4-Day Hands-on Workshop on:

Python for Scientific Computing

and

TensorFlow for Artificial Intelligence

By Dr Stephen Lynch NATIONAL TEACHING FELLOW FIMA SFHEA

Inventor of BINARY OSCILLATOR COMPUTING

Author of PYTHON™, MATLAB®, MAPLE™ AND MATHEMATICA® BOOKS

STEM Ambassador, Public Engagement Champion and Speaker for Schools

s.lynch@mmu.ac.uk

https://www.mmu.ac.uk/computing-and-maths/staff/profile/dr-stephen-lynch

Instructor

Stephen Lynch is a world leader in the use of mathematics packages in teaching, learning, assessment, research and employability. He started using packages in the mid 1980's whilst studying for his PhD in Pure Mathematics. Upon completion of his PhD, he started his lecturing career at Southampton University at the age of 24.

He has authored 2 international patents for inventions, 8 books, 4 book chapters, over 45 journal articles and a few conference proceedings.

In 2022, Stephen was named a **National Teaching Fellow** for his work in Widening Participation, programming in the Maths curriculum and his interdisciplinary research feeding in to teaching. Stephen is a Fellow of the Institute of Mathematics and Its Applications (FIMA), a Senior Fellow of the Higher Education Academy (SFHEA), a Reader with Manchester Metropolitan University and was concurrently an Associate Lecturer with the Open University (2008-2012). In 2010, Stephen volunteered as a STEM Ambassador, in 2012, MMU awarded him a Public Engagement Champion award and in 2014 he became a Speaker for Schools.

Instructor: ResearchGate

ResearchGate Search or Discover by subject area Join for free Log in

Stephen Lynch

PhD Mathematics NTF FIMA SFHEA · Professor (Associate)

Binary oscillator computing with Josephson junctions and biological neurons. Seeking researchers in CMOS and memristors.

Research Interest Score 1,665

Citations 1,647

h-index 23

Citations over time

Institution and department

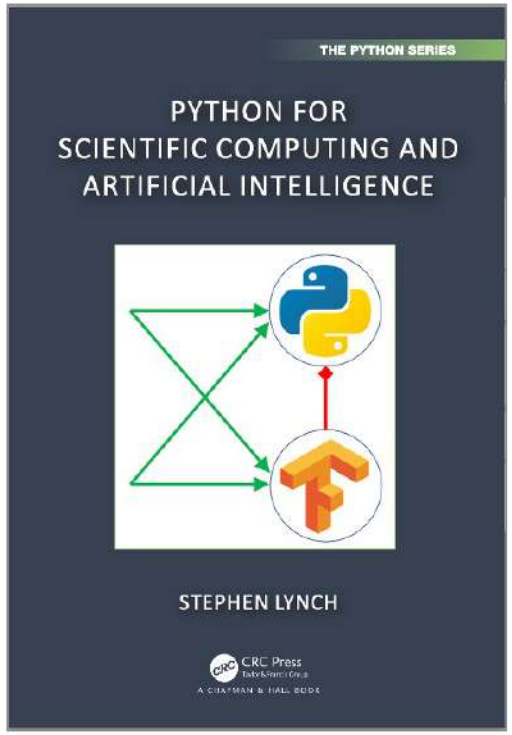
Manchester Metropolitan University · Department of Computing and Mathematics

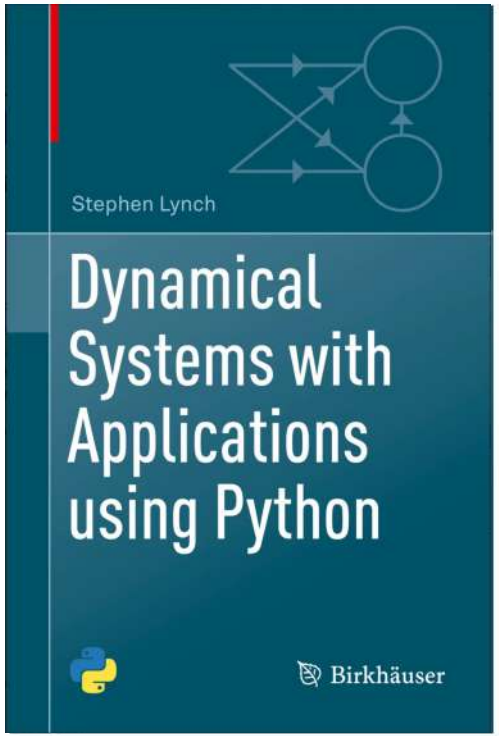
Skills

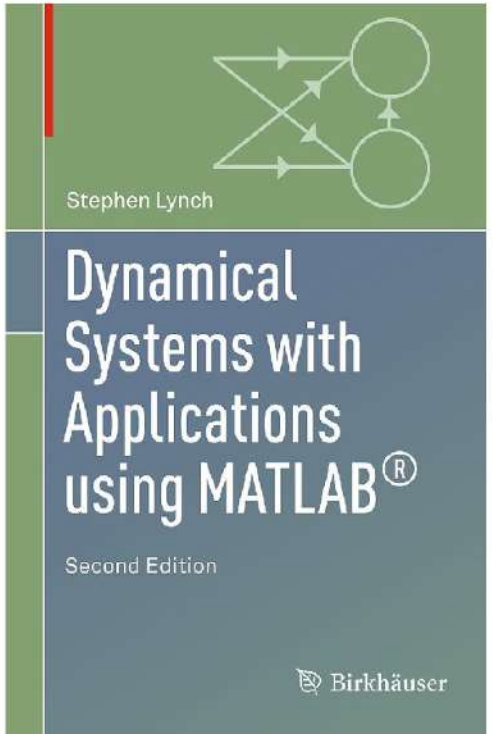
Mathematica Programming · Maple (Software) · MATLAB + 46 others

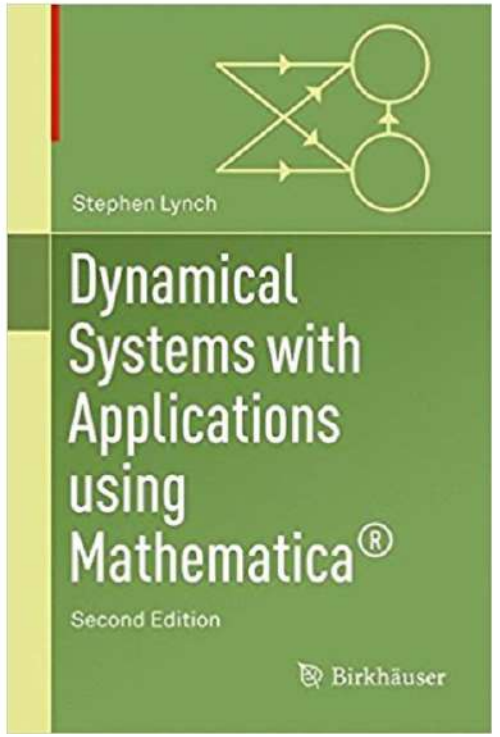
https://www.researchgate.net/profile/Stephen_Lynch?ev=prf_highl

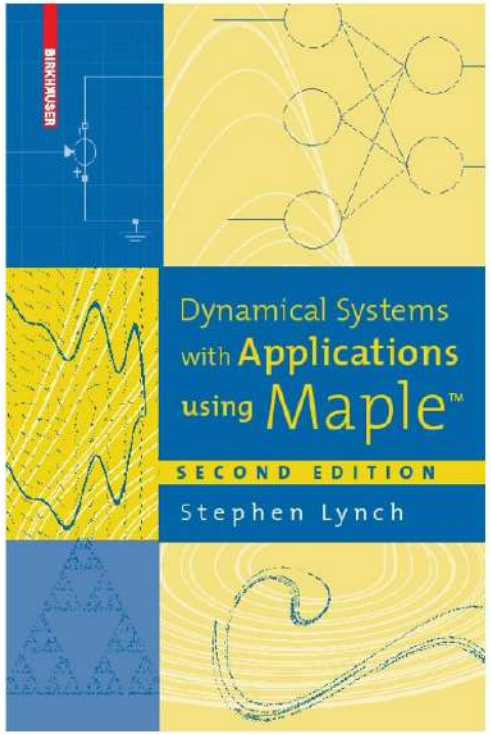
Instructor: Books











There are over 600 Programming Languages

PYPL PopularitY of Programming Language

Worldwide, Apr 2023 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	27.43 %	-0.8 %
2		Java	16.41 %	-1.7 %
3		JavaScript	9.57 %	+0.3 %
4		C#	6.9 %	-0.3 %
5		C/C++	6.65 %	-0.5 %

IMA Maths Careers (2021): Python for A-Level Maths, Undergraduate Maths and Employability
https://www.mathscareers.org.uk/python-for-a-level-maths-undergraduate-maths-and-employability/

Schedule Day 1 (See Brochure for 5-Day Itinerary)

