rgrimson actualizo repo

1 contributor

Raw Blame

418 lines (314 sloc) 12.8 KB

Contenidos | Anterior (3 Números) | Próximo (5 Listas)

1.4 Cadenas

En esta sección veremos cómo trabajar con textos.

Representación de textos

Las cadenas de caracteres entre comillas se usan para representar texto en Python. En este caso, fragmentos del Martín Fierro.

```
# Comillas simples
a = 'Aquí me pongo a cantar, al compás de la vigüela'

# Comillas dobles
b = "Los hermanos sean unidos porque ésa es la ley primera"

# Comillas triples
c = '''
Yo no tengo en el amor
```

```
Quien me venga con querellas;
Como esas aves tan bellas
Que saltan de rama en rama
Yo hago en el trébol mi cama
Y me cubren las estrellas.
'''
```

Normalmente las cadenas de caracteres solo ocupan una linea. Las comillas triples nos permiten capturar todo el texto encerrado a lo largo de múltiples lineas.

No hay diferencia entre las comillas simples (') y las dobles ("). *Pero el mismo tipo de comillas que se usó para abrir debe usarse para cerrar*.

Código de escape

Los códigos de escape (escape codes) son expresiones que comienzan con una barra invertida, `\` y se usan para representar caracteres que no pueden ser fácilmente tipeados directamente con el teclado. Estos son algunos códigos de escape usuales:

```
'\n'      Avanzar una línea
'\r'      Retorno de carro
'\t'      Tabulador
'\''      Comilla literal
'\"'      Comilla doble literal
'\\'      Barra invertida literal
```

El *retorno de carro* (código '\r') mueve el cursor al comienzo de la línea pero sin avanzar una línea. El origen de su nombre está relacionado con las máquinas de escribir.

Representación en memoria de las cadenas

Las cadenas se representan en Python asociando a cada caracter un número entero o código Unicode. Es posible definir un caracter usando su código y códigos de escape como `s = '\U0001D120'` para la clave de sol.

Indexación de cadenas

Las cadenas funcionan como vectores, permitiendo el acceso a los caracteres individuales. El índice comienza a contar en cero. Los índices negativos se usan para especificar una posición respecto al final de la cadena.

```
a = 'Hello world'
b = a[0]          # 'H'
c = a[4]          # 'o'
d = a[-1]         # 'd' (fin de cadena)
```

También se puede *rebanar* (slice) o seleccionar subcadenas especificando un range de índices con `:`.

```
d = a[:5]     # 'Hello'
e = a[6:]     # 'world'
f = a[3:8]    # 'lo wo'
g = a[-5:]    # 'world'
```

El caracter que corresponde al último índice no se incluye. Si un extremo no se especifica, significa que es *desde el comienzo* o *hasta el final*, respectivamente.

Operaciones con cadenas

Concatenación, longitud, pertenencia y replicación.

```
# Concatenación (+)
a = 'Hello' + 'World'   # 'HelloWorld'
b = 'Say ' + a          # 'Say HelloWorld'

# Longitud (len)
s = 'Hello'
len(s)                  # 5

# Test de pertenencia (in, not in)
t = 'e' in s            # True
f = 'x' in s            # False
g = 'hi' not in s       # True

# Replicación (s * n)
rep = s * 5             # 'HelloHelloHelloHelloHello'
```

Métodos de las cadenas

Las cadenas en Python tienen *métodos* que realizan diversas operaciones con este tipo de datos.

Ejemplo: sacar (strip) los espacios en blanco sobrantes al inicio o al final de una cadena.

```
s = '  Hello '
t = s.strip()     # 'Hello'
```

Ejemplo: Conversión entre mayúsculas y minúsculas.

```
s = 'Hello'
l = s.lower()     # 'hello'
```

```
u = s.upper()     # 'HELLO'
```

Ejemplo: Reemplazo de texto.

```
s = 'Hello world'
t = s.replace('Hello' , 'Hallo')   # 'Hallo world'
```

Más métodos de cadenas:

Los strings (cadenas) ofrecen una amplia variedad de métodos para testear y manipular textos. Estos son algunos de los métodos:

```
s.endswith(suffix)     # Verifica si termina con el sufijo
s.find(t)              # Primera aparición de t en s
s.index(t)             # Última  aparición de t en s
s.isalpha()            # Verifica si los caracteres son alfabéticos
s.isdigit()            # Verifica si los caracteres son numéricos
s.islower()            # Verifica si los caracteres son minúsculas
s.isupper()            # Verifica si los caracteres son mayúsculas
s.join(slist)          # Une una lista de cadenas usando s como delimitado
s.lower()              # Convertir a minúsculas
s.replace(old,new)     # Reemplaza texto
s.split([delim])       # Parte la cadena en subcadenas
s.startswith(prefix)   # Verifica si comienza con un sufijo
s.strip()              # Elimina espacios en blanco al inicio o al final
s.upper()              # Convierte a mayúsculas
```

