Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

Clase grabada

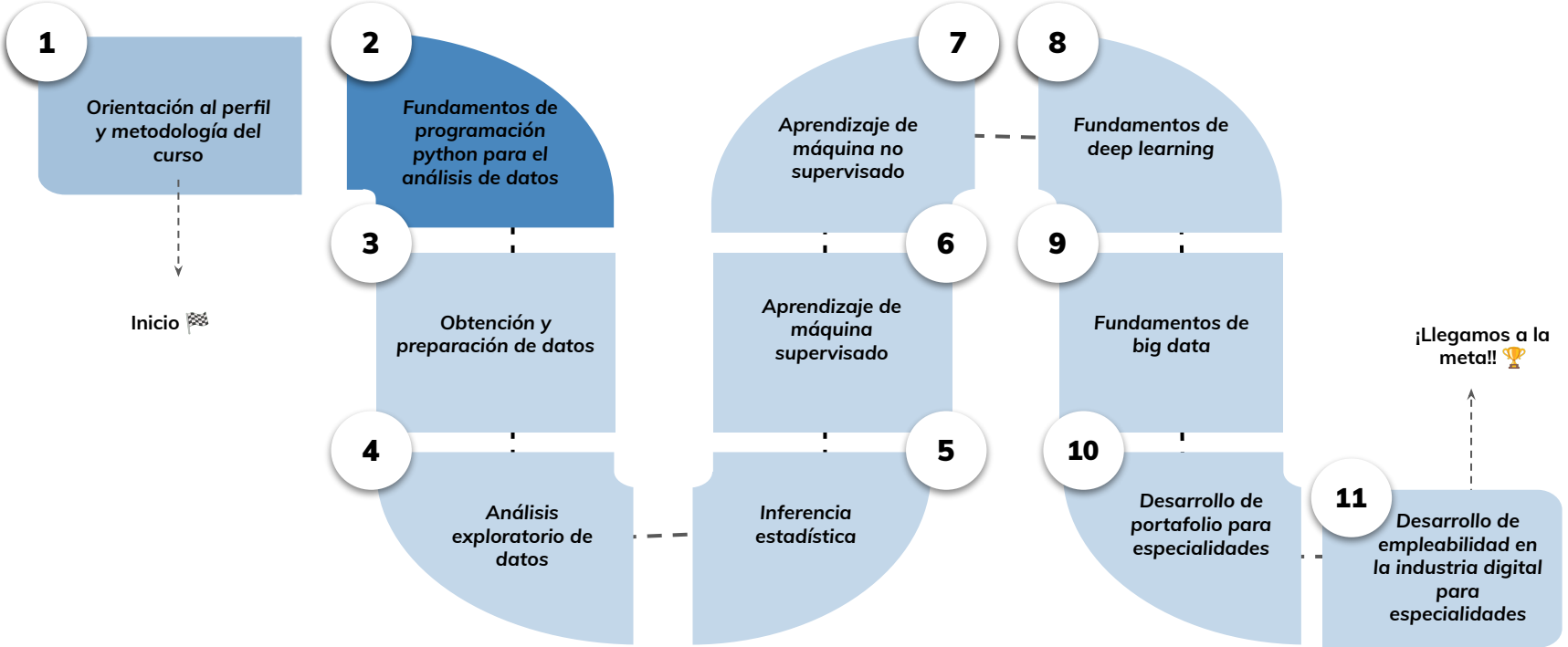
Sentencias básicas del lenguaje python- Parte 2

Aprendizaje Esperado 2: Aplicar el concepto de variable, tipos de dato fundamentales y expresiones aritméticas utilizando el lenguaje python para la creación de una rutina de baja complejidad.

