Programación I

Condicionales y Anidamiento

Clase 7

Ingeniería en ciberseguridad

La excelencia no se improvisa

1. INTRODUCCIÓN DE LA CLASE

¡Bienvenidos a la Clase 7! Hasta el momento, hemos explorado la ejecución secuencial y la entrada/salida básica en Python. Ahora, profundizaremos en el manejo de decisiones dentro de un programa, enfocándonos en condicionales (if, else, elif) y la manera en que podemos anidar estas estructuras para enfrentar múltiples escenarios. Imagina que, hasta este punto, tu software ejecutaba las instrucciones de forma lineal; con las sentencias condicionales, podrás bifurcar la lógica y responder a situaciones diversas, desde verificar rangos de valores hasta procesar opciones de un menú.

En esta sesión también aprenderemos a usar operadores relacionales (>, <, ==, etc.) que te permitirán comparar valores numéricos y cadenas para tomar decisiones en tiempo de ejecución. Dominar las estructuras condicionales es crucial para crear aplicaciones interactivas y robustas, pues el programa deja de ser un simple flujo lineal y se convierte en un sistema capaz de reaccionar según las condiciones que se presenten. ¡Comencemos con esta herramienta esencial en la programación!

Clase 7: Condicionales y anidamiento (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:

7. **Condicionales y anidamiento (Parte 1)**

En esta Clase 7, nos centraremos en dos temas que amplían la capacidad de un programa para responder a distintas situaciones:

- **Condicionales if, else, elif:** Aprenderemos a manejar estructuras condicionales simples y múltiples, evaluando condiciones que bifurcan la ejecución del programa. Veremos ejemplos que van desde la verificación de un valor único hasta la selección entre varias opciones.
- **Uso de operadores relacionales (>, <, ==, etc.):** Analizaremos cómo comparar valores numéricos y cadenas para determinar si algo es mayor, menor, igual o distinto, creando la base lógica para las decisiones en if-elif-else. Observaremos la importancia de la exactitud en la comparación de cadenas (orden lexicográfico) y la relevancia de >=, <=, etc.

Al finalizar esta clase, podrás crear programas que no solo sigan un flujo secuencial, sino que evalúen condiciones y respondan de manera distinta según el caso, estableciendo las bases para anidar múltiples decisiones en el futuro.

7.1. Sentencias if, else, elif.

Las sentencias condicionales representan uno de los pilares fundamentales de la programación, ya que permiten que un programa rompa su linealidad y ejecute diferentes bloques de código en función de valores o situaciones específicas. Sin estas estructuras, el flujo de ejecución se vería obligado a seguir un camino único

de principio a fin. Con if, else y elif, el software puede reaccionar a la información ingresada, evaluando condiciones que bifurcan la lógica y otorgando un comportamiento más inteligente y adaptativo.

Veamos cómo if, else y elif trabajan conjuntamente para manejar desde decisiones simples (por ejemplo, verificar si una persona es mayor de edad) hasta casos más complejos que involucran múltiples intervalos de valores o diversos escenarios (por ejemplo, clasificar notas, edades, o estados). Además, revisaremos la importancia de la legibilidad cuando se anidan condicionales y cómo evitar un uso excesivo que dificulte la comprensión del código.

7.1.1. If y Else Básico

La forma más elemental de toma de decisiones en Python se da mediante la estructura if-else. Con ella, el programa evalúa una condición booleana y, dependiendo de si esta es verdadera (True) o falsa (False), ejecuta uno de dos bloques de código. Observemos el siguiente ejemplo:

```
edad = int(input("Ingrese su edad: "))
if edad >= 18:
    print("Mayor de edad.")
else:
    print("Menor de edad.")
```

1. **Evaluación de la Condición:** La expresión edad >= 18 se verifica para determinar si es True o False. Si edad es 18 o mayor, la condición resulta True; de lo contrario, es False.
2. **Rama Verdadera (If):** En caso de ser True, el programa ejecuta el bloque de instrucciones alineado debajo de if, en este ejemplo, imprime "Mayor de edad."
3. **Rama Falsa (Else):** Si la condición es False, el programa salta al bloque asociado a else, mostrando "Menor de edad."
4. **Interrupción de la Secuencialidad Pura:** Hasta este punto, un programa secuencial habría continuado con la siguiente línea sin bifurcar. Sin embargo, la estructura if-else divide el flujo en dos posibles caminos, con resultados distintos.

