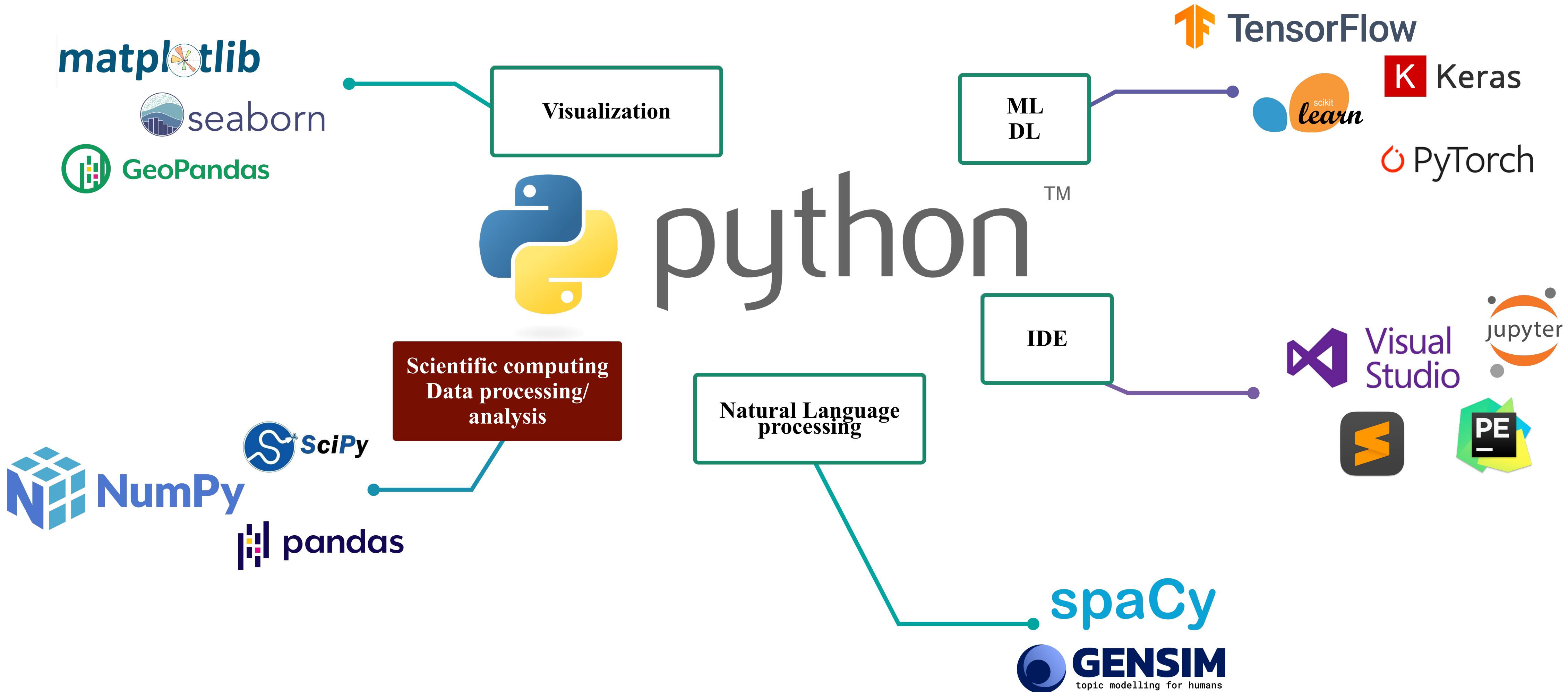




Tien-Lam Pham
Ph-D
Faculty of Computer Science
Van-Quyen Nguyen

1. INTRODUCTION

- Numpy is a Python library for scientific computations



2. COMMON ATTRIBUTES

- dtype**
- nbytes**
- ndim**
- shape**
- size**

Data type

Number of bytes used to store the data

Number of dimensions

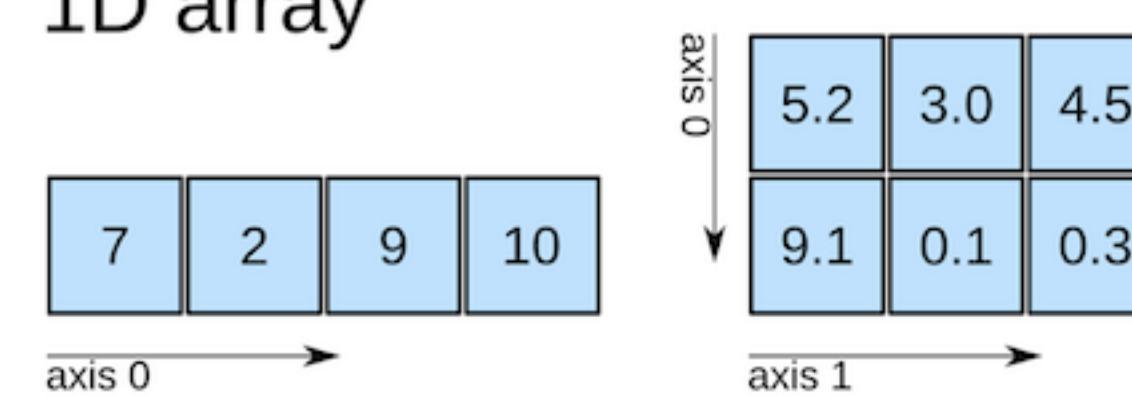
A tuple of number of elements in each dimension

