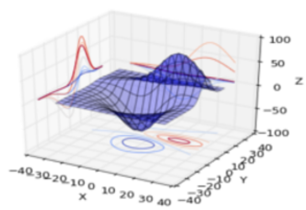
Getting Started with Python in Engineering

Neal Gordon

Dec 20, 2016



0.1 Summary

- Why python
- Installation
- Help and Documentation
- Python Syntax
  - Comments and Strings
  - Printing
  - Variables and Datatypes
  - Lists and Arrays
  - Indentation and Conditionals
  - Loops
  - Functions
  - Modules
- Numerical Python
  - Arrays

  - Logical Indexing
  - Multi-Dimensional Arrays (matrices)
  - Plotting

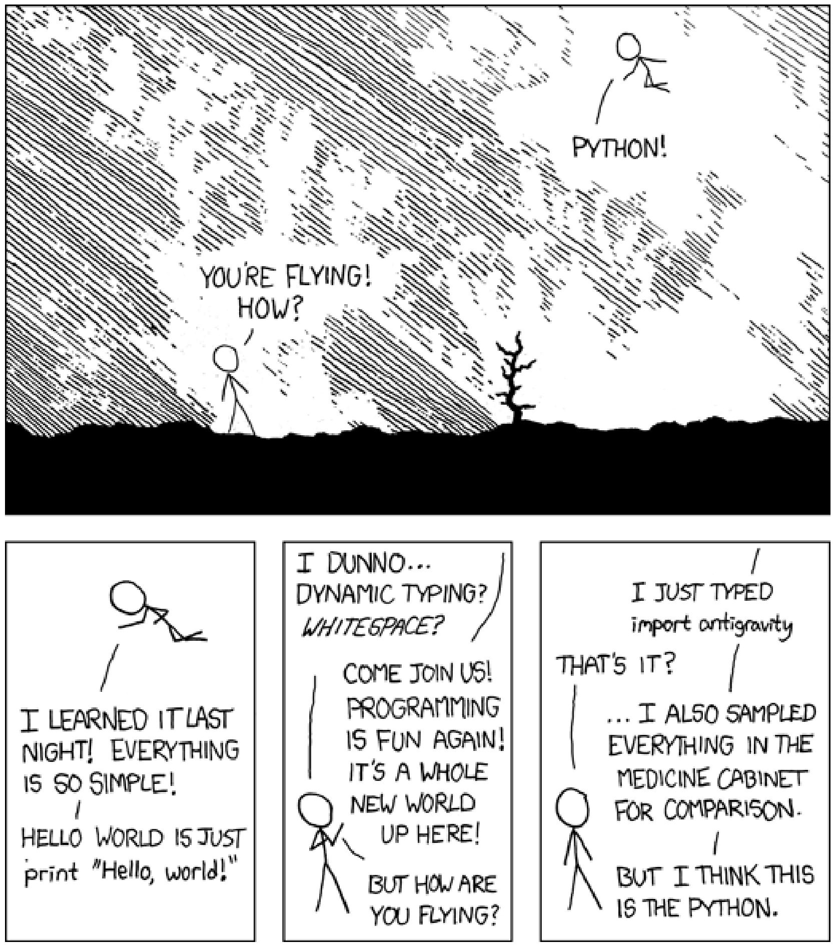
- References and Links

0.2 Why code

It's just another tool to solve engineering problems

- Hand calculations are by far the best way to start solving a problem, but not always the best way to find the solution you need.
- Spreadsheets are a great mix of a database and calculator, but not great at either
- Databases are the best way to store data, but typically are interfaced with code
- The best calculators are the ones that work out of the box, but are also 100% customizable
- diversify skills other than just basic data entry or calculations, such as web design and development, software development, CAD/CAE API development
- Automate the boring parts of your job with code so you can enjoy the awesome parts

