Lecture-11

NumPy Function, Slice & Reshape in Python

Contents

- Universal functions
- Indexing and slicing
- Reshaping and transposing

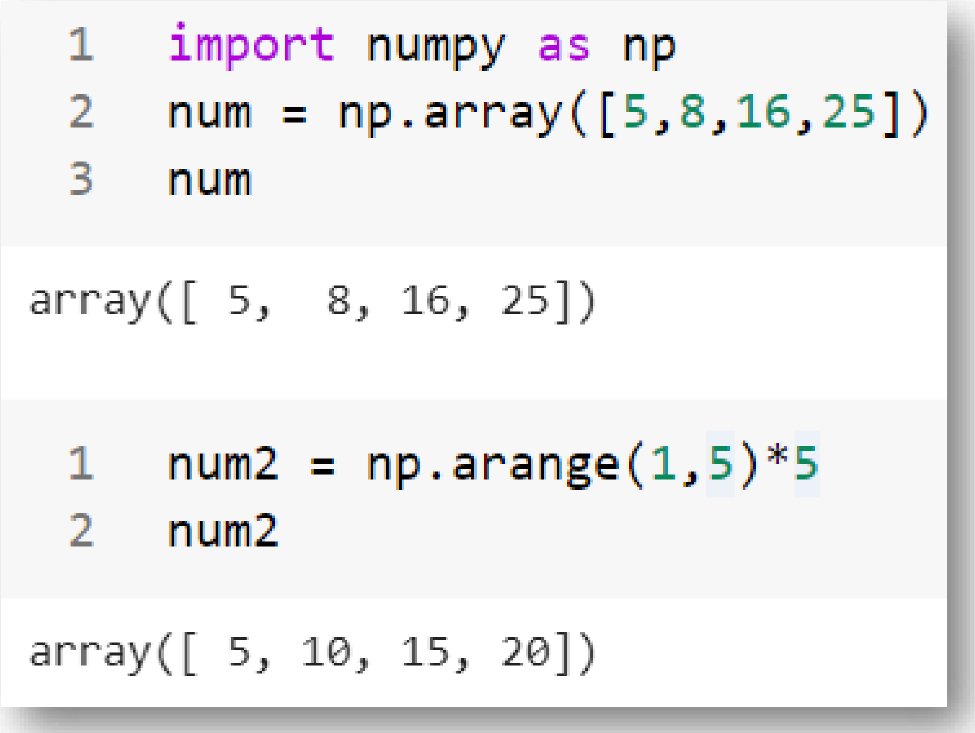
Universal Functions

- NumPy offers dozens of standalone universal functions that perform various element-wise operations. Some of these functions are called when you use operators like + and * on arrays.
- The NumPy documentation lists universal functions in five categories—math, trigonometry, bit manipulation, comparison and floating point.

Math	add, subtract, multiply, divide, remainder, exp, log, sqrt, power, and more.
Trigonometry	sin, cos, tan, hypot, arcsin, arccos, arctan, and more.
Bit manipulation	bitwise_and, bitwise_or, bitwise_xor, invert, left_shift and right_shift.
Comparison	greater, greater_equal, less, less_equal, equal, not_equal, logical_and,logical_or, logical_xor, logical_not, minimum, maximum, and more.
Floating point	floor, ceil, isinf, isnan, fabs, trunc, and more.