alkemy

Hoja de ruta

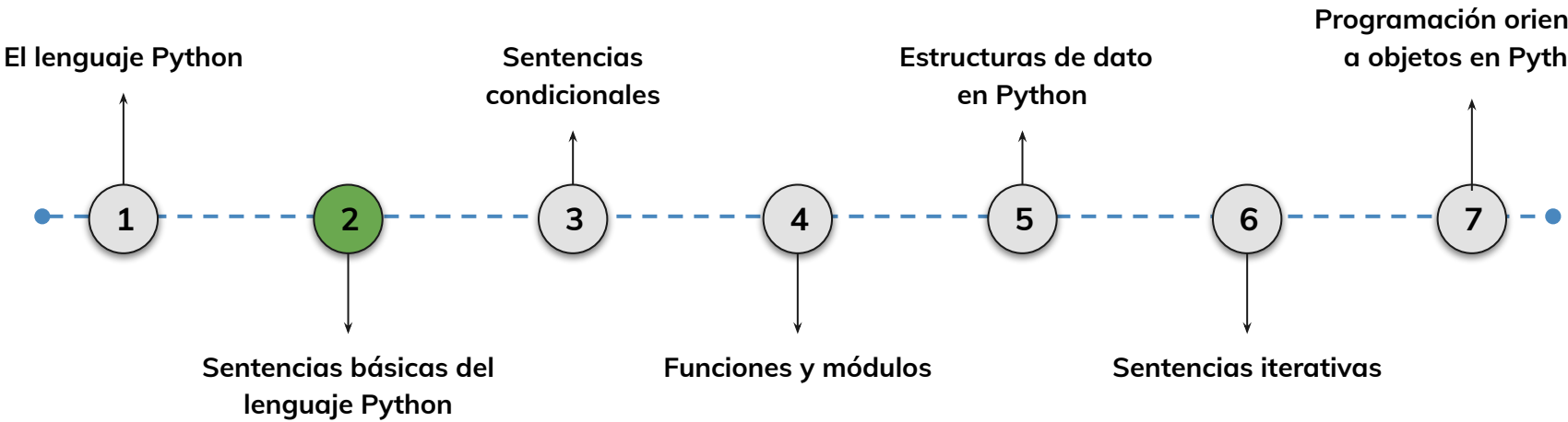
¿Cuáles skills conforman el programa? **Fundamentos de Ciencia de Datos**



alkemy

Roadmap de lecciones

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



alkemy

Learning Path

¿Cuáles temas trabajaremos hoy?

alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Aplicar expresiones aritméticas en Python para resolver problemas simples
- Usar operadores relacionales y lógicos para evaluar condiciones
- Incorporar entradas del usuario mediante input()
- Convertir y manipular tipos de datos para realizar cálculos
- Modelar una rutina de baja complejidad utilizando estructuras básicas

Repaso clase anterior

¿Quedó alguna duda?

En la clase anterior trabajamos 📚:

- → *Aplicamos variables, tipos de datos fundamentales y operaciones aritméticas.*
- → *Vimos cómo crear una rutina simple en Python para resolver cálculos básicos.*
- → *Exploramos ejemplos prácticos con suma, resta, multiplicación, división y más.*
- → *Comprendimos cómo Python gestiona el tipo de variable dinámicamente.*

#Momentode Preguntas...

¿?

¿Qué diferencia hay entre = y == en Python?

¿?

¿Para qué se utiliza la función input() y cómo convierte el tipo de dato?

¿?