0.3 Why python



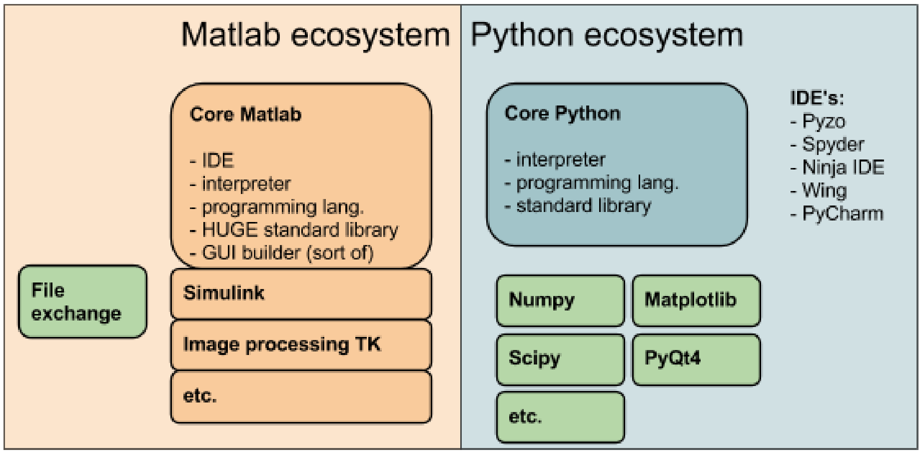
0.4 Why python

Python is a general purpose programming language, and is used in web development, graphical user interfaces, IT automation, and SCIENTIFIC COMPUTING

0.5 Python vs Matlab

Matlab is a powerful scientific computing language and software development package that is superior for many mathematical operations, specifically linear algebra, signal processing, and modeling dynamic systems(via Simulink). However, for many engineering applications is not worth the cost. Although many of the Matlab libraries may be arguably better than comparable python libraries, they are closed source and cannot be modified or inspected by users. Also, sharing code with collaborators or publications requires other users to have licenses.

Python, on the other hand, is a general programming language so it is very good at many tasks, but was not designed exclusively for scientific computing



0.6 Python vs Matlab

Python Positives

- being easy and fun to learn with great online forumns and open communities
- many libraries for scientific computing, server and desktop management and automation
- works great on Linux, Mac and Windows and easily integrates with other languages (fortran, C, java)
- scripts are easily integrated with other applications
- FREE, open-source and customizable. Every module/library can be modified

Matlab Positives

- professional technical support
- fast(er) execution
- being straightforward for simple scientific calculations

0.7 Installation

To get started, choose the python distribution you want. I would recommend anaconda, which has most of the scientific packages that are needed in one installation, but more importantly, comes with a package manager called `conda`, which is a big help when installing and updating python packages (especially on Windows) and managing environments if we want to use multiple versions of python.

- Anaconda or Miniconda for Linux, Mac, and Windows
- pyzo For Linux, Mac, and Windows
- winpython for Windows
- python(x,y) for Windows

Installing new packages and updating is easy. `conda` takes care of dependencies and version compatibility.

install a single library conda install numpy

update a single library conda update numpy

update all packages conda update –all

0.8 A Note on Versions

Python currently has 2 popular versions, 2x and 3x. 2x has a lot of legacy code still in use, but support is being phased out. Either version works, but I would recommend always trying to default to the latest release (v3.5).

Among many major changes, the two changes I find most troublesome is the print command and division.

print changes [fontsize=,linenos=false,mathescape,baselinestretch=1.0,fontfamily=tt,xleftmargin=7mm]pyt
v2.7 print "hello world" hello world

