Tema 3: Cadenas de texto

Héctor Xavier Limón Riaño

April 3, 2024

Contents

1 Generalidades sobre texto

- El texto es el tipo de dato más general, con texto se puede representar cualquier otro tipo de dato
- Los sistemas operativos tratan de forma especial al texto, ya que muchas cosas en el sistema se representan como texto:
  - Interfaz de línea de comandos
  - Archivos de configuración
  - Bitácoras del sistema
  - Entrada y salida estándar
  - Datos intercambiados por protocolos (como http)
  - Scripts
- Saber manipular texto es esencial en la automatización de tareas
- Solo se necesita texto para interactuar con una computadora (los administradores de servidores saben esto bien)
- Una computadora sólo maneja números binarios, esto es, ceros y unos

1.1 Terminología

1.1.1 Texto plano:

- Se refiere al texto que no ha sido procesado de forma especial

- Cuando hablamos de texto en este curso, hacemos referencia a texto plano
- Un archivo de texto plano, sólo tiene texto, no lo acompaña otros datos binarios
- Por ejemplo, el texto que escribes en un procesador de texto como word, no es texto plano, dado que se codifica a un formato propietario de Microsoft, que puede incluir imágenes u otros elementos binarios
- El código que escribes en tu editor es un ejemplo de texto plano, toda la programación se hace en texto plano
- El texto plano está estandarizado y no es propiedad de ninguna organización u empresa (es libre)
- Si valoras tu información usa formatos libres, como el texto plano

1.1.2 Parsing de texto

- Se refiere en general a procesar texto
- Está asociado al procesamiento en modo lectura (sin modificar el texto original)
- Se utiliza para lograr algunas de las siguientes cosas:
  - Recuperar elementos particulares en el texto
  - Transformar el texto entre formatos (renderizar a imagen por ejemplo)
  - Construir una estructura de datos que facilite el acceso a elementos: por ejemplo, construir un árbol a partir de un archivo `html`
- En computación se cuenta con una herramienta general muy poderosa para el parsing de texto: expresiones regulares (es un tema avanzado para este curso, lo ven más adelante en la carrera)

1.1.3 Parsers de texto

- Son herramientas creadas para facilitar el parsing de algún formato de texto en particular
- Suelen usar expresiones regulares

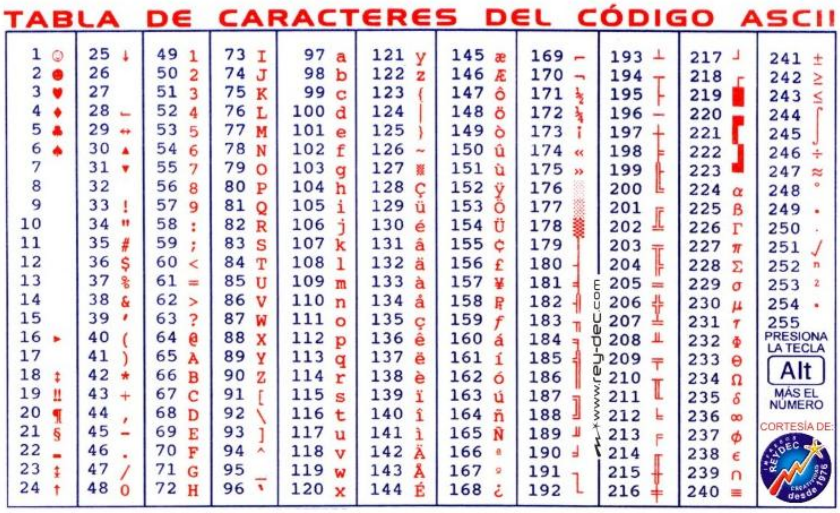
- Por ejemplo, si quieres parsear un archivo `html` conviene primero buscar si existe un parser para `html`
- El parser normalmente le entrega al programador (u otras partes del programa) alguna estructura de datos más fácil de procesar, un ejemplo común es un árbol
- Cabe decir que muchos tipos de programas como los compiladores y los navegadores web incluyen sus propios parsers
- Por ejemplo, un compilador primero parsea (mediante un parser del lenguaje de programación) el código fuente y luego construye un árbol sintáctico, con el cual es más fácil generar instrucciones

