PIAS
PHENIKAA INSTITUTE FOR ADVANCED STUDY

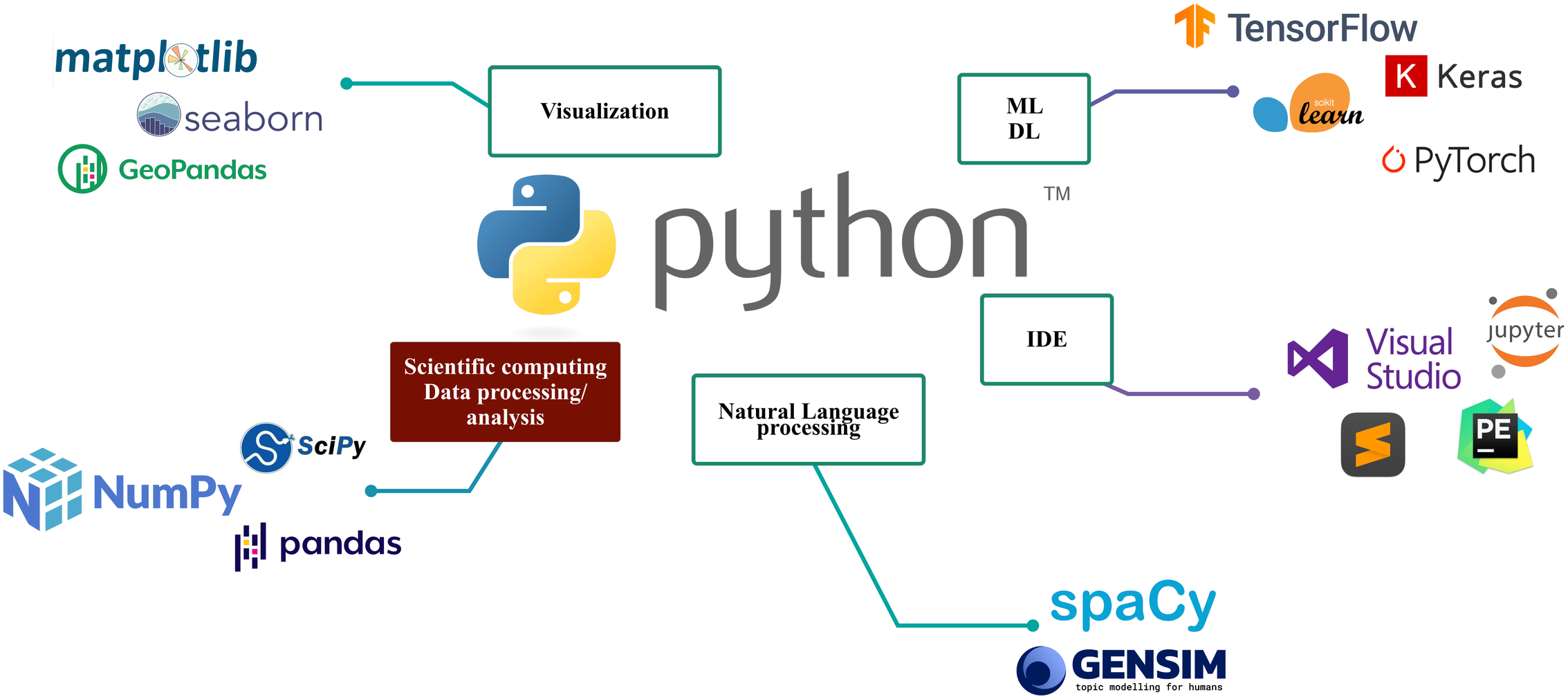
PHENIKAA UNIVERSITY
Realizing your potential

Tien-Lam Pham
Ph-D
Falculty of Computer Science

Van-Quyen Nguyen

1. INTRODUCTION

- **Numpy is a Python library for scientific computations**



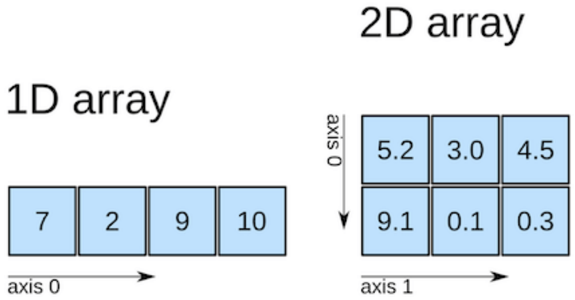
2. COMMON ATTRIBUTES

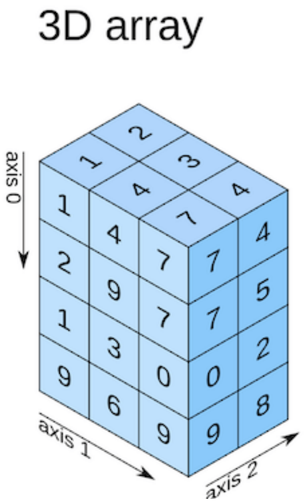
- dtype — Data type
- nbytes — Number of bytes used to store the data
- ndim — Number of dimensions
- shape — A tuple of number of elements in each dimension
- size — Total number elements

```
Desktop — IPython: mac/Desktop — ipython — 55×19

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]:
```





