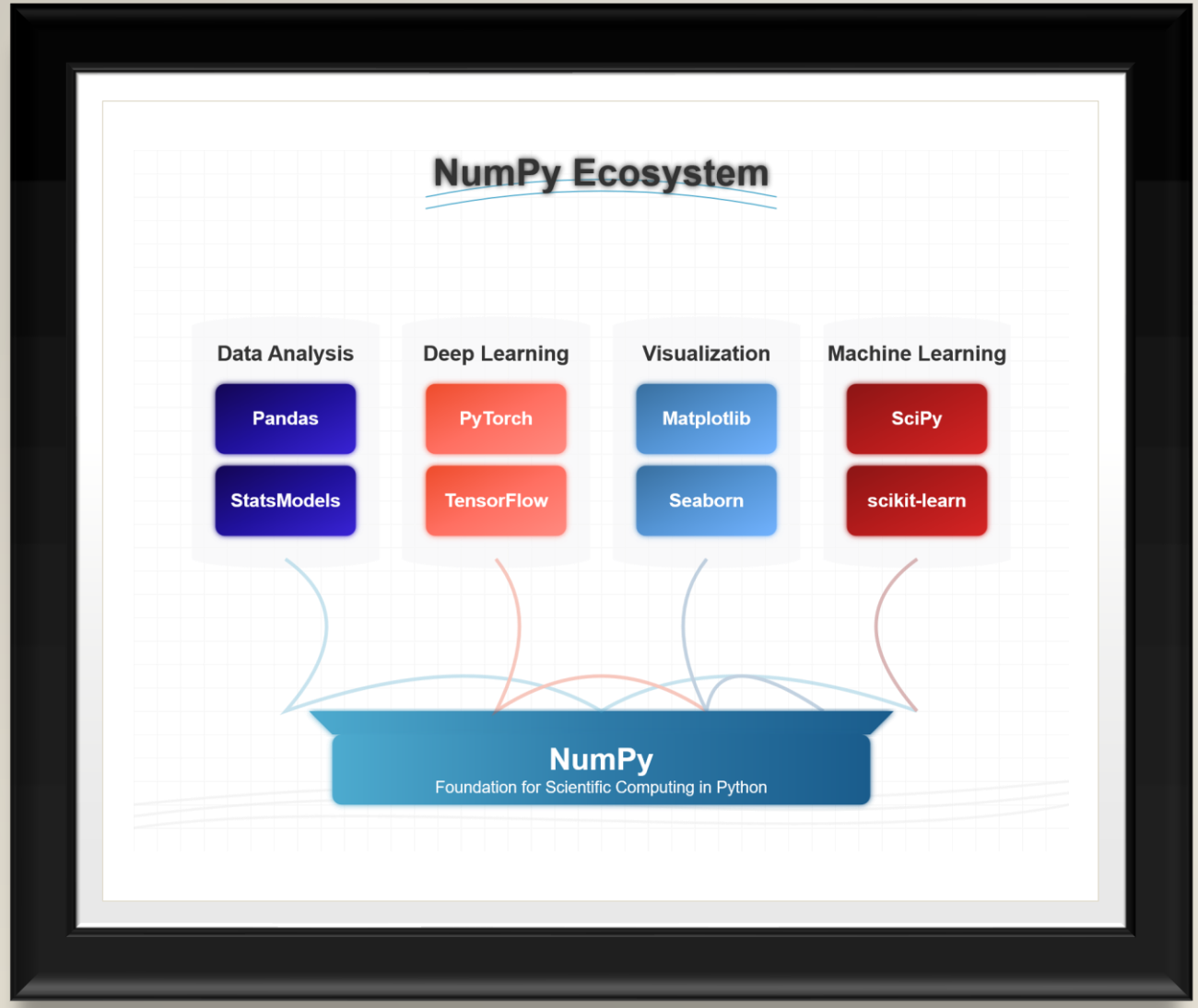
NUMPY

NUMPY

- NumPy, short for "Numerical Python," is a fundamental package for scientific computing in Python.
- It is widely used in data analysis due to its ability to handle large, multi-dimensional arrays and matrices, along with a variety of mathematical functions that can be applied to these arrays efficiently.



