

«Talento Tech»

Iniciación a la

Programación con Python

Clase 04



Clase N° 4 | Condicionales II

Temario:

- Estructuras condicionales avanzadas: `elif` y `match`.
 - Manipulación de cadenas: acceso a caracteres, concatenación, longitud.
 - Métodos de cadenas: `.lower()`, `.upper()`, `.title()`, etc.
 - Formateo de cadenas con f-Strings.
 - Ruta de avance.
-

Objetivos de la clase

El principal objetivo de esta clase es ampliar el manejo de estructuras condicionales en Python, introduciendo herramientas avanzadas como **`elif`** y **`match`**, que permiten gestionar múltiples escenarios en un mismo programa de forma más clara y eficiente. Estas estructuras son fundamentales para diseñar soluciones más completas y adaptativas.

Además, se busca que los estudiantes comprendan cómo manipular cadenas de texto, accediendo a sus caracteres, concatenándolas, calculando su longitud y transformándolas mediante métodos como `.lower()` y `.upper()`. Estas habilidades son esenciales para procesar y presentar información de manera efectiva.

Por último, se trabajará en el formateo de cadenas utilizando f-Strings, una técnica que simplifica la generación de salidas dinámicas y profesionales. Al finalizar la clase, estarán en condiciones de combinar estas herramientas para construir programas que gestionen datos textuales de manera avanzada, respondiendo a necesidades reales de las y los usuarios.

TechLab: Jornada 4 ☀️



En lo que ya se está convirtiendo en una costumbre, ni bien ingresás a la sede de **TalentLab**, te encontrás con Mariana. Café en mano, te felicita por tus avances, y te comenta que:



“Nuestro cliente nos pide que el programa formatee correctamente los textos ingresados y que clasifique a las y los clientes por rango etario (niña o niño, adolescente, adulto o adulta) basándose en su edad.”

Claramente, es algo que no podés hacer con tus conocimientos actuales. Afortunadamente, Luis está ahí para ayudarte. Buscan un despacho libre, y comienzan a ver que necesitás saber para poder resolver este nuevo desafío.

Estructuras condicionales avanzadas: elif

Estructuras condicionales avanzadas: elif

En la clase anterior aprendimos a usar las estructuras condicionales **if** y **if...else**, que nos permiten tomar decisiones básicas dentro de un programa. Sin embargo, en situaciones donde necesitamos manejar múltiples condiciones, escribir varios bloques **if** puede volver nuestro código repetitivo y difícil de seguir. Ahí es donde entra en juego **elif**.



El bloque **elif** (abreviatura de "else if") nos permite manejar múltiples casos dentro de una misma estructura condicional, evitando redundancias y haciendo que nuestro código sea más eficiente y legible.

Python evalúa las condiciones de manera secuencial: si una condición **if** es verdadera, ejecuta ese bloque y se saltea el resto; si no, pasa al siguiente **elif** y así sucesivamente, hasta llegar al **else** (si está presente).

Sintaxis básica de elif

```
edad = 25

if edad < 13:
    print("Sos menor a trece años.")
elif edad < 18:
    print("Sos un o una adolescente.")
elif edad < 60:
    print("Sos una persona adulta.")
else:
    print("Sos una persona adulta mayor.")
```

En este ejemplo, el programa evalúa la edad del usuario o usuaria y muestra un mensaje correspondiente al rango etario. Observá cómo **elif** simplifica el manejo de múltiples condiciones, haciendo que el flujo del programa sea claro y directo.

Cuándo usar elif

Usamos **elif** cuando:

1. Hay múltiples condiciones mutuamente excluyentes (es decir, sólo una puede ser verdadera).

2. Queremos evitar escribir varios bloques **if** independientes que podrían complicar el seguimiento del código.

Ejemplo práctico: Clasificación de notas

Imaginá que estás programando un sistema para clasificar notas escolares. El programa debería asignar una categoría según el puntaje del estudiante:

```
nota = int(input("Ingresá la nota del o de la estudiante: "))

if nota >= 90:
    print("Excelente.")
elif nota >= 75:
    print("Muy bien.")
elif nota >= 60:
    print("Bien.")
elif nota >= 40:
    print("Suficiente.")
else:
    print("Insuficiente.")
```

Este ejemplo muestra cómo **elif** permite manejar fácilmente varias categorías sin repetir código innecesariamente.

