

NIVEL 3

LISTAS - INTRODUCCIÓN



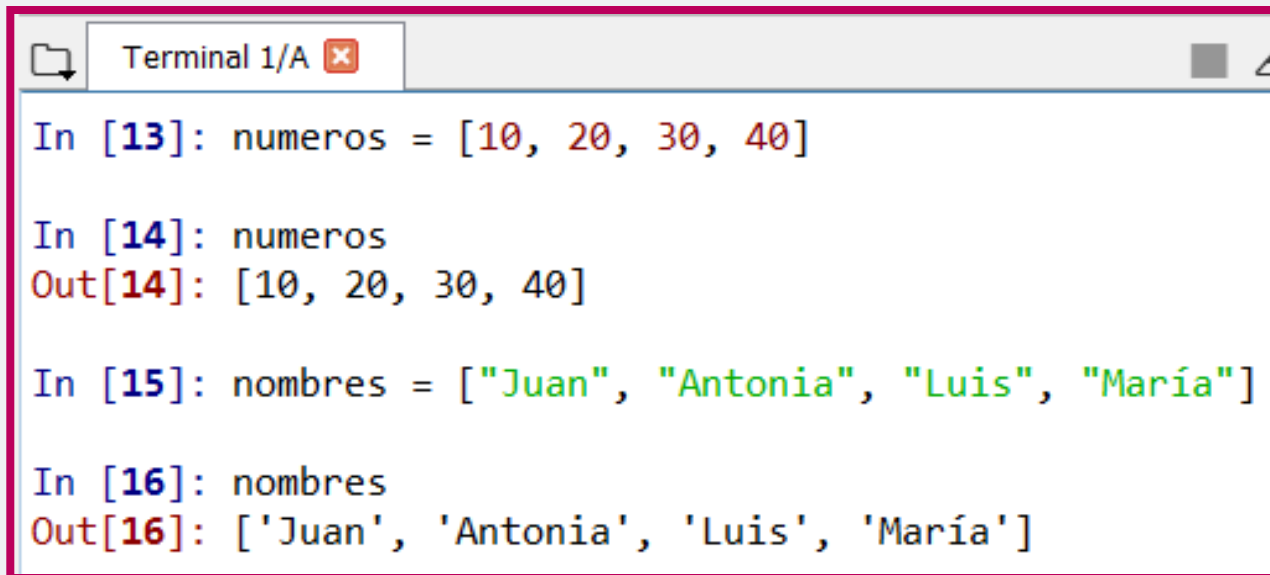


LISTAS

- ✓ Son colecciones ordenada de valores y es un tipo de Python: **list**
- ✓ Los valores que conforman una **lista** son llamados **elementos** o **ítems**
- ✓ Las **listas** son similares a los strings, es decir, colecciones ordenadas de caracteres. Se diferencian en que los elementos de una **lista** pueden ser de cualquier tipo (enteros, flotantes, o incluso de cadenas)
- ✓ Las **listas** y los strings (y otras colecciones que mantienen el orden de sus ítems) son llamados **secuencias**
- ✓ En una **lista** podemos registrar, entre muchos otros:
 - las notas de los estudiantes de una clase,
 - los coeficientes de un polinomio
 - la evolución de la temperatura hora a hora
 - la relación de nombres de personas asistentes a una reunión

CREANDO UNA LISTA

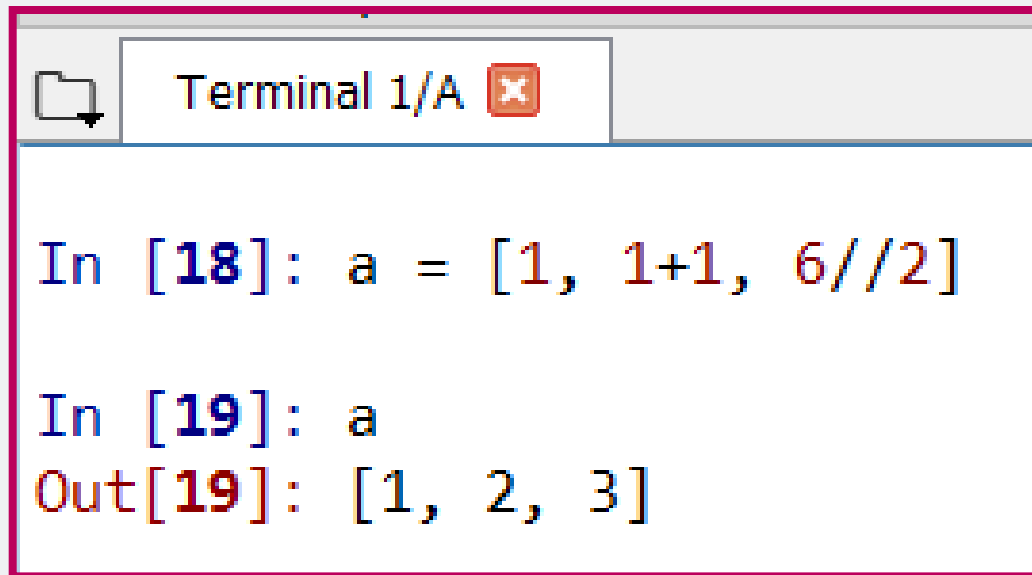
Hay varias formas de crear una lista; la más simple es encerrando los elementos dentro de corchetes cuadrados:



```
Terminal 1/A x  
In [13]: numeros = [10, 20, 30, 40]  
  
In [14]: numeros  
Out[14]: [10, 20, 30, 40]  
  
In [15]: nombres = ["Juan", "Antonia", "Luis", "María"]  
  
In [16]: nombres  
Out[16]: ['Juan', 'Antonia', 'Luis', 'María']
```

CREANDO UNA LISTA

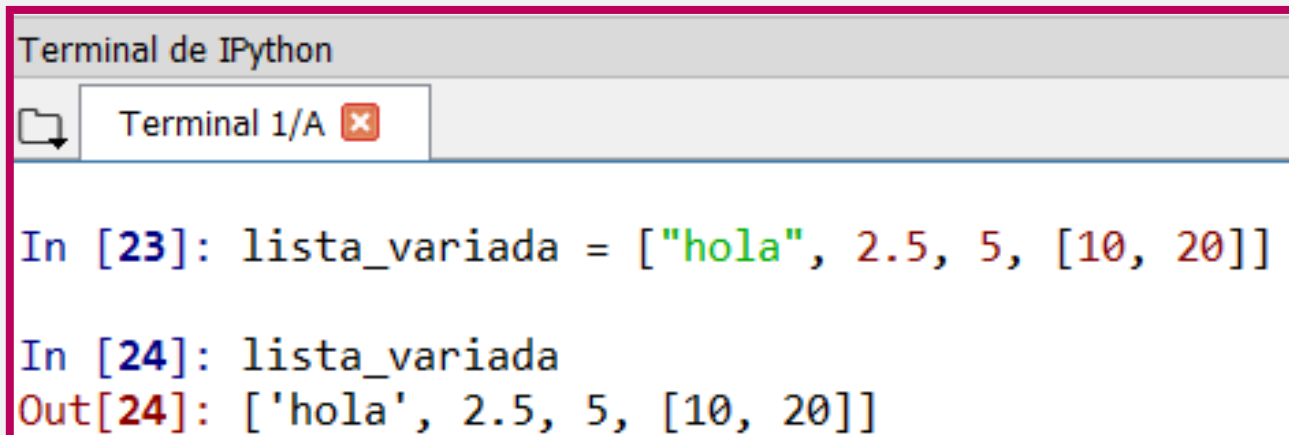
También podemos usar expresiones para calcular el valor de cada elemento de una lista:



```
Terminal 1/A ✕  
  
In [18]: a = [1, 1+1, 6//2]  
  
In [19]: a  
Out[19]: [1, 2, 3]
```

VALORES DE UNA LISTA

- Una lista sin elementos es llamada una lista vacía y se denota []
- Los elementos de una lista no tienen que ser del mismo tipo:



Terminal de IPython

Terminal 1/A

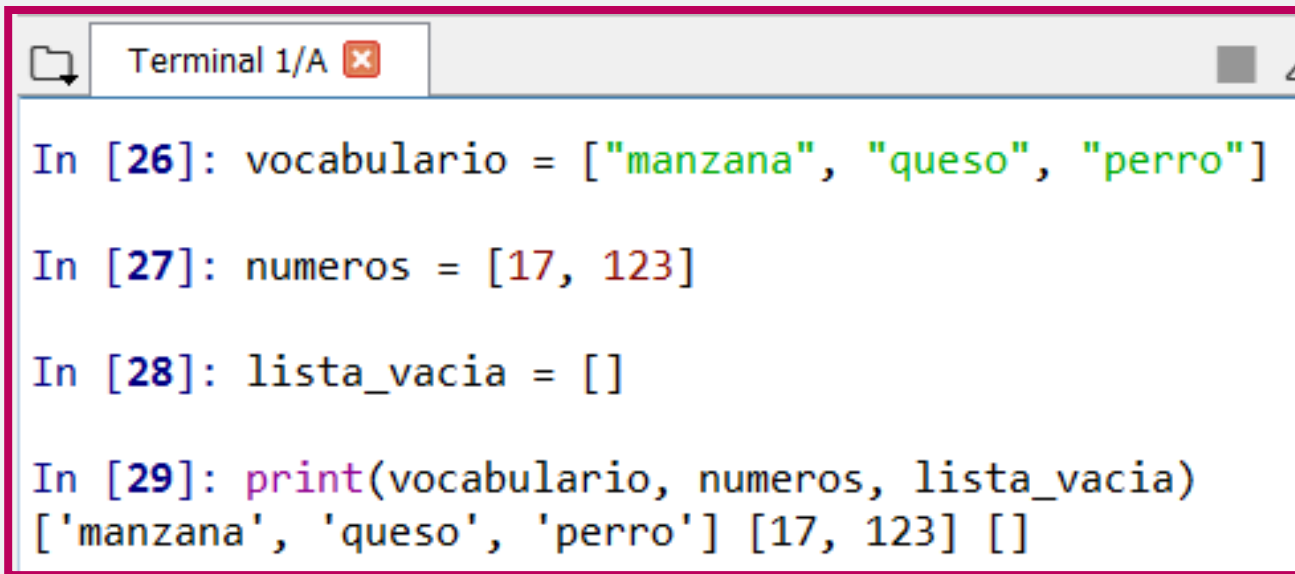
```
In [23]: lista_variada = ["hola", 2.5, 5, [10, 20]]  
  
In [24]: lista_variada  
Out[24]: ['hola', 2.5, 5, [10, 20]]
```

A blue line points from the nested list `[10, 20]` in the output to the text below.

Un elemento de una lista
puede ser incluso una lista

LISTAS COMO PARÁMETROS

Podemos pasar listas como parámetros a una función, por ejemplo a la función print:

A screenshot of a terminal window titled "Terminal 1/A". The window contains four lines of Python code and their corresponding output. The first line defines a list of strings, the second a list of integers, the third an empty list, and the fourth prints all three lists. The output shows the lists as they would appear in the console.

```
In [26]: vocabulario = ["manzana", "queso", "perro"]  
In [27]: numeros = [17, 123]  
In [28]: lista_vacia = []  
In [29]: print(vocabulario, numeros, lista_vacia)  
['manzana', 'queso', 'perro'] [17, 123] []
```

LO QUE YA SABEMOS SIN SABERLO

- Python proporciona operadores y funciones similares para trabajar con tipos de datos similares
- Dado que 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

- La función `len`, aplicada sobre una lista, nos dice cuántos elementos la integran:

La longitud de una lista vacía es cero

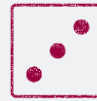
```
Terminal 1/A x
In [34]: a = [1, 2, 3]
In [35]: len(a)
Out[35]: 3
In [36]: len([])
Out[36]: 0
```

- El operador `+` concatena listas:

```
Terminal 1/A x
In [38]: a = [1, 2]
In [39]: b = [3, 4]
In [40]: c = a + b
In [41]: c
Out[41]: [1, 2, 3, 4]
In [42]: [5, 6] + [7, 8]
Out[42]: [5, 6, 7, 8]
In [43]: [10, 20] + a
Out[43]: [10, 20, 1, 2]
```

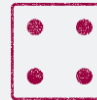
LO QUE YA SABEMOS SIN SABERLO

- Python proporciona operadores y funciones similares para trabajar con tipos de datos similares
- Dado que 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



El operador de indexación `[]` aplica a las listas:

```
Terminal 2/A x
In [2]: a = [1, 2, 3]
In [3]: a[1]
Out[3]: 2
In [4]: a[len(a)-1]
Out[4]: 3
In [5]: a[-1]
Out[5]: 3
In [6]: [1, 2, 3][0]
Out[6]: 1
```

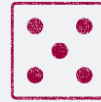


El operador de corte `[n:m]` aplica a las listas, para extraer «pedazos» (slices) de estas:

```
Terminal 2/A x
In [15]: a = [1, 2, 3, 4, 5]
In [16]: a[1:3]
Out[16]: [2, 3]
In [17]: a[1:-1]
Out[17]: [2, 3, 4]
In [18]: a[1:]
Out[18]: [2, 3, 4, 5]
```


LO QUE YA SABEMOS SIN SABERLO

- Python proporciona operadores y funciones similares para trabajar con tipos de datos similares
- Dado que 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



El iterador **for-in** también recorre los elementos de una lista:

```
Terminal 2/A x
In [24]: for i in [1, 2, 3]:
...:     print(i)
...:
1
2
3
```



Las funciones **min** y **max** sirven sobre listas:

```
Terminal 7/A x
In [2]: numeros = [12, 3, 8, 32, 5, 34, 6, 2, 87, 22]

In [3]: min(numeros)
Out[3]: 2

In [4]: max(numeros)
Out[4]: 87
```

OTRAS FORMAS DE CONSTRUIR LISTAS

Combinando las funciones `list` y `range`

```
Terminal 2/A x
```

```
In [26]: a = list(range(1, 4))
```

```
In [27]: print(a)
```

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

Usando el operador `*`

Una forma corriente de construir listas que contienen réplicas de un mismo valor es con el operador `*`. Supongamos que necesitamos una lista de 10 elementos, todos los cuales valen 0

```
Terminal de IPython
```

```
Terminal 2/A x
```

```
In [28]: [0] * 10
```

```
Out[28]: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```