Como trabajar con formatos de cadenas de texto en Python 3

Hemos escrito y referenciado -bastante- sobre cómo trabajar bajo la linea de comandos en GNU/Linux y para trabajar con Python de esta manera muchas veces debemos presentar los datos y resultados de una manera agradable a la vista. Para ello echaremos mano de unos comando que vienen por defecto en Python 3.5.2 , la versión que usamos a la fecha. Hagamos pues, ¡manos a la obra!

Introducción.

En el lenguaje de programación Python (y esto es un brevísimo repaso) las cadenas de texto deben ser contenidas o encerradas entre comillas simples o comillas dobles, con apertura y cierre correspondiente y por pares, y dichas comillas **no serán mostradas por pantalla al ordenar imprimirlas,** veamos el ejemplo:

```
'¡Hola! ¿cómo están?'  "Nosotros bien, ¿y ustedes?"
```

Hasta aquí no necesita mayor explicación, ustedes decidirán cuáles pares de comillas utilizar, a su gusto y elección pero eso sí, tratemos de ser constantes en su uso, si son simples o dobles. Para este tutorial utilizaremos comillas simples y nuestra balanza se inclina hacia allá porque al uno introducir una cadena de texto entrecomillada doble y presionar intro en el **shell** de Python por pantalla nos mostrará la misma cadena pero con comilla simples, *un detalle a observar.* Pero para poder mostrar estas cadenas de caracteres en un guión o **script** debemos utilizar el comando **print()** y dentro del par de paréntesis introduciremos dichas cadenas de texto:

```
print('¡Hola! ¿cómo están?')  print("Nosotros bien, ¿y ustedes?")
```

Y el resultado por pantalla será la cadena de texto sin comilla alguna.

Concatenado de cadenas de texto.

Si queremos imprimir en una sola línea de texto dos cadenas, debemos usar el símbolo de suma "+" pero en este caso simplemente nos muestra la primera cadena y luego la segunda (podemos incluir en las mismas espacios para separar las palabras):

```
print('Simón ' + 'José ' + 'Antonio ' + 'de la Santísima Trinidad ' + 'Bo
lívar y Palacios')
```

Ahora analizamos que aunque usemos el símbolo de la suma *no quiere decir que podemos pasar cualquier número para que sea interpretado como texto,* el siguiente comando producirá un error y el segundo es el que debemos utilizar:

```
print('Simón Bolívar nació en el año ' + 1783)  print('Simón Bolívar nació en el año ' + '1783')
```

Nótese que hemos encerrado el número entre comillas para que sea considerado una cadena de texto, en el lenguaje Python nos atreveríamos a decir que los datos deber ser explícitos, el lenguaje es estricto en su manipulación (si son caracteres, caracteres serán siempre, si son números igualmente, más adelante profundizaremos en esto).

Repetición de cadenas de texto.

Muchas veces necesitamos repetir uno o más caracteres varias veces, especialmente con propósitos decorativos. Esto es importante en el caso de las licencias de software donde se muchas veces se utiliza el signo de igualdad '=' para encerrar los títulos de la misma. Así como usamos el símbolo de suma ahora utilizaremos el signo de multiplición en programación que es el asteristo '*' (la equis la utilizamos para los humanos, como por ejemplo para representar las tablas de multiplicación). Para ello encerramos entre comillas el texto deseado y seguido de un asterisco y luego un número que representa las veces que repetiremos el caracter o caracteres:

```
print('='*40)  print('+-'*40)
```

Almacenando cadenas de texto en variables.

Pues bien, si en nuestro programa vamos a repetir muchas veces una frase pues lo mejor es almacenarla en una variable para mostrarla donde la necesitemos y escribirla una sola vez al principio del software que desarrollemos:

```
gracias = '¡Muchas gracias por su colaboración!'
```

Y una vez la tengamos declarada podemos imprimirla con el famoso comando **print()** -en Python 3 los paréntesis son obligatorios-:

```
print(gracias)
```

Reutilizando el último resultado.

Gracias al sr. Juan J. Merelo Guervós quien amablemente publica por Twitter una presentación sobre Python, nos enteramos del uso de el guion bajo "_" para utilizar el último valor que hayamos presentado por la consola interactiva de Python *teneindo cuidado de darle el tratamiento adecuado al valor: si es texto, número, etcétera* y valga para que aplique a las combinaciones posibles, echad un ojo al siguiente ejemplo:

Comillas y apóstrofos.

Sí, así como lo leen APÓSTROFOS y no , es común caer en este error, pero ese es el nombre correcto: apóstrofos (tal vez influye en errar el como se escribe en inglés: "apostrophes"). Lo cierto del caso es que en castellano no tenemos mayor problema en la escritura, más en el habla coloquial vaya sí que usamos la "contractura" de la palabra (elisión le dice la Real Academia Española), un caso muy común: "para allá" lo pronunciamos "pa'llá". Como véis igual hacemos en Python, solo que esta vez rompemos la regla de utilizar comillas simples porque de lo contrario se produce un error si escribimos **'pa'llá'** pues la comilla simple en el medio de la frase indica que allí termina la cadena de texto (si queréis ver algo más avanzado, por favor leed nuestro otro tutorial sobre el tema). Algunos nombres que vienen del idioma portugués también utilizan el apóstrofo, como por ejemplo el apellido "D'acosta", y si lidiamos con eso debemos usar distintos tipos de comillas, tengamos precaución con esto al momento de programar.

Si necesitamos mostrar las comillas en sí mismas pues las encerramos debidamente de forma anidada, ejemplo:

```
print('Sí, eso fue lo que ella dijo:"Hacia el sur"')
```

Líneas múltiples.

Si pretendemos mostrar varias líneas podemos almacenar dichas líneas encerrandolas entre comillas triples (sencillas o dobles, según necesitemos o queramos):

```
agua = '''El agua está compuesta de:  -Oxígeno  -Hidrógeno'''  print(agua
)
```

Caracteres de escape.

Con el caracter de escape barra invertida "\" podemos escribir nuestro apóstrofo encerrado entre comillas simples de la siguiente manera:

```
print('El ciudadano Pedro D\'acosta.')
```

Como pueden observar le indicamos a Python que tome el caracter que está a la derecha de la barra invertida como caracter en sí mismo y que no lo tome como un caracter especial para el lenguaje (en este caso como delimitador de cadena de texto).

También podemos usar dicho caracter de escape con la comilla doble:

```
print("Vendedme un tubo de 2\" de diámetro.")  # El siguiente comando tam
bién hará el mismo trabajo:  print('Vendedme un tubo de 2" de diámetro.')
```

En Python si una línea comienza con el símbolo numeral "#", todo lo que esté hacia la derecha hasta el retorno de carro será considerado comentario -sirve para documentar nuestro código para futuras generaciones-.

Además, si el caracter que le agregamos a la derecha NO es un caracter especial (por ejemplo, la letra "n") nos permitirá "quebrar" o insertar un retorno de carro en una sola línea:

```
print('Línea 1\nLínea 2')
```

Y con la barra invertida y la letra "t" podremos presentar por pantalla la composición química del agua del ejemplo anterior pero numerada y tabulada:

```
agua = 'El agua está compuesta de:\n1\tátomo de Oxígeno\n2\tátomos de Hid
rógeno'  print(agua)
```

Y para imprimir la barra invertida...

Y si necesitamos imprimir la barra invertida por pantalla con el comando print() debemos

anteceder la letra "r" justo antes de la cadena de texto:

```
print(r'El agua está compuesta de:\n1\tátomo de Oxígeno\n2\tátomos de Hidrógeno')
```

Formateando cadenas de texto con marcadores de posición.

Así como utilizamos el comando **print()** en este caso utilizaremos otro comando intrínseco en Python: **format()**. En lenguaje Python podemos "heredar" o "unir" a una variable una función -es por ello que dicen y comentan que es un lenguaje de programación avanzado-: por medio del punto "." unimos primero la variable y luego la función **format()** auxilíandonos con un par de corchetes "{}" -marcador de posición- que le indicará a dicha función dónde colocar el valor que necesitemos. Vayamos de la teoría a la práctica:

```
print('Venezuela tiene {} estados.'.format(24))
```

Y veremos por pantalla lo siguiente:

```
Venezuela tiene 24 estados.
```

Tal como lo presentamos es poco útil en verdad, pero vayamos un poco más allá: en Venezuela el ahora estado Amazonas antes era un territorio federal, así que el número de estados a lo largo del tiempo será algo que puede variar. Asimismo muchos países están divididos por estados (otros por comunas como Chile, otros por cantones como Suiza, etcétera). Veamos el siguiente código:

```
pais = "Venezuela"  estados = 24  print(pais + ' tiene {} estados.'.format(estados))
```

Ahora tiene más sentido ya que dichas variables 'pais' -sin acento, así se acostumbra escribir los nombres de variables- y 'estados' podrían ser llenados, por ejemplo, desde los valores almacenados en una base de datos con varios países y sus respectivas cantidades de estados y podremos hacer un ciclo para mostrarlos a todos. Lo que incluiremos entre corchetes no necesariamente debe ser un número, también puede ser otra cadena de texto como veremos a continuación.

Formateando cadenas de texto *con varios* marcadores de posición.

Continuando con el ejemplo anterior, es posible reescribirlo de la siguiente manera:

```
pais = "Venezuela"  estados = 24  print('{} tiene {} estados.'.format(pais, estados))
```

Siempre hay que tener en cuenta el colocar las variables a insertar en el mismo orden que van de izquierda a derecha según los marcadores de posición, esto es así porque dentro de los corchetes no hemos insertado aún órdenes especiales; *por lo tanto lo siguiente NO mostrará correctamente la información:*

```
pais = ''Venezuela'  estados = 24  print('{} tiene {} estados.'.format(estados, pais))
```

Para corregir lo anterior debemos colocar el número correspondiente (comenzando desde cero) a la enumeración de elementos -valga la redundancia-. Es decir que *lo que insertemos dentro de los paréntesis del comando* ***format*** *-separados por comas- automáticamente tomará el primer elemento el índice cero, el segundo, uno y así sucesivamente:*

```
pais = "Venezuela"  estados = 24  print('{1} tiene {0} estados.'.format(estados, pais))
```

Podremos escribir muchos más elementos que la cantidad de marcadores de posición -corchetes {}- ***pero no menos, porque produce un error; veamos el próximo modelo:***

```
pais = "Venezuela"  print('{} tiene {} estados.'.format(pais))
```

Allí tenemos dos marcadores de posición pero una sola variable (podemos complicar el ejercicio con muchos más elementos, pero trabajar con 2 ó 3 elementos ilustra bien lo que queremos enseñar). Igualmente, si a los marcadores de posición le insertamos índices que no existen, pues también nos arroja error:

```
pais = "Venezuela"  estados = 24  print('{1} tiene {2} estados.'.format(e
```

```
stados, pais))
```

Tal como les indicamos, se debe enumerar desde cero, es así que el elemento 2 no existe ni está declarada dicha variable (está fuera de rango, produce error).

Si se nos hace difícil el trabajar con base cero, también Python nos permite colocarles nombres clave a las variables *abramos nuestra mente a los llamados alias:*

```
pais = "Venezuela"  estados = 24  print('{uno} tiene {dos} estados.'.form
at(uno = estados, dos = pais))
```

Si analizamos bien, en realidad estamos llamando a la variable "pais" con el alias "uno" y "estados" con el alias "dos", lo cual nos muestra lo poderoso que es el lenguaje Python.

Pasando los formatos específicos.

Hasta ahora hemos utilizado los marcadores de posición vacios o con un índice o alias pero podemos ir más allá. Por medio de los dos puntos ":" nos permite separar el índice o alias del formato con que queramos presentar las variables. Para ilustrar imaginemos que tenemos una lista de países pero sus nombres, desde luego, no tienen el mismo número de caracteres. Si queremos mostrarlos de una manera muy ordenada nos fijaremos cual tiene la cifra más larga y asi dejaremos sufiente espacio para los otros más cortos. Hagamos un sencillo listado de 3 países hartos conocidos por nosotros los venezolanos:

```
pais = ("Colombia", "Chile", "Venezuela")  division = ("departamentos", "
comunas", "estados")
```

Con estas declaraciones le estamos indicando a Python que nos haga una matriz (ellos lo llaman "tupla", una palabra no reconocida por la RAE) de 1 fila y 3 columnas para la variable "pais" e igualmente para la variable "division". Fijaos que el más largo es Venezuela con 9 letras y le sumaremos uno más para separar del resto de la oración con un espacio; haremos uso de un **ciclo for:**

```
for k in range(0,3):    print("{0:10} está dividido políticamente en {1}.
".format(pais[k], division[k]))
```

Ya os dijimos que las numeraciones arrancan desde cero, pero es conveniente aclarar que la variable "k" es evaluada por Python al inicio del ciclo, por eso debemos agregarle uno más a nuestros índices de matriz. Veremos la siguiente salida por pantalla:

```
Colombia está dividido políticamente en departamentos. Chile está d
ividido políticamente en comunas. Venezuela está dividido políticamente
en estados.
```

¡Qué práctico! Lo más interesante es que podemos alinear las variables al centro o a la derecha (por defecto alinea a la izquierda) si le colocamos el signo "^" o "" para alinear a la derecha:

```
for k in range(0,3):    print("{0:10} está dividido políticamente en {1}.
".format(pais[k], division[k]))  Colombia está dividido políticamente en
 departamentos.      Chile está dividido políticamente en comunas. Venez
uela está dividido políticamente en estados.
```

Incluso podemos presentar los nombres de los países de la misma manera que uno hace al escribir los nombres de los beneficiarios en un cheque bancario: agregando asteriscos (o cualquier otro caracter) a la izquierda y derecha y centrando el nombre y agregaremos 12 "espacios" en vez de 10, probad en vuestra terminal lo siguiente (no publicamos imágenes del resultado para "forzaros" a practicar en vuestro ordenador):

```
for k in range(0,3):    print("{0:*^12} está dividido políticamente en {1
}.".format(pais[k], division[k]))
```

Tal vez con los números hallaremos la máxima utilidad al formateo de cadenas con marcadores de posición, veamos algunos de ellos.

Pasando los formatos *numéricos* específicos.

Muchas veces necesitamos en nuestros programas las tasas de los impuestos que recauda el Estado para el buen funcionamiento de la República. En Venezuela el ente encargado es el Servicio Nacional Integrado de Administración Tributaria y Aduanera (SENIAT por sus siglas) quien legalmente está facultado a publicar y hacer cumplir las variaciones en los porcentajes. El Impuesto al Valor Agregado (IVA por sus siglas) recordamos que comenzó con un 10% en 1994 (año en que fue creado el SENIAT de la mano de la Guardia Nacional quienes nos visitaron para

cumplir y hacer cumplir el impuesto) y ha sufrido subidas y bajadas estando hoy en 12% -en España tenemos entendido que está en 25% al momento de escribir estas líneas-. Imaginemos que queremos informar esto a nuestros usuarios:

```
iva1994 = 10  iva2016 = 12  print('La tasa de impuesto del IVA en 1994 er
a de {:f}%'.format(iva1994))  print('Hoy en día, año 2016:{:f}%'.format(i
va2016))
```

Como recordaremos, en el marcador de posición debemos colocar los dos puntos para separar el índice (por defecto cero porque es un solo elemento y se puede omitir) del formato numérico que deseamos presentar. En nuestra vida común los números del 0 al 100 toman especial importancia y tanto nuestra moneda y tasas se representa hasta con dos decimales a la derecha de la coma *-mejor dicho, empleamos hasta la centésima parte de la unidad-.* Es por esto que, si habéis practicado los comandos de arriba, notaréis que se "imprime" con seis decimales porque utilizamos el parámetro **"f"** -número con separador decimal flotante- y para limitar la salida a dos decimales debemos agregar "**.2**" al formato:

```
print('La tasa de impuesto del IVA en 1994 era de {:.2f}%'.format(iva1994
))  print('Hoy en día, año 2016:{:.2f}%'.format(iva2016))
```

Ahora se ve un poco mejor, pero con el inconveniente de que nuestro separador decimal es la coma y para el mundo anglosajón es el punto (los sistemas operativos modernos lidian con esto muy bien todo el tiempo simplemente asignandole un identificador especial que *tal vez sea el Unicode 2396 "**?**" y mostrando al usuario de cada región su símbolo particular en cada salida por pantalla y/o impresora*). Nota: los parámetros en la función **format()** deben ser en estricto orden: "{0:.2f}" es lo correcto -índice, separador, número de decimales y tipo de número, "f" para flotante-; mientras que "{0:f.2}" *no funcionará.*

Usando formatos de texto para presentar datos.