Day 1			
Using Python as a Calculator	10am-11am	Numpy and Matplotlib	2pm-3pm
Simple Programming	11am-12pm	Sympy – Symbolic Computation	3pm-4pm
Simple Plots using Turtle	12pm-1pm		

Download all files from GitHub:

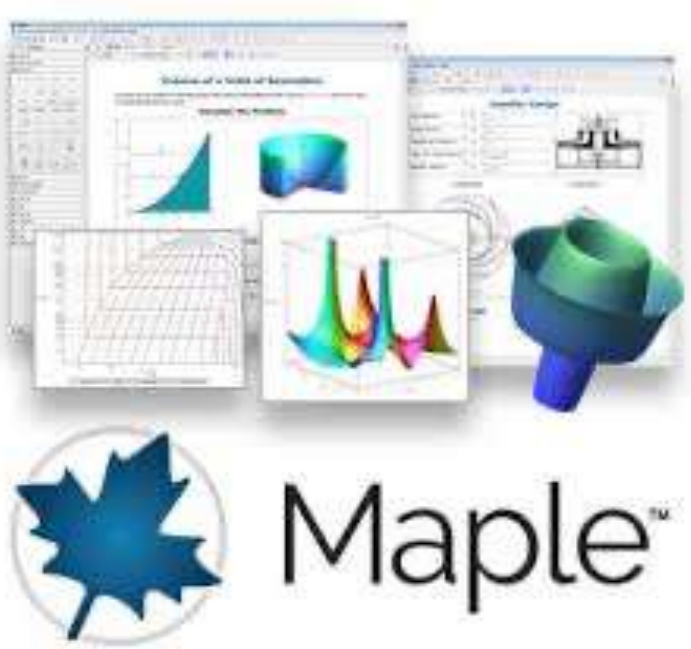
https://github.com/proflynch/CRC-Press/

Solutions to the Exercises in Section 1:

https://drstephenlynch.github.io/webpages/Solutions_Section_1.html

Manchester Metropolitan University

Maple, MATLAB and Mathematica: Symbolic Computation and Modelling



https://www.maplesoft.com

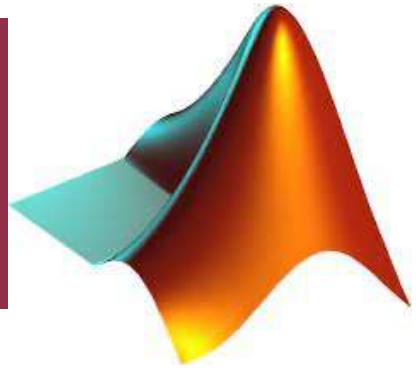
https://www.mathworks.com

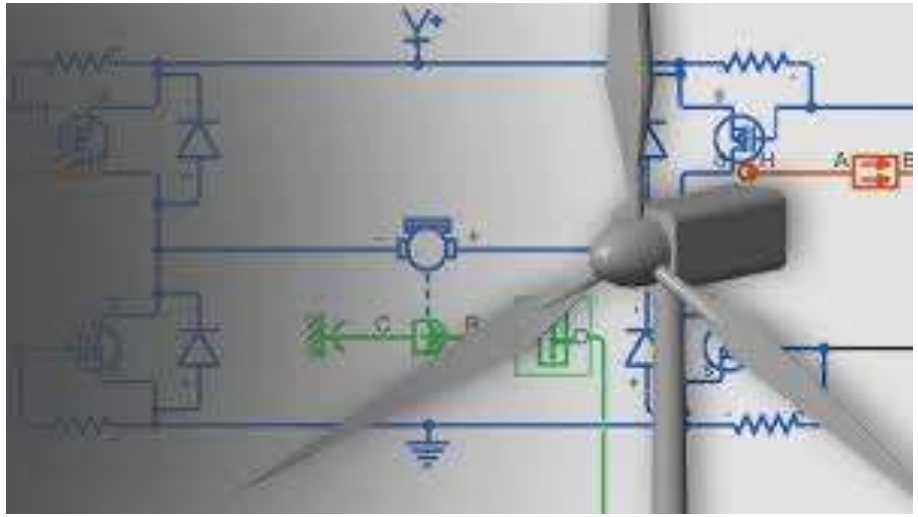
https://www.wolfram.com/mathematica/

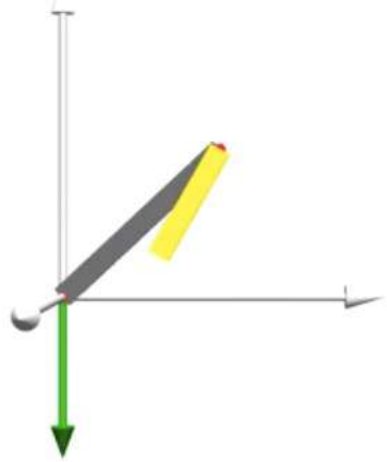
Maple, MATLAB and Mathematica for SIMULATION

MapleSim™
Advanced System-Level Modeling

MATLAB®
Simulink®
Simscape®





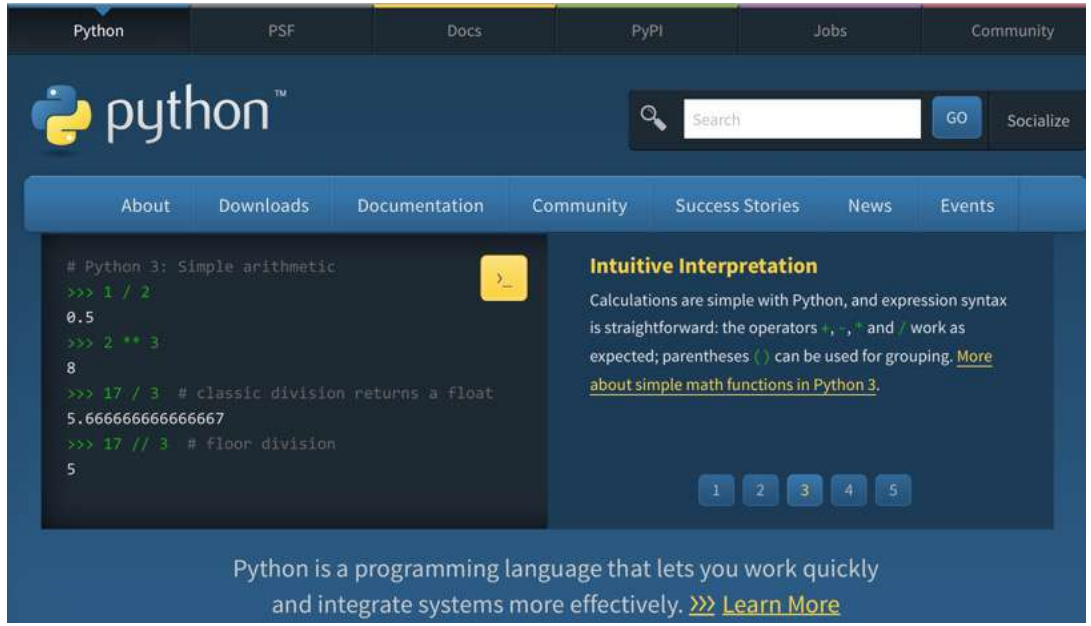


Wolfram
SystemModeler™

Download URLs for Python

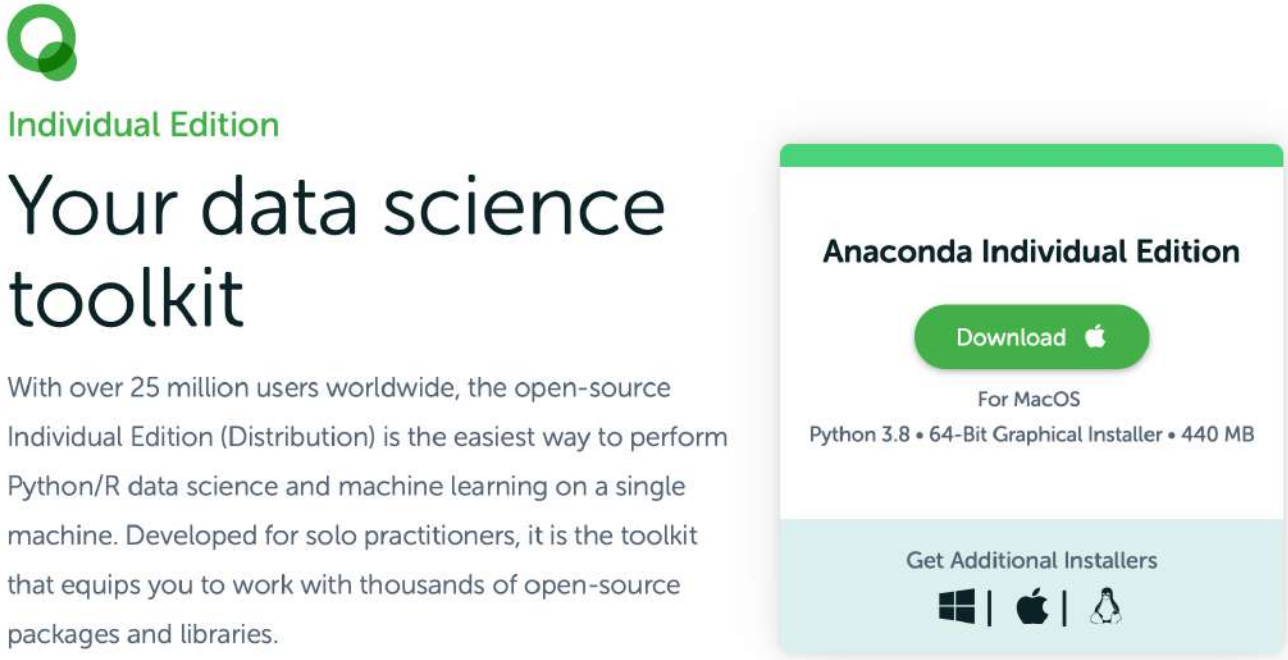
1. To download Python (IDLE):