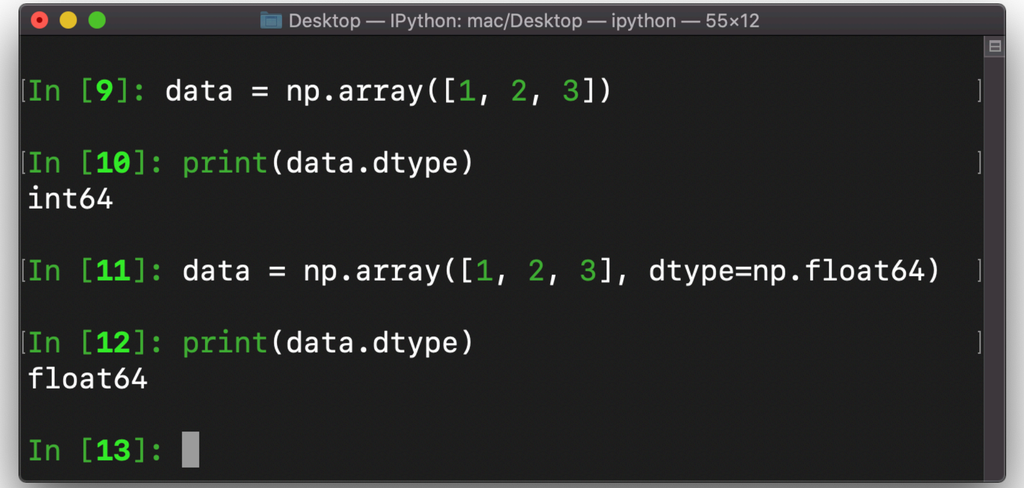

```
Desktop — IPython: mac/Desktop — ipython — 55×12

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]:
```