Una aplicación práctica de la vida real es mostrar a nuestros párvulos las tablas de multiplicar, escojamos -¡ejem!- el número 7, nuestro número de la suerte, y hagamos un programita:

```
for k in range(1,10):    print('{} x {} = {}'.format(7, k, 7*k))
```

Como véis no representa mayor problema la salida pues el único elemento díscolo es el resultado

de "7 x 1", vamos a ampliar la tabla hasta 30 (recordad sumar uno más) y mostramos su respuesta a continuación:

```
for k in range(1,31):    print('{} x {} = {}'.format(7, k, 7*k))  7 x 1 =
 7  7 x 2 = 14  7 x 3 = 21  7 x 4 = 28  7 x 5 = 35  7 x 6 = 42  7 x 7 = 4
9  7 x 8 = 56  7 x 9 = 63  7 x 10 = 70  7 x 11 = 77  7 x 12 = 84  7 x 13
= 91  7 x 14 = 98  7 x 15 = 105  7 x 16 = 112  7 x 17 = 119  7 x 18 = 126
  7 x 19 = 133  7 x 20 = 140  7 x 21 = 147  7 x 22 = 154  7 x 23 = 161  7
 x 24 = 168  7 x 25 = 175  7 x 26 = 182  7 x 27 = 189  7 x 28 = 196  7 x
29 = 203
```

Aquí está el meollo del asunto, a medida que los números "engordan" nuestras columnas no se presentan de manera tabulada. Para observarlo más a nuestro gusto, el de los seres humanos (imagino que las computadoras cuando gobiernen al mundo obviarán el arte por completo, privando la funcionalidad por encima de todo, pero hasta que esos días lleguen no nos preocuparemos por #Skynet) necesitamos que cada número, sin importar su valor, ocupe máximo 3 espacios o casillas, si lo queremos ver de esa manera. Para ello agregamos "{:3d]" en cada marcador de posición y listamos el resultado (en este ejemplo colocamos lo que se ve en la línea de comandos de Phyton):

¡Mucho mejor para nosotros los obsesivos por el orden!

Formatos soportados por Python.

Como era de esperarse, hay una gran cantidad de parámetros en el comando **format()** y acá explicamos los hasta ahora conocidos teniendo en cuenta que hay que respetar el orden en que pasamos los parámetros, a saber son los siguientes:

1. El índice o alias de la variable que queremos formatear, generalmente se omite ya que acostumbramos a colocarlas en el orden correspondiente, cos*as de nosotros los seres humanos.*
2. Caracter separador, los dos puntos ":".
3. Caracter de relleno, cualquiera que se necesite (hicimos un ejemplo con un asterisco, el ejemplo de los cheques bancarios ¿lo recordáis?).
4. Alineación (recordad el ejemplo de los países), abajo especificamos más.
5. Signo positivo, negativo o cero (abajo aclaramos).
6. En el caso de números en otros sistemas de numeración podemos usar "#" o "0", de nuevo abajo explicamos mejor el asunto.
7. Anchura del campo, un número entero de no muchas cifras ya que el ancho de pantalla

generalmente es 80 caracteres en remembranza de los antiguos monitores monocromáticos.

8. Opción de agrupamiento: *aquí si que hay mucha tela que cortar, mirad el punto 8 que lo hacemos muy detalladamente.*
9. Precisión: en el caso de las tasas de impuesto usamos 2 decimales, pero podemos usar los que necesitemos, lo único que no explicamos es que *esta función redondea hacia arriba (si es mayor o igual a 5) o hacia abajo según el primer decimal fuera del alcance de precisión.*
10. Tipo de dato: lo más importante queda de último, ya que podemos indicarle explícitamente qué tipo de datos pasamos e incluso que conversión hacemos, leed el punto 10 en detalle.

Los puntos 1 y 2 ya los hemos explicado muy bien, veamos los siguientes.

3.-Caracter de relleno.

Cualquier caracter INCLUSO los caracteres especiales no presentan ningún problema, recordad que lo TODO lo que está en el marcador de posición "{}" Python los considera parámetros, no comandos especiales; haced la prueba colocando comillas simples, dobles, barra y barra invertida y os lo mostrará sin problema alguno.

4.-Alineación de caracteres.

- **"":** alinea a la derecha.
- **"="**: solamente para formatos numéricos que presentan signo, por ejemplo "+000000120", rara vez se utiliza.

5.-Signo numérico.

- **"+", "-" y " ":** si necesitamos colocarle signo a nuestro resultado usamos el signo de suma o de resta, *o simplemente dejamos un solo espacio en blanco reservado para el signo, si es negativo se muestra sino coloca un espacio en blanco.*

6.-Forma alterna para la conversión.

