📝 Apuntes de Programación: El Formato de Cadenas con F-Strings

Las F-Strings (cadenas literales con formato) son el método moderno y más potente de Python para incrustar expresiones y variables dentro de una cadena. Son rápidas, legibles y permiten aplicar modificadores de formato complejos.

1. La Evolución del Formato

Antes de las F-Strings, se usaban métodos como el operador `%` o el método `.format()`. Es crucial conocer el método `%` (presente en tu código) para entender la evolución.

A. Formato Antiguo (Operador `%`)

Este método requiere memorizar códigos de tipo (como `%s` para string, `%d` para entero, `%.2f` para float con 2 decimales).

```
print('Mi nombre %s, tengo %d años y cobro %.2f euros al mes' % (nombre, edad, 1000.501
# %s: placeholder para string
# %d: placeholder para decimal (entero)
# %.2f: placeholder para float (flotante) con 2 decimales
```

B. El Estándar Moderno: F-Strings

Las F-Strings se identifican por la letra `f` (o `F`) que precede a la cadena. Las variables o expresiones se colocan directamente dentro de llaves `{}`.

```
#* Forma 1: Inserción simple
print(f"mi nombre {nombre} tengo {edad} años y cobro {sueldo} euros al mes")
```

2. Modificadores de Formato (F-Strings Avanzadas)

Dentro de las llaves, puedes añadir un signo de dos puntos `:` seguido de un *modificador* para controlar la presentación de la variable.

A. Control de Decimales y Notación Científica

Se usa el modificador `:.Nf` para indicar `N` decimales.

```
#* Modificador float con 2 decimales (redondea automáticamente)
print(f"mi nombre {nombre} tengo {edad} años y cobro {sueldo : .2f} euros al mes")
```

B. Porcentajes y Separador de Millares

Modificador	Propósito	Ejemplo de Código	Salida Típica

`:.N%`	Formatea el número como porcentaje con `N` decimales. Multiplica por 100 automáticamente.	`print(f"Porcentajes: {ratio:.2%}")`	`Porcentajes: 8.39%`
`:,`	Añade comas o puntos como separador de miles (depende de la configuración regional).	`print(f"Población: {habitantes:,} habitantes")`	`Población: 712,670,000 habitantes`

C. Alineación y Relleno

Los modificadores de alineación se usan para reservar un espacio mínimo y colocar el texto dentro de él.

Modificador	Propósito	Ejemplo
`:<20`	Alinea a la **izquierda** en un espacio de 20 caracteres.	`print(f"***{texto:<20}****")`
`:>20`	Alinea a la **derecha** en un espacio de 20 caracteres.	`print(f"***{texto:>20}****")`
`:^20`	Alinea al **centro** en un espacio de 20 caracteres.	`print(f"***{texto:^20}****")`
`: 4d`	Alinea un entero (d) a la derecha, reservando 4 espacios.	`print(f"{num1: 4d}")`

3. Características Avanzadas

A. F-Strings Multilínea

Puedes usar comillas triples (`"""`) para crear bloques de texto que abarquen varias líneas, y las F-Strings seguirán insertando variables.

```
ficha=f"""
ficha del profesor/a:
====================
Nombre: {nombre}
...
"""
```

Nota sobre llaves (`{}`): Si necesitas imprimir una llave literal dentro de una F-String multilínea, debes duplicarla (ej. `{{euros€}}`).

B. Debugging con el Signo `=`

Esta es una característica moderna y muy útil para la depuración. Añadir `=` dentro de las llaves muestra el nombre de la variable, un signo igual y su valor.

```
#* Muestra la variable, un = y su valor.
print(f"{num1=}\n{num2=}\n{texto=}")
# Salida: num1=45 / num2=123 / texto='Python'
```

C. Operadores Condicionales Ternarios

Puedes incluir lógica simple directamente en las llaves para mostrar diferentes resultados basados en una condición (ideal para estados "Sí/No", "Par/Impar", "Alto/Medio/Bajo").

- **Sintaxis:** `{'valor_si_verdadero' if condicion else 'valor_si_falso'}`

```
print(f" ¿numero es par? {'verdadero'if numero%2==0 else 'falso'}")

# Lógica encadenada (si... sino si... sino...)
print(f"Valoraciones: {'Alto' if numero > 50 else 'Medio' if numero>25 else 'Bajo'}
```

4. Consejos para Mejorar la Programación con F-Strings (Examen Pro-Tip)

🎯 Principio 1: ¡Deja de usar el viejo `%` !

- **Consejo:** En código moderno, utiliza siempre F-Strings. Son más rápidas y eliminan la necesidad de recordar códigos de formato como `%s` o `%d` .

🎯 Principio 2: Usa F-Strings para la Conversión Rápida

- Cuando necesites convertir un número (o cualquier tipo de dato) a una cadena para una operación rápida (ej. logging), simplemente insértalo en una F-String vacía.

```
cadena_numero = f"{num1}" # Es más legible que str(num1)
```

🎯 Principio 3: Inclusión de Expresiones

- Recuerda que puedes poner **cualquier expresión válida de Python** dentro de las llaves `{}` . Esto incluye llamadas a funciones, operaciones matemáticas e incluso métodos de objeto.

```
nombre = "moha"
```