Lineare Algebra Refresher

Short Intro about Vector

https://www.youtube.com/watch?v=s0Q3CojqRfM

Vector

- Is an n by 1 matrix
  - Usually referred to as a lower case letter
  - n rows
  - 1 column
  - e.g.

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

- Is a 4 dimensional vector
  - Refer to this as a vector R4
- Vector elements
  - v_i = i^{th} element of the vector
  - Vectors can be 0-indexed (C++) or 1-indexed (MATLAB)
  - In math 1-indexed is most common
    - But in machine learning 0-index is useful
  - Normally assume using 1-index vectors, but be aware sometimes these will (explicitly) be 0 index ones

Matrix

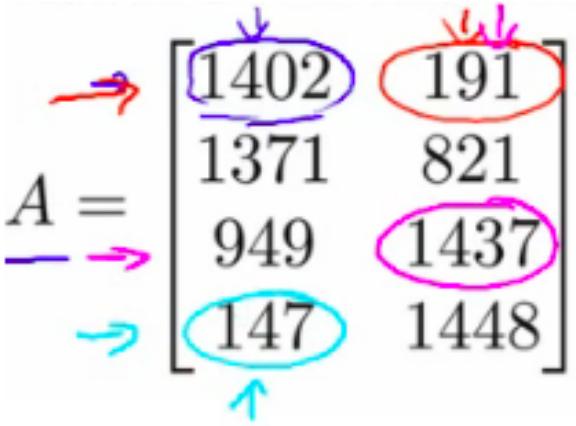
- Rectangular array of numbers written between square brackets
  - 2D array
  - Named as capital letters (A,B,X,Y)
- Dimension of a matrix are [Rows x Columns]
  - Start at top left
  - To bottom left
  - To bottom right
  - $R^{[r \times c]}$ means a matrix which has r rows and c columns

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

    - Is a [4 x 2] matrix

Matrix

- Matrix elements
  - $A_{(i,j)}$ = entry in i^{th} row and jth column



$A_{11} = 1402$

$A_{12} = 191$

$A_{32} = 1437$

$A_{41} = 147$

- Provides a way to organize, index and access a lot of data

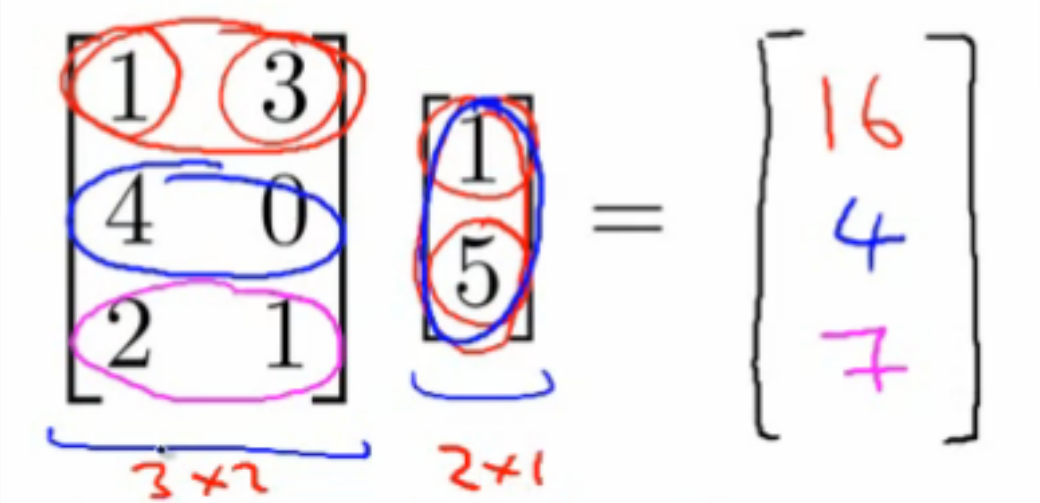
Matrix Addition

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0.5 \\ 4 & 10 \\ 3 & 2 \end{bmatrix}$$

Matrix Scalar Multiplication

$$3 \times \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 6 & 15 \\ 9 & 3 \end{bmatrix}$$

Matrix Vector Multiplication



$1 \times 1 + 3 \times 5 = 16$

$4 \times 1 + 0 \times 5 = 4$

$2 \times 1 + 1 \times 5 = 7$

Matrix Matrix Multiplication

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} = ?$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} = \begin{bmatrix} 11 \\ 9 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

Matrix Transpose

$$\underline{A} = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix} \qquad \underline{A^T} = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$$

Matrix Identity

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
2 x 2

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
3 x 3

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
4 x 4

- 1 is the identity for any scalar
  - i.e. 1 x z = z
    - for any real number
- In matrices we have an identity matrix called *I*
  - Sometimes called $I_{\{n \times n\}}$

- See some identity matrices above
  - Different identity matrix for each set of dimensions
  - Has
    - 1s along the diagonals
    - 0s everywhere else
  - 1x1 matrix is just "1"
- Remember that matrices are not commutative AB != BA
  - Except when B is the identity matrix
  - Then AB == BA

