



UNIVERSIDAD  
DE MÁLAGA



# Strings

## Unidad 6

Python para principiantes  
[dgm@uma.es](mailto:dgm@uma.es)



# El tipo de datos string

- Un string es una **secuencia** de caracteres (**cadena de caracteres**)
- Los literales string utilizan **comillas** 'Hello' o "Hello"
- Para strings, + significa “**concatenar**”
- Cuando un string contiene números, es todavía un string
- Podemos convertir los números de un string en números utilizando int()

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

# Leyendo y Convirtiendo

- Es mejor leer los datos usando strings y después procesarlos y **convertirlos** al tipo que necesitamos
- Esto nos da más control sobre situaciones de error y/o **entradas** del usuario **incorrectas**
- Los números de entrada deben convertirse desde strings

```
>>> name = input('Escribe:')
Escribe:Chuck
>>> print(name)
Chuck
>>> apple = input('Escribe:')
Escribe:100
>>> x = apple - 10
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: unsupported operand
type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print(x)
90
```



# Mirando dentro de los strings

- Podemos acceder a cualquier **carácter** simple de un string usando un índice especificado entre **corchetes [ ]**
- El índice debe ser un entero, y **comienzan en cero**
- El valor del índice puede ser una expresión calculada

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

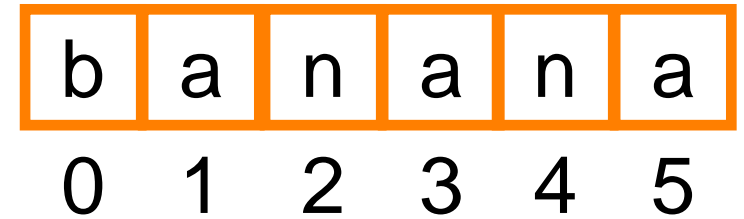
# Un carácter demasiado lejano

- Obtendremos un error python si intentamos acceder a un índice **más allá del final** del string
- Ten cuidado cuando construyas los índices y slices (ver más adelante)

```
>>> zot = 'abc'
>>> print(zot[5])
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
IndexError: string index out
of range
>>>
```

# Los strings tienen longitud

La función interna **len** proporciona la **longitud** de un string

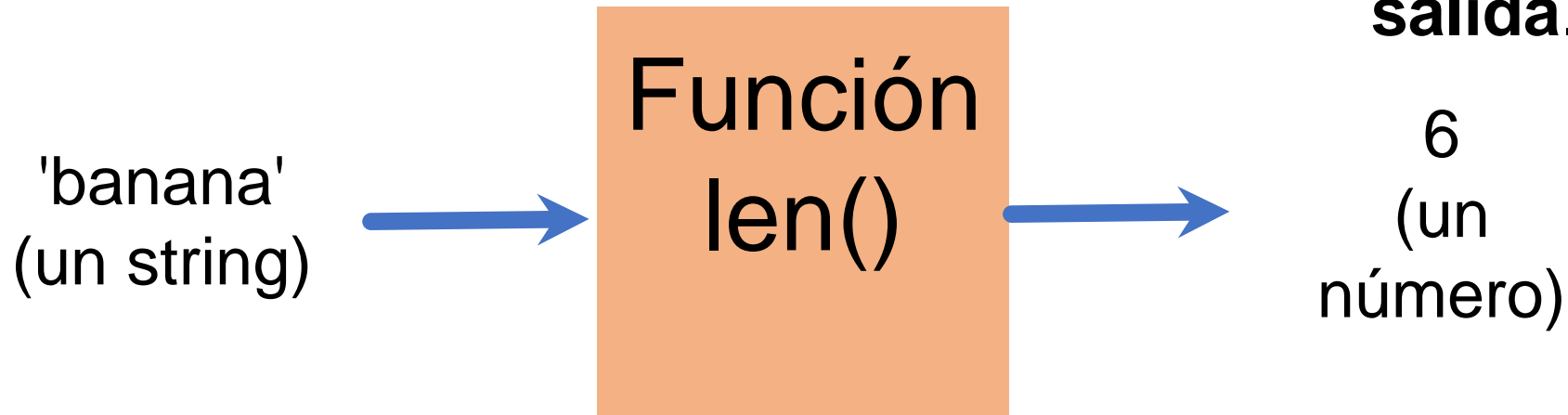


```
>>> fruit = 'banana'
>>> print(len(fruit))
6
```

# La función len

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

Una función es algún código almacenado que podemos usar. Una función toma una **entrada** y produce una **salida**.