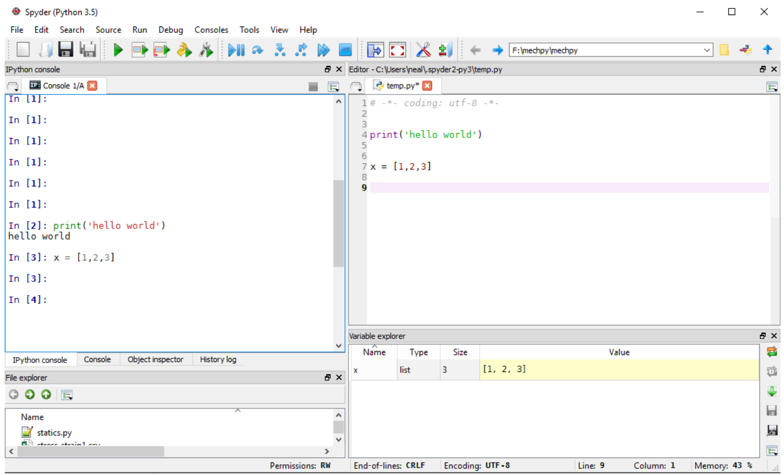
v3.5 print("hello world") hello world

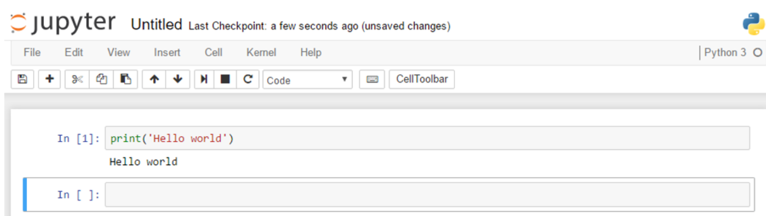
division changes v2.7 7 / 4 1 7 / 4.0 1.75 7 // 4 1

v3.5 7 / 4 1.75 7 // 4 1

0.9 Development Environments

Python has many options for programming, and graphical user interface development. Many different integrated development environments (IDE), such as spyder. Although not a polished, spyder is similar to the Matlab interface. Browser based jupyter notebooks are a great way to write web-ready content or sharing calculations or plots





0.10 First Program

Before we go any further, let's put together a simple python script and run it. First, open a text file and type

[fontsize=,linenos=false,mathescape,baselinestretch=1.0,fontfamily=tt,xleftmargin=7mm]python print('hello world')

save it as `hello.py`. Now, lets open a command window. In windows, my preferred way is to shift+right-click in a window and select **open in terminal**. The other option is to run `cmd` from the start menu. Type the following and hit enter.

```
python hello.py
```

```
hello world
```

You should have seen your print statement printed to the console. If so, congratulations!, you have executed your first python program. Some other ways to run your program are through the ipython interpreter, which can be launched from your command line by running `ipython` or using the ipython install shortcut

In [1]: run hello.py

```
hello world
```

0.11 Help and Documentation

if help is ever needed, for example of the math module, in the ipython console, type

help(print) or print?

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

For anything else, there is a handy company on the internet that is good at finding stuff.

0.12 Comments and Strings

Making notes or comments in code can be more important than the code itself. If the program is so obscure, hard to read, or confusing, it will not be enjoyable to use, or if it breaks, it will be terrible to repair. Comments should contain info about the command or variable and can be on the same row as a command or be on a new line. Also, it is good practice to include a short description at the beginning of your functions called a **docstring**. Using these can make documentation much easier. Oneline comments are simply prefixed with `#`, and strings can either be enclosed with *'single'* or *"double"* quotes.

this is a comment, it is not executed every new line of a comment must have a

”’this is a single line docstring with single quotes”’

"""this is a single line docstring with double quotes"""

”’this is a multi-line string or comment enclosed with 3 single quotes. This is an easy way to create a multi-line comment, or a docstring at the beginning of your functions”’

0.13 String Pitfalls

One troublesome feature of using a cross-platform language like python is dealing with the difference between operating systems. One frustrating difference is the filepath separators. Linux/MAC use the forwardslash `/`,and windows uses the backslash‘

‘. In python, the backslash has a different function, it is an escape character, so the windows file paths need `C:\mydir` must be written as `C:\\mydir` or as a forward-slash `C:/mydir` in python print('newline escape example with another linebreak')

```
newline escape
example with
another
```

print('C:́)

```
SyntaxError: EOL while scanning string literal
```

One solution to this error is to use double quotes \\, in Windows double slash escapes the first print('C:
myname
mydirectory
')

```
C:\\myname\\mydirectory\\
```

0.14 Printing

The simplest way to print a statement with some variables is to simple create groups of strings separated by commas and inserting variables in the middle. This is easy, but controlling output is difficult.