Matrix Multiplication properties

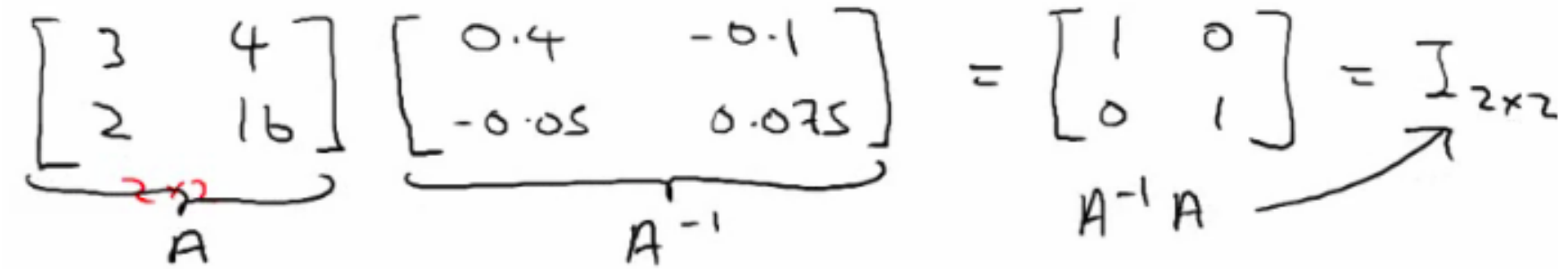
- Can pack a lot into one operation
  - However, should be careful of how you use those operations
  - Some interesting properties
- **Commutativity**
  - When working with raw numbers/scalars multiplication is commutative
    - 3 * 5 == 5 * 3
  - This is not true for matrix
    - A x B != B x A
    - **Matrix multiplication is not commutative**
- **Associativity**
  - 3 x 5 x 2 == 3 x 10 = 15 x 2
    - Associative property
  - **Matrix multiplications is associative**
    - A x (B x C) == (A x B) x C

Matrix Inverse

- How does the concept of "the inverse" relate to real numbers?
  - 1 = "identity element" (as mentioned above)
    - Each number has an inverse
      - This is the number you multiply a number by to get the identify element
      - i.e. if you have x, x * 1/x = 1
  - e.g. given the number 3
    - $3 * 3^{-1} = 1$ (the identity number/matrix)
  - In the space of real numbers **not everything has an inverse**
    - e.g. 0 does not have an inverse

Matrix Inverse

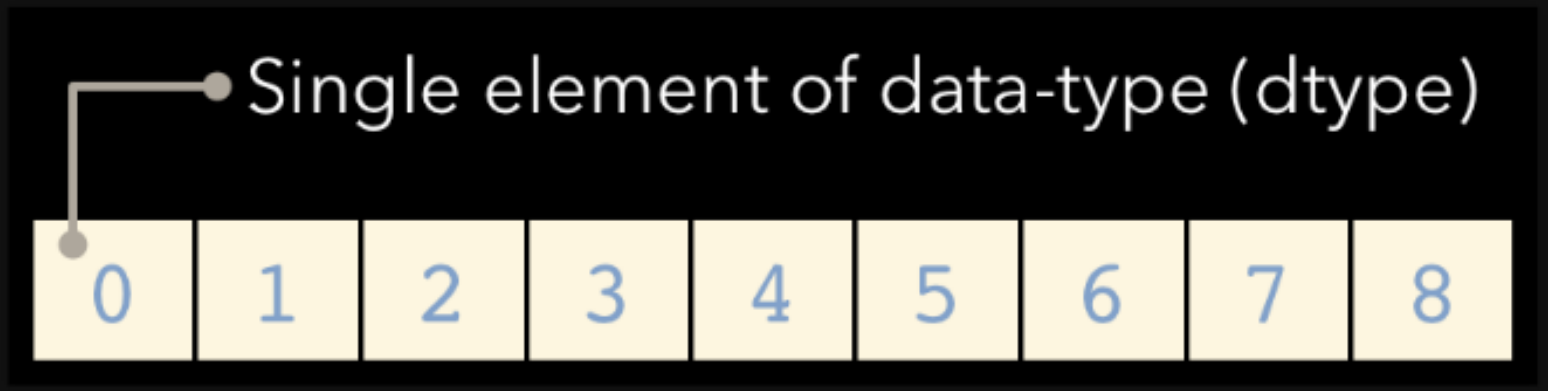
- What is the inverse of a matrix
  - If A is an m x m matrix, then A inverse = A^{-1}
  - So $A*A^{-1} = I$
  - Only matrices which are m x m have inverses
    - Square matrices only!
- Example
  - 2 x 2 matrix



Lineare Algebra with Numpy

What is Numpy?

- NumPy is a Python C extension library for array-oriented computing
  - Efficient
  - In-memory
  - Contiguous (or Strided)
  - Homogeneous (but types can be algebraic)

