Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

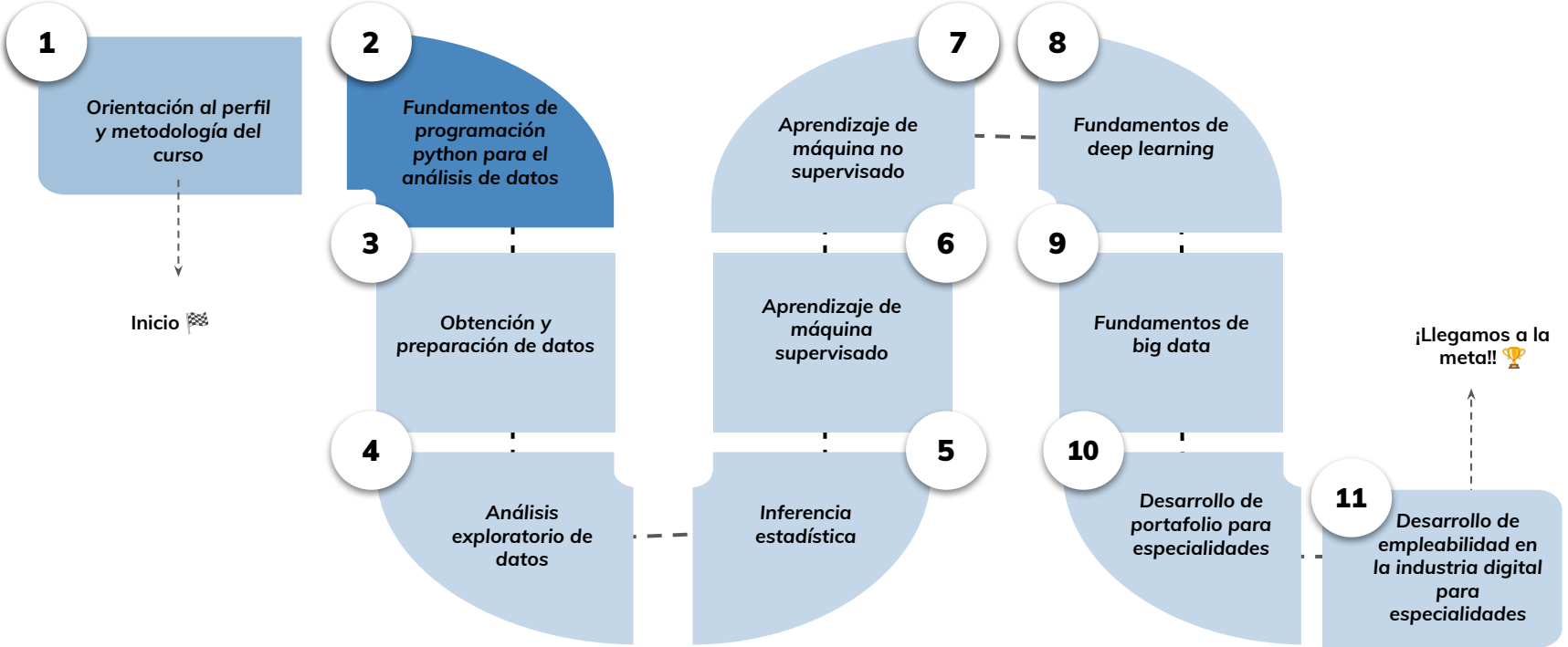
Clase grabada

› Sentencias iterativas - Parte I

Aprendizaje Esperado 6: Codificar una rutina utilizando sentencias iterativas para resolver un problema de baja complejidad en python.