Combinando elif con operadores lógicos

Podemos combinar **elif** con operadores lógicos para evaluar condiciones más complejas. Por ejemplo:

```
# Solicitar al usuario o usuaria que ingrese su edad
edad = int(input("Ingresá tu edad: ")) # Convertimos a entero

# Solicitar que informe su ingreso mensual y convertimos
# el dato volcado a un número entero

ingreso = int(input("Escribí tu ingreso mensual: "))

# Evaluar las condiciones para clasificar al o a la usuario o usuaria
```



```
if edad < 18:
    # Si la edad es menor a 18, es menor de edad
    print("Sos menor de edad.")
elif edad >= 18 and ingreso < 50000:
    # Si la edad es mayor o igual a 18 y el ingreso mensual
    # es menor a $50,000
    print("Sos mayor de edad, pero tenés ingresos bajos.")
elif edad >= 18 and ingreso >= 50000:
    # Si la edad es mayor o igual a 18 y el ingreso mensual
    # es igual o mayor a $50,000
    print("Sos mayor de edad y tenés ingresos altos.")
else:
    # En caso de que no se cumpla ninguna condición anterior
    print("Datos no válidos.")
```

Este programa clasifica a las personas según su edad y nivel de ingresos, mostrando cómo **elif** se combina con operadores para resolver problemas más realistas.



El uso de **elif** simplifica el manejo de múltiples escenarios en tus programas, haciéndolos más fáciles de entender y mantener.

Estructura condicional avanzada: match

Estructura condicional avanzada: match

La estructura **match** fue introducida en la versión 3.10 de Python, como una alternativa más clara y poderosa al uso de múltiples bloques **if...elif...else**, especialmente cuando queremos comparar un valor específico con varias opciones posibles. Es similar a la estructura **switch** de otros lenguajes de programación y se utiliza para simplificar el manejo de casos múltiples.

¿Por qué usar match?

Aunque **if...elif...else** es muy versátil, en casos donde se necesita comparar un valor con múltiples posibilidades, el código puede volverse más extenso y difícil de leer. **Match** aborda esta limitación al permitir evaluar un valor contra diferentes patrones, ofreciendo:

- **Claridad y legibilidad:** Su sintaxis es más compacta y clara para casos múltiples.
- **Flexibilidad:** Puede trabajar con patrones avanzados, no solo con valores literales.

- **Mantenibilidad:** Simplifica el flujo del programa, especialmente cuando hay muchas condiciones.

Comparación entre match y elif:

Característica	match	elif
Propósito	Comparar un valor contra múltiples patrones.	Resolver condiciones más generales.
Legibilidad	Más claro para múltiples casos específicos.	Puede volverse extenso y repetitivo.
Flexibilidad	Permite patrones complejos.	Mejor para condiciones generales o combinadas.
Introducción	Python 3.10 y posteriores.	Disponible en todas las versiones.

La sintaxis de match

La estructura match en Python se utiliza para comparar el valor de una variable con diferentes patrones, ejecutando el bloque de código asociado al primer patrón que coincida. Esta herramienta es especialmente útil cuando tenés que manejar múltiples opciones de forma clara y organizada.

La sintaxis general de match es la siguiente:

```
match variable:
    case valor1:
        # Código si variable coincide con valor1
    case valor2:
        # Código si variable coincide con valor2
    case _:
        # Código si no coincide ningún caso (opcional)
```

- **variable:** Es el valor que queremos comparar. Puede ser de cualquier tipo, como números, cadenas, tuplas, etc.
- **case valor1:** Define un caso específico. Si **variable** coincide con este valor, se ejecuta el bloque de código asociado.

- **_ (guión bajo)**: Representa el caso por defecto, que se ejecuta si no hay coincidencia con ninguno de los casos anteriores. Es equivalente al **else** en un bloque condicional.

Python evalúa los casos en orden, deteniéndose en el primero que coincide. Esto hace que el flujo del programa sea eficiente y fácil de seguir. Pero veamos un ejemplo:

```
fruta = input("Ingresa una fruta: ")

match fruta:
    case "manzana":
        print("Es una fruta roja o verde.")
    case "banana":
        print("Es una fruta amarilla.")
    case "naranja":
        print("Es una fruta anaranjada.")
    case _:
        print("No tengo información sobre esa fruta.")
```

En este ejemplo:

1. La variable **fruta** es evaluada por **match**.
2. Cada **case** especifica un valor posible para **fruta**.
3. Si no hay coincidencia, se ejecuta el bloque del caso **_**.

Ahora, veamos un ejemplo más, para comparar **elif** con **match**. Supongamos que te piden escribir un programa que reciba el número de un día de la semana (donde 1 corresponde a Lunes, 2 a Martes, 3 a Miércoles, y así sucesivamente) e imprima el nombre del día correspondiente. Si el número ingresado no corresponde a un día válido, el programa debe mostrar el mensaje: "Día no válido". Esto puede resolverse tanto usando **elif**, como usando **match**:

Con if...elif...else:

```
dia = 3

if dia == 1:
    print("Lunes")
elif dia == 2:
    print("Martes")
elif dia == 3:
    print("Miércoles")
```









```
else:
    print("Día no válido")
```

Con match:

```
dia = 3

match dia:
    case 1:
        print("Lunes")
    case 2:
        print("Martes")
    case 3:
        print("Miércoles")
    case _:
        print("Día no válido")
```

Usando **match** el código es más claro y evita la repetición de la variable en cada comparación. Ahora, ya podemos resolver uno de los pedidos que nos ha hecho Luis:

 Recargar  Redactar  Responder  Reenviar  Eliminar  Favorito

Buscar correo

De: Luis – Desarrollador Senior

Asunto: Clasificador de paquetes

¡Hola!

A partir de lo que has aprendido, intenta escribir un programa que clasifique paquetes según su peso. El programa debe solicitar que se ingrese el peso del paquete (en kilogramos) y, en base al valor ingresado, clasificar el paquete en una de las siguientes categorías:

1. **Paquete pequeño:** Para paquetes que pesen hasta 5 kg inclusive.
2. **Paquete mediano:** Para paquetes cuyo peso sea mayor a 5 kg y hasta 20 kg inclusive.
3. **Paquete grande:** Para paquetes que pesen más de 20 kg.
4. **Peso no válido:** En caso de que el peso ingresado sea negativo o no corresponda a un valor numérico válido.

Luis Rodríguez
Desarrollador Senior
DataDive Solutions

Este código resuelve lo que te han solicitado:

```
peso = float(input("Ingresá el peso del paquete (en kg): "))

match peso:
    case p if p <= 5:
        print("Paquete pequeño.")
    case p if 5 < p <= 20:
        print("Paquete mediano.")
    case p if p > 20:
        print("Paquete grande.")
    case _:
        print("Peso no válido.")
```

Queda claro que **match** es una herramienta poderosa que complementa y, en algunos casos, reemplaza al uso de múltiples bloques **elif**. Ofrece una sintaxis más limpia y legible, especialmente útil cuando se necesita manejar múltiples casos específicos.



No olvides que para condiciones más generales o complejas, **if...elif...else** sigue siendo una opción válida y flexible.

Manipulación de cadenas en Python

Manipulación de cadenas en Python

Ya sabés que en Python una cadena de caracteres (o simplemente cadena) es una secuencia encerrada entre comillas simples (') o dobles ("). Además, son uno de los tipos de datos más utilizados porque nos permiten trabajar con texto, desde nombres de personas hasta mensajes más complejos. Pero para poder hacerlo correctamente, necesitamos saber cómo manipularlas.

Concatenación de cadenas

La **concatenación** es el proceso de unir dos o más cadenas de texto para formar una nueva. En Python, podés usar el operador **+** para lograrlo de manera sencilla.

Cuando trabajás con datos textuales, la concatenación es una herramienta clave. Por ejemplo, podés combinar el nombre y el apellido de una persona para mostrar su nombre completo en pantalla.

```
nombre = "María"
apellido = "González"

# Se agrega un espacio entre las cadenas
nombre_completo = nombre + " " + apellido

print(nombre_completo) # Salida: María González
```

En este ejemplo:

1. **nombre** y **apellido** son cadenas separadas que contienen diferentes partes de un dato.
2. Usamos el operador **+** para unirlos, incluyendo un espacio **" "** entre ellas para que el resultado sea legible.
3. La variable **nombre_completo** almacena la nueva cadena resultante.



Este tipo de operación es muy común cuando querés presentar información de manera clara, como generar un mensaje personalizado o mostrar datos de usuarios o usuarias.

Podés concatenar tantas cadenas como necesites en una sola línea. Por ejemplo:

```
saludo = "Hola"
nombre = "Lucía"
mensaje = saludo + ", " + nombre + ". ¿Cómo estás?"
print(mensaje) # Salida: Hola, Lucía. ¿Cómo estás?
```

En este caso combinamos tres cadenas: saludo, nombre y una cadena literal (el texto dentro de comillas). Usamos comas, espacios y signos de puntuación para hacer que el mensaje tenga sentido y sea amigable.

Si necesitás unir una cadena con otro tipo de dato, como un número o un valor booleano, primero tenés que convertir ese dato en texto usando la función **str()**. Esto es necesario porque, como sabés, Python no permite combinar distintos tipos de datos directamente.

```
edad = 30
mensaje = "Tenés " + str(edad) + " años."
print(mensaje) # Salida: Tenés 30 años
```

En este caso la variable **edad** es un número entero (tipo int), por lo que usamos **str(edad)** para convertirlo en texto. Luego, concatenamos la cadena resultante con otras cadenas para formar el mensaje final.

Longitud de una cadena de caracteres y la función len()

Longitud de una cadena de caracteres y la función len()

La longitud de una cadena de caracteres es la cantidad total de caracteres que contiene. Esto incluye letras, números, espacios, símbolos y cualquier otro carácter que forme parte del texto. En Python, podés determinar la longitud de una cadena utilizando la función incorporada **len()**.



La función len() es muy útil para validar datos ingresados por la o el usuario, analizar cadenas o asegurarte de que cumplen con ciertos requisitos, como una cantidad mínima o máxima de caracteres.

La función len() toma como argumento una cadena y devuelve un número entero que representa la cantidad de caracteres en esa cadena.

```
mensaje = "Hola, mundo"
print(len(mensaje)) # Salida: 11
```

La cadena "Hola, mundo" tiene 11 caracteres en total, contando el espacio y la coma. Python incluye todos los caracteres visibles y no visibles (como los espacios) en la longitud.

Podés usar len() para asegurarte que se ingresa un dato válido. Por ejemplo, evitar que un campo como "nombre" quede vacío:

```
nombre = input("Ingresá tu nombre: ")

if len(nombre) == 0:
    print("El nombre no puede estar vacío.")
else:
    print(";Hola, " + nombre + "!")
```

Al no ingresarse ningún dato, aparece el mensaje: "El nombre no puede estar vacío." Si ingresa un nombre, el programa le saluda con un mensaje personalizado.

Algunas aplicaciones requieren que un dato, como una contraseña, tenga cierta cantidad de caracteres. Podés validar esto con `len()`.

```
contraseña = input("Ingresá tu contraseña: ")

if len(contraseña) < 8:
    print("La contraseña debe tener al menos 8 caracteres.")
else:
    print("Contraseña válida.")
```

En este caso, si la contraseña tiene menos de 8 caracteres, el programa muestra un mensaje avisando de esta situación. Si cumple con el requisito, el mensaje confirma que es válida.

Rompiendo las cadenas

Rompiendo las cadenas

Las cadenas en Python no son más que una secuencia ordenada de caracteres. Cada letra, número, espacio o símbolo ocupa una posición específica dentro de esa cadena, y podemos acceder a ellos de forma individual utilizando un índice. Esta característica es muy útil cuando necesitás trabajar con partes específicas de un texto, como extraer una letra, analizar un segmento o incluso modificar el formato de ciertas secciones.

En Python, el índice de una cadena comienza en cero. Esto significa que el primer carácter de la cadena está en la posición 0, el segundo en la posición 1, y así sucesivamente. Podés utilizar la notación con corchetes `[]` para acceder a un carácter específico. Por ejemplo, si

tenés la cadena `mensaje = "Hola"`, podés acceder al primer carácter escribiendo `mensaje[0]`, que en este caso te devolvería "H". Del mismo modo, si querés obtener el cuarto carácter, podés usar `mensaje[3]`, que devolvería "a".

```
mensaje = "Hola"

# Accedemos al primer carácter
print(mensaje[0]) # Salida: H

# Accedemos al cuarto carácter
print(mensaje[3]) # Salida: a
```

Una de las grandes ventajas de esta funcionalidad es que también podés usar índices negativos. Cuando usás un índice negativo, Python empieza a contar desde el final de la cadena. Por ejemplo, con la misma cadena "Hola", `mensaje[-1]` te daría el último carácter, "a", mientras que `mensaje[-2]` te devolvería "l". Mirá este ejemplo:

```
mensaje = "Hola"

# Usamos un índice negativo para obtener el último carácter
print(mensaje[-1]) # Salida: a

# Obtenemos el penúltimo carácter
print(mensaje[-2]) # Salida: l
```

Podés extraer una porción de la cadena utilizando **slicing** (rebanado). Esto se hace indicando un rango en los corchetes [`inicio:fin`]. El rango incluye el carácter en la posición de inicio, pero no el del fin. Si escribís `mensaje[0:2]`, vas a obtener "Ho", que incluye los caracteres en las posiciones 0 y 1, pero no el de la posición 2 (el rango es exclusivo en el límite superior). Si omitís el inicio o el final del rango, Python asume que querés ir desde el principio o hasta el final de la cadena, respectivamente. Por ejemplo, `mensaje[:2]` te devuelve "Ho", y `mensaje[1:]` te da "ola".

```
# Extraemos los dos primeros caracteres
print(mensaje[0:2]) # Salida: Ho

# Desde el segundo carácter hasta el final
```



```
print(mensaje[1:]) # Salida: ola

# Desde el inicio hasta el segundo carácter (sin incluirlo)
print(mensaje[:2]) # Salida: Ho
```

Esta herramienta podés utilizarla para validar texto, formatearlo o analizarlo con mucha precisión. Por ejemplo, si quisieras verificar si una cadena empieza con cierta letra, podrías hacer algo como `mensaje[0] == "H"`, y eso te devolvería `True` si el primer carácter es efectivamente "H".



Las cadenas son **inmutables**, lo que significa que no podés cambiar un carácter directamente asignándole un nuevo valor usando los índices. Si intentás algo como `mensaje[0] = "J"`, Python va a generar un error.

Métodos de cadenas en Python

Métodos de cadenas en Python

Las cadenas en Python no solo son una secuencia de caracteres, sino que también son objetos que vienen con una variedad de métodos incorporados. Los métodos son funciones asociadas a las cadenas que podés utilizar para realizar operaciones comunes, como cambiar el formato del texto, buscar contenido o realizar transformaciones específicas. Lo interesante de los métodos de cadenas es que no modifican la cadena original, sino que generan una nueva con los cambios aplicados.

Estos métodos son fáciles de usar y te permiten trabajar con texto de manera más eficiente. Por ejemplo, podés convertir una cadena a minúsculas, mayúsculas o formatearla automáticamente como un título. Cada método se invoca usando la notación de punto, como **`cadena.lower()`**, y muchos no requieren parámetros adicionales.

A continuación, vamos a ver algunos de los métodos más comunes que podés usar con las cadenas en Python, junto con ejemplos prácticos para que veas cómo aplicarlos.

Método	Ejemplo de uso	Explicación
<code>.lower()</code>	<pre>texto = "Hola Mundo" print(texto.lower()) # Salida: hola mundo</pre>	Convierte todos los caracteres de la cadena a minúsculas. Ideal para hacer

		comparaciones insensibles a mayúsculas y minúsculas.
.upper()	<pre>texto = "Hola Mundo" print(texto.upper()) # Salida: HOLA MUNDO</pre>	Convierte todos los caracteres de la cadena a mayúsculas. Útil para normalizar texto antes de guardarlo o procesarlo.
.title()	<pre>texto = "python es genial" print(texto.title()) # Salida: Python Es Genial</pre>	Capitaliza la primera letra de cada palabra en la cadena. Muy útil para formatear nombres o títulos.
.strip()	<pre>texto = " Hola Mundo " print(texto.strip()) # Salida: Hola Mundo</pre>	Elimina los espacios en blanco al principio y al final de la cadena. Perfecto para limpiar entradas de usuarios o datos importados.
.replace()	<pre>texto = "Hola Mundo" print(texto.replace("Mundo", "Python")) # Salida: Hola Python</pre>	Reemplaza una subcadena por otra dentro de la cadena. Muy útil para realizar modificaciones en texto.
.startswith()	<pre>texto = "Hola Mundo" print(texto.startswith("Hola")) # Salida: True</pre>	Devuelve True si la cadena comienza con la subcadena especificada; de lo contrario, devuelve False . Ideal para validaciones.
.endswith()	<pre>texto = "Hola Mundo" print(texto.endswith("Mundo")) # Salida: True</pre>	Devuelve True si la cadena termina con la subcadena especificada; de lo contrario, devuelve False .
.find()	<pre>texto = "Hola Mundo" print(texto.find("Mundo")) # Salida: 5</pre>	Devuelve la posición de la primera aparición de una subcadena dentro de la cadena, o -1 si no la encuentra.
.isdigit()	<pre>texto = "123" print(texto.isdigit()) # Salida: True</pre>	Devuelve un valor booleano True si todos los valores de la cadena de entrada son

