NumPy and NumPy Arrays

What is NumPy?

NumPy, which stands for Numerical Python, is a powerful library for numerical and mathematical operations in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays.

Originally, Numeric, crafted by Jim Hugunin, served as the precursor to NumPy. Another package, Numarray, was concurrently developed, offering additional functionalities. In 2005, Travis Oliphant integrated the features of Numarray into the Numeric package, giving rise to the creation of the NumPy package. Numerous contributors have played a role in advancing this open-source.

Data analysis, enhanced by tools like NumPy, involves inspecting, cleaning, transforming, and modeling data. NumPy facilitates efficient numerical operations and array handling, aiding in discovering insights and supporting decision-making processes.

Operations with NumPy: With NumPy, developers can execute the following operations:

- Mathematical and logical operations on arrays.
- **Fourier transforms** and functions for manipulating shapes.
- Operations associated with **linear algebra**. NumPy incorporates built-in functions for both linear algebra and random number generation. **Numpy**

Setting Up NumPy Environment:

The regular Python setup doesn't include NumPy. A simple solution is to add NumPy to your Python using the popular installer called **pip**.

```
pip install numpy
```

You can check if NumPy is installed successfully by opening a Python interpreter and trying to import NumPy:

```
import numpy as np
```

If there are no error messages, NumPy is successfully installed.

To effectively activate NumPy, it is recommended to install a binary package tailored for your operating system. These packages encompass the complete SciPy stack, incorporating NumPy, SciPy, **matplotlib**, **IPython**, **SymPy**, and **nose packages**, in addition to the core Python components.

For Windows-

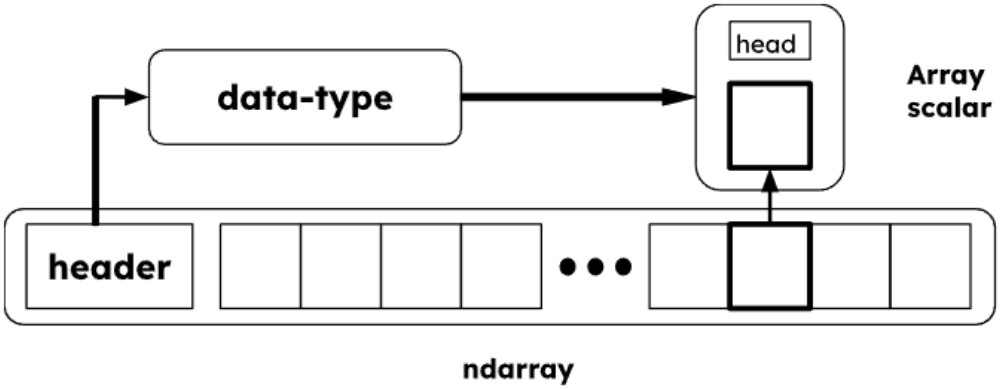
- **Anaconda**, accessible at continuum.io, provides a complimentary Python distribution encompassing the complete SciPy stack, and it is compatible with Linux and Mac operating systems.
- Canopy, accessible at enthought.com/products/canopy/, offers both a free and a commercial distribution that includes the entire SciPy stack, catering to Windows, Linux, and Mac users.
- Python(x, y), available for free, is a Python distribution equipped with the SciPy stack and the Spyder IDE, designed specifically for the Windows operating system. It can be downloaded from python-xy.github.io/.

Linux users can employ the package managers of their respective distributions to install individual or multiple packages within the **SciPy stack**.

Numpy Arrays: It is an N-dimensional array designed to hold a collection of items of the same data type. Accessing these items is achieved using a zero-based indexing scheme. It is also called ndarray. Here are key characteristics and concepts related to NumPy arrays:

1. **Homogeneous Data Types**: NumPy arrays have a fixed data type, which ensures that all elements in the array are of the same type. This allows for efficient storage and computation.

2. **Multidimensional**: NumPy arrays can have multiple dimensions (axes). For example, a 1-dimensional array is like a list, a 2-dimensional array is like a matrix, and a 3-dimensional array is like a cube of numbers.
3. **Zero-Based Indexing:** NumPy arrays use a zero-based indexing scheme, which means the index of the first element is 0, the second element is at index 1, and so on.
4. **Broadcasting**: NumPy supports broadcasting, a powerful mechanism that allows operations between arrays of different shapes and sizes. Broadcasting automatically adjusts the dimensions of smaller arrays to perform element-wise operations with larger arrays.
5. **Vectorized Operations**: NumPy supports vectorized operations, which means that you can perform operations on entire arrays without the need for explicit loops.
6. **Memory Efficiency and Data Type Specification**: Each element within a ndarray occupies a fixed-size memory block, and its data type is specified by the **dtype** attribute. When extracting a portion of a ndarray using slicing, the resulting object belongs to one of the **array scalar types**.



The dtype defines the kind of data that can be stored in the ndarray, such as integers, floats, or strings. The header stores information about the ndarray, such as its shape and size.

NumPy arrays, represented by the ndarray object, form the backbone of numerical computing in Python. They offer a convenient and efficient way to work with large datasets, enabling a wide range of mathematical and scientific operations.

Numpy Datatypes: NumPy offers a broader range of numerical types compared to Python. The table below illustrates some common data types defined in NumPy.

1. **bool_**: Represents Boolean values (True or False) stored as a byte.
2. **int_**: Default integer type, akin to C long; typically, int64 or int32.
3. **intc**: Corresponds to C int, usually int32 or int64.
4. **int8**: Represents a byte, ranging from -128 to 127.
5. **int16**: Represents a 16-bit integer, ranging from -32768 to 32767.
6. **int32**: Represents a 32-bit integer, ranging from -2147483648 to 2147483647.
7. **int64**: Represents a 64-bit integer, ranging from -9223372036854775808 to 9223372036854775807.
8. **uint8**: Represents an unsigned 8-bit integer, ranging from 0 to 255.
9. **float_**: Shorthand for float64, representing double precision floating-point numbers.
10. **float16**: Represents a half-precision float with a sign bit, 5 bits exponent, and 10 bits mantissa.
11. **float32**: Represents a single-precision float with a sign bit, 8 bits exponent, and 23 bits mantissa.
12. **float64**: Represents a double-precision float with a sign bit, 11 bits exponent, and 52 bits mantissa.
13. **complex_**: Shorthand for complex128, representing double precision complex numbers

Data Type Objects (dtype): Data type Objects in NumPy define how to interpret a fixed block of memory corresponding to an array. The interpretation depends on several factors:

1. **Type of data:** Specifies whether the data is an integer, float, or a Python object.
2. **Size of data:** Defines the size of the data in terms of bytes.
3. **Byte order:** Determines the byte order, whether it is little-endian or big-endian.
4. **For structured types:** Includes the names of fields, the data type of each field, and the portion of the memory block occupied by each field.
5. **For subarrays:** Specifies the shape and data type of the subarray.

Syntax: Construct a dtype object

```
numpy.dtype(object, align, copy)
```

The parameters are:

- **Object:** The entity to be converted into a dtype object.
- **Align:** If true, it adds padding to the field to align it with a C-struct.
- **Copy:** If set to true, it creates a new copy of the dtype object. If false, the result is a reference to the built-in data type object.

Example Code:

```
# using array-scalar type
import numpy as np
dt = np.dtype(np.int32)
print(dt)

Output
int32
```

The code defines a NumPy data type (**int32**) using the array-scalar type constructor and prints the data type.