# La función len

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

Una función es algún código almacenado que podemos usar. Una función toma una **entrada** y produce una **salida**.

'banana'  
(un string)



```
def len(inp):
    blah
    blah
    for x in y:
        blah
        blah
```



6  
(un  
número)



# Iterando sobre strings

- Usando un **bucle while**, una **variable de iteración**, y la función **len**, podemos mirar cada una de las letras de un string individualmente

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

0 b  
1 a  
2 n  
3 a  
4 n  
5 a

# Iterando sobre strings

- Pero es mucho más elegante usar un **bucle for**
- La variable de iteración va tomando como valor **cada una de las letras** del string

```
fruit = 'banana'  
for letter in fruit:  
    print(letter)
```

b  
a  
n  
a  
n  
a

# Iterando sobre strings

- Pero es mucho más elegante usar un **bucle for**
- La variable de iteración va tomando como valor **cada una de las letras** del string

```
fruit = 'banana'
for letter in fruit :
    print(letter)
```

```
index = 0
while index < len(fruit) :
    letter = fruit[index]
    print(letter)
    index = index + 1
```

b  
a  
n  
a  
n  
a

# Iterando y contando

- Ejemplo de bucle simple que **itera** a través de las letras de un string y cuenta el número de **apariciones de la letra 'a'**


```
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print(count)
```

# Mirando en detalle in

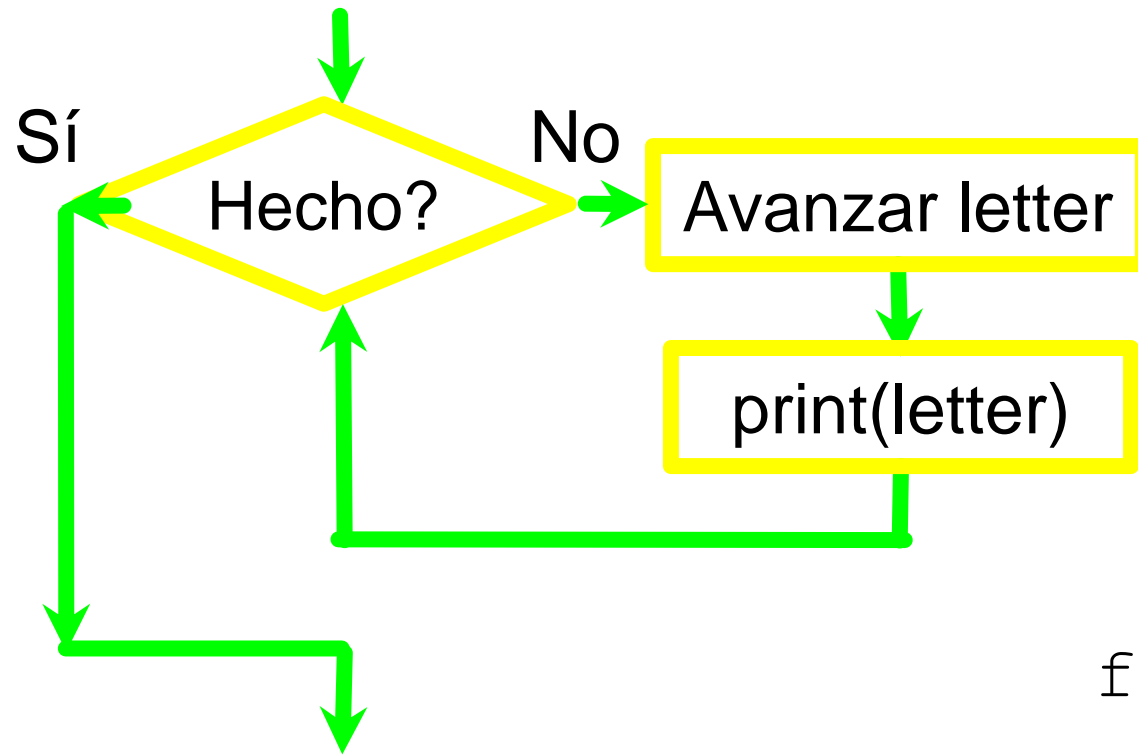
- La variable de iteración “**itera**” a través de la secuencia (conjunto ordenado)
- El bloque (cuerpo) de código es ejecutado una vez **para cada valor** en (in) la secuencia
- La variable de iteración **se mueve** a través de todos los valores en (in) la secuencia