		dígitos; de lo contrario, devuelve False
--	--	---

Acá tenés un ejemplo práctico que utiliza varios de los métodos mencionados, con comentarios detallados para explicar cada paso:

```
# Texto inicial
texto = "    Python es un LENGUAJE poderoso.    "

# Convertimos todo el texto a minúsculas
texto_min = texto.lower()
print("Texto en minúsculas:", texto_min)
# Salida: "    python es un lenguaje poderoso.    "

# Convertimos todo el texto a mayúsculas
texto_may = texto.upper()
print("Texto en mayúsculas:", texto_may)
# Salida: "    PYTHON ES UN LENGUAJE PODEROSO.    "

# Eliminamos los espacios en blanco al principio y al final
texto_sin_espacios = texto.strip()
print("Texto sin espacios:", texto_sin_espacios)
# Salida: "Python es un LENGUAJE poderoso."

# Reemplazamos la palabra "LENGUAJE" por "lenguaje"
texto_reemplazado = texto_sin_espacios.replace("LENGUAJE", "lenguaje")
print("Texto modificado:", texto_reemplazado)
# Salida: "Python es un lenguaje poderoso."

# Verificamos si el texto comienza con "Python"
comienza_con_python = texto_reemplazado.startswith("Python")
print("¿El texto comienza con 'Python'?", comienza_con_python)
# Salida: True

# Verificamos si el texto termina con un punto
termina_con_punto = texto_reemplazado.endswith(".")
print("¿El texto termina con un punto?", termina_con_punto) # Salida:
True
```

```
# Capitalizamos la primera letra de cada palabra
texto_titulo = texto_reemplazado.title()
print("Texto en formato título:", texto_titulo)  # Salida: "Python Es
Un Lenguaje Poderoso."
```

Veamos otro ejemplo práctico. El código siguiente solicita confirmar si está seguro o no, y utiliza algunos de los métodos explicados para procesar y evaluar la respuesta:

```
# Solicitamos al usuario que confirme si está seguro
respuesta = input("¿Estás seguro? (si/no): ")

# Convertimos la respuesta a minúsculas para evitar
# problemas con mayúsculas o minúsculas y quitamos
# espacios en blanco
respuesta = respuesta.strip().lower()

# Evaluamos la respuesta
if respuesta == "si":
    print("Confirmación recibida. Procedemos con la acción.")
elif respuesta == "no":
    print("Cancelaste la acción.")
else:
    print("No entendí tu respuesta. Por favor ingresá 'si' o 'no'.")
```

Usamos **.strip()** para eliminar espacios al principio o al final (en caso de que el usuario presione la barra espaciadora antes o después de escribir). También aplicamos **.lower()** para convertir todo a minúsculas, asegurándonos de que no importe si escribiste "Sí", "SÍ" o "si".

Formateo de cadenas con f-Strings

Formateo de cadenas con f-Strings

El formateo de cadenas con **f-Strings** es una de las maneras más eficientes y legibles de combinar texto con valores variables en Python. Introducidas en la versión 3.6, las f-Strings permiten integrar directamente expresiones y variables dentro de una cadena, utilizando llaves `{}` para delimitar el contenido dinámico. Esto hace que el código sea más claro y fácil de mantener en comparación con otras técnicas de formateo. Su utilidad es enorme, ya que

podés generar mensajes personalizados, reportes dinámicos o simplemente mostrar información formateada de manera profesional con un esfuerzo mínimo. Además, las f-Strings soportan expresiones complejas, lo que las convierte en una herramienta poderosa para la creación de cadenas dinámicas.

Las f-Strings permiten no solo incluir valores dinámicos en cadenas, sino también aplicar comandos y formatos para presentar la información de manera clara y profesional. Estas herramientas hacen que las f-Strings sean extremadamente versátiles para formatear números, fechas y texto. Veamos los principales comandos y formatos que podés usar, junto con algunos ejemplos:

1. Inclusión de variables

La forma básica de usar f-Strings es incluir variables directamente dentro de las llaves {}.

```
nombre = "María"
edad = 30
print(f"Hola, {nombre}. Tenés {edad} años.")
# Salida: Hola, María. Tenés 30 años.
```

Las variables se colocan dentro de las llaves y Python las reemplaza automáticamente por su valor.

2. Expresiones dentro de las llaves

Podés incluir cálculos y expresiones directamente dentro de las llaves.

```
a = 5
b = 3
print(f"La suma de {a} y {b} es {a + b}.")
# Salida: La suma de 5 y 3 es 8.
```

Esto te permite realizar operaciones sin necesidad de calcular el valor antes de la f-String.