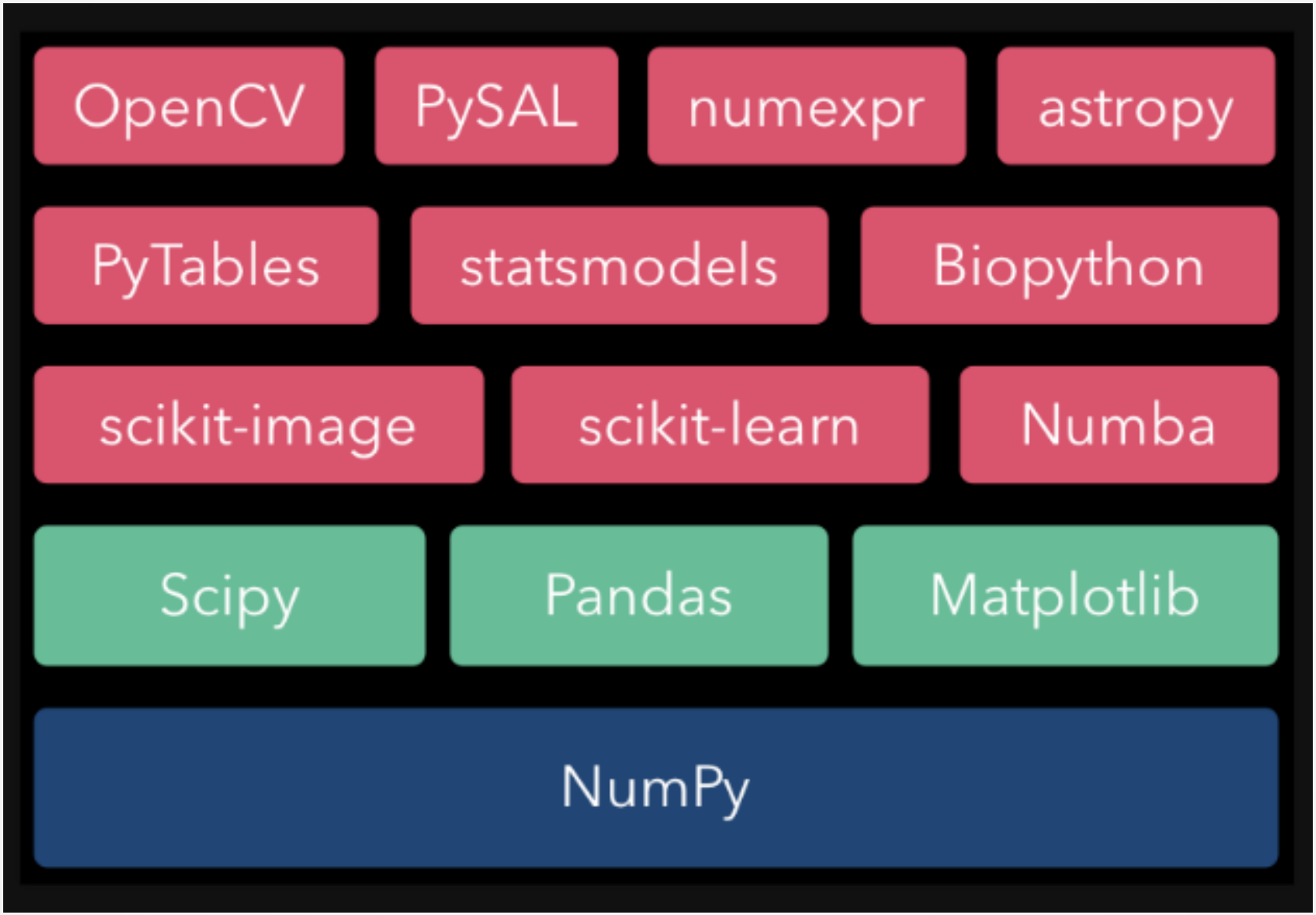


- NumPy is suited to many applications
  - Image processing
  - Signal processing
  - Linear algebra
  - A plethora of others

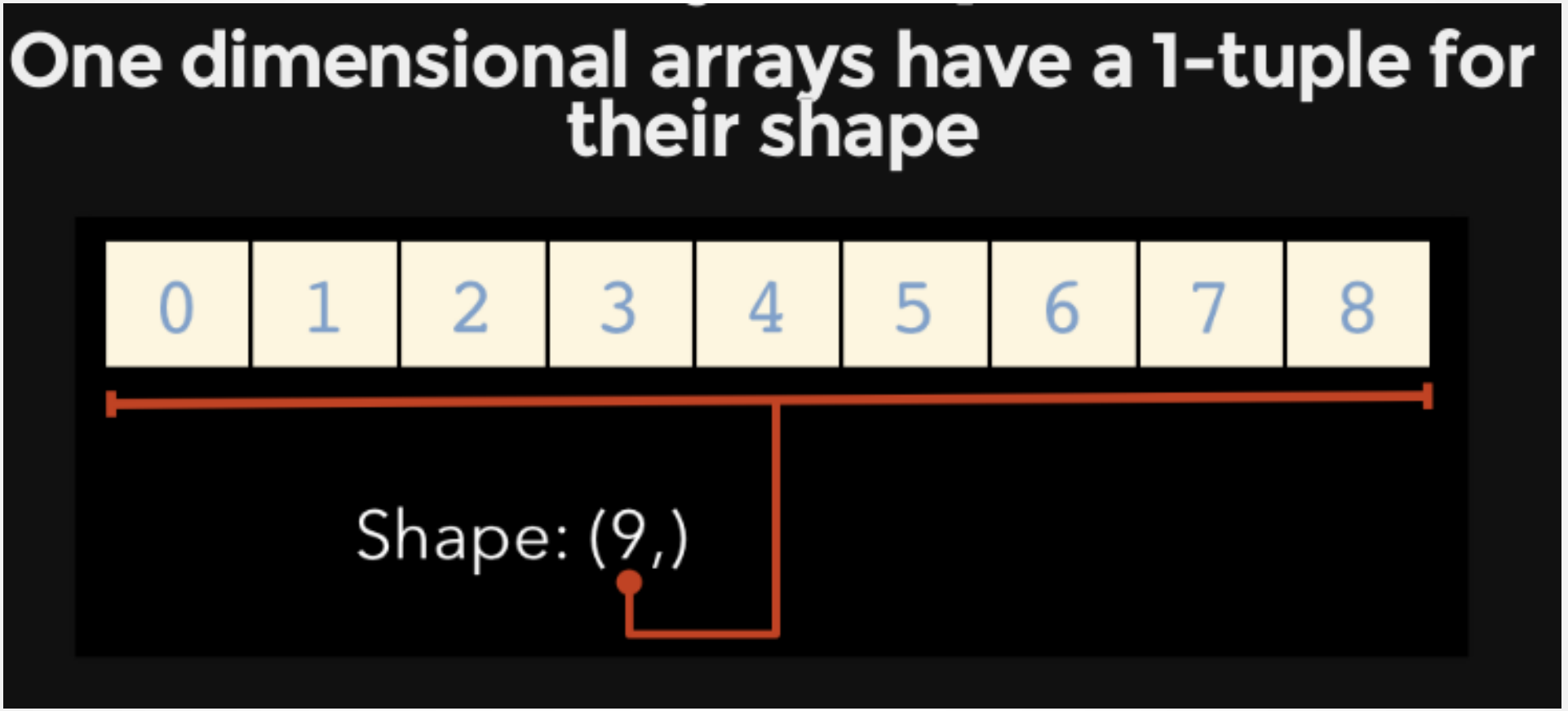
What is Numpy?