Universal Functions

- Array Arithmetic Function

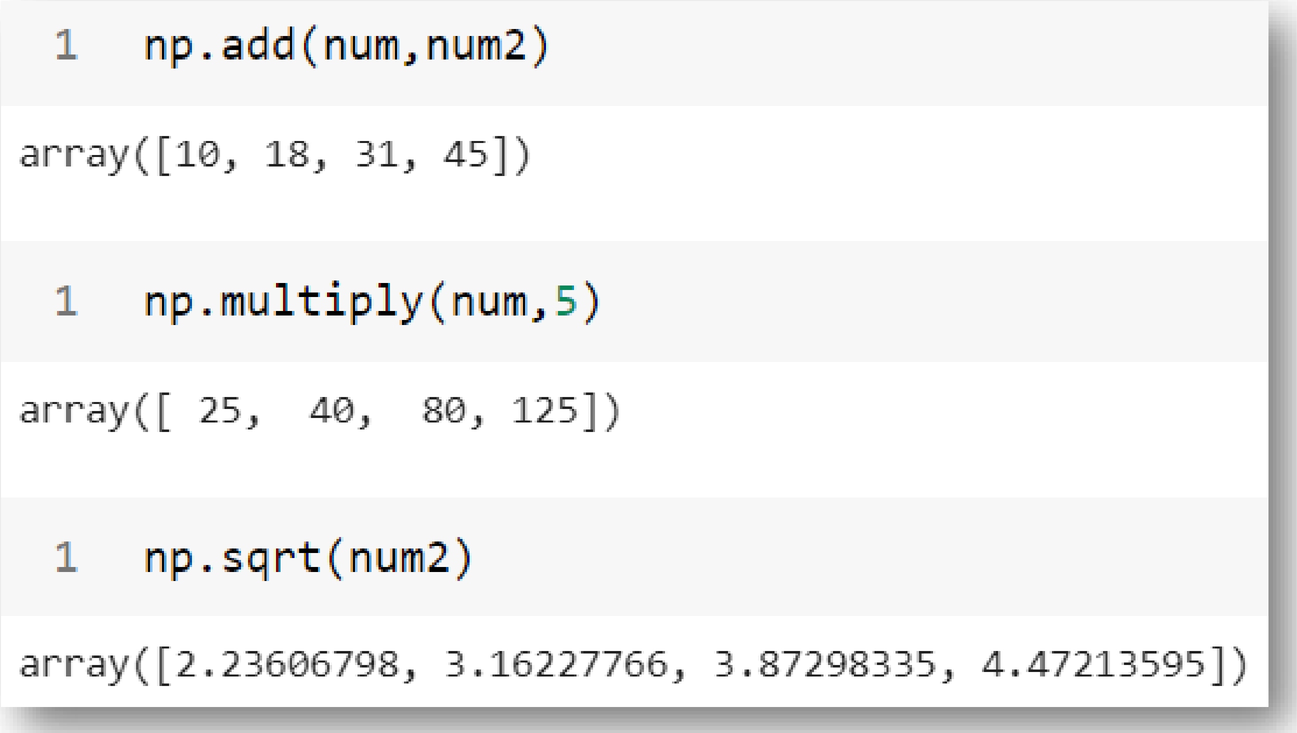


```
import numpy as np
num = np.array([5,8,16,25])
num
```

```
array([ 5,  8, 16, 25])
```

```
num2 = np.arange(1,5)*5
num2
```

```
array([ 5, 10, 15, 20])
```



```
np.add(num,num2)
```

```
array([10, 18, 31, 45])
```

```
np.multiply(num,5)
```

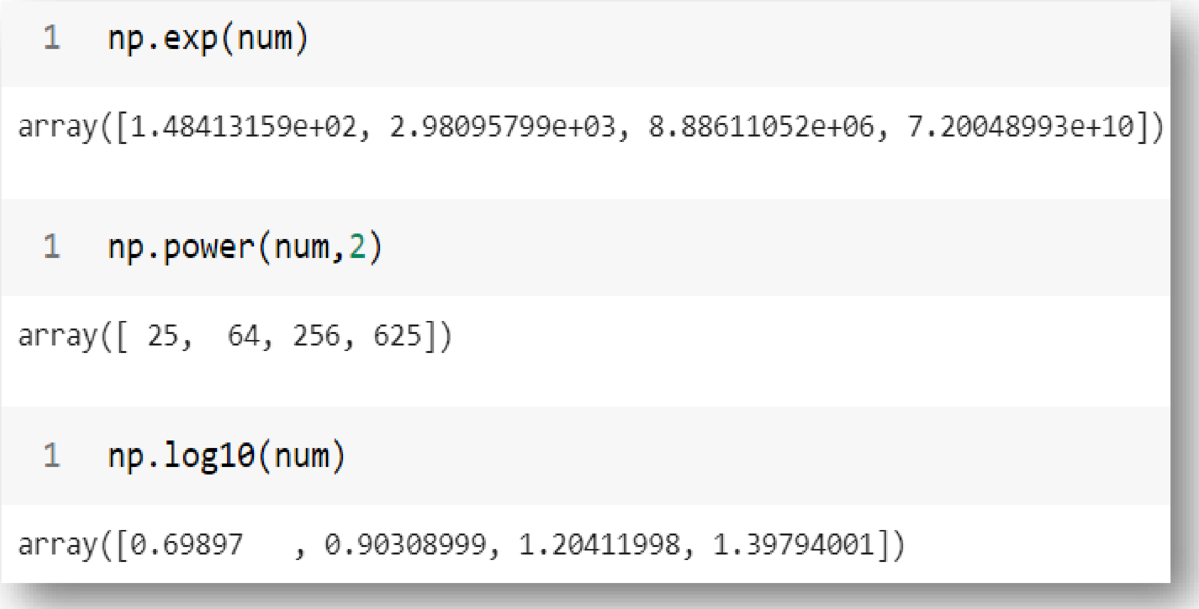
```
array([ 25,  40,  80, 125])
```

```
np.sqrt(num2)
```

```
array([2.23606798, 3.16227766, 3.87298335, 4.47213595])
```

