Python: strings

SISTEMAS DE GESTIÓN EMPRESARIAL – 148FA (DAM)

Strings

Cualquier variable que contenga un valor de tipo string será tratada como un subtipo del objeto string. Este objeto contiene varios métodos, como hemos visto con listas.

```
a = “Hola mundo”
b = “3”
c = “Hola” + “ qué tal”
```

Strings

Como si fuera una lista, podemos acceder a un "*subelemento*" del string o cadena de caracteres utilizando **[ ]** y un **índice**:

```
mi_cadena = "Hola mundo"
print (mi_cadena[0:4])
>>> "Hola"
```

Podemos poner o no poner el límite a la izquierda o a la derecha. Si no ponemos a la izquierda, comenzará desde la izquierda del todo. Si no ponemos en la derecha, llegará hasta el final.

```
print (mi_cadena[1:])
>>> "ola Mundo"
print (mi_cadena[:4])
>>> "Hola"
```

Strings

Para **concatenar** dos cadenas de caracteres podemos utilizar el **símbolo + entre las dos**.

```
a = “Hola”
b = “Mundo”
c = a + b
print (c)
>>> “Hola Mundo”
```

Strings - métodos

Convertir una cadena a mayúsculas

- Método: **upper()**
- Acción: devuelve una copia de la cadena en mayúsculas

```
mi_string = “Hola Mundo”
print (mi_string.upper())  # HOLA MUNDO
```

Convertir una cadena a minúsculas

- Método: **lower()**
- Acción: devuelve una copia de la cadena en minúsculas

```
mi_string = “Hola Mundo”
print (mi_string.lower())  # hola mundo
```

Strings - métodos

Convertir a mayúscula la primera letra

- Método: **capitalize()**
- Acción: devuelve una copia de la cadena con la primera letra en mayúsculas

```
mi_string = “hola mundo”
print (mi_string.capitalize())  # Hola mundo
```

Convertir mayúsculas a minúsculas y viceversa

- Método: **swapcase()**
- Acción: devuelve una copia de la cadena con las mayúsculas convertidas en minúsculas y viceversa

```
mi_string = “Hola Mundo”
print (mi_string.swapcase())  # hOLA mUNDO
```

Strings - métodos

Convertir una cadena en formato título

- Método: **title()**
- Acción: devuelve una copia de la cadena convertida

```
mi_string = “hola mundo”
print (mi_string.title())  # Hola Mundo
```

Strings - métodos

Centrar un texto

- Método: **center(longitud[, "caracter de relleno"])**
- Acción: devuelve una copia de la cadena centrada

```
mi_string = “esta es mi cadena de texto”.capitalize()
print (mi_string.center(50, “=“))
```

==========Esta es mi cadena de texto==========

```
print (mi_string.center(50, “ “))
```

Esta es mi cadena de texto

Strings - métodos

Alinear un texto a la izquierda

- Método: **ljust(longitud[, "caracter de relleno"])**
- Acción: devuelve una copia de la cadena alineada a la izquierda

```
mi_string = “esta es mi cadena de texto”.capitalize()
print (mi_string.ljust(50, “=“))
```

Esta es mi cadena de texto==========

Strings - métodos

Alinear un texto a la derecha

- Método: **rjust(longitud[, "caracter de relleno"])**
- Acción: devuelve una copia de la cadena alineada a la derecha

```
mi_string = “esta es mi cadena de texto”.capitalize()
print (mi_string.rjust(50, “=“))
```

==========Esta es mi cadena de texto

```
print (mi_string.rjust(50, “ “))
```

Esta es mi cadena de texto

Strings - métodos

Rellenar un texto anteponiendo ceros

- Método: **zfill(longitud)**
- Acción: devuelve una copia de la cadena con ceros a la izquierda hasta llegar a la longitud que se indica

```
numero_socio= 37
print (numero_socio.zfill(4))
```

```
>>> 0037
```

Strings - métodos

Contar la cantidad de apariciones de una subcadena

- Método: **count("subcadena" [, posicion_inicio, posicion_fin])**
- Acción: devuelve un número entero que indica la cantidad de apariciones de la subcadena

```
mi_cadena= “la palabra azul tiene una a”
print (mi_cadena.count(“a”))

>>> 7
```

Strings - métodos

Buscar una subcadena dentro de una cadena

- Método: **find("subcadena" [, posicion_inicio, posicion_fin])**
- Acción: devuelve un entero representando la posición donde se inicia la subcadena dentro de cadena. Si no la encuentra, devuelve -1

```
mi_cadena= “Hola mundo, estoy estudiando Python”
print (mi_cadena.find(“Python”))

>>> 29

print (mi_cadena.find(“Python”, 0, 20))


>>> -1
```