NumPy is the foundation of the python scientific stack

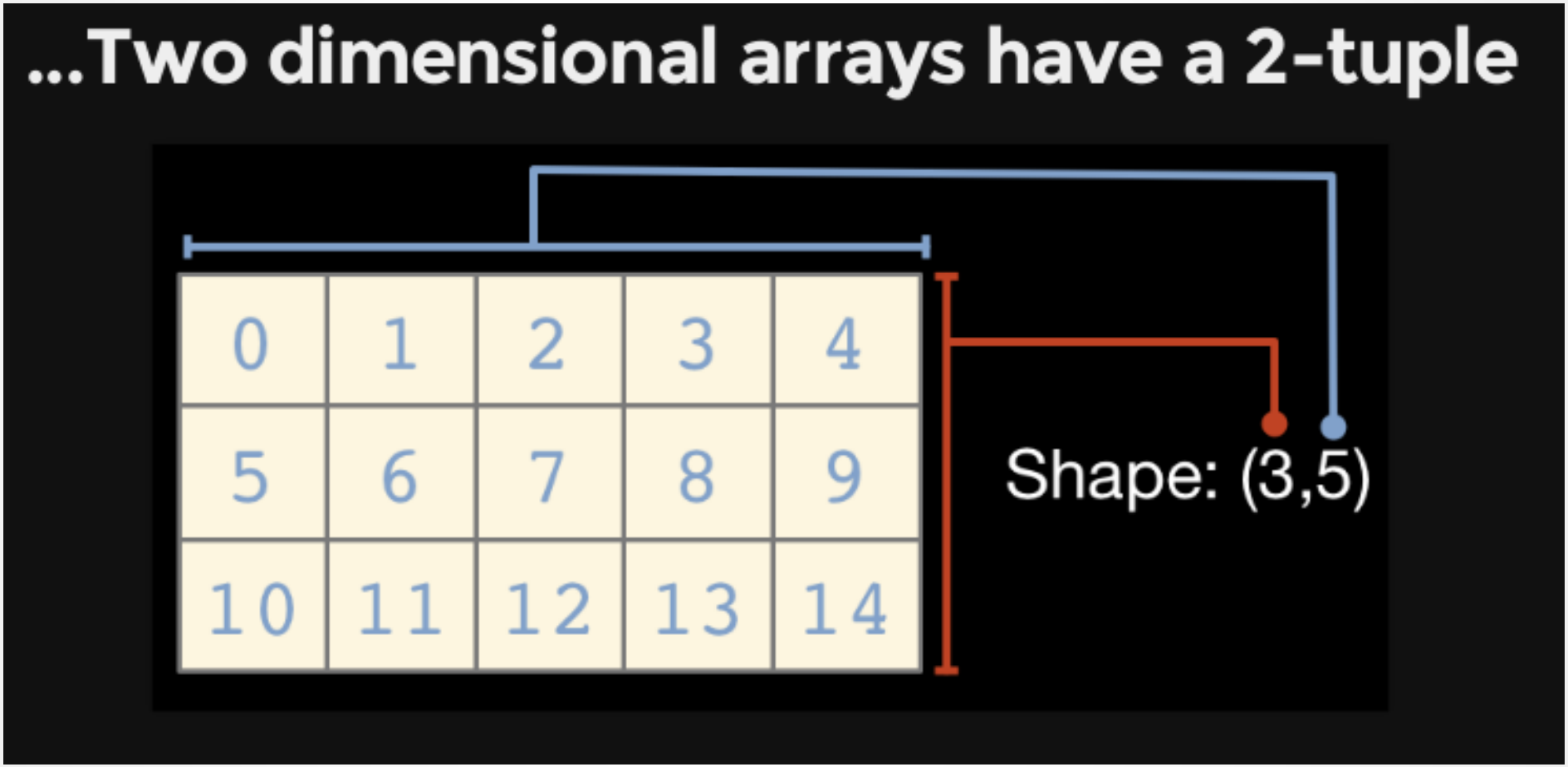
Numpy Ecosytem



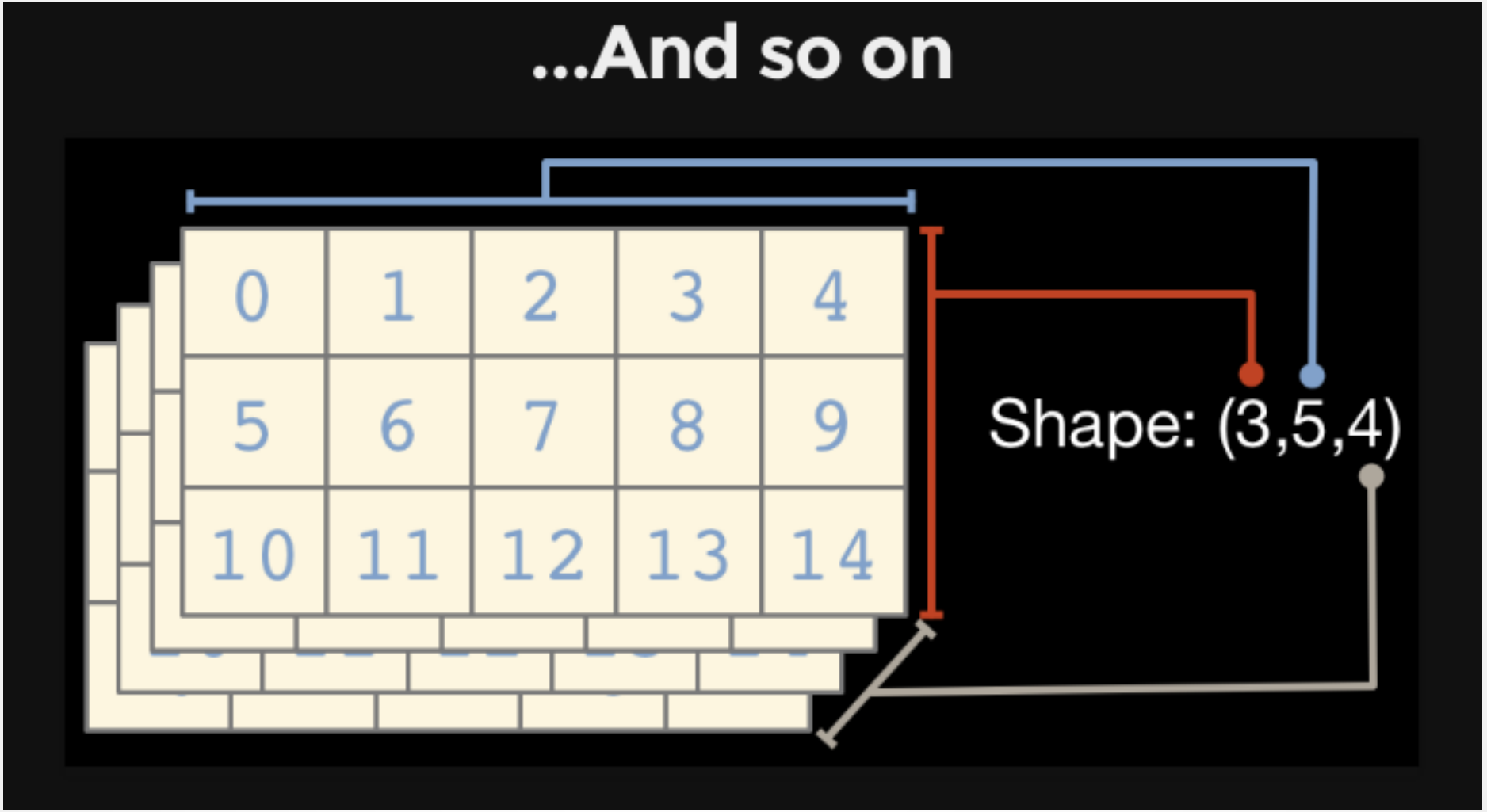
Array Shape


One dimensional arrays have a 1-tuple for their shape
0
1
2
3
4
5
6
7
8
Shape: (9,)

Array Shape


...Two dimensional arrays have a 2-tuple
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
Shape: (3,5)

Array Shape



Matrix and Vector

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
import numpy as np

M = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

v = np.array([[1],
              [2],
              [3]])
```

Matrix and Vector

- Import numpy as np
- M=np.array([[1,2,3], [4,5,6], [7,8,9]], dtype=np.float32)
- print(M.dtype)

Matrix and Vector

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
print M.shape
>>> (3, 3)

print v.shape
>>> (3, 1)

v_single_dim = np.array([1, 2, 3])
print v_single_dim.shape
>>> (3,)
```

Matrix and Vector

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
print v + v
>>> [[2]
     [4]
     [6]]

print 3*v
>>> [[3]
     [6]
     [9]]
```

Other Ways to Create Matrices and Vectors

NumPy provides many convenience functions for creating matrices/vectors.

```
a = np.zeros((2,2))  # Create an array of all zeros
print a              # Prints "[[ 0.  0.]
                     #          [ 0.  0.]]"

b = np.ones((1,2))   # Create an array of all ones
print b              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7) # Create a constant array
print c               # Prints "[[ 7.  7.]
                      #          [ 7.  7.]]"

d = np.eye(2)        # Create a 2x2 identity matrix
print d              # Prints "[[ 1.  0.]
                     #          [ 0.  1.]]"

