VARIABLES AND SPYDER WORKSPACE

• Spyder is a Python IDE that’s a part of the Anaconda distribution.

• Spyder has a Python console – useful to run commands quickly and variables can be seen in the Variable Explorer. Similar to MATLAB’s command window.

• a = 3 - defines a variable. No need to specify variable type. [Documentation.]

• print(type(a)) # Prints "<class 'int'>"  

• print(a + 1)   # Addition; prints "4"

• print(a ** 2)  # Exponentiation; prints "9". ** Represents exponentiation, not ^.

• print(a *= 2)  # Prints "6"

• Comments start with a %. In Spyder, use #%% to define a region (Each IDE/text editor has its own command). Multiline comments are between a pair of ""."
• Python implements all the usual operators for Boolean logic, but uses English words rather than symbols (&&, ||, etc.)

• t = True

• f = False

• print(type(t)) # Prints "<class 'bool'>"

• print(t and f) # Logical AND; prints "False"

• print(t or f) # Logical OR; prints "True"

• print(not t) # Logical NOT; prints "False"

• print(t != f) # Logical XOR; prints "True"
LISTS

• Python has many different data structures like lists, dictionaries, sets and tuples. In this tutorial we’ll take a look at just lists. Documentation. More on lists.

• Note: Unlike MATLAB, Python indexing starts at 0.

• $xs = [3, 1, 2]$  # Create a list

• print(xs, xs[2])  # Prints "[3, 1, 2] 2"

• $xs[2] = 'foo'$  # Lists can contain elements of different types

• print(xs)  # Prints "[3, 1, 'foo']"

• $xs.append('bar')$  # Add a new element to the end of the list

• print(xs)  # Prints "[3, 1, 'foo', 'bar']"

• $x = xs.pop()$  # Remove and return the last element of the list

• print(x, xs)  # Prints "bar [3, 1, 'foo']"
SLICING IN LISTS

• Slicing in lists is pretty useful in Python. Here’s a brief introduction. We’ll mostly focus on slicing using NumPy.

• `nums = list(range(5))`  # range is a built-in function that creates a list of integers

• `print(nums)`  # Prints "[0, 1, 2, 3, 4]"

• `print(nums[2:4])`  # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"

• `print(nums[2:])`  # Get a slice from index 2 to the end; prints "[2, 3, 4]"

• `nums[2:4] = [8, 9]`  # Assign a new sublist to a slice

• `print(nums)`  # Prints "[0, 1, 8, 9, 4]"
Python functions are defined using the def keyword. [Conditional statements documentation, Functions documentation.]

def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
# Prints "negative", "zero", "positive"

**NOTE:** Indentation in Python is used to determine the grouping of statements. e.g.: Loops, If-Else, Functions. Use TABS
A NumPy array is a grid of values, all of the same type. The shape of an array is a tuple of integers giving the size of the array along each dimension.

**Array Creation**

```python
import numpy as np

a = np.array([1, 2, 3])  # Create a rank 1 array
print(a.shape)            # Prints "(3,)". Indicates 3 elements along a dimension.
print(a[0], a[1], a[2])   # Prints "1 2 3"

b = np.array([[1, 2, 3], [4, 5, 6]])  # Create a rank 2 array
print(b.shape)            # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```
OTHER METHODS TO CREATE ARRAYS

```python
import numpy as np

a = np.zeros((2,2))  # Create an array of all zeros
b = np.ones((1,2))  # Create an array of all ones

c = np.full((2,2), 7)  # Create a 2x2 array where all elements are equal to 7.
d = np.eye(2)  # Create a 2x2 identity matrix

e = np.random.random((2,2))  # Create an array filled with random values
```

Documentation.
NUMPY DATATYPES

Every NumPy array is a grid of elements of the same type. NumPy provides a large set of numeric datatypes that you can use to construct arrays. NumPy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype. Here is an example:

```python
import numpy as np

x = np.array([1.0, 2.0])  # Let numpy choose the datatype
print(x.dtype)            # Prints "float64"

x = np.array([1, 2], dtype=np.int64)  # Force a particular datatype
print(x.dtype)            # Prints "int64"
```
IMPORTANT FUNCTIONS

• `np.sin`, `np.cos`, `np.tan`, `np.radians`, `np.angles` – Elementwise
• `np.round`, `np.ceil`, `np.floor` – Elementwise
• `np.cumsum`, `np.log`, `np.exp` – Elementwise
• `np.max`, `np.min`, `np.interp` – Elementwise
• `np.linalg.norm`, `np.linalg.det`, `np.trace`, `np.linalg.inv`, `np.linalg.pinv`
• `np.matlib.repmat`, `np.lib.pad`
• `np.sort`, `np.argsort`, `np.count_nonzero`
• `np.copyto`, `np.reshape`, `np.ravel`, `np.asarray`
Numpy offers several ways to index into arrays.

Slicing: Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array.

Boolean array indexing: np.where() returns a Boolean array.

CODE.

Documentation.
ROW MAJOR AND COLUMN MAJOR

• By default, NumPy is Row Major.
• It can, however, be forced to operate in Column Major.
• CODE.
Numpy Operations - Math

• Elementwise Sum.
• Elementwise difference.
• Elementwise product.
• Elementwise division.
• Elementwise square root.

Unlike MATLAB, * is an elementwise multiplication, not multiplication. Use np.dot() to multiply matrices or to computer inner product of two vectors.

Documentation.
SAVE AND LOAD — PKL, NPY, MAT

• Pickle
• NumPy
• MAT Files
• CODE.
PLOTTING AND DISPLAYING IMAGES

• 2D Plot with 1 function
• 2D Plot with multiple functions
• Subplots
• Displaying images
• CODE.
DEBUGGING

• Breakpoints
• Pdb – Useful for text editors like Sublime, notepad++.
• CODE.

PDB Tutorial.
MORE RESOURCES

• Parts of this tutorial have been taken from the CS231n Python NumPy Tutorial: http://cs231n.github.io/python-numpy-tutorial/

• Learn Python Online: https://www.learnpython.org/

• Learn NumPy Online: https://www.tutorialspoint.com/numpy/ (Also Python)

• Python 2.7 documentation: https://docs.python.org/2/index.html

• NumPy and SciPy documentation: https://docs.scipy.org/doc/