https://www.python.org/

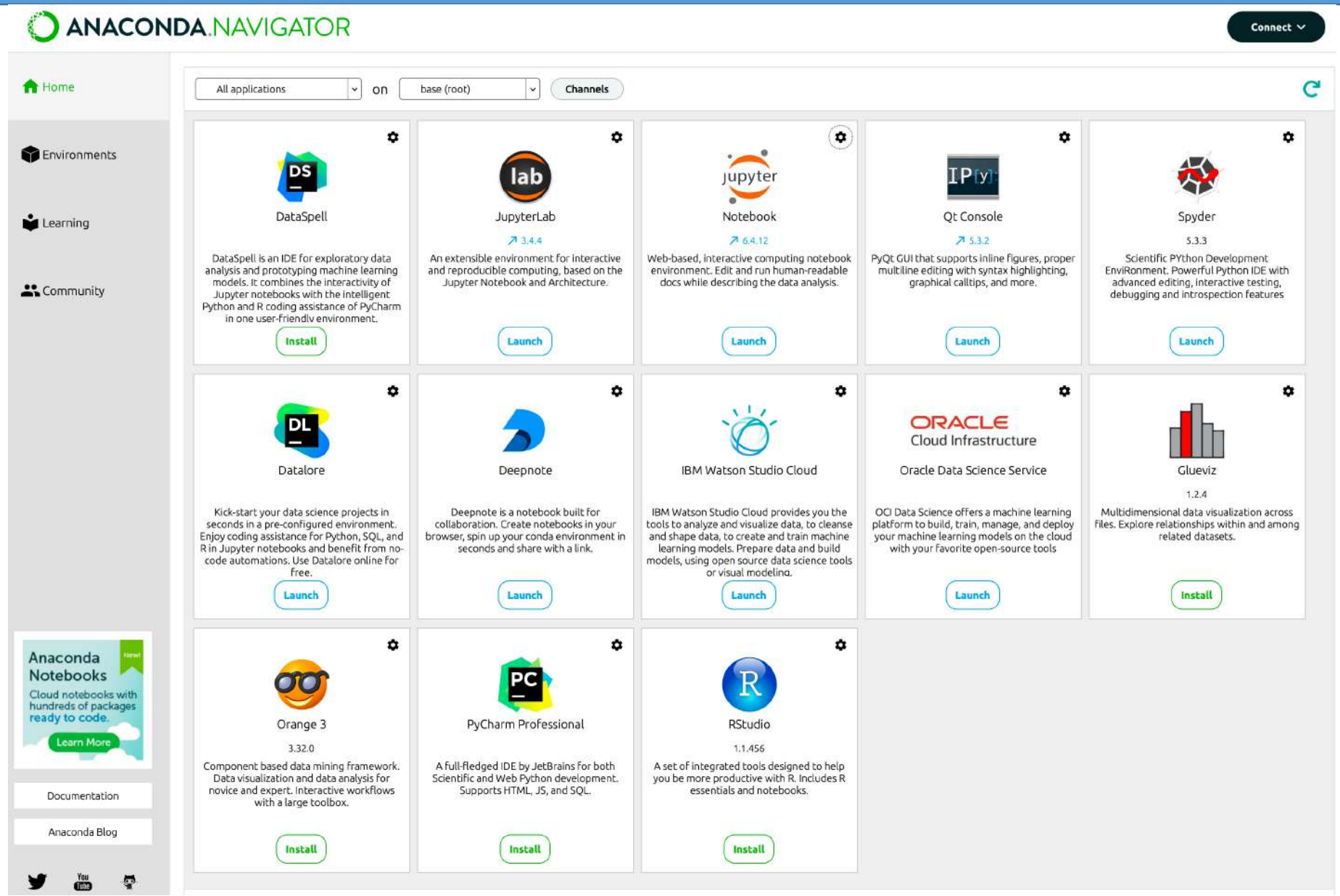


2. To download Anaconda:

https://www.anaconda.com/products/individual



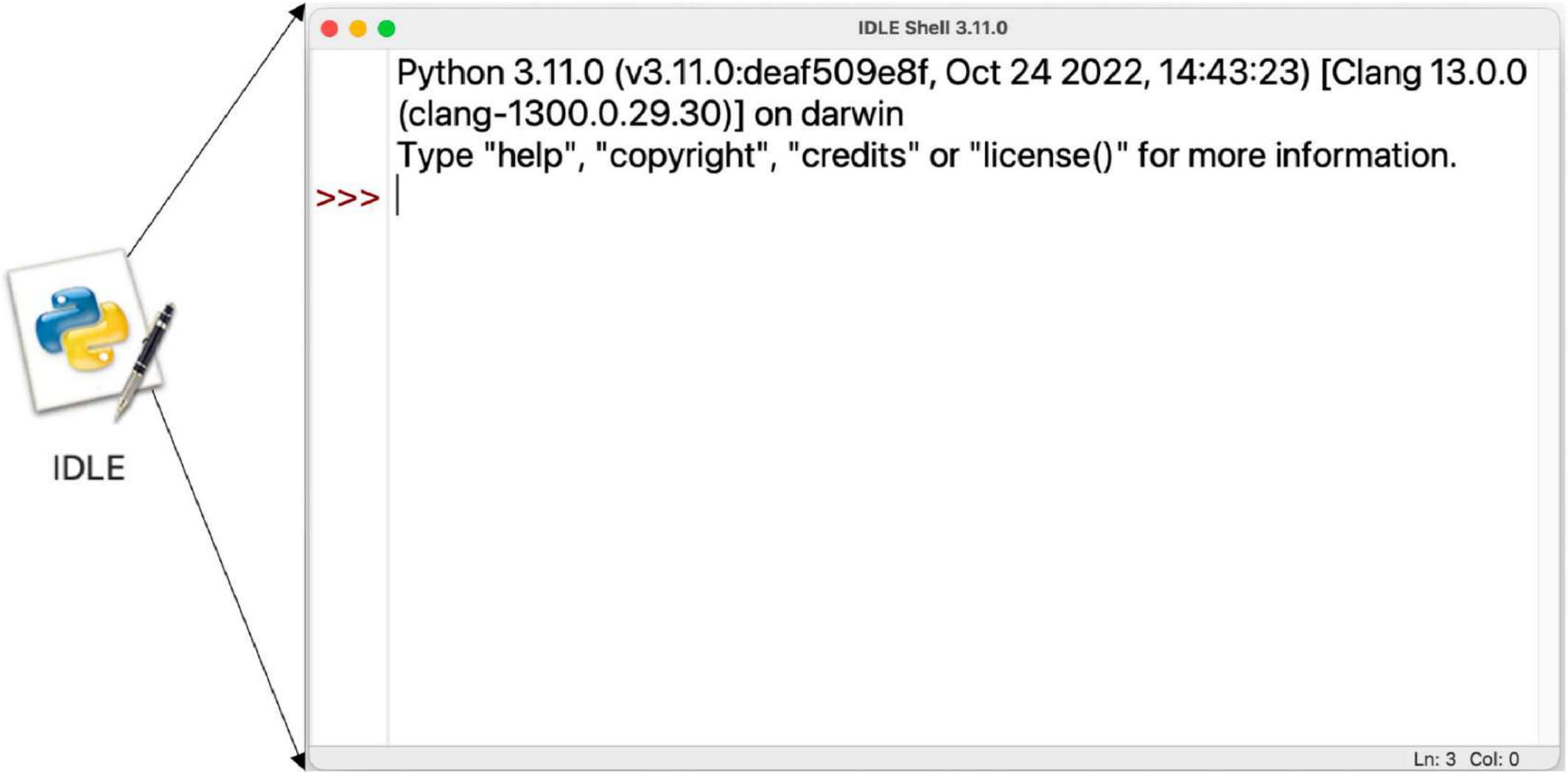
Anaconda Free Package Manager



Manchester Metropolitan University

Let’s Start with Python IDLE: Mac Launchpad

The IDLE (Integrated Development Learning Environment) Shell



Using Python as a Powerful Calculator

Python Command Lines	Comments
`>>> # This is a comment.`	`# Writing comments in Python.`
`>>> 4 + 5 - 3`	`# Addition and subtraction.`
`>>> 2 * 3 / 6`	`# Multiplication and division.`
`>>> 2**8`	`# Powers.`
`>>> import math`	`# Import the math module (or library).`
`>>> help(math)`	# List the functions.
`>>> math.sqrt(9)`	`# Prefix math for square root.`
`>>> from math import *`	`# Import all math functions.`
`>>> sin(0.5)`	`# The sine function(radians).`
`>>> asin(0.4794)`	`# Inverse sine function.`
`>>> degrees(pi)`	`# Convert radians to degrees.`
`>>> radians(90)`	`# Convert degrees to radians.`
`>>> log(2)`	`# Natural logarithm.`
`>>> log10(10)`	`# Logarithm base 10.`

