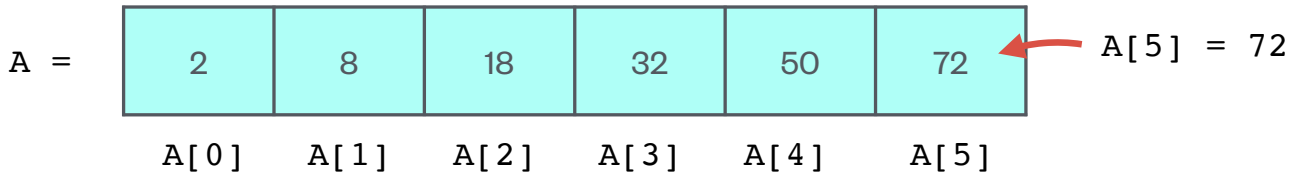
Chapter 8. NumPy Arrays: 2D

```
import numpy as np

me = 9.11e-31     # mass of electron
c  = 299792458    # speed of light

u  = 0.1 * c      # particle velocity

gamma = 1 / np.sqrt(1-(u/c)**2)   # gamma factor

KE = (gamma-1) * me * c**2        # relativistic kinetic energy
```

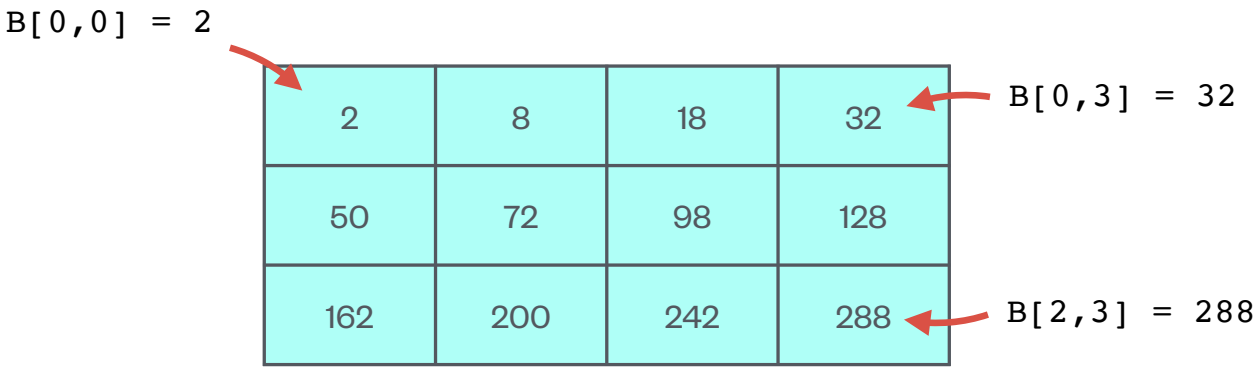
Python for Physicists

1D Array



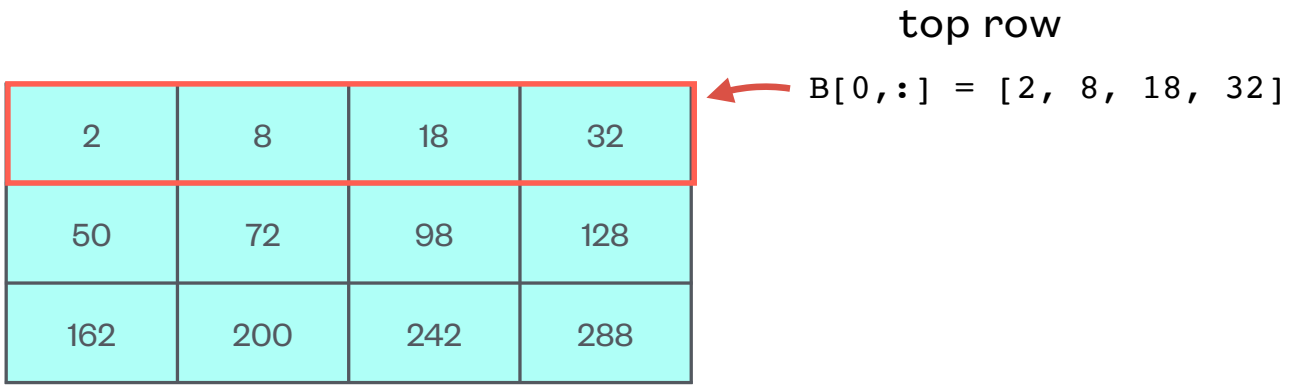
- Array indices start at 0
- Python does not distinguish between vertical (column) arrays and horizontal (row) arrays

