)
except ValueError:
    # Se captura el error si la conversión falla (por ejemplo, al ingresar
    print("Error: Por favor, ingresa un número válido.")
```

Imagen 4: Manejo de Excepciones al Usar input()

Damián Nicolalde Rodríguez. (2024). Manejo de Excepciones al Usar input().

Para complementar y profundizar, se sugieren acceder a este recurso:

- **Título del enlace relacionado**: Excepciones en Python: manejo de errores con try-except
- **Descripción del enlace relacionado:** El video "Excepciones en Python: manejo de errores con try-except" explica cómo capturar y manejar errores de manera elegante, evitando que el programa se bloquee al presentarse condiciones inesperadas (por ejemplo, convertir una cadena no numérica a entero). Se muestran ejemplos prácticos que ilustran cómo envolver el código en un bloque try-except, detectar el tipo de excepción y responder con mensajes claros al usuario.

- **Enlace:** EXCEPCIONES en PYTHON: Manejo de errores con Try-Except

Referencias citadas en la Clase 5.

- https://elibro.puce.elogim.com/es/ereader/puce/230298
- https://puce.odilo.us/info/facil-aprendizaje-estructuras-de-datos-algoritmos-c-aprenda-facilmente-estructuras-de-datos-graficamente-03127232
- https://puce.odilo.us/info/aprende-c-en-un-fin-de-semana-03105596

Definición de los términos citados en la Clase 5.

Secuencialidad: principio según el cual las instrucciones de un programa se ejecutan de manera lineal y en orden estricto, sin bifurcaciones ni saltos lógicos. Cada línea de código se ejecuta inmediatamente después de la anterior, sirviendo de base para la programación fundamental.

Leer datos: Proceso de solicitar y recibir información ingresada por el usuario (u otra fuente) para su posterior uso en el programa. En Python, se realiza principalmente mediante la función input(), que retorna una cadena que puede convertirse a tipos numéricos o booleanos para su procesamiento.

Profundización Clase 5.

Recurso_profundizacion_clase5.docx

PUCE Virtual

La excelencia no se improvisa

síguenos