e = np.random.random((2,2)) # Create an array filled with random values
print e                     # Might print "[[ 0.91940167  0.08143941]
                            #               [ 0.68744134  0.87236687]]"
```

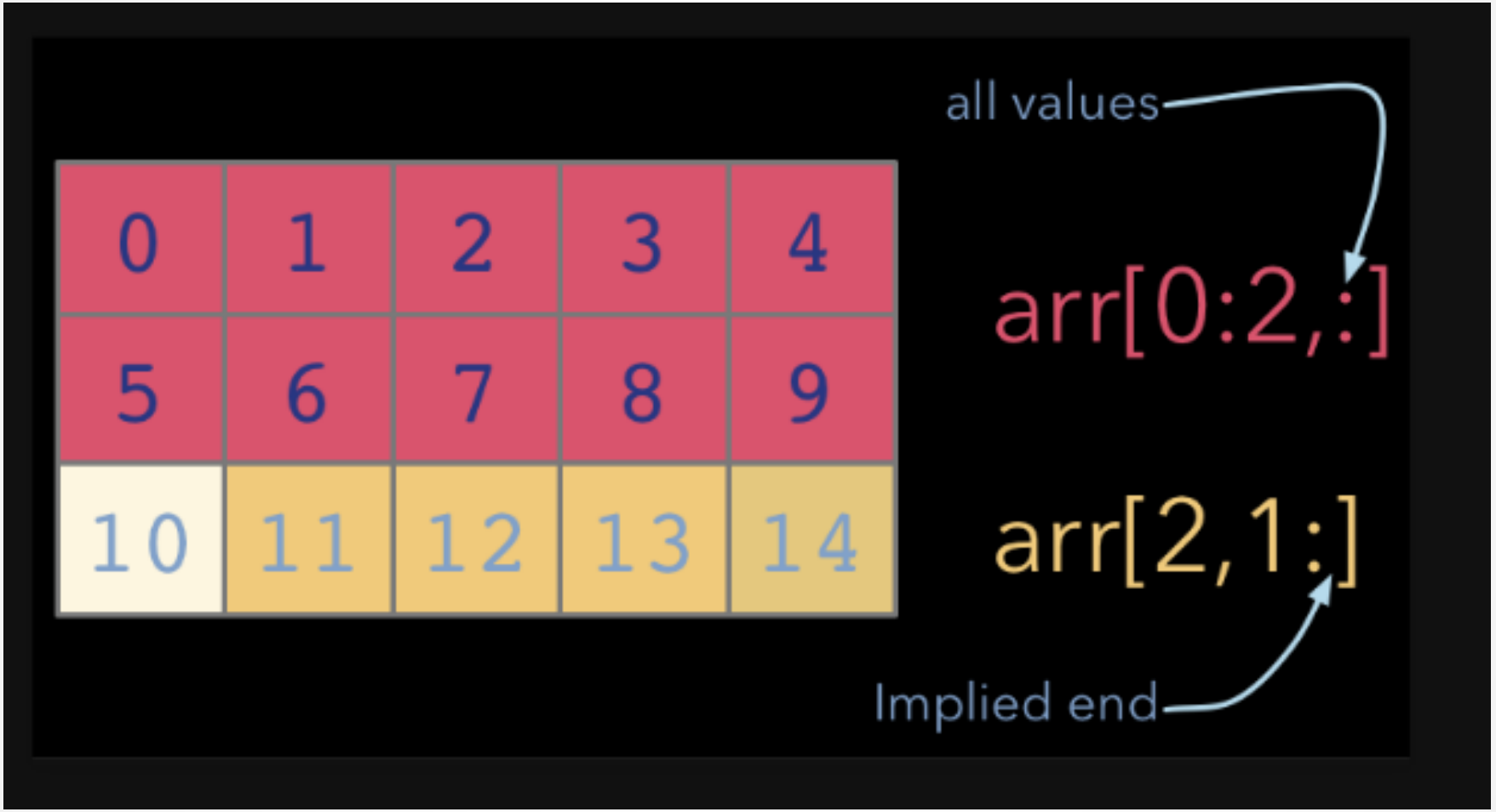
Other Ways to Create Matrices and Vectors

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

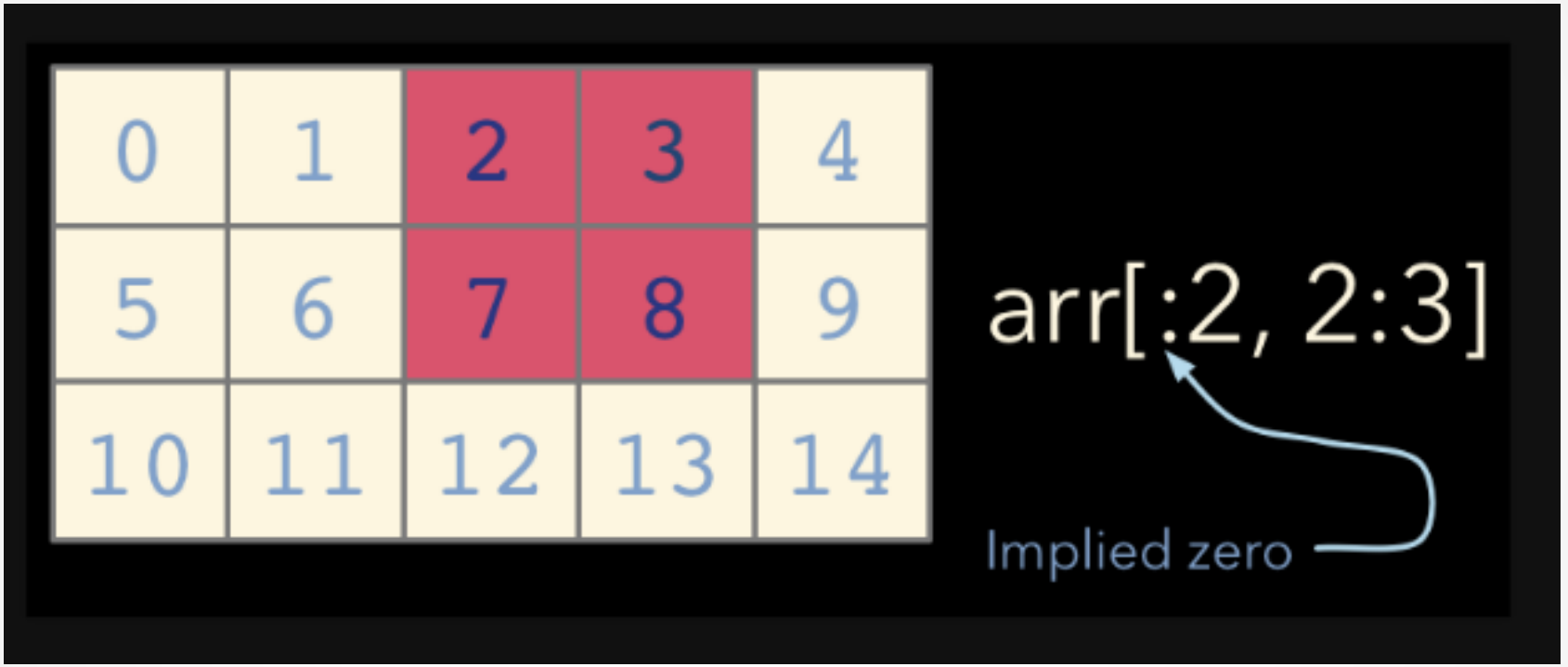
```
v1 = np.array([1, 2, 3])
v2 = np.array([4, 5, 6])
v3 = np.array([7, 8, 9])
M = np.vstack([v1, v2, v3])
print M
>>> [[1 2 3]
     [4 5 6]
     [7 8 9]]

