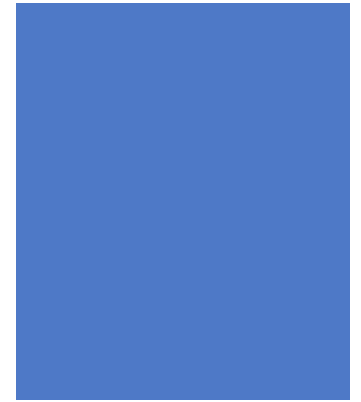


Tema 7.

Tipos estructurados (secuencias)



Curso 2020/2021
Programación



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Índice

1. Cadenas:

- indexación
- codificación y operaciones básicas.
- Tratamiento: recorridos y búsquedas

2. Listas y listas anidadas:

- creación, acceso y otras operaciones básicas.
- Tratamiento: recorridos y búsquedas

3. Operaciones sobre strings y listas

1. Cadenas o Strings

- En el Tema 2 ya presentamos el tipo de dato *cadena* o *string* y sus operaciones básicas
 - Concatenación (+)
 - `"Hola" + " " + "todos!"` es `"Hola todos!"`
 - Repetición (*)
 - `3 * "Hola "` es `"Hola Hola Hola "`
 - Conversiones desde/a otros tipos de datos
 - Con `str()`
 - Comparaciones entre cadenas
 - `==` , `!=` , `<` , `<=` , `>` y `>=`

1. Indexación de cadenas

- También ya vimos: indexación de cadenas
- Acceder a cada uno de los caracteres de una cadena utilizando el operador de indexación []
 - Si a es una cadena, $a[0]$ es el primer carácter de la cadena, $a[1]$ el segundo, y así sucesivamente
- Ej. la cadena 'Hola, mundo.'

```
>>> a = "Hola, mundo."  
>>> a[0]  
'H'  
  
>>> a[1]  
'o'
```

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a	,		m	u	n	d	o	.

1. Indexación de cadenas

```
>>> a = "Hola, mundo."  
>>> a[0]  
'H'  
  
>>> a[1]  
'o'
```

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a	,		m	u	n	d	o	.

- El último carácter de una cadena `a` es `a[len(a)-1]`

```
>>> a[len(a)-1]  
'.'
```

- Es muy importante que el índice que usemos sea válido:
 - Si usamos un índice fuera de rango, Python lanza una excepción *IndexError*

```
>>> a[len(a)]  
Traceback (most recent call last):  
  File "<pyshell>", line 1, in <module>  
IndexError: string index out of range
```

1. Indexación de cadenas

- También se pueden utilizar índices negativos: los valores negativos acceden a los caracteres de derecha a izquierda. El último carácter de una cadena tiene índice -1 , el penúltimo, -2 , y así sucesivamente

```
>>> a = "Ejemplo"
```

```
>>> a[-1]
```

```
'o'
```

```
>>> a[-2]
```

```
'l'
```

```
>>> a[-4]
```

```
'm'
```

```
>>> a[-8]
```

```
Traceback (most recent call last):  
  File "<pyshell>", line 1, in <module>  
IndexError: string index out of range
```

```
>>> |
```

0	1	2	3	4	5	6
E	j	e	m	p	l	o
-7	-6	-5	-4	-3	-2	-1

1. Subcadenas: el operador de corte

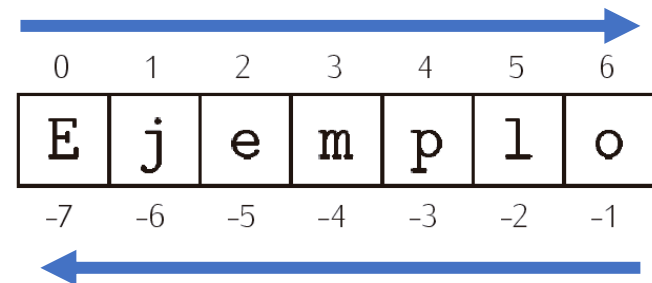
- El operador de corte se denota con dos índices separados por dos puntos (:), dentro de los corchetes de indexación
- La expresión `a[i:j]` devuelve la subcadena formada por los caracteres desde `a[i]` hasta `a[j-1]`

```
>>> a = "Ejemplo"
>>> a[2:5]
'emp'
>>> a[0:4]
'Ejem'
```

