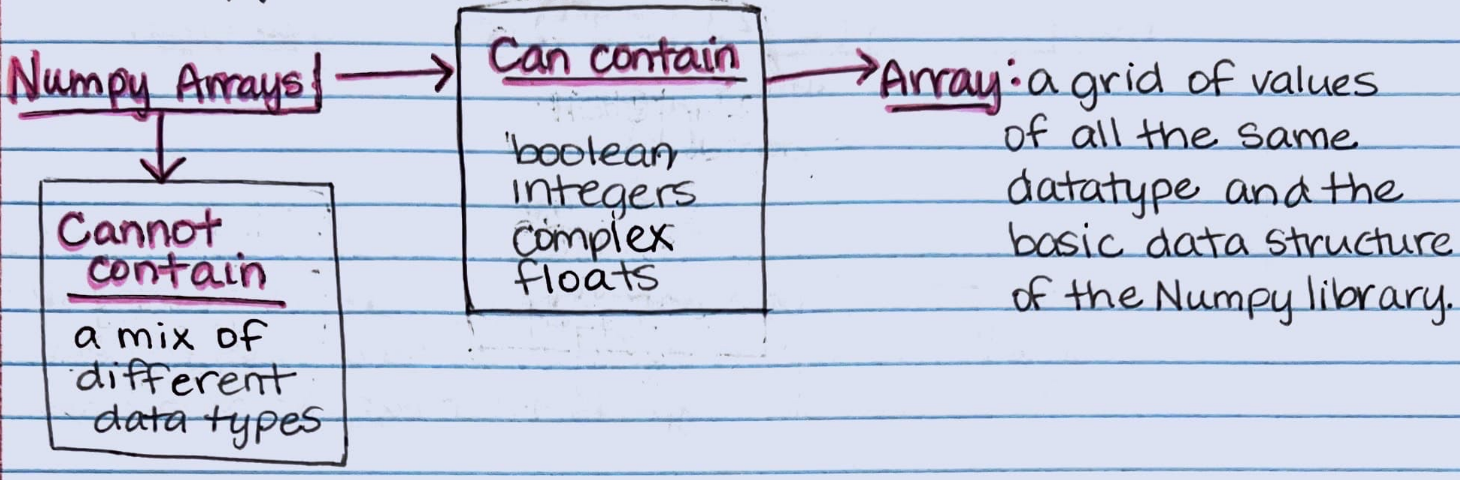
Jessica Meyer

Numpy Cheatsheet

General Purpose A powerful library that can be used to perform a wide variety of calculations on scalers;vectors, multi-dimensional arrays, and matrices more efficiently than basic python.



Numpy Arrays

Can contain

boolean
integers
complex
floats

Cannot contain

a mix of different data types

Array: a grid of values of all the same datatype and the basic data structure of the Numpy library.

Creating numpy Arrays

numpy.arange(integer): creates a 1D-Array with regular intervals as datatype=any

```
numpy.arange(10)
    output:[1,2,3,4,5,6,7,8,9,10]
```

you can also specify datatype and change the intervals

```
numpy.arange(1., 5 ,0.5)
   output:[1.0,1.5,2.0,2.5,3,3.5,4,4.5,5]
```

general 1D-Array creation

numpy.arange (start, stop, step)

numpy.array([1,2,3,4]) : also creates a 1D-Array

numpy.array() can create multi-dimensional arrays aswell

numpy.array ([1,2],[3,4]) : creates a 2D-Array

```
output: array([1,2],
              [3,4])
```

numpy.zeros() or **numpy.ones()** create numpy 1D- or multi-D- Arrays filled with zeros or ones, respectively. The default datatype is float but can be specified to int

numpy.zeros ((rows,columns, dtype=))

1D-Array

```
numpy.zeros((2, ))
    output: array([0,0])
```

→ a single 1D-Array with two zeros

2D-Array

```
numpy.zeros ((3,2))
    output: array([[0,0],
                   [0,0],
                   [0,0]])
```

→ a single 2D-Array with 3 1D-Arrays, each 1D-Array has 2 values/zeros.

```
numpy.ones ((2,3))
    output: array([[1,1,1],
                   [1,1,1]])
```

→ a single 2D-Array with 2 1D-Arrays, each with 3 values/ones

numpy.full((#of1D-Arrays, #ofvalues), fill value) yields an array filled will a user-set value.

