Manejo de Cadenas (Strings) y Listas en Python

Materia: Algoritmos y Programación

Prof. D.Sc. BARSEKH-ONJI Aboud

Facultad de Ingeniería
Universidad Anáhuac México

7 de diciembre de 2025

Agenda

Extrayendo Caracteres: Los Índices

Los Strings son Secuencias Ordenadas

Cada carácter en un string tiene una posición o **índice**. En Python, la numeración de los índices siempre comienza en **cero (0)**.

P	Y	T	H	O	N
0	1	2	3	4	5

Extrayendo Caracteres: Los Índices

Índices Positivos (de Izq. a Der.)

Se usan para acceder a los caracteres desde el principio del string.

```
palabra = 'PYTHON'

# Acceder al primer caracter
primer_letra = palabra[0] # 'P'

# Acceder al tercer caracter
tercer_letra = palabra[2] # 'T'

```

Índices Negativos (de Der. a Izq.)

Permiten acceder a los caracteres desde el final del string. -1 es el último carácter.

```
palabra = 'PYTHON'

# Acceder al ultimo caracter
ultima_letra = palabra[-1] # 'N'

# Acceder al penultimo caracter
penultima_letra = palabra[-2] # 'O
    '

```

Agenda

Introducción a las Listas: Contenedores Ordenados

¿Qué es una Lista?

Una **lista** es una colección de elementos que está **ordenada** y es **mutable** (es decir, se puede cambiar después de crearla). Se definen con corchetes `[]` y los elementos se separan por comas.

- Pueden contener diferentes tipos de datos a la vez.

Introducción a las Listas: Contenedores Ordenados

Accediendo a Elementos con Índices

Las listas usan el **mismo sistema de índices que los strings** para acceder a sus elementos:

- El primer elemento está en el índice 0.
- El último elemento está en el índice -1.

```
# Una lista con 4 elementos de diferentes tipos
calificaciones = [10, 9.5, 'Aprobado', True]
# Acceder al primer elemento
primer_elemento = calificaciones[0] # 10
# Acceder al tercer elemento
tercer_elemento = calificaciones[2] # 'Aprobado'
# Acceder al ultimo elemento
ultimo_elemento = calificaciones[-1] # True

```

Agenda

Caracteres Especiales: Secuencias de Escape

¿Cómo Escribir Caracteres "Invisibles"?

Para representar caracteres especiales como saltos de línea, tabulaciones o incluso comillas dentro de un string, usamos **secuencias de escape**, que comienzan con una barra invertida (\).

Secuencias Comunes

Secuencia	Significado
`\n`	Salto de línea
`\t`	Tabulación
`\\`	Barra invertida
`\'`	Comilla simple
`\''`	Comilla doble

Caracteres Especiales: Secuencias de Escape

Examples

```
# Salto de linea con \n
mensaje = 'Linea 1 \n Linea 2'
print(mensaje)
# Linea 1
# Linea 2

# Tabulacion con \t
lista = 'Productos: \n \t -Manzanas \n \t -Leche'
print(lista)

# Comillas dentro de un string
cita = 'El me dijo: \'Python es genial\'.'
print(cita)
# El me dijo: 'Python es genial'.

```

Agenda

Formateando Strings con f-strings

El Método 'Antiguo'

Antes, para combinar texto y variables, se usaba la concatenación y la función 'str()', lo que podía volverse verboso y difícil de leer.

```
nombre = 'Maria'
edad = 30
promedio = 9.5

# Dificil de leer y escribir
print('Estudiante: ' + nombre + ',
     Edad: ' + str(edad) + ',
    Promedio: ' + str(promedio))

```

La Solución Moderna: f-strings

Un **f-string** (string formateado) simplifica enormemente esto. Solo debes poner una `f` antes de las comillas e insertar las variables directamente dentro de llaves {}.

```
nombre = 'Maria'
edad = 30
promedio = 9.5

# Mucho mas limpio y legible
print(f'Estudiante: {nombre}, Edad
    : {edad}, Promedio: {promedio}'
    )

```

Formateo Avanzado de Números con f-strings

Controlando la Apariencia de los Números

Los f-strings no solo insertan valores, sino que también nos permiten controlar con precisión cómo se muestran los números, lo cual es fundamental para crear salidas de datos limpias y profesionales.

