Manipulación de Cadenas

Ya sabemos cómo concatenar o replicar cadenas con los operadores + y *, pero todavía hay muchas otras cosas que podemos hacer como extraer valores parciales de una cadena, agregar o remover espacios, convertir a mayúsculas, a minúsculas o viceversa, entre otras.

Comillas Dobles

Las cadenas o strings pueden comenzar o terminar con comillas dobles al igual que como veníamos utilizando las comillas simples. Un beneficio de esto es que las comillas dobles nos permiten utilizar la comilla simple dentro de las comillas dobles.

```
>>> spam = "Esa es mi colección de DVD’s"
```

Como la cadena comienza con comilla doble, Python sabe que la comilla simple es parte de la cadena. Pero si necesitamos utilizar ambas, dobles y simples, necesitaremos hacer uso de los caracteres de escape.

Caracteres de Escape

Un carácter de escape nos permite utilizar caracteres que de otra manera serían imposibles poner en una cadena. Un carácter de escape consiste de la barra invertida \ seguida del carácter que queremos agregar a la cadena. Por ejemplo, el carácter de escape para una comilla simple es \’. Podemos usarlo dentro de una cadena que comienza y termina con comillas simples.

```
>>> spam = 'Que 'simpático' que sos.'
```

Carácter de escape	Imprime…
\'	Comilla simple
\"	Comilla Doble
\t	Tabulación
\n	Nueva línea
\\	Barra invertida

```
>>> print("Buenas tardes!\nCómo estás?\nPudiste ver el Director\'s Cut?")
```

Raw String o Cadena sin procesar

Se puede agregar una r antes de la comilla de una cadena para convertirla en una Raw String o cadena sin procesar. Esto hace que se ignoren por completo todos los caracteres de escape y se impriman las barras invertidas.

```
>>> print(r'Esos son los CD\'s de Claudio.')
```

Este tipo de cadenas son útiles cuando necesitamos utilizar cadenas que contengan muchas barras invertidas.

Cadena de línea múltiple con triple comillas

Hasta ahora venimos utilizando \n para crear nuevas líneas, pero a veces nuestro código es más fácil de leer si utilizamos cadenas de líneas múltiples. Una cadena multilínea en Python se escribe utilizando tres comillas simples o tres comillas dobles. Cualquier comilla, tabulación, o líneas nuevas dentro de esta cadena se las considerará parte de la misma. La indentaciones de Python se ignoran en cadenas multilínea.

Ejemplo:

```
print('''Querida Alicia,

El gato de Eva ha sido arrestado por secuestro, robo y extorsión.

Sinceramente,
Roberto''')
```

El equivalente sin triple comilla sería:

```
print('Querida Alicia,\n\nEl gato de Eva ha sido arrestado por secuestro, robo y extorsión.\n\nSinceramente,\nRoberto')
```

Comentarios multilínea

Nuestros comentarios que escribimos con # también pueden abarcar múltiples líneas si utilizamos tres comillas dobles.

```
"""Este es un programa de prueba.
Escrito en Buenos Aires, Argentina. 2020.

Este programa fue diseñado para Python 3.
"""
```

Índices y Rangos de cadena

Podemos utilizar índices y rangos con cadenas de la misma manera que con listas. La cadena ‘Hola mundo!’ funciona como una lista donde cada carácter es un elemento de una lista con su índice correspondiendo a su posición en la cadena.

```
>>> spam = 'Hola mundo!'
>>> spam[0]
>>> spam[4]
>>> spam[-1]
>>> spam[0:5]
>>> spam[:5]
>>> spam[6:]
>>> spam = 'Hola mundo!'
>>> espuma = spam[0:5]
```

Operador in y not in con cadenas

Podemos utilizar los operadores in y not in con cadenas igual que como con listas. La expresión será evaluada a un valor booleano True o False.

```
>>> 'Hola' in 'Hola Mundo'
>>> 'Hola' in 'Hola'
>>> 'HOLA' in 'Hola Mundo'
>>> '' in 'spam'
>>> 'gatos' not in 'gatos y perros'
```

Nota: estos operadores son case-sensitive.

Cadenas dentro de cadenas

```
>>> nombre = 'Alberto'
>>> edad = 4000
>>> 'Hola, mi nombre es ' + nombre + '. Tengo ' + str(edad) + 'años.'
```

También se puede utilizar interpolación de cadenas. Lo cual tiene la ventaja de no necesitar la función str() para convertir las variables a string.

```
>>> 'Mi nombre es %s. Tengo %s años.' % (nombre, edad)
```

A partir de Python 3.6 existen las f strings, que nos permiten utilizar llaves en vez de %s y poner las variables directamente dentro de ellos.

```
>>> f'Mi nombre es {nombre}. El año que viene tendré {edad + 1}.'
```

Métodos de cadenas

upper(), lower(), isupper(), islower()

Los métodos upper() y lower() devuelven una nueva con todos sus caracteres en mayúscula o minúscula, respectivamente.

```
>>> spam = 'Hola mundo!'
>>> spam = spam.upper()
>>> spam = spam.lower()
```

Estos métodos pueden ser útiles cuando necesitamos hacer una comparación no case-sensitive.

```
print('Cómo estás?')
meSiento = input()
if meSiento.lower() == 'bien':
    print('Yo me siento genial también!')
else:
    print('Espero que el resto de tu día mejore.')
```

Los métodos isupper() y islower() nuevamente reciben una cadena y devuelven un booleano True o False dependiendo de si todos los caracteres de la cadena están en mayúscula o en minúscula, respectivamente.

```
>>> spam = 'Hola mundo!'
>>> spam.islower()
```

```
>>> spam.isupper()
>>> 'HOLA'.isupper()
>>> 'abc12345'.islower()
>>> '12345'.islower()
>>> '12345'.isupper()
```
Todo junto:
```
>>> 'Hola'.upper()
>>> 'Hola'.upper().lower()
>>> 'Hola'.upper().lower().upper()
>>> 'HOLA'.lower()
>>> 'HOLA'.lower().islower()
```
Métodos isX()

A parte de los métodos islower() y isupper(), existen varios métodos más que comienzan con is. Todos estos métodos devuelven un booleano para describir la cadena que les pasamos.

- isalpha() devuelve True si la cadena contiene solo letras.
- isalnum() devuelve True si la cadena contiene solo letras y números.
- isdecimal() devuelve True si la cadena contiene solo números.
- isspace() devuelve True si la cadena contiene solo espacios.
- istitle() devuelve True si la cadena contiene solo palabras que comienzan con mayúcula.

```
>>> 'hola'.isalpha()
>>> 'hola123'.isalpha()
>>> 'hola123'.isalnum()
>>> 'hola'.isalnum()
>>> '123'.isdecimal()
>>> '    '.isspace()
>>> 'Este Es Un Título'.istitle()
>>> 'Este Es Un Título 123'.istitle()
>>> 'Este no Es Un Título '.istitle()
>>> ' Este TAMPOCO Es Un Título '.istitle()
```

Todos estos métodos son útiles a la hora de validar el input/entrada del usuario.

Ejemplos:

```
while True:
   print('Ingresar edad:')
   edad = input()
   if edad.isdecimal():
      break
   print('Por favor ingrese un número para la edad.')
while True:
   print('Ingrese una contraseña (letras y números solamente):')
   password = input()
   if password.isalnum():
      break
   print('La contraseña solo puede tener letras y números')
```

Métodos startswith(), endswith() y find()

Los métodos startswith() u endswith() devuelven un booleano True o False si la cadena empieza o termina, respectivamente, con el argumento que le pasamos.

```
>>> 'Hola mundo!'.startswith('Hola')
>>> 'Hola mundo!'.endswith('mundo!')
>>> 'abc123'.startswith('abcdef')
>>> 'abc123'.endswith('12')
>>> 'Hola mundo!'.startswith('Hola mundo!')
>>> 'Hola mundo!'.endswith('Hola mundo!')
```