# There is also a way to do this horizontally => hstack
```

Matrix indexing and slicing

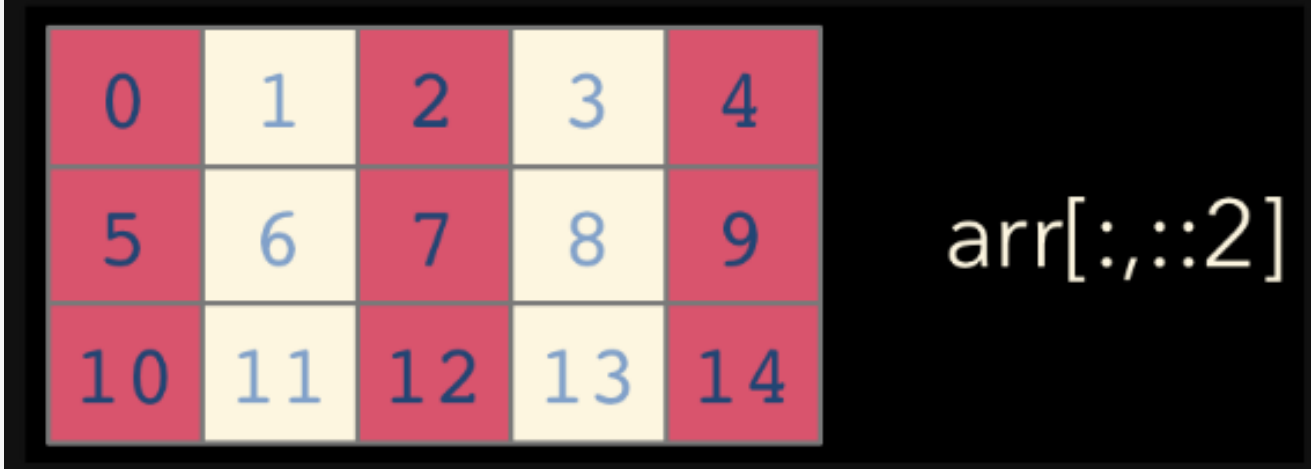


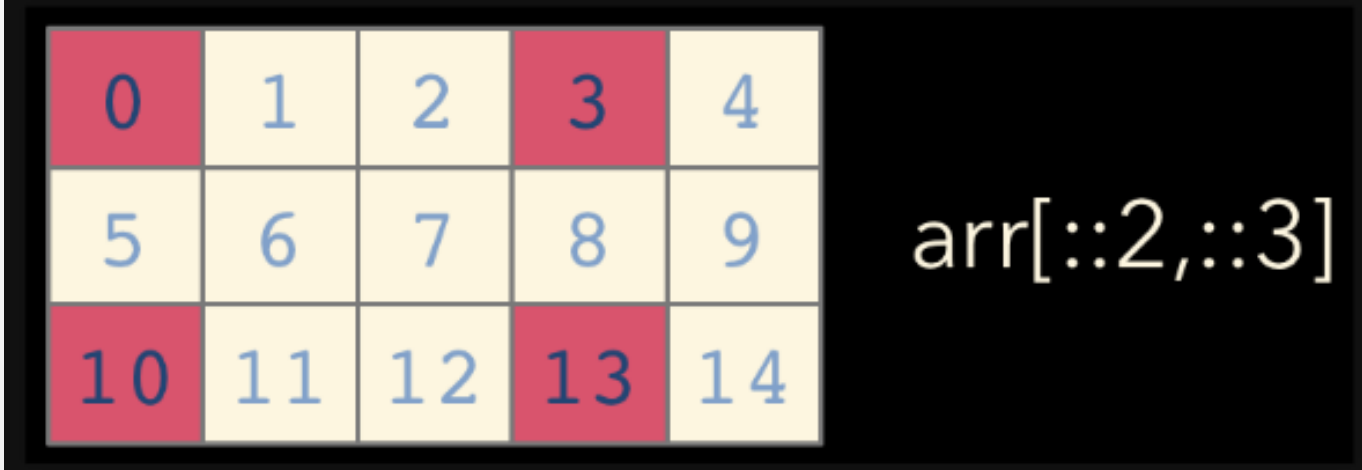
Matrix indexing and slicing



Matrix indexing and slicing

NumPy array indices can also take an optional stride





Matrix Reshape

```
>>> import numpy as np
>>> m=np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14])
>>> m
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
>>> rm=m.reshape((5,3))
>>> rm
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

Matrix Addition

```
>>> b = np.array([[1,0],[2,5],[3,1]])
>>> c = np.array([[4,0.5],[2,5],[0,1]])
>>> b
array([[1, 0],
       [2, 5],
       [3, 1]])
>>> c
array([[ 4. ,  0.5],
       [ 2. ,  5. ],
       [ 0. ,  1. ]])
>>> d=b+c
>>> d
array([[  5. ,   0.5],
       [  4. ,  10. ],
       [  3. ,   2. ]])
```

Scalar Multiplication and Addition

```
>>> a = np.array([[1,0],[2,5],[3,1]])
>>> add_a_3=a+3
>>> add_a_3
array([[4, 3],
       [5, 8],
       [6, 4]])
>>> mul_a_3=a*3
>>> mul_a_3
array([[ 3,  0],
       [ 6, 15],
       [ 9,  3]])
```

Combination of Operands

```
>>> e = np.array([[1],[4],[2]])
>>> f = np.array([[0],[0],[5]])
>>> g = np.array([[3],[0],[2]])
>>> h=3 * e + f - g / 3
>>> e
array([[1],
       [4],
       [2]])
>>> f
array([[0],
       [0],
       [5]])
>>> g
array([[3],
       [0],
       [2]])
>>> h
array([[  2.        ],
       [ 12.        ],
       [ 10.33333333]])
```

Matrix Vector Multiplication

We can multiply a matrix by a vector as long as the number of columns of the matrix is the same as the number of rows of the vector. In other words, the matrix must be as wide as the vector is long.

```
>>> h = np.array([[1,3],[4,0],[2,1]]) # 3x2
>>> i = np.array([[1],[5]]) # 2x1
>>> h*i
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: operands could not be broadcast together with shapes (3,2) (2,1)
>>> dot_h_i=np.dot(h,i)
>>> h
array([[1, 3],
       [4, 0],
       [2, 1]])
>>> i
array([[1],
       [5]])
>>> dot_h_i
array([[16],
       [ 4],
       [ 7]])
```