Variable de  
iteración

String de 6 letras



```
for letter in 'banana' :  
    print(letter)
```



```
for letter in 'banana' :  
    print(letter)
```

La variable de iteración “**itera**” a través del string y el bloque (cuerpo) de código es ejecutado una vez **para cada valor** en la secuencia

# Más operaciones de strings

# Cortando Strings (slices)

- Podemos mirar cualquier sección continua (rodaja) de un string utilizando el **operador dos puntos** (:)
- El primer número indica desde dónde **empezamos** a cortar
- El segundo número indica el **final** del slice (rodaja) – **pero sin incluirlo**
- Si el segundo número está más allá del final del string, se queda en el final
- Ten en cuenta que comenzamos en **0**

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```



# Cortando Strings (slices)

- Si no ponemos el primer o último número del slice, se asume que se está usando la primera o última letra del string respectivamente

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

# Concatenación de strings

Cuando se aplica el **operador +** a strings, significa “**concatenación**”

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print(b)
HelloThere
>>> c = a + ' ' + 'There'
>>> print(c)
Hello There
>>>
```

# Usando in como operador lógico

- La palabra clave in puede ser usada para comprobar si un **string está "en" otro string**
- El uso de in funciona como una expresión lógica que retorna **True o False** y puede ser usada en una sentencia if

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Encontrado!')
...
Encontrado!
>>>
```

# Comparación de strings

```
if word == 'banana':  
    print('Todo bien, bananas.')  
if word < 'banana':  
    print('Tu palabra,' + word + ', es menor que banana.')elif word > 'banana':  
    print('Tu palabra,' + word + ', es mayor que banana.')else:  
    print('Todo bien, bananas.')
```

# La librería de strings

- Python tiene muchas funciones para el manejo de strings en la **librería de strings**
- Estas funciones son realmente internas en cada string – podemos invocarlas simplemente poniéndolas con el **operador punto (.)** a continuación del string ('string'.función())
- Estas funciones **no modifican el string**, en su lugar retornan un nuevo string resultante de aplicar la función

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print(zap)
hello bob
>>> print(greet)
Hello Bob
>>> print('Hi There'.lower())
hi there
>>>
```

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartmention', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

<https://docs.python.org/3/library/stdtypes.html#string-methods>

**str.replace(*old*, *new*[, *count*])**

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

**str.rfind(*sub*[, *start*[, *end*]])**

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within `s[start:end]`. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

**str.rindex(*sub*[, *start*[, *end*]])**

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

**str.rjust(*width*[, *fillchar*])**

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

**str.rpartition(*sep*)**

Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

**str.rsplit(*sep=None*, *maxsplit=-1*)**

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.

# Librería de strings

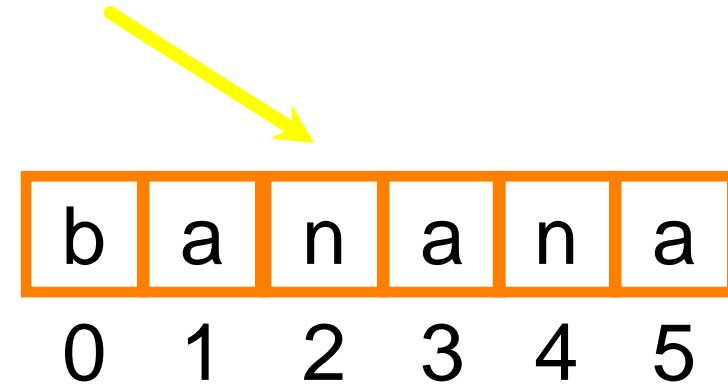
```
str.capitalize()  
str.center(width[, fillchar])  
str.endswith(suffix[, start[, end]])  
str.find(sub[, start[, end]])  
str.lstrip([chars])
```