Hoja de ruta

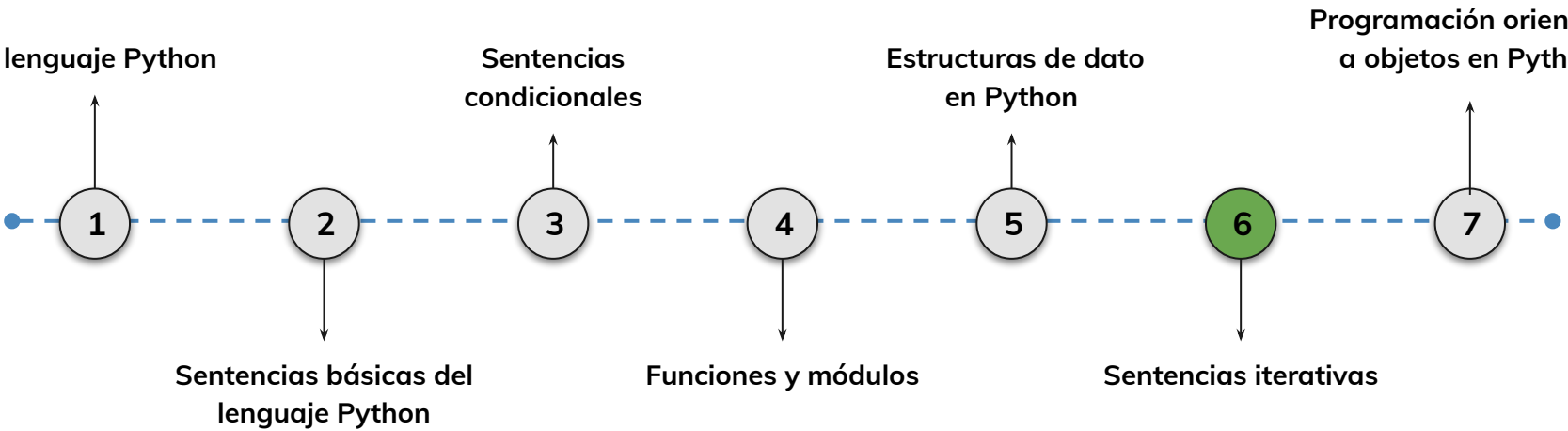
¿Cuáles skills conforman el programa? **Fundamentos de Ciencia de Datos**



alkemy

Roadmap de lecciones

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



alkemy

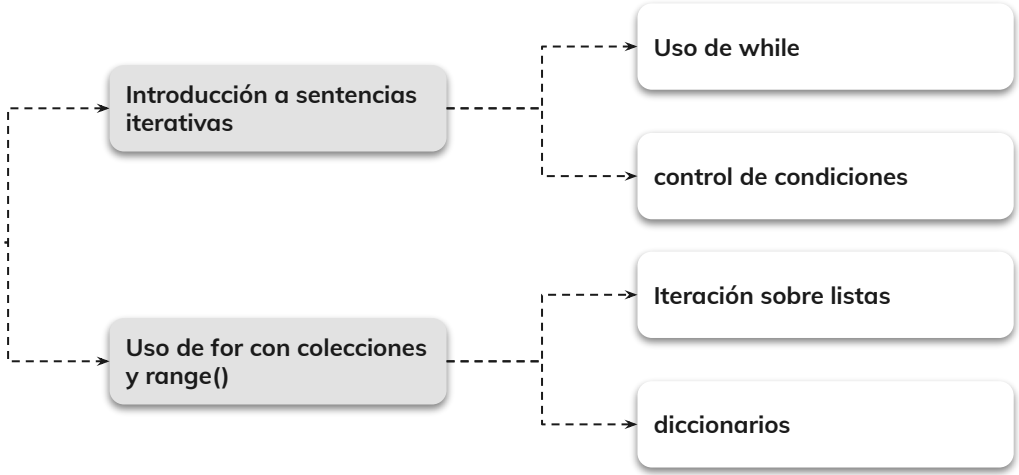
Learning Path

¿Cuáles temas trabajaremos hoy?

6.

Uso de bucles en Python para automatizar tareas repetitivas.

Aprenderemos a utilizar estructuras iterativas (while y for) para ejecutar bloques de código múltiples veces. Identificamos cuándo usar cada una y cómo aplicarlas para recorrer colecciones, controlar repeticiones y simplificar algoritmos.



alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Comprender qué son las sentencias iterativas y su función
- Utilizar bucles while y for para controlar el flujo del programa
- Recorrer listas y diccionarios con for
- Aplicar range() para generar secuencias numéricas
- Emplear comprensiones de listas y diccionarios para transformar datos

Repaso clase anterior

¿Quedó alguna duda?