single line string print('this is a string') single line string with a line break print('this is a string')

```
this is a string
```

the simplest way to print x = 1/3 print(x,' spam and ',3,' eggs')

```
0.3333333333333333  spam and  3  eggs
```

or force integers and floats to be a string and concatenate print(str(x)+' spam and '+str(3)+' eggs')

```
0.3333333333333333  spam and  3  eggs
```

[fontsize=,linenos=false,mathescape,baselinestretch=1.0,fontfamily=tt,xleftmargin=7mm]python The legacy printing variables syntax follows a percent sign and the datatype print('

```
5 spam and 3.000000 eggs
```

0.15 Advanced Printing

The preferred method for printing is using the format dot operator. This format allows full control on how the information is displayed. The current syntax for printing variables uses the `.format` dot operator

[fontsize=,linenos=false,mathescape,baselinestretch=1.0,fontfamily=tt,xleftmargin=7mm]python basic print with 2 variables print(' spam and '.format(5,3,'eggs'))

```
5 spam and 3 eggs
```

[fontsize=,linenos=false,mathescape,baselinestretch=1.0,fontfamily=tt,xleftmargin=7mm]python designating float data type print(':f cans of '.format(1/3, 'spam'))

```
0.333333 cans of spam
```

[fontsize=,linenos=false,mathescape,baselinestretch=1.0,fontfamily=tt,xleftmargin=7mm]python scientific notation with 2 digits print('::.2e cans of '.format(12345689, 'spam'))

```
1.23e+07 cans of spam
```

[fontsize=,linenos=false,mathescape,baselinestretch=1.0,fontfamily=tt,xleftmargin=7mm]python 2 digits and pad 6 places with zeros print(':06.2f cans of '.format(3.141592653589793, 'spam'))

```
003.14 cans of spam
```

0.16 Variables and Datatypes

Everything in python is an object, meaning a variable can be a simple text string to a huge nxm array of complex numbers and they all have properties specific to them. Python has dynamic typing, or duck typing ("walks like a duck, sounds like a duck, must be a duck"). It takes a guess at what kind of data you are trying to store

x=1 type(x)

```
int
```

y=1.5 type(y)

```
float
```

y='spam' type(y)

```
string
```

z = [1,2,3,4] type(z)

```
list
```

0.17 Lists and Arrays

Lists are a standard, python datatype for storing and processing strings and numbers. A list can contain different datatypes. Arrays are not a standard python datatype, but can be imported from the numerical python library, numpy. Arrays can only hold one type of data, but are optimized for numerics(see section 0.28 for demo).

define a list with int and string datatype z = [1,2,3,4, 'five'] prints the list z

```
[1,2,3,4, 'five']
```

Parts of a list can be sliced out of a list using the indices return the first element of the list z[0]

```
1
```

return the last element of the list z[-1]

```
'five'
```

return the 3rd and 4th element of the list. z[2:5]

```
[3,4,'five']
```

0.18 More Lists

Since lists are so useful, lets do a few more examples.

mylist = [[1],[2,3],[4,5,6]] extract second element of the list mylist[1]

```
[2,3]
```

add on to the end of a list mylist.append([0,1,2]) print list mylist

```
[[1], [2, 3], [4, 5, 6], [0, 1, 2]]
```

delete element two del mylist[1] print(mylist)

```
[[1], [4, 5, 6], [0, 1, 2]]
```

returns the len of the list len(mylist)

```
3
```

0.19 List and Array Pitfalls

Reassigning lists can be dangerous business with unexpected results. If you are used to Matlab syntax, the following example is a reasonable way to create a copy of list x named y

create list x x = [1,2,3] create another list y of the same values as x y = x reassigned the 2 list location with 7 x[1] = 7 print('x =',x) print('y =',y)

```
x = [1,7,3]
y = [1,7,3]
```

Notice anything strange? Changing x also changes y! If that is what you expected, then you are probably a computer scientist or something. Making this mistake can lead to weird behavior, so be careful.

