Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

Clase grabada

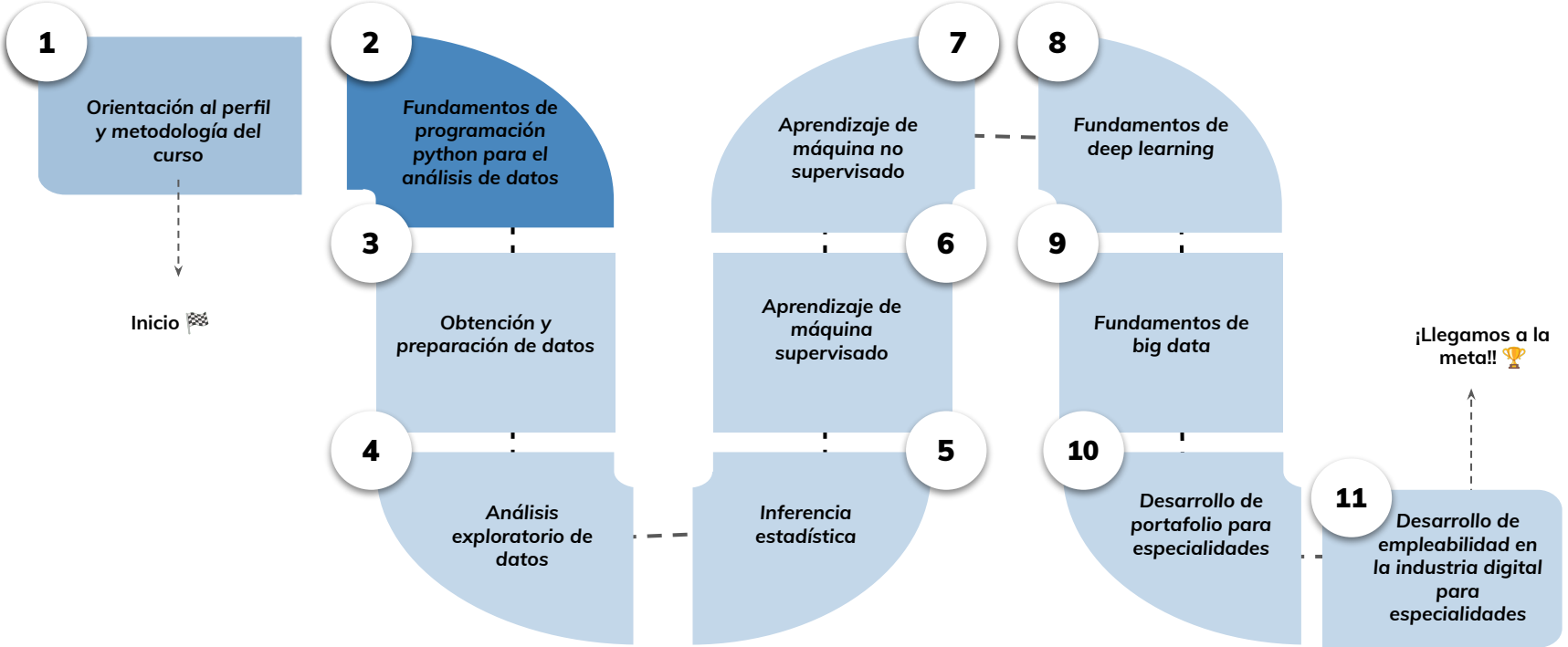
Sentencias condicionales - Parte 1

Aprendizaje Esperado 3: Codificar una rutina utilizando estructuras condicionales y expresiones booleanas para resolver un problema de baja complejidad de acuerdo al lenguaje Python.