Mutabilidad de cadenas

Los strings son "inmutables" o de sólo lectura. Una vez creados, su valor no puede ser cambiado.

```
>>> s = 'Hello World'
>>> s[1] = 'a' # Intento cambiar la 'e' por una 'a'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

Esto implica que las operaciones y métodos que manipulan cadenas deben crear nuevas cadenas para almacenar su resultado.

Conversión de cadenas

Usá `str()` para convertir cualquier valor a cadena. El resultado es una cadena con el mismo contenido que hubiera producido el comando `print()` sobre la expresión entre paréntesis.

```
>>> x = 42
>>> str(x)
'42'
>>>
```

f-Strings

Las f-Strings son cadenas en las que ciertas expresiones son formateadas

```
>>> nombre = 'Naranja'
>>> cajones = 100
>>> precio = 91.1
>>> a = f'{nombre:>10s} {cajones:10d} {precio:10.2f}'
>>> a
'   Naranja        100      91.10'
>>> b = f'Costo = ${cajones*precio:0.2f}'
>>> b
'Costo = $9110.00'
>>>
```

Nota: Esto requiere Python 3.6 o uno más nuevo. El significado de los códigos lo veremos más adelante.

Ejercicios

En estos ejercicios vas a experimentar con operaciones sobre el tipo de dato string de Python. Hacelo en el intérprete interactivo para ver inmediatamente los resultados.

Recordamos:

> En los ejercicios donde interactuás con el intérprete, el símbolo `>>>` es el que usa Python para indicarte que espera un nuevo comando. Algunos comandos ocupan más de una línea de código --para que funcionen, vas a tener que apretar 'enter' algunas veces. Acordate de no copiar el `>>>` de los ejemplos.

Comencemos definiendo una cadena que contiene una lista de frutas así::

```
>>> frutas = 'Manzana,Naranja,Mandarina,Banana,Kiwi'
>>>
```

Ejercicio 1.14: Extraer caracteres individuales y subcadenas

Los strings son vectores de caracteres. Tratá de extraer algunos carateres:

```
>>> frutas[0]
?
>>> frutas[1]
?
>>> frutas[2]
?
>>> frutas[-1]        # Último caracter
?
>>> frutas[-2]        # Índices negativos se cuentan desde el final
?
>>>
```

Como ya dijimos, en Python los strings son sólo de lectura. Verificá esto tratando de cambiar el primer caracter de `frutas` por una m minúscula 'm'.

```
>>> frutas[0] = 'm'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

Ejercicio 1.15: Concatenación de cadenas

A pesar de ser sólo de lectura, siempre podés reasignar una variable a una cadena nueva. Probá el siguiente comando que concatena la palabra "Pera" al final de `frutas`:

```
>>> frutas = frutas + 'Pera'
>>> frutas
'Manzana,Naranja,Mandarina,Banana,KiwiPera'
>>>
```

Ups! No es exactamento lo que queríamos. Reparalo para que quede `'Manzana,Naranja,Mandarina,Banana,Kiwi,Pera'`.

```
>>> frutas = ?
>>> frutas
'Manzana,Naranja,Mandarina,Banana,Kiwi,Pera'
>>>
```

Agregá 'Melón'` al principio de la cadena:

```
>>> frutas = ?
>>> frutas
'Melón,Manzana,Naranja,Mandarina,Banana,Kiwi,Pera'
>>>
```

Podría parecer en estos ejemplos que la cadena original está siendo modificada, contradiciendo la regla de que las cadenas son de sólo lectura. No es así. Las operaciones sobre cadenas crean una nueva cadena cada vez. Cuando la variable `frutas` es reasignada, apunta a la cadena recientemente creada. Luego, la cadena vieja es destruída dado que ya no está siendo usada.

Ejercicio 1.16: Testeo de pertenencia (test de subcadena)

Experimentá con el operador `in` para buscar subcadenas. En el intérprete interactivo probá estas operaciones:

```
>>> 'Naranja' in frutas
?
>>> 'nana' in frutas
True
>>> 'Lima' in frutas
?
>>>
```

¿Por qué la verificación de `'nana'` dió `True` ?

Ejercicio 1.17: Iteración sobre cadenas

Usá el comando `for` para iterar sobre los caracteres de una cadena.

```
>>> cadena = "Ejemplo con for"
>>> for c in cadena:
        print('caracter:', c)
# Mirá el output.
```

Modificá el código anterior de manera que dentro del ciclo el programa cuente cuántas letras "o" hay en la cadena.

Sugerencia: usá un contador como con los meses de la hipoteca.

Ejercicio 1.18: Geringoso rústico

Usá una iteración sobre el string `cadena` para agregar la sílaba 'pa', 'pe', 'pi', 'po', o 'pu' según corresponda luego de cada vocal.

```
>>> cadena = 'Geringoso'
>>> capadepenapa = ''
>>> for c in cadena:
        ?
>>> capadepenapa
Geperipingoposopo
```

Podés probar tu código cambiando la cadena inicial por otra palabra, como 'apa' o 'boligoma'.

Guardá el código en un archivo `geringoso.py`.

Ejercicio 1.19: Métodos de cadenas

En el intérprete interactivo experimentá con algunos de los métodos de cadenas introducidos antes.

```
>>> frutas.lower()
?
>>> frutas
?
>>>
```

Recordá, las cadenas son siempre de sólo lectura. Si querés guardar el resultado de una operación, vas a necesitás asignárselo a una variable:

```
>>> lowersyms = frutas.lower()
>>>
```

Probá algunas más:

```
>>> frutas.find('Mandarina')
?
>>> frutas[13:17]
?
>>> frutas = frutas.replace('Kiwi','Melón')
>>> frutas
?
>>> nombre = '   Naranja   \n'
>>> nombre = nombre.strip()    # Remove surrounding whitespace
>>> nombre
?
>>>
```

Ejercicio 1.20: f-strings

A veces querés crear una cadena que incorpore los valores de otras variables en ella.

Para hacer eso, usá una f-string. Por ejemplo:

```
>>> nombre = 'Naranja'
>>> cajones = 100
>>> precio = 91.1
>>> f'{cajones} cajones de {nombre} a ${precio:0.2f}'
'100 cajones de Naranja a $91.10'
>>>
```

Modificá el programa `hipoteca.py` del Ejercicio 1.11 de la sección anterior para que escriba su salida usando f-strings. Tratá de hacer que la salida quede bien alineada.

Ejercicio 1.21: Expresiones regulares

Una limitación de las operaciones básicas de cadenas es que no ofrecen ningún tipo de transformación usando patrones más sofisticados. Para eso vas a tener que usar el módulo `re` de Python y aprender a usar expresiones regulares. El manejo de estas expresiones es un tema en sí mismo. A continuación presentamos un corto ejemplo:

```
>>> texto = 'Hoy es 6/8/2020. Mañana será 7/8/2020.'
>>> # Encontrar las apariciones de una fecha en el texto
>>> import re
>>> re.findall(r'\d+/\d+/\d+', texto)
['6/8/2020', '7/8/2020']
>>> # Reemplazá esas apariciones, cambiando el formato
>>> re.sub(r'(\d+)/(\d+)/(\d+)', r'\3-\2-\1', texto)
'Hoy es 2020-8-6. Mañana será 2020-8-7.'
>>>
```

Para mas información sobre el módulo `re`, mirá la documentación oficial en inglés o algún tutorial en castellano.

Commentario

A medida que empezás a usar Python es usual que quieras saber qué otras operaciones admiten los objetos con los que estás trabajando. Por ejemplo. ¿cómo podés averiguar qué operaciones se pueden hacer con una cadena?

Dependiendo de tu entorno de Python, podrás ver una lista de métodos disponibles apretando la tecla tab. Por ejemplo, intentá esto:

```
>>> s = 'hello world'
>>> s.<tecla tab>
>>>
```

Si al presionar tab no pasa nada, podés volver al viejo uso de la función `dir()`. Por ejemplo:

```
>>> s = 'hello'
>>> dir(s)
['__add__', '__class__', '__contains__', ..., 'find', 'format',
'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',
'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
>>>
```

`dir()` produce una lista con todas las operaciones que pueden aparecer luego del parámetro que le pasaste, en este caso `s`. También podés usar el comando `help()` para obtener más información sobre una operación espcífica:

```
>>> help(s.upper)
Help on built-in function upper:

upper(...)
    S.upper() -> string

    Return a copy of the string S converted to uppercase.
>>>
```

Contenidos | Anterior (3 Números) | Próximo (5 Listas)