Using Python as a Powerful Calculator

```
>>> exp(2)                          # Exponential function.
>>> e**2                            # Exponential function using e.
>>> cosh(0.3)                       # Hyperbolic coshine function.
>>> fmod(13, 6)                     # Modulo arithmetic.
>>> 13 % 6                          # Returns the remainder.
>>> gcd(123, 321)                   # Greatest common divisor.
>>> 1 / 3 + 1 / 4                   # Floating point arithmetic.
>>> from fractions import Fraction  # Load the fractions function Fraction.
>>> Fraction(1, 3) + Fraction(1, 4) # Symbolic computation.
>>> pi                              # The number π.
>>> round(_, 5)                     # Round last output to 5 decimal places.
>>> factorial(52)                   # Gives 52!
>>> ceil(2.5)                       # Ceiling function.
>>> floor(2.5)                      # Floor function.
>>> trunc(-2.5)                     # Truncates nearest integral to zero.
>>> quit()                          # Quits Python IDLE.
```

Using Python as a Powerful Calculator (Lists)

Python Command Lines	Comments
`>>> a = [1, 2, 3, 4, 5]`	`# A simple list.`
`>>> type(a)`	`# a is a class list.`
`>>> a[0]`	`# 1st element, zero-based indexing.`
`>>> a[-1]`	`# The last element.`
`>>> len(a)`	`# The number of elements.`
`>>> min(a)`	`# The smallest element.`
`>>> max(a)`	`# The largest element.`
`>>> 5 in a`	`# True, 5 is in a.`
`>>> 2 * a`	`# [1,2,3,4,5,1,2,3,4,5].`
`>>> a.append(6)`	`# Now a=[1, 2, 3, 4, 5, 6]`
`>>> a.remove(6)`	`# Removes the first 6.`
`>>> print(a)`	`# a=[1, 2, 3, 4, 5].`
`>>> a[1 : 3]`	`# Slice to get [2, 3].`

Using Python as a Powerful Calculator (Lists) (END SESSION 1)

```
>>> a[1:]                          # Slice to get [2, 3, 4, 5]
>>> a[:-2]                         # Slice to get [1, 2, 3]
>>> a[1 : 3]                       # Slice to get [2, 3].
>>> list(range(5))                 # [0, 1, 2, 3, 4].
>>> list(range(4 , 9))             # [4, 5, 6, 7, 8].
>>> list(range(2, 10, 2))          # [2, 4, 6, 8].
>>> list(range(10, 5, -2))         # [10, 8, 6].
>>> A = [[1, 2], [3, 4]]           # A list of lists.
>>> A[0][1]                        # Second element in list one.
>>> names = ["Jon", "Seb", "Liz"]  # A list of names.
>>> names.index("Seb")             # Returns 1.
>>> names.pop(1)                   # Returns 'Seb' and removes from names.
>>> quit()                         # Quits Python IDLE.
```

Python Programming (Session 2)

We will concentrate on three programming structures:

1. defining functions;

2. for and while loops;

3. if, elif, else constructs.

Programming: Philosophy of the Book

1. The reader is encouraged to learn programming from exemplar programs listed in the book and available to download online.

2. The reader should look up syntax to understand how the programs work.

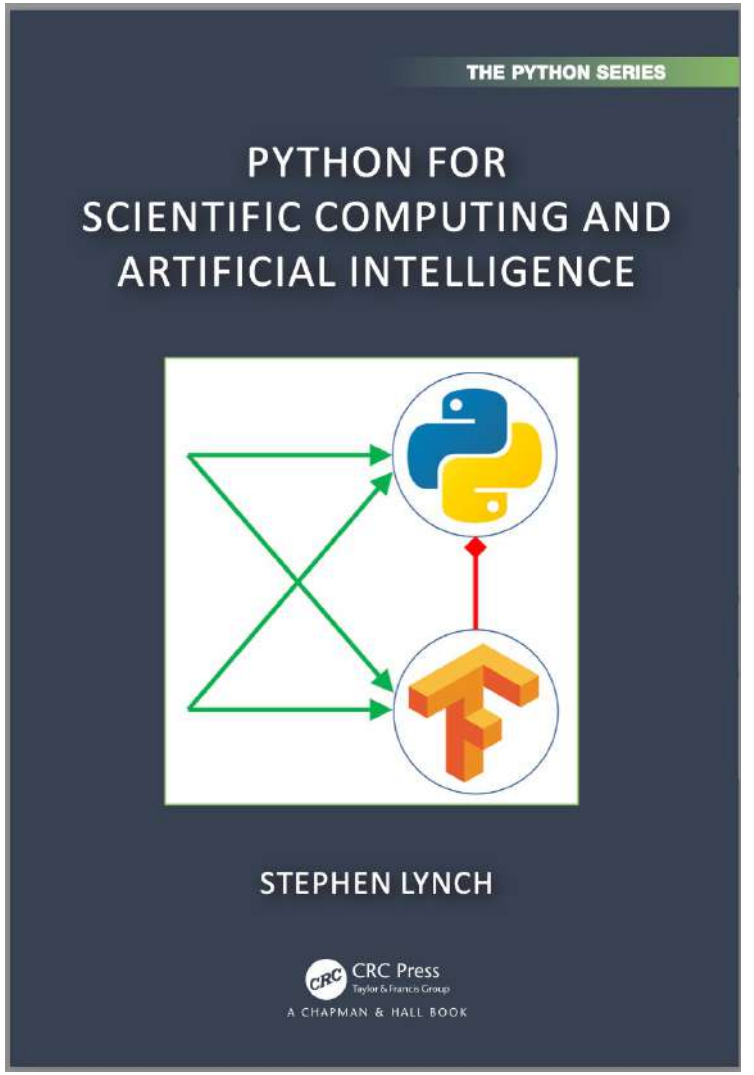
3. The reader should edit working programs before attempting to write their own code from scratch.

Download all files from GitHub:

https://github.com/proflynch/CRC-Press/

Solutions to the Exercises in Section 1:

https://drstephenlynch.github.io/webpages/Solutions_Section_1.html



Hints for Programming

1. Indentation: The indentation level in Python code is significant.

2. Common typing errors: Include all operators, make sure parentheses match up in correct pairs, Python is case sensitive, check syntax using the help command.