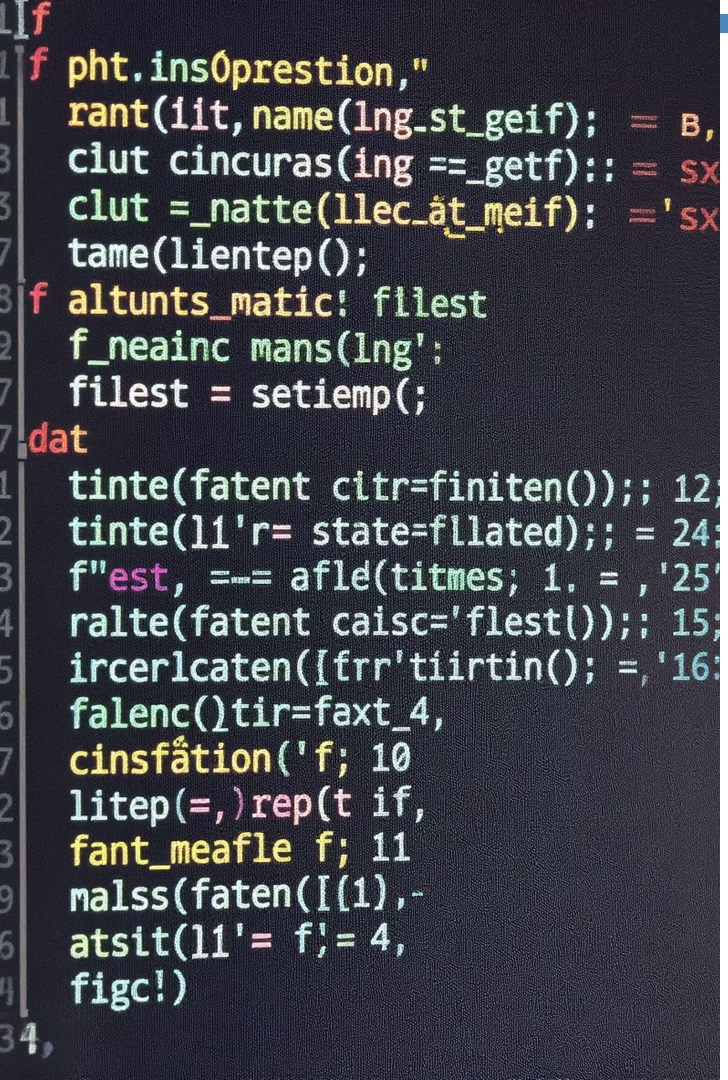
¿Qué operadores permiten evaluar condiciones compuestas?

¿?

¿Por qué es importante el orden de los operadores en una expresión?

Sentencias básicas del lenguaje python

alkemy



Operaciones Compuestas

Python permite combinar varias operaciones en una misma expresión aritmética, respetando el orden de operaciones (PEMDAS: paréntesis, exponentes, multiplicación y división, y finalmente suma y resta).

Esto facilita la creación de expresiones complejas en una sola línea de código, lo que puede ser útil en aplicaciones que requieren cálculos matemáticos elaborados.

alkemy

< >

Uso de Paréntesis para **Controlar el Orden de Operaciones**

El uso de paréntesis en Python permite definir el orden de las operaciones en una expresión aritmética. Esto es particularmente útil para evitar errores y asegurar que los cálculos se realicen de la manera deseada.

Por ejemplo, en la expresión (2 + 3) * 4, los paréntesis aseguran que la suma se realice antes de la multiplicación. Controlar el orden de operaciones es crucial para evitar resultados inesperados y garantizar la precisión de los cálculos.

< >

Ejemplos de Uso de Paréntesis

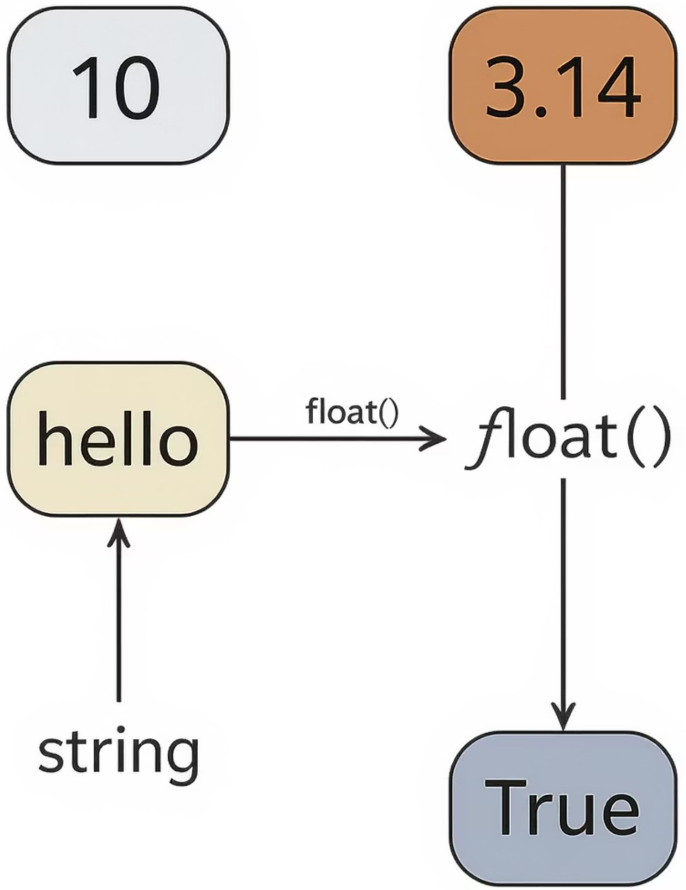
Los paréntesis permiten controlar explícitamente el orden de evaluación de las expresiones, lo que es esencial para obtener los resultados esperados.

```
resultado = 2 + 3 * 4  # Sin paréntesis, multiplica primero
print("Resultado sin paréntesis:", resultado)  # Salida: 14

resultado = (2 + 3) * 4  # Con paréntesis, suma primero
print("Resultado con paréntesis:", resultado)  # Salida: 20
```