Total number elements

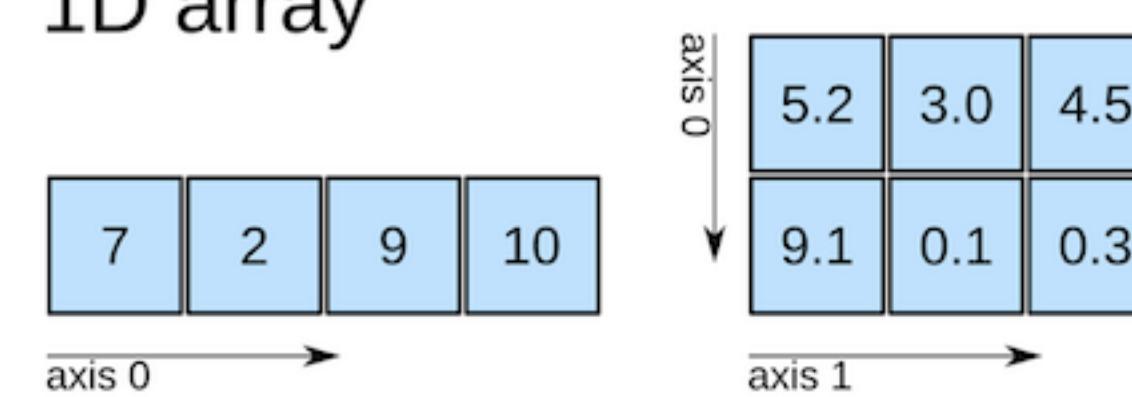
```
In [2]: import numpy as np
...
...: data = np.array([7, 2, 9, 10])
...
...: print(data)
...: print(data.shape)
...: print(data.size)
...: print(data.dtype)
...: print(data.nbytes)
...: print(data.ndim)
[7 2 9 10]
(4,)
4
int64
32
1

In [3]:
```

