

Cadenas de texto

Edison Achalma

Escuela Profesional de Economía, Universidad Nacional de San Cristóbal de Huamanga

Resumen

Este abstract será actualizado una vez que se complete el contenido final del artículo.

Tabla de contenidos

Introduction	2
0.1 Motivación	2
0.2 Intuición	2
0.3 Modelo mental	2
0.4 Desarrollo	2
0.4.1 1. Creación de Strings	2
0.4.2 2. Acceso a caracteres (indexación)	2
0.4.3 3. Longitud de un string	3
0.4.4 4. Slicing (rebanado)	3
0.4.5 5. Iteración sobre strings	3
0.4.6 6. Verificar contenido con in	3
0.4.7 7. Operaciones básicas con strings	4
0.5 Ejemplos	4
0.6 Errores comunes	5
0.7 Buenas prácticas	5
0.8 Ejercicio	6
0.9 Síntesis	7
0.10 Qué sigue	7
1 Publicaciones Similares	7

Edison Achalma  <https://orcid.org/0000-0001-6996-3364>

El autor no tiene conflictos de interés que revelar. Los roles de autor se clasificaron utilizando la taxonomía de roles de colaborador (CRediT; <https://credit.niso.org/>) de la siguiente manera: Edison Achalma: conceptualización, metodología, análisis formal, investigación, recursos, curación de datos, redacción, visualización, supervisión, administración del proyecto

La correspondencia relativa a este artículo debe dirigirse a Edison Achalma, Escuela Profesional de Economía, Universidad Nacional de San Cristóbal de Huamanga, Portal Independencia N° 57, Ayacucho, AYA 5001, Perú, Email: elmer.achalma.09@unsch.edu.pe

Cadenas de texto

0.1 Motivación

Las cadenas de texto (strings) son uno de los tipos de datos más utilizados en programación. Desde mostrar mensajes al usuario hasta procesar datos de archivos o bases de datos, el manejo de texto es fundamental. Python ofrece una forma potente y flexible de trabajar con strings, con una sintaxis clara y numerosos métodos para manipularlos.

0.2 Intuición

Imagina un string como una **fila de caracteres** donde cada posición tiene un número (índice). Es como un tren donde cada vagón es una letra, y puedes acceder a cualquier vagón por su número. Python trata los strings como secuencias inmutables: una vez creados, no puedes cambiar los caracteres individuales, pero puedes crear nuevos strings a partir de ellos.

0.3 Modelo mental

- **Strings son secuencias:** cada carácter tiene una posición (índice), empezando desde 0.
- **Inmutables:** no puedes cambiar un carácter directamente; creas uno nuevo.
- **Indexación:** accedes con `[i]` (i desde 0). Índices negativos empiezan desde el final.
- **Multilínea:** con triple comilla (`"""` o `'''`) preserva saltos de línea.
- **Inmutabilidad:** los métodos devuelven nuevos strings, no modifican el original.

0.4 Desarrollo

0.4.1 1. Creación de Strings

```
# Comillas simples o dobles
nombre = "Ana"
apellido = 'García'

# Strings con comillas internas
texto1 = "It's a beautiful day"
texto2 = 'Ella dijo "Hola"'

# Strings multilínea (con triple comilla)
poema = """Roses are red,
Violets are blue,
Python is awesome,
And so are you!"""
print(poema)
```

0.4.2 2. Acceso a caracteres (indexación)

```
texto = "Python"
print(texto[0]) # 'P'
print(texto[1]) # 'y'
print(texto[2]) # 't'
```

```
# Índices negativos (desde el final)
print(texto[-1]) # 'n'
print(texto[-2]) # 'o'

# Error: índice fuera de rango
# print(texto[10]) # IndexError
```

0.4.3 3. Longitud de un string

```
texto = "Hola Mundo"
print(len(texto)) # 10 (incluye el espacio)
```

0.4.4 4. Slicing (rebanado)

Extrae partes de un string:

```
texto = "Python Programming"

# texto[inicio:fin] (fin no incluido)
print(texto[0:6]) # "Python"
print(texto[7:18]) # "Programming"

# Desde el inicio
print(texto[:6]) # "Python"

# Hasta el final
print(texto[7:]) # "Programming"

# Con índices negativos
print(texto[-11:]) # "Programming"
print(texto[-11:-1]) # "Programmin"
```

0.4.5 5. Iteración sobre strings

```
palabra = "Python"
for letra in palabra:
    print(letra)
# Salida: P, y, t, h, o, n

# Con índice y carácter
for i, letra in enumerate(palabra):
    print(f"Posición {i}: {letra}")
```

0.4.6 6. Verificar contenido con in

```
frase = "El gato está en el tejado"

# Verificar si contiene una subcadena
print("gato" in frase)      # True
print("perro" in frase)    # False
print("tejado" not in frase) # False

# Uso en condicional
if "gato" in frase:
    print("¡Hay un gato!")
```

0.4.7 7. Operaciones básicas con strings

```
# Concatenación
nombre = "Ana"
apellido = "García"
nombre_completo = nombre + " " + apellido
print(nombre_completo) # "Ana García"

# Repetición
eco = "Hola! " * 3
print(eco) # "Hola! Hola! Hola! "

# Comparación (lexicográfica)
print("abc" < "abd") # True (c < d)
print("Ana" == "ana") # False (diferentes mayúsculas)
print("Ana".lower() == "ana".lower()) # True
```

0.5 Ejemplos

Mínimo — Creación y acceso:

```
mensaje = "Hola Mundo"
print(mensaje[0]) # H
print(mensaje[5]) # M
print(len(mensaje)) # 10
```