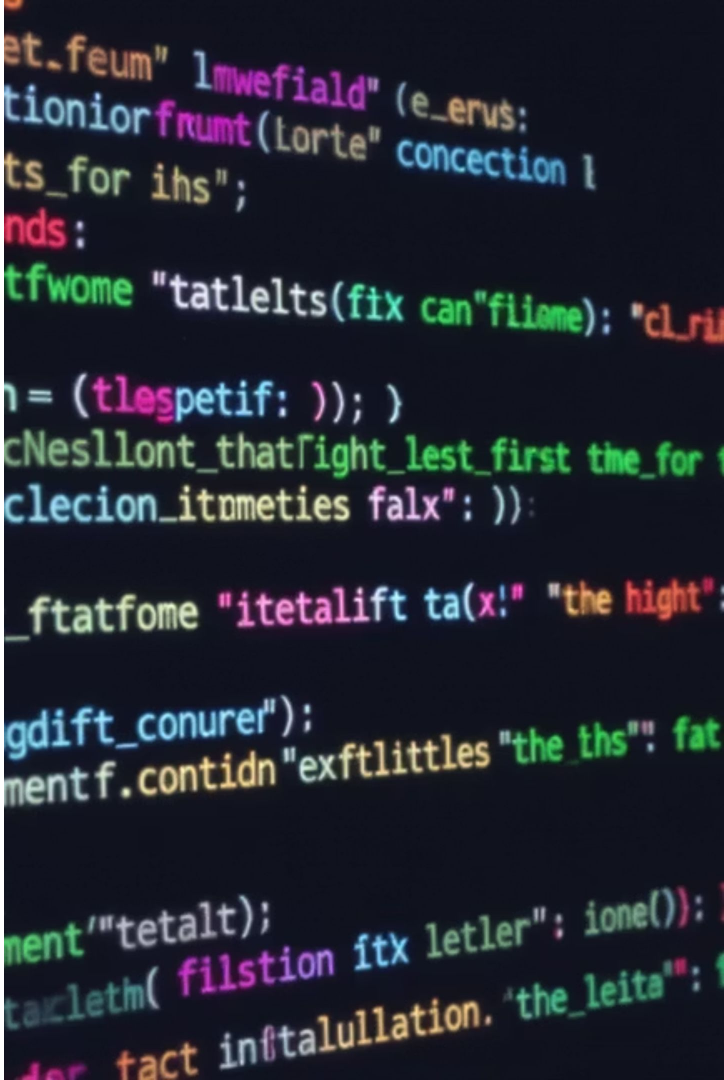
En la clase anterior trabajamos 📚:

- → *Aplicamos estructuras de datos como listas, diccionarios, tuplas y sets*
- → *Creamos estructuras anidadas para modelar datos complejos*
- → *Usamos comprensiones para transformar y filtrar colecciones*
- → *Reforzamos la selección de estructuras según la situación a resolver*

Sentencias iterativas

Sentencias iterativas

En el mundo de la programación, una de las necesidades más comunes es realizar operaciones repetitivas de forma automática. Las **sentencias iterativas** en programación permiten que un bloque de código se ejecute múltiples veces, facilitando el trabajo en tareas repetitivas, especialmente cuando se trabaja con grandes cantidades de datos o se requiere realizar la misma operación sobre una colección de elementos.



Introducción a las Sentencias Iterativas

Las **sentencias iterativas** en programación permiten que un bloque de código se ejecute múltiples veces, facilitando el trabajo en tareas repetitivas, especialmente cuando se trabaja con grandes cantidades de datos o se requiere realizar la misma operación sobre una colección de elementos. Estas sentencias ahorran tiempo, minimizan errores y hacen que el código sea más legible y eficiente.

Herramientas de Iteración en Python

Python proporciona estructuras iterativas muy poderosas que se pueden emplear en diversas situaciones. Las sentencias iterativas en Python incluyen while y for, las cuales se complementan con herramientas como la función range para controlar el número de iteraciones.

While

Ejecuta un bloque de código mientras una condición sea verdadera

For

Itera a través de elementos en una colección o secuencia

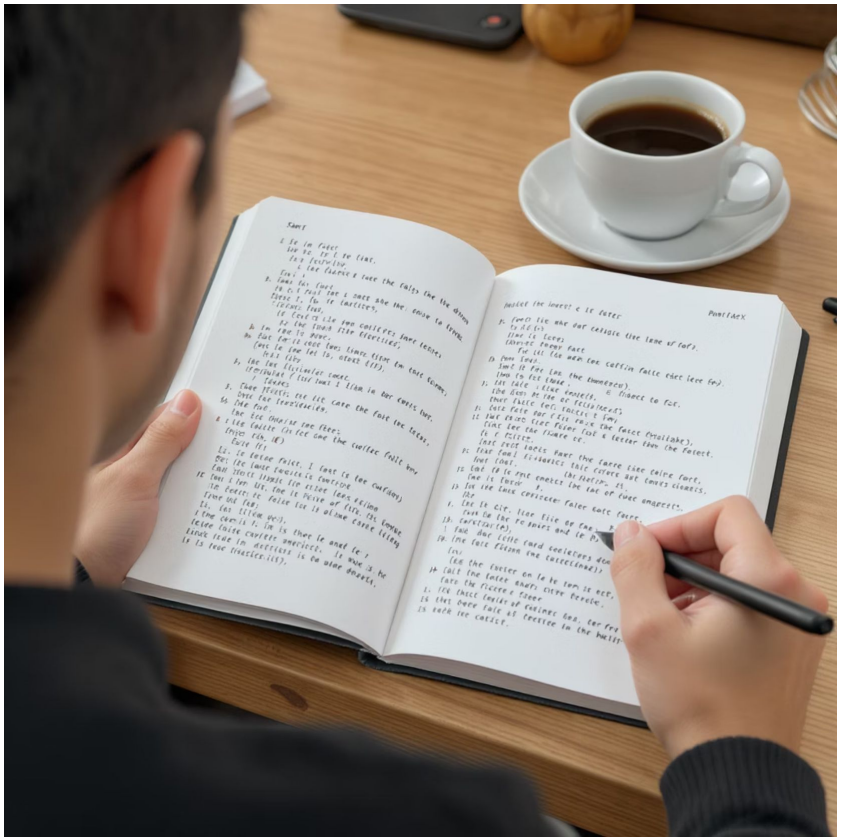
Range

Genera secuencias numéricas para controlar iteraciones

Propósito del Manual

El propósito de este manual es proporcionar una comprensión clara de cada tipo de sentencia iterativa y su importancia en la programación. Cada tema cuenta con ejemplos prácticos para facilitar la comprensión y permitir al lector aplicar estos conocimientos en sus propios proyectos.

La idea es ofrecer una base sólida para que, al finalizar el manual, puedas emplear de manera efectiva las sentencias iterativas en tus programas de Python.



< >

Desarrollo

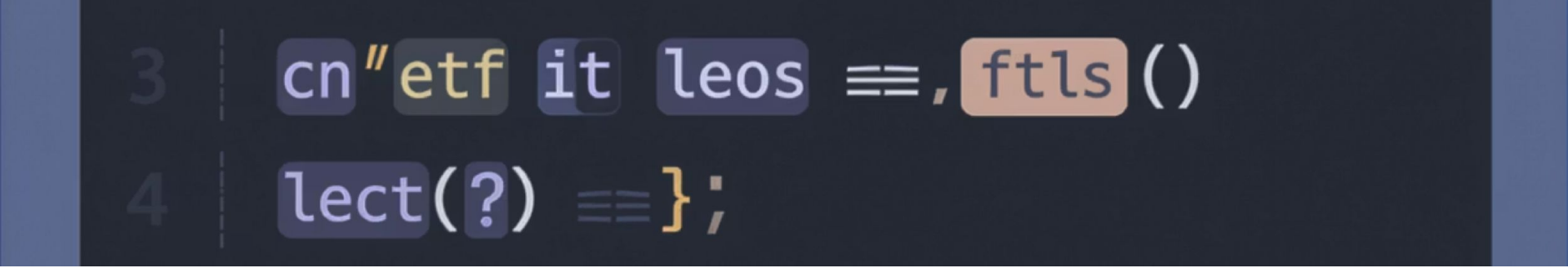
Sentencias Iterativas

A lo largo de esta clase, exploraremos en profundidad estas herramientas, viendo ejemplos prácticos y comprendiendo su utilidad en distintos contextos de programación.

Ejemplo con FOR:

```
# Iterar sobre una lista de números e imprimir solo los pares
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print("Números pares en la lista:")
for numero in numeros:
    if numero % 2 == 0:
        print(numero)
```



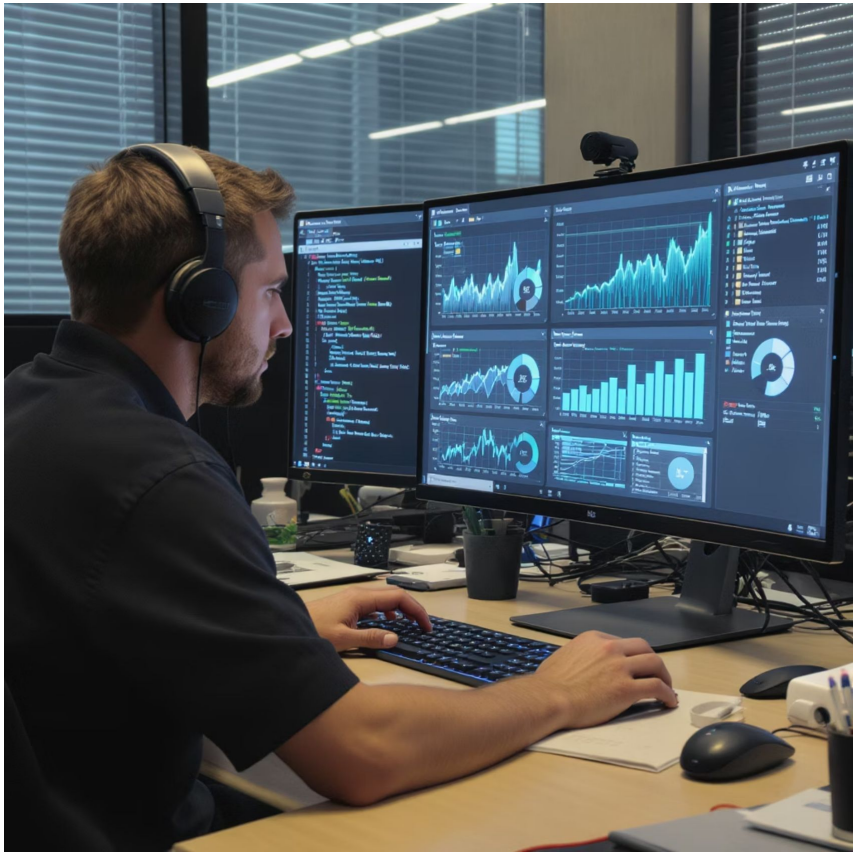
¿Qué es una Sentencia Iterativa?

Las **sentencias iterativas** son bloques de instrucciones que permiten ejecutar un conjunto de operaciones repetidamente. Su principal ventaja es la eficiencia: en lugar de escribir el mismo código una y otra vez, se puede estructurar en un bucle, lo que simplifica el código y facilita su mantenimiento.

¿Por Qué se Necesitan las Sentencias Iterativas?

Las sentencias iterativas son esenciales en programación, donde los datos suelen presentarse en grandes conjuntos, y la repetición manual de tareas sería impráctica y propensa a errores.

Se vuelven especialmente útiles cuando se trabaja con estructuras de datos, como listas, tuplas y diccionarios, que almacenan múltiples elementos.



Ventajas de las Sentencias Iterativas

1. **Eficiencia**

   Permiten ejecutar código repetitivo sin necesidad de escribirlo múltiples veces

2. **Mantenimiento**

   Facilitan la lectura y el mantenimiento del código al reducir la redundancia

3. **Flexibilidad**

   Se adaptan a diferentes escenarios según las necesidades del programa

4. **Optimización**

   Permiten crear algoritmos complejos de manera simplificada

< >

Tipos de Sentencias Iterativas en Python

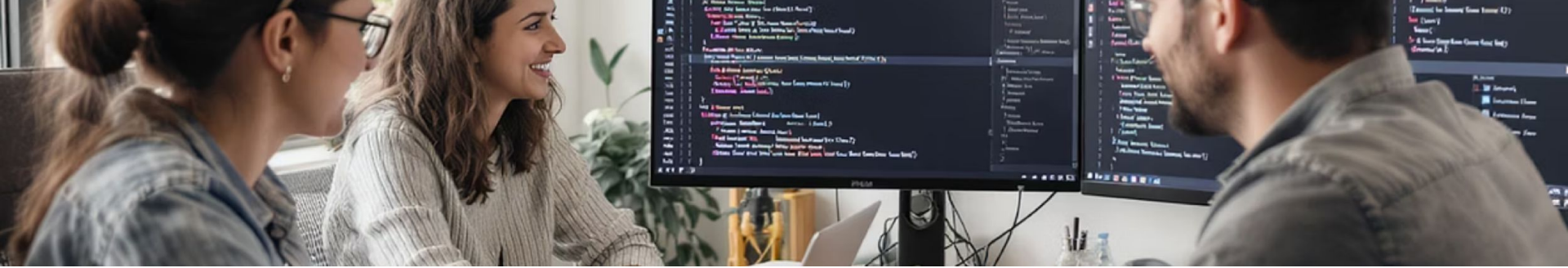
While

Permite que el código se repita mientras una condición sea verdadera, lo que es útil cuando no se conoce el número exacto de iteraciones necesarias.