Ventajas del If-Else Básico:

- El if-else expresa una decisión binaria de forma muy intuitiva: "¿Cumple la condición? Sí/No."
- Ideal para casos sencillos, como verificar si una variable supera cierto umbral, o chequear si un valor coincide con una regla simple (por ejemplo, contraseñas, rangos mínimos).
- Con pocas líneas, se maneja la bifurcación del programa sin necesidad de estructuras adicionales.

Consideraciones:

- Cuando solo existen dos rutas posibles (True/False), el if-else es suficiente. Sin embargo, si se requieren más de dos rutas (por ejemplo, varios intervalos de edad), se recurre a elif (explicado en la siguiente sección).
- El if-else básico es fácil de leer, siempre y cuando la condición sea clara y no se aniden demasiadas lógicas dentro del mismo bloque.

El if-else permite un control básico de flujo, esencial para introducir decisiones simples en un programa, como validar mayor/menor de edad, determinar si un número es positivo/negativo, etc.

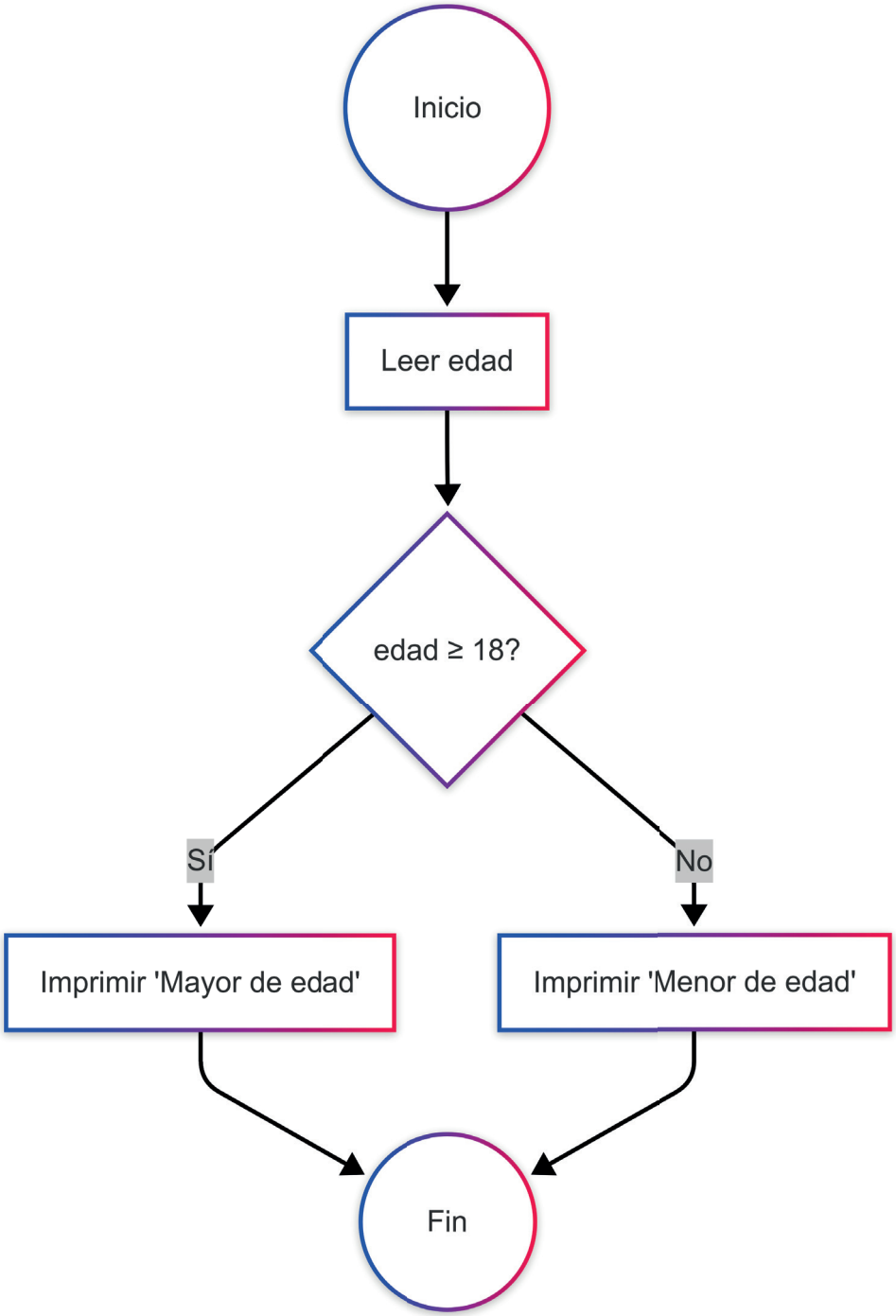

Imagen 1: Diagrama de Flujo de un If-Else

Damián Nicolalde Rodríguez. (2024). Diagrama de Flujo de un If-Else.