alkemy

Hoja de ruta

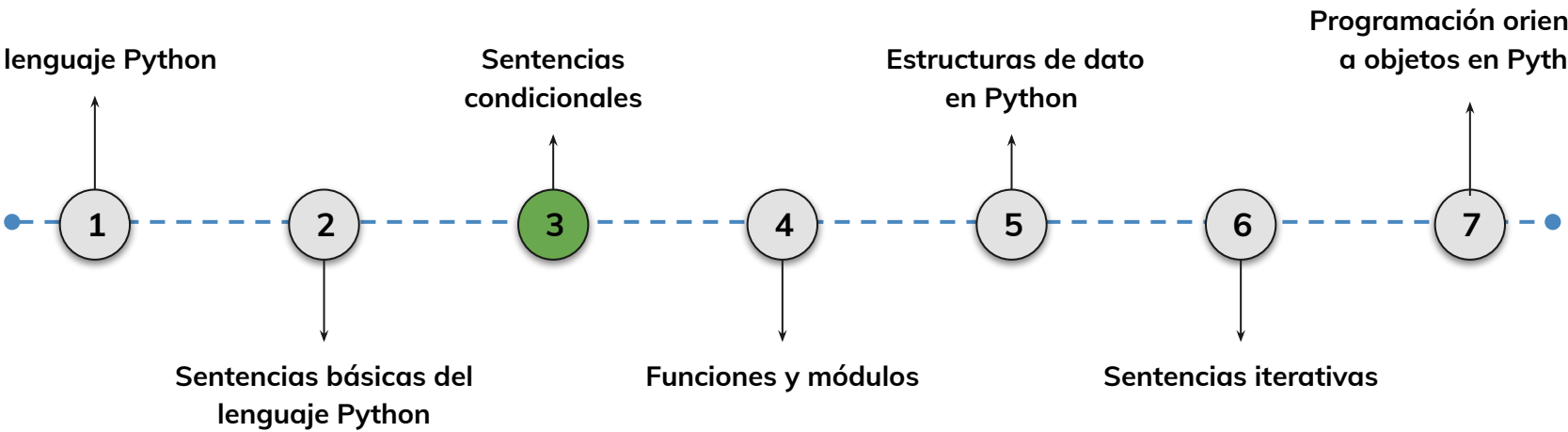
¿Cuáles skills conforman el programa? **Fundamentos de Ciencia de Datos**



alkemy

Roadmap de lecciones

*¿Cuáles **lecciones** estaremos estudiando en este módulo?*



alkemy

Learning Path

¿Cuáles temas trabajaremos hoy?

3.

Diseño de lógica condicional en Python para resolver situaciones reales

En esta clase aprenderemos a tomar decisiones lógicas en nuestros programas utilizando sentencias condicionales y operadores booleanos. Construiremos rutinas dinámicas que respondan a condiciones específicas.

- **Sentencias condicionales básicas**
  - **if, else, elif**
  - **Ejecución de bloques según condiciones**
- **Operadores booleanos y de comparación**
  - **and, or, not**
  - **==, !=, >, <, >=, <= y uso de paréntesis**

alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Comprender la función de las sentencias condicionales en la lógica de un programa
- Utilizar if, else y elif para controlar el flujo de ejecución
- Construir expresiones booleanas usando operadores lógicos
- Evaluar condiciones múltiples mediante el uso de paréntesis
- Aplicar operadores de comparación en rutinas prácticas

Repaso clase anterior

¿Quedó alguna duda?

En la clase anterior trabajamos 📚:

- → *Aplicamos variables, tipos de datos fundamentales y operaciones básicas*
- → *Usamos input() para capturar datos y print() para mostrar resultados*
- → *Aprendimos a convertir entre tipos (int(), float(), str())*
- → *Construimos una rutina simple de interacción con el usuario basada en entradas y cálculos*

Sentencias condicionales

Introducción a las Sentencias Condicionales

Las sentencias condicionales son fundamentales en cualquier lenguaje de programación, ya que permiten ejecutar diferentes bloques de código en función de si una condición es verdadera o falsa. En Python, las estructuras condicionales son especialmente importantes para controlar el flujo de un programa, permitiendo tomar decisiones lógicas basadas en datos o en los resultados de operaciones previas. Estas sentencias ayudan a que el programa sea más interactivo, adaptable y capaz de responder a diversas situaciones.

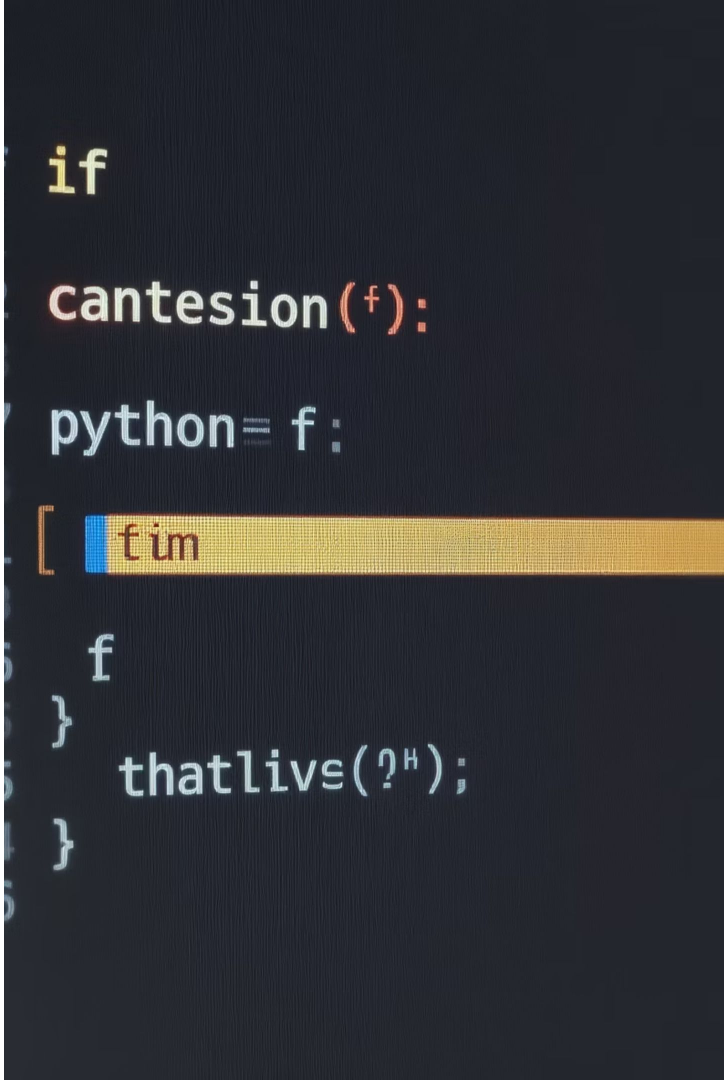
Base de las Sentencias Condicionales

Las sentencias condicionales de Python se basan en operadores de comparación y operadores booleanos que evalúan expresiones y devuelven valores de verdad. Con estas herramientas, es posible construir decisiones simples y complejas, adecuadas para gestionar cualquier flujo lógico en el programa. Este manual presenta las sentencias condicionales y los operadores necesarios, explicando cómo utilizarlos con ejemplos prácticos.

Sección 1: Qué es una Sentencia Condicional

Concepto de Sentencia Condicional

Una sentencia condicional es una estructura que permite ejecutar un bloque de código solo si se cumple una condición específica. En Python, las sentencias condicionales se expresan mediante palabras clave como if, else y elif. Estas estructuras permiten al programa tomar decisiones, lo cual es fundamental para crear aplicaciones dinámicas y adaptativas.



Importancia de las Sentencias Condicionales

Sin las sentencias condicionales, un programa no podría responder a diferentes situaciones ni personalizar la ejecución en función de los datos de entrada o de los resultados de operaciones previas.

Por ejemplo, una sentencia condicional permite verificar si el usuario tiene la edad suficiente para acceder a un contenido, realizar operaciones en función del estado de una variable o calcular resultados específicos basados en ciertas condiciones.

