El futuro digital es de todos

MinTIC

Cadenas de string - continuación

OPERADO POR:

Universidad Tecnológica de Pereira

Mision TIC 2022

RUTA DE APRENDIZAJE 1

Comparación de string:

Los operadores de comparación trabajan con strings. Para ver si dos cuerdas son iguales:

```
palabra = 'banana'
if palabra == 'banana':
    print('Está bien, bananas')
```

```
Está bien, bananas
```

Así como trabajamos con los operadores lógicos anteriormente para diferentes tipos de comparaciones, podemos hacer este procedimiento para comparar cadenas de texto.

Otras operaciones de comparación son útiles para poner las palabras en orden alfabético:

```
if palabra < 'banana':
    print('Tu palabra,' + palabra + ', viene antes de banana')
elif palabra > 'banana':
    print('Su palabra,' + palabra + ', viene después de banana.')
else:
    print('Está bien, su palabra es banana')
```

```
Está bien, su palabra es banana
```

Python no maneja las mayúsculas y minúsculas de la misma manera como lo hace la gente. Todas las mayúsculas vienen antes que las minúsculas, así que:
La palabra, **Piña**, viene antes que **banana**.

Una forma común de abordar este problema es convertir las cadenas a un formato estándar, como todas las minúsculas, antes de realizar la comparación. Tengan eso en cuenta en caso de que tienes que defenderte de un hombre armado con una piña.

Métodos de un string

Los string son un ejemplo de los objetos de Python. Un objeto contiene tanto datos (la cadena propiamente dicha) como métodos, que son en realidad funciones que están incorporadas en el objeto y que están disponibles para cualquier instancia del mismo.

Python tiene una función llamada dir que enumera los métodos disponibles para un objeto. La función type muestra el tipo de un objeto y la función dir muestra los métodos disponibles.

```
In [51]: Cadena = 'Hola mundo'

In [52]: type(Cadena)
Out[52]: str
```

Si bien la función **dir** enumera los métodos, y se puede utilizar la ayuda para obtener una documentación sencilla sobre un método, una mejor fuente de documentación para los métodos de cuerda seria: **https://docs.python.org/library/stdtypes.html#string-methods.**

Llamar a un método es similar a llamar a una función (toma argumentos y devuelve un valor) pero la sintaxis es diferente. Llamamos a un método añadiendo el nombre del método al nombre de la variable utilizando el punto como delimitador.

Por ejemplo, el método upper toma un string y devuelve un nuevo string con todas las letras mayúsculas:

En lugar de la función sintaxis upper(palabra), utiliza el método sintaxis palabra.upper().

```
In [56]: palabra = 'banana'
         palabra_nueva = palabra.upper()
         print(palabra_nueva)

         BANANA
```

Esta forma de notación puntual especifica el nombre del método, **upper**, y el nombre de la cadena a la que se aplica el método, **palabra**. Los paréntesis vacíos indican que este método no toma ningún argumento.
La llamada a un método se llama una **invocation**; en este caso, diríamos que estamos invocando upper sobre **palabra**.

Por ejemplo, hay un método de cadena llamado **find** que busca la posición de una cadena dentro de otra:

```
palabra = 'banana'
index = palabra.find('a')
print(index)
```

```
1
```

En este ejemplo, invocamos a find sobre palabra y pasamos la letra que buscamos como parámetro. El método find puede encontrar subcadenas así como caracteres:

```
In [58]: palabra.find('na')
```

```
Out[58]: 2
```

Una tarea común es eliminar los espacios en blanco (espacios, tabulaciones o nuevas líneas) del principio y el final de una cadena utilizando el método de la tira:

```
In [61]: linea = '        Aquí vamos
         linea.strip()

Out[61]: 'Aquí vamos'
```

Algunos métodos como el de empezar con valores booleanos de retorno.

```
In [63]: linea = 'Que Tengas Un Buen Día'
         linea.startswith('Que')

Out[63]: True
```

```
In [63]: linea = 'Que Tengas Un Buen Día'
         linea.startswith('Que')

Out[63]: True

In [64]: linea.startswith('q')

Out[64]: False
```