Conversiones de Tipo

Las conversiones de tipo en Python permiten transformar datos de un tipo a otro, facilitando operaciones entre diferentes tipos de datos.



Conversión de **Enteros a Flotantes**

Python permite convertir enteros a números de punto flotante utilizando la función float(). Esta conversión es útil en casos donde se necesita precisión decimal, como en cálculos financieros o científicos.

Convertir un entero a flotante también evita posibles errores en operaciones que requieren decimales, como la división, asegurando que el resultado se ajuste al contexto.

< >

Ejemplos de Conversión de Entero a Flotante

La conversión de enteros a flotantes es especialmente útil cuando se necesita precisión decimal en los cálculos.

```
entero = 10
decimal = float(entero)
print("Convertido a decimal:", decimal)  # Salida: Convertido a decimal: 10.0
```

Conversión de **Flotantes a Enteros**

Para convertir un número decimal a entero, se utiliza la función int(). Esta conversión trunca los decimales, lo que puede ser útil en situaciones donde se necesita redondear hacia abajo un valor.

Sin embargo, es importante recordar que la conversión de float a int implica una pérdida de precisión, ya que el valor decimal se elimina. En aplicaciones que requieren redondeo, Python ofrece funciones como round() para redondear al entero más cercano.

< >

Ejemplos de Conversión de Decimal a Entero

Al convertir de float a int, es importante recordar que se pierde la parte decimal del número.

```
numero = 15.7
entero = int(numero)
print("Convertido a entero:", entero)  # Salida: Convertido a entero: 15
```

< >

Conversión de **Números a Cadenas y Viceversa**

La conversión entre números y cadenas es común en Python, especialmente cuando se necesita manipular datos textuales que contienen valores numéricos. La función str() convierte un número en una cadena de texto, permitiendo su concatenación con otras cadenas.

Para convertir una cadena a un número, se utilizan las funciones int() o float(), dependiendo de si se necesita un número entero o decimal.

< >

Ejemplos de Conversión entre Números y Cadenas

Estas conversiones son esenciales para la manipulación de datos en aplicaciones que interactúan con el usuario o procesan información textual.

```
numero = 123
texto = str(numero)
print("Número como texto:", texto)  # Salida: Número como texto: 123
```

Conversión de **Cadenas a Booleanos**

La función bool() convierte una cadena de caracteres en un valor booleano. En Python, las cadenas vacías ("") se consideran False, mientras que cualquier otra cadena se considera True.

Esta conversión es útil en estructuras de control, donde se necesita verificar la existencia de un valor en una cadena o en operaciones que involucren valores condicionales.

< >

Ejemplos de Conversión de Cadena a Booleano

Entender cómo Python interpreta los valores booleanos es crucial para el control de flujo en los programas.

```
cadena = "Python"
es_verdadero = bool(cadena)
print("¿Es verdadero?", es_verdadero)  # Salida: ¿Es verdadero? True
```

Conversión Implícita en Operaciones

Python realiza conversiones de tipo implícitas en algunas operaciones, como al sumar un entero y un flotante, donde el entero se convierte automáticamente en flotante para evitar errores.