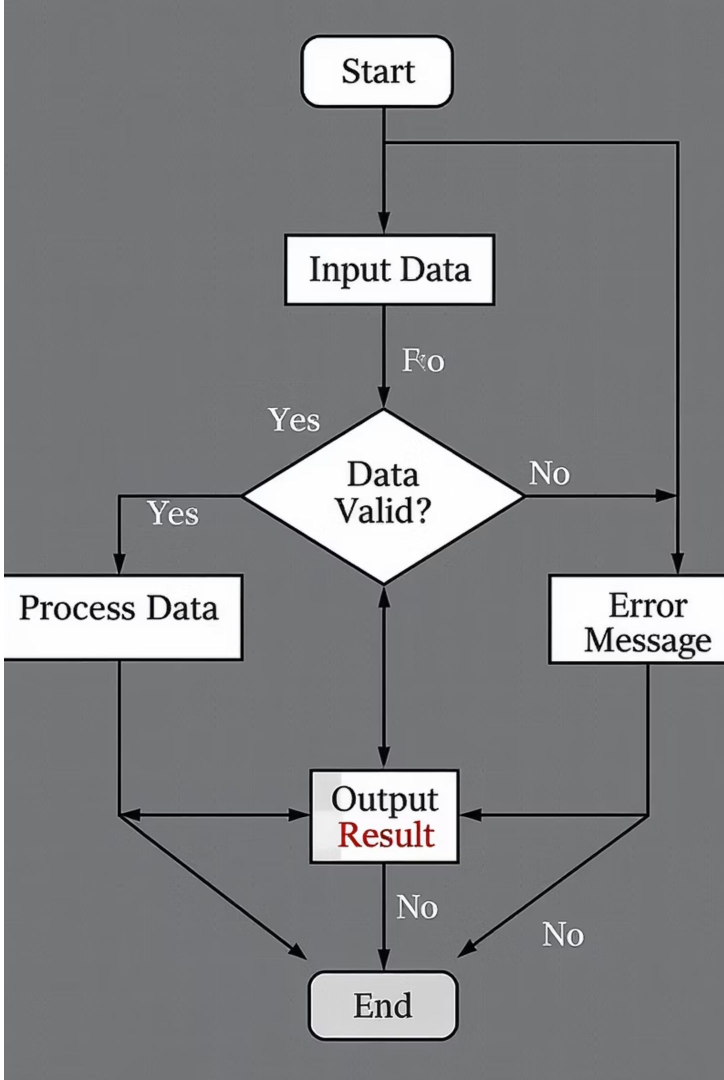
Ejemplo Básico de Condicional

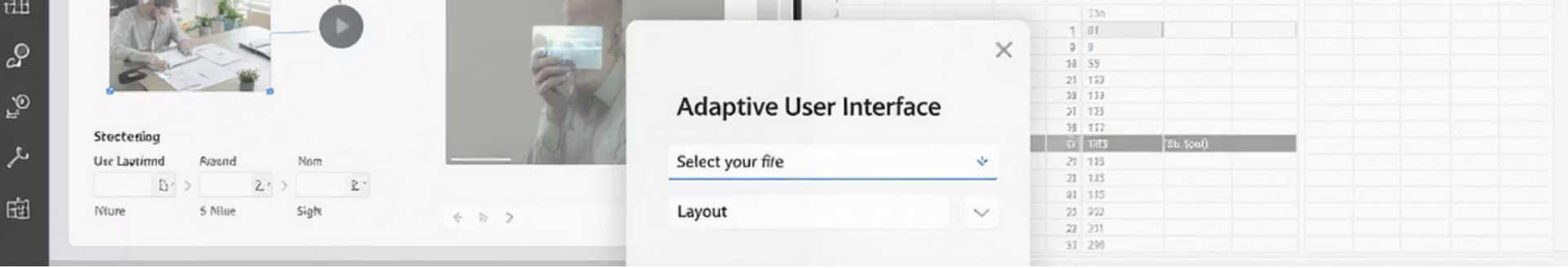
En este ejemplo, el programa verifica si la variable edad es mayor o igual a 18 para determinar el mensaje adecuado. Dependiendo del valor de edad, el programa elige uno de los bloques para ejecutarse.

```
edad = 18
if edad >= 18:
    print("Eres mayor de edad.")
else:
    print("Eres menor de edad.")
# Salida: Eres mayor de edad.
```

La Importancia de las Condiciones en la Lógica de un Programa

Las condiciones son esenciales para controlar el flujo de ejecución de un programa y determinar qué acciones deben realizarse en cada caso. Esto permite crear programas interactivos y personalizados que reaccionan a los datos de entrada, lo cual es crucial en aplicaciones como juegos, sistemas de gestión de usuarios, formularios en línea y mucho más.