NumPy serves as the foundation for many other data science libraries such as Pandas, SciPy, and scikit-learn, making it an essential tool for anyone working with data in Python.

KEY FEATURES

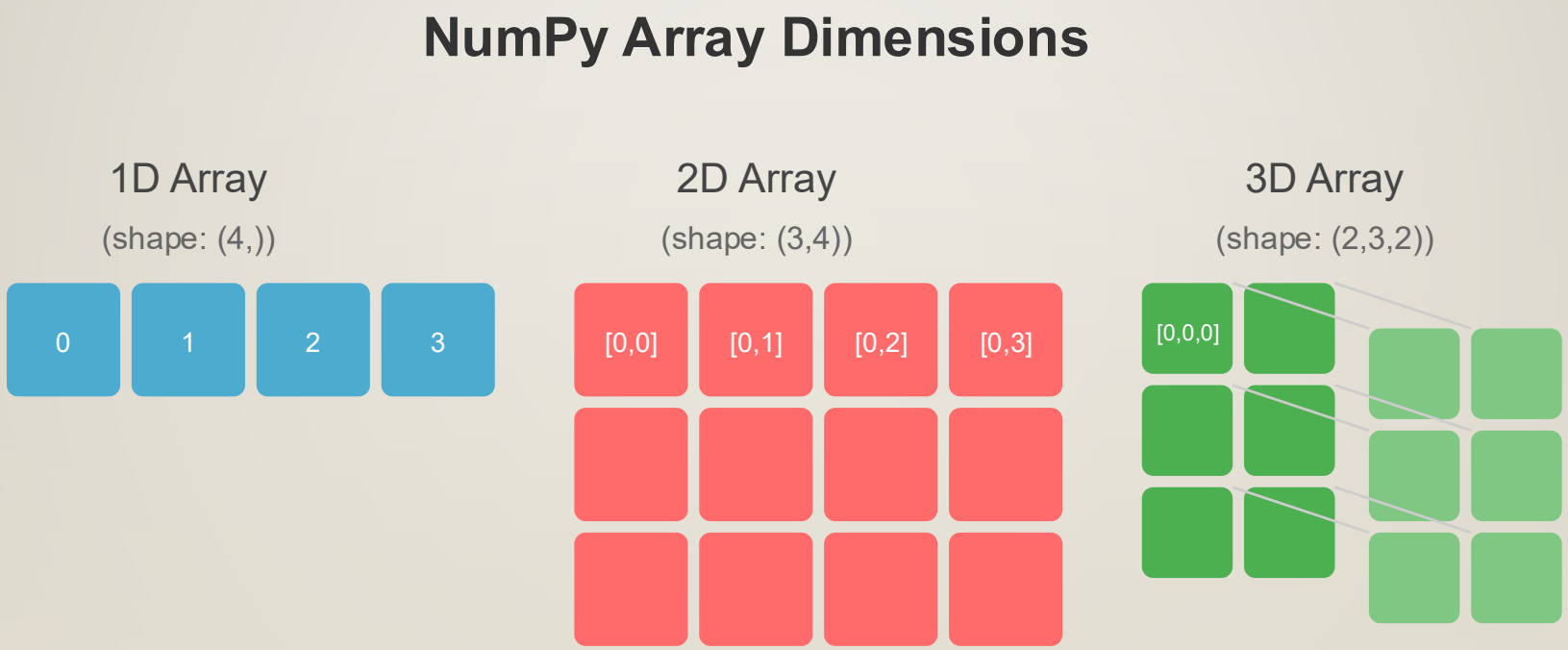
- N-Dimensional Array Object (ndarray): Efficient handling of arrays (1D, 2D, and multi-dimensional).
- Mathematical Functions: Fast mathematical operations on entire arrays of data without having to write loops
- Data Simulation: Generate random numbers for simulations and sampling.
- Linear Algebra Operations: Includes matrix operations, eigenvalue computation, and more.
- Broadcasting: Simplifies operations on arrays of different shapes.
- Integration with Other Libraries: Easily integrates with data science libraries like Pandas and SciPy.