Universal Functions

- Exponents and logarithms



```
np.exp(num)
```

```
array([1.48413159e+02, 2.98095799e+03, 8.88611052e+06, 7.20048993e+10])
```

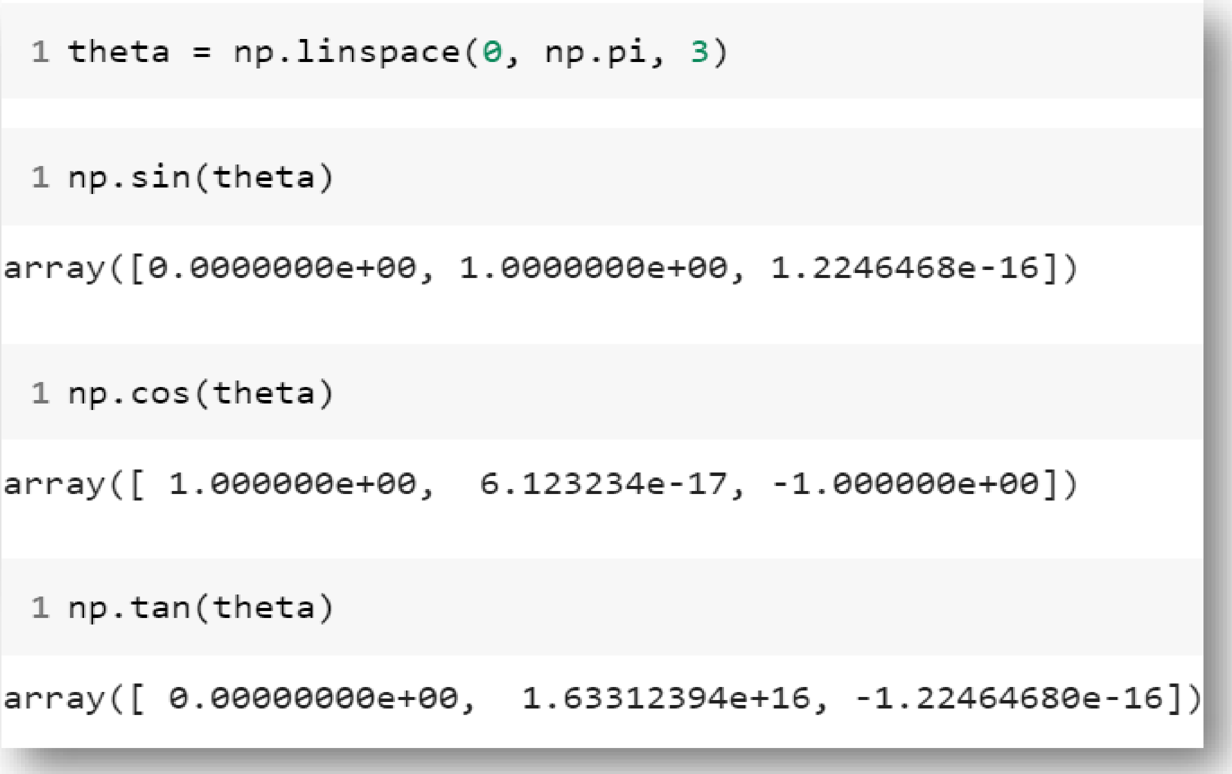
```
np.power(num,2)
```

```
array([ 25,  64, 256, 625])
```

```
np.log10(num)
```

```
array([0.69897   , 0.90308999, 1.20411998, 1.39794001])
```

- Trigonometric functions



```
theta = np.linspace(0, np.pi, 3)
```

```
np.sin(theta)
```

```
array([0.0000000e+00, 1.0000000e+00, 1.2246468e-16])
```

```
np.cos(theta)
```

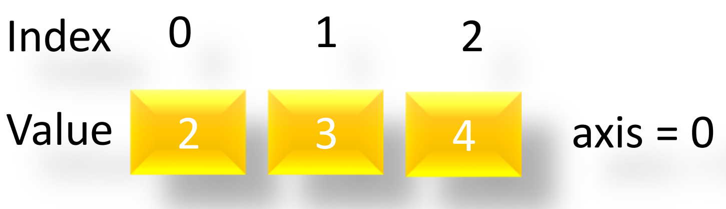
```
array([ 1.000000e+00,  6.123234e-17, -1.000000e+00])
```

```
np.tan(theta)
```