El utilizar "#" o "0" nos hes tremendamente útil si hacemos conversiones a otros sistemas de numeración (binarios, por ejemplo), mirad el punto 10.

7.-Anchura del campo.

Ya dijimos que está limitado por nuestra noció de estética pero en realidad si le pasamos un valor muy alto pues simplemente Python lo mostrará en tantas líneas como necesite, una abajo de la otra, dando al traste cualquier tipo de formato que deseemos, cuidadito con esta cifra.

8.-Opción de agrupamiento.

¿Recuerdan que hablamos del separador decimal y que en nuestro país es la coma? De hecho nosotros usamos el punto como separador de miles y esto trae muchas veces un dolor de cabeza para el mundo de la banca y negocios. Acá os mostramos cómo lo maneja Ubuntu 16.04:

Pues acá va que Python se empeña en utilizar el punto como separador decimal y la coma como separador de miles a pesar de nosotros tener nuestra configuración regional como corresponde. Es por ello que en la PEP378 le buscan una solución al asunto haciendo uso de un comando y un artilugio: cambiamos las comas "," que separan los miles en formato anglosajón por guiones bajos "_", luego cambiamos el punto "." que es separador decimal en formato anglosajón por una coma "," y finalmente cambiamos los guiones bajos "_" por puntos "." ¿complicado en idioma castellano? Veamos como se escribe en lenguaje Python:

```
monto = 1234567.89  print('Su saldo bancario es Bs.' + '{:,}'.format(mont
o).replace(",","_").replace(".", ",").replace("_",".")) 
```

El comando clave es replace(,) y nótese que hemos dejado el símbolo de "Bs." fuera del valor numérico para que ese punto no sea sustituido por una coma y nuestro trabajo se ve plasmado así:

¿Complicado? Pues aún no hemos comenzado a programar de verdad, ja ja ja ;-)

En realidad nuestros amigos que desarrollan Python de manera desinteresada (monetariamente hablando) se han compadecido de nuestro predicamento porque, al menos nos han permitido el parámetro de la coma "," como separador de miles, *de parte de nosotros está comenzar a programar nuestras propias utilerías y "subirlas" a GitHub para que estén disponibles de manera pública por medio del Software Libre.* Como ustedes ya imaginarán ya otros colegas habrán pasado por esto y en la internet habrá una solución ya realizada, **lo que hay es que buscarla** o sino hacerla nosotros mismos.

10.-Tipos de datos (y conversiones).

Pues ya va finalizando nuestra clase del día, lo último y tal vez más importante es que le podemos decir a Python el tipo de dato que le vamos a pasar para que nos lo represente adecuadamente *e incluso podemos hacer conversiones a otros sistemas de numeración,* miremos:

- **"s":** formato de cadena de texto, es el valor por omisión, eso quiere decir que es tácito y

sobre entendido que lo que le pasamos al comando format() es una cadena de texto sin más; se puede omitir.

- **"b":** nos representará un número en de numeración formato binario, *si le pasamos la coma como separador de agrupamiento lo hará de 4 en 4 caracteres.*
- **"c":** un número entero que corresponde a un caracter ASCII, así la letra "A" es el número 34, le pasamos ese número y nos dibujará una letra "A".
- **"d":** sistema de base decimal, si se lo pasamos en binario hace la conversión correspondiente.
- **"o":** sistema de numeración de base 8.
- **"x" y "X":** sistema de numeración de base 16 con los caracteres en mayúsculas o minúsculas para los valores por encima de nueve.
- **"n":** supuestamente para indicar que se utilize los separadores decimales y sepradores de miles segúna la configuración regional del sistema operativo del usuario (*lo probamos y no funciona para nada bajo Python 3.5.2, no señor,* ***la teoría es hermosa pero la práctica que hace nuestra experiencia ES MARAVILLOSA***).

https://twitter.com/ks7000/status/789549159372230656

Ejemplos de conversiones:

Fuentes consultadas:

En idioma inglés:

- "An Introduction to Working with Strings in Python 3" by Lisa Tagliaferri at Digital Ocean . com
- "How To Use String Formatters in Python 3" by Lisa Tagliaferri at Digital Ocean . com
- "Format Specification Mini-Language" at Python . org