OUR FOCUS

We will focus only on features directly related to data analysis

- Fast array-based operations for data wrangling such as subsetting, filtering, transformation, and cleaning.
- Functions for performing descriptive statistics such as mean, and standard deviation.
- Ways to combine and restructure datasets.

CREATING AN ARRAY



ARRAY

Names for these arrays in Math

- 1d array = vector
- 2d array = matrix
- 3d+ array = tensor

CREATING AN ARRAY

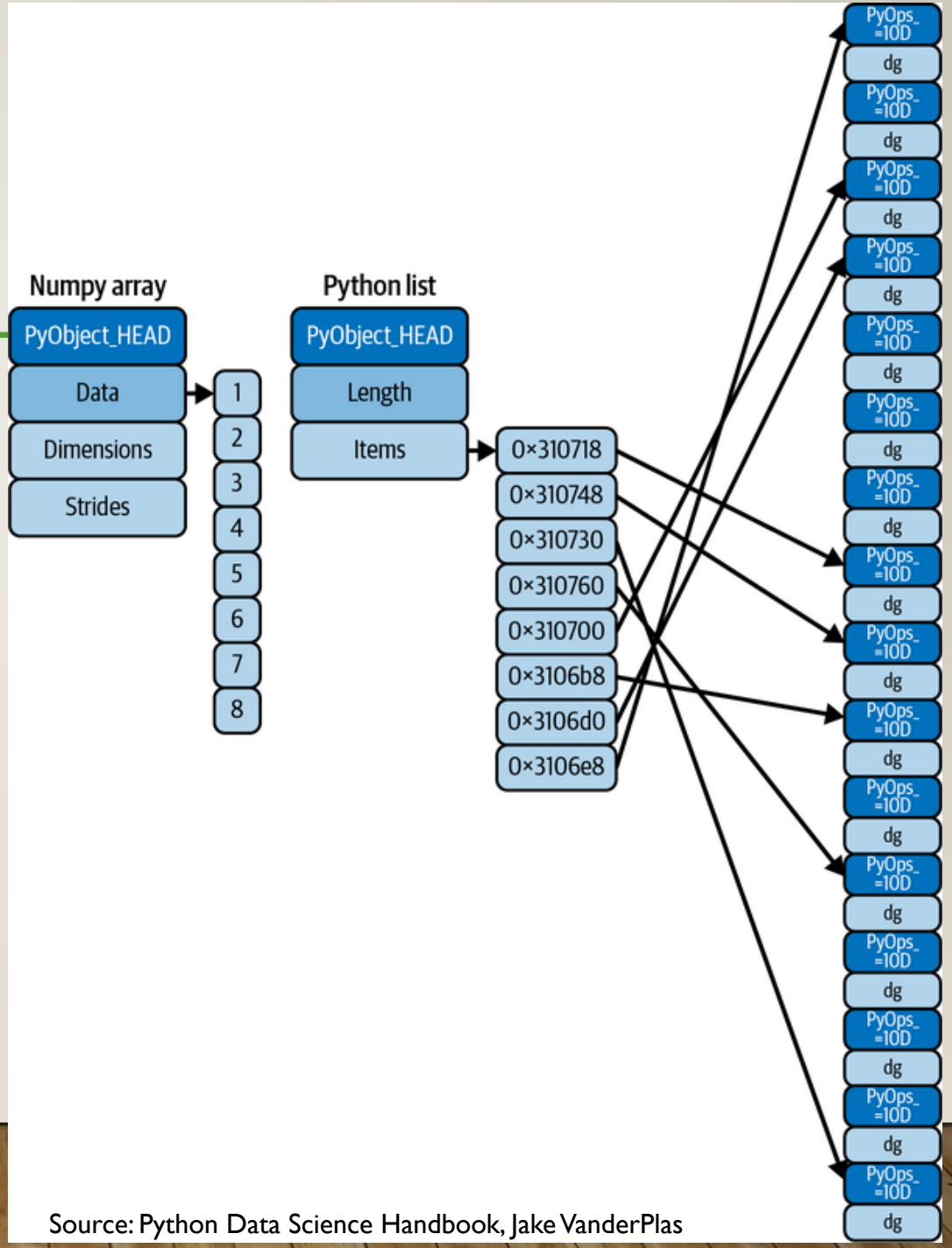
- Numpy arrays can be created by
  - From a list using np.array()
  - shortcut functions like np.arange, np.zeros(), np.full()