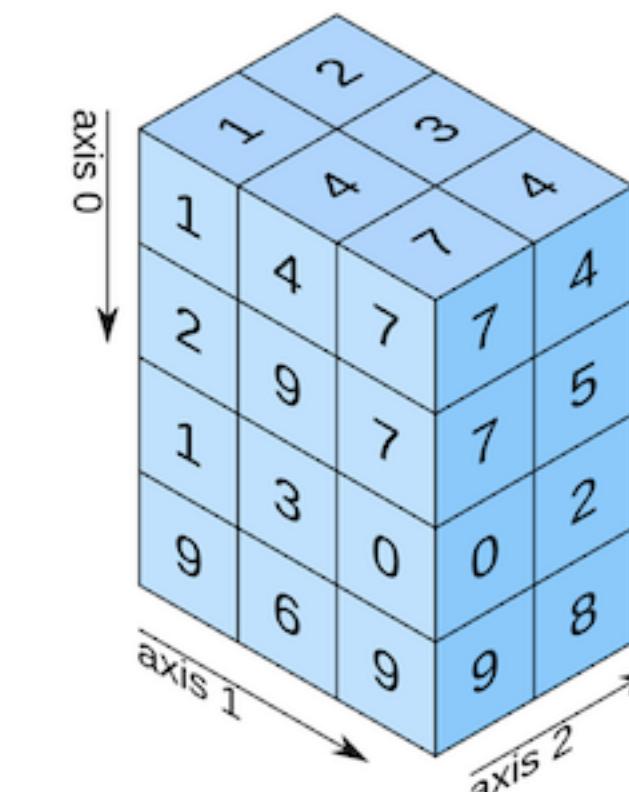
1D array



2D array



3D array

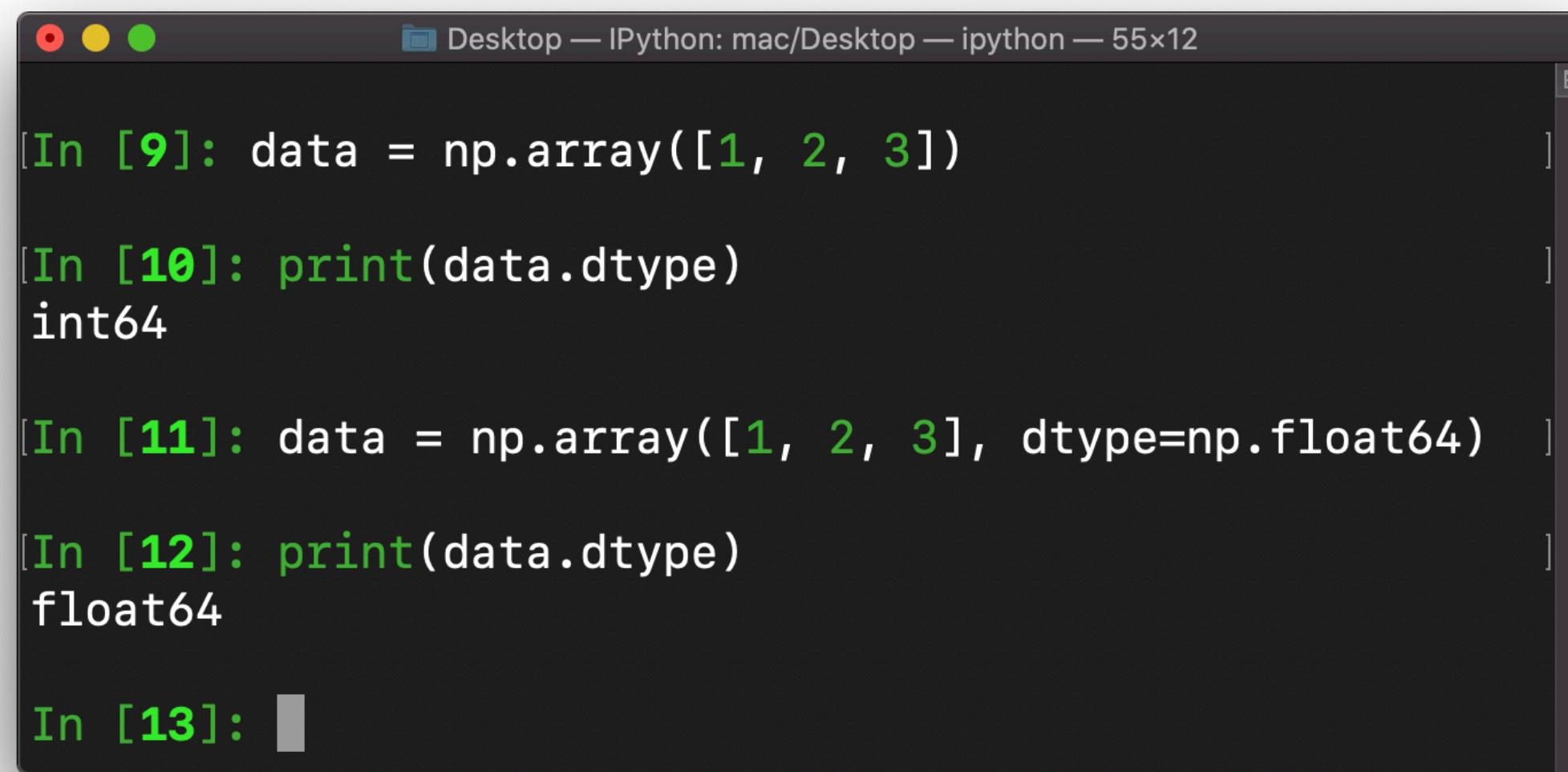


```
In [3]: data = np.array([[5.2, 3.0, 4.5],
...:                     [9.1, 0.1, 0.3]])
...: print(data)
...: print(data.shape)
...: print(data.ndim)
[[5.2 3.  4.5]
 [9.1 0.1 0.3]]
(2, 3)
2

In [4]:
```

2. COMMON ATTRIBUTES

○ **dtype** Data type



```
[In [9]: data = np.array([1, 2, 3])
[In [10]: print(data.dtype)
int64
[In [11]: data = np.array([1, 2, 3], dtype=np.float64)
[In [12]: print(data.dtype)
float64
In [13]: ]
```

dtype	Variants	Description
int	int8, int16, int32, int64, ...	Integers
uint	uint8, uint16, uint64, ...	Unsigned integers
bool	Bool	Boolean
float	float16, float32, float64, ...	Floating-point numbers
complex	complex64, complex128, ...	Complex-valued float number