7.1.2. **If-Elif-Else para Múltiples Casos**

A menudo, un problema no se limita a dos rutas (True/False), sino que puede requerir varias salidas posibles. Para estos casos, Python introduce la cláusula elif, que te permite encadenar evaluaciones:

```
nota = float(input("Ingrese su nota (0-10): "))
if nota < 5:
    print("Reprobado")
elif nota < 7:
    print("Aprobado con suficiencia")
elif nota < 9:
    print("Aprobado con mérito")
else:
    print("Excelente")
```

1. Cadena de Condiciones: Cada elif se evalúa en orden, y la primera condición que resulte True interrumpe la evaluación de las siguientes, ejecutando el bloque correspondiente.
2. Rama Final (Else): En caso de que ninguna de las condiciones anteriores se cumpla, la cláusula else actúa como "ruta por defecto".
3. Legibilidad vs. Anidamiento: El uso de elif evita la necesidad de anidar varios if dentro de else, algo que podría volverse confuso. Así, la lógica se expresa de manera lineal y clara.

Uso Típico de If-Elif-Else:

- Menús de Opciones: (1 para sumar, 2 para restar, 3 para multiplicar, etc.).
- Clasificación de Valores: Como el ejemplo de notas (0-5 reprobado, 5-7 regular, 7-9 bien, 9-10 excelente).
- Diferentes Rutas de Ejecución: Permite diseñar respuestas múltiples sin complicar el código con anidamientos excesivos.

Ventajas:

- Escalabilidad**:** Soporta fácilmente la adición de nuevos casos (nuevos elif) sin reescribir la lógica anterior.
- Legibilidad Mejorada**:** Comparado con anidar if-else, elif hace la secuencia de evaluaciones más fácil de seguir.

Limitaciones:

- Demasiados elif**:** Aunque elif es preferible al anidamiento, demasiadas condiciones en línea también pueden volverse difíciles de gestionar. En esos casos, puede resultar conveniente usar estructuras de datos (diccionarios) o funciones separadas para manejar la complejidad.

Tabla 1: Ejemplos de Condicionales y Anidamiento

Estructura	Descripción	Ejemplo de Código

If simple	Verifica una única condición y ejecuta su bloque si es True.	if edad >= 18: print(“Mayor de edad”)

If-Else	Maneja un camino alterno cuando la condición no se cumple.	if edad >= 18:
		print(“Mayor de edad”)
		else:
		print(“Menor de edad”)

If-Elif-Else (múltiples)	Soporta varios escenarios; la primera condición verdadera se ejecuta.	if nota < 5:
		print(“Reprobado”)
		elif nota < 7:
		print(“Regular”)
		else:
		print(“Bien”)

If Anidado (a moderar)	Se anidan if dentro de otros if, útil para casos específicos, pero puede volverse confuso si abunda.	if edad > 0:
		if edad < 18:
		print(“Menor”)
		else:
		print(“Mayor”)

Combinación con Lógica	Uso de operadores relacionales y lógicos (and, or, not) para condicionar la ejecución.	if (edad >= 18) and (edad <= 65): print(“Adulto en edad laboral”)

Damián Nicolalde Rodríguez. (2024).