3D-Array

```
numpy.full((2,3,4),8)
    output: array([[[8,8,8,8],
                    [8,8,8,8],
                    [8,8,8,8]],
                   [[8,8,8,8],
                    [8,8,8,8],
                    [8,8,8,8]]])
```

→ a single 3D-Array with 2 depths;
each depth has 3 1D-Arrays;
each 1D-Array has 4 values/8's

Array Slicing

To understand arrays, we must understand indices.

→ An array's index begins with 0
- → the first value in a 1D array has an index of 0
- → the third value in a 1D array has an index of 2

Ex 1:

```
array_1d = numpy.arange(4,15,2)
    output: array([4,6,8,10,12,14])
```

```
new_array_1d = array_1d[2:4]
    output: array([8,10,12])
```

→ values start at index 2 (included) and stop at index 4 (discluded),

```
new_array2_1d = array_1d[3:]
    output: array([10,12,14])
```

→ values start at index 3

```
new_array3_1d = array_1d[1:5:2]
    output: array([6,10])
```

→ values start at index 1, stop at index 5, and **skip** → every other value

Ex 2: 2D-Arrays

```
array-2d = numpy.array([[1,3,7,8,12,15],[2,6,6,9,13,14]])
   output: array([[1,3,7,8,12,15],
                  [2,6,6,9,13,14]])
 new-array = array-2d[1, 1:4] → within index 1 1D-Array,
  output: array([6,6,9])          start at index 1 and
           ↳ return is 1D         stop at index 4

new-array-2d = array-2d[0:2, 1:4] → within index 0 and
  output: array([[3,7,8],              index 1, 1D-Arrays,
                 [2,6,6]])             start at index 1 and
                                       stop at index 4
```

Ex 3:

```
row-array = array-2d[1, :]
   output: array([2,6,6,9,13,14])

column-array = array-2d[:, 3]
   output: array([8,9])

range-of_values = (array-2d[:, :] > 7)
   output: array([[8,12,15],
                  [9,13,14]])
```

Helpful Numpy Functions

numpy.linspace (start, stop, sample number)

→ returns evenly spaced numbers over a specified interval

ex: numpy.linspace (3, 19, 6)

output: array([3, 6, 9, 12, 15, 18])

numpy.random.randint(range min, range max, sample number)

→ returns a specified number of random integers with the specified range

ex: numpy.random.randint (5, 10, 7)

output: array([6, 5, 9, 8, 5, 5, 7])

numpy.round (array, number of decimal points to keep)

→ returns rounded integers within the given array

ex: example_array = numpy.array([0.3579, 0.2468, 0.5709])

numpy.round(example_array, decimals=2)

output: array([0.36, 0.25, 0.57])

numpy.union1d (array_1_1d, array_2_1d)

→ combines both arrays into a new array

ex: array_1 = numpy.array([3, 8, 4, 2])

array_2 = numpy.array([12, 9, 5, 16])

numpy.union1d(array_1, array_2)

output: array([3, 8, 4, 2, 12, 9, 5, 16])

numpy.vsplit(2D-Array or multidimensional-Array)

→ splits the data vertically into a specified number of equal parts

ex: array_2d = numpy.array([[1, 3, 5, 7], [2, 4, 6, 8]])

numpy.vsplit(array_2d, 2)

output: array([[1, 3, 5, 7]]), array([[2, 4, 6, 8]])

numpy.vstack((array,array))

→ vertically stack arrays of same dimensions

```
ex:  array_1 =numpy.array([1,2,3,4])
     array_2=numpy.array([5,6,7,8])
     numpy.vstack(array-1,array_2)
        output: array([[1,2,3,4],
                        [5,6,7,8]])
```

numpy.tile(repeating integer/object/string, times to repeat)

→ create an array by repeating integer/object/float a specified number of times.

```
ex:numpy.tile(numpy.arange(1,8,1),2)
      output:array([1,2,3,4,5,6,7,1,2,3,4,5,6,7])
```