3. UPDATE AN ELEMENT

index 0 1 2

data =

1	2	3
---	---	---

data[0] = 8

data =

8	2	3
---	---	---

```
Desktop — IPython: mac/Desktop — ipython — 55x12

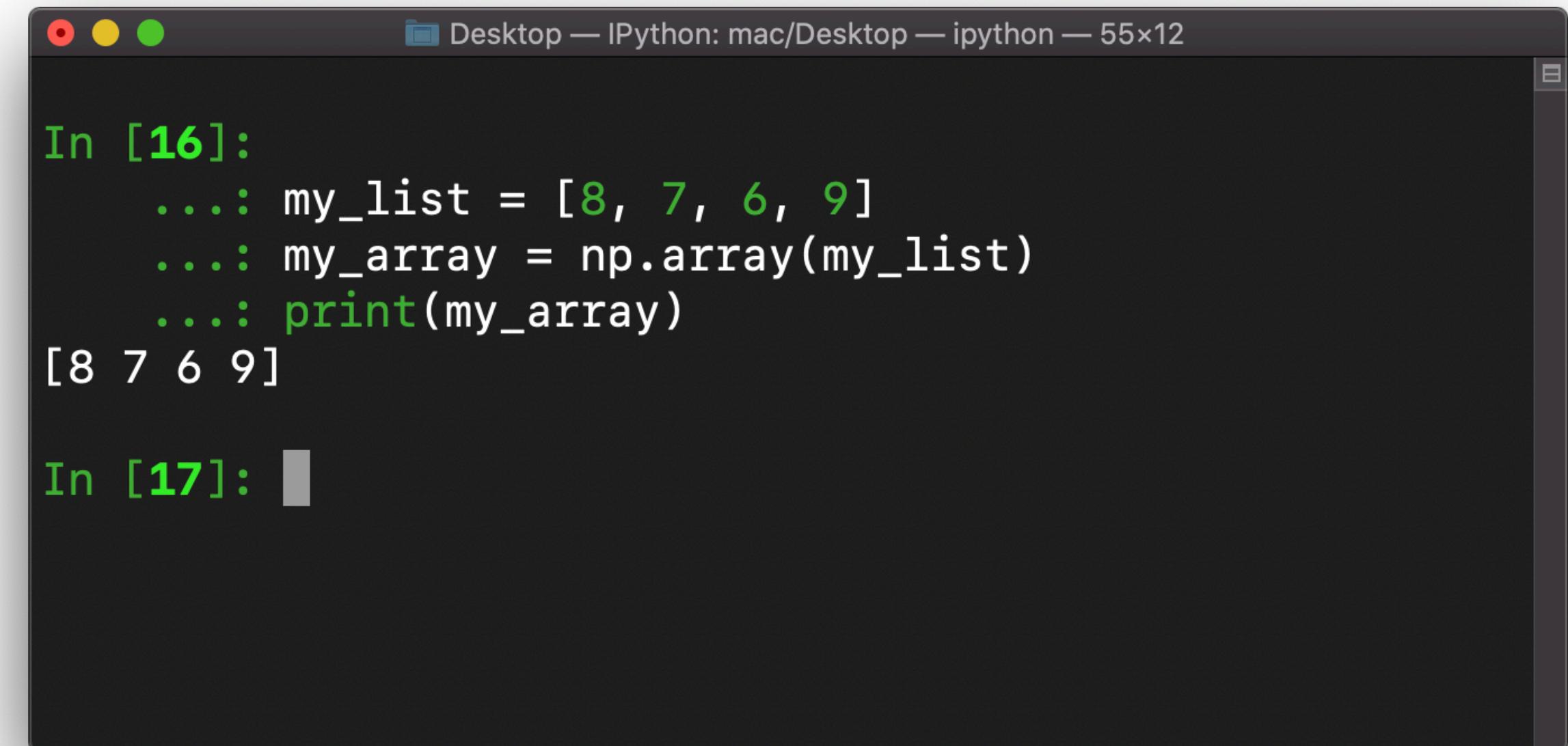
In [14]: data = np.array([1, 2, 3])
...: print(data)
...: data[0] = 8
...: print(data)
[1 2 3]
[8 2 3]

In [15]:
```

4. CREATE NUMPY ARRAY

- From list

arr_np = np.array(python_list)



The screenshot shows a terminal window titled "Desktop — IPython: mac/Desktop — ipython — 55x12". It displays two code cells. Cell In [16] contains the following Python code:

```
In [16]:  
...: my_list = [8, 7, 6, 9]  
...: my_array = np.array(my_list)  
...: print(my_array)  
[8 7 6 9]
```

Cell In [17] is currently empty, indicated by a small gray square icon.

4. CREATE NUMPY ARRAY

○ From common functions

○ zeros()

0	0	0
0	0	0

○ ones()

1	1	1
1	1	1

○ full()

8	8	8
8	8	8

```
Desktop — IPython: mac/Desktop — ipython — 37x9

In [23]: arr = np.zeros((2, 3))
...: print(arr)
[[0. 0. 0.]
 [0. 0. 0.]]

In [24]:
```

```
Desktop — IPython: mac/Desktop — ipython — 37x9

In [26]: arr = np.ones((2, 3))
...: print(arr)
[[1. 1. 1.]
 [1. 1. 1.]]

In [27]:
```

```
Desktop — IPython: mac/Desktop — ipython — 37x9

In [30]: arr = np.full((2, 3), 8)
...: print(arr)
[[8 8 8]
 [8 8 8]]

In [31]:
```