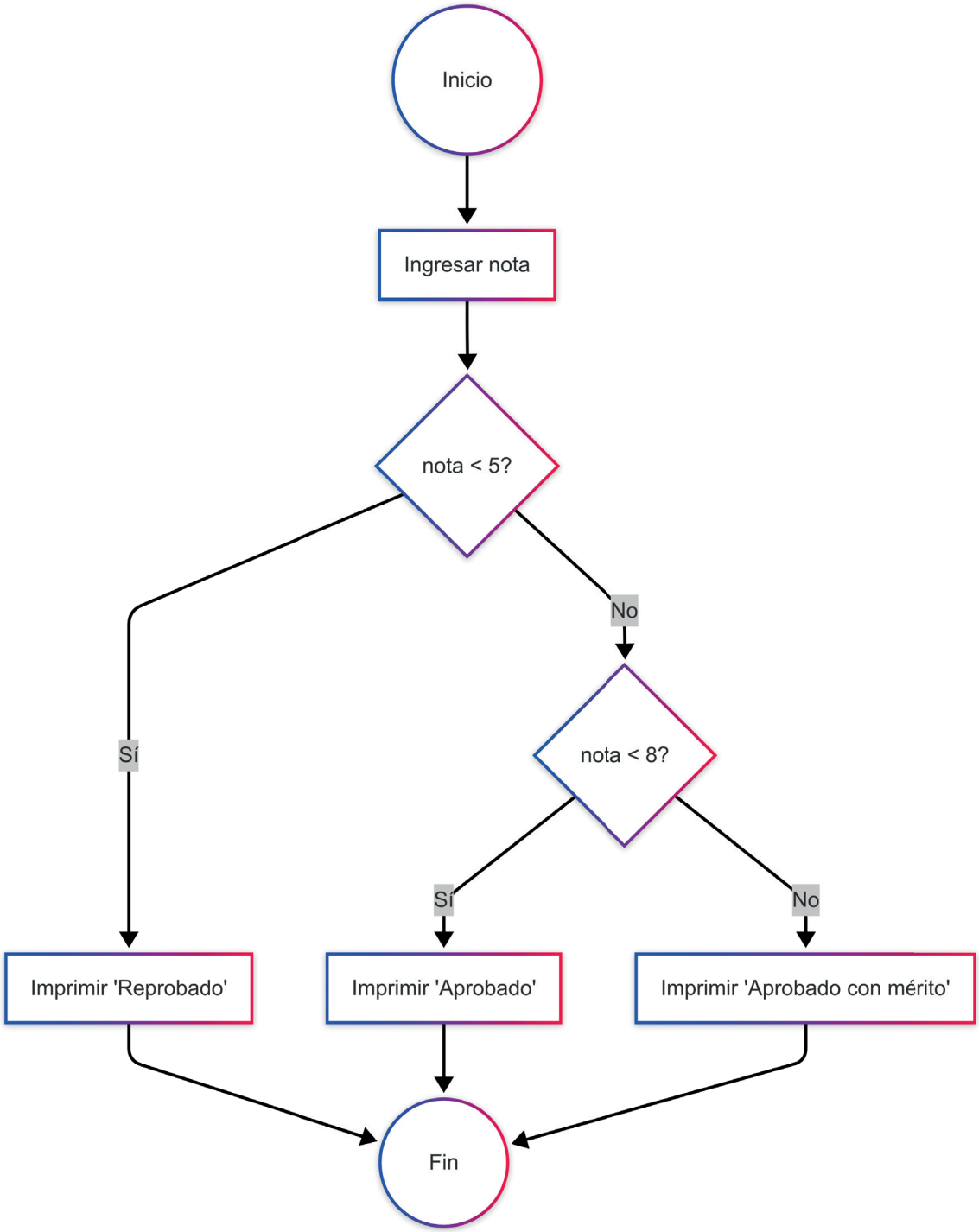

Imagen 2: Salida Diferenciada según Rangos de Nota

Damián Nicolalde Rodríguez. (2024). Salida Diferenciada según Rangos de nota.

7.1.3. **Condicionales Anidados**

Existen casos donde un simple if-elif-else no abarca la lógica jerárquica que deseas expresar. Entonces, es posible anidar condicionales, es decir, colocar un if-else dentro de la rama verdadera o falsa de otro if. Esto crea un flujo en capas:

```
edad = int(input("Edad: "))
if edad >= 18:
    if edad > 60:
        print("Adulto mayor")
    else:
        print("Adulto")
else:
    print("Menor de edad")
```

1. Flujo en Capas: Primero se verifica edad >= 18. Si True, se anida otra condición (edad > 60). Si False, se va a la rama else de "Menor de edad".
2. Riesgo de Legibilidad: Aunque puede ser útil para reflejar lógica compleja, un exceso de anidamiento vuelve el código difícil de seguir.
3. Alternativas: Cuando la complejidad es alta, se recomienda usar elif o dividir la lógica en funciones o bloques separados, evitando profundizar en demasiados niveles de if.

