



[LCD] MÓDULO 2

Bienvenido al **Módulo 2** del curso **Lenguaje de Ciencia de Datos I.**

INTRODUCCIÓN

Estimado estudiante:

En el presente módulo aprenderás a trabajar con tipos de datos avanzados en Python, como cadenas de texto y colecciones (listas, tuplas, conjuntos y diccionarios), además de explorar las estructuras de control de flujo, incluyendo condicionales y bucles. Estas herramientas les permitirán crear programas dinámicos capaces de procesar información compleja, tomar decisiones y automatizar tareas repetitivas, lo cual constituye la base de la programación aplicada en ciencia de datos.

Te deseamos una excelente experiencia de aprendizaje en este curso.

¡Muchos éxitos!

 TEMA 2: TIPOS DE DATOS AVANZADOS Y CONTROL DE FLUJO

 CIERRE

TEMA 2: TIPOS DE DATOS AVANZADOS Y CONTROL DE FLUJO

TIPOS DE DATOS AVANZADOS Y CONTROL DE FLUJO

Llegó el momento para que te conviertas en el protagonista de tu aprendizaje.

- Descubre y analiza la información que se presenta en la lección interactiva para transformarla en nuevo conocimiento.

Cuando se dominan las bases de Python —variables, tipos primitivos e interacción básica con el usuario—, el siguiente paso es construir programas más sofisticados. En este tema abordaremos los tipos de datos avanzados (cadenas y colecciones) y las estructuras de control de flujo (condicionales y bucles). Estas herramientas permiten que un programa no sea lineal, sino dinámico, capaz de tomar decisiones, procesar información compleja y automatizar tareas repetitivas, lo que constituye el corazón de la programación aplicada.

Cadenas de Texto (str)

En Python, las cadenas de texto (strings) son secuencias ordenadas de caracteres que pueden incluir letras, números, símbolos o espacios. Se utilizan ampliamente para procesar

información alfanumérica: desde un nombre o palabra, hasta párrafos completos.

En Python, las cadenas de texto (strings) son secuencias ordenadas de caracteres que pueden incluir letras, números, símbolos o espacios. Se utilizan ampliamente para procesar información alfanumérica: desde un nombre o palabra, hasta párrafos completos.

Operación	Descripción	Ejemplo	Resultado
Indexación	Acceder a un carácter mediante su posición (el primer índice es 0)	mensaje = "Python" mensaje[0]	'P'
Rebanado (slicing)	Extraer un segmento de la cadena indicando rango de índices	mensaje[0:3]	'Pyt'
Concatenación	Unir dos o más cadenas	"Hola" + "Python"	'HolaPython'
Repetición	Repetir varias veces el contenido de la cadena	"ha" * 3	'hahaha'
Métodos	Aplicar funciones integradas para	"hola".upper()	'HOLA'

transformar o
manipular texto

Métodos útiles de cadenas

En Python, las cadenas de texto no solo permiten almacenar información alfanumérica, sino también transformarla y manipularla de múltiples maneras. Para ello, el lenguaje ofrece una serie de métodos que facilitan tareas comunes como cambiar mayúsculas a minúsculas, eliminar espacios innecesarios, dividir el texto en fragmentos, reemplazar caracteres o buscar patrones específicos.

Métodos de manipulación de cadenas

.find()

Encuentra la posición de un subtexto.



.lower()

Convierte una cadena a minúscula.



.replace()

Reemplaza caracteres en una cadena.



.strip()

Elimina espacios en blanco iniciales y finales.



.split()

Divide una cadena en una lista.



```
frase = " Python es poderoso "
print(frase.strip())      # "Python es poderoso"
print(frase.split())      # ["Python", "es", "poderoso"]
print(frase.replace("o","0")) # " Pyth0n es p0der0s0 "
```



Las cadenas en Python son inmutables; cualquier cambio genera una nueva cadena.