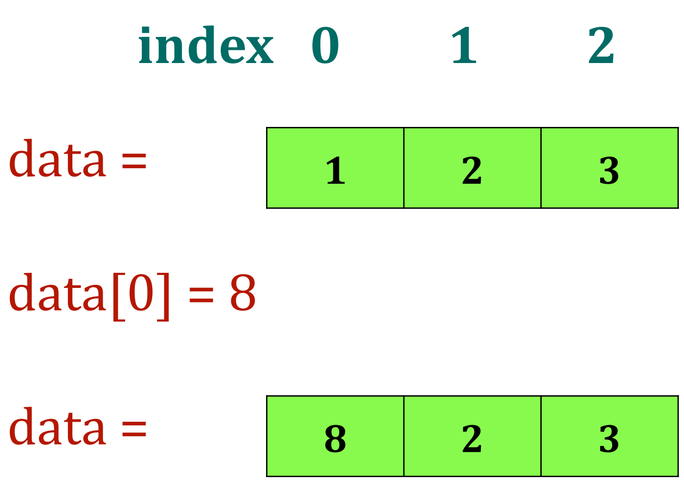
2. COMMON ATTRIBUTES

- dtype Data type



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

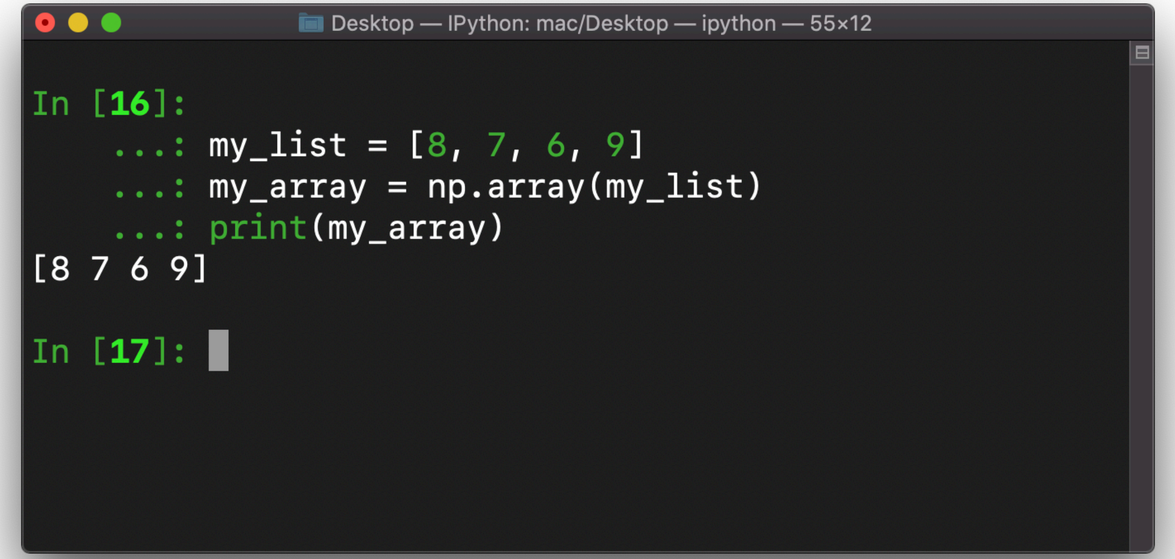
3. UPDATE AN ELEMENT



4. CREATE NUMPY ARRAY

- From list

arr_np = np.array(python_list)

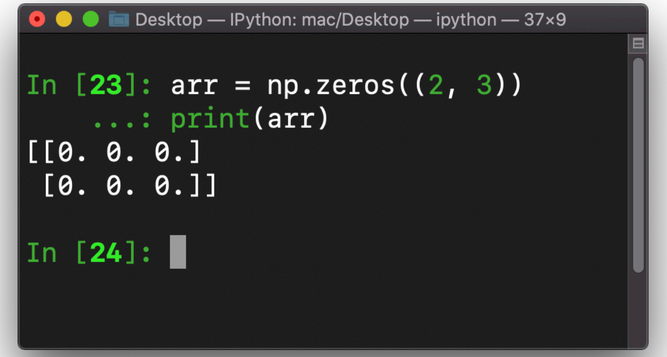


4. CREATE NUMPY ARRAY

- From common fuctions

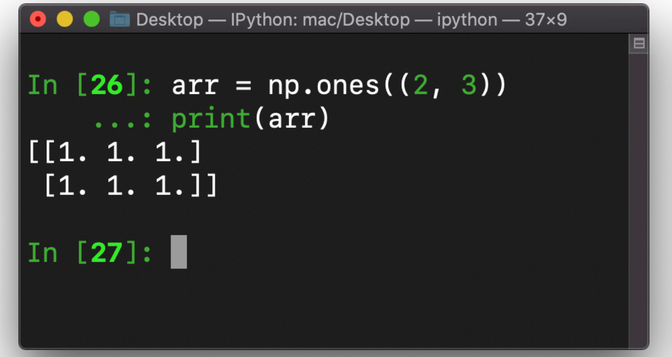
zeros()

0	0	0
0	0	0



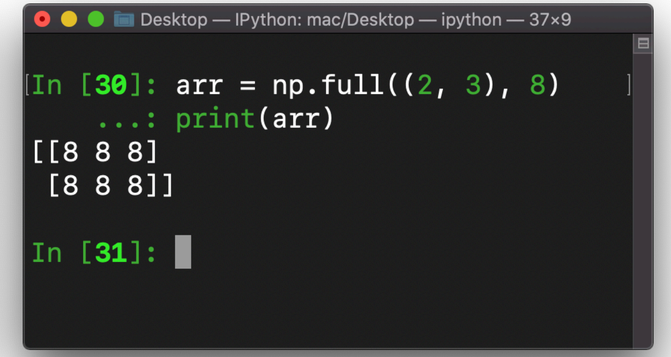
ones()

1	1	1
1	1	1



full()

8	8	8
8	8	8



4. CREATE NUMPY ARRAY

- From common fuctions

arange()

arr1 =

0	1	2	3

arr2 =

0	5	10

```
Desktop — IPython: mac/Desktop — ipython — 37×11

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

```
Desktop — IPython: mac/Desktop — ipython — 37×11

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

In [35]:
```

random()