Progresivo — Manipulación básica:

```
nombre = "Carlos"
print(nombre.upper()) # CARLOS
print(nombre.lower()) # carlos
print(nombre[0]) # C
print(nombre[-1]) # s
print(len(nombre)) # 6
```

Real — Sistema de validación de usuario:

```
# Datos de usuario
nombre_usuario = "  Ana_Perez  "
email = "ANA@GMAIL.COM"

# Limpieza de datos
nombre_limpio = nombre_usuario.strip() # "Ana_Perez"
email_limpio = email.lower()           # "ana@gmail.com"

# Validación
if "@" in email_limpio:
    print(f"Email válido: {email_limpio}")
else:
    print("Email inválido")

# Extracción de nombre de usuario
if "_" in nombre_limpio:
    partes = nombre_limpio.split("_")
    print(f"Nombre: {partes[0]}") # "Ana"
    print(f"Apellido: {partes[1]}") # "Perez"
```

0.6 Errores comunes

Error	Síntoma	Solución
Índice fuera de rango	IndexError: string index out of range	Verifica que el índice sea < len(string)
Confundir = con ==	Asignación en lugar de comparación	Usa == para comparar
Modificar string directamente	TypeError: 'str' object does not support item assignment	Crea un nuevo string o usa listas
Olvidar convertir tipos	TypeError: can only concatenate str to str	Convierte con str() o usa f-strings
Espacios invisibles	Comparaciones fallan	Usa .strip() para limpiar

0.7 Buenas prácticas

1. Usa f-strings para formateo:

```
nombre = "Ana"
edad = 25
mensaje = f"Hola, {nombre}. Tienes {edad} años"
```

2. Limpia strings antes de comparar:

```
usuario = "  ana  "
if usuario.strip().lower() == "ana":
    print("Usuario válido")
```

3. Usa métodos de string en lugar de bucles manuales:

```
# Menos eficiente
resultado = ""
for char in "abc":
    resultado += char.upper()

# Mejor
resultado = "abc".upper()
```

4. Prefiere in para búsquedas:

```
# Claro y eficiente
if "error" in mensaje:
    print("Hay un error")
```

5. Usa slicing para extraer partes:

```
email = "usuario@email.com"
nombre = email[:email.index("@")]
dominio = email[email.index("@") + 1:]
```

0.8 Ejercicio

Intermedio — Procesamiento de texto

Escribe un programa que:

1. Tenga un string texto con “El rápido zorro marrón salta sobre el perro perezoso”
2. Extraiga y muestre:
 - El primer carácter
 - Los últimos 5 caracteres
 - La palabra “zorro” (usando slicing)
 - La longitud total
 - El texto en mayúsculas
 - El texto en minúsculas
3. Verifique si contiene “perro” y “gato”
4. Reemplace “perro” por “gato” en el texto
5. Divida el texto en palabras y muestre cuántas hay

Solución (no mires hasta haberlo intentado):

```
texto = "El rápido zorro marrón salta sobre el perro perezoso"

# 1. Caracteres
print(f"Primer carácter: {texto[0]}")
```

```
print(f"Últimos 5: {texto[-5:]}")
print(f"Palabra 'zorro': {texto[10:15]}") # Asumiendo posición fija

# 2. Longitud y caso
print(f"Longitud: {len(texto)}")
print(f"Mayúsculas: {texto.upper()}")
print(f"Minúsculas: {texto.lower()}")

# 3. Búsquedas
print(f"¿Contiene 'perro'? {'perro' in texto}")
print(f"¿Contiene 'gato'? {'gato' in texto}")

# 4. Reemplazo
texto_modificado = texto.replace("perro", "gato")
print(f"Texto modificado: {texto_modificado}")

# 5. División en palabras
palabras = texto.split()
print(f"Número de palabras: {len(palabras)}")
print(f"Primeras 3 palabras: {palabras[:3]}")
```

0.9 Síntesis

- **Strings:** secuencias de caracteres, inmutables, indexación desde 0.
- **Comillas:** simples ('), dobles ("), triples (""" o ''').
- **Indexación:** [i] (positivo) y [-i] (negativo desde el final).
- **Slicing:** [inicio:fin:paso] para extraer subcadenas.
- **len():** obtiene la longitud del string.
- **Métodos comunes:** .upper(), .lower(), .strip(), .split(), .replace(), .join().
- **in:** verifica si una subcadena está presente.
- **F-strings:** la forma recomendada de formatear strings con variables.

0.10 Qué sigue

- **Relacionado:** métodos de string, f-strings, expresiones regulares
- **Siguiente tema:** Métodos de strings — profundiza en las herramientas para manipular texto.

1 Publicaciones Similares

Si te interesó este artículo, te recomendamos que explores otros blogs y recursos relacionados que pueden ampliar tus conocimientos. Aquí te dejo algunas sugerencias:

1. [📄 Instalacion De Anaconda](#)
2. [📄 Configurar Entorno Virtual Python Anaconda](#)
3. [📄 01 Introducion A La Programacion Con Python](#)
4. [📄 02 Variables Expresiones Y Statements Con Python](#)
5. [📄 03 Objetos De Python](#)

6.  [04 Ejecucion Condicional Con Python](#)
7.  [05 Iteraciones Con Python](#)
8.  [06 Funciones Con Python](#)
9.  [07 Dataframes Con Python](#)
10.  [08 Prediccion Y Metrica De Performance Con Python](#)
11.  [09 Metodos De Machine Learning Para Clasificacion Con Python](#)
12.  [10 Metodos De Machine Learning Para Regresion Con Python](#)
13.  [Visualizacion De Datos Con Python](#)

Esperamos que encuentres estas publicaciones igualmente interesantes y útiles. ¡Disfruta de la lectura!