Esta característica hace que el código sea más flexible, aunque es importante entender cómo Python realiza estas conversiones para evitar resultados inesperados. Conocer y aprovechar las conversiones implícitas permite escribir código que se adapta automáticamente al tipo de datos involucrado.

< >

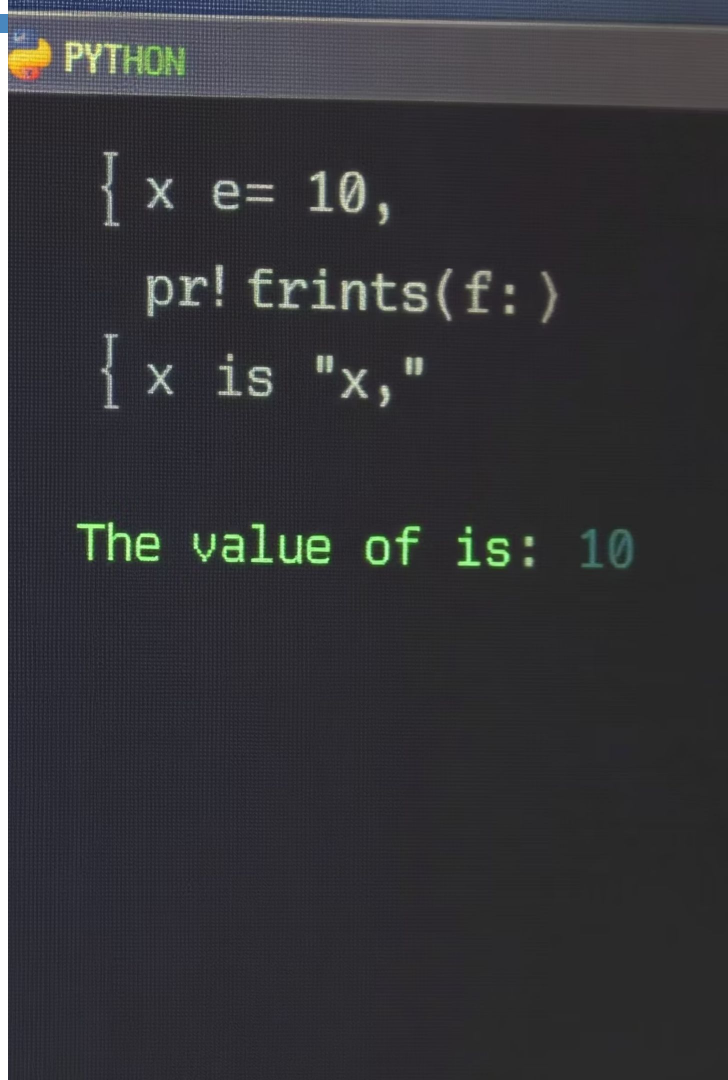
Ejemplos de Conversión Implícita

Las conversiones implícitas simplifican el código al no requerir conversiones explícitas en muchas operaciones

```
entero = 5
decimal = 2.5
resultado = entero + decimal
print("Resultado:", resultado)  # Salida: Resultado: 7.5
```