Formateo Avanzado de Números con f-strings

Precisión Fija para Flotantes

Podemos especificar el número de decimales a mostrar usando `:.Nf`, donde `N` es el número de decimales.

```
pi = 3.14159265
total = 1234.5
# Mostrar pi con 2 decimales
print(f'El valor de pi es: {pi:.2f}')
# Salida: El valor de pi es: 3.14
# Formato de moneda
print(f'Total a pagar: ${total:.2f}')
# Salida: Total a pagar: $1234.50

```

Ceros a la Izquierda para Enteros

Podemos rellenar un número con ceros a la izquierda para que tenga un ancho fijo usando `:0Nd`, donde `N` es el ancho total.

```
numero_factura = 45
dia = 7
mes = 9
# Rellenar a 5 digitos
print(f'Factura No: {numero_factura:05d}')
# Salida: Factura No: 00045
# Formato de fecha
print(f'Fecha: {dia:02d}/{mes:02d}/2023')
# Salida: Fecha: 07/09/2023

```

Conociendo Nuestras Variables: 'type()' y 'id()'

Funciones de Introspección

Python nos ofrece funciones para 'preguntar' a nuestras variables qué son y dónde están en la memoria.

Conociendo Nuestras Variables: 'type()' y 'id()'

'type()' - ¿Qué es esto?

Devuelve el **tipo de dato** de una variable. Es muy útil para depurar y entender cómo se están manejando los datos.

```
numero = 100
texto = 'Hola'
es_valido = True
print(type(numero))
# Salida: <class 'int'>
print(type(texto))
# Salida: <class 'str'>
print(type(es_valido))
# Salida: <class 'bool'>

```

Conociendo Nuestras Variables: 'type()' y 'id()'

'id()' - ¿Dónde está esto?

Devuelve el **identificador de memoria** único de un objeto. Es un número que representa la dirección donde está guardado el dato en la memoria RAM.

```
x = 10
y = x # 'y' apunta al mismo objeto que 'x'
z = 10 # Python es eficiente, reutiliza el objeto
# Muestran el mismo ID, porque apuntan
# al mismo objeto '10' en memoria
print(id(x))
print(id(y))
print(id(z))

```

Tarea 1: Decodificador de Mensaje Secreto

Objetivo

Usar el conocimiento de **índices** para extraer caracteres de un string y revelar un mensaje oculto. Deberás usar un bucle `for` para resolverlo.

Tarea 1: Decodificador de Mensaje Secreto

El Código a Completar

```
# El mensaje cifrado contiene la informacion oculta
mensaje_cifrado = 'aPzYleTtnHhOoNnlax'

# Estos son los indices de los caracteres correctos
indices_secretos = [1, 5, 7, 8, 10, 12]

# Variable para guardar el resultado
mensaje_decodificado = ''
# ------------------------
# Debes crear un bucle 'for' que recorra los 'indices_secretos'.
# En cada paso, extrae el caracter de 'mensaje_cifrado'
# usando el indice actual y anadelo a 'mensaje_decodificado'.
# Pista: usa el operador '+=' para anadir caracteres.
# Al final, imprime el mensaje decodificado
print('El mensaje secreto es:', mensaje_decodificado)
# La salida deberia ser: Python

```

Tarea 1: Decodificador de Mensaje Secreto

Revisa el código completo

https://github.com/AboudOnji/ExamplesAyP/blob/main/Example15.py

Tarea 2: Generador de Recibos Formateado

Objetivo

Utilizar **listas** para almacenar datos y **f-strings** para generar un recibo de compra con un formato profesional y alineado.