The correct way to create variable y x = [1,2,3] y = x.copy() x[1] = 7 print('x =',x) print('y =',y)

```
x = [1,7,3]
y = [1,2,3]
```

That makes more sense, although it's a bit cumbersome. Oh well :(

0.20 Indentation and Conditionals

Unlike many other languages that use brackets to enclose commands, python uses the whitespace to control the program flow. For example, if statements, loops, and functions are indented 4 spaces to indicate what code is in the function. Once the indent is removed, the code is no longer in the function. This forces a clean, readable coding style. For boolean **is** or **==**, or binary **0** , **1** or logical **True** or **False** .

typical if-else. Can remove the else for just an if statement x = True if x: print('x is true') else: print('x is not true')

```
x is true
```

typical if-elif-else x = 'maybe' if x: print('x is true') elif x=='maybe': print('x is maybe true') else: print('x is not true')

```
x is true
```

Python also has clean syntax for one-liners x = 0 s = 'x is true' if x is True else 'x is not true' print(s)

```
x is not true
```

0.21 Loops

Loops in python are a great way to iterate through a list, perform element-wise calculations, or for sorting data. Here are a few examples of **for** and **while** loops. In python, it is preferred to loop through a list as compared to generating an index array, but both work fine.

typical for loop x = [] for i in range(10): x.append(i) print('x =',x)

```
x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Notice the print statement in the previous code snippet. It is unindented from the for loop, indicating it is executed after the for-loop is complete. Below, is another example of a one-liner for loop, which is called a **list comprehension** x = [y for y in range(10)] x

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

A while loops requires a little more work, but is handy when the number of loops is unknown.

i = 0 ; x = [] while i < 10: x.append(i) i=i+1 i+=1 print(x)

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

0.22 Functions

The bread-and-butter of any language are creating custom user-defined functions. This compartmentalizes code to reusable chunks. Simple functions can be defined as a one-liner *lambda* function. This example creates a function *myfunc* with two inputs, *x* and *y* which looks like *myfunc(x,y)*

[fontsize=,linenos=false,mathescape,baselinestretch=1.0,fontfamily=tt,xleftmargin=7mm]python lambda functions are one-liner function definitions myfunc = lambda x,y: x**2 / y

For more complicated functions with docstrings, the following example performs the same calculation

another way to define a function with the docstring def myfunc(x,y): "'neat two variable function to do some math"' z = x**2 / y return z

Here is an example of calling the function

a = myfunc(5,3) print(a)

```
8.333333333333334
```

0.23 Using Modules

Since python is a generic programming language, it does not have math operations built in, they must be imported from modules. Modules are simply a ***.py** file that has lots of functions that can be used in any other script if it is imported. Imports are typically done at the beginning of a script, but can be done at anytime, as a long as it is before you call the function. The concept of namespace and reserved names is very important when naming variables and importing modules. Ensure you do not name variables or modules the same as any reserved names, such as `list`, `type`, `print`.

Least confusing way to import modules import numpy numpy.sin(numpy.pi / 45)

```
0.069756473744125302
```

You can also import the entire module but as an alias, in this case **np**. This is the preferred option because all the numpy functions are imported and available, but will not overwrite other functions with the same name.

Less confusing and convientient way to import a whole module import numpy as np np.sin(np.pi/45)

```
0.069756473744125302
```

Individual functions can be imported which is safe, but can be laborious if many functions are required safe but cumbersome way to import modules from numpy import sin, pi sin(pi/45)

```
0.069756473744125302
```

The easiest but riskiest method of module import is importing all functions directly into the main namespace. For complex programs, this can be dangerous unless you know every single module name in the numpy library, because you can overwrite other functions that you may have imported previous.

most convientient but most dangerous way to import modules from numpy import * sin(pi/45)

```
0.069756473744125302
```

0.24 Creating Modules

For many calculations, importing existing modules like `numpy` or `scipy` is sufficient, but you may eventually need to define your own, custom module. Let's make a custom module. Open a text editor and enter the function we made earlier. Save it as `mymodule.py`

another way to define a function with the docstring def myfunc(x,y): "'neat two variable function to do some math"' z = x**2 / y return z

to execute function, call it myfunc(5,3)

```
8.333333333333334
```

Save the file named as `mymodule.py`. Ensuring your console is in the same directory as your file, (`pwd` to check), lets first execute the function then import and use as a custom module.

The function is called in the script so we should see output when we run the file.