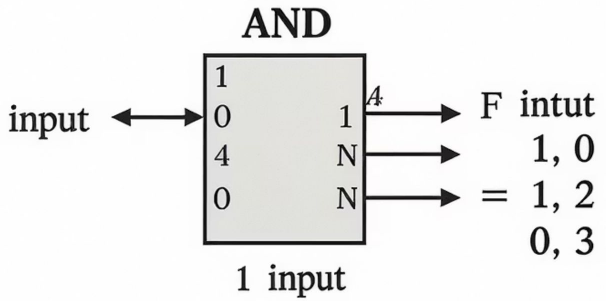
Adaptabilidad y Flexibilidad en Programación

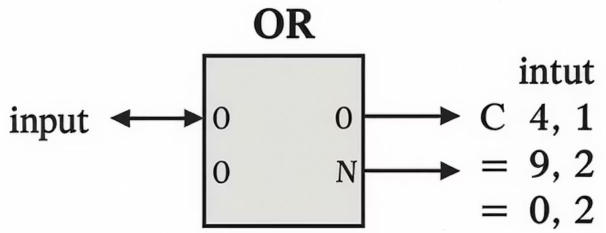
Gracias a las sentencias condicionales, los programas pueden manejar excepciones, validar entradas de usuario y ejecutar diferentes lógicas de procesamiento según el contexto. Esto mejora la adaptabilidad de las aplicaciones y permite ofrecer experiencias de usuario más ricas y personalizadas.

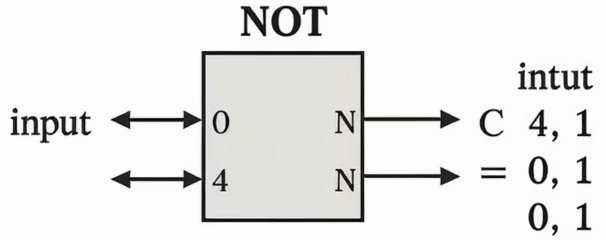
Sección 2:
Operadores Booleanos

Concepto de Operadores Booleanos

Los operadores booleanos son operadores lógicos que permiten construir expresiones de verdad o falsedad en un programa. Estos operadores son fundamentales para combinar y evaluar múltiples condiciones en una sola expresión lógica. En Python, los operadores booleanos más utilizados son and, or y not. Cada uno de ellos permite combinar o modificar condiciones de manera diferente, ofreciendo flexibilidad en la evaluación lógica.







Operador and

El operador and devuelve True solo si ambas condiciones en una expresión son verdaderas. Si alguna de las condiciones es False, la expresión completa será False.

Tabla de verdad del operador and

True and True = True

True and False = False

False and True = False

False and False = False

Ejemplo de uso de and

En este caso, la persona puede entrar solo si cumple con ambas condiciones: ser mayor de edad y tener identificación.

```
edad = 20
tiene_identificacion = True

if edad >= 18 and tiene_identificacion:
    print("Puedes entrar al club.")
else:
    print("No puedes entrar.")
# Salida: Puedes entrar al club.
```

Operador or

El operador or devuelve True si al menos una de las condiciones en la expresión es verdadera.

Solo será False si ambas condiciones son falsas.

Tabla de verdad del operador or

True or True = True

True or False = True

False or True = True

False or False = False

Ejemplo de uso de or

Aquí, la persona tiene descuento si es estudiante o docente, o ambos.

```
es_estudiante = True
es_docente = False

if es_estudiante or es_docente:
    print("Tienes acceso al descuento.")
else:
    print("No tienes descuento.")
# Salida: Tienes acceso al descuento.
```

Operador not

El operador not invierte el valor de verdad de una condición. Si la condición es True, not la convierte en False, y viceversa.