Tarea 2: Generador de Recibos Formateado

El Código a Completar

```
productos = ['Leche Entera', 'Pan de Caja', 'Huevo (12pza)']
precios = [25.50, 42.00, 38.95]
total = 0.0
print('--- RECIBO DE COMPRA ---')
print('No. | Producto      | Precio')
print('--------------------------')
# Crea un bucle 'for' que recorra las listas usando un rango
# de indices (pista: for i in range(len(productos)):).
# Dentro del bucle, imprime una linea formateada para cada producto:
# 1. El numero de item (i+1), con un cero a la izquierda (ej: 01).
# 2. El nombre del producto (productos[i]).
# 3. El precio (precios[i]), con exactamente 2 decimales.
# 4. Acumula el precio en la variable 'total'.
# Al final, imprime una linea de separacion y el total
print('--------------------------')


```

Tarea 2: Generador de Recibos Formateado

El Código a Completar

```
# print(f'TOTAL:          ${total:.2f}')
# La salida deberia verse asi:
# --- RECIBO DE COMPRA ---
# No. | Producto      | Precio
# --------------------------
# 01  | Leche Entera  | 25.50
# 02  | Pan de Caja   | 42.00
# 03  | Huevo (12pza) | 38.95
# --------------------------
# TOTAL:          $106.45

```

Tarea 2: Generador de Recibos Formateado

Revisa el código completo

https://github.com/AboudOnji/ExamplesAyP/blob/main/Example16.py

Traduciendo Caracteres: 'ord()' y 'chr()'

El Código Detrás de Cada Carácter

Cada carácter que ves en la pantalla (como 'A', 'b', o '$') está almacenado en la memoria como un número. El estándar que define esta correspondencia se llama **Unicode** (y su subconjunto más conocido es ASCII).

Traduciendo Caracteres: 'ord()' y 'chr()'

De Carácter a Número: 'ord()'

La función `ord()` (de 'ordinal') toma un carácter y devuelve su código numérico Unicode.

```
# Obtener el codigo de 'A'
codigo_A = ord('A')
print(codigo_A) # Imprime: 65

# Obtener el codigo de 'b'
codigo_b = ord('b')
print(codigo_b) # Imprime: 98

```

De Número a Carácter: 'chr()'

La función `chr()` (de 'character') hace lo opuesto: toma un código numérico y devuelve el carácter correspondiente.

```
# Obtener el caracter para 65
caracter_1 = chr(65)
print(caracter_1) # Imprime: 'A'

# Obtener el caracter para 98
caracter_2 = chr(98)
print(caracter_2) # Imprime: 'b'

```

Tarea 3: Cifrador César Simple

Objetivo

Escribir un programa que cifre un mensaje moviendo cada letra un número determinado de posiciones en el alfabeto. Para esto, deberás combinar el uso de bucles `for` con las funciones `ord()` y `chr()`.

Tarea 3: Cifrador César Simple

El Código a Completar

```
mensaje_original = input('Introduce el mensaje a cifrar: ')
desplazamiento = int(input('Introduce el desplazamiento (ej: 3): '))

mensaje_cifrado = ''

# --- TU CODIGO AQUI ---
# Crea un bucle 'for' que recorra cada 'letra' en 'mensaje_original'.
# Dentro del bucle:
# 1. Obtiene el codigo numerico de la 'letra' con ord().
# 2. Sumale el 'desplazamiento' para obtener el nuevo codigo.
# 3. Convierte el nuevo codigo de vuelta a un caracter con chr().
# 4. Anade el nuevo caracter a 'mensaje_cifrado'.
print('Mensaje cifrado:', mensaje_cifrado)
# Ejemplo de salida:
# Si mensaje='HOLA' y desplazamiento=3, la salida deberia ser 'KROD'

```

Tarea 3: Cifrador César Simple

Revisa el código completo

https://github.com/AboudOnji/ExamplesAyP/blob/main/Example17.py