Strings – métodos de validación

startswith("subcadena" [, posicion_inicio, posicion_fin]): saber si una cadena empieza con una subcadena determinada

endswith("subcadena" [, posicion_inicio, posicion_fin]): saber si una cadena termina con una cadena determinada

isalnum(): saber si una cadena es alfanumérica

isalpha(): saber si una cadena es alfabética

isdigit(): saber si una cadena es numérica

islower(): saber si una cadena sólo tiene minúsculas

isupper(): saber si una cadena sólo tiene mayúsculas

isspace(): saber si una cadena sólo tiene espacios en blanco

istitle(): saber si una cadena tiene formato de título

Strings – métodos de sustitución

Reemplazar texto en una cadena

Método: **replace("subcadena a buscar", "subcadena por la cual reemplazar")**

Acción: devuelve la cadena resultante

```
buscar = “Java”
reemplazar=“Python”
print (“Estoy estudiando Java”.replace(buscar, reemplazar))
>>> Estoy estudiando Python
```

Strings – métodos de sustitución

Eliminar caracteres a la izquierda y a la derecha de una cadena

Método: strip(["caracter"])

Acción: devuelve la cadena resultante sin espacios ni por la derecha ni por la izquierda

```
mi_cadena = “       Esta es una cadena de texto      ”
print (mi_cadena.strip())
>>> Esta es una cadena de texto
```

Strings – métodos de sustitución

Eliminar caracteres a la izquierda y a la derecha de una cadena

Método: strip(["caracter"])

Acción: devuelve la cadena resultante convertida

```
mi_cadena = “       Esta es una cadena de texto      ”
print (mi_cadena.strip())
>>> Esta es una cadena de texto
```

Strings – métodos de sustitución

Eliminar caracteres a la izquierda

Método: **lstrip(["caracter"])**

Acción: devuelve la cadena convertida

```
mi_cadena = “www.google.es”
print (mi_cadena.lstrip(“w.”))
>>> google.es
```

Eliminar caracteres a la derecha

Método: **rstrip(["caracter"])**

Acción: devuelve la cadena convertida

```
mi_cadena = “www.google.es”
print (mi_cadena.rstrip(“es”))
>>> www.google.
```

Strings – métodos de sustitución

Dar formato a una cadena sustituyendo elementos dinámicamente

Método: **format**

```
mi_cadena = ‘Hoy es {0} y voy a estudiar {1}’

print (mi_cadena.format(‘viernes’,’Python’))

>>> Hoy es viernes y voy a estudiar Python
```

Strings – métodos de unión y división

Unir cadena de forma iterativa

Método: **join(iterable)**

Acción:

```
numLista = ['1', '2', '3', '4’]

separador = ','

print(seperator.join(numLista))

>>> 1, 2, 3, 4
```

```
test = {'Python', 'Java', 'Ruby’}

s = '->->’

print(s.join(test))
```

Strings – métodos de unión y división

Partir una cadena usando un separador

Método: **partition("separador")**

```
mi_cadena = ‘Python es divertido’

print(mi_cadena.partition('es’))

>>> (‘Python’, ’es’, ‘divertido’)

print(mi_cadena.partition(‘no’))

>>> (‘Python es divertido’, ‘’, ‘’)
```

Strings – métodos de unión y división

Separar una cadena en elementos de una lista

Método: **split("separador")**

```
mi_cadena = ‘Python es divertido’
print(mi_cadena.split())
>>> [‘Python’,’es’,’divertido’]
mi_cadena2 = ‘Python es, divertido’
print(mi_cadena.split(‘,’))
>>> [‘Python es’,’divertido’]
```

Strings – Ejercicios (1)

1. Elige un carácter e insértalo entre cada letra de una cadena de caracteres.

Por ejemplo: **'separar'** y **','** debería devolver **'s,e,p,a,r,a,r'**

2. Reemplaza todos los espacios por el carácter.

Por ejemplo: **'mi archivo de texto.txt'** y **'_'** debería devolver **'mi_archivo_de_texto.txt'**

3. Reemplaza todos los dígitos en la cadena por el carácter que elijas.

Por ejemplo **'su clave es: 1540'** y **'X'** debería devolver **'su clave es: XXXX'**

4. Inserte el carácter cada 3 dígitos en la cadena.

Por ejemplo, **'2552552550'** y **'.'** debería devolver **'255.255.255.0'**

Strings – Ejercicios (2)

5. Dadas las cadenas a y b, debes devolver una cadena simple con a y b separadas con un espacio. Debes intercambiar los dos primeros caracteres de cada cadena. Por ejemplo:

   a = “juglar”
   b = “jaguar”
   >>> “jaglar juguar”

Strings – Ejercicios (3)

Crear un módulo para validación de nombres de usuarios. Dicho módulo, deberá cumplir con los siguientes **criterios de aceptación**:

- El nombre de usuario debe contener un mínimo de 6 caracteres y un máximo de 12.
- El nombre de usuario debe ser alfanumérico.
- Si el nombre de usuario con menos de 6 caracteres, devuelve el mensaje *"El nombre de usuario debe contener al menos 6 caracteres"*.
- Si el nombre de usuario con más de 12 caracteres, devuelve el mensaje *"El nombre de usuario no puede contener más de 12 caracteres"*.
- Si el nombre de usuario con caracteres distintos a los alfanuméricos, devuelve el mensaje *"El nombre de usuario puede contener solo letras y números"*.
- Nombre de usuario válido, devuelve *True*.