Tabla de verdad del operador not

not True = False

not False = True

Ejemplo de uso de not

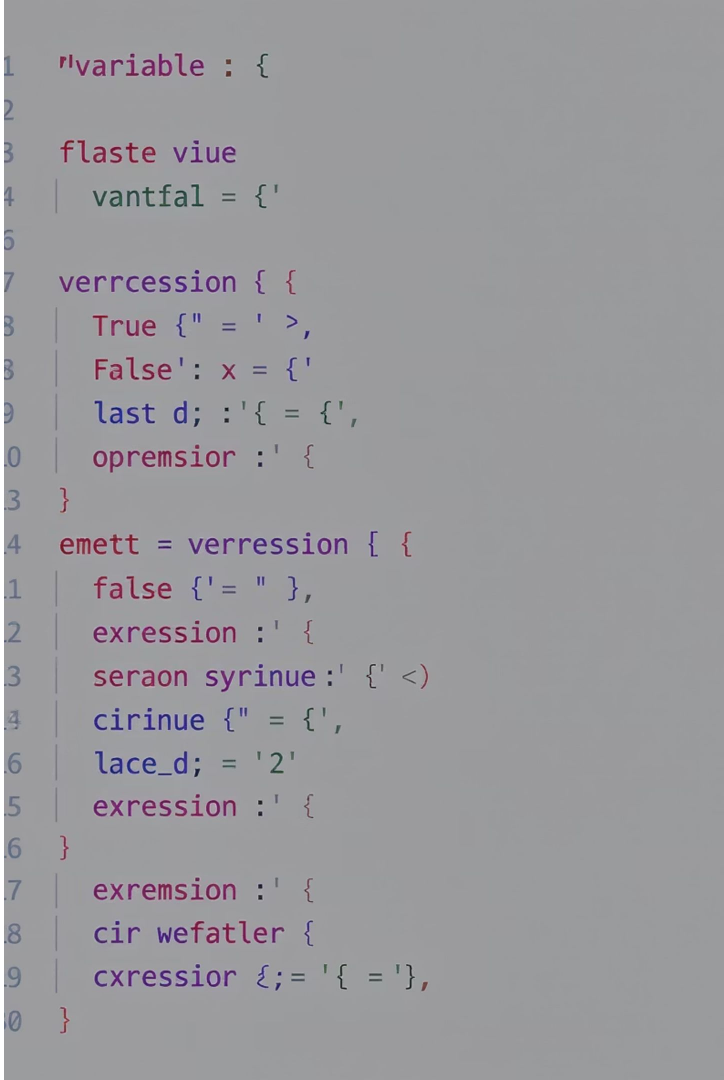
En este ejemplo, se verifica si un usuario no está bloqueado para permitir su acceso al sistema.

```
es_mayor = False

if not es_mayor:
    print("No eres mayor de edad.")
# Salida: No eres mayor de edad.
```

Sección 3: Operadores de Comparación

Concepto de Operadores de Comparación

Los operadores de comparación permiten comparar dos valores y devolver un resultado booleano (True o False). Son esenciales para construir condiciones en sentencias if, ya que permiten determinar si una expresión cumple con ciertos criterios.



Operadores de Comparación en Python

Mayor que (>)

Verifica si el valor de la izquierda es mayor que el de la derecha

Mayor o igual que (>=)

Verifica si el valor de la izquierda es mayor o igual que el de la derecha

Menor que (<)

Verifica si el valor de la izquierda es menor que el de la derecha

Menor o igual que (<=)

Verifica si el valor de la izquierda es menor o igual que el de la derecha

Igual que (==)

Verifica si ambos valores son iguales

Distinto que (!=)