3. Use continuation lines: Use a backslash to split code across multiple lines.

4. Preallocate arrays using the zeros command.

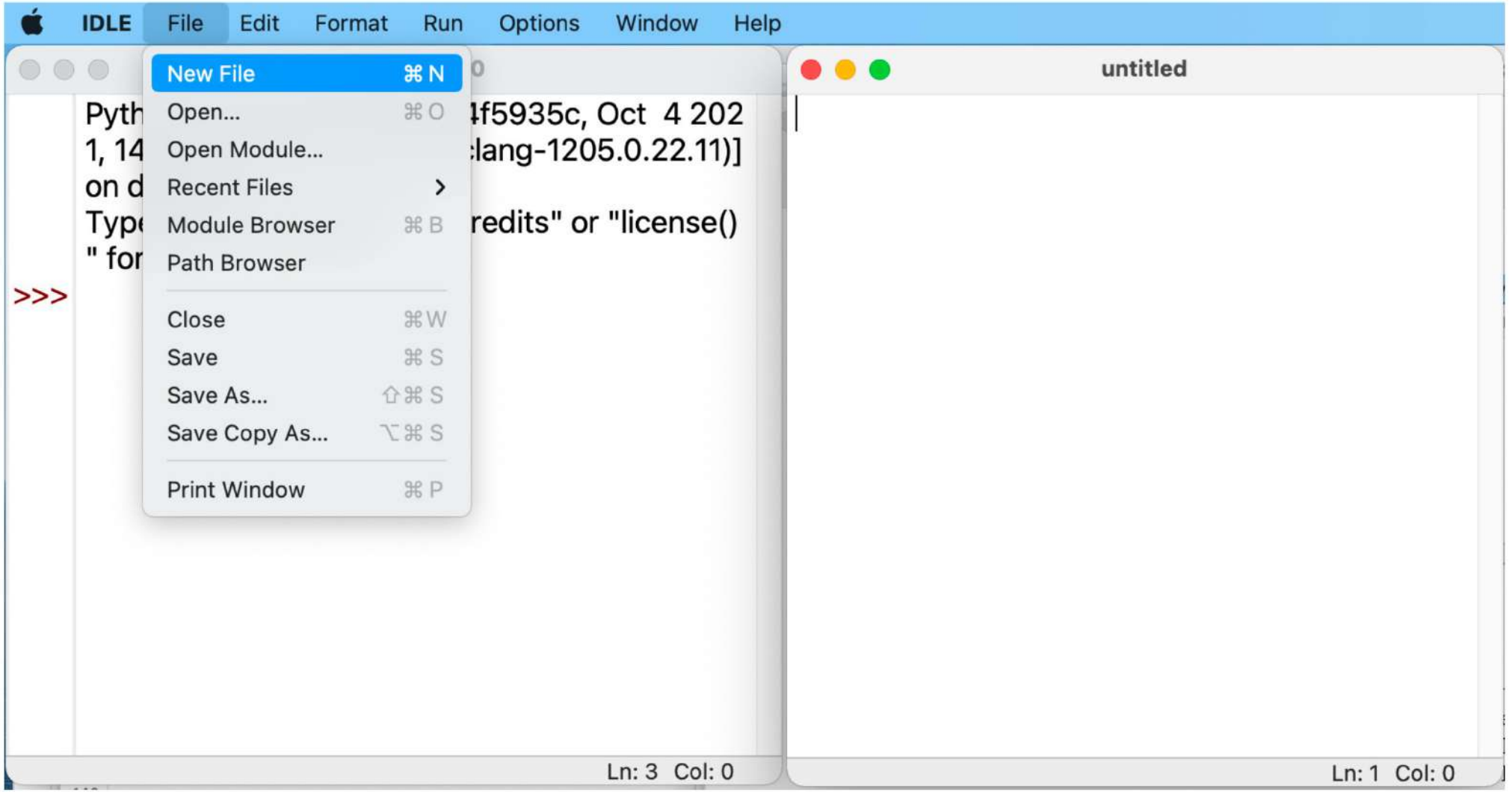
5. If a program involves a lot of iterations, 100,000, say, then run the code for two iterations initially and use print.

6. Read the warning messages supplied by Python before running the code.

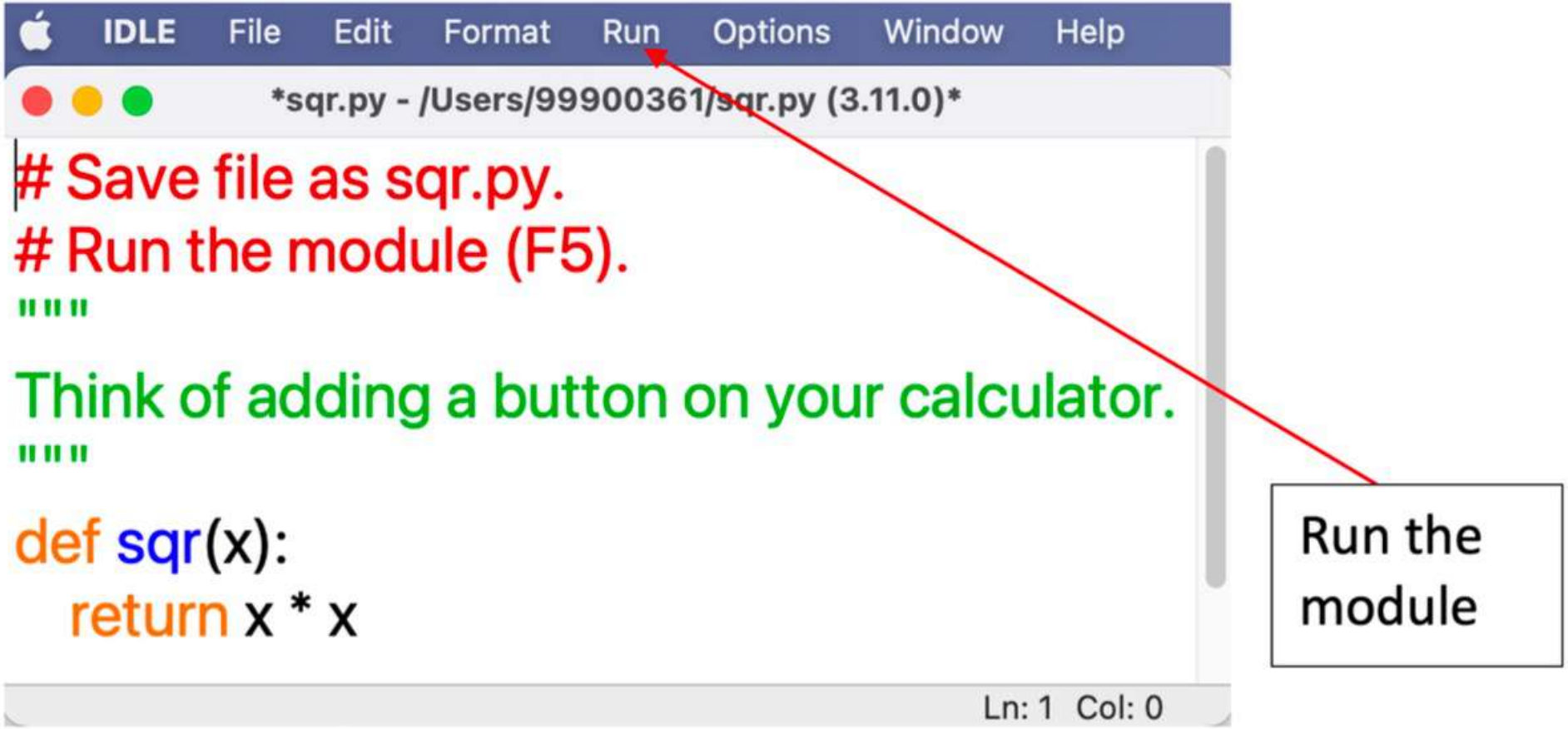
7. Check that you are using the correct libraries and modules.

8. If you cannot get your program to work, look for similar programs (including Maple, Mathematica and MATLAB programs) on the World Wide Web.

Simple Programming: New File



Simple Programming: Run the Module Before Calling the Function



Simple Programming (Functions)

sqr.py – Editor Window

```
# The square function - save file as sqr.py.
# Run the Module (or type F5).
"""
This is our first Python program.
Think of adding a square button onto your
Python calculator.
"""
def sqr(x):
    return x * x
```

Python Shell

```
>>> sqr(-9)
81
```

Simple Programming (Functions)

IDLE File Edit Format Run Options Window Help

*f_mu.py - /Users/slynch/Documents/Stephen/Maths_Packages/Python/Pyt...

```
# The logistic map function - save file as f_mu.py.
# Run the Module (or type F5).
"""
You can write your own text here.
Created on Mon Mar 12 09:23:47 2018
@author: sladmin
"""
def f_mu(mu, x):
    return mu * x * (1 - x)
```

Ln: 1 Col: 0

```
>>> f_mu(2, 0.8)
0.31999999999999995
```

Simple Programming (Functions)

IDLE File Edit Format Run Options Window Help

F2K.py - /Users/slynch/Documents/Stephen/MMU Python Workshop/Python_Programs/F2K.py (3.8.2)

```
# A function to convert degrees Fahrenheit to Kelvin.
# Save file as F2K.py.
# Run the Module (or type F5).
def F2K():
    F = float(input('Enter temperature in degrees Fahrenheit: '))
    K = (F + 459.67) * 5 / 9
    print('Temperature in Kelvin is {:08.4f} K'.format(K))
```

Ln: 7 Col: 38

```
>>> F2K()
Enter temperature in degrees Fahrenheit: 35.68
Temperature in Kelvin is 275.1944 K
```

Simple Programming (For Loops)

Fibonacci.py – Editor

```
# A function to list the n terms of the Fibonacci sequence.
# Save file as Fibonacci.py.
def Fibonacci(n):
    a , b = 0 , 1
    print(a) , print(b) , print(a+b)
    for i in range(n-3):
        a , b = b , a + b
        print(a + b)
```

>>> Fibonacci(20)

0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181

Simple Programming (While Loops)

IDLE File Edit Format Run Options Window Help

sum_n.py - /Users/99900361/Documents/Python/sum_n.py (3.10.0)

```
# Sum natural numbers to N.
# Save as sum_n.py.
# Run the module (or type F5).
def sum_n(n):
    sum , i = 0 , 1
    while i <= n:
        sum += i      # sum = sum + i
        i += 1        # i = i + 1
    print('The sum is ', sum)
```

Ln: 1 Col: 0

```
>>> sum_n(100)
The sum is 5050
```

Simple Programming (If, Elif, Else): End Session 2

Grade.py – Editor Window

```
# A program that grades scores.
# Save file as Grade.py.
# Run the Module (or type F5).
def Grade(score):
    if score >= 70:
        letter = "A"
    elif score >= 60:
        letter = "B"
    elif score >= 50:
        letter = "C"
    elif score >= 40:
        letter = "D"
    else:
        letter = "F"
    return letter
```