DIMENSIONS

- NumPy is an ndarray, which means it can have one or more dimensions
  - .shape contains information about dimensionality
- An n-dimensional array can be thought of as a set of elements on which a structure is imposed
- Shape can be changed
  - .flatten() converts an n-dimensional array into a 1-d array-
  - .reshape() can change the structure (if it is possible)

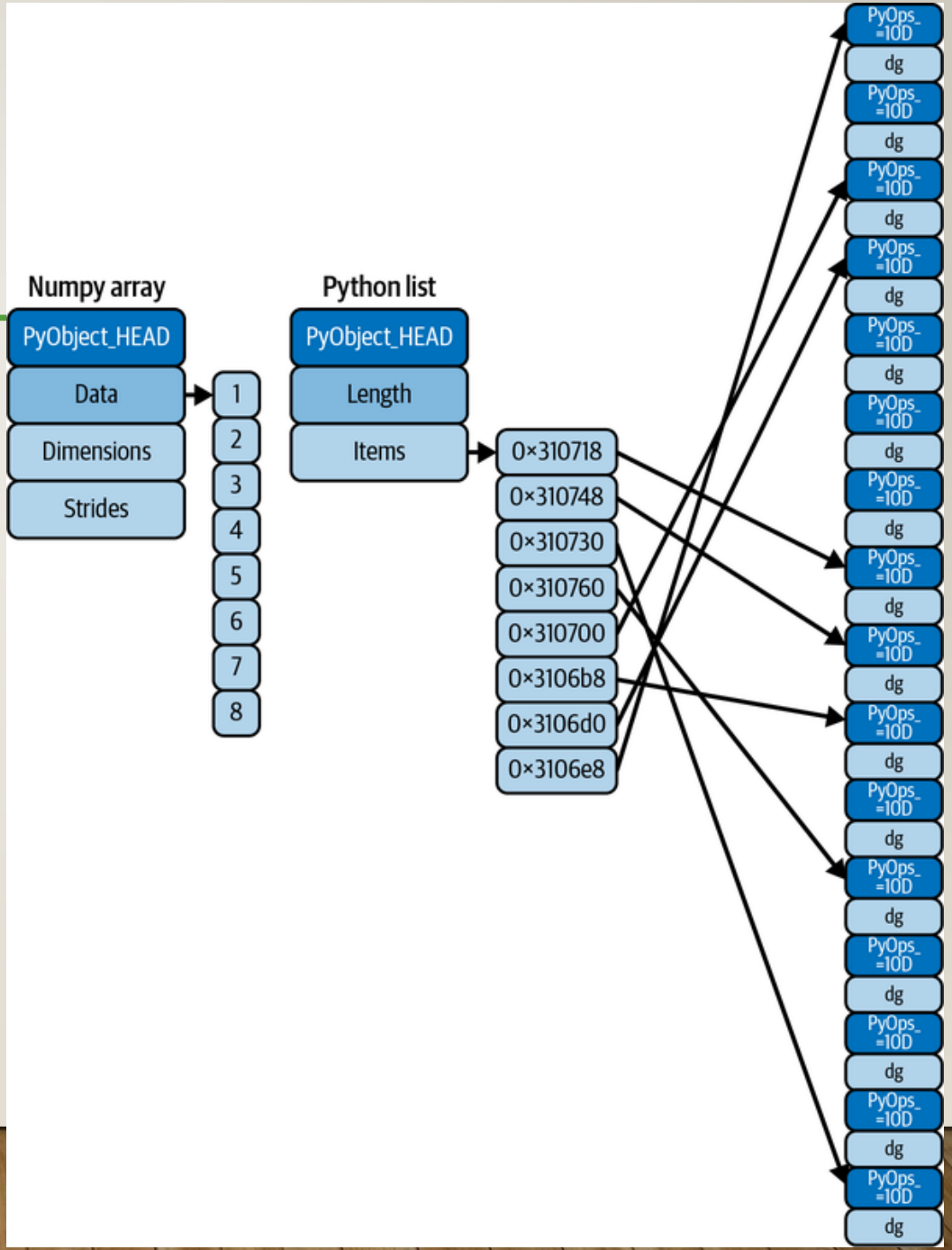
DTYPE

- The computational speed of NumPy arrays is attributable to the type constraints.
- A NumPy array can only contain data of a single type.
- Unless explicitly specified, the array structure will infer a suitable data type for the array that it creates.
- The data type is stored in a special *dtype* medadata object.



Source: Python Data Science Handbook, Jake VanderPlas

NUMPY VS.


Numpy array
PyObject_HEAD
Data
Dimensions
Strides
1
2
3
4
5
6
7
8
Python list
PyObject_HEAD
Length
Items
0×310718
0×310748
0×310730
0×310760
0×310700
0×3106b8
0×3106d0
0×3106e8
PyOps_
=10D
dg

DTYPE

- Here are some of the commonly used data types (dtype).
  - Numbers: int8, int16, int32, int64, float16, float32, float64, float128. Python as such only allows int and float!
  - Complex: complex64, complex128, complex256
  - Boolean: bool
  - Text: <U6, <U12, etc. This is a unicode str with a length constraint, as opposed to generic str
- It is possible to convert one data type to another using astype()
- Since NumPy arrays can contain data of only one type, if NumPy encounters elements of differnt types, it will convert them to the same type.
- If a dtype argument is not passed, NumPy will examine the data and automatically assign a data type