2 Estándares de texto

- Dado que las computadoras sólo pueden manejar números, es necesario contar con estándares para mapear entre si números y caracteres de texto
- A estos estándares también se les conoce como "charsets"
- A un mapeo en particular entre carácter y un número se le conoce como "code point"
- Existen muchos charsets, en este curso se mencionarán dos:

2.1 ASCII:

- Es el estándar clásico
- Originalmente pensado sólo para el alfabeto en inglés
- Define 128 caracteres, incluyendo dígitos, letras (mayúsculas y minúsculas) y caracteres especiales
- Para español se tiene una versión extendida de ascii con 256 caracteres (ASCII extendido)
- En un solo byte (hasta 256 valores) se pueden representar todos los caracteres
- Muchos caracteres son "no imprimibles" esto es, no los puedes ver en una pantalla, por ejemplo el carácter de `backspace`



2.1.1 Unicode:

- Un problema de ASCII es que no cubre muchos de los caracteres de varios idiomas: japones, griego, árabe, etc.
- Esto obligaba a tener muchos estándares diferentes, lo cual causa muchos problemas de compatibilidad
- En la era de Internet este es un problema serio
- Unicode es el estándar moderno que cubre todos los posibles caracteres de todos los idiomas
- Actualmente tiene definidos al rededor de 140,000 caracteres con espacio para definir más si hace falta
- Esto incluye cosas como emoticones
- Es una extensión de ASCII, esto es, los primeros 256 caracteres son los mismos
- Un problema de unicode es que ya no basta un byte para representar caracteres
- En python, la función `ord` regresa el code point correspondiente a un carácter
- Mientras que la función `chr` hace la operación inversa

```
print(ord('a'))
print(ord('A'))

print(chr(97))
print(chr(65))

# un carácter en japones
print(ord(''))

# un emoji
print(ord(''))
```

```
97
65
a
A
12371
128512
```

2.2 Codificación de texto

- Se refiere a cómo representar caracteres en código binario
- Esto permite hacer conversiones de texto a binario y vise-versa
- Esto es necesario, por ejemplo, al querer abrir un archivo de texto (internamente es binario, como todo), para que el sistema pueda mostrar los caracteres adecuados
- En `ASCII` la conversión es muy simple, cada byte es un carácter
- En Unicode la cosa es más complicada, se requieren a su vez estándares de codificación
- Estos estándares establecen la correspondencia entre bytes y caracteres
- Los archivos de texto tienen un metadato conocido como `BOM` que le indica al sistema el estándar de codificación con que fue almacenado
- Existen muchos estándares de codificación para Unicode
- Tratar diferentes codificaciones al mismo tiempo es una tarea en extremo compleja y propensa a errores

- Por esta razón, en la actualidad la mayoría de sistemas adoptan el estándar `utf-8` para todo
- Este estándar usa de 1 a 4 bytes (dependiendo del caracter) para representar caracteres y se considera eficiente (no desperdicia mucho espacio)
- Trata de usar siempre Unicode y `utf-8`
- Lo anterior es fácil en cualquier sistema diferente de Windows, ya que usan `utf-8` por defecto
- Si tu sistema no usa `utf-8` por defecto, puede ser que tengas que guardar manualmente con esa codificación (en las opciones de guardar como de tu editor)
- Si guardaste un archivo (código por ejemplo) en una codificación diferente a `utf-8` y lo intentas abrir como `utf-8` (cosa que puede pasar si creas un archivo en windows y luego lo quieres abrir en Linux) es posible que parte del texto se visualice corrupto (sobre todo en caracteres que rebasen ASCII)

```
# Revisar sistema de codificación por defecto del SO

import sys
print(sys.getdefaultencoding())

utf-8
```

```
# Ejemplo de codificación simple (solo caracteres ASCII)
s = 'hola'
print(len(s))
b = s.encode('utf-8') # convertir a binario
print(b) # ver cadena binaria
print(len(b))
print(list(b)) # ver valores de bytes
print(b.decode('utf-8')) # regresar a texto
```

```
4
b'hola'
4
[104, 111, 108, 97]
hola
```

```
# Ejemplo de codificación más complicado
s = ’’ # hola en japones
print(len(s)) # python cuenta caracteres, no bytes
b = s.encode(’utf-8’) # convertir a binario
print(b) # ver cadena binaria
print(len(b)) # hay más de 5 bytes
print(list(b)) # ver valores de bytes
print(b.decode(’utf-8’)) # regresar a texto
```

```
5
b’\xe3\x81\x93\xe3\x82\x93\xe3\x81\xab\xe3\x81\xa1\xe3\x81\xaf’
15
[227, 129, 147, 227, 130, 147, 227, 129, 171, 227, 129, 161, 227, 129, 175]
```