Casos Típicos:

- Situaciones en las que, tras verificar una condición principal, se necesitan subverificaciones (ej. verificar la edad y, si es adulta, ver si supera 60 años).
- Validaciones complejas, donde primero se corrobora un criterio y luego se desglosan subcriterios.

El anidamiento debe usarse con moderación para no comprometer la legibilidad. Si requieres muchos niveles de profundidad, considera reorganizar tu lógica o utilizar elif en su lugar.

Tabla 2: Ejemplos de Condicionales Anidados

Caso	Ejemplo de Código	Comentario

If dentro de If	if edad >= 18:
	if edad >= 60:
	print("Adulto mayor")
	else:
	print("Adulto")
	else:
	print("Menor")
		Primero se determina si la persona es mayor de edad; luego, dentro de esa rama, se evalúa si es un adulto mayor.

If-Elif con Subcondición en else	if nota < 5:
	print("Reprobado")
	elif nota < 7:
	if nota < 6:
	print("Aprobado con D")
	else:
	print("Aprobado con C")
	else:
	print("Excelente")
		Dentro de un elif, se anida otro if para desglosar el rango de nota (5-7) en subrangos (por ejemplo, 5-6 y 6-7).

Uso de Operadores Lógicos + Anidado	if (edad >= 18) and (edad <= 60):
	if edad < 30:
	print("Adulto joven")
	else:
	print("Adulto maduro")
		Combina un if con operadores lógicos y, dentro, se anida otra verificación para subclasificar la edad.

Damián Nicolalde Rodríguez. (2024).

Las sentencias if, else y elif ofrecen la columna vertebral para que un programa abandone la linealidad y responda a condiciones variables. Con ellas, podemos clasificar datos, decidir rutas de ejecución, construir menús, validar rangos, entre otros. Sin embargo, es vital mantener la legibilidad, usando elif para varios casos y moderando el anidamiento, a fin de que el código sea fácil de entender y de mantener. De este modo, tu software se vuelve mucho más versátil, capaz de adaptarse a situaciones diferentes sin perder claridad en la lógica.

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

- Título del enlace relacionado: Estructura Condicional IF, ELSE, ELIF en Python
- **Descripción del enlace relacionado:** Este video explica el funcionamiento de las estructuras condicionales IF, ELSE y ELIF, enseñando cómo tomar decisiones en programas según condiciones lógicas. El contenido muestra ejemplos prácticos desde comparaciones simples hasta condiciones múltiples, ayudando a principiantes a dominar este concepto fundamental de programación en Python..
- **Enlace:** Estructura Condicional IF, ELSE, ELIF en Python | Curso Python 3 🐍 # 13

7.2. Uso de operadores relacionales (>, <, ==, etc.).

Para que un programa tome decisiones de forma efectiva, no basta con las estructuras condicionales en sí (if, else, elif). Es necesario comparar valores para determinar si cumplen o no una determinada condición. En este sentido, los operadores relacionales resultan fundamentales, ya que permiten establecer relaciones lógicas entre dos operandos (ya sean numéricos, cadenas o incluso otros tipos) y producir un resultado booleano (True o False). Estas evaluaciones constituyen la base sobre la cual las sentencias condicionales deciden qué ruta de ejecución seguir.

La relevancia de los operadores relacionales se hace evidente en múltiples escenarios: desde validar si una edad supera un umbral (mayoría de edad) hasta clasificar cadenas alfabéticamente o verificar si un valor se encuentra dentro de un intervalo. A continuación, profundizaremos en los operadores principales, sus usos en datos numéricos y en cadenas, y los detalles que se deben tener en cuenta para evitar resultados inesperados o errores sutiles.

7.2.1. **Principales Operadores**

Python dispone de seis operadores relacionales clave:

1. **== (Igualdad):**
    - Verifica si ambos operandos tienen el mismo valor**.**
    - Retorna True si son iguales; False en caso contrario.
    - Uso típico: comprobar si una variable coincide con un valor esperado (por ejemplo, if edad == 18: ...).
2. **!= (Desigualdad):**
    - Evalúa si los operandos difieren en su valor.
    - Retorna **True** si no son iguales; **False** si son idénticos.
    - Uso típico: para descartar un caso particular (por ejemplo, if nombre != “”: ... para asegurar que la cadena no está vacía).
3. **> (Mayor que):**
    - Devuelve **True** si el operando izquierdo es mayor que el derecho.
    - Se aplica principalmente en datos numéricos, aunque con ciertas restricciones también puede usarse en cadenas, comparando su orden lexicográfico.
    - Ejemplo: if salario > 1000: ... determina si un salario supera un cierto umbral.
4. **< (Menor que):**
    - Retorna **True** si el operando izquierdo es menor que el derecho.
    - Similar a >, se utiliza para verificar si un valor es inferior a otro.
    - Ejemplo: if nota < 5: ... para clasificar notas como reprobadas cuando estén por debajo de 5.
5. **>= (Mayor o igual):**

- Combina el significado de “mayor” con “igual”.
- Retorna **True** si el valor izquierdo es **mayor o igual** que el derecho.
- Ejemplo: if edad >= 65: ... para determinar si alguien es un adulto mayor.

6. **<= (Menor o igual):**
   - Retorna **True** si el valor izquierdo es **menor o igual** que el derecho.
   - Uso frecuente en intervalos, como if 0 <= nota <= 10: para verificar que la nota se halle dentro de un rango válido.

Ejemplo:

```
x = 10
y = 5


if x > y:
    print(“x es mayor que y”)
if x == 10:
    print(“x vale 10”)
```

- **Primera Comparación (x > y):** Verifica si 10 > 5, que resulta True, imprimiendo “x es mayor que y”.
- **Segunda Comparación (x == 10):** Evalúa si x es igual a 10, también True, imprimiendo “x vale 10”.

Observa cómo estas evaluaciones habilitan que el programa tome distintas acciones basadas en la relación entre x y y. Sin estos operadores, el software se vería obligado a ejecutar una sola ruta lineal, sin la capacidad de responder de forma diferenciada a los valores.

Ventajas de los Operadores Relacionales:

- Expresan relaciones matemáticas o alfabéticas de forma directa (edad >= 18, nombre1 < nombre2).
- Funcionan tanto en enteros, flotantes y, en cierto grado, con cadenas.
- Se combinan con operadores lógicos (and, or, not) para construir condiciones complejas.

Riesgos y Errores Frecuentes:

- Un error común es usar = en lugar de ==, lo que provoca un fallo de sintaxis o de lógica.
- Intentar comparar un número con una cadena (ej. “10” < 5) puede derivar en errores. Asegúrate de que ambos operandos sean del mismo tipo o que la comparación tenga sentido semántico.
- Al comparar valores flotantes, las imprecisiones de punto flotante pueden resultar en resultados inesperados. Puede requerirse una tolerancia o margen de error.

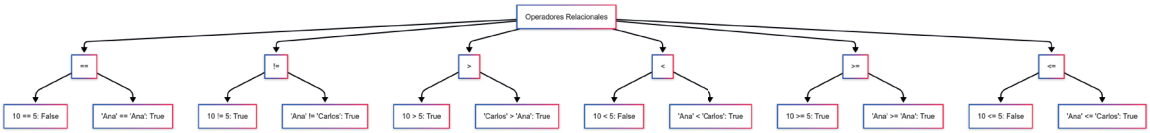

Imagen 3: Operadores Relacionales en un Diagrama de jerarquía

Damián Nicolalde Rodríguez. (2024). Operadores Relacionales en un Diagrama.

7.2.2. Comparaciones Numéricas y de Cadenas

Aunque los operadores relacionales suelen asociarse de manera inmediata con números, Python también permite comparar cadenas, aplicando un orden lexicográfico. A continuación, profundizamos en ambos contextos.

Comparaciones Numéricas

1. **Verificación de Rangos:** Es muy habitual querer comprobar si un valor numérico (por ejemplo, una nota, un salario, una edad) se halla en un intervalo. Python facilita esto con la posibilidad de encadenar comparaciones, como 0 <= nota <= 10, lo que mejora la legibilidad en comparación con un if anidado (ej. if nota >= 0 and nota <= 10:).

2. **Igualdad vs. Asignación:** Se debe distinguir entre = (asignar valor a una variable) y == (comparar si dos valores son iguales). Un error típico consiste en usar = en un if, provocando un fallo. Por ejemplo:

```
x = 5  # Asignación

if x == 5:  # Comparación

   print(“x es 5”)
```

   Cambiar == por = resultaría en un error o en una lógica incorrecta (dependiendo de la versión de Python y el contexto).

