

Cadenas

El Tipo de dato Cadena

- Una cadena (string) es una secuencia de caracteres.
- Una cadena utiliza comillas 'Hola' o "Hola"
- Para las cadenas, + significa “concatenar”.
- Cuando una cadena contiene números, aún sigue siendo una cadena.
- Podemos convertir números dentro de una cadena, a enteros, utilizando `int()`

```
>>> str1 = "Hola"
>>> str2 = 'ahí'
>>> bob = str1 + str2
>>> print(bob)
Holaahí
>>> 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 datos

- Preferimos leer datos de entrada utilizando **cadenas** y después analizar y convertir los datos conforme sea necesario
- Esto nos da más control sobre situaciones de error y/o datos de entrada del usuario incorrectos
- Los números como datos de entrada deben ser **convertidos** de cadenas a enteros

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



Buscando dentro de una Cadena

- Podemos obtener cualquier carácter en una cadena usando un índice especificado en **corchetes**
- El valor del índice debe ser un entero y comienza desde el cero
- El valor del índice puede ser una expresión que se ha calculado

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

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

Un carácter muy lejano

- Vas a obtener un **error de Python** si tratas de acceder un índice más allá del final de la cadena.
- Así que sé cuidadoso cuando construyas valores de índices y rebanadas

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

Las Cadenas tienen Tamaño

La función nativa **len** nos retorna el tamaño de una cadena

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

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

Función len

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

Una función es un código almacenado que utilizamos. Una función toma **datos de entrada** y produce **datos de salida**.

'banana'
(una cadena)



Función
len()



6
(un número)

Función len

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

Una función es un código almacenado que utilizamos. Una función toma **datos de entrada** y produce **datos de salida**.

'banana'
(una cadena)



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



6
(un número)

Recorriendo una Cadena

Utilizando una sentencia **while**, una **variable de iteración**, y la función **len**, podemos construir un bucle para mirar cada una de las letras de una cadena de forma individual

```
fruta = 'banana'
indice = 0
while indice < len(fruta):
    letra = fruta[indice]
    print(indice, letra)
    indice = indice + 1
```

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

Recorriendo una Cadena

- Un bucle finito utilizando una sentencia **for** es mucho más elegante
- La **variable de iteración** es completamente manejada por el bucle **for**

```
fruta = 'banana'
for letra in fruta:
    print(letra)
```

b
a
n
a
n
a

Recorriendo una Cadena

- Un bucle finito utilizando una sentencia **for** es mucho más **elegante**
- La **variable de iteración** es completamente manejada por el bucle **for**

```
fruta = 'banana'
for letra in fruta :
    print(letra)
```

```
indice = 0
while indice < len(fruta) :
    letra = fruta[indice]
    print(letra)
    indice = indice + 1
```

b
a
n
a
n
a

Recorriendo y Contando

Este es un bucle sencillo que itera a través de cada letra en una cadena y cuenta el número de veces que el bucle encuentra el carácter 'a'

```
palabra = 'banana'
contador = 0
for letra in palabra :
    if letra == 'a' :
        contador =
contador + 1
print(contador)
```

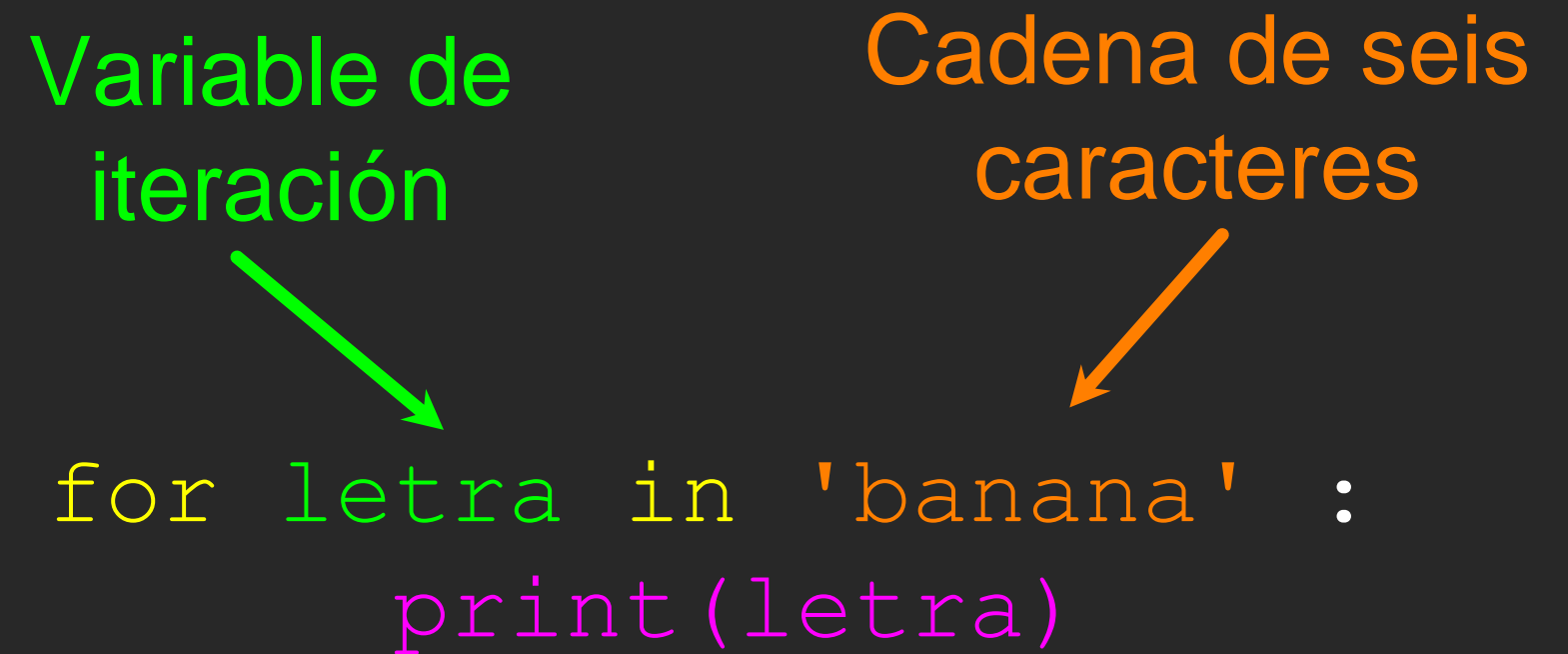
Analizando in más a fondo

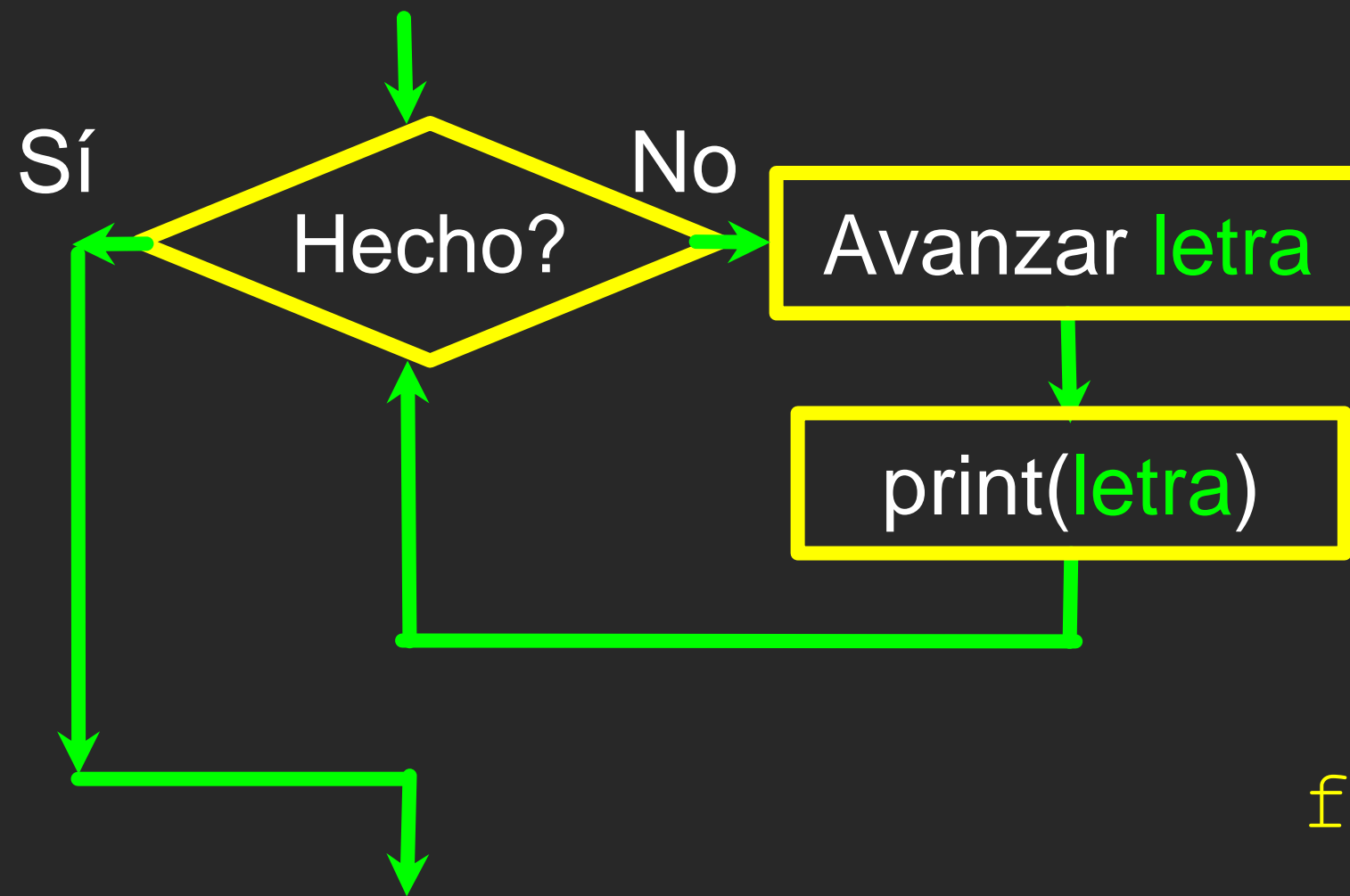
- La **variable de iteración** “itera” a través de una **secuencia** (un conjunto ordenado)
- El **bloque (cuerpo)** de código es ejecutado una vez para cada valor **en (in)** la **secuencia**
- La **variable de iteración** se mueve a través de todos los valores **en (in)** la **secuencia**

Variable de iteración

Cadena de seis caracteres

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





b	a	n	a	n	a
---	---	---	---	---	---

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

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

Más Operaciones de Cadenas

Rebanado de Cadenas

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

- También podemos mirar a cualquier sección continua de una cadena utilizando el **operador dos puntos**
- El segundo número es un número más allá del final de la rebanada - “hasta pero no incluyendo”
- Si el segundo número está más allá del final de la cadena, entonces termina al final

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


Rebanado de Cadenas

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

Si dejamos en blanco el primer o el último número de la rebanada, se asume que es el inicio o el final de la cadena, respectivamente

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

Concatenación de Cadenas

Cuando el operador **+** es aplicado a una cadena, significa “**concatenación**”

```
>>> a = 'Hola'
>>> b = a + 'Ahí'
>>> print(b)
HolaAhí
>>> c = a + ' ' + 'Ahí'
>>> print(c)
Hola Ahí
>>>
```

Utilizando **in** como Operador Lógico

- La palabra **in** puede ser utilizada para revisar si una cadena se encuentra “**en (in)**” otra cadena
- La expresión **in** es una expresión lógica que retorna **True** o **False** y puede ser utilizada una sentencia **if**

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

Comparación de Cadenas

```
if palabra == 'banana':  
    print('Muy bien, bananas.')  
if palabra < 'banana':  
    print('Tu palabra,' + palabra + ', está antes de banana.')elif palabra > 'banana':  
    print('Tu palabra,' + palabra + ', está después de banana.')else:  
    print('Muy bien, bananas.')
```

Librería String

- Python tiene un número de **funciones de cadenas** que están en la **librería string (cadena)**
- Esas **funciones** ya están previamente **construidas dentro** de cada cadena – las invocamos al agregar la función a la variable de la cadena
- Esas **funciones** no modifican la cadena original, sino que retornan una nueva cadena que ha sido modificada

```
>>> saludo = 'Hola Bob'
>>> zap = saludo.lower()
>>> print(zap)
hola bob
>>> print(saludo)
Hola Bob
>>> print('Hola Ahí'.lower())
hola ahí
>>>
```

```
>>> cosa = 'Hola mundo'
>>> type(cosa)
<class 'str'>
>>> dir(cosa)
['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',
'rstrip', '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 String

```
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()
```

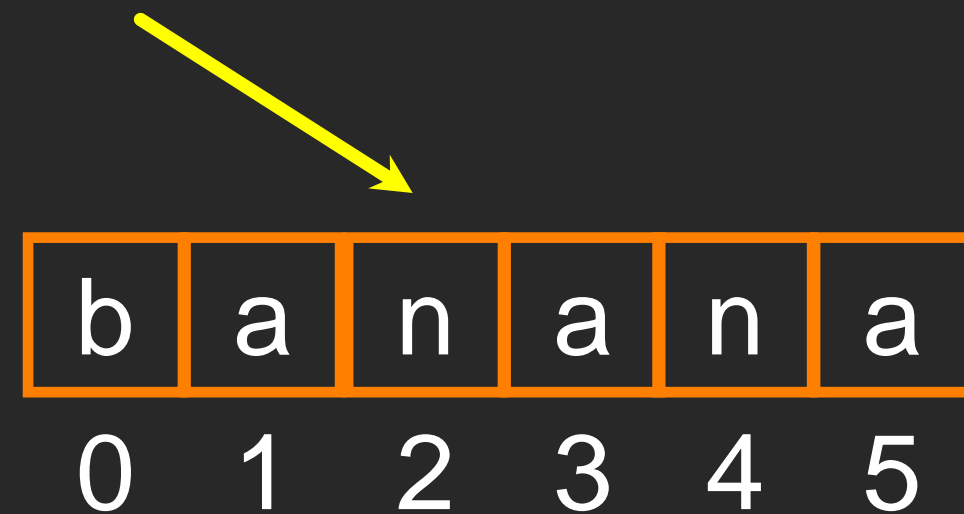
```
str.rstrip([chars])
```

```
str.strip([chars])
```

```
str.upper()
```


Buscando una Cadena

- Utilizamos la función `find()` para buscar una subcadena dentro de otra cadena
- `find()` encuentra la primer ocurrencia de la subcadena
- Si la subcadena no se encuentra, `find()` regresa `-1`
- Recuerda que las posiciones de una cadena comienzan en cero.



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

Convirtiéndolo Todo a MAYÚSCULAS

- Puedes crear una copia de una cadena en **minúsculas** o **mayúsculas**
- Frecuentemente cuando estamos buscando una cadena utilizando **find()** primero convertimos la cadena a minúsculas, de modo que podemos buscar una cadena sin importar si está en mayúsculas o minúsculas

```
>>> saludo = 'Hola Bob'
>>> nnn = saludo.upper()
>>> print(nnn)
HOLA BOB
>>> www = saludo.lower()
>>> print(www)
hola bob
>>>
```

Buscar y Reemplazar

- La función `replace()` es como una operación “buscar y reemplazar” en un editor de texto
- Esta función reemplaza **todas las ocurrencias** de una **cadena de búsqueda** con una **cadena de reemplazo**

```
>>> saludo = 'Hola Bob'
>>> ncad = saludo.replace('Bob', 'Jane')
>>> print(ncad)
Hola Jane
>>> ncad = saludo.replace('o', 'x')
>>> print(ncad)
HXla BXb
>>>
```

Removiendo Espacios en Blanco

- A veces queremos tomar una cadena y remover los espacios en blanco al inicio y/o al final
- **lstrip()** y **rstrip()** remueven los espacios en blanco a la izquierda o a la derecha
- **strip()** remueve espacios en blanco tanto al inicio como al final de la cadena

```
>>> saludo = '    Hola Bob    '  
>>> saludo.lstrip()  
'Hola Bob '  
>>> saludo.rstrip()  
'    Hola Bob'  
>>> saludo.strip()  
'Hola Bob'  
>>>
```

Prefijos

```
>>> linea = 'Que tengas un buen día'
```

```
>>> linea.startswith('Que')
```

```
True
```

```
>>> linea.startswith('q')
```

```
False
```

Análisis y Extracción

21



31



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

```
>>> datos = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> arrpos = datos.find('@')
>>> print(arrpos)
21
>>> esppos = datos.find(' ', arrpos)
>>> print(esppos)
31
>>> direccion = datos[arrpos+1 : esppos]
>>> print(direccion)
uct.ac.za
```

Dos tipos de Cadenas

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, todas las cadenas son Unicode