2D Array (matrix)



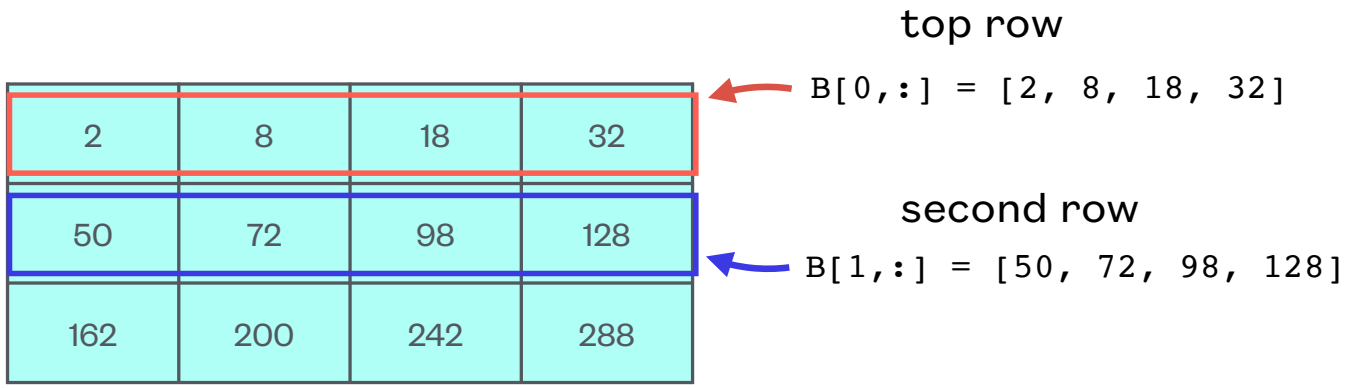
- Indices start at [0,0] in top right corner of array
- First index is row number, second is column number

Slicing 2D Arrays



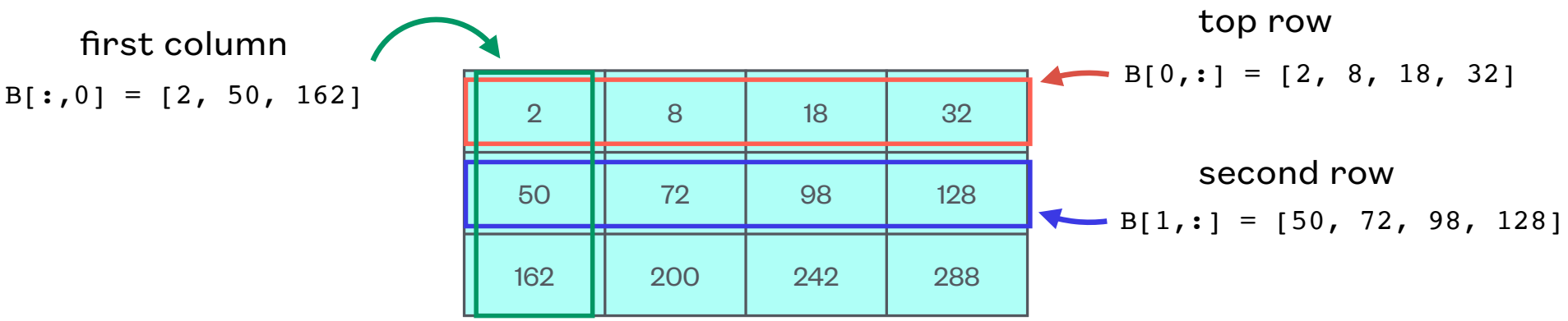
- `B[0,:]` = first (top) row

Slicing 2D Arrays



- `B[0,:]` = first (top) row
- `B[1,:]` = second row

Slicing 2D Arrays



- `B[0,:]` = first (top) row
- `B[1,:]` = second row
- `B[:,0]` = first column

Creating 2D Arrays

- We can create 2D arrays filled with 0's or 1's using the following commands:

```
np.zeros((n,m))
np.ones((n,m))
```