For

Se usa cuando el número de iteraciones es conocido o depende de la cantidad de elementos en una colección.

Ambas estructuras ofrecen flexibilidad para adaptarse a diferentes escenarios.



Importancia en el Desarrollo de Software

La necesidad de sentencias iterativas se evidencia en la capacidad que tienen para optimizar el código y hacerlo más limpio y comprensible. Sin bucles, el código se volvería extenso y repetitivo, lo que dificulta su mantenimiento y aumenta las probabilidades de errores.

Los bucles permiten la creación de algoritmos complejos de manera simplificada, lo que facilita la resolución de problemas avanzados en programación.

< >

Aplicaciones de las Sentencias Iterativas

Análisis de Datos

Procesamiento de grandes conjuntos de datos para extraer información

Videojuegos

Creación de bucles de juego y sistemas de animación

Sistemas de Recomendación

Análisis de preferencias de usuarios para generar sugerencias

Las sentencias iterativas son la base para el manejo de algoritmos y estructuras de datos en Python.

La Sentencia While

La sentencia while ejecuta un bloque de código mientras una condición determinada sea verdadera. Esto la convierte en una herramienta valiosa cuando se necesita una repetición basada en una condición que puede cambiar durante la ejecución del programa.

< >

Sintaxis de While

```
# Inicialización de la variable de control
contador = 1

# Mientras la condición sea True, se ejecuta el bloque
while contador <= 5:
    print("Contador:", contador)  # Bloque de código a ejecutar
    contador += 1                 # Actualización de la variable de control
```

La sintaxis de while es simple, pero poderosa, y permite un control detallado sobre el flujo de iteración.

Ejemplo de While: *Contar hasta 5*

```
contador = 1while contador <= 5:
print("Número:", contador)    contador += 1
```

En este código, el bucle while se ejecuta hasta que contador alcanza el valor de 6, momento en el cual la condición contador <= 5 deja de ser verdadera y el bucle termina.

Este tipo de bucle es útil en situaciones donde el número de iteraciones depende de una condición específica que puede no estar relacionada con una colección de elementos.

< >

Bucles Infinitos

Una característica importante de while es que puede dar lugar a bucles infinitos si no se modifica la variable de control adecuadamente.

En el ejemplo anterior, si olvidáramos contador += 1, el bucle seguiría ejecutándose indefinidamente.

Por esta razón, es fundamental asegurarse de que la condición de salida se cumpla para evitar este tipo de errores.

alkemy

Usos Comunes de While

1 Interacción con el Usuario

Esperar a que un usuario ingrese un valor válido antes de continuar

2 Monitoreo en Tiempo Real

Verificar continuamente el estado de una conexión o proceso

3 Procesamiento Condicional

Ejecutar código hasta que se cumpla una condición específica

El bucle while es comúnmente utilizado en situaciones donde la cantidad de repeticiones depende de la interacción del usuario o de datos en tiempo real.

La Sentencia For

La sentencia for en Python permite iterar a través de elementos en una colección, como una lista, tupla o diccionario, y es particularmente útil cuando el número de iteraciones es conocido de antemano.

A diferencia de while, el bucle for se centra en recorrer cada elemento de una colección, asignando temporalmente cada uno a una variable, y ejecutando el bloque de código asociado.

< >

Sintaxis de For

```
for elemento in colección:    # Bloque de código a ejecutar    # para cada elemento
```

Esta estructura permite recorrer automáticamente todos los elementos de una colección sin necesidad de gestionar manualmente una variable de control.

Ejemplo de For: Iterando una Lista

```
numeros = [1, 2, 3, 4, 5]
for numero in numeros:
    print("Número:", numero)
```

En este caso, for asigna cada elemento de la lista numeros a la variable numero en cada iteración y luego imprime su valor.

Este bucle es útil para realizar acciones específicas sobre cada elemento en una colección de datos sin tener que gestionar manualmente una variable de control, como ocurre en while.

Ventajas de For sobre While

1 Prevención de Bucles Infinitos

El número de iteraciones está determinado por la cantidad de elementos en la colección

2 Simplicidad

No requiere gestionar manualmente variables de control

3 Legibilidad

El código es más claro y expresa mejor la intención de iterar sobre una colección

Esto lo convierte en una opción más segura cuando se trabaja con estructuras de datos o cuando se conoce la cantidad exacta de veces que se desea ejecutar el bucle.

< >

For con Range

La estructura for también permite utilizar la función range para crear un conjunto de valores a iterar, lo que es especialmente útil cuando se desea realizar un número controlado de repeticiones.

```
for i in range(5):    print(i)  # Imprime 0, 1, 2, 3, 4
```

Además, Python permite combinar for con otras funciones y métodos para realizar operaciones más complejas, como filtrar, mapear o transformar datos durante la iteración.

< >

Iterando Listas de Elementos

Las listas son una estructura de datos común en Python y suelen contener elementos del mismo tipo. Al iterar listas, el bucle for permite acceder a cada elemento de manera sencilla y realizar operaciones sobre ellos.

```
frutas = ["manzana", "banana", "cereza"]for fruta in frutas:    print(fruta)
```

Este código imprime cada fruta de la lista frutas, lo que facilita la manipulación de cada elemento sin necesidad de conocer la longitud de la lista.

< >

Operaciones con Listas

Este tipo de iteración es útil para operaciones simples, como imprimir valores, o para tareas más complejas, como modificar los elementos o filtrar aquellos que cumplen ciertas condiciones.

Otra técnica común es modificar los elementos de una lista dentro del bucle.

```
frutas = ["manzana", "banana",
"cereza"]frutas_mayusculas = [fruta.upper()
for fruta in
frutas]print(frutas_mayusculas)# ["MANZANA",
"BANANA", "CEREZA"]
```

Comprensión de Listas

Este ejemplo utiliza una **comprensión de listas** (list comprehension) para convertir cada elemento en mayúsculas y almacenarlos en una nueva lista.

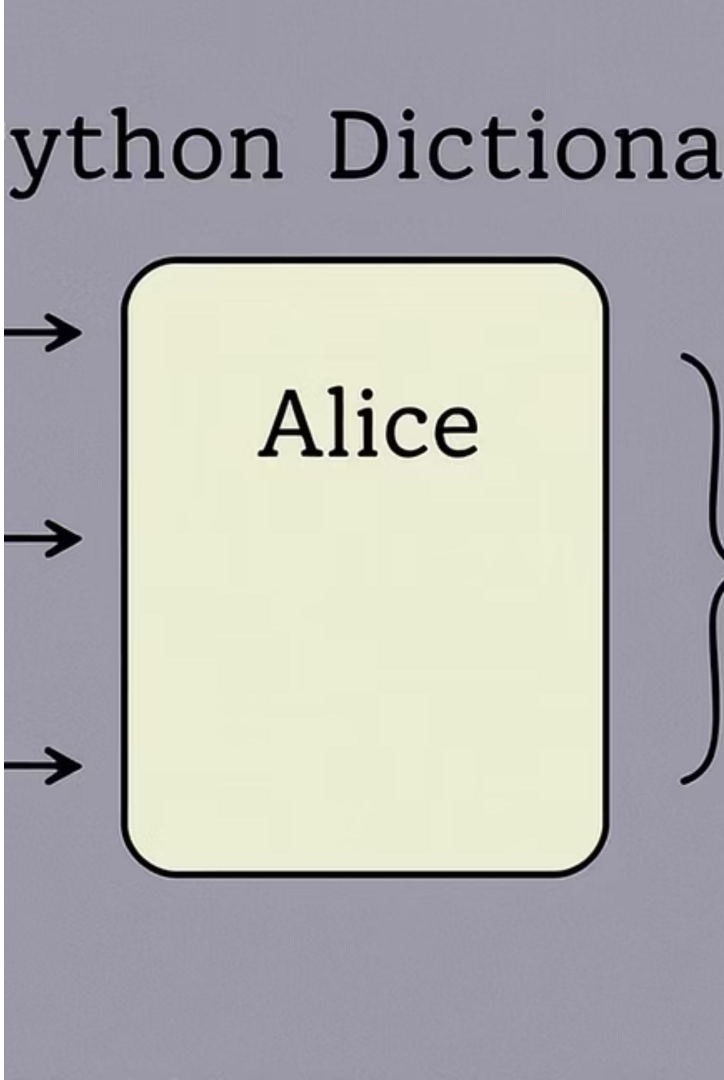
```
# Sintaxis generalnueva_lista = [expresión for elemento in iterable if condición]
```

Esta técnica es eficiente y evita la necesidad de crear un bucle for explícito, lo que reduce la cantidad de líneas de código.

Iterando Diccionarios de Elementos

Los diccionarios en Python son estructuras de datos que almacenan pares de clave-valor, permitiendo asociar un valor específico a una clave única.

A diferencia de las listas, donde los elementos se acceden mediante un índice, en los diccionarios se utilizan las claves para acceder a los valores.