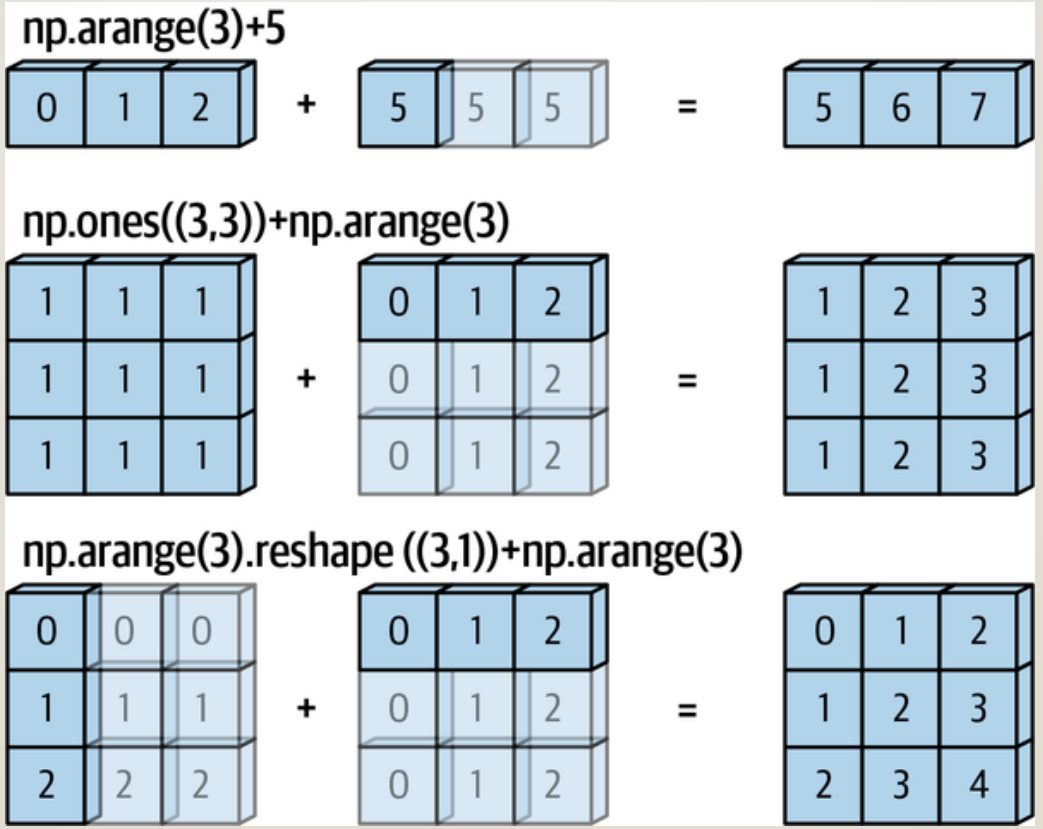
ELEMENT WISE OPERATION

- Lists in Python do not do elementwise operation.
- On the other hand, for numpy arrays, elementwise operation is the default.
- Elementwise operations can be conducted using loops, but loops in Python are slow. Instead numpy conducts vectorized operations where in the operation is carried out in C, a low-level language.

BROADCASTING

- In math, one can only add two matrices of the same shape. Numpy will allow addition of arrays of different shapes (but not any shape).
- Numpy can only add compatible arrays. The arrays should have the same shape except for one dimension. The dimension that does not have the same value must be 1. E.g, (4,4) and (4,1); (4,4) and (1,1); but not (4,4) and (2,2)
- Python will not reshape an array under the hood to make the computation work. For instance, if you intend to use a vertical array, then you would need to explicitly reshape a 1-d array with n elements to an array with shape, (n,1).
- Broadcasting can be done for all arithmetic operations, not only addition.
- Broadcasting is similar to (but less inclusive) than recycling behavior in R.

BROADCASTING



INDEXING 1-D ARRAY

- Extract a single element by specifying location index in square brackets
- numpy array indices begin at 0