- Python (a partir de su versión 3) es un lenguaje diseñado para trabajar con Unicode (esto no es así para muchos lenguajes que trabajan por defecto con ASCII)
- Esto le facilita mucho la vida a los programadores
- Por ejemplo, en el código de antes, la longitud de la cadena "" es 5, a pesar de que internamente puedan ser varios bytes (dependiendo de la codificación)
- A menos que tengas que hacer conversiones entre texto en diferentes estándares, no es necesario preocuparse mucho de cómo se maneja internamente el texto
- Mantente usando siempre `utf-8` y todo va a estar bien

3 Manejo de cadenas

3.1 Tipo `str`

- Es el tipo básico para manejar cadenas, con soporte nativo de Unicode
- Definido en la el núcleo del lenguaje (no hay que importar nada para usarlas)
- Existen otros tipos relacionados, pero en el curso sólo se verá este tipo
- Es un tipo lineal, estático y no mutable

- En lenguajes como `C` es un arreglo de caracteres (en Python no es necesariamente así)
- Cualquier tipo de dato puede potencialmente transformarse a cadena mediante la función `str`
- Dado que es lineal es indexable

```
s = 'hola mundo'
print(s[0])  # primera
print(s[-1]) # última
print(s[len(s)-1]) # también última
```

```
h
o
o
```

3.2 Creación de cadenas

- Existen dos formas literales válidas:

  ```
  'Ejemplo de cadena'
  "Ejemplo de cadena"
  ```

- En general se prefiere el estilo de comilla simple (estilo PEP8)
- Se puede usar comilla doble si se quiere tener una comilla simple literal, o bien se puede usar el carácter de escape (backslash)
- El carácter de escape es muy útil en varias situaciones y es una forma de insertar caracteres especiales no imprimibles como salto de línea y tabulador

  ```
  print("My father's house")
  print('My father\'s house') # se escapa la '
  print('Primera línea\nSegunda línea')
  print('\tCon sangría')
  ```

  ```
  My father's house
  My father's house
  Primera línea
  Segunda línea
  Con sangría
  ```

– También existen las cadenas multi-linea para lo cual se usa triple comilla simple o doble

```
s = '''Esta es una
cadena de varias líneas
útil para crear formatos de texto
o documentar funciones y módulos
'''
print(s)
```

```
Esta es una
cadena de varias líneas
útil para crear formatos de texto
o documentar funciones y módulos
```

3.3 Recorrido de cadenas

- Se pueden recorrer los caracteres simplemente con un for:

```
for caracter in 'hola':
    print(caracter)
```

```
h
o
l
a
```

- O al estilo `C`

```
s = 'hola'
for i in range(len(s)):
    print(s[i])
```

```
h
o
l
a
```

3.4 Subcadenas

- Una subcadena es cualquier fragmento de una cadena, desde un solo carácter hasta la cadena completa
- La forma más directa que tiene Python para obtener subcadenas es con el formato de "rebanadas" (slices)
- Con este método cada subcadena es una nueva cadena
- Este método funciona para otras estructuras lineales (como listas)
- El formato de rebanadas requiere de dos índices (pude ser cualquier expresión entera):
  - Índice izquierdo inclusivo: posición inicial a partir de la cual se quiere obtener la subcadena
  - Índice derecho no inclusivo: posición final hasta la que se quiere incluir la subcadena, siempre se toma una posición antes

```
s = 'hola'
print(s[0:4]) # copia completa
print(s[:len(s)]) # lo mismo
print(s[:]) # otra vez lo mismo

# todos menos el primer caracter
print(s[1:])

# todos menos el último
print(s[:-1])

# las dos letras de enmedio
print(s[1:3])
```

```
hola
hola
hola
ola
hol
ol
```