3. Formateo de números

Con las f-Strings, podés aplicar formatos para ajustar cómo se muestran los números.

Decimales: Podés limitar la cantidad de decimales utilizando `:.nf`, donde `n` es el número de decimales deseados.

```
pi = 3.14159
print(f"El valor de  $\pi$  con 2 decimales es {pi:.2f}.")
```

```
# Salida: El valor de  $\pi$  con 2 decimales es 3.14.
```

Números grandes: Podés agregar separadores de miles con `:,.` (dos puntos seguidos del separador, que puede ser un punto o una coma):

```
numero = 1000000
print(f"El número es {numero:,}.")
# Salida: El número es 1,000,000.
```

Porcentajes: Usá `%` para formatear como porcentaje.

```
porcentaje = 0.85
print(f"El porcentaje es {porcentaje:.0%}.")
# Salida: El porcentaje es 85%.
```

4. Formateo de texto

Podés usar especificadores para ajustar cómo se muestra el texto.

Alineación: Usá `<`, `>` o `^` para alinear texto a la izquierda, derecha o centro, respectivamente.

```
texto = "Python"
print(f"{texto:<10}") # Salida: Python (alineado a la izquierda)
print(f"{texto:>10}") # Salida: Python (alineado a la derecha)
print(f"{texto:^10}") # Salida: Python (centrado)
```

Relleno: Podés agregar un carácter de relleno antes de los especificadores de alineación.

```
print(f"{texto:*<10}") # Salida: Python****
print(f"{texto:*>10}") # Salida: ****Python
print(f"{texto:*^10}") # Salida: **Python**
```

5. Escapando llaves

Si necesitás incluir llaves `{}` como texto y no como parte de una f-String, simplemente duplicalas.

```
print(f"Este es un ejemplo de llaves: {{y esta es una llave dentro de la cadena}}.")
```



```
# Salida: Este es un ejemplo de llaves: {y esta es una llave dentro de la cadena}.
```

Las f-Strings no sólo son prácticas para incluir variables, sino que también te permiten formatear números, texto y otros tipos de datos con un control increíble. Ya sea que estés creando reportes, mensajes personalizados o (como veremos más adelante) visualizando datos complejos, las f-Strings te ofrecen las herramientas necesarias para hacerlo de manera clara y simple.

Ruta de avance hacia el Trabajo Final Integrador (TFI)

Ruta de avance hacia el Trabajo Final Integrador (TFI)

Ahora que llegaste a la clase 4, estás en una posición ideal para comenzar a trabajar en los cimientos del programa que desarrollarás como parte del **Trabajo Final Integrador (TFI)**. Como sabés, en este proyecto, vas a crear una aplicación en Python que gestione información sobre los productos de una tienda. Es importante que empieces a planificar cómo integrar lo que aprendiste hasta ahora.

Con los conocimientos adquiridos en las clases anteriores, ya podés diseñar la estructura básica del programa. Por ejemplo, podés usar cadenas de texto para almacenar nombres, descripciones y cualquier otro dato textual. También podés aplicar los métodos de cadenas para validar y formatear la información que se ingrese por la terminal, como convertir todo a minúsculas o eliminar espacios en blanco innecesarios.

La validación de las entradas es otro aspecto fundamental que podés abordar con lo aprendido en las clases 3 y 4. Las estructuras condicionales como **if**, **elif** y **match** te permiten manejar diferentes escenarios. Por ejemplo, podés validar que un correo electrónico incluya el carácter **@** o que una edad ingresada sea un número positivo. Esto hace que tu programa sea más robusto y resistente a errores de ingreso.

También es importante planificar cómo va a interactuar el usuario con tu programa. Podés usar lo que aprendiste para mostrar menús simples que permitan elegir entre distintas opciones, como registrar un cliente o consultar los datos ingresados. Pensá en aplicar condicionales y métodos de cadenas para hacer más clara y amigable la presentación de estas opciones.

Aunque todavía no aprendiste a trabajar con bases de datos ni a guardar datos de manera persistente, no te preocupes. Más adelante en el curso, vamos a integrar estas herramientas con bases de datos para completar la funcionalidad del proyecto.

Como tarea concreta, te sugerimos que crees un programa que solicite los datos de un producto y los muestre en formato de tarjeta o ficha, utilizando **f-Strings** para formatear la salida. Aprovechá los métodos de cadenas para validar las entradas y asegurarte de que estén en el formato correcto.