- Negative indices count from the end of the array

INDEXING AND SLICING

0	1	2	3	4	5	6	7	8	9
6	0	8	8	0	3	3	5	4	2

INDEXING 2-D ARRAY

- Extract a single element by specifying row and column location index in square brackets separated by a comma
- numpy array indices begin at 0
- Negative indices count from the end of the array

INDEXING AND SLICING

	0	1	2	3	4	5	6	7	8	9
0	6	0	8	8	0	3	3	5	4	2
1	2	5	8	0	3	8	4	5	0	5
2	7	6	0	3	3	0	8	3	3	5
3	5	0	7	2	3	0	1	3	3	6
4	3	7	0	1	2	4	3	6	3	1
5	4	4	3	8	2	7	6	5	8	1
6	7	5	1	1	4	5	0	8	5	4
7	0	2	2	3	4	8	2	0	8	6
8	0	0	6	1	8	6	3	0	5	6
9	2	4	3	3	6	6	6	4	3	1

SLICING 1-D ARRAY

- Extract more than one element
- Specify more than one location as a list. [[1,2,3,4]]
- Specify a range of locations. First element is included, last is not. [1:5]
- Use colon to refer to everything before, or everything after. [:5]
- Negative indexes to begin from the end [-9:-5]

SLICING 2-D ARRAY

- Extract more than one element
- Specify a range of locations for rows and columns. First element is included, last is not. [3:7,3:7]
- Use colon to refer to everything before, or everything after. [:3,7:]
- Negative indexes to begin from the end [-10:-7,-3:]
- You can also slice with steps by specifying the start, end and step value separated by colon. [3:7:3,3:7:3]

BOOLEAN INDEXING

- Arrays are subset based on a condition.

