Strings

Los strings o cadena de caracteres son secuencias ordenadas de caracteres. Existen diversa formas de definir los strings. Por ejemplo:

```
“Hola Mundo!”        # Con comillas dobles
‘Hola Mundo!’        # Con comillas simples
“””Hola Mundo!”””   # Con triple comillas dobles
‘’’Hola Mundo!’’’      # Con triple comillas simples
```

Las primeras dos formas nos permiten definir strings de una sola línea, es decir, sin saltos de línea. En la tercera y cuarta forma, en cambio, se pueden definir strings con saltos de línea. Esta es la principal diferencia entre los diversos modos de definirlos. Para saber si utilizar comillas simples o dobles existen convenciones y no hay ninguna limitante técnica. Por lo cual, dependerá del código que estés modificando, seguir las convenciones adoptadas, o si es un código propio o nuevo, adoptar una convención.

Como los strings son una secuencia podemos utilizar algunas funciones u operaciones de las secuencias con los strings. Por ejemplo, para acceder a un carácter o a un subsecuencia de caracteres del string podemos utilizar la indexación. También podemos pedirle la longitud con la función *len*. Veamos algunos ejemplos:

```
a_string = ‘Hola Mundo!’

# Acceso a caracteres del string.
a_string[0]    # Devuleve H
a_string[-1]   # Devuelve !

# Slicing de un string
a_string[:4]      # Devuelve Hola
```

```
a_string[5:9]    # Devuelve Mundo

# Longitud del string
len(a_string)     # Devuelve 11
```

Los strings son inmutables. Esto quiere decir que no se pueden modificar. Lo que sí se puede hacer es construir un nuevo string a partir de uno o más strings o hacer una copia del mismo. En caso de querer modificar un string se levantará una excepción del tipo *TypeError*. Veamos algunos ejemplos:

```
a_string = 'Hola Mundo!'

a_string[6] = 'o'     # Dará un error de tipo TypeError

new_string = a_string + '?'    # Genera el string Hola Mundo!?

new_string = a_string[:6] +'o' +  a_string[7:]    # Genera el string Hola Mondo!
```

Como se puede ver con el operador + se concatenan strings de manera que se crea un nuevo string a partir de dos strings.

Caracteres de escape

En algunas ocasiones es necesario escapar algún carácter. Por ejemplo si estoy definiendo el string con comilla simple y el contenido del string tiene una comilla simple, para que se interprete correctamente debo escapar esa comilla simple. Para escapar caracteres dentro de un string se utiliza la barra invertida (\). En la siguiente tabla mostramos las secuencia de escape más utilizadas.

Secuencia de escape	Descripción	Ejemplo	Resultado
\newline	Barra invertida y nueva línea ignorada.	print("linea1 \ linea2 \ linea3")	linea1 linea2 linea3

\\	Barra invertida	print("\\")	\
\’	Comilla simple	print('\'')	‘
\”	Comilla doble	print(“\””)	“
\n	Salto de línea (LF)	print(“Hola \n Mundo!”)	Hola Mundo!
\r	Salto de línea (CR)	print(“Hola \r Mundo!”)	Hola Mundo!
\t	Tabulador horizontal	print(“Hola \t Mundo!”)	Hola Mundo!

Strings con contenido dinámico

Por otro lado, se pueden crear strings con contenido dinámico. A continuación mostramos algunos ejemplos.

```
name = "Agustín"
"Hola %s" % name                    # Resultado: Hola Agustín
"El número es %d" % 5               # Resultado: El número es 5
"El número es %02d" % 5             # Resultado: El número es 005
"El decimal es %f" % 6.5            # Resultado: El número es 6.500000
"El decimal es %.2f" % 6.5          # Resultado: El número es 6.50
"Hola %(name)s" % {'name': name}    # Resultado: Hola Agustín
```

Esta es una de las formas heredadas del lenguaje C[1]. Luego se presentará una forma más actual utilizando la función format.

Codificación de los strings

Por último, veamos la codificación de los strings. En Python 3 todos los strings son unicode. Esto es una gran ventaja con respecto a la versión anterior de Python. Por un lado, se pueden representar todos los caracteres que necesitemos. Por otro lado, dentro del programa no tendremos errores de codificación de los strings ya que todos son unicode.

[1] https://docs.python.org/3/library/stdtypes.html#printf-style-string-formatting

Entonces, ¿Cuando tendremos que codificar los strings?

Los strings habrá que codificarlos (pasar de unicode a un byte array con una codificación determinada) o decodificarlos (pasar de un byte array con una codificación determinada a unicode) en dos momentos. Cuando ingresa un string desde afuera (desde un archivo, un web service, etc.), si el mismo no es unicode, habrá que decodificarlo a unicode. Y cuando necesitemos extraer datos de nuestro sistema, cuando los datos los necesitemos en cierta codificación, los tendremos que codificar con dicha codificación. Veamos un ejemplo de cómo codificar y decodificar los strings.

```
a_string = ’Otoño’

# Codifico el string a utf-8
coding_string = a_string.encode('utf-8')          # Respuesta: b'Oto\xc3\xb1o'

# Decodifico el byte array en utf-8
coding_string.decode('utf-8')                     # Respuesta: 'Otoño'
```