📝 Apuntes de Programación: Slicing y Métodos Avanzados de Cadenas

Las Cadenas de Caracteres (`str`) son secuencias ordenadas de caracteres en Python. Su manejo eficiente y la capacidad de extraer subcadenas (Slicing) son habilidades esenciales en programación.

1. El Slicing (Rebanado o Sintaxis "Bocadillo")

El *slicing* permite obtener una porción (subcadena) de una cadena original. Utiliza la sintaxis de corchetes con hasta tres parámetros opcionales.

Sintaxis General

$$\text{cadena}[\text{inicio} : \text{fin} : \text{paso}]$$

Parámetro	Propósito	Valor por Defecto
Inicio	Posición donde comienza la porción (incluida).	`0` (principio)
Fin	Posición donde termina la porción (**excluida**).	Final de la cadena
Paso	Número de caracteres a saltar entre cada elemento.	`1` (consecutivo)

A. Uso con Índices Positivos y Negativos

- **Índices Positivos:** Comienzan en `0` desde la izquierda.

```
texto = "Hola mundo"
print(texto[3:8]) # 'a mun' (Desde la posición 3 hasta la 7)
```

- **Índices Negativos:** Comienzan en `-1` desde la derecha (el último caracter es `-1`).

```
texto3 = "Hola mundo"
print(texto3[-5:-2]) # 'mun' (Desde 'm' hasta antes de 'd')
```

- **Mezcla:** Se pueden combinar, siempre que el índice de inicio sea lógicamente anterior al índice de fin.

```
print(texto4[2:-2]) # 'la mun' (Desde 'l' hasta antes de 'd')
```

B. El Tercer Parámetro (`Paso`)

El parámetro `paso` permite saltarse caracteres durante el rebanado o invertir la secuencia.

Sintaxis	Resultado	Propósito
`[::2]`	Caracteres en posiciones pares (`0, 2, 4...`).	Extraer caracteres pares.
`[1::2]`	Caracteres en posiciones impares (`1, 3, 5...`).	Extraer caracteres impares.
`[::-1]`	Caracteres al revés (paso de `-1`).	**Invertir la cadena.**
`[::-2]`	Caracteres al revés, saltando de dos en dos.	Invertir y saltar.

2. Recorrido y Concatención

A. Recorrido (Iteración)

Existen dos formas estándar de recorrer una cadena, ya que las cadenas son iterables.

1. **Iteración Directa (Solo valor):** Más legible y común.

```
for caracter in texto6:
    print(caracter, end="-")
```

2. **Iteración por Posición (Índice):** Útil si necesitas saber el índice.

```
for position in range(0, len(texto6)):
    print(position, "-", texto6[position])
```

B. Concatenación

- **Operador `+` :** El método más sencillo, pero crea una nueva cadena en memoria.

```
texto6 = "Hola" + "mundo" + "cruel" + str(33)
```

- **Función `len()` :** Devuelve la longitud (número de caracteres) de la cadena.

```
print(len(texto6))
```

3. Métodos Clave de Manipulación

Método	Propósito	Ejemplo
`str.upper()`	Convierte toda la cadena a mayúsculas.	`texto7.upper()`
`str.lower()`	Convierte toda la cadena a minúsculas.	`texto7.lower()`

`str.swapcase()`	Invierte el caso de las letras (mayúsculas ↔ minúsculas).	`texto7.swapcase()`
`str.find(sub, start)`	Devuelve el índice de la **primera** ocurrencia de la subcadena. Devuelve `-1` si no se encuentra. El parámetro opcional `start` permite iniciar la búsqueda desde una posición específica.	`texto7.find("o", 2)`
`str.count(sub)`	Devuelve el número de veces que aparece una subcadena o carácter.	`texto7.count("o")`
`str.replace(old, new, count)`	Sustituye todas las ocurrencias de `old` por `new`. El parámetro opcional `count` limita el número de sustituciones.	`texto7.replace(" ", "x", 1)`

4. Consejos para Mejorar la Programación con Cadenas de Texto

🎯 Principio 1: Concatenación Eficiente (Método `join`)

- **Rendimiento para Examen:** Cuando necesitas concatenar muchas cadenas o elementos de una lista, el operador `+` es lento porque crea una nueva cadena en cada operación. La mejor práctica (y más eficiente) es usar el método `str.join()`.

```
elementos = ["Parte1", "Parte2", "Parte3"]
# Mucho más eficiente que usar '+' repetidamente
resultado = "".join(elementos) # Resultado: "Parte1Parte2Parte3"
```

🎯 Principio 2: Slicing para Copiar

- Dado que las cadenas son inmutables, la forma más rápida y concisa de obtener una copia completa de una cadena (aunque no estrictamente necesario por la inmutabilidad) es usar el *slicing* completo: `cadena[:]`.

🎯 Principio 3: Usar `in` en lugar de `find()` para Búsqueda Simple

- Si solo quieres saber si una subcadena existe (y no necesitas el índice), usar el operador `in` es más legible y a menudo más rápido que `find()`:

```
if "mundo" in texto7:
    print("La palabra existe.")
```