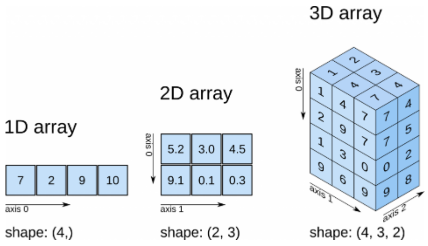
NumPy Cheat Sheet

Image: https://numpy.org

General Purpose of NumPy

- Numerical Python is a library for working with numerical data in Python.
- Contains multidimensional array and matrix data structures and a n-dimensional array
- Uses to perform a variety of mathematical operations on arrays



https://predictivehacks.com/tips-about-numpy-arrays/

All other images from:
https://jakevdp.github.io/PythonDataScienceHandbook

NumPy Arrays

- Central data structure that contains a grid of values and info about the raw data
- Has a grid of elements that can be indexed in several ways:
  - by a tuple of nonnegative integers, by Booleans, by another array, or by integers
- Elements are all of the same type, (i.e., all integers, floats, text strings, etc.).

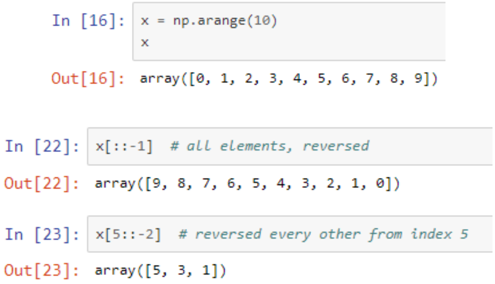
Example 1D Array: avg_monthly_precip = np.array([0.70, 0.75, 1.85])

Example 2D Array: precip_2002_2013 = np.array([[1.07, 0.44, 1.50], [0.27, 1.13, 1.72]])

Slicing NumPy Arrays

- For 1D arrays, you only need to specify **one index value**, which is the position of the element in the NumPy array (e.g. arrayname[index]).
  - *To get* ***third element****, use index value* ***2*** *(Python indexing begins with 0).*
    - *Example:* avg_monthly_precip[2]
  - Use .shape to reveal how many elements a 1D array has in it
    - avg_monthly_precip.shape returns (12,)
  - To select a range, specify using [starting_value, ending_value]
  - [:5] # first five elements
  - [::2] # every other element
  - [1::2] # every other element, starting at index 1
  - [::-1] # all elements, reversed
  - [5::-2] # reversed every other from index 5

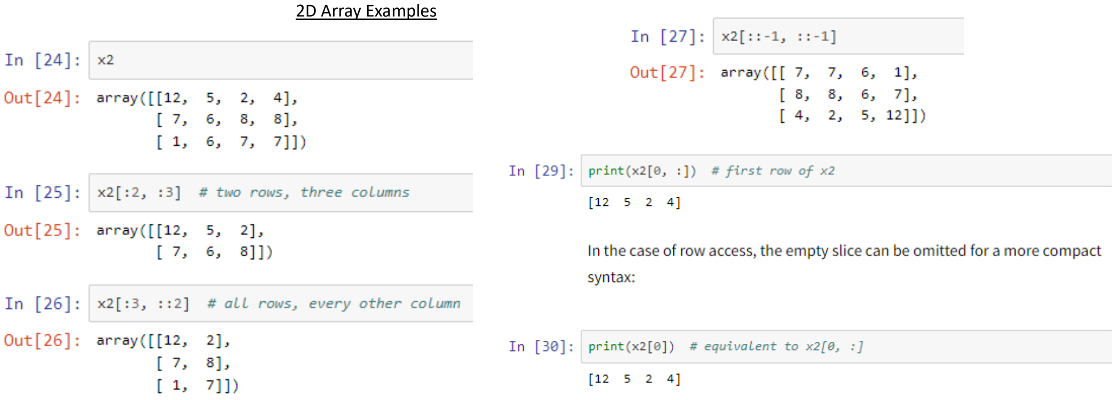
Basic 1D Array Examples



```
In [16]: x = np.arange(10)
         x

Out[16]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [22]: x[::-1]  # all elements, reversed

Out[22]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

In [23]: x[5::-2]  # reversed every other from index 5

Out[23]: array([5, 3, 1])
```

- For 2D arrays, you need to specify both a **row index** and a **column index**
  - Rows are first columns are second! [row index, column index]
  - To select the element in the second row, third column, you can use: [1, 2]
  - To select a range, specify using [start_row_index:end_row_index, start_column_index:end_column_index]
    - To select the elements in first row, first two columns: array[0:1, 0:2]
  - To select all rows of a column (entire column), use a colon for the row index: array[:,0]
  - To select all column values of a row (entire row) use a colon for the column index: array[0,:]