FIND AND REPLACE

- numpy.where() can be used to find an element. It can also be used to find an element and replace it with another

USEFUL NUMPY METHODS

- .concatenate(): To combine two or more arrays with compatible shapes. Does not work with 1d arrays.
- .delete(): Remove a column or row
- .sort(): Sort a ndarray by rows (axis = 0) or columns (axis = 1)
- .transpose(): swaps axes, essentially reshaping the data
- .flip(): reverses the order of elements along a given axis
- .stack(): Combine arrays by adding a new dimension
- .split(): Split an array into multiple subarrays

VECTORIZED OPERATIONS AND UFUNCS

- Python's default implementation (known as CPython) does some operations very slowly. This is partly due to the dynamic, interpreted nature of the language; types are flexible, so sequences of operations cannot be compiled down to efficient machine code as in languages like C and Fortran. This relative sluggishness of Python can generally be seen in situations where many small operations are being repeated; for instance, looping over arrays to operate on each element.
- For many types of operations, NumPy provides a convenient interface into just this kind of statically typed, compiled routine. This is known as a vectorized operation. For simple operations like the element-wise division here, vectorization is as simple as using Python arithmetic operators directly on the array object. This vectorized approach is designed to push the loop into the compiled layer that underlies NumPy, leading to much faster execution.

UFUNCS

- Vectorized operations in NumPy are implemented via ufuncs, whose main purpose is to quickly execute repeated operations on values in NumPy arrays. Ufuncs are extremely flexible. Here are a few commonly used Ufuncs. The names are intuitive.
- Unary functions: mod, abs, sqrt, square, exp, log, log10, sign, ceil, floor, rint (round to the nearest integer), modf (return fractional and integeral parts of array), isnan (wether each value is NaN), isfinite, isinf, cos, cosh, sin, sinh, tan, tanh, logical_not
- Binary functions: add, subtract, multiply, divide, floor_divide, power, maximum, minimum, mod, greater, greater_equal, less, less_equal, equal, not_equal, logical_and, logical_or, logical_xor.
- Statistical Methods: sum, mean, std, var, min, max, cumsum, cumprod
- Other Methods: sort, unique, intersect1d, union1dalent to ~arr)

VECTORIZED OPERATIONS

- Elementwise operations can be conducted using loops, but loops in Python are slow.
- Instead numpy conducts vectorized operations where in the operation is carried out in C, a low-level language.
- Native Python functions can be vectorized by passing them to numpy.vectorize()

UNIVERSAL FUNCTIONS

Universal functions perform elementwise operations on ndarrays. Here are a few commonly used functions. The names are intuitive.

- Unary functions: abs, sqrt, square, exp, log, log10, sign, ceil, floor, rint (round to the nearest integer), modf (return fractional and integeral parts of array), isnan (wether each value is NaN), isfinite, isinf, cos, cosh, sin, sinh, tan, tanh, logical_not
- Binary functions: add, subtract, multiply, divide, floor_divide, power, maximum, minimum, mod, greater, greater_equal, less, less_equal, equal, not_equal, logical_and, logical_or, logical_xor.
- Statistical Methods: sum, mean, std, var, min, max, cumsum, cumprod
- Other Methods: sort, unique, intersect1d, union1dalent to ~arr

SUMMARY

- numpy makes Python a powerful data analysis language
  - Useful data structures
  - Efficient vectorized operations
  - Useful data-analysis functions
  - Foundation for other data analysis libraries such as pandas and scikit-learn.

REFERENCES

- Python for Data Analysis by Wes McKinney. O'Reilly Media, 2023.
- Python Data Science Handbook, 2nd Edition by Jake VanderPlas, O'Reilly Media, 2022.