4. CREATE NUMPY ARRAY

- From common functions

- arange()

```
arr1 = 

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
|---|---|---|---|


```

```
arr2 = 

|   |   |    |
|---|---|----|
| 0 | 5 | 10 |
|---|---|----|


```

```
In [32]: arr1 = np.arange(5)
...: print(arr1)
...:
...: arr2 = np.arange(0, 11, 5)
...: print(arr2)
[0 1 2 3 4]
[ 0  5 10]

In [33]:
```

- eye()

1	0	0
0	1	0
0	0	1

```
In [34]: arr = np.eye(3)
...: print(arr)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

In [35]:
```

- random()

```
[In 36]: arr = np.random.random((2,3))
...:
...: print(arr)
[[0.29766378 0.05093395 0.39236048]
 [0.79031391 0.47018842 0.02838079]]

In [37]:
```

4. CREATE NUMPY ARRAY

○ numpy.hstack()

arr1 =

0	1	2	3
---	---	---	---

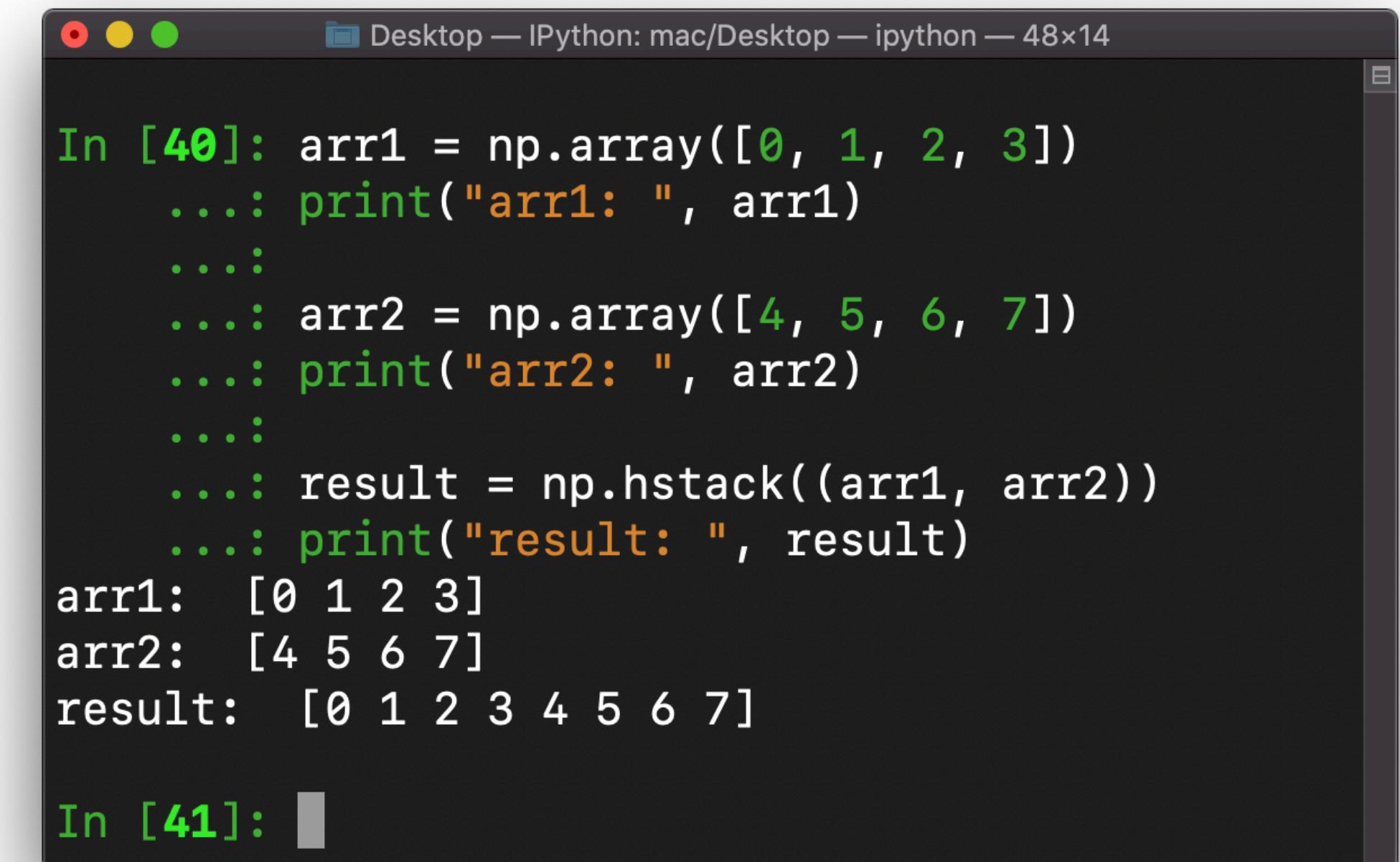
arr2 =

4	5	6	7
---	---	---	---

hstack((arr1, arr2))

result =

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---



```
In [40]: arr1 = np.array([0, 1, 2, 3])
...: print("arr1: ", arr1)
...:
...: arr2 = np.array([4, 5, 6, 7])
...: print("arr2: ", arr2)
...:
...: result = np.hstack((arr1, arr2))
...: print("result: ", result)
arr1: [0 1 2 3]
arr2: [4 5 6 7]
result: [0 1 2 3 4 5 6 7]

In [41]:
```