```
Desktop — IPython: mac/Desktop — ipython — 37×11

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

PHENIKAA UNIVERSITY
Realizing your potential

numpy.hstack()

arr1 =

0	1	2	3

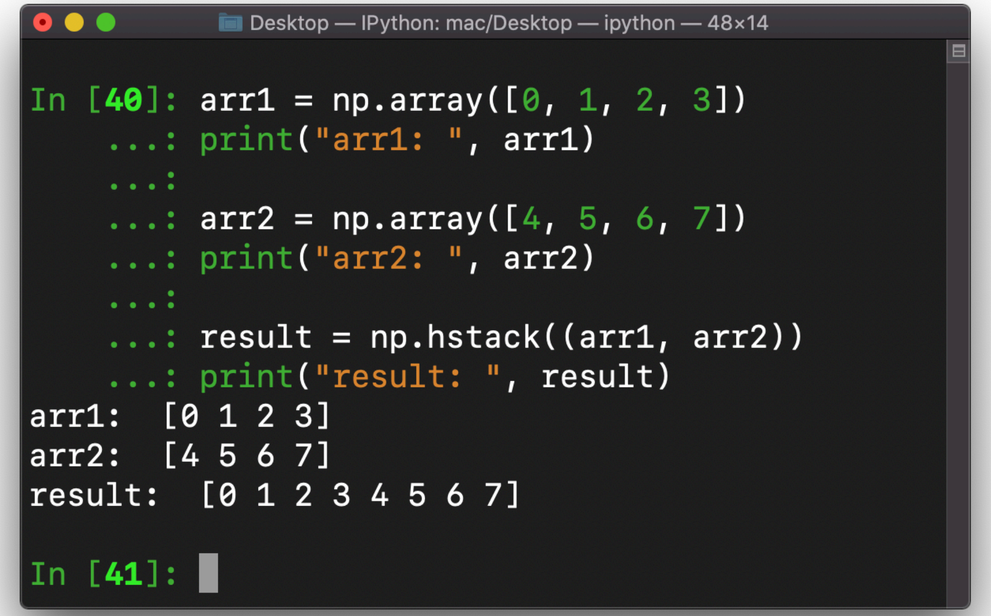
arr2 =

4	5	6	7

hstack((arr1, arr2))

result =

0	1	2	3	4	5	6	7



4. CREATE NUMPY ARRAY

PHENIKAA UNIVERSITY
Realizing your potential

numpy.vstack()

arr1 =

0	1	2	3

arr2 =

4	5	6	7

vstack((arr1, arr2))

result =

0	1	2	3
4	5	6	7

4. CREATE NUMPY ARRAY

PHENIKAA UNIVERSITY
Realizing your potential

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

```
Desktop — IPython: mac/Desktop — ipython — 48×23
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]:
```

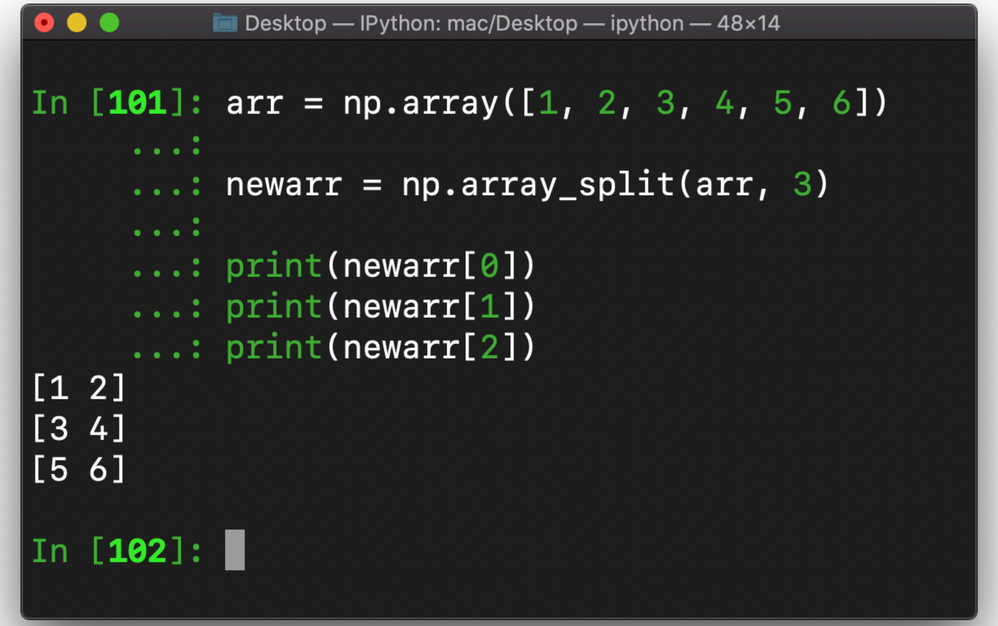
4. CREATE NUMPY ARRAY

- **numpy.array_split()**
  numpy.split()