Impresión en Consola

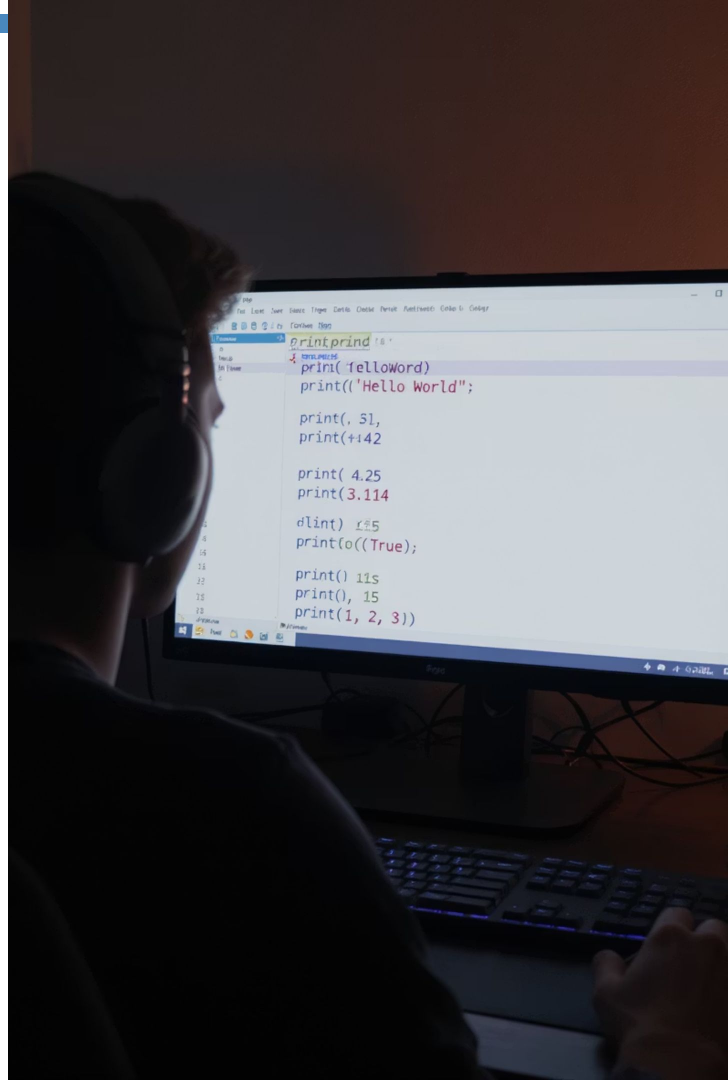
La impresión en consola es una herramienta fundamental para mostrar información al usuario y para depurar programas durante el desarrollo.



Función print() para Mostrar Información

La función print() en Python permite mostrar información en la consola, lo que es útil para presentar resultados, mensajes al usuario y el seguimiento de la ejecución del programa.

Esta función es fundamental para depurar el código y comunicar información, y es ampliamente utilizada en la fase de desarrollo para verificar el comportamiento del programa.



< >

Impresión de Variables y Textos Combinados

Python permite combinar variables y textos en una sola sentencia print() utilizando la concatenación de cadenas o las técnicas de formato. Con la concatenación, se puede unir texto y variables utilizando el operador +.

Sin embargo, en Python es más común utilizar f-strings o el método format() para combinar texto y variables de manera elegante y legible, como en print(f"El resultado es {resultado}").

< >

Ejemplos de f-string para Impresión Combinada

Los f-strings proporcionan una forma elegante y legible de combinar texto y variables en una salida.

```
nombre = "Ana"
edad = 30
print(f"{nombre} tiene {edad} años.")  # Salida: Ana tiene 30 años.
```

< >

Control de la Separación y el Final de la Impresión

La función print() permite controlar el separador entre elementos y el carácter final de la impresión. Por ejemplo, print("Hola", "Mundo", sep="-") imprime "Hola-Mundo" en lugar de agregar un espacio entre las palabras.

También se puede utilizar el parámetro end para definir el carácter final, como en print("Hola", end="!"), lo que evita que se agregue una nueva línea al final de la impresión.

```
    prin(( {World
  sep-="-, end="!)
}
```

```
Hell-World!
```

< >

Impresión de **Formato Avanzado**

Python permite imprimir en formatos avanzados utilizando la interpolación de cadenas, f-strings y el método format(). Esto es útil para controlar la cantidad de decimales en números flotantes, alinear texto y presentar datos de manera organizada.