3. **Precisión en Flotantes:** En muchos escenarios (por ejemplo, cálculos científicos, monetarios), comparar dos flotantes con == puede dar resultados sorprendentes debido a pequeñas imprecisiones de punto flotante. Por ejemplo, 0.1 + 0.2 no es exactamente 0.3. Para evitar falsos negativos, se recomienda emplear una tolerancia (ej. abs((0.1 + 0.2) - 0.3) < 1e-9).

4. **Uso Práctico en la Ejecución Secuencial:** Un programa secuencial puede solicitar datos (edad, nota, salario) y, antes de continuar, usar operadores relacionales para chequear su validez (edad >= 0, nota <= 10). Si la comparación falla, puede desviar el flujo a un else que notifique el error o solicite un nuevo valor.

Comparaciones con Cadenas

1. **Orden Lexicográfico:** Python compara cadenas carácter por carácter según la tabla Unicode (o ASCII). Esto significa que “Ana” < “Carlos” es True porque “A” precede a “C” en el alfabeto. Sin embargo, “abc” < “Abc” puede ser True o False dependiendo de cómo se interpreten las mayúsculas. En ASCII, el código de ‘A’ (65) es menor que el de ‘a’ (97), por lo que “A” < “a”.

2. **Implicaciones Prácticas:**

- **Ordenar Listas de Nombres:** Se pueden usar estas comparaciones para determinar qué cadena precede a otra, facilitando la clasificación alfabética.
- **Filtrar o Buscar:** Combinando operadores relacionales y lógicos, podemos filtrar nombres que comiencen con cierta letra o que se encuentren en un rango alfabético (por ejemplo, if "Carlos" < nombre < "Luis": ...).
- **Comparación de Contraseñas:** if password == "segura": ... evalúa la exactitud de la cadena ingresada.

3. **Consideraciones de Mayúsculas/Minúsculas:** Dado que "Z" puede ser menor que "a" en ASCII, la comparación de cadenas puede resultar en comportamientos contraintuitivos si no se tiene en cuenta la normalización o conversión (.lower(), .upper()). Para casos más complejos, existen librerías de localización que aplican reglas específicas del idioma.

Ejemplo Combinado:

```
nombre1 = "Carlos"
nombre2 = "Ana"


if nombre1 < nombre2:
    print("Carlos precede a Ana en orden alfabético.")
else:
    print("Ana precede a Carlos o ambos son iguales.")
```

Aquí, "Carlos" < "Ana" se evalúa carácter por carácter. 'C' (67 en ASCII) vs 'A' (65 en ASCII). Al ser 'C' mayor, la condición resulta False, imprimiendo la rama else.

Tabla 3: Comparaciones Numéricas y de Cadenas

Tipo de Comparación	Ejemplo	Resultado
Numérico (==, >, <)	x = 5 if x > 3: print("Mayor que 3")	Imprime "Mayor que 3" si x = 5
Cadenas (==, !=, <)	if "Ana" < "Carlos": print("Ana precede a Carlos")	True si "Ana" precede a "Carlos" en orden lexicográfico

Rango numérico	nota = 8 if 0 <= nota <= 10: print(“Nota válida”)	Verifica que nota se ubique entre 0 y 10 inclusive
Mezcla con Operadores Lógicos	if (edad >= 18) and (edad < 65): print(“Adulto laboral”)	Combina relacionales con lógicos (and, or) para mayor flexibilidad

Damián Nicolalde Rodríguez. (2024).