El método find() funciona de manera similar, nos permite buscar en un rango de la cadena en particular y devolverá -1 si no lo encuentra o la posición donde se encontró la cadena.

```
>>> 'abc123'.find('abc')
>>> 'abc123'.find('abc',0,-1)
>>> 'abc123'.find('bc',1,3)
>>> 'abc123'.find('123',3,-1)
```

Métodos join() y split()

Los métodos join() y split() nos sirven a la hora de unir listas de cadenas. El método join() se llama en una cadena y toma como argumento una lista con valores string que se concatenaran con la cadena separándolos. Por el otro lado el método split() devuelve una lista de cadenas donde cada valor serán las palabras de la cadena, por defecto los espacios en blanco se utilizan para delimitarlas.

```
>>> ', '.join(['gatos', 'ratas', 'elefantes'])
>>> ' '.join(['Mi', 'nombre', 'es', 'Simon'])
>>> 'ABC'.join(['Mi', 'nombre', 'es', 'Simon'])
>>> 'Mi nombre es Simon'.split()
```

Si le pasamos un argumento a split() podemos definir otros caracteres para delimitar nuestros elementos.

```
>>> 'MiABCnombreABCesABCSimon'.split('ABC')
>>> 'Mi nombre es Simon'.split('m')
```

Justificación de texto con rjust(), ljust() y center()

Los métodos rjust() y ljust() devuelven una cadena con una cantidad de espacios agregados a la derecha o izquierda, respectivamente, para que contándolos junto a los caracteres de la cadena den el número que recibido en el argumento.

```
>>> 'Hola'.rjust(10)
>>> 'Hola'.rjust(20)
>>> 'Hola Mundo'.rjust(20)
>>> 'Hola'.ljust(10)
```

Existe un segundo argumento opcional para definir si queremos reemplazar los espacios con otro carácter.

```
>>> 'Hola'.rjust(20, '*')
```

```
>>> 'Hola'.ljust(20, '-')
```

El método center() es una combinación de ljust() y rjust().

```
>>> 'Hola'.center(20)
>>> 'Hola'.center(20, '=')
```

```
def imprimirPicnic(elementosDict, anchoIzq, anchoDer):
   print('Elementos del Picnic'.center(anchoIzq + anchoDer, '-'))
   for k, v in elementosDict.items():
      print(k.ljust(anchoIzq, '.') + str(v).rjust(anchoDer))
elementosPicnic = {'sanguches': 4, 'manzanas': 12, 'vasos': 4, 'galletitas': 8000}
imprimirPicnic(elementosPicnic, 12, 5)
imprimirPicnic(elementosPicnic, 20, 6)
```

Remover espacios en blanco con strip(), rstrip() y lstrip()

A veces necesitamos borrar los espacios vacíos alrededor de una cadena, para esto se puede utilizar el método strip(), o rstrip() y lstrip() para borrar los espacios de un solo lado.

```
>>> spam = '    Hola Mundo     '
>>> spam.strip()
>>> spam.lstrip()
>>> spam.rstrip()
```

También podemos cambiar qué carácter borrar en vez de los espacios en blanco.

```
>>> spam = 'SpamSpamCerdoSpamHuevosSpamSpam'
>>> spam.strip('ampS')
```

Nota: el orden de los caracteres no importa en este caso.

Copiando y Pegando cadenas con Pyperclip

El módulo pyperclip contiene las funciones copy() y paste() que pueden recibir o enviar texto del portapapeles de nuestra computadora.

Este módulo no viene incluido con Python. Para instalarlo habrá que ejecutar el comando pip install pyperclip en el símbolo de sistema.

```
>>> import pyperclip
>>> pyperclip.copy('Hola mundo!')
>>> pyperclip.paste()
>>> pyperclip.paste()
```