La impresión de formato avanzado es especialmente útil en aplicaciones que presentan datos al usuario, ya que facilita la legibilidad y organización de la información.

alkemy

< >

Ejemplos de Impresión Avanzada

El formato avanzado permite presentar datos de manera profesional y organizada.

```
precio = 49.99
print(f"El precio es: ${precio:.2f}")  # Salida: El precio es: $49.99
```

alkemy

< >

Usos Prácticos de **print() en Depuración**

La función print() es una herramienta valiosa para la depuración, ya que permite verificar el valor de variables en diferentes partes del programa y rastrear el flujo de ejecución.

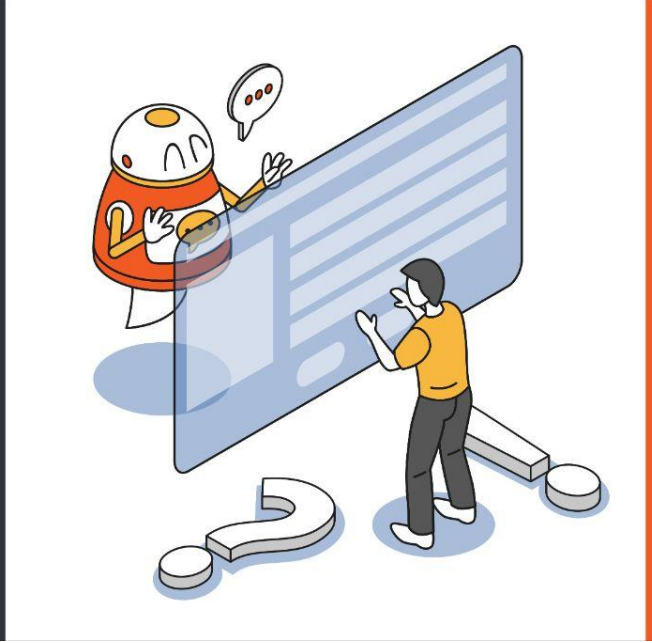
Al imprimir el estado de las variables en momentos clave, los programadores pueden identificar problemas y ajustar el código. Esto es especialmente útil en etapas iniciales del desarrollo, donde el seguimiento y la retroalimentación inmediata son esenciales.

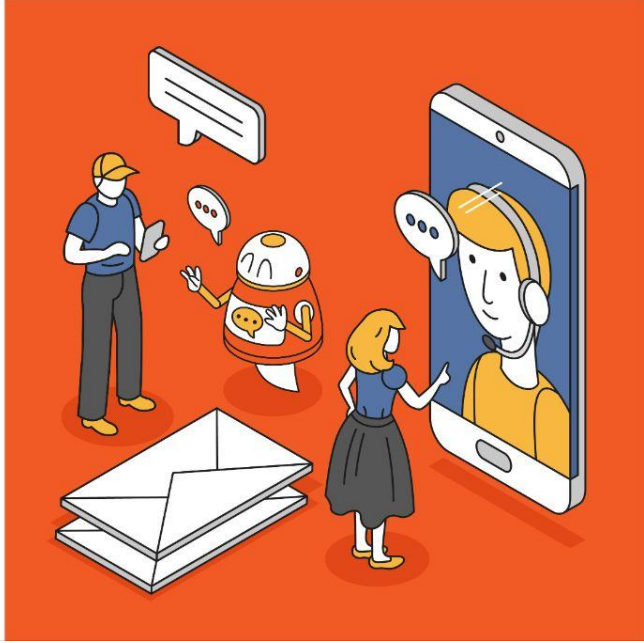
< >

Ejemplos de Depuración con print()

Utilizar print() para depuración permite identificar rápidamente problemas en el código durante el desarrollo.

```
contador = 0
for i in range(5):
    contador += i
    print("Contador:", contador)  # Salida en cada iteración
```





Entrada de Datos en Consola

La entrada de datos en consola permite que los programas interactúen con el usuario, recibiendo información que puede ser procesada durante la ejecución.

< >

Función input() para **Capturar Datos del Usuario**

La función input() permite capturar datos ingresados por el usuario desde la consola, convirtiéndolos automáticamente en una cadena de texto. Esta funcionalidad es esencial en aplicaciones interactivas, ya que permite personalizar la ejecución del programa según las respuestas del usuario.