< >

Estructura de los Diccionarios

```
edades = {    "Ana": 25,    "Carlos": 30,    "María": 22}
```

Esto permite crear colecciones de datos más complejas, donde cada valor puede tener un identificador único.

Para iterar sobre un diccionario, el bucle for permite acceder tanto a las claves como a los valores de manera eficiente.

Iterando con items()

Para iterar en un diccionario, se utiliza el método items(), que devuelve cada par clave-valor como una tupla.

Esto permite trabajar con ambas partes del par en cada iteración.

```
edades = {"Ana": 25, "Carlos": 30, "María": 22}for nombre, edad in edades.items():
print(f"{nombre} tiene {edad} años")
```

< >

Iterando Claves y Valores

En cada iteración, la clave se asigna a nombre y el valor a edad, permitiendo que se acceda y se manipule cada elemento del diccionario con facilidad.

Esta estructura es útil en situaciones donde se necesita procesar o mostrar información en función de claves específicas.

```
# Solo claves
for nombre in edades.keys():
    print("Nombre:", nombre)

# Solo valores
for edad in edades.values():
    print("Edad:", edad)
```

< >

Métodos para Iterar Diccionarios

1 items()

Devuelve pares de clave-valor como tuplas

```
for k, v in
dict.items(): ...
```

2 keys()

Devuelve solo las claves del diccionario

```
for k in dict.keys():
...
```

3 values()

Devuelve solo los valores del diccionario

```
for v in dict.values():
...
```

< >

Modificando Diccionarios

Es importante mencionar que, al modificar un diccionario dentro de un bucle, es recomendable crear una copia del diccionario o realizar cambios en una estructura separada, ya que modificar el diccionario directamente puede causar errores de ejecución.

```
# Método seguro usando comprensión de diccionarioedades = {"Ana": 25, "Carlos": 30, "María": 22}edades_en_meses = {nombre: edad * 12 for nombre, edad in edades.items()}
```

Un método seguro es almacenar los resultados en un nuevo diccionario, utilizando una comprensión de diccionario.

Live Coding

¿En qué consistirá la Demo?

Vamos a simular un sistema que recorre datos de usuarios utilizando bucles. Se mostrará cómo controlar repeticiones, recorrer listas y diccionarios, y aplicar condiciones dentro del bucle.

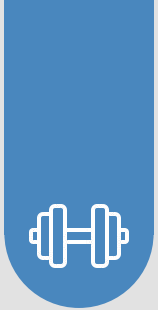
1. *Usar un while para repetir hasta que se ingrese una opción válida*
2. *Crear una lista de productos y recorrerla con for*
3. *Generar una secuencia numérica con range()*
5. *Usar for con items() para mostrar contenido de un diccionario*
6. *Construir una nueva lista con list comprehension*
7. *Filtrar elementos de un diccionario con comprensión de diccionario*

Tiempo: 25 Minutos

Momento:

Time-out!

⏰5 -10 min.



Ejercicio N° 1

Recorriendo el inventario

Recorriendo el inventario

Contexto: 🙌

Un almacén necesita revisar su inventario. Desean recorrer una lista de productos con su stock actual, verificar cuáles tienen stock bajo, y generar una lista de productos para reponer automáticamente.

Consigna: ✍️

Codificar una rutina utilizando sentencias iterativas (for, while) para analizar un inventario y generar alertas sobre stock bajo.

Tiempo 🕐: *35 Minutos*

Recorriendo el inventario

Paso a paso: ⚙️

1. Crear una lista de diccionarios, donde cada producto tenga nombre, stock y precio.
2. Usar un for para recorrer los productos:
    a. Imprimir solo los productos cuyo stock sea menor a 5
3. Usar list comprehension para crear una nueva lista llamada reposición con los nombres de productos a reponer.
4. Imprimir el total de productos en bajo stock y su valor total.
5. Usar un while que simule el ingreso de productos al sistema hasta que el usuario escriba "salir".

¿Alguna consulta?

alkemy

Resumen

¿Qué logramos en esta clase?

- ✓ ***Comprendimos el uso de bucles while y for***
- ✓ ***Aplicamos range() y estructuras iterables para controlar repeticiones***
- ✓ ***Iteramos listas y diccionarios de forma efectiva***
- ✓ ***Usamos comprensiones para filtrar y transformar colecciones***
- ✓ ***Creamos una rutina realista que automatiza tareas repetitivas***

¡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