3.5 Operaciones sobre cadenas

3.5.1 Longitud

- `len` es una función genérica de Python que permite determinar el número de elementos en varias estructuras de datos
- En el caso de una cadena establece el número de caracteres (sin importar su tamaño en bytes)

3.5.2 Concatenación

- Permite unir dos cadenas creando una nueva
- Se utiliza el operador `+`

```
s = 'hola ' + 'mundo'
print(s)
```

```
hola mundo
```

3.5.3 strip

- Quita caracteres de espaciadores a izquierda y derecha

```
s = '   hola    mundo    \n\n\n'.strip()
print(s)
```

```
hola    mundo
```

3.5.4 split

- Dada una cadena separadora, genera una lista de cadenas considerando el separador

```
s = 'nombre,edad,carrera,matricula'
partes = s.split(',')
print(partes)

s = 'hola-->mundo'
print(s.split('-->'))
```

```
['nombre', 'edad', 'carrera', 'matricula']
```

3.5.5 startswith

- Regresa verdadero si una cadena empieza con una subcadena

```
print('hola mundo'.startswith('hola'))
```

```
True
```

3.5.6 endswith

- Regresa verdadero si una cadena termina con cierta subcadena

```
print('hola mundo'.endswith('mundo'))
```

```
True
```

3.5.7 join

- Concatena cadenas en una lista de cadenas, usando la cadena actual como separador

```
print(','.join(['hola', 'mundo', 'mundial']))
```

```
hola,mundo,mundial
```

- Si necesitas modificar caracteres de una cadena, una forma simple y eficiente de hacerlo es primero convertir la cadena a lista, mediante la función `list` y luego de las modificaciones regresar a cadena con la función `join` usando la cadena vacía como separador

```
s = 'hola mundo'
ls = list(s)
print(ls)
ls[0] = 'H'
s = ''.join(ls)
print(s)
```

```
['h', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o']
Hola mundo
```

3.6 Plantillas de texto

- Son una forma conveniente de crear cadenas con espacios que se van a rellenar después
- Son la forma principal de escribir mensajes largos que dependen de valores de expresiones (como variables)
- Son reutilizables
- Se basan en dejar espacios reemplazables, también llamados `place-holders`
- Existen tres formas principales en Python:
  - Mediante el operador `%`
  - Usando lo función `format` de una cadena
  - Usando `cadenas f`

3.6.1 Operador %

- Las sustituciones son posicionales
- Es conveniente para pocas sustituciones
- Si se sustituye más de un valor es necesario agregar `()` en la sustitución
- Hay varios tipos de `place-holders`, `%s` es el más general

```
s = 'Pepe'
print('hola %s, cómo estas?' % s )

nombre = 'Juan'
edad = 15
print('Hola %s, tienes %s años' % (nombre, edad))

plantilla = 'Hola %s, tienes %s años'
print(plantilla % ('José', 16))
print(plantilla % ('Brenda', 21))
```

```
hola Pepe, cómo estas?
Hola Juan, tienes 15 años
Hola José, tienes 16 años
Hola Brenda, tienes 21 años
```

3.6.2 `format`

- Es una función de cadenas para crear plantillas
- Utiliza `place-holders` nombrados
- No es posicional como el operador `%`

```
plantilla = 'hola {nombre}'
print(plantilla.format(nombre='Pepe'))

plantilla = 'Hola {nombre}, tu edad es {edad}, adiós {nombre}'
print(plantilla.format(edad=18, nombre='Juanito'))

hola Pepe
Hola Juanito, tu edad es 18, adiós Juanito
```

3.6.3 `cadenas f`

- No es exactamente para crear plantillas (dado que no son reutilizables)
- Son una forma conveniente de remplazar un `place-holder` por una variable que ya se tiene
- Se crean poniendo una `f` al inicio de la cadena

```
nombre = 'Pepe'
edad = 18
print(f'Hola {nombre}, tu edad es {edad}')

Hola Pepe, tu edad es 18
```