Experimentá. Recordá que cada paso que avances ahora va a facilitar el desarrollo del proyecto final. Es mejor empezar con algo pequeño e ir construyendo sobre eso, en lugar de intentar resolver todo de una sola vez.

Ejercicio Práctico

Luego de haber pasado el día con Luis y aprendido a utilizar condicionales avanzados y los métodos de las cadenas, estás en condiciones de resolver la tarea que te mencionó Mariana y que acaba de formalizar mediante un correo electrónico:









Nuestro cliente nos pide que el programa ahora haga lo siguiente:

- Formatee correctamente los textos ingresados en “apellido” y “nombre”, convirtiendo la primera letra de cada palabra a mayúsculas y el resto en minúsculas.
- Asegurarse que el correo electrónico no tenga espacios y contenga solo una “@”.
- Que clasifique a sus clientes por rango etario basándose en su edad (“Niño/a” para los y las menores de 15 años, “Adolescente” de 15 a 18 y “Adulto/a” para personas mayores de 18 años.)

El programa debe mostrar el apellido, nombre y dirección de correo con el formato pedido, y el texto correspondiente a su rango etario.

¡Estoy segura de que harás un excelente trabajo!

 Recargar
  Redactar
  Responder
  Reenviar
  Eliminar
  Favorito

De: Mariana – Gerente de Proyectos

Asunto: Recorrer y analizar datos

¡Hola!

Espero que estes bien. Nuestro cliente nos pide que el programa ahora haga lo siguiente:

- Formatee correctamente los textos ingresados en "apellido" y "nombre", convirtiendo la primera letra de cada palabra a mayúsculas y el resto en minúsculas.
- Asegurarse que el correo electrónico no tenga espacios y contenga solo una "@".
- Que clasifique a los clientes por rango etario basándose en su edad ("Niño" para los menores de 15 años, "Adolescente" de 15 a 18 y "Adulto" para los mayores de 18 años.)

El programa debe mostrar el apellido, nombre y dirección de correo con el formato pedido, y el texto correspondiente a su rango etario.
¡Estoy segura de que harás un excelente trabajo!

Saludos,
Mariana Gómez
Gerente de Proyectos

Materiales y Recursos Adicionales:

Artículos:

Python Docs: [Más herramientas para control de flujo](#)

FreeCodeCamp: [Tutorial de f-strings en Python](#)

El libro de Python: [Cadenas Python](#)

Preguntas para reflexionar:

1. ¿Cómo creés que las herramientas avanzadas como **elif** y **match** pueden simplificar la lógica de un programa en comparación con los condicionales básicos? Pensá en ejemplos concretos donde estas estructuras serían especialmente útiles.
2. ¿De qué manera los métodos de cadenas como **.lower()**, **.upper()** o **.title()** te permiten mejorar la calidad de los datos ingresados por los y las usuarios? ¿Qué impacto podría tener esto en programas más complejos o en aplicaciones del mundo real?

Próximos pasos:

En la próxima clase, incorporaremos el manejo del flujo de control aprendiendo a trabajar con los **bucles while**. Esta herramienta es indispensable para repetir acciones de forma controlada, permitiéndote realizar tareas que antes requerían múltiples líneas de código con estructuras condicionales. Aprenderás a usar contadores y acumuladores para procesar datos de manera más eficiente, lo cual es fundamental para trabajar con listas y grandes volúmenes de información.

También vas a explorar las sentencias **break** y **continue**, que te permiten salir anticipadamente de un bucle o saltar a la siguiente iteración, adaptando el comportamiento de tu programa a situaciones específicas. Estas herramientas, combinadas con las estructuras condicionales que ya manejas, te darán el control necesario para construir programas más dinámicos y robustos.

Por último, introduciremos las **listas**, una estructura poderosa para almacenar y manipular múltiples valores. Esto te permitirá empezar a organizar datos de una manera más flexible, como guardar la información de varios clientes o productos en un solo lugar, preparándote para los desafíos más complejos del Trabajo Final Integrador.

¡Seguimos avanzando hacia programas más eficientes y completos! Llegar preparado con una buena comprensión de las herramientas vistas hasta ahora hará que puedas aprovechar al máximo las nuevas técnicas que vamos a desarrollar en la próxima clase.



Buenos Aires
aprende
Agencia de Habilidades para el Futuro

BA Buenos
Aires
Ciudad