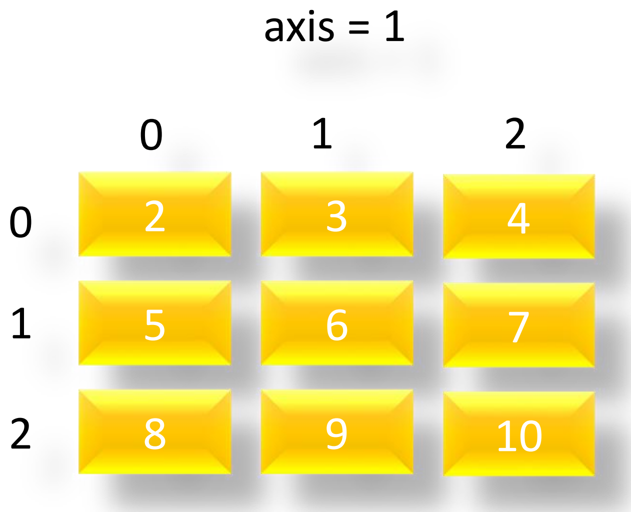
```
array([ 0.00000000e+00,  1.63312394e+16, -1.22464680e-16])
```

Indexing and Slicing

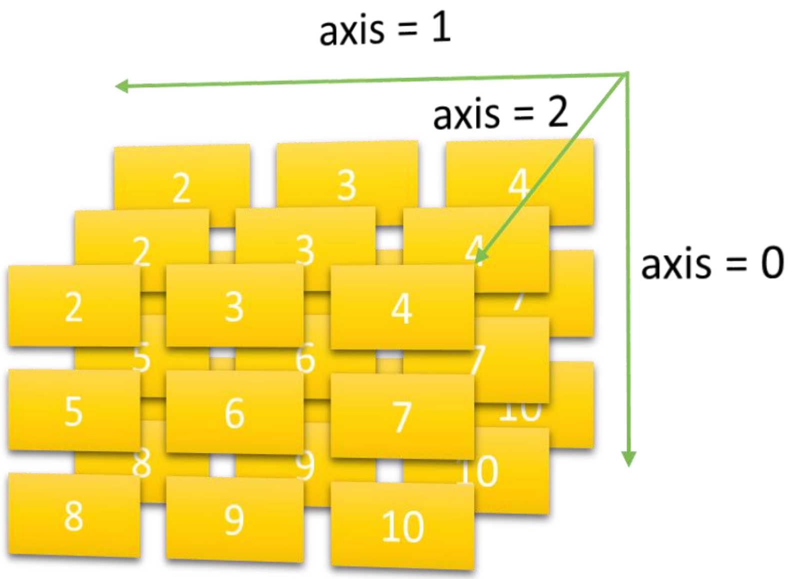
- Array element can access by referring to its index number.
- The indexes in NumPy arrays start with 0, meaning that the first element has index 0, then second element has index 1 etc.



1D array



2D array

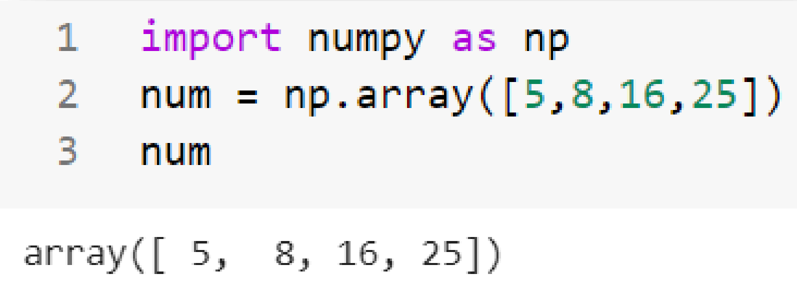


3D array

Indexing and Slicing

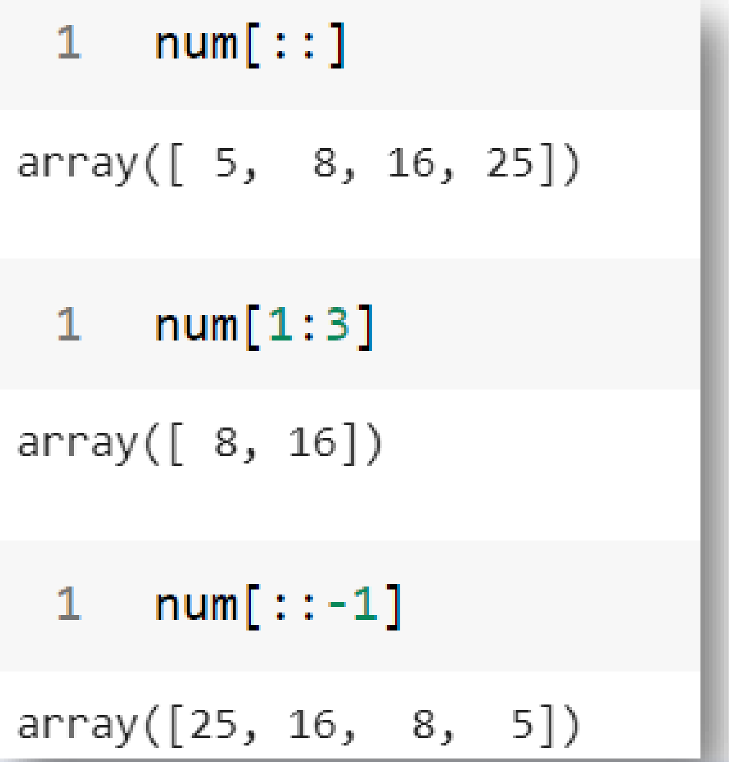
- Slicing step describes the spacing between two values and is optional [start:stop] with a default value of 1. Negative values are supported (e.g [::-1] reverses the order).