4. CREATE NUMPY ARRAY

○ numpy.vstack()

arr1 =

0	1	2	3
---	---	---	---

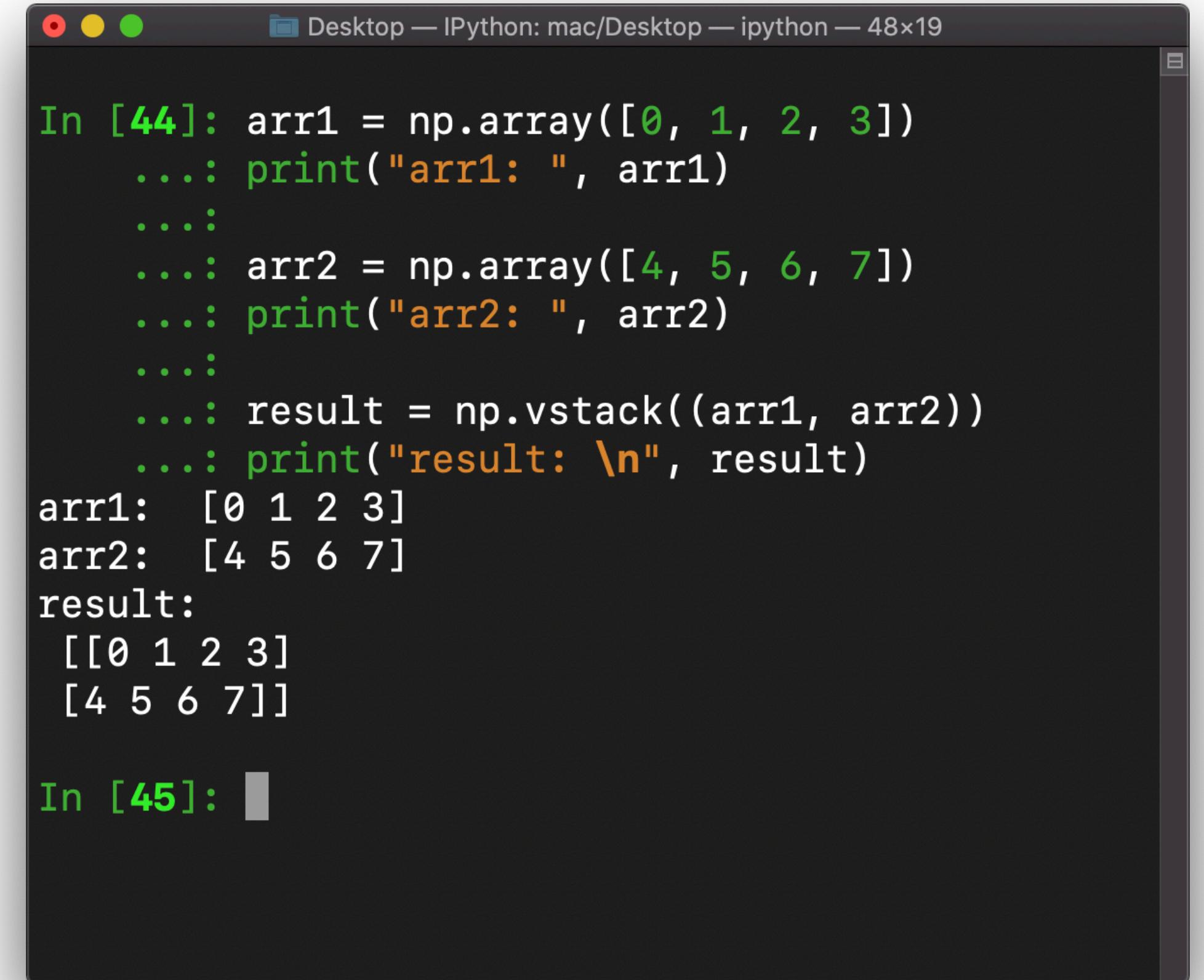
arr2 =

4	5	6	7
---	---	---	---

vstack([arr1, arr2])

result =

0	1	2	3
4	5	6	7



```
In [44]: arr1 = np.array([0, 1, 2, 3])
...: print("arr1: ", arr1)
...:
...: arr2 = np.array([4, 5, 6, 7])
...: print("arr2: ", arr2)
...:
...: result = np.vstack((arr1, arr2))
...: print("result: \n", result)
arr1: [0 1 2 3]
arr2: [4 5 6 7]
result:
[[0 1 2 3]
 [4 5 6 7]]

In [45]:
```