Por ejemplo, nombre = input("¿Cuál es tu nombre? ") solicita el nombre y lo almacena en la variable nombre.

Ejemplos de input()

La función input() es fundamental para crear programas interactivos que respondan a las entradas del usuario.

```
nombre = input("¿Cuál es tu nombre? ")
print("Hola,", nombre)
```

< >

Conversión de **Entradas de Texto a Otros Tipos de Datos**

Como input() siempre devuelve una cadena, es común convertir los datos ingresados a otros tipos, como enteros o decimales, para realizar cálculos o comparaciones.

Por ejemplo, se puede utilizar int(input("Ingresa tu edad: ")) para capturar la edad como un número entero, permitiendo realizar operaciones aritméticas con el valor ingresado.

Ejemplos de Conversión de input() a Número

La conversión de tipos es esencial cuando se necesita realizar operaciones matemáticas con datos ingresados por el usuario.

```
edad = int(input("¿Cuántos años tienes? "))
print("Tendrás", edad + 1, "años el próximo año.")
```

< >

Manejo de Errores en Entradas de Datos

Al capturar entradas del usuario, es importante anticipar posibles errores, como el ingreso de texto cuando se espera un número. Python permite manejar estos errores mediante sentencias try y except, lo que garantiza que el programa no se detenga abruptamente y que el usuario reciba mensajes de corrección.

Esta validación de entradas es crucial en aplicaciones que requieren datos específicos.

```
try:
    numero = int(input("Ingresa un número: "))
    print("El doble es:", numero * 2)
except ValueError:
    print("Por favor, ingresa un número válido.")
```

Live Coding

¿En qué consistirá la Demo?

Veremos cómo construir una rutina interactiva que solicite información, realice cálculos y devuelva un resultado útil, usando operadores y tipos de datos.

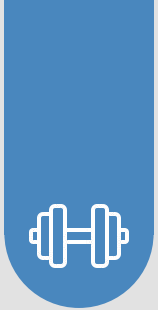
1. *Solicitar nombre y edad al usuario con input()*
2. *Calcular años faltantes para jubilarse*
3. *Leer tres calificaciones numéricas desde el teclado*
4. *Calcular promedio y determinar si aprueba*
5. *Mostrar resultados con print() y expresiones combinadas*
6. *Usar int(), float() y str() para convertir datos*
7. *Aplicar operadores relacionales para mostrar un mensaje condicional*
8. *Agregar una variable booleana como estado del usuario*

Tiempo: 25 Minutos

Momento:

Time-out!

⏰5 -10 min.



Ejercicio N° 1

Calculadora exprés: rutina inteligente con variables

Calculadora exprés: *rutina inteligente con variables*

Contexto: 🙌

Una organización educativa necesita una pequeña herramienta para calcular el rendimiento académico de estudiantes de forma automatizada. Para eso te piden construir una rutina básica en Python.

Consigna: ✍️

Aplicar el concepto de variable, tipos de dato fundamentales y expresiones aritméticas utilizando el lenguaje Python para la creación de una rutina de baja complejidad.

Tiempo 🕛**:** *30 Minutos*

Calculadora exprés: *rutina inteligente con variables*

Paso a paso: ⚙️

1. Pedir al usuario su nombre, edad y tres calificaciones numéricas
2. Calcular:
    a. promedio de las calificaciones
    b. edad en meses
    c. si la persona aprueba (promedio ≥ 6)
3. Mostrar un mensaje personalizado con los resultados
4. Usar al menos 2 conversiones de tipo y 1 comparación lógica

¿Alguna consulta?

alkemy

Resumen

¿Qué logramos en esta clase?

- ✓ ***Aplicamos variables, tipos de datos y operaciones básicas con Python***
- ✓ ***Usamos input() para capturar información del usuario***
- ✓ ***Comprendimos cómo hacer conversiones entre tipos de datos***
- ✓ ***Construimos expresiones lógicas y aritméticas combinadas***
- ✓ ***Creamos una rutina de baja complejidad con interacción real***

¡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 clave 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