```
8.3333333
```

To import from a module and use the function, use the following code.
from mymodule import myfunc myfunc(1,2)

```
8.3333333
```

0.25 Basic Python Example

Let's wrap up our introductory tutorial by doing using a few of the examples together. All of these functions perform the same task, but use slightly different python syntax. We want all the numbers divisible by 5 up to our input y. The example shows how to do this using a for loop, while loop, list comprehension, and a numpy array. This example also shows that there are many ways to solve the same problem.

from numpy import array

def divby$5_1(y = 100) :''' forloopwithifconditionalandmodulooperator, x = []forkinrange(1, y) : ifkx.append(k)returnx$

def divby$5_2(y = 100) :''' whileloopwithifconditional, andmodulooperator, x = []; z = 1whilez <= y : ifkx.append(k)z = z + 1z+ = 1returnz$

def divby$5_3(y = 100) :''' pythonlistcomprehensionandmodulooperator, x = [kforkinrange(1, y)ifkreturnx$

def divby$5_4(y = 100) :''' numpylogicalindexingwithmodulooperator, x = array(range(1, y))i = xreturnx[i]$

Since we provided a default value in the function with `y=100`, we can call the function without an input like so

divby$5_4()$

```
array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95])
```

0.26 Ipython Handy Hints

When using the command line, there are a few commands that are very handy. The first is the `cd` or change directory command is used to navigate directories.

change directory to mydir cd mydir go up one directory towards the root cd ..

Another handy command is `pwd`, or print working directory which is self-explanatory

```
pwd
```

```
C:/user/myproject
```

use `whos` to see variables in current console

```
Variable   Type                          Data/Info
----------------------------------------------------
a          float                         8.333333333333334
array      builtin_function_or_method    <built-in function array>
divby5_1   function                      <function divby5_1 at 0x7f2ab8e10a60>
divby5_2   function                      <function divby5_2 at 0x7f2ab8e10c80>
divby5_3   function                      <function divby5_3 at 0x7f2ab80639d8>
divby5_4   function                      <function divby5_4 at 0x7f2ab8042378>
i          int                           10
myfunc     function                      <function myfunc at 0x7f2ab8e7d598>
mylist     list                          n=4
x          list                          n=10
y          list                          n=3
z          list                          n=5
```

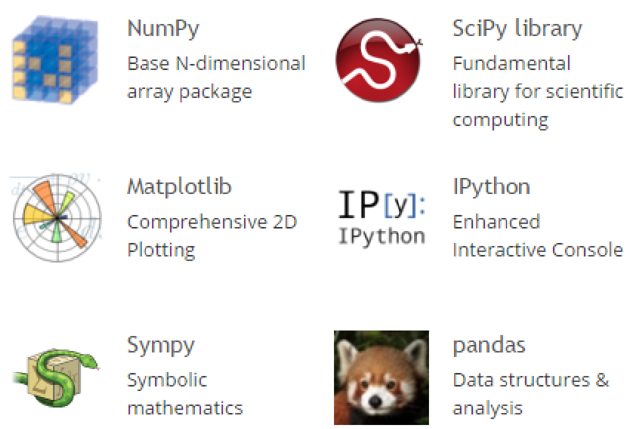
If there are undesired variables, use the general `reset` command or specify varaible `del x` to clear variables in the current namespace. It is also helpful to try and keep scripts in functions to prevent your namespace from becoming cluttered

removes a single variable from namespace del x

clears all variables from current namespace

0.27 Numerical Python

Although python was not originally intended to be a numerical language, it's helpful community, simple syntax, and free, open-source codebase lend it to being a great academic and applied language for theoretical math to physics and engineering calculations.



0.28 Arrays

Numpy arrays are similar to lists, but can only contain one datatype, but capable of N-dimensional arrays(or matrices) . Arrays and are optimized for numerics and linear algebra. Arrays are around 30x faster than lists.

Note - numpy does have a matrix class which was designed for linear algebra, but it is recommended to use the array class to avoid confusion when performing calculations.

make sure to import the functions you need from numpy import array, arange, linspace, sin If you need more functions, import the whole module import numpy as np

using the range function to create a numpy array x = array(range(10)) print('x = ',x, 'as a',x.dtype)