CONTINUAR

Estructuras de Datos de Colección

Las colecciones permiten agrupar y manipular conjuntos de elementos. Cada estructura tiene propiedades específicas:

Tipo	Mutabilidad	Orden	Duplicados	Ejemplo	Uso principal
Lista (list)	Sí	Sí	Sí	["rojo","verde"]	Conjuntos dinámicos, modificables
Tupla (tuple)	No	Sí	Sí	(10,20)	Datos fijos, coordenadas
Conjunto (set)	Sí	No	No	{1,2,3}	Eliminar duplicados, operaciones matemáticas
Diccionario (dict)	Sí	No	Clave única	{"nombre":"Ana","edad":25}	Mapeo clave-valor rápido

Operaciones comunes con colecciones

A continuación te mostramos ejemplos de **operaciones básicas con colecciones en Python**, específicamente con listas. Estas operaciones son esenciales para la manipulación de datos, ya que permiten trabajar con grupos de elementos de manera ordenada y dinámica:

Operaciones comunes con colecciones



Agregar elementos

Usando `append()` se pueden incluir nuevos valores al final de la lista.



Acceder a elementos

Mediante índices (empezando desde 0) se obtiene un valor específico dentro de la colección



Calcular la longitud

Con `len()` se determina cuántos elementos contiene la lista.



Verificar pertenencia

Con la expresión “`elemento`” in `lista` se comprueba si un valor está presente.

Estas operaciones resultan especialmente útiles en programación porque facilitan la organización, el acceso y la verificación de datos, lo que es clave para estructurar algoritmos eficientes en proyectos de análisis, clasificación o almacenamiento de información.



```
colores = ["rojo","verde","azul"]
colores.append("amarillo")    # Agregar elemento
print(colores[0])            # Acceder a elemento
print(len(colores))          # Longitud de la colección
print("verde" in colores)    # Verificar pertenencia
```

CONTINUAR

Estructuras de Selección

Las estructuras condicionales constituyen la base de la toma de decisiones en programación. Su función principal es evaluar condiciones lógicas y, en función de su resultado, ejecutar distintos bloques de código. De este modo, permiten que el flujo del programa no sea lineal, sino dinámico y adaptable a diferentes escenarios. En Python, estas estructuras se implementan con las sentencias **if**, **elif** y **else**, que posibilitan crear desde decisiones simples hasta condiciones múltiples y anidadas, fundamentales para resolver problemas complejos y diseñar algoritmos inteligentes.

Forma	Ejemplo
if	if edad < 18: print("Menor de edad")
if-else	if edad >= 18: print("Adulto")

if-elif-else

```
else:  
    print("Menor de edad")  
  
if edad < 18:  
    print("Menor de edad")  
elif edad < 65:  
    print("Adulto")  
else:  
    print("Adulto mayor")
```

Los condicionales se pueden anidar y combinar con operadores lógicos (and, or, not) para condiciones más complejas.

¿Cómo funcionan las estructuras condicionales if, elif y else en Python?

Condition is True

```
number = 10  
if number > 0:  
    # code  
  
# code after if
```

Condition is False

```
number = -5  
if number > 0:  
    # code  
  
# code after if
```

La imagen ilustra cómo funcionan las estructuras condicionales **if**, **elif** y **else** en Python. Se presentan tres casos:

1. **Primera condición verdadera:** cuando number = 5, se cumple if number > 0, por lo que se ejecuta el bloque correspondiente al if.
2. **Segunda condición verdadera:** cuando number = -5, no se cumple el if, pero sí el elif number < 0, ejecutándose este bloque.
3. **Todas las condiciones falsas:** cuando number = 0, ni el if ni el elif se cumplen, por lo que se ejecuta el bloque del else.

Esto muestra cómo Python evalúa las condiciones en orden y ejecuta solo el bloque asociado a la primera condición verdadera, o al else si ninguna se cumple.

Uso de una condición simple con if

Condition is True

```
number = 10
if number > 0:
    # code
else:
    # code
# code after if
```

Condition is False

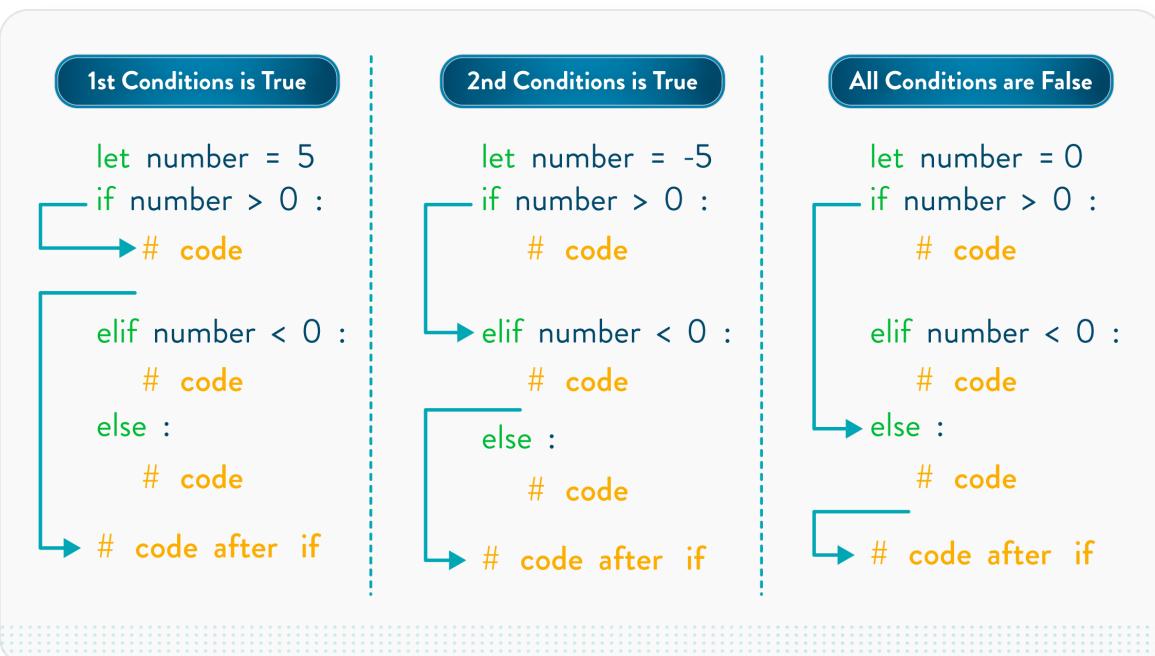
```
number = -5
if number > 0:
    # code
else:
    # code
# code after if
```

Se comparan dos situaciones:

1. **Condición verdadera:** si `number = 10`, la expresión `if number > 0` es cierta, por lo que se ejecuta el bloque de código dentro del `if`, seguido por el código posterior.
2. **Condición falsa:** si `number = -5`, la condición `number > 0` no se cumple, por lo que el bloque dentro del `if` se omite y solo se ejecuta el código que está después.

Este ejemplo evidencia la diferencia entre ejecutar código cuando una condición es cierta y omitirlo cuando no lo es, manteniendo la secuencia del programa.

¿Cómo funcionan las estructuras condicionales `if`, `elif` y `else` en programación?



1. Primera condición verdadera (number = 5):

- Se evalúa if number > 0. Como la condición es cierta, se ejecuta el bloque de código dentro del if.
- Los demás bloques (elif y else) no se ejecutan.
- Al final, se continúa con el código posterior al condicional.

2. Segunda condición verdadera (number = -5):

- La condición del if no se cumple porque el número no es mayor a 0.
- Se evalúa la condición del elif number < 0, que sí es cierta, por lo que se ejecuta su bloque de código.
- El bloque else se ignora.
- El programa sigue con el código posterior.

3. Todas las condiciones falsas (number = 0):

- El if number > 0 no se cumple.
- El elif number < 0 tampoco se cumple.
- Por lo tanto, se ejecuta el bloque del else.
- Finalmente, se continúa con el código después del condicional.