```
str.replace(old, new[, count])  
str.lower()  
str.rstrip([chars])  
str.strip([chars])  
str.upper()
```



# Buscando en un String

- Usamos la función **find()** para buscar un **substring** dentro de otro string
- **find()** encuentra la primera aparición del substring
- Si el substring no se encuentra,, **find()** retorna -1
- Recuerda que las posiciones de los strings comienzan en cero



```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

# Convirtiendo todo a MAYÚSCULAS

- Podemos conseguir copias de un string en minúsculas o mayúsculas
- Frecuentemente, cuando buscamos en un string con `find()`, primero debemos convertir el string a minúsculas para poder buscar independientemente de cómo haya escrito el usuario

```
>>> greet = 'Hello Bob'
>>> nnn = greet.upper()
>>> print(nnn)
HELLO BOB
>>> www = greet.lower()
>>> print(www)
hello bob
>>>
```

# Buscar y Reemplazar

- La función **replace()** es como el "**buscar y reemplazar**" de un procesador de textos
- Reemplaza todas las ocurrencias del string buscado con el string de reemplazamiento y devuelve el nuevo string

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob', 'Jane')
>>> print(nstr)
Hello Jane
>>> nstr = greet.replace('o', 'X')
>>> print(nstr)
HellX BXb
>>>
```

# Eliminando espacios

- Algunas veces nos interesará **quitar los espacios** al principio o al final de un string
- **lstrip()** o **rstrip()** borran los espacios en blanco a la izquierda o la derecha respectivamente
- **strip()** borra los espacios al principio y al final

```
>>> greet = '    Hello Bob    '  
>>> greet.lstrip()  
'Hello Bob    '  
>>> greet.rstrip()  
'    Hello Bob'  
>>> greet.strip()  
'Hello Bob'  
>>>
```

# Prefijos

```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

# Procesando (parsing) y extrayendo

21



31



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

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ', atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```



# Dos clases de strings

```
Python 2.7.10
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

```
Python 3.5.1
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```

En Python 3, todos los strings son Unicode

# Resumen

- El tipo String
- Leer/Convertir
- Indexando strings []
- Cortando strings [2:4]
- Iterando strings con for y while
- Concatenando strings con +
- Operaciones de strings
- La librería de strings
- Comparaciones de strings
- Buscando en strings
- Reemplazando texto
- Eliminando espacios



## Ejercicio 1

Utilizar find() y los slices (:) para extraer el número en un string como el que aparece a continuación. Convertir el número a flotante y mostrarlo.

¡Ojo! No asumir que el número es 0.8475, es solo un ejemplo. No sabemos qué número es, tenemos que extraerlo. Sí podemos suponer que el resto de la línea es igual. Esto es, comienza por "X-DSPAM-Confidence..."

"X-DSPAM-Confidence: 0.8475"

## Ejercicio 2

Pedir un nombre al Usuario de la forma:

NOMBRE APELLIDO1 APELLIDO2

Por ejemplo: José García Pérez

Dar la Vuelta al nombre y presentarlo en el formato:

APELLIDO1 APELLIDO2, NOMBRE

Por ejemplo: García Pérez, José

## Ejercicio 3

Pedir una contraseña al Usuario repetidas veces hasta que la contraseña sea considerada fuerte. Para ello, debe tener, como mínimo 8 caracteres y contener, al menos, tanto una mayúscula como un dígito

estaEslaCon8traseña sería válida

abcdefghijklmnopq no sería válida porque no cumple las condiciones

La solución debe incluir una función que reciba un string y retorne si es una contraseña fuerte o no

Pista: usar las funciones `isdigit`, `isupper`

# Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here  
Spanish Version: Daniel Garrido ([dgm@uma.es](mailto:dgm@uma.es))