```
x =  [0 1 2 3 4 5 6 7 8 9] as a int64
```

using numpy function arange x = arange(10) print('x = ',x, 'as a',x.dtype)

```
x =  [0 1 2 3 4 5 6 7 8 9] as a int64
```

numpy array x = linspace(0,9,10) print('x = ',x, 'as a',x.dtype)

```
x =  [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.] as a float64
```

0.29 Array Operations

Element-wise operations element-wise array multiplcation x*3

```
array([  0.,   3.,   6.,   9.,  12.,  15.,  18.,  21.,  24.,  27.])
```

element-wise array exponentiation x**2

```
array([  0.,   1.,   4.,   9.,  16.,  25.,  36.,  49.,  64.,  81.])
```

sin(x)

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,    -0.95892427, -0.2794155
```

0.30 Array Operations

Often, objects in python have many methods that can be performed on that data-type. To see what methods are available, a `.` can be typed to see what is available or the `dir()` function can be used

show all methods available for variable/object x print([k for k in dir(x) if k[:2] != '_'])

```
['T', 'all', 'any', 'argmax', 'argmin', 'argpartition', 'argsort',\ 'astype', 'base', 'byteswap',
```

using method sum for variable x x.sum() also use the sum function from the numpy module np.sum(x)

```
45
```

0.31 Logical Indexing

A powerful filtering technique in Matlab and python called logical indexing is a great way to perform calculations to specific numbers or filter data

logical indexing 3 < x

```
array([False, False, False, False,  True,  True,  True,  True,  True,  True], dtype=bool)
```

negate logical index (3 < x)

```
array([ True,  True,  True,  True, False, False, False, False, False, False], dtype=bool)
```

and logical index operator (3 < x) (x < 5)

```
array([False, False, False, False,  True, False, False, False, False, False], dtype=bool)
```

or logical index operator (3 < x) | (x < 5)

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,  True], dtype=bool)
```

multiple and logical index operator (3 < x) (x < 5)

```
array([False, False, False, False,  True, False, False, False, False, False], dtype=bool)
```

0.32 Multi-Dimensional Arrays (matrices)

Numpy has both array classes and a matrix class. The array is a more general object, where the matrix class is specifically for linear algebra. Matrices are only 2-dimensional, which can limit functionality, where the array can be n-dimensional. All matrix operations can be performed on an array, so it is recommended to just use arrays to avoid confusion.

First, to define a 1x3 array c = array([[1,2,3]]) c

```
array([[1, 2, 3]])
```

Now the transpose does what we expected to a 3x1 array c.transpose()

```
array([[1],
       [2],
       [3]])
```

If we want to define a 3x1 array, we define the array like so c = array([[1],[2],[3]])

```
array([[1],
       [2],
       [3]])
```

0.33 Matrix Pitfalls

Defining an array is simple but if you are used to Matlab syntax, there could be some confusion. In Matlab, every array(or vector) is at least a (1,1). In python, it is possible to define a (3,), which can be very frustrating if, for example you are interested in transposing your array from a row to a column. Lets take a look.

create a 3 element array c = array([1,2,3]) c

```
array([1, 2, 3])
```

c.transpose()

```
array([1, 2, 3])
```

That is not what I expected. I thought it would be a column array not, but no. Upon further inspection, welcome to the (3,) dimension array.

c.shape

```
(3,)
```

This can be addressed in a few ways. The safest solution is to change how the arrays are defined.

0.34 Matrix Operations

lets creat some matrices (but recall use the array)

a = array([[1, 2], [3, 4]]) a

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

b = array([[1, 2, 4],[5, 6, 7]]) b

```
array([[1, 2, 4],
       [5, 6, 7]])
```

To perform a matrix multiplication on an array, use the `@`

a @ b