0	1	2	3	4	5	6
E	j	e	m	p	l	o
-7	-6	-5	-4	-3	-2	-1

9. Subcadenas: el operador de corte

- Al igual que en los rangos, al operador de corte se le puede añadir un tercer parámetro que indica el **incremento** del índice y la **dirección**
- Un numero POSITIVO indica ir de izquierda a la derecha
- Un numero NEGATIVO indica ir de derecha a la izquierda



```
>>> a[-4:-2:1]
'mp'
```

```
>>> a[-4:-2:-1]
''
```

```
>>> a[-2:-4:-1]
'lp'
```

```
>>> a[-2:-4:1]
''
```

```
>>> a[1:3:1]
'je'
```

```
>>> a[1:3:-1]
''
```

```
>>> a[3:1:1]
''
```

```
>>> a[3:1:-1]
'me'
```


9. Métodos de manejo de cadenas

Método	Descripción	Ejemplo (si <code>s = 'Valencia'</code>)
<code>s.count(t)</code>	Número de ocurrencias de la cadena <code>t</code> dentro de <code>s</code>	<pre>>>> s.count('a') 2</pre>
<code>s.find(t)</code>	Posición de la 1ª ocurrencia de <code>t</code> en <code>s</code> . Si no esta devuelve -1	<pre>>>> s.find('en') 3 >>> s.find('Pa') -1</pre>
<code>s.find(t, b, e)</code>	Posición de la 1ª ocurrencia de <code>t</code> en <code>s[b:e]</code>	<pre>>>> s.find('en', 5, 8) -1 >>> s.find('en', 0, 3) -1</pre>
<code>s.upper()</code>	Devuelve una copia de <code>s</code> en mayúsculas	<pre>>>> s.upper() 'VALENCIA'</pre>
<code>s.lower()</code>	Devuelve una copia de <code>s</code> en minúsculas	<pre>>>> s.lower() 'valencia'</pre>
<code>s.startswith(t)</code>	Indica si <code>s</code> comienza por <code>t</code>	<pre>>>> s.startswith('Val') True</pre>
<code>s.endswith(t)</code>	Indica si <code>s</code> termina con <code>t</code>	<pre>>>> s.endswith('ia') True</pre>
<code>s.replace(u, v)</code>	Devuelve una copia de <code>s</code> con todas las apariciones de <code>u</code> reemplazadas por <code>v</code>	<pre>>>> s.replace('a', 'oo') 'Voolencioo'</pre>
<code>s.isdigit()</code>	Devuelve True si <code>s</code> consiste solo de dígitos 0 a 9	<pre>>>> 'Hola!'.isdigit() False >>> '123'.isdigit() True</pre>

1. Recorrido de cadenas

- Una propiedad interesante de los datos secuenciales es que pueden recorrerse de izquierda a derecha con un bucle *for*
 - Por ejemplo, los siguientes bucles recorren los caracteres de una cadena de izquierda a derecha y los imprimen cada uno en una línea:

```
a = 'Ejemplo'
for letra in a:
    print(letra)
```

E
j
e
m
p
l
o

```
a = 'Ejemplo'
for i in range(len(a)):
    print(a[i])
```

E
j
e
m
p
l
o

1. Recorrido de cadenas

- Si quisiéramos recorrer la cadena al revés, tendríamos únicamente la opción de usar bucles con rangos:

```
for i in range(len(a)-1, -1, -1):  
    print(a[i])
```

o
l
p
m
e
j
E

```
for i in range(len(a)):  
    print(a[len(a)-i-1])
```


o
l
p
m
e
j
E

1. Búsquedas en cadenas

- Las búsquedas en una cadena pueden realizarse mediante un recorrido, comparando los caracteres con los que buscamos
- Ejemplo: **busca espacios en una frase para averiguar el número de palabras que contiene**

```
frase = input('Escribe una frase: ').strip()
if (len(frase) > 0):
    npal = 1
    for i in range(len(frase)):
        if (frase[i] == ' '):
            npal += 1
else: # Cadena vacía
    npal = 0
print('La frase tiene', npal, 'palabras')
```

strip() elimina los espacios al principio y al final de una cadena



1. Búsquedas en cadenas

- Otro ejemplo: **cuenta las palabras en una frase, pero haciendo uso de la función *find***

```
frase = input('Escribe una frase: ').strip()
if (len(frase) > 0):
    npal = 1
    pos = frase.find(' ')          # Buscamos el 1er espacio
    while pos != -1:               # Si lo encontramos
        npal += 1                 # Incrementamos el número de palabras
        pos = frase.find(' ', pos + 1) # Buscamos el siguiente espacio
else: # Cadena vacía
    npal = 0
print('La frase tiene', npal, 'palabras')
```

2. Listas

- Una lista es una secuencia de valores de cualquier tipo
- Las listas se representan entre corchetes y separados por comas
- Ejemplos:

<code>[1, 2, 3]</code>	<i># Lista con los números del 1 al 3</i>
<code>[1, 1+1, 6//2]</code>	<i># Igual que la lista anterior</i>
<code>['Juan', 'Antonia', 'Luis', 'María']</code>	<i># Lista de nombres</i>
<code>[1]</code>	<i># Lista con un solo elemento</i>
<code>[]</code>	<i># Lista vacía (sin elementos)</i>
<code>['Juan', 5, 'Antonia', 7]</code>	<i># Lista mixta de cadenas y enteros</i>

2. Listas y cadenas

- Python proporciona operadores y funciones similares para trabajar con tipos de datos similares
 - Las cadenas y las listas tienen algo en común: ambas son secuencias de datos
- Muchos de los operadores y funciones que trabajan sobre cadenas también lo hacen sobre listas. Ejemplos:
 - La función *len*, aplicada sobre una lista, nos dice cuántos elementos la integran:

```
a = [1, 2, 3]  
len(a)
```

3

- El operador *+* para concatenar listas:

```
[1, 2] + [3, 4]
```

[1, 2, 3, 4]

2. Listas y cadenas

- El operador `*` para la repetición de listas:

```
>>> [1,2,3] * 3  
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

0	1	2	3	4	5	6	7	8	9	10	11
1	2	3	4	5	6	7	8	9	4	0	8
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

[6:10]

[-12:-7]

- El operador de indexación:

```
>>> a = [1,2,3,4,5,6,7,8,9,4,0,8]  
>>> print(a[1], a[len(a)-1], a[-4])  
2 8 9
```

- Así como el operador de corte:

```
>>> a = [1,2,3,4,5,6,7,8,9,4,0,8]  
>>> a[0:-2]  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 4]
```

La expresión `a[i:j]` devuelve la lista formada por los elementos desde `a[i]` hasta `a[j-1]`

2. Listas y cadenas

- Los operadores de comparación también trabajan con listas
 - Operadores de igualdad (==) y desigualdad (!=): dos listas son iguales si tienen la misma longitud y todos sus elementos son iguales y siguen en el mismo orden. Son distintas en caso contrario

```
[1, 2, 3] == [1, 2, 3]
```

True

```
[1, 2, 4] != [4, 2, 1]
```

True

- Los operadores <, >, <= y >= funcionan con listas del mismo modo que con las cadenas
 - Una lista, por ejemplo, es menor que otra si el primer elemento que es distinto en ambas listas, recorriendo de izquierda a derecha, es menor en la primera lista

```
[1, 2, 3, 4] < [1, 2, 4, 0]
```

True

```
[3, 2, 1] >= [3, 3]
```

False

2. Conversión de cadenas a listas y viceversa

- El método `split()` de una cadena devuelve una lista con las palabras que contiene:

```
a = 'Estoy programando en Python'  
a.split()
```

```
['Estoy', 'programando', 'en', 'Python']
```

- El método `split()` acepta un parámetro opcional: el carácter de separación de palabras que, por defecto, es el espacio

```
a = 'Conversión-de-cadena-a-lista'  
a.split('-')
```

```
['Conversión', 'de', 'cadena', 'a', 'lista']
```

2. Conversión de cadenas a listas y viceversa

- La función predefinida `list(cadena)` también convierte una cadena en una lista, pero formada por los caracteres individuales de la cadena:

```
list('Uno y dos')
```

```
['U', 'n', 'o', ' ', 'y', ' ', 'd', 'o', 's']
```

- El método `cadena.join(lista)` hace lo contrario que `split()`: une los elementos de la lista, separándolos mediante la cadena indicada, en una nueva cadena:

```
'-'.join(['Tengo', '2', 'manitas'])
```

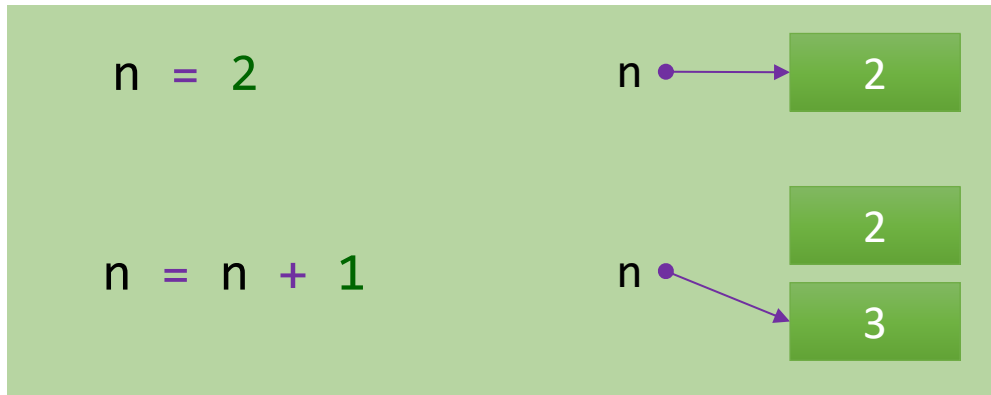
```
'Tengo-2-manitas'
```

2. Funciones y métodos de manejo de listas/cadenas

Función	Descripción	Ejemplo
<code>list(s)</code>	Convertir una cadena a un lista de sus caracteres.	<pre>>>> list('Hi Py!') ['H', 'i', ' ', 'P', 'y', '!']</pre>
Métodos	Descripción	Ejemplo
<code>s.split()</code>	Crear una lista de las palabras separado por un espacio de una cadena.	<pre>>>> "Hi Py!".split() ['Hi', 'Py!']</pre>
<code>s.split(c)</code>	Crear una lista de las palabras separado por c de una cadena.	<pre>>>> "Hi Py!".split('i') ['H', ' Py!'] >>> "Hi Py!".split('P') ['Hi ', 'y!']</pre>
<code>s.join(lista)</code>	<i>Crear una cadena con los elementos de una lista usando s como separador</i>	<pre>>>> "*".join(['a','b','c']) 'a*b*c'</pre>
<code>s.strip()</code>	Eliminar los espacios al principio y al final de una cadena	<pre>>>> " Hello Py! ".strip() 'Hello Py! '</pre>

2. Mutabilidad e inmutabilidad

- Los tipos básicos en Python (`int`, `float`, `bool`) son **inmutables**: su valor no puede ser modificado
- Mira lo que ocurre en memoria al incrementar una variable `n`:



Python no modifica el valor `2` con un `3`. En su lugar, crea un nuevo valor, el `3`, y el antiguo será eliminado ya que no se utiliza.

- Las cadenas también son inmutables
 - Un intento de modificar algún carácter provocará un error:

```
a = 'Esta cadena es inmutable'  
a[0] = 'e'
```

TypeError

Traceback (most recent call last)

2. Mutabilidad e inmutabilidad

- Las listas, por el contrario, son **mutables**: podemos modificar su contenido
 - Podemos asignar valores a elementos particulares de una lista gracias al operador de indexación:

```
a = [10, 20, 30]  
a[1] = 25  
print(a)
```

```
[10, 25, 30]
```



- Es importante a partir de ahora distinguir entre métodos que *modifican el contenido* de una lista (o cualquier otro objeto mutable) y funciones que devuelven una *nueva copia* de la lista con alguna modificación

2. Mutabilidad e inmutabilidad

- Ejemplo:

- La **función** `sorted(lista)` devuelve una *nueva lista* con los elementos de la lista que recibe como parámetro ordenados:

```
a = [9, 4, 3, 7, 1]
sorted(a)
```

```
[1, 3, 4, 7, 9]
```

- Se puede comprobar que la lista original no se ha modificado:

```
print(a)
```

```
[9, 4, 3, 7, 1]
```

- El **método** `lista.sort()` ordena los elementos de la lista:

```
a = [9, 4, 3, 7, 1]
a.sort()
print(a)
```

```
[1, 3, 4, 7, 9]
```

- No crea ninguna lista nueva, simplemente reordena los elementos sobre la propia lista

Compara

lista1 → [4, 6, 1, 2, 10, -3]

lista2 → [-3, 1, 2, 4, 6, 10]

```
>>> lista1 = [4,6,1,2,10,-3]
>>> lista2 = sorted(lista1)
>>> lista1 is lista2
False
>>> lista1
[4, 6, 1, 2, 10, -3]
>>> lista2
[-3, 1, 2, 4, 6, 10]
```

lista1 → [-3, 1, 2, 4, 6, 10]

lista2 →

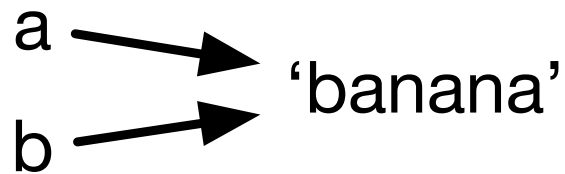
no devuelve nada

```
>>> lista1 = [4,6,1,2,10,-3]
>>> lista2 = lista1.sort()
>>> lista1 is lista2
False
>>> lista1
[-3, 1, 2, 4, 6, 10]
>>> lista2
```


Objetos, valores e igualdad

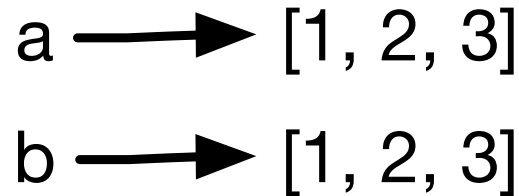
Strings

```
>>> a = "banana"
>>> b = "banana"
>>> a is b
True
>>> a == b
True
```



Lists

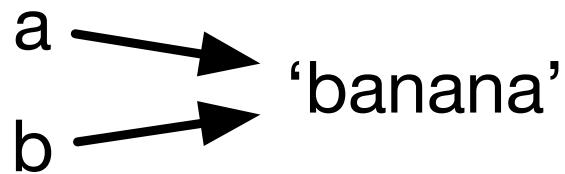
```
>>> a = [1,2,3]
>>> b = [1,2,3]
>>> a is b
False
>>> a == b
True
```



Objetos, valores e igualdad

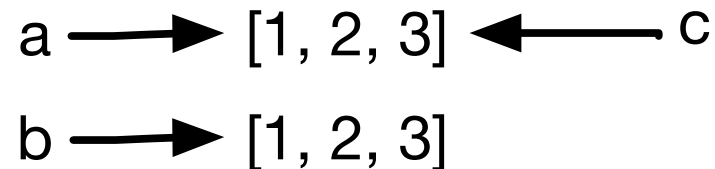
Strings

```
>>> a = "banana"
>>> b = "banana"
>>> a is b
True
>>> a == b
True
```



Lists

```
>>> a = [1,2,3]
>>> b = [1,2,3]
>>> a is b
False
>>> a == b
True
```



```
>>> c = a
>>> c is a
True
>>> c == a
True
```

Hay que mirar bien, ¿la misma referencia o no?

lista1 → [7,8,9]

lista2 → [7,8,9]

```
>>> lista1 = [7,8,9]
>>> lista2 = lista1[:]
>>> lista1 is lista2
```

False

```
>>> lista1 == lista2
```

True

```
>>> lista1.append(10)
```

```
>>> lista1
```

[7, 8, 9, 10]

```
>>> lista2
```

[7, 8, 9]

hace
una
copia

lista1 → [7,8,9]

lista2 →

```
>>> lista1 = [7,8,9]
>>> lista2 = lista1
>>> lista1 is lista2
```

True

```
>>> lista1 == lista2
```

True

```
>>> lista1.append(10)
```

```
>>> lista1
```

[7, 8, 9, 10]

```
>>> lista2
```

[7, 8, 9, 10]

misma
referencia

2. Adición de elementos a una lista

- El método `append(e)` añade el elemento `e` al final de la lista:

```
a = [1, 2, 3]
a.append(4)
print(a)
```

[1, 2, 3, 4]

- Usar este método es más eficiente que la concatenación, ya que el operador `+` crea una nueva lista (copiando todos los elementos de las listas a concatenar a la nueva):

```
a = [1, 2, 3]
a = a + [4]
print(a)
```

[1, 2, 3, 4]

2. Borrado de elementos de una lista

- Para borrar elementos de una lista podemos usar la sentencia `del`:

```
a = [1, 2, 3, 1, 4]
del a[0]      # Eliminamos el 1er elemento de la lista
print(a)
```

[2, 3, 1, 4]

- El borrado de elementos también funciona con rangos:

```
del a[1:3]    # Eliminamos el 2º y 3er elemento
print(a)
```

[2, 4]

- La sentencia `del` no produce una copia de la lista sin la celda borrada, sino que modifica directamente la lista sobre la que opera
- `del` es un keyword no es una función/método

2. Pertenencia de un elemento a una lista

- La pregunta de si un elemento pertenece o no a una lista es tan frecuente que Python proporciona el operador binario `in`
 - El operador `in` recibe un elemento por su parte izquierda y una lista por su parte derecha y devuelve cierto o falso:

```
3 in [1, 2, 3, 4]
```

True

```
5 in [1, 2, 3, 4]
```

False

- El operador «`not in`» es el operador `in` negado

```
5 not in [1, 2, 3, 4]
```

True

2. Recorridos de listas

- El iterador `for` permite recorrer los elementos de una lista:

```
a = [1, 2, 3]
for i in a:
    print(i)
```

1
2
3

```
a = [1, 2, 3]
for i in range(len(a)):
    print(a[i])
```

1
2
3

- La función `list` permite también construir fácilmente una lista a partir de un rango:

```
a = list(range(1, 10))
print(a)
```

[1, 2, 3, 4, 5, 6, 7, 8, 9]

2. Búsquedas en listas

- El método `index` nos devuelve el índice de la primera aparición de un elemento en una lista:

```
a = ['Pepe', 'Luis', 'María', 'Anna']  
a.index('María')
```

2

- Lanza la excepción `ValueError` si el elemento no está en la lista

2. Búsquedas en listas

- Para búsquedas más complejas, necesitamos un recorrido
- Ejemplo: buscar el primer número negativo de una lista

```
a = [1, 2, -1, -4, 5, 2]
i = 0
```

```
while (i < len(a) and a[i] >= 0):
    i += 1
```

```
if i < len(a):
    print('Es %d y está en la posición %d.' % (a[i], i))
else:
    print('No hay ningún número negativo.')
```

2. Listas anidadas

- Las listas pueden contener cualquier tipo de elemento. Es posible, por lo tanto, definir una lista cuyos elementos son, a su vez, listas
- Ejemplo:

```
m = [[1, 2, 3], [2, 12, 6], [1, 0, -3], [0, -1, 0]]
```

- El primer elemento de la lista m es la lista [1, 2, 3], el segundo la lista [2, 12, 6], el tercero la [1, 0, -3] y el cuarto la [0, -1, 0]

```
m[0]
```

```
[1, 2, 3]
```

- La longitud de m es, por lo tanto, 4

```
len(m)
```

```
4
```

2. Listas anidadas

- Si queremos acceder, por ejemplo, al valor 3, deberemos acceder en primer lugar al primer elemento de m , la lista $[1, 2, 3]$, y luego al tercer elemento de dicha lista:

```
m = [[1, 2, 3], [2, 12, 6], [1, 0, -3], [0, -1, 0]]  
m[0]  # primer elemento de m
```

```
[1, 2, 3]
```

```
m[0][2]  # tercer elemento del primer elemento de m
```

```
3
```

- Podemos considerar una lista de listas como una matriz
 - En notación matemática, el elemento que ocupa la fila i -ésima y la columna j -ésima de una matriz M se representa con $M_{i,j}$
 - En Python se accede mediante $m[i-1][j-1]$, ya que los índices comienzan en cero

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 12 & 6 \\ 1 & 0 & -3 \\ 0 & -1 & 0 \end{bmatrix}$$

2. Listas anidadas

- *Ejemplo:* construcción de una matriz nula (cuyos elementos son todos igual a cero) de tamaño 3x6:

```
m = [] # creamos una lista vacía
for i in range(3): # Le añadimos sus tres componentes mediante un bucle
    m.append([0] * 6) # cada componente es una lista con seis ceros
print(m)
```

```
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
```

- Con `len(m)` podemos averiguar el número de filas de la matriz:

```
len(m)
```

3

- Para saber el número de columnas podemos preguntar la longitud de cualquiera de los elementos de `m`:

```
len(m[0])
```

6

3. Operaciones sobre secuencias

s puede ser un string, una lista,

Operación	Resultado
x in s	True si un elemento de s es igual a x, sino False
x not in s	False si un elemento de s es igual a x, sino True
s + t	La concatenación de s y t
s * n, n * s	Añadir s, n veces (n entero), delante y detrás
s[i]	i-esimo elemento de s
s[i:j]	Trozo de s de i a j-1
s[i:j:k]	Trozo de s de i a j-1 con step k
len(s)	Longitud de s
sum(s)	Suma los elementos de s
min(s)	El elemento más pequeño de s
max(s)	El elemento más grande de s
s.index(x)	Índice en la secuencia de la primera ocurrencia de x en s
s.count(x)	Número de ocurrencias de x en s

¡Solo hay una forma de aprender a programar!



practicando, practicando, practicando, practicando