1	2	3	4	5	6

aray_split(arr, 3)

newarr[0]	1	2
newarr[1]	3	4
newarr[2]	5	6



4. CREATE NUMPY ARRAY

PHENIKAA UNIVERSITY
Realizing your potential

○ where() fuction

arr =

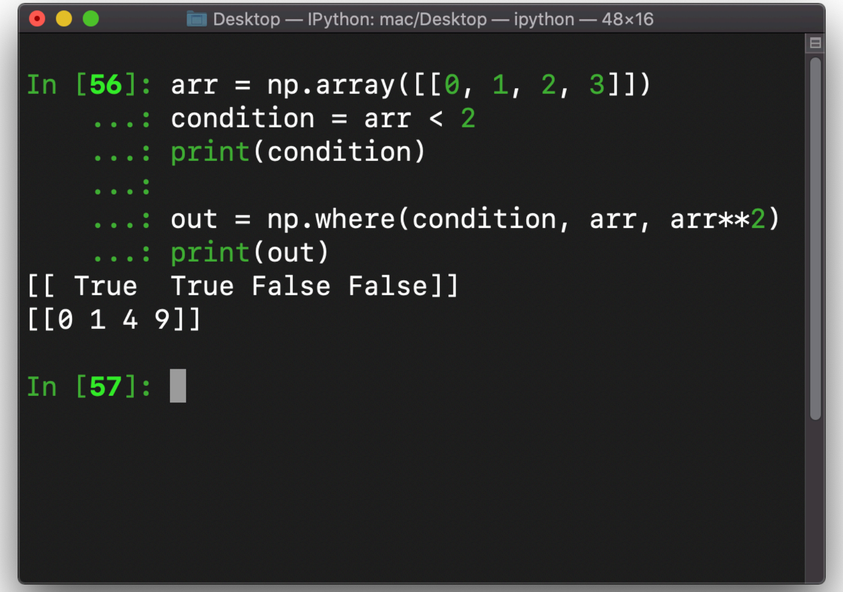
0	1	2	3

arr<2 =

T	T	F	F

out =

0	1	4	6



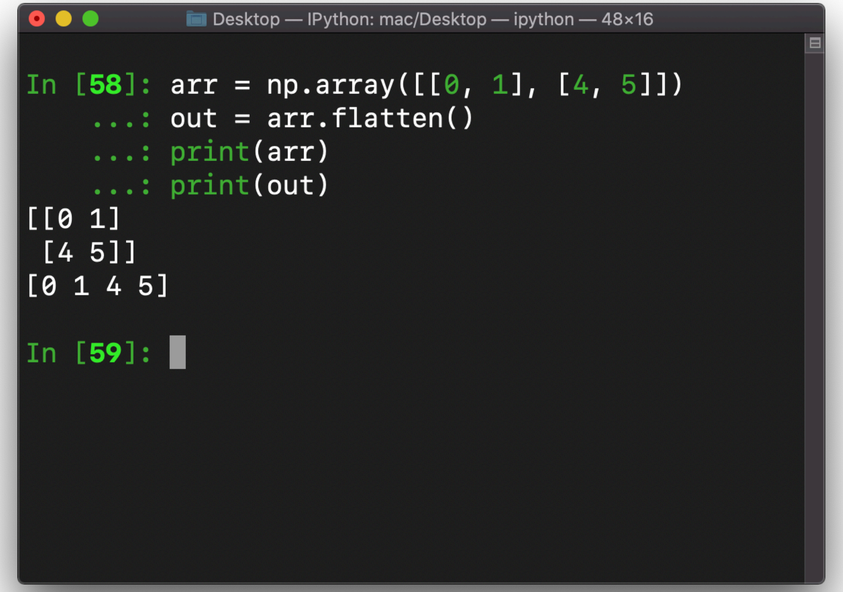
○ flatten() fuction

arr =

0	1
4	5

out =

0	1	4	5



4. CREATE NUMPY ARRAY

PHENIKAA UNIVERSITY
Realizing your potential

○ reshape() fuction

arr =

0	1	2	3
4	5	6	7

out =

0	1
2	3
4	5
6	7

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
2	3
4	5
6	7

arr[1:3]

0	1
2	3
4	5
6	7

arr[0:2, 0]