Verifica si ambos valores son diferentes

Operador Mayor Que y Mayor o Igual Que

El operador > se usa para verificar si un valor es mayor que otro, mientras que >= verifica si es mayor o igual. Estos operadores son comunes en comparación de valores numéricos.

```
edad = 21

if edad > 18:
    print("Puedes votar.")
if edad >= 21:
    print("Puedes beber alcohol legalmente en EE. UU.")
# Salida: Puedes votar.
#         Puedes beber alcohol legalmente en EE. UU.
```

Operador Menor Que y Menor o Igual Que

El operador < verifica si un valor es menor que otro, mientras que <= evalúa si es menor o igual. Se usa principalmente en validaciones de límites inferiores.

```
nota = 7

if nota < 5:
    print("Suspenso")
elif nota <= 7:
    print("Aprobado")
# Salida: Aprobado
```

Operador Igual Que

El operador == evalúa si dos valores son iguales. Es uno de los operadores más utilizados en condiciones.

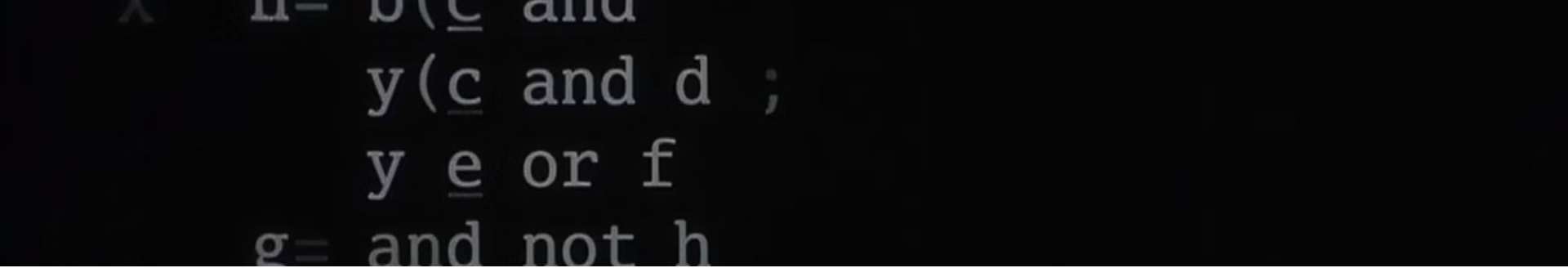
Es importante recordar que el operador == se utiliza para comparar valores, mientras que el operador = se utiliza para asignación de variables.

```
respuesta = "sí"

if respuesta == "sí":
    print("Aceptado")
else:
    print("Rechazado")
# Salida: Aceptado
```

Operador Distinto Que

El operador != devuelve True si dos valores son distintos. Es útil para asegurar que un valor no sea igual a otro.

```
usuario = "admin"

if usuario != "invitado":
    print("Tienes acceso completo.")
# Salida: Tienes acceso completo.
```



Sección 4: Paréntesis y Expresiones Booleanas

Uso de Paréntesis para Agrupar Condiciones

En expresiones complejas, los paréntesis permiten agrupar condiciones para controlar el orden de evaluación. Esto facilita la claridad y la precisión en las expresiones booleanas, asegurando que el programa evalúe primero las condiciones agrupadas.

Ejemplo de uso de paréntesis en una condición

En este ejemplo, los paréntesis agrupan las condiciones relacionadas con la edad y el género, permitiendo una evaluación clara y precisa de la expresión completa.

```
edad = 25
es_miembro = True
codigo_descuento = "VIP"

if (edad > 18 and es_miembro) or codigo_descuento == "VIP":
    print("Descuento aplicado.")
else:
    print("No tienes descuento.")
# Salida: Descuento aplicado.
```

Precedencia de Operadores Booleanos

En Python, los operadores tienen una jerarquía de evaluación: primero se evalúa not, luego and y finalmente or. Sin embargo, el uso de paréntesis permite anular esta precedencia.

Orden de precedencia

1. Paréntesis ()
2. not
3. and
4. or

Los paréntesis siempre tienen la mayor precedencia, lo que permite controlar explícitamente el orden de evaluación.

Combinar Múltiples Condiciones con Paréntesis

Los paréntesis ayudan a combinar múltiples condiciones para definir expresiones complejas. Esto es especialmente útil en programas que requieren verificar varias condiciones antes de ejecutar un bloque de código.

```
salario = 3000
es_tiempo_completo = True
antiguedad = 2

if (salario > 2500 and es_tiempo_completo) or antiguedad > 5:
    print("Eres elegible para el bono.")
else:
    print("No eres elegible para el bono.")
# Salida: Eres elegible para el bono.
```

Mejorar la Claridad en Expresiones Booleanas

Los paréntesis no solo definen el orden de evaluación, sino que también mejoran la claridad y legibilidad del código. Esto es esencial en equipos de desarrollo, ya que facilita la comprensión de expresiones complejas.

Sin paréntesis (confuso)

```
if a > b and c > d or e == f:    # código
```

Con paréntesis (claro)

```
if (a > b and c > d) or e == f:    # código
```

El uso de paréntesis hace que el código sea más fácil de entender y mantener.

Live Coding

¿En qué consistirá la Demo?

Crearemos un pequeño sistema de validación para determinar si un usuario puede acceder a una actividad, según edad, rol o estado.

1. *Solicitar nombre, edad y rol (estudiante/docente/otro)*
2. *Evaluar si puede acceder a una función especial según edad y rol*
3. *Aplicar if, elif, else con operadores and, or, not*
4. *Mostrar un mensaje personalizado en función de la lógica evaluada*
5. *Reforzar el uso correcto de operadores de comparación*
6. *Usar paréntesis para mejorar claridad y orden lógico*
7. *Introducir una condición negativa (not)*
8. *Mostrar todos los resultados con print()*

Tiempo: 25 Minutos

#Momentode Preguntas...

¿Qué diferencia hay entre if y elif?

¿Qué operador se usa para comparar igualdad entre dos valores?

¿Qué hace el operador not en una expresión lógica?

¿Por qué es útil usar paréntesis en condiciones compuestas?

Momento:

Time-out!

⏰5 -10 min.

Ejercicio N° 1

¡Acceso permitido!

¡Acceso permitido!

Contexto: 🙌

Un sistema necesita determinar si un usuario puede acceder a una actividad especial, según sus datos personales y su rol.

Consigna: ✍️

Codificar una rutina utilizando estructuras condicionales y expresiones booleanas para resolver un problema de baja complejidad de acuerdo al lenguaje Python.

Tiempo 🕐**:** *30 Minutos*

Paso a paso: ⚙️

1. Solicitar al usuario:

- nombre
- edad
- rol (estudiante/docente/otro)

2. Evaluar condiciones:

- Si tiene más de 18 años Y es estudiante o docente, permitir acceso
- Si tiene menos de 18, denegar acceso
- Si el rol no es válido, pedir que lo revise

3. Mostrar mensajes personalizados según los distintos caminos
4. Usar if, elif, else con operadores lógicos y comparación

¿Alguna consulta?

alkemy

Resumen

¿Qué logramos en esta clase?

- ✓ ***Comprendimos el uso de sentencias condicionales (if, else, elif)***
- ✓ ***Utilizamos operadores booleanos (and, or, not) para evaluar lógica***
- ✓ ***Aplicamos operadores de comparación para controlar decisiones***
- ✓ ***Construimos una rutina funcional basada en condiciones reales***
- ✓ ***Usamos paréntesis para organizar expresiones compuestas***

¡Ponte a prueba!

Momento de ejercitación

Te invitamos a aprovechar esta última sección del espacio sincrónico para realizar de manera individual las ***actividades disponibles en la plataforma.*** *Estas propuestas son claves para afianzar lo trabajado y* ***forman parte obligatoria del recorrido de aprendizaje.***

💪 Análisis de caso ——— 💪 Selección Múltiple

💪 Comprensión lectora

Si al resolverlas surge alguna duda, compartela o tráela al próximo encuentro sincrónico.

alkemy

¡Muchas gracias!

alkemy