👉 En resumen, el flujo del programa avanza evaluando las condiciones en orden: **si una es verdadera, ejecuta su bloque y se detiene; si ninguna lo es, se ejecuta el else.**

CONTINUAR

Estructuras de Repetición

Los bucles son herramientas fundamentales en programación porque permiten automatizar procesos y recorrer colecciones de datos sin necesidad de escribir instrucciones repetitivas. Con ellos es posible ejecutar un mismo bloque de código múltiples veces, bajo condiciones específicas, lo que incrementa la eficiencia y reduce errores.

Bucle	Uso	Ejemplo	Resultado esperado
for	Recorrer secuencias o rangos	for i in range(3): print(i)	0,1,2
while	Repetir mientras una condición sea verdadera	contador=0 while contador<3: print(contador) contador+=1	0,1,2
break	Detener el bucle por completo	for i in range(5): if i == 3: break	0, 1, 2

		print(i)	
continue	Saltar a la siguiente iteración	<pre>for i in range(5): if i == 2: continue print(i)</pre>	0,1,3,4
pass	No hacer nada (placeholder)	<pre>for i in range(3): if i == 1: pass print(i)</pre>	0,1,2

Palabras clave útiles: **break** (detiene el ciclo), **continue** (salta a la siguiente iteración).

Uso de break en bucles (for y while)

Ahora te mostraremos cómo funciona la instrucción **break** dentro de bucles en Python:

- En un **bucle for**, se recorren los elementos de una secuencia. Si se cumple una condición dentro del ciclo, la instrucción **break** detiene inmediatamente la ejecución del bucle, sin importar si quedaban elementos por recorrer.
- En un **bucle while**, ocurre lo mismo: mientras la condición principal se cumple, el bucle se ejecuta; pero si en algún momento la condición interna activa el **break**, el ciclo se

interrumpe por completo.

👉 En resumen, break se usa para **detener el bucle de manera abrupta**.

```
for val in sequence:
```

```
    # code
```

```
    if condition:
```

```
        break
```

```
        # code
```

```
while condition:
```

```
    # code
```

```
    if condition:
```

```
        break
```

```
        # code
```

Uso de break en bucles (for y while)

Aquí te explicamos el funcionamiento de la instrucción **continue**:

- En un **bucle for**, si se cumple la condición, el continue **salta directamente a la siguiente iteración**, omitiendo el resto del código dentro del ciclo en curso.

- En un **bucle while**, funciona de manera equivalente: cuando se cumple la condición, el `continue` evita ejecutar el código posterior en esa vuelta y regresa al inicio del bucle para evaluar de nuevo la condición.

👉 En resumen, continue **no detiene el ciclo**, sino que **omite la iteración actual y continúa con la siguiente**.

```
→ for val in sequence:  
    # code  
    if condition:  
        continue  
    # code  
-----  
→ while condition:  
    # code  
    if condition:  
        continue  
    # code
```

CONTINUAR

El dominio de los tipos de datos avanzados y las estructuras de control en Python constituye un paso esencial para pasar de simples programas a soluciones dinámicas y robustas. Al comprender cómo

manipular cadenas y colecciones, y cómo aplicar condicionales y bucles, los estudiantes están preparados para resolver problemas reales con mayor eficiencia.



[CONTINUAR](#)

Para finalizar el tema, recordemos algunas ideas claves.

1

Tipos de datos avanzados como cadenas y colecciones permiten organizar, transformar y acceder a la información de forma eficiente.

2

Estructuras condicionales (if, elif, else) hacen posible que los programas tomen decisiones dinámicas en función de condiciones lógicas.

3

Bucles (for, while) automatizan procesos repetitivos y permiten recorrer secuencias de datos, optimizando el trabajo con grandes volúmenes de información.

4

Sentencias de control como break, continue y pass otorgan flexibilidad adicional en la gestión de ciclos, mejorando el control del flujo del programa.

CONTINUAR

CIERRE



¡FELICITACIONES!

Hemos llegado al final del tema.

Continúa tu aprendizaje en el siguiente
módulo.