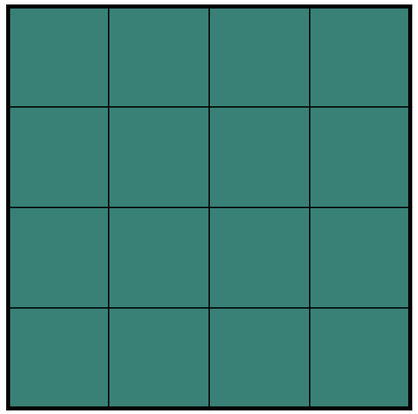
0	1
2	3
4	5
6	7

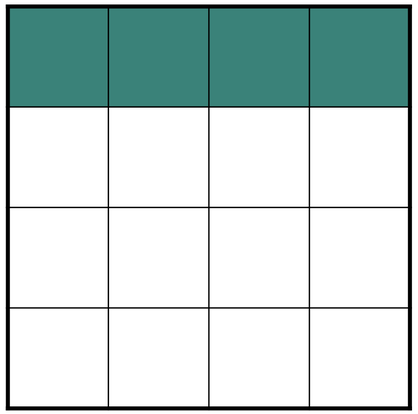
arr[:, 1:]

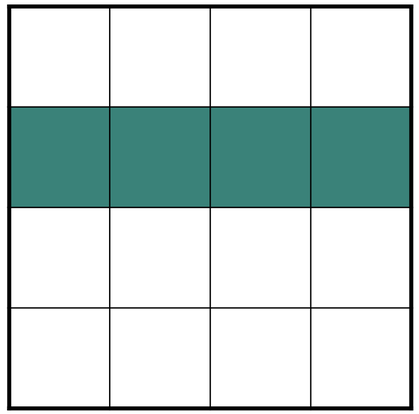
0	1
2	3
4	5
6	7

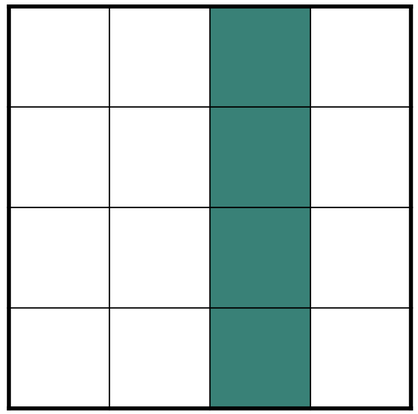
5. ARRAY INDEXING

Excercise





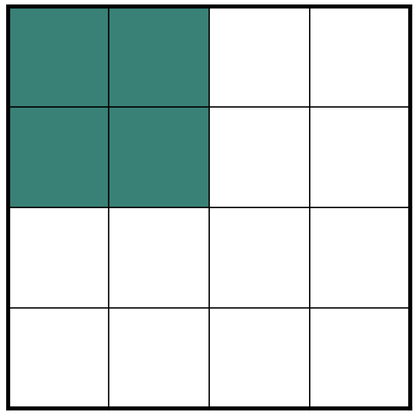


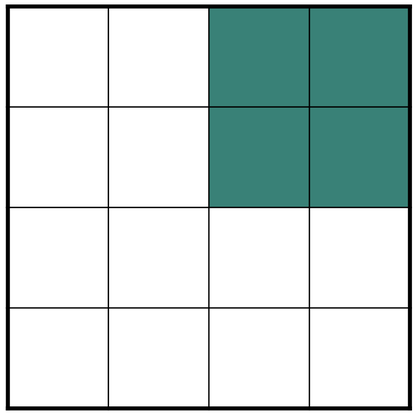


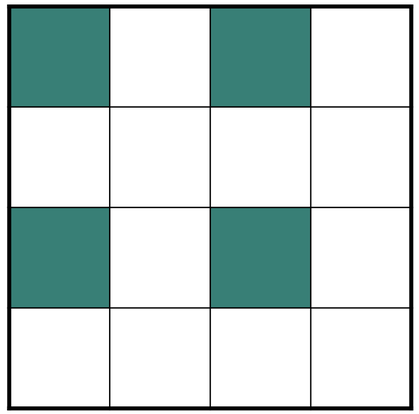
5. ARRAY INDEXING

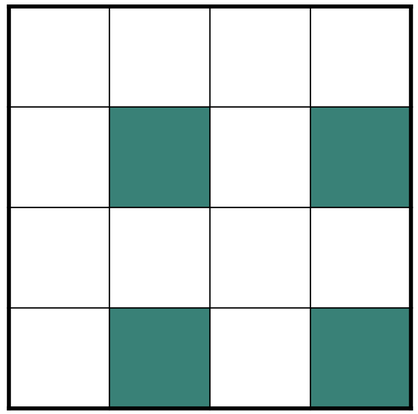
PHENIKAA UNIVERSITY
Realizing your potential

Excercise





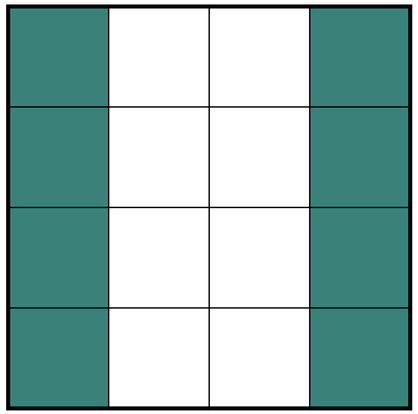


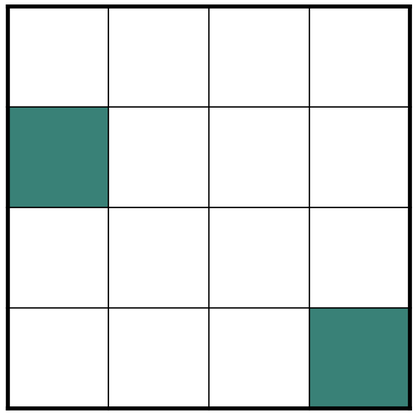


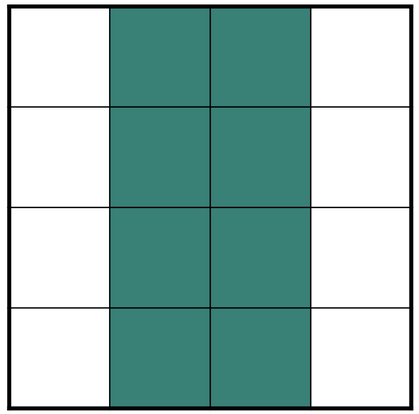
5. ARRAY INDEXING

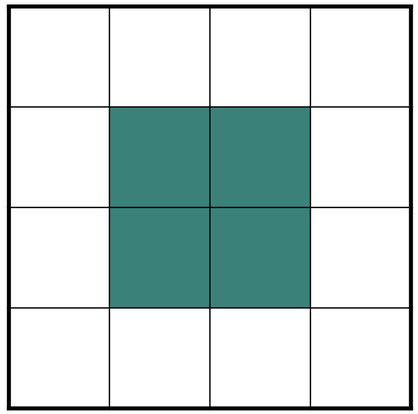
PHENIKAA UNIVERSITY
Realizing your potential

Excercise





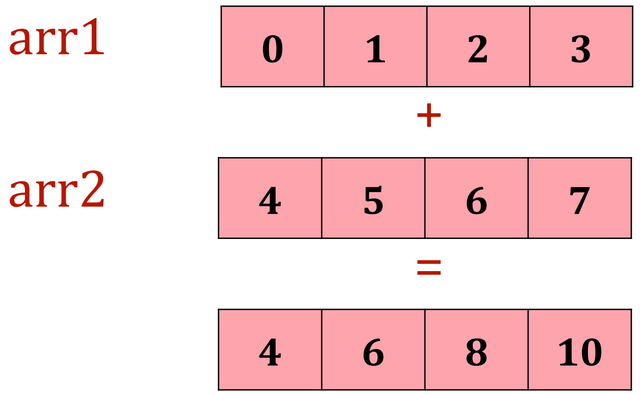




5. NUMPY ARRAY OPERATION

PHENIKAA UNIVERSITY
Realizing your potential

○ Addition

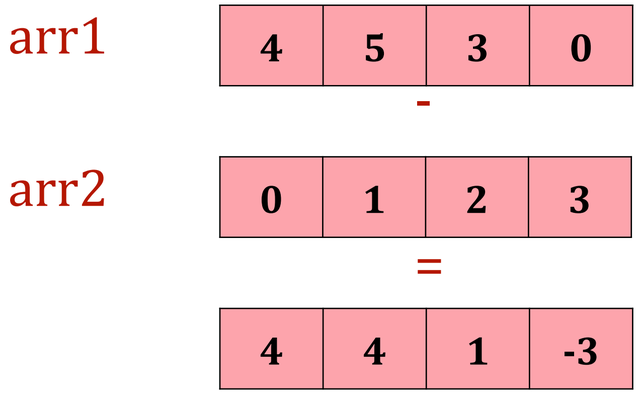



```
Desktop — IPython: mac/Desktop — ipython — 48×14

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]:
```

○ Subtraction




```
Desktop — IPython: mac/Desktop — ipython — 48×14

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

PHENIKAA UNIVERSITY
Realizing your potential

○ Multiplication

arr1

0	1	2	3

*

arr2

4	5	6	7

=

0	5	12	21

```
Desktop — IPython: mac/Desktop — ipython — 48×14

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]:
```

○ Devision

arr1

4	5	3	1

/

arr2

1	1	2	3

=

4	5	1.5	0.333

```
Desktop — IPython: mac/Desktop — ipython — 48×14

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

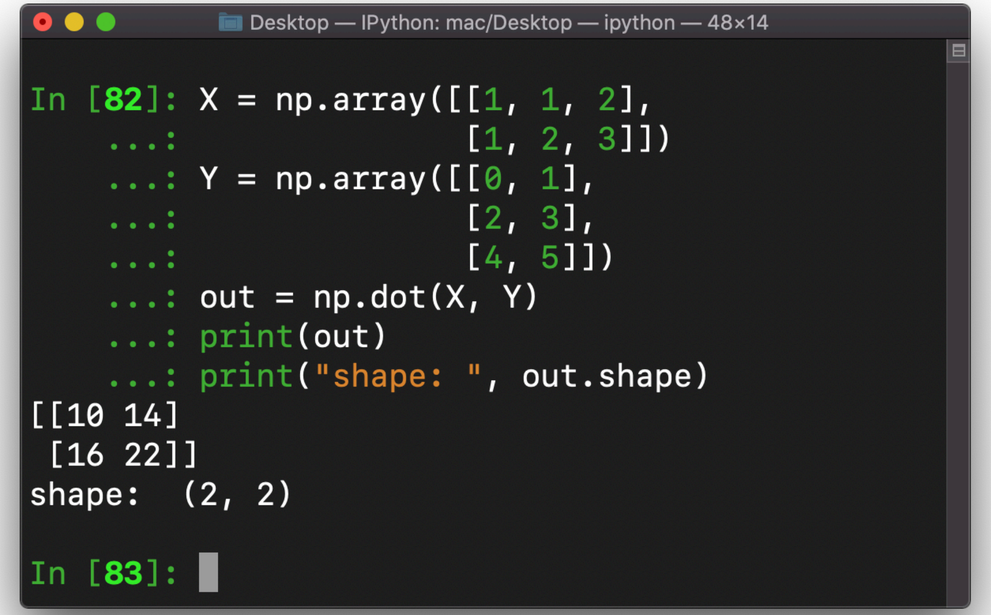
X Y out

1	1	2
1	2	3

0	1
2	3
4	5

=

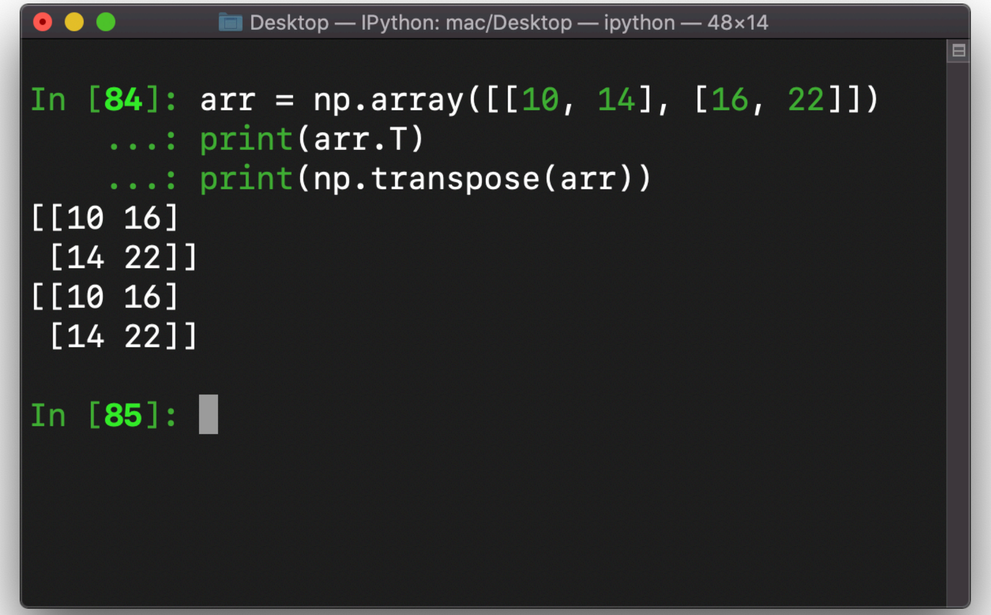
10	14
16	22



5. NUMPY ARRAY OPERATION

○ Transpose

		T			
10	14		=	10	16
16	22			14	22



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

PHENIKAA UNIVERSITY
Realizing your potential

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.