Indexing with One-Dimensional arrays



```
import numpy as np
num = np.array([5,8,16,25])
num
```

```
array([ 5,  8, 16, 25])
```



```
num[::]
```

```
array([ 5,  8, 16, 25])
```

```
num[1:3]
```

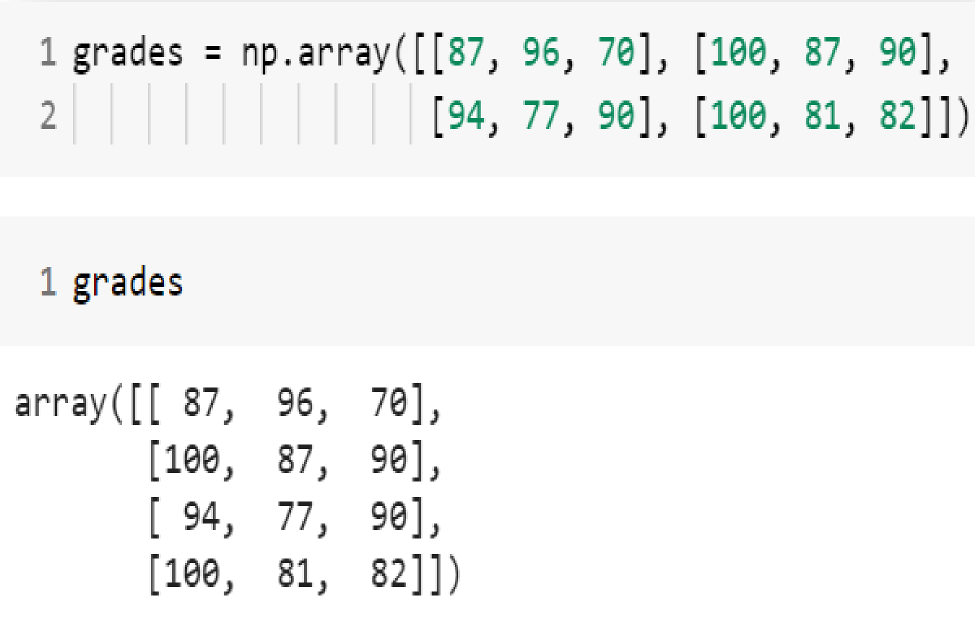
```
array([ 8, 16])
```

```
num[::-1]
```

```
array([25, 16,  8,  5])
```

Indexing and Slicing

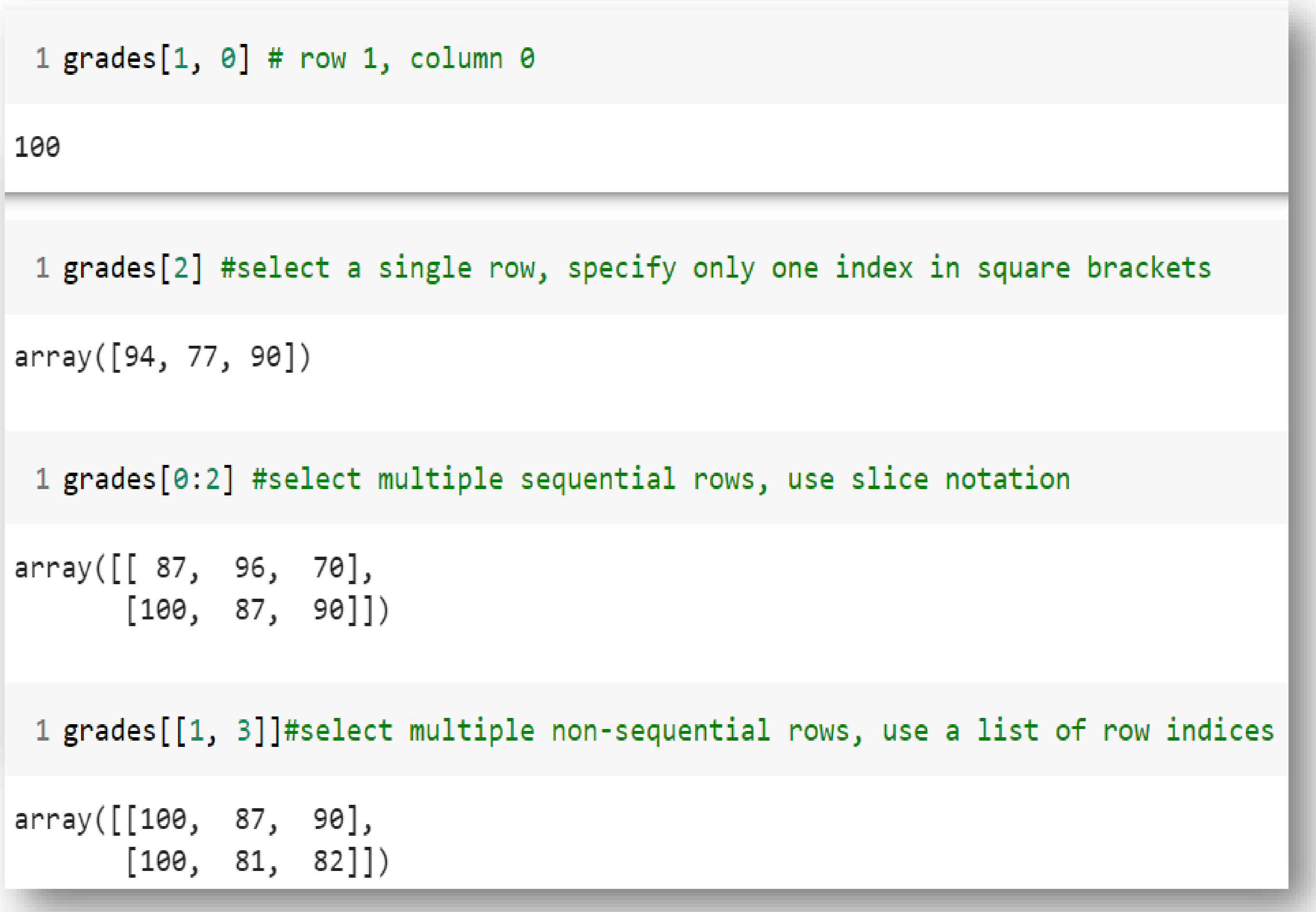
- Indexing with Two-Dimensional arrays



```
grades = np.array([[87, 96, 70], [100, 87, 90],
                   [94, 77, 90], [100, 81, 82]])
```

```
grades
```

```
array([[ 87,  96,  70],
       [100,  87,  90],
       [ 94,  77,  90],
       [100,  81,  82]])
```



```
grades[1, 0] # row 1, column 0
```

```
100
```

```
grades[2] #select a single row, specify only one index in square brackets
```

```
array([94, 77, 90])
```

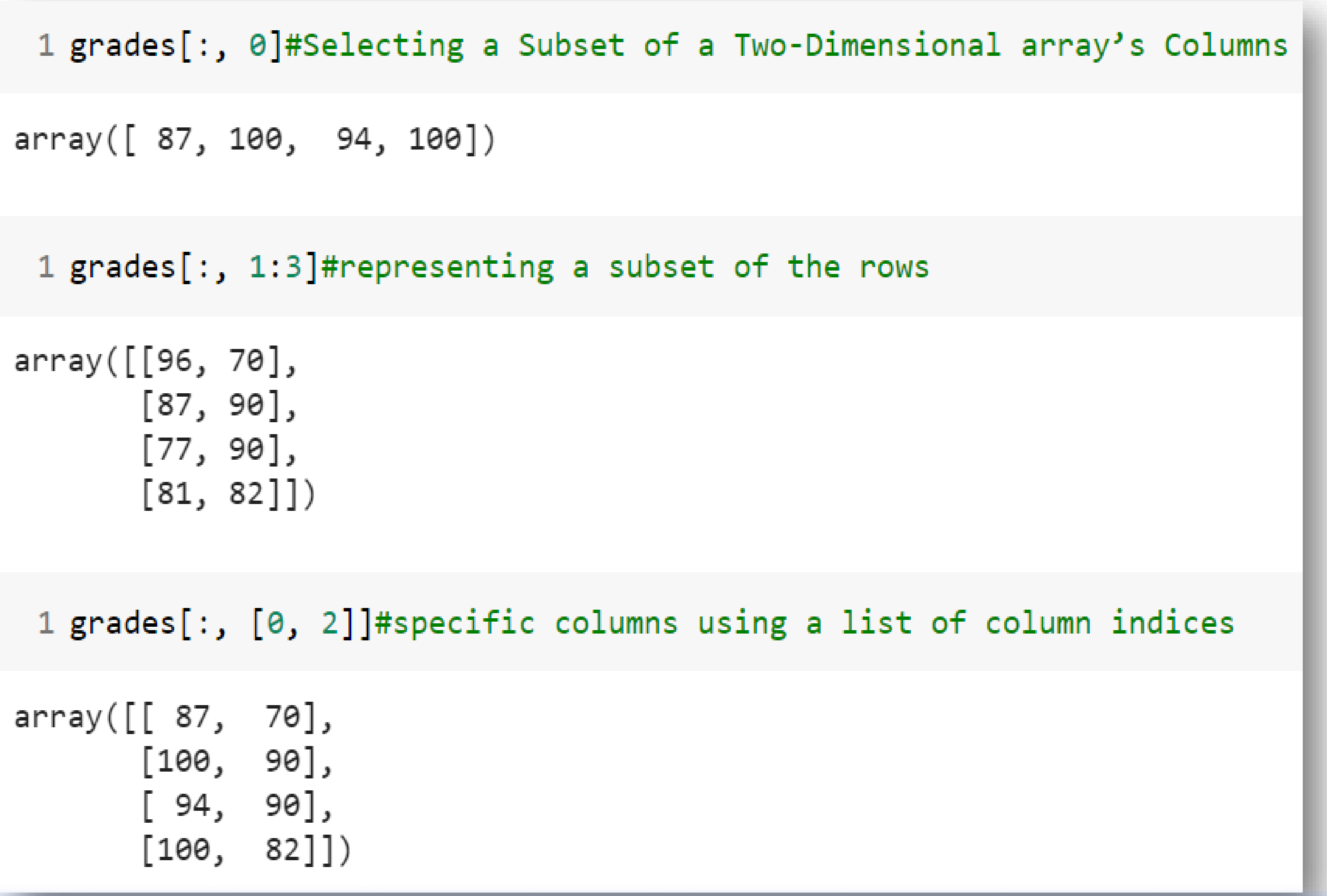
```
grades[0:2] #select multiple sequential rows, use slice notation
```

```
array([[ 87,  96,  70],
       [100,  87,  90]])
```

```
grades[[1, 3]]#select multiple non-sequential rows, use a list of row indices
```

```
array([[100,  87,  90],
       [100,  81,  82]])
```

Indexing and Slicing



```
grades[:, 0]#Selecting a Subset of a Two-Dimensional array’s Columns
```

```
array([ 87, 100,  94, 100])
```

```
grades[:, 1:3]#representing a subset of the rows
```

```
array([[96, 70],
       [87, 90],
       [77, 90],
       [81, 82]])
```

```
grades[:, [0, 2]]#specific columns using a list of column indices
```

```
array([[ 87,  70],
       [100,  90],
       [ 94,  90],
       [100,  82]])
```

Indexing and Slicing

Exercise :

Given the following array:

array([[1, 2, 3, 4, 5],[6, 7, 8, 9, 10],[11, 12, 13, 14, 15],[16,17,18,19,20]])

a) Select the second row.

b) Select the first and third rows.

c) Select the middle three columns.

d) Show 12 and 13 from array.

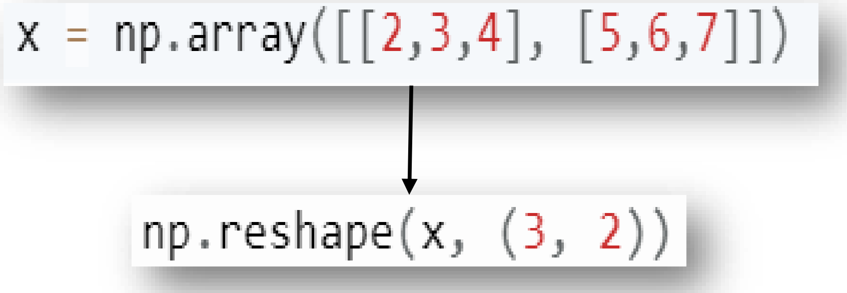
e) Show reverse order of third rows values.

f) Select the fourth rows and find **max**, **min**, **sum**, **mean, std** and **var** of values.

g) Find **log10**, **log2**, **log** values of rows one.

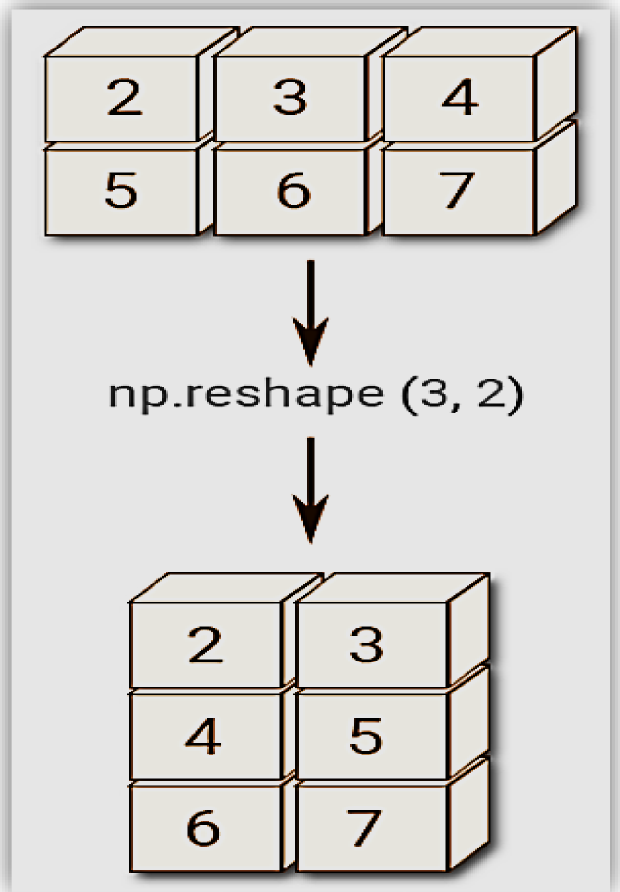
Reshaping

- The reshape() function is used to give a new shape to an array without changing its data.



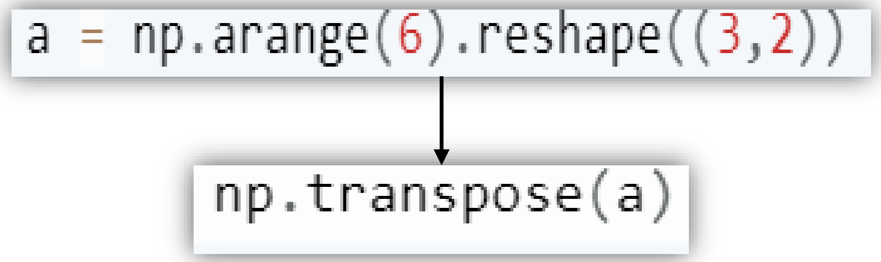
```
x = np.array([[2,3,4], [5,6,7]])
```

```
np.reshape(x, (3, 2))
```



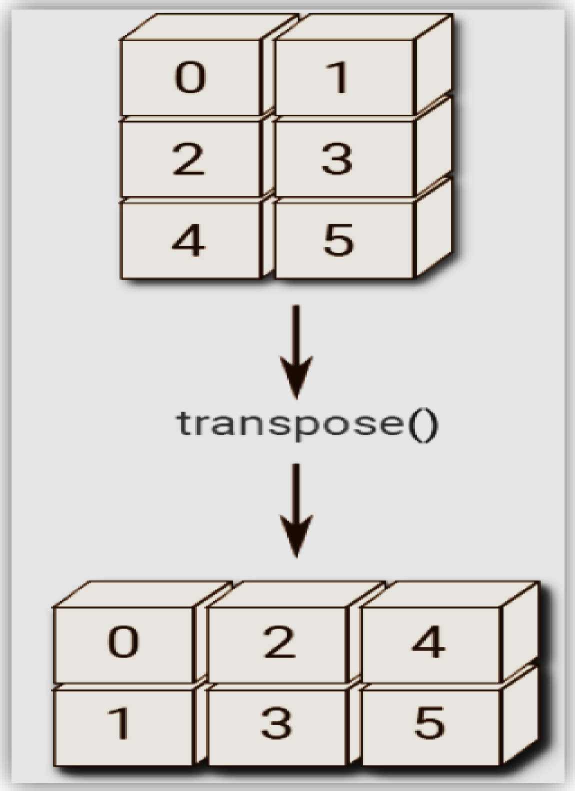
Transposing

- Returns the transpose of a matrix.
- If the matrix shape is (X,Y), then transpose matrix shape will be (Y,X). It switches the row and column indices of a matrix.



```
a = np.arange(6).reshape((3,2))
```

```
np.transpose(a)
```



Reshaping and transposing

Exercise:

Given a 3-by-4 array:

array([[[1, 3, 2,4],[8, 6, 5,7],[11,10,12,9]]])

a) Find transpose of given array.

b) Reshape the array.

c) Sort the array.

Thank You