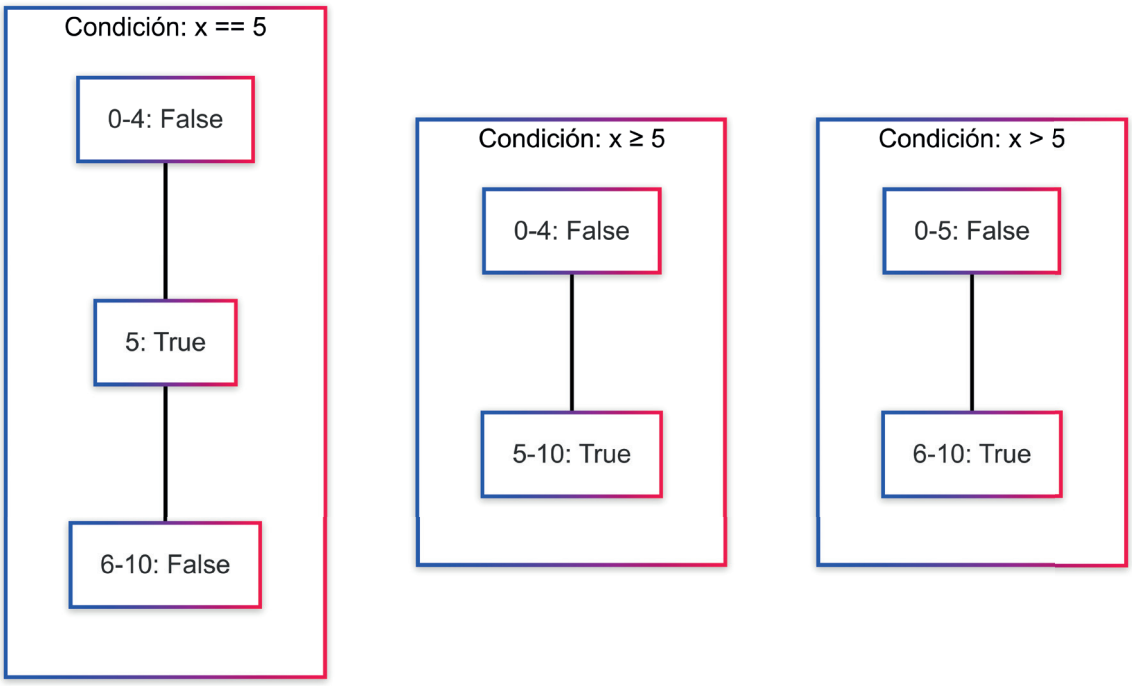

Imagen 4: Visualización de Comparaciones

Damián Nicolalde Rodríguez. (2024). Visualización de Comparaciones.

Los operadores relacionales (==, !=, >, <, >=, <=) constituyen el núcleo de la toma de decisiones en un programa, ya que permiten determinar si un valor es igual, distinto, mayor o menor que otro. Su aplicación abarca tanto datos numéricos como cadenas, con particular énfasis en la comparación alfabética en estas últimas. Comprender los matices de cada operador —así como la precisión en flotantes o el orden lexicográfico en strings— es esencial para que las sentencias if, else y elif funcionen según lo esperado, evitando sorpresas o errores sutiles.

Al combinar estos operadores con la lógica booleana (and, or, not), se pueden construir condiciones aún más complejas, abriendo paso a un sinfín de posibilidades en el control del flujo de un programa. Por ejemplo, verificar si un valor numérico está en un rango determinado y, simultáneamente, si un texto precede a otro alfabéticamente, todo dentro de una misma condición. De esta forma, tu software adquiere la flexibilidad necesaria para manejar escenarios variados y tomar decisiones inteligentes basadas en los datos ingresados.

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

- **Título del enlace relacionado**: Operadores Relacionales en Python
- **Descripción del enlace relacionado:** Este video del Curso Python explica los operadores relacionales (como >, <, ==, !=, >=, <=) que permiten comparar valores y devolver resultados booleanos (True/False). El contenido muestra cómo estos operadores funcionan con distintos tipos de datos y su aplicación en estructuras condicionales. Se presentan ejemplos prácticos para que los principiantes entiendan cómo implementar comparaciones en sus programas Python..
- **Enlace:** Curso Python 🐍| Operadores Relacionales en Python ⏸️◀️▶️

Referencias citadas en la Clase 7.

- 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 7.

Tomar decisiones: En programación, tomar decisiones consiste en que el software evalúe una o varias condiciones y, según su resultado (verdadero o falso), seleccione una ruta de ejecución distinta. Esto permite que un programa abandone la linealidad, responda a múltiples escenarios y ejecute acciones específicas dependiendo de valores o situaciones, haciendo su comportamiento más dinámico y adaptable.

Tiempo de ejecución: El tiempo de ejecución se refiere al período en el que un programa se está ejecutando en un sistema, recibiendo entradas, procesándolas y produciendo salidas. Durante este lapso, se aplican validaciones, toma de decisiones y manejo de errores, definiendo cómo el software reacciona a los datos y condiciones reales que se presentan mientras corre.

Profundización Clase 7.

Recurso_profundizacion_clase7.docx

PUCE Virtual

La excelencia no se improvisa

síguenos