2D Array Examples



```
In [24]: x2

Out[24]: array([[12,  5,  2,  4],
                [ 7,  6,  8,  8],
                [ 1,  6,  7,  7]])

In [25]: x2[:2, :3]  # two rows, three columns

Out[25]: array([[12,  5,  2],
                [ 7,  6,  8]])

In [26]: x2[:3, ::2]  # all rows, every other column

Out[26]: array([[12,  2],
                [ 7,  8],
                [ 1,  7]])
```

```
In [27]: x2[::-1, ::-1]

Out[27]: array([[ 7,  7,  6,  1],
                [ 8,  8,  6,  7],
                [ 4,  2,  5, 12]])

In [29]: print(x2[0, :])  # first row of x2

         [12  5  2  4]
```

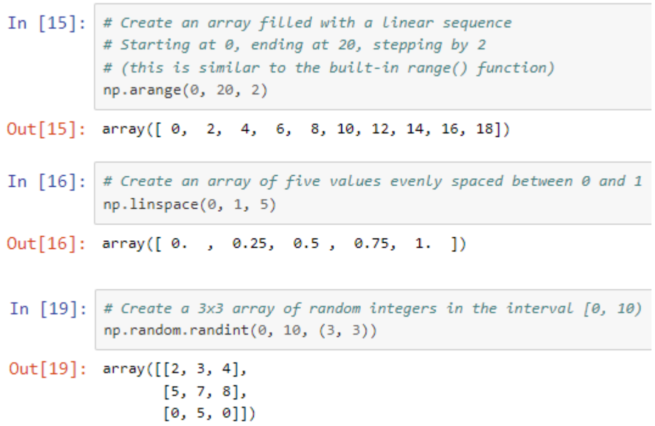
In the case of row access, the empty slice can be omitted for a more compact syntax:

```
In [30]: print(x2[0])  # equivalent to x2[0, :]

         [12  5  2  4]
```

Image: https://numpy.org

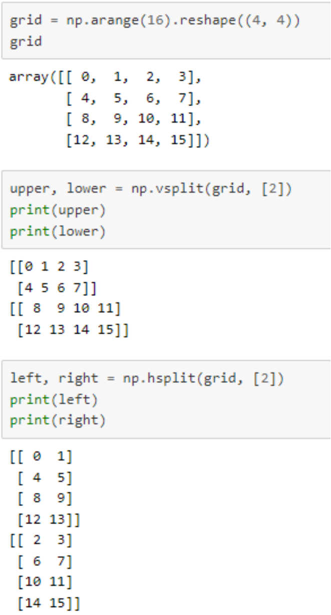
Cheat Sheet

- **Creating NumPy Arrays**
  - Enter data directly for a 1D: np.array([1,2,3])
  - Enter data directly for a 2D: np.array([[1,2,3], [4,5,6]])
  - Make a set of random numbers: np.random.randint(10, size = (3,4)) #2d array 3 rows for columns, integers 0-10
  - Make a set of zeros or ones: np.zeros((dim1,dim2)) or np.ones((dim1,dim2,…))
  - Make a set of sequential values: np.arange(start, stop) (note: this is a 1-d array but you can use .reshape to change dimensions)
  - # Create a length-10 integer array filled with zeros: np.zeros(10, dtype=int)
  - # Create a 3x5 floating-point array filled with ones: np.ones((3, 5), dtype=float)



```
In [15]: # Create an array filled with a linear sequence
         # Starting at 0, ending at 20, stepping by 2
         # (this is similar to the built-in range() function)
         np.arange(0, 20, 2)

Out[15]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])

In [16]: # Create an array of five values evenly spaced between 0 and 1
         np.linspace(0, 1, 5)

Out[16]: array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])

In [19]: # Create a 3x3 array of random integers in the interval [0, 10)
         np.random.randint(0, 10, (3, 3))

Out[19]: array([[2, 3, 4],
                [5, 7, 8],
                [0, 5, 0]])
```

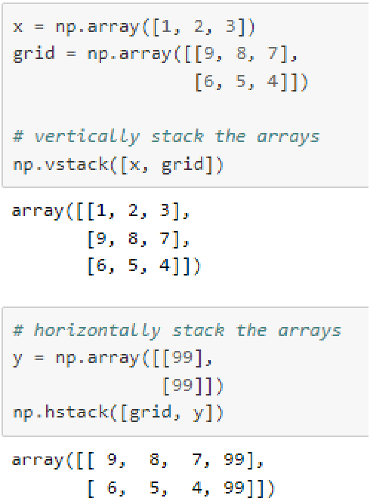
Splitting Examples



```
grid = np.arange(16).reshape((4, 4))
grid

array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])

upper, lower = np.vsplit(grid, [2])
print(upper)
print(lower)

[[0 1 2 3]
 [4 5 6 7]]
[[ 8  9 10 11]
 [12 13 14 15]]

left, right = np.hsplit(grid, [2])
print(left)
print(right)

[[ 0  1]
 [ 4  5]
 [ 8  9]
 [12 13]]
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
```