- The shape of the array is specified using a `(n,m)` tuple where

  n = # rows

  m = # columns

`A = np.zeros((3,4))`

0	0	0	0
0	0	0	0
0	0	0	0

`A = np.ones((4,2))`

1	1
1	1
1	1
1	1

Creating Random 2D Arrays

- We can create random 2D arrays using the random number generator discussed in Chapter 7
- We specify the shape of the array using the `size=(n,m)` option

Example: Create a 3×4 array filled with random integers with $0 \leq y \leq 9$

```
rng = np.random.default_rng()
A = rng.integers(1, 9, size=(3,4),endpoint=True)
```

6	8	1	0
7	2	2	6
5	3	8	9

Array Attributes

The following three attributes contain basic info about the array:

- `array.ndim` = number of dimensions (2 for a 2D array)
- `array.size` = total number of elements (= rows × columns for 2D array)
- `array.shape` = tuple with number of elements in each dimension (nrows,ncols)

Example: 3×4 array

A =

6	8	1	0
7	2	2	6
5	3	8	9

```
A.ndim          # = 2 in this example
A.size          # = 12
A.shape         # = (3,4)

A.shape[0]      # = 3    (number of rows)
A.shape[1]      # = 4    (number of columns)
```

Reshaping Arrays

Transpose flips the rows and columns of an array:

- Use `.T` attribute to return the transpose

Example:

`A =`

6	8	1	0
7	2	2	6
5	3	8	9

`A.T =`

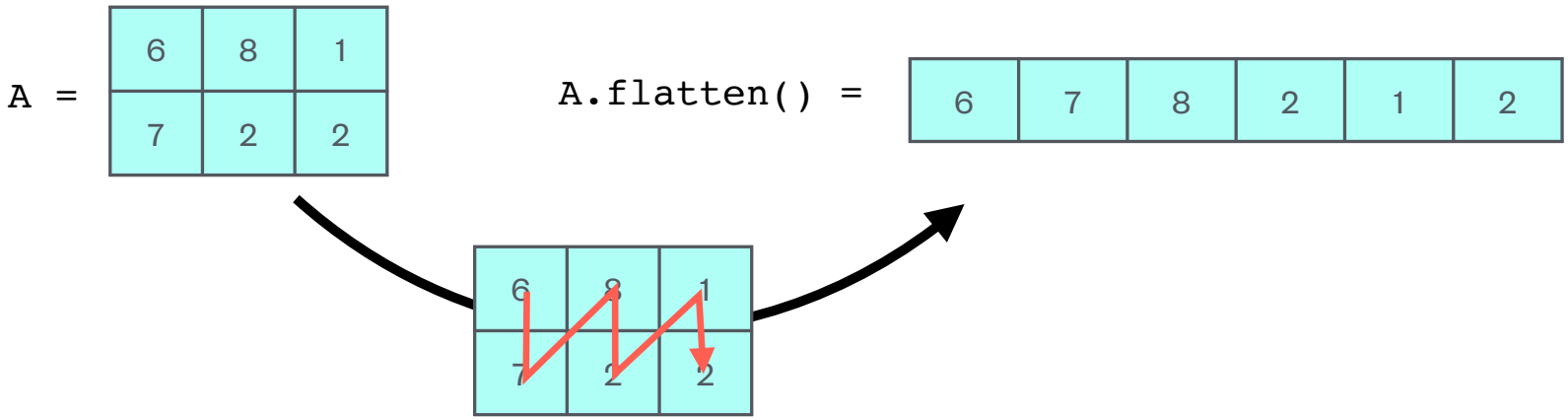
6	7	5
8	2	3
1	2	8
0	6	9

Reshaping Arrays

Flatten “unwraps” a higher-dimensional array into a 1D array

- To flatten a 2D array, one starts with the top-left element and reads down the first column, then second column, etc. to produce the flattened array.

Example:



Reshaping Arrays

Reshape reorders the elements in a given array to fit a new array.

- The total # of elements of the original and reshaped arrays must be the same
- `reshape((n,m))` takes a tuple (n,m) as it's argument

Examples:

`A =`

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

`A.reshape((3,4)) =`

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

`A.reshape((6,2)) =`

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

Computing Statistics along Rows or Columns

.mean() calculates the mean values of array elements in different ways

- `A.mean()` with no arguments returns the mean of all values in the array
- `A.mean(axis=0)` returns the mean along the array columns
- `A.mean(axis=1)` returns the mean along the array rows

Example: