Universidad Tecnológica de Pereira

Mision TIC 2022

El futuro digital es de todos

MinTIC

String

Operadores

Hechos QUE CONECTAN

Contenido

- String
  - Obtener la longitud de un string usando len()
  - Rebanadas de String
  - ¿Los string son inmutables?
  - El operador "in“
  - Comparación de string
  - Métodos de un string

Contenido

- Analizar los string
- Operador de formato
- Caracteres especiales en el string
- Contar número de veces que el substring aparece en el string : count
- Reemplazar substring en string : replace

Contenido

- el método format() de string
- Concatenar
- Multiplicar
- Añadir

Contenido

String

Esta lección es una rápida introducción a técnicas de manipulación de cadenas de caracteres (o strings) en Python. Saber cómo manipular cadenas de caracteres juega un papel fundamental en la mayoría de las tareas de procesamiento de texto. Si quieres experimentar con las siguientes lecciones puedes escribir y ejecutar pequeños programas tal como lo hicimos en lecciones previas, o puedes abrir tu intérprete de comandos de Python / Terminal para probarlos ahí

Contenido

Una string es una secuencia de caracteres. Puede acceder a los caracteres de uno en uno con el operador de paréntesis:

```
fruta = 'fresa’

letra = fruta[1]
print(letra)
```

La segunda afirmación extrae el carácter en la posición 1 del índice de la variable de la fruta y lo asigna a la variable de la letra. La expresión entre paréntesis se llama índice. El índice indica el carácter de la secuencia que se desea (de ahí el nombre). Pero puede que no consigas lo que esperas:

Contenido

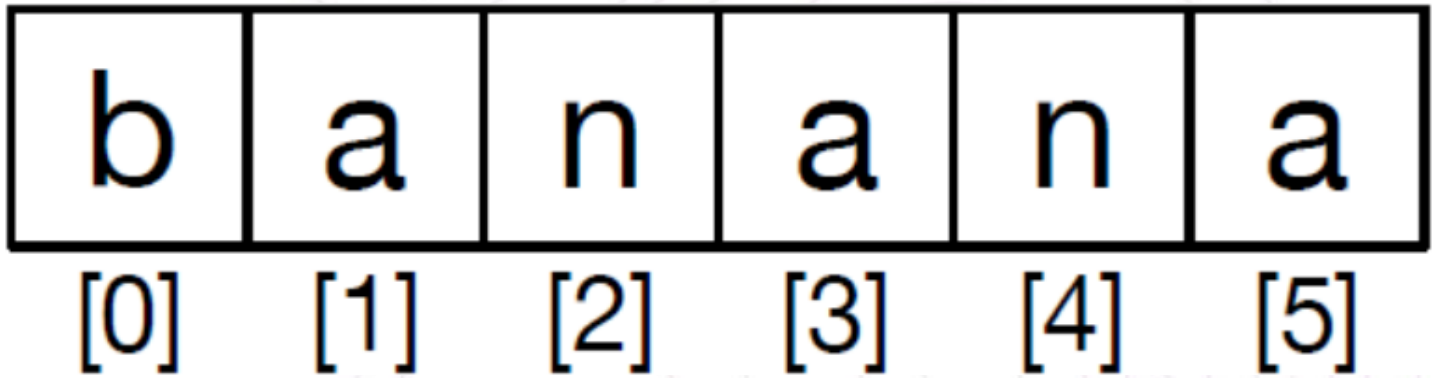
Para la mayoría de la gente, la primera letra de "banana" es "b", no "a". Pero en Python, el índice es un desplazamiento del principio de la cadena, y el desplazamiento de la primera letra es cero (0).

```
fruta = ‘banana’

letra = fruta[0]
print(letra)
```

Así que "b" es la letra en la posición 0 de la palabra "banana", "a" es la letra en la posición 1, y "n" seria la letra en la posición 2.
Se puede usar cualquier expresión, incluyendo variables y operadores, como un índice, pero el valor del índice tiene que ser un número entero. De lo contrario, se obtiene un error

Contenido



Contenido

Obtener la longitud de un string usando len()

len es una función incorporada que devuelve el número de caracteres de una cadena:

fruta = 'banana'
len(fruta)

Para obtener la última letra de un string, podrías estar tentado de intentar algo como esto:

```
fruta = 'banana'
longitud = len(fruta)
ultimo = fruta[longitud - 1]
print(ultimo)
```

Contenido

Alternativamente, puedes usar índices negativos, que cuentan hacia atrás desde el final de la cuerda. La expresión fruta[-1] da la última letra, fruta[-2] da la el penúltimo, y así sucesivamente.

Contenido

Rebanadas de String

Un segmento de un string se llama un trozo. Seleccionar una rebanada es similar a seleccionar una caracter:

```
s = 'Monty Python'
print(s[0:5])
print(s[6:12])
```

Contenido

Rebanadas de String

El operador devuelve la parte de la cadena del "n-ésimo" carácter "m-ésimo" al carácter, incluyendo el primero pero excluyendo el último [n,m). Si se omite el primer índice (antes de los dos puntos), el trozo comienza al principio de la cadena. Si se omite el segundo índice, la rebanada va al final de la cadena:

```
fruta = 'banana’
print(fruta[:3])
print(fruta[3:])
```

Contenido

Rebanadas de String

Si el primer índice es mayor o igual que el segundo el resultado es una cadena vacía, representada por dos comillas:

```
fruta = 'banana'
fruta[3:3]
''
```

Una cadena vacía no contiene caracteres y tiene una longitud de 0, pero aparte de eso, es igual a cualquier otra cadena.

Contenido

Ejercicio 1: ¿ Qué ocurre cuando se utiliza fruta[:]?

```
fruta = 'banana'
fruta[3:3]
```

Contenido

¿Los string son inmutables?

Es tentador utilizar el operador del lado izquierdo de una asignación, con la intención de cambiar un caracter en un string. Por ejemplo:

saludo = '¡Hola, mundo!'
saludo[0] = ‘J’

El "objeto" en este caso es la cadena y el "ítem" es el caracter que trataste de asignar. Por ahora, un objeto es lo mismo que un valor, pero afinaremos esa definición más adelante. Un ítem es uno de los valores de una secuencia. La razón del error es que las cadenas son **INMUTABLES**, lo que significa que no se puede cambiar una cadena existente. Lo mejor que puedes hacer es crear una nueva cadena que sea una variación de la original:

Contenido

```
saludo = '¡Hola, mundo!'
nuevo_saludo = 'J' + saludo[1:]
print(nuevo_saludo)
```

Este ejemplo concatena una nueva primera letra en un trozo de saludo. No tiene ningún efecto en la cadena original.

Contenido

El operador "in"

La palabra "in" es un operador booleano que toma dos cadenas y devuelve True si la primera aparece como una subcadena en la segunda:

```
var1 = 'a'
var2 = 'banana'
print(var1 in var2)

var1 = 'ola'
var2 = 'banana'
print(var1 in var2)
```

Contenido

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')
```

Contenido

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

```
palabra = 'pera'

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')
```

Contenido

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.

Contenido

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.

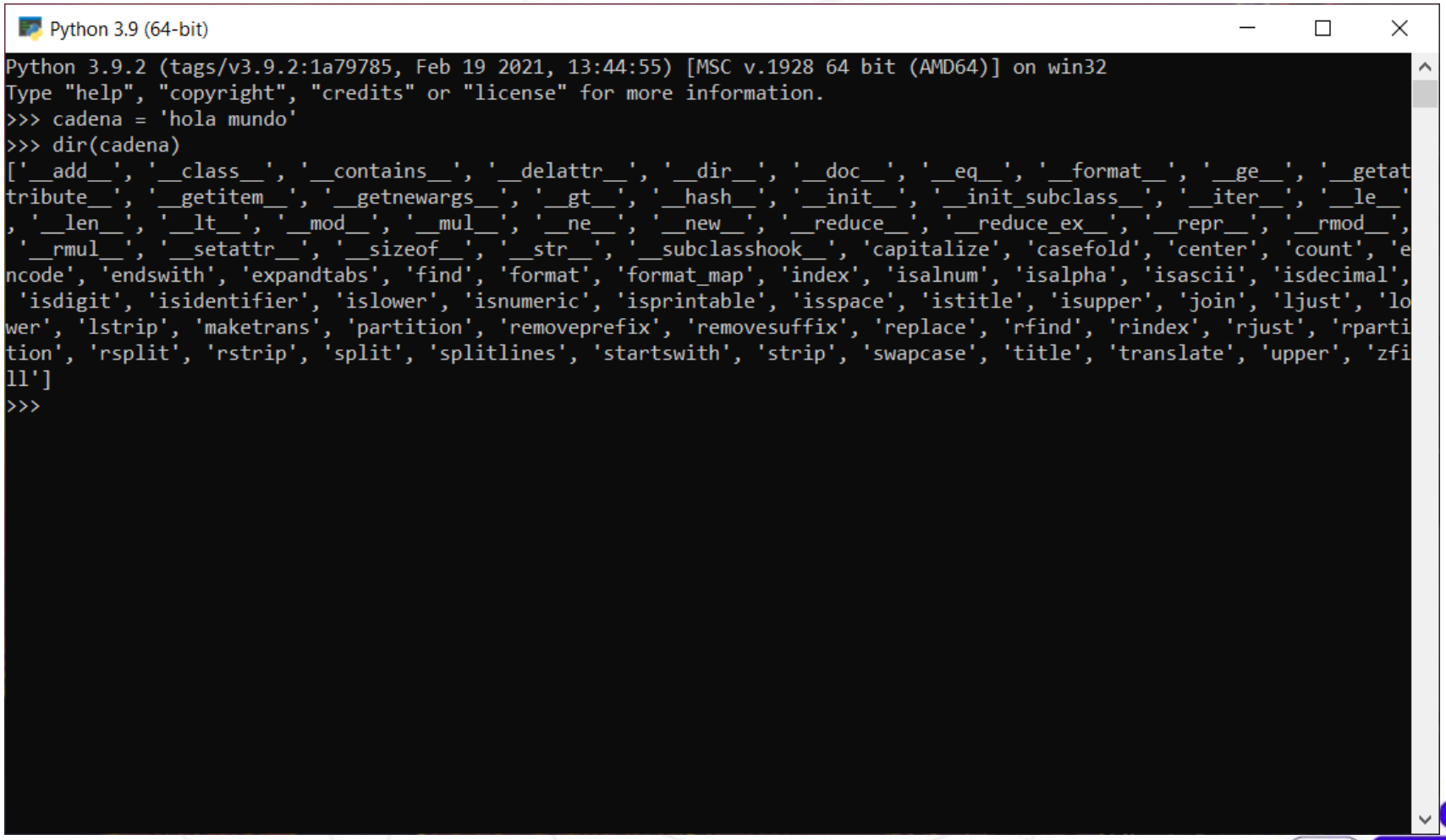
Contenido

Método dir()

Cadena = 'Hola mundo’

type(Cadena)

dir(Cadena)



Contenido

Métodos de String

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 sería

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.

Contenido

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().

```
palabra = 'banana'
palabra_nueva = palabra.upper()
print(palabra_nueva)
```

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**.

Contenido

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)
```

En este ejemplo, invocamos a find sobre palabra y pasamos la letra que buscamos como parámetro.

Contenido

El método find puede encontrar subcadenas así como caracteres:

```
palabra = 'banana'
print(palabra.find('na’))
```

Puede tomar como segundo argumento el índice donde debe comenzar:

```
palabra = 'banana'
print(palabra.find('na', 3))
```

Contenido

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:

```
linea = '       Aquí vamos          '
print(linea.strip())
```

Contenido

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

```
linea = 'Que Tengas Un Buen Día'
print(linea.startswith('Que’))

print(linea.startswith(‘q’))
```

Contenido

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.

```
linea = 'Que Tengas Un Buen Día'
print(linea.startswith('t'))

print(linea.lower().startswith('q'))

print(linea.lower()) #que tengas un buen día
```

En el penúltimo ejemplo, se llama el método "lower" y luego usamos "startswith" para ver si la cadena de minúsculas resultante comienza con la letra "Q". Siempre que seamos cuidadosos con el orden, podemos hacer múltiples llamadas al método en una sola expresión.

Contenido

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.

Contenido

```
#Consigue la @
data = 'De stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
enlaposicion = data.find('@')
print(enlaposicion)

#Consigue el espacio luego de uct.ac.za
espacioenlaposicion = data.find(' ',enlaposicion)
print(espacioenlaposicion)

#Corta el fragmento  localizado  previamente
host = data[enlaposicion+1:espacioenlaposicion]
print(host)
```

Contenido

La documentación para el método de búsqueda **find** está disponible en https://docs.python.org/library/stdtypes.html#string-methods.

Contenido

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"):

```
camellos = 42
print('%d' % camellos)
```

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

Contenido

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

```
camellos = 42
print('He visto %d camellos' % camellos)
```

Contenido

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.

Contenido

Caracteres especiales en el string

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

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

Contenido

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 ejemplos

```
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.
```

Contenido

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

```
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"
```

Contenido

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

```
# 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))
```

Contenido

el método format() de string

Si a los parámetros les damos nombre, podemos usar ese nombre en vez de su número de posición

```
# saca "2 y 1"
print ("{pepe} y {juan}".format(juan=1, pepe=2))
```

Podemos dar formato a los números para indicar cuántas cifras y decimales queremos, siguiendo una sintaxis similar al operador % que acabamos de ver. También podemos mezclar este formato con el número o nombre de posición del parámetro. Para mas información, consulta: https://docs.python.org/es/3/tutorial/inputoutput.html

Contenido

Concatenar

La declaración print se ejecuta sólo si pasamos los dos condicionales, para que podamos obtener el mismo efecto con el operador y:

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”

```
mensaje1 = 'Hola' + ' ' + 'Mundo'
print(mensaje1)
```

Contenido

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.

```
mensaje2a = 'Hola ' * 3
mensaje2b = 'Mundo'
print(mensaje2a + mensaje2b)
```

Contenido

Añadir

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

```
mensaje3 = 'Hola'
mensaje3 += ' '
mensaje3 += 'Mundo'
print(mensaje3)
```