Notarán que el comienzo requiere que el caso coincida, así que a veces tomamos una línea y lo mapeamos todo a minúsculas antes de hacer cualquier verificación usando el método de abajo.

```
In [65]: linea = 'Que Tengas Un Buen Día'
         linea.startswith('t')

Out[65]: False

In [66]: linea.lower() #que tendas un buen dia

Out[66]: 'que tengas un buen día'

In [69]: linea.lower().startswith('q')

Out[69]: True
```

Analizar los string

A menudo, queremos mirar en una cadena y encontrar una subcadena. Por ejemplo, si estuviéramos presentó una serie de líneas con el siguiente formato:

De stephen.marquard@ uct.ac.za Sat Jan 5 09:14:16 2008

y queríamos sacar sólo la segunda mitad de la dirección (es decir, uct.ac.za) de cada línea, podemos hacer esto usando el método **find** y corte de cadenas.

Primero, encontraremos la palabra que se encuentra en la posición anterior. Luego encontraremos la posición del primer espacio de la palabra que se encuentra en la posición anterior Y luego usaremos el corte de la cadena para extraer la porción de la palabra que estamos buscando.

```
In [72]: data = 'De stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
         enlaposicion = data.find('@')
         print(enlaposicion)

         19

In [73]: espacioenlaposicion = data.find(' ',enlaposicion)
         print(espacioenlaposicion)

         29

In [74]: host = data[enlaposicion+1:espacioenlaposicion]
         print(host)

         uct.ac.za
```

Utilizamos una versión del método de búsqueda que nos permite especificar una posición en la cadena donde queremos encontrar para empezar a buscar. Cuando cortamos, extraemos la personajes desde "uno más allá de la señal de **"@"** hasta, pero sin incluir, el espacio del **caracter**".

Operador de formato

El operador de formato, % nos permite construir cadenas, reemplazando partes de las cadenas con los datos almacenados en las variables. Cuando se aplica a números enteros, % es el operador de módulo. Pero cuando el primer operando es una cadena, % es el operador de formato.

El primer operando es la cadena de formato, que contiene una o más secuencias de formato que especifican cómo se formatea el segundo operando. El resultado es una cadena.

Por ejemplo, la secuencia de formato %d significa que el segundo operando debe formatearse como un número entero ("d" significa "decimal"):

```
In [99]: camellos = 42
         '%d' % camellos

Out[99]: '42'
```

El resultado es la cadena "42", que no debe confundirse con el valor entero 42.

Una secuencia de formato puede aparecer en cualquier parte de la cadena, de modo que se puede incrustar un valor en una frase:

```
In [100]: camellos = 42
          'He visto %d camellos' % camellos

Out[100]: 'He visto 42 camellos'
```

Caracteres especiales en el string

Como en otros lenguajes, Python tiene caracteres especiales que interpreta dentro de los string. Por ejemplo \n es un retorno de carro (nueva línea), \t es un tabulador. En string and bytes literals tienes una tabla completa con todos los caracteres de escape de los string de Python.

Podemos, al definir una cadena, usar una r justo delante de la apertura de las comillas. Esto evitará que Python interprete los caracteres especiales y los tratará como caracteres normales.

```
In [101]: # Saldrán dos líneas en pantalla, una con Hola, la otra con Mundo
          cadena = "Hola\nMundo"
          print(cadena)

          Hola
          Mundo
```

```
In [102]: # Saldrá una única línea en pantalla y la \ y la n será visibles como caracteres normales.
          cadena2 = r"Hola\nMundo"
          print(cadena2)

          Hola\nMundo
```

Contar número de veces que el substring aparece en el string : count

El método count nos dice cuántas veces aparece el substring dentro del string. Admite un parámetro que es el substring a buscar, y opcionalmente otros dos, los índices de inicio y fin de búsqueda. Veamos un ejemplo:

In [103]:
```
cadena = "un uno, un dos, un tres"

print (cadena.count("un"))         # Saca 4, hay 4 "un" en cadena.
print (cadena.count("un",10))      # Saca 1, hay 1 "un" a partir de la posición 10 de cadena.
print (cadena.count("un",0,10))    # Saca 3, hay 3 "un" entre la posición 0 y la 10.
```

```
4
1
3
```

Reemplazar substring en string : replace

El método replace nos permite obtener una copia de la cadena original (no reemplaza en la cadena original) en la que se reemplaza el substring que se le indique por uno nuevo. Admite tres parámetros, el primero es el substring que tiene que buscar, el segundo es el nuevo substring que tiene que poner y el tercero, opcional, es cuántas veces tiene que hacer el reemplazo. Veamos ejemplos

In [104]:
```
cadena = "un uno, un dos, un tres"

print (cadena.replace("un", "XXX"))        # saca por pantalla "XXX XXXo, XXX dos, XXX tres"
print (cadena.replace("un", "XXX", 2))     # Sólo reemplaza 2 "un", así que saca por pantalla "XXX XXXo, un dos, un tres"
```

```
XXX XXXo, XXX dos, XXX tres
XXX XXXo, un dos, un tres
```

El método format() de string:

Se pueden definir variables y poner llaves {} en el string donde van a ir los números o caracteres. Entre paréntesis pasamos los valores. Unos ejemplos sencillos

In [105]:

```
# saca "El valor es 12
print ("El valor es {}".format(12))

# saca "El valor es 12.3456
print ("El valor es {}".format(12.3456))

# Tres conjuntos {}, el primero para el primer parámetro de format(), el segundo para el segundo
# y así sucesivamente.
# saca "Los valores son 1, 2 y 3"
print ("Los valores son {}, {} y {}".format(1,2,3))

# Entre las llaves podemos poner la posición del parámetro. {2} es el tercer parámetro de format()
# {0} es el primer parámetro de format.
# saca "Los valores son 3, 2 y 1"
print ("Los valores son {2}, {1} y {0}".format(1,2,3))
```

```
El valor es 12
El valor es 12.3456
Los valores son 1, 2 y 3
Los valores son 3, 2 y 1
```

Concatenar

Este término significa juntar cadenas de caracteres. El proceso de concatenación se realiza mediante el operador de suma (+). Ten en cuenta que debes marcar explícitamente dónde quieres los espacios en blanco y colocarlos entre comillas.
En este ejemplo, la cadena de caracteres “mensaje1” tiene el contenido “Hola Mundo”

```
In [107]: mensaje1 = 'Hola' + ' ' + 'Mundo'
          print(mensaje1)

          Hola Mundo
```

Multiplicar

Si quieres varias copias de una cadena de caracteres utiliza el operador de multiplicación (*). En este ejemplo, la cadena de caracteres mensaje2a lleva el contenido “Hola” tres veces, mientras que la cadena de caracteres mensaje2b tiene el contenido “Mundo”. Ordenemos imprimir las dos cadenas.

In [108]:
```
mensaje2a = 'Hola ' * 3
mensaje2b = 'Mundo'
print(mensaje2a + mensaje2b)
```

```
Hola Hola Hola Mundo
```

Añadir

¿Qué pasa si quieres añadir material de manera sucesiva al final de una cadena de caracteres? El operador especial para ello es compuesto (+=).

```
In [109]: mensaje3 = 'Hola'
          mensaje3 += ' '
          mensaje3 += 'Mundo'
          print(mensaje3)
```

```
Hola Mundo
```

Con estas estructuras tenemos listas las bases del manejo de cadenas de **string**, y con esto se abre un nuevo mundo para nosotros en el manejo de información.

Con estos manejos básicos podemos comenzar a utilizar nuestra información del problema de una forma diferente, entramos en el mundo de las **bases de datos**. Con las bases de datos podremos trabajar de manera mas eficiente y encontrar o utilizar la información de nuestro problema de forma rápida. Y con esto nuestro código también es mas eficiente.

En la próxima semana revisaremos como iniciar este proceso de base de datos en **Python**.

El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022