4. CREATE NUMPY ARRAY

○ numpy.concatenate()

arr1 =

0	1	2	3
---	---	---	---

arr2 =

4	5	6	7
---	---	---	---

concatenate([arr1, arr2], axis = 0)

result =

0	1	2	3
4	5	6	7

concatenate([arr1, arr2], axis = 1)

result =

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

```
In [52]: arr1 = np.array([[0, 1, 2, 3]])
....: print("arr1: ", arr1)
....:
....: arr2 = np.array([[4, 5, 6, 7]])
....: print("arr2: ", arr2)
....:
....: result1 = np.concatenate((arr1, arr2),
....: axis=0)
....: print("result1: \n", result1)
....:
....:
....: result2 = np.concatenate((arr1, arr2),
....: axis=1)
....: print("result2: \n", result2)
arr1: [[0 1 2 3]]
arr2: [[4 5 6 7]]
result1:
[[0 1 2 3]
 [4 5 6 7]]
result2:
[[0 1 2 3 4 5 6 7]]

In [53]:
```

4. CREATE NUMPY ARRAY

○ **numpy.array_split()** **numpy.split()**

1	2	3	4	5	6
---	---	---	---	---	---

`array_split(arr, 3)`

`newarr[0]`

1	2
---	---

`newarr[1]`

3	4
---	---

`newarr[2]`

5	6
---	---

```
In [101]: arr = np.array([1, 2, 3, 4, 5, 6])
.....
....: newarr = np.array_split(arr, 3)
....:
....: print(newarr[0])
....: print(newarr[1])
....: print(newarr[2])
[1 2]
[3 4]
[5 6]

In [102]:
```

4. CREATE NUMPY ARRAY

○ where() function

arr =

0	1	2	3
---	---	---	---

arr<2 =

T	T	F	F
---	---	---	---

out =

0	1	4	6
---	---	---	---

```
Desktop — IPython: mac/Desktop — ipython — 48x16

In [56]: arr = np.array([[0, 1, 2, 3]])
...: condition = arr < 2
...: print(condition)
...:
...: out = np.where(condition, arr, arr**2)
...: print(out)
[[ True  True False False]]
[[0 1 4 9]]

In [57]:
```

○ flatten() function

arr =

0	1
4	5

out =

0	1	4	5
---	---	---	---

```
Desktop — IPython: mac/Desktop — ipython — 48x16

In [58]: arr = np.array([[0, 1], [4, 5]])
...: out = arr.flatten()
...: print(arr)
...: print(out)
[[0 1]
 [4 5]]
[0 1 4 5]

In [59]:
```

4. CREATE NUMPY ARRAY

○ reshape() function

arr =

0	1	2	3
4	5	6	7

out =

0	1
2	3
4	5
6	7

```
In [64]: arr = np.array([[0, 1, 2, 3], [4, 5, 6,
....: 7]])
....: print(arr)
....: print("shape: ",arr.shape)
....:
....: out = np.reshape(arr, (4,2))
....: print(out)
....: print("shape: ",out.shape)
[[0 1 2 3]
 [4 5 6 7]]
shape: (2, 4)
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
shape: (4, 2)

In [65]:
```

5. ARRAY INDEXING

arr[for_axis_0, for_axis_1, ...]

- “:” **get all the elements**
- “a:b” **get the elements from a^{th} to $(b^{th}-1)$**

arr	0	1
0	0	1
1	2	3
2	4	5
3	6	7

arr[0,1]	0	1
0	0	1
1	2	3
2	4	5
3	6	7

arr[1:3]	0	1
0	0	1
1	2	3
2	4	5
3	6	7

arr[0:2, 0]	0	1
0	0	1
1	2	3
2	4	5
3	6	7

arr[:, 1:]	0	1
0	0	1
1	2	3
2	4	5
3	6	7

5. ARRAY INDEXING

Excercise

5. ARRAY INDEXING

Excercise

5. ARRAY INDEXING

Excercise

5. NUMPY ARRAY OPERATION

○ Addition

arr1

0	1	2	3
---	---	---	---

+

arr2

4	5	6	7
---	---	---	---

=

4	6	8	10
---	---	---	----

○ Subtraction

arr1

4	5	3	0
---	---	---	---

-

arr2

0	1	2	3
---	---	---	---

=

4	4	1	-3
---	---	---	----

```
Desktop — IPython: mac/Desktop — ipython — 48x14

In [69]: arr1 = np.array([[0, 1, 2, 3]])
....: arr2 = np.array([[4, 5, 6, 7]])
....:
....:
....: out = np.add(arr1, arr2)
....: print("result1: \n", out)
result1:
[[ 4  6  8 10]]

In [70]:
```

```
Desktop — IPython: mac/Desktop — ipython — 48x14

In [72]: arr1 = np.array([[4, 5, 3, 0]])
....: arr2 = np.array([[0, 1, 2, 3]])
....:
....:
....: out = np.subtract(arr1, arr2)
....: print("result1: \n", out)
result1:
[[ 4  4  1 -3]]

In [73]:
```