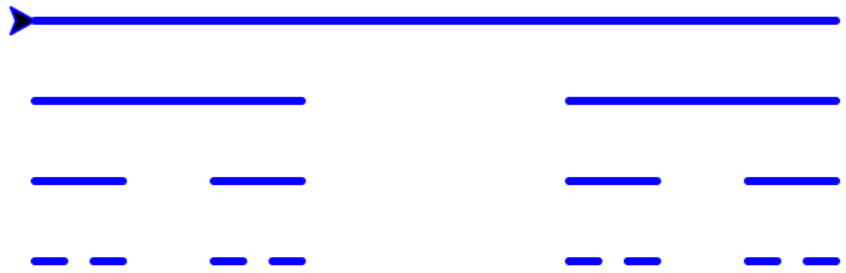
```
>>> grade(90)
'A'
```

The Turtle Module (The Cantor Set): Start Session 3

```
# Cantor fractal set.
# Save file as cantor.py.
# Run the module (F5).
from turtle import *
def cantor(x, y, length):
    if length >= 5:                 # Exit program if length < 5.
        speed(0)                    # Set fastest speed.
        penup()                     # Raise the turtle.
        pensize(3)                  # Line thickness.
        pencolor('blue')
        setpos(x , y)               # Coordinates of start point.
        pendown()                   # Put turtle down.
        fd(length)                  # Forward.
        y -= 30                     # y = y - 30.
        cantor(x , y, length / 3)
        cantor(x + 2 * length / 3, y, length / 3)
        penup()
        setpos(x , y + 30)
```

Cantor set: Remove the middle third segment at each stage.

```
>>> cantor(-200, 200, 300)
```

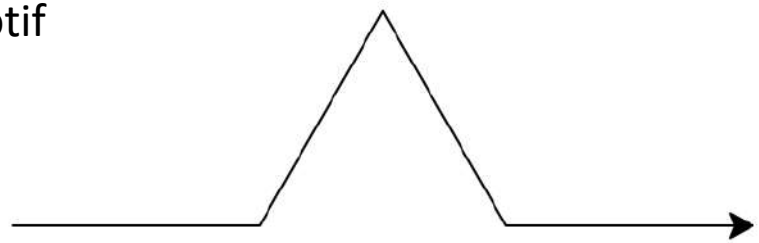


Problem: Edit the program to plot a variant of the Cantor set where the two middle fifth segments are removed at each stage.

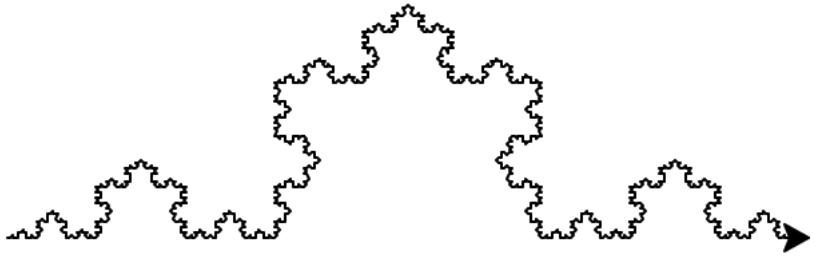
The Turtle Module (The Koch Curve)

```
# Koch curve fractal.
# Save file as koch_curve.py.
# Rumn module (F5).
from turtle import *
def koch_curve(length, stage):
    speed(0)
    if stage==0:
        fd(length)
        return
    koch_curve(length / 3 , stage - 1)
    lt(60)
    koch_curve(length / 3 , stage - 1)
    rt(120)
    koch_curve(length / 3 , stage - 1)
    lt(60)
    koch_curve(length / 3 , stage - 1)
```

Motif



```
>>> koch_curve(300 , 5)
```



Problem: Edit the program to plot a Koch square fractal, where one segment (one third length) is replaced with 5 segments.

The Turtle Module (The Sierpinski Triangle Fractal)

```
# Sierpinski triangle.
# Save file as sierpinski.py.
# Run the module (F5).
from turtle import *
def sierpinski(length , level):
    speed(0)
    if level == 0:
        return
    begin_fill()      # Fill shape.
    color('red')
    for i in range(3):
        sierpinski(length / 2 , level - 1)
        fd(length)
        lt(120)       # Left turn 120 degrees.
    end_fill()
```

```
>>> sierpinski(200 , 5)
```

a

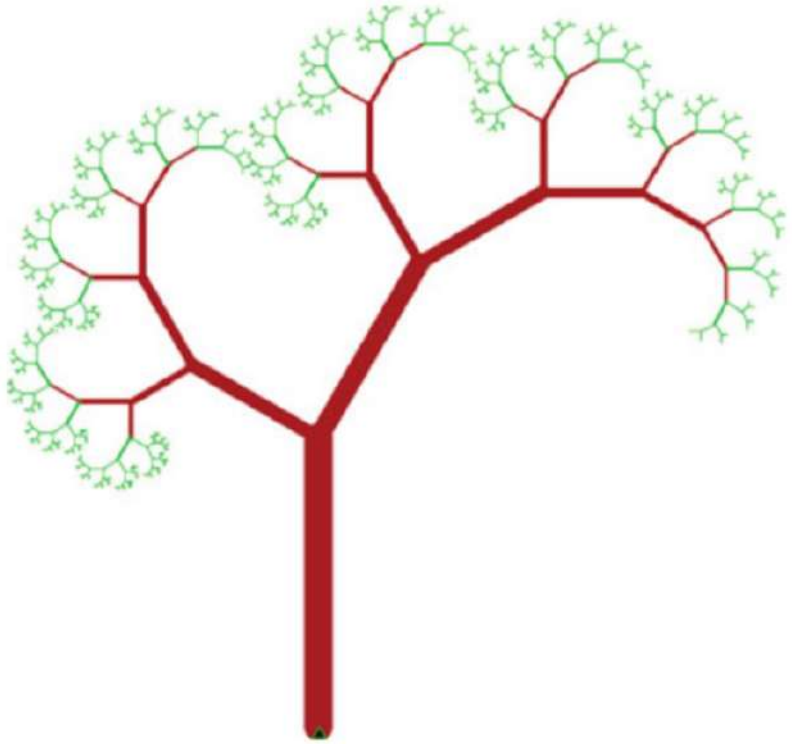
b

Problem: Edit the program to plot a Sierpinski square fractal, where the central square is removed at each stage.

The Turtle Module (A Fractal Tree) End Session 3

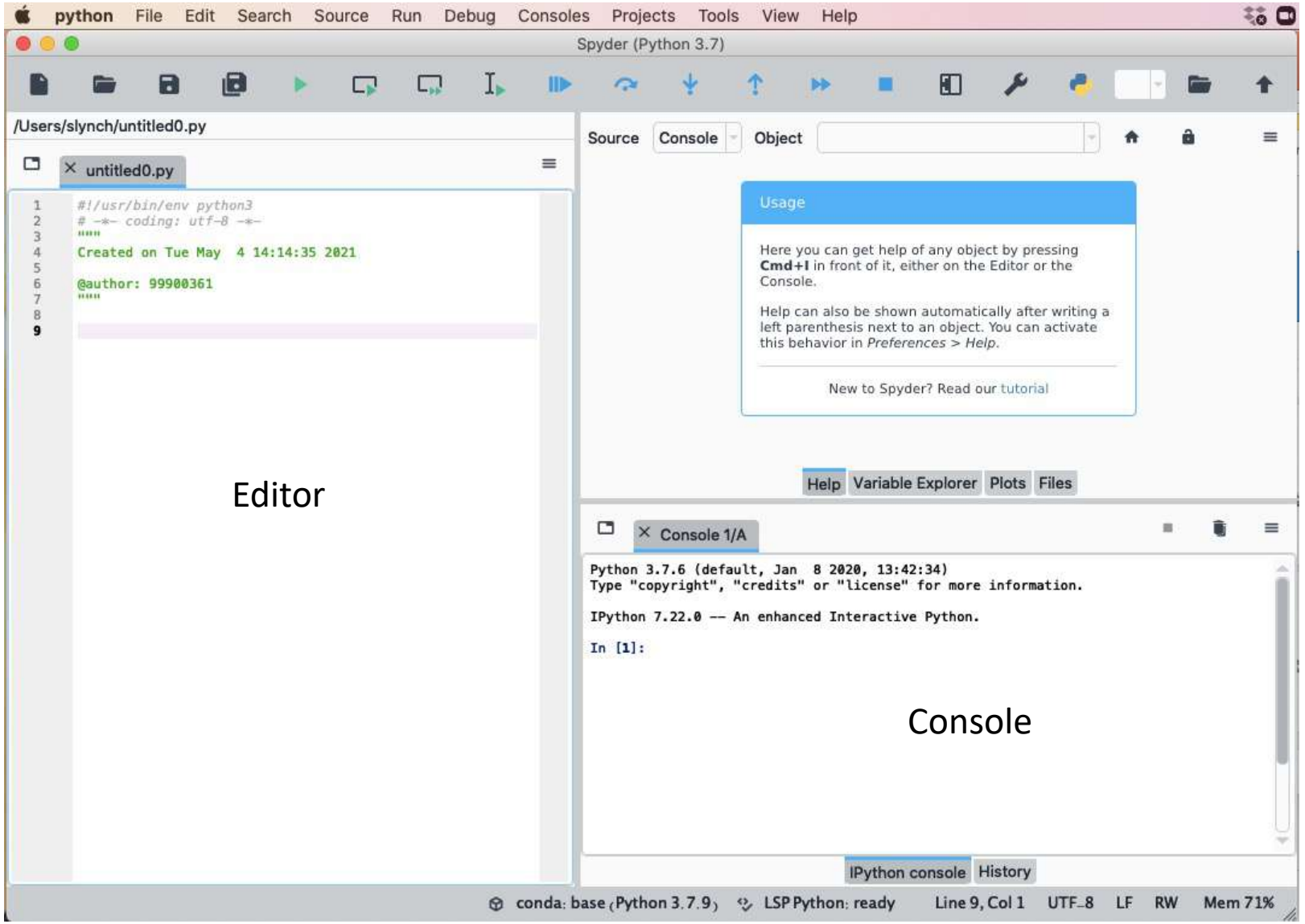
```
# A colour fractal tree.
# In the IDLE Shell type fractal_tree_color(200, 10).
from turtle import *
setheading(90)          # Turtle points up.
penup()
setpos(0, -250)
pendown()
def fractal_tree_color(length, level):
    pensize(length / 10)
    if length < 20:
        pencolor('green')
    else:
        pencolor('brown')
    speed(0)
    if level > 0:
        fd(length)              # forward
        rt(30)                  # right turn 30 degrees
        fractal_tree_color(length * 0.7, level - 1)
        lt(90)                  # left turn 90 degrees
        fractal_tree_color(length * 0.5, level - 1)
        rt(60)                  # So turtle points stright up
        penup()
        bk(length)
        pendown()
```

```
>>> fractal_tree_color(200, 10)
```