Matrix Matrix Multiplication

For Numpy matrix matrix multiplication is same as a matrix matrix multiplication

```
>>> l = np.array([[1,3,2],[4,0,1]])
>>> m = np.array([[1,3],[0,1],[5,2]])
>>> dot_l_m=np.dot(l,m)
>>> l
array([[1, 3, 2],
       [4, 0, 1]])
>>> m
array([[1, 3],
       [0, 1],
       [5, 2]])
>>> dot_l_m
array([[11, 10],
       [ 9, 14]])
```

Matrix Multiplication Properties

Show that matrix multiplication is not commutative

```
>>> A = np.array([[1,1],[0,0]]) # 2x2
>>> B = np.array([[0,0],[2,0]]) # 2x2
>>> dot_A_B=np.dot(A,B)
>>> dot_B_A=np.dot(B,A)
>>> A
array([[1, 1],
       [0, 0]])
>>> B
array([[0, 0],
       [2, 0]])
>>> dot_A_B
array([[2, 0],
       [0, 0]])
>>> dot_B_A
array([[0, 0],
       [2, 2]])
```

Identity Matrix

Show that for any matrix A, AI=IA=A.

```
>>> A = np.array([[4,2,1],[4,8,3],[1,1,0]]) # 3x3
>>> I = np.identity(3, dtype=int)
>>> dot_A_I=np.dot(A,I)
>>> dot_I_A=np.dot(I,A)
>>> A
array([[4, 2, 1],
       [4, 8, 3],
       [1, 1, 0]])
>>> I
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
>>> dot_A_I
array([[4, 2, 1],
       [4, 8, 3],
       [1, 1, 0]])
>>> dot_I_A
array([[4, 2, 1],
       [4, 8, 3],
       [1, 1, 0]])
```

Inverse Matrix

Show that if A is an mxm matrix, and if it has an inverse, then $A(A^{-1}) == (A^{-1})A == I$.

```
>>> A = np.array([[4,2,1],[4,8,3],[1,1,0]]) # 3x3
>>> inv_A=np.linalg.inv(A)
>>> A
array([[4, 2, 1],
       [4, 8, 3],
       [1, 1, 0]])
>>> inv_A
array([[ 0.3, -0.1,  0.2],
       [-0.3,  0.1,  0.8],
       [ 0.4,  0.2, -2.4]])
>>> np.dot(A,inv_A)
array([[  1.00000000e+00,  -2.77555756e-17,   0.00000000e+00],
       [ -1.66533454e-16,   1.00000000e+00,   0.00000000e+00],
       [ -5.55111512e-17,  -1.38777878e-17,   1.00000000e+00]])
>>> np.dot(inv_A,A)
array([[  1.00000000e+00,   0.00000000e+00,  -2.77555756e-17],
       [ -2.22044605e-16,   1.00000000e+00,  -6.93889390e-17],
       [  0.00000000e+00,   0.00000000e+00,   1.00000000e+00]])
```

Transpose Matrix

```
>>> A = np.array([[1,2,0],[3,5,9]])
>>> A_T=A.T
>>> A
array([[1, 2, 0],
       [3, 5, 9]])
>>> A_T
array([[1, 3],
       [2, 5],
       [0, 9]])
```

Numpy functions

```
>>> A = np.array([[1,2,7],[3,5,9]])
>>> log_A=np.log(A)
>>> pow_A=np.power(A,2)
>>> A
array([[1, 2, 7],
       [3, 5, 9]])
>>> log_A
array([[ 0.        ,  0.69314718,  1.94591015],
       [ 1.09861229,  1.60943791,  2.19722458]])
>>> pow_A
array([[ 1,  4, 49],
       [ 9, 25, 81]])
```

NumPy has many built-in ufuncs

- comparison: `<, <=, ==, !=, >=, >`
- arithmetic: `+, -, *, /, reciprocal, square`
- exponential: `exp, expm1, exp2, log, log10, log1p, log2, power, sqrt`
- trigonometric: `sin, cos, tan, acsin, arccos, atctan`
- hyperbolic: `sinh, cosh, tanh, acsinh, arccosh, atctanh`
- bitwise operations: `&, |, ~, ^, left_shift, right_shift`
- logical operations: `and, logical_xor, not, or`
- predicates: `isfinite, isinf, isnan, signbit`
- other: `abs, ceil, floor, mod, modf, round, sinc, sign, trunc`

Array Methods

- Predicates
  - `a.any()`, `a.all()`
- Reductions
  - `a.mean()`, `a.argmin()`, `a.argmax()`, `a.trace()`, `a.cumsum()`, `a.cumprod()`
- Manipulation
  - `a.argsort()`, `a.transpose()`, `a.reshape(...)`, `a.ravel()`, `a.fill(...)`, `a.clip(...)`
- Complex Numbers
  - `a.real`, `a.imag`, `a.conj()`

Numpy Functions

- Data I/O
  - `fromfile, genfromtxt, load, loadtxt, save, savetxt`
- Mesh Creation
  - `mgrid, meshgrid, ogrid`
- Manipulation
  - `einsum, hstack, take, vstack`

Other Subpackages

- `numpy.fft` — Fast Fourier transforms
- `numpy.polynomial` — Efficient polynomials
- `numpy.linalg` — Linear algebra
  - `cholesky, det, eig, eigvals, inv, lstsq, norm, qr, svd`
- `numpy.math` — C standard library math functions
- `numpy.random` — Random number generation
  - `beta, gamma, geometric, hypergeometric, lognormal, normal, poisson, uniform, weibull`