5. NUMPY ARRAY OPERATION

○ Multiplication

arr1

0	1	2	3
*			

arr2

4	5	6	7
=			

0	5	12	21
---	---	----	----

○ Devision

arr1

4	5	3	1
/			

arr2

1	1	2	3
=			

4	5	1.5	0.333
---	---	-----	-------

```
Desktop — IPython: mac/Desktop — ipython — 48x14

In [75]: arr1 = np.array([[0, 1, 2, 3]])
....: arr2 = np.array([[4, 5, 6, 7]])
....:
....:
....: out = np.multiply(arr1, arr2)
....: print("result1: \n", out)
result1:
[[ 0  5 12 21]]

In [76]:
```

```
Desktop — IPython: mac/Desktop — ipython — 48x14

In [78]: arr1 = np.array([[4, 5, 3, 1]])
....: arr2 = np.array([[1, 1, 2, 3]])
....:
....:
....: out = np.divide(arr1, arr2)
....: print("result1: \n", out)
result1:
[[4.          5.          1.5         0.33333333]]

In [79]:
```

5. NUMPY ARRAY OPERATION

○ Matrix Multiplication

$$\begin{array}{c}
 \text{X} \qquad \text{Y} \qquad \text{out} \\
 \begin{array}{|c|c|c|} \hline 1 & 1 & 2 \\ \hline 1 & 2 & 3 \\ \hline \end{array} \qquad
 \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 2 & 3 \\ \hline 4 & 5 \\ \hline \end{array} \qquad
 \begin{array}{|c|c|} \hline 10 & 14 \\ \hline 16 & 22 \\ \hline \end{array} \\
 = \qquad \qquad \qquad
 \end{array}$$

```

Desktop — IPython: mac/Desktop — ipython — 48x14

In [82]: X = np.array([[1, 1, 2],
....          [1, 2, 3]])
....: Y = np.array([[0, 1],
....:              [2, 3],
....:              [4, 5]])
....: out = np.dot(X, Y)
....: print(out)
....: print("shape: ", out.shape)
[[10 14]
 [16 22]]
shape: (2, 2)

In [83]: 
```

5. NUMPY ARRAY OPERATION

○ Transpose

$$\begin{bmatrix} 10 & 14 \\ 16 & 22 \end{bmatrix}^T = \begin{bmatrix} 10 & 16 \\ 14 & 22 \end{bmatrix}$$

```
Desktop — IPython: mac/Desktop — ipython — 48x14

In [84]: arr = np.array([[10, 14], [16, 22]])
....: print(arr.T)
....: print(np.transpose(arr))
[[10 16]
 [14 22]]
[[10 16]
 [14 22]]

In [85]:
```

5. NUMPY ARRAY OPERATION

NumPy Function	Description
np.add, np.subtract, np.multiply, np.divide	Addition, subtraction, multiplication, and division of two NumPy arrays.
np.power	Raises first input argument to the power of the second input argument (applied elementwise).
np.remainder	The remainder of division.
np.reciprocal	The reciprocal (inverse) of each element.
np.real, np.imag, np.conj	The real part, imaginary part, and the complex conjugate of the elements in the input arrays.
np.sign, np.abs	The sign and the absolute value.
np.floor, np.ceil, np.rint	Convert to integer values.
np.round	Rounds to a given number of decimals.

5. NUMPY ARRAY OPERATION

NumPy Function	Description
np.cos, np.sin, np.tan	Trigonometric functions.
np.arccos, np.arcsin, np.arctan	Inverse trigonometric functions.
np.cosh, np.sinh, np.tanh	Hyperbolic trigonometric functions.
np.arccosh, np.arcsinh, np.arctanh	Inverse hyperbolic trigonometric functions.
np.sqrt	Square root.
np.exp	Exponential.
np.log, np.log2, np.log10	Logarithms of base e, 2, and 10, respectively.

5. NUMPY ARRAY OPERATION

NumPy Function	Description
np.mean	The average of all values in the array.
np.std	Standard deviation.
np.var	Variance.
np.sum	Sum of all elements.
np.prod	Product of all elements.
np.cumsum	Cumulative sum of all elements.
np.cumprod	Cumulative product of all elements.
np.min, np.max	The minimum/maximum value in an array.
np.argmin, np.argmax	The index of the minimum/maximum value in an array.
np.all	Returns True if all elements in the argument array are nonzero.
np.any	Returns True if any of the elements in the argument array is nonzero.