Problem: Can you plot a trifurcating tree?

Spyder: Launch from Anaconda: Start Session 4



Numpy (NUMeric PYthon) in the Spyder Console

Python Command Lines	Comments
`In[1]: import numpy as np`	`# Import numpy into the np namespace.`
`In[2]: A = np.arange(5)`	`# An array, A=[0 1 2 3 4].`
`In[3]: print(A)`	`# Print the array. Note, no commas.`
`In[4]: A.tolist()`	`# Convert an array to a list.`
`In[5]: print(Out[4])`	`# You can use previous Output.`
`In[6]: u = np.array([1 , 2 , 3])`	`# A 3D vector and rank one tensor.`
`In[7]: v = np.array([4 , 5 , 6])`	`# A 3D vector.`
`In[8]: np.dot(u , v)`	`# The dot product of two vectors.`
`In[9]: np.cross(u , v)`	`# The cross product of two vectors.`
`In[10]: B = np.array([[1,1],[0,1]])`	`# A 2D array, rank 2 tensor.`
`In[11]: C = np.array([[2,0],[3,4]])`	`# A 2D array.`
`In[12]: B * C`	`# Elementwise product.`
`In[13]: np.dot(B , C)`	`# Matrix product.`
`In[14]: D=np.arange(9).reshape(3,3)`	`# A 2D array.`
`In[15]: D.sum(axis = 0)`	`# Sum each column.`

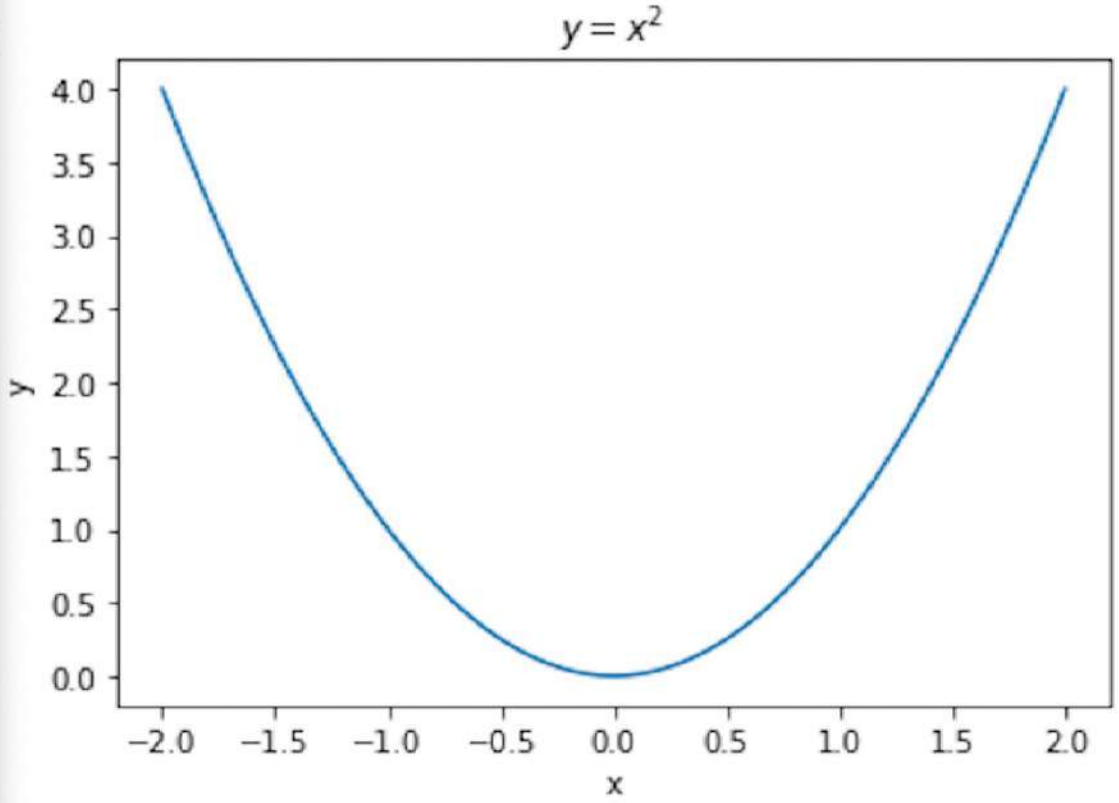
NumPy (NUMeric PYthon) in the Spyder Console

```
In[16]: D.max(axis = 0)                 # The maximum of each column.
In[17]: D.min(axis = 1)                 # The minimum of each row.
In[18]: D.cumsum(axis = 1)              # Cumulative sum of each row.
In[19]: type(D)                         # numpy.ndarray.
In[20]: D.ndim                          # The tensor rank of D is 2.
In[21]: E = np.zeros((3 , 3))           # Creates a 3 × 3 array of zeros.
In[22]: np.ones((3 , 3))                # Creates a 3 × 3 array of ones.
In[23]: np.full((3 , 3) , 7)            # Creates a 3 × 3 array of 7s.
In[24]: I = np.eye(3)                   # Creates a 3 × 3 identity matrix.
In[25]: M=np.arange(12).reshape(3,4)    # A 3 × 4 array.
In[26]: M[1 , 3]                        # The element in row 2 column 4.
In[27]: M[: , 1]                        # Column 2, array([1,5,9]).
In[28]: N = M[: 2 , 1 : 3]              # Slice to get subarray.
In[29]: print(N)                        # First 2 rows, and cols 2 and 3.
In[30]: quit                            # Restart the kernel.
```

MatPlotLib (MATrix PLOTting LIBrary) in Spyder

plot1.py

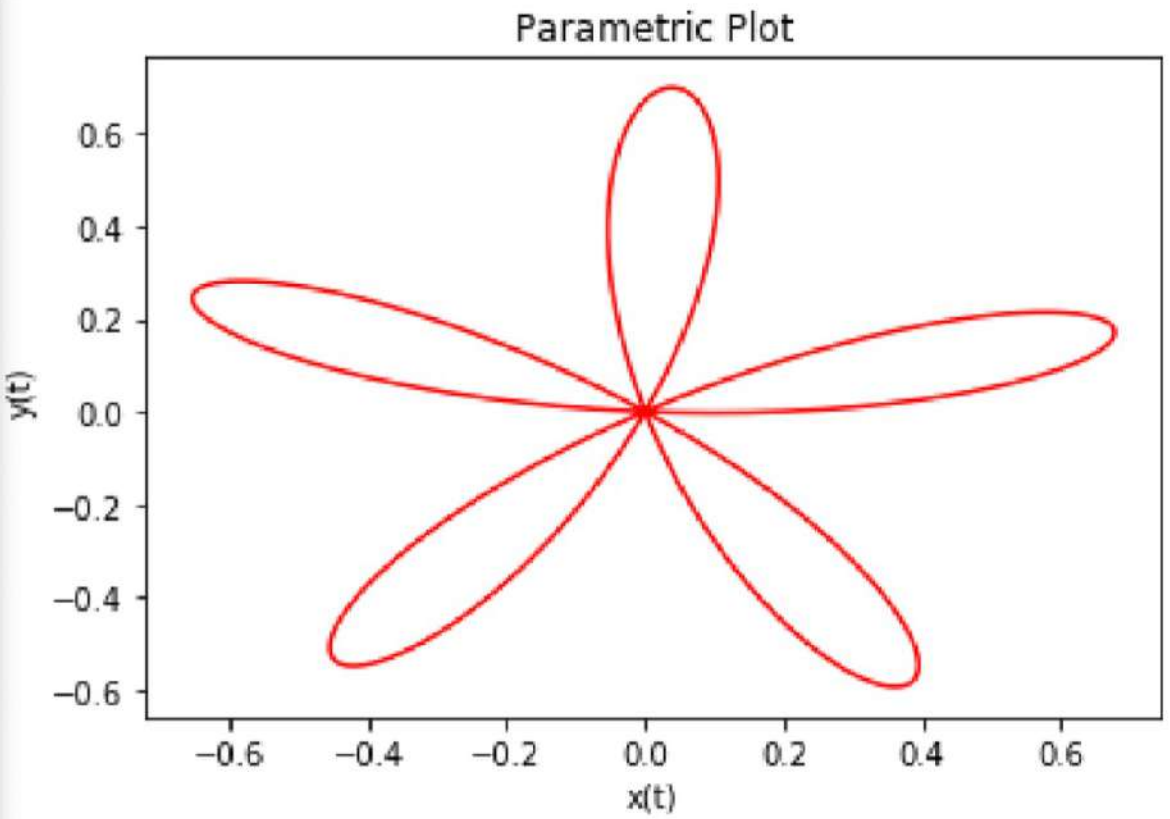
```
1  #  Save file as plot1.py.
2  import numpy as np
3  import matplotlib.pyplot as plt
4  x = np.linspace(-2 , 2, 100)
5  y = x**2
6  plt.xlabel("x")
7  plt.ylabel("y")
8  plt.title("$y=x^2$")
9  plt.plot(x , y)
10 plt.savefig("Parabola.png" , dpi = 400)
11 plt.show()
12
```



Matplotlib (MATrix PLOTting LIBrary) in Spyder

plot2.py

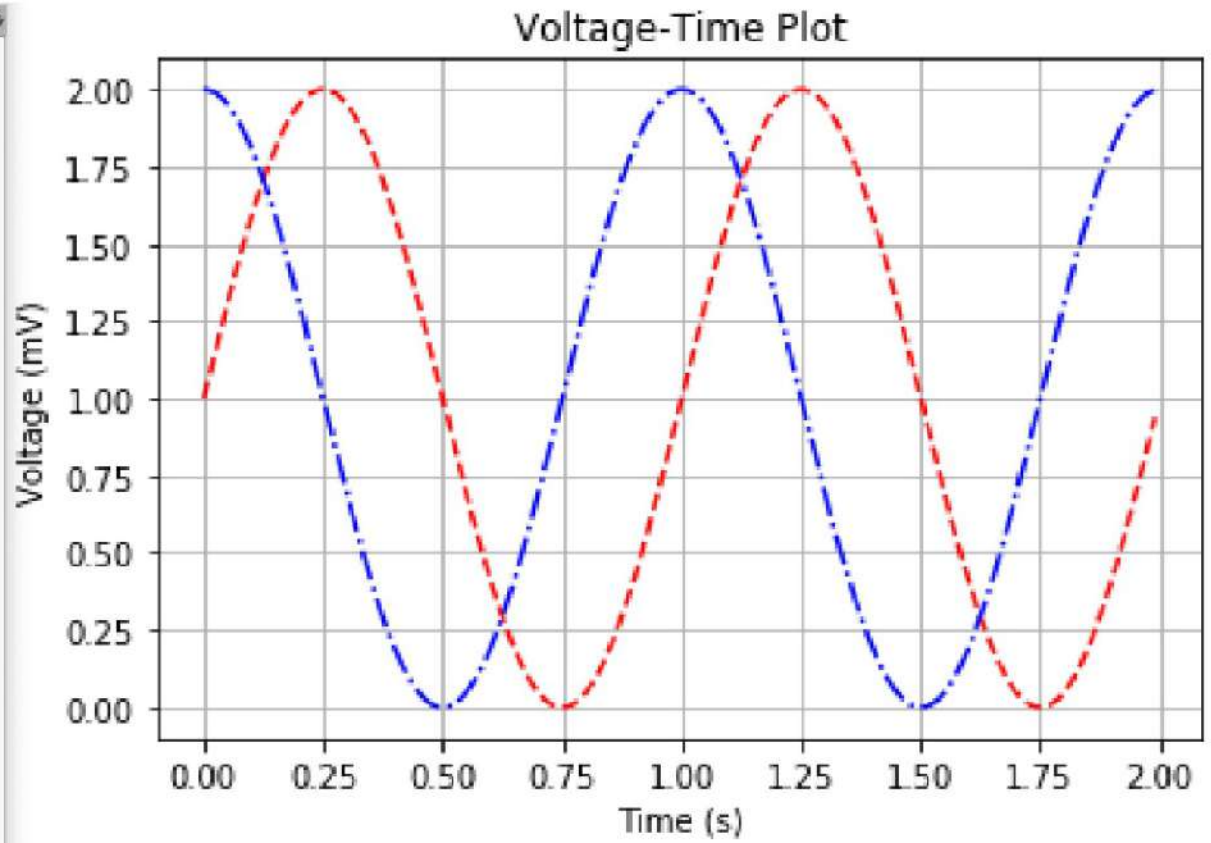
```
1# Save as plot2.py.
2import numpy as np
3import matplotlib.pyplot as plt
4t = np.linspace(0 , np.pi , 1000)
5x = 0.7 * np.sin(t+1)*np.sin(5*t)
6y = 0.7 * np.cos(t+1)*np.sin(5*t)
7plt.xlabel("x(t)")
8plt.ylabel("y(t)")
9plt.title("Parametric Plot")
10plt.plot(x , y , color = "red")
11plt.show()
12
```



Matplotlib (MATrix PLOTting LIBrary) in Spyder

plot3.py*

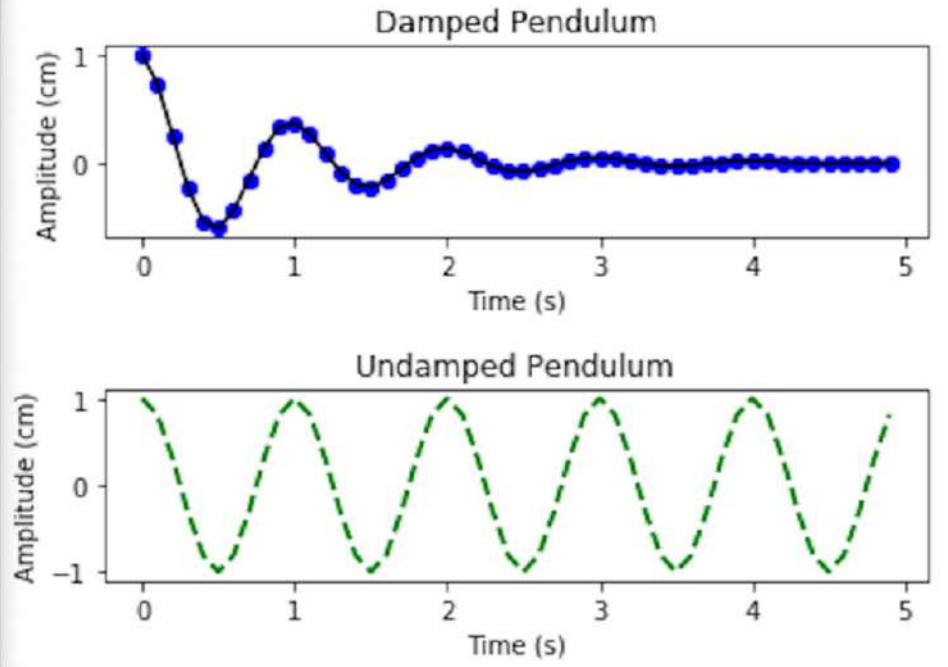
```
1# Save as plot3.py.
2# Voltage-Time Plot.
3import numpy as np
4import matplotlib.pyplot as plt
5t = np.arange(0 , 2 , 0.01)
6c = 1 + np.cos(2 * np.pi * t)
7s = 1 + np.sin(2 * np.pi * t)
8plt.plot(t , s , "r--" , t , c , "b-.")
9plt.xlabel("Time (s)")
10plt.ylabel("Voltage (mV)")
11plt.title("Voltage-Time Plot")
12plt.grid(True)
13plt.savefig("Voltage-Time Plot.png")
14plt.show()
15
```



Matplotlib (MATrix PLOTting LIBrary) in Spyder

plo4.py*