```
array([[11, 14, 18],
       [23, 30, 40]])
```

To perform element wise operation, use `*`

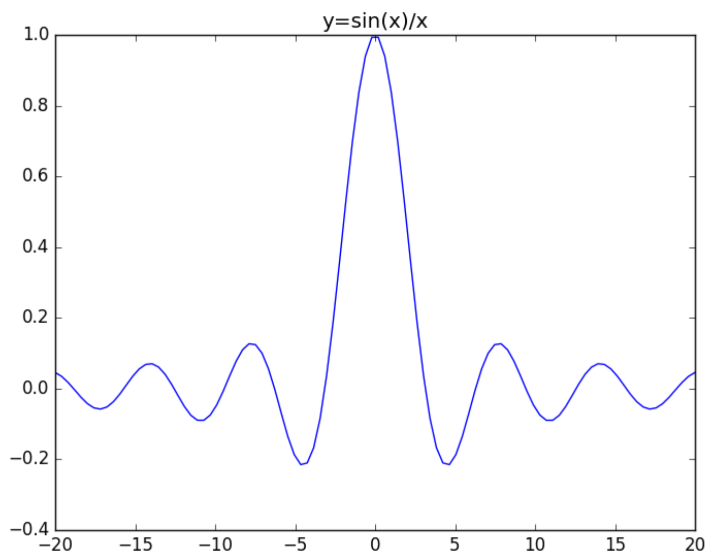
b * 27

```
array([[ 27,  54, 108],
       [135, 162, 189]])
```

0.35 Plotting

Finally, to wrap up this scientific computing tutorial, we end with plotting. Sometimes it is difficult to communicate a calculation or understand complex datasets. Graphs and Plots are such an important part of science and engineering that the python science stack comes with a powerful yet simple plotting packaged called *matplotlib*. Plotting can be just as complicated as the calculations, so here is a simple example to get started. Check out the matplotlib gallery for many more examples.

from numpy import * from matplotlib.pyplot import * x = linspace(-20,20,100) plot(x,sin(x)/x) title('y=sin(x)/x') savefig('fig/matplotlib.png') show()



0.36 Python in Engineering Summary

A brief introduction to the python programming language has been presented to demonstrate the capabilities of python for scientific computing applied to physics and engineering. If your curiosity got the better part of you, please check out the following links that I have found very useful in the (0.37) section.

0.37 References and Links

MATLAB vs Python. Other comparisons of python versus MATLAB
`http://fperez.org/py4science/warts.html`
`http://www.pyzo.org/whypython.html`
`http://www.pyzo.org/python_vs_matlab.html`
`https://www.mathworks.com/products/matlab/matlab-vs-python.html#comparison_table`

https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

Python Syntax Cheatsheets. ipython-chetasheet
conda-chetasheet
python for matlab users

Python Engineering Books. Numerical Python - A Practical Techniques Approach for Industry with source code
Elementary Mechanics Using Python
A Primer on Scientific Programming With Python
Coding the Matrix
Python Scripting for Computational Science
A Primer on Scientific Programming with Python with partial free download
Computation Physics

Python Engineering Library Documentation. Python has mature scientific computation packages, namely scipy

Scipy.
Scipy Cookbook
Scipy Guide
Scipy Lectures
Learning Scipy
Scipy Tutorial

Numpy.
Numpy
Official Docs
Numpy for matlab users
100 numpy examples
numpy on windows

Sympy.
sympy tutorial
sympy features
sympy physics

Matplotlib.
matplotlib gallery and matplotlib tutorial
interactive matplotlib

General Python. The Hitchhikers Guide to Python
A very basic introduction to scientific Python programming

Python and Excel Spreadsheets. xlwings
pandas

Python Online Courses. scientific-python-lectures
Practical Numerical Methods with Python with source code
Aerodynamics / Hydrodynamics with Python
Cornell
Python numerical methods mooc
Computational Physics with Python
Coding the Matrix
How to Think Like a Computer Scientist

Programming Games. project euler
checkio
Numeric Python exercises