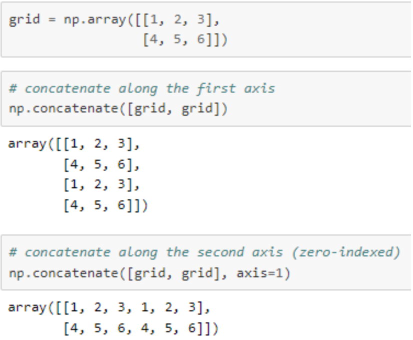
- **Helpful NumPy Functions**
  - Np.copy(array) → copies your array to a new
  - Statistics: np.mean, min, max, median
    - most np functions have an axis argument if you want to summarize on just one axis
      - axis=0 summary of each column across all rows
      - axis= 1 summary of each row across all columns
  - Combining arrays: concatenate, vstack, hstack:
    - np.concatenate((array1,array2),axis=0) → adds array2 as rows to end of array1
    - np.concatenate((array1,array2),axis=1) → adds array2 as columns to end of array1
  - Dividing arrays:
    - np.split(array,3) → splits array into 3 sub-arrays
    - np.hsplit(array,5) → splits array horizontally on the 5th index
  - Trig Functions:
    - Sin, Cos, Tan, Arcsin, Arccos, Arctan…
    - Logs and Exps…

Stacking Examples



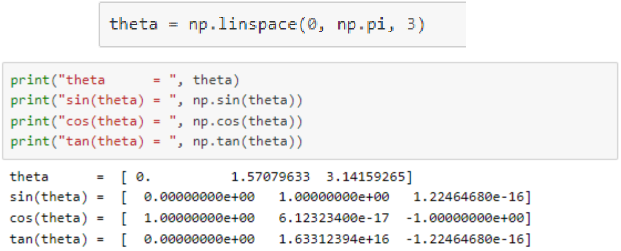
```
x = np.array([1, 2, 3])
grid = np.array([[9, 8, 7],
                 [6, 5, 4]])

# vertically stack the arrays
np.vstack([x, grid])

array([[1, 2, 3],
       [9, 8, 7],
       [6, 5, 4]])

# horizontally stack the arrays
y = np.array([[99],
              [99]])
np.hstack([grid, y])

array([[ 9,  8,  7, 99],
       [ 6,  5,  4, 99]])
```

Concatenate Examples

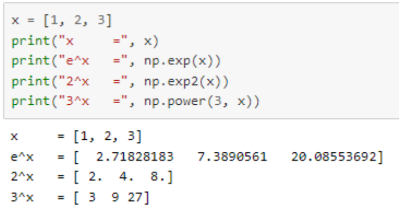


```
grid = np.array([[1, 2, 3],
                 [4, 5, 6]])

# concatenate along the first axis
np.concatenate([grid, grid])

array([[1, 2, 3],
       [4, 5, 6],
       [1, 2, 3],
       [4, 5, 6]])

# concatenate along the second axis (zero-indexed)
np.concatenate([grid, grid], axis=1)

array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

Trig Functions



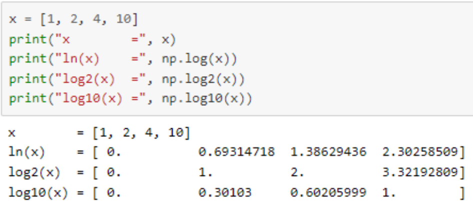
```
theta = np.linspace(0, np.pi, 3)

print("theta      = ", theta)
print("sin(theta) = ", np.sin(theta))
print("cos(theta) = ", np.cos(theta))
print("tan(theta) = ", np.tan(theta))

theta      =  [ 0.          1.57079633  3.14159265]
sin(theta) =  [  0.00000000e+00   1.00000000e+00   1.22464680e-16]
cos(theta) =  [  1.00000000e+00   6.12323400e-17  -1.00000000e+00]
tan(theta) =  [  0.00000000e+00   1.63312394e+16  -1.22464680e-16]
```



```
x = [1, 2, 3]
print("x     =", x)
print("e^x   =", np.exp(x))
print("2^x   =", np.exp2(x))
print("3^x   =", np.power(3, x))

x     = [1, 2, 3]
e^x   = [  2.71828183   7.3890561   20.08553692]
2^x   = [ 2.  4.  8.]
3^x   = [ 3  9 27]
```

Logarithms

Note: np.log gives natural log; use log2 for base 2, log10 for base10, etc.



```
x = [1, 2, 4, 10]
print("x        =", x)
print("ln(x)    =", np.log(x))
print("log2(x)  =", np.log2(x))
print("log10(x) =", np.log10(x))

x        = [1, 2, 4, 10]
ln(x)    = [ 0.          0.69314718  1.38629436  2.30258509]
log2(x)  = [ 0.          1.          2.          3.32192809]
log10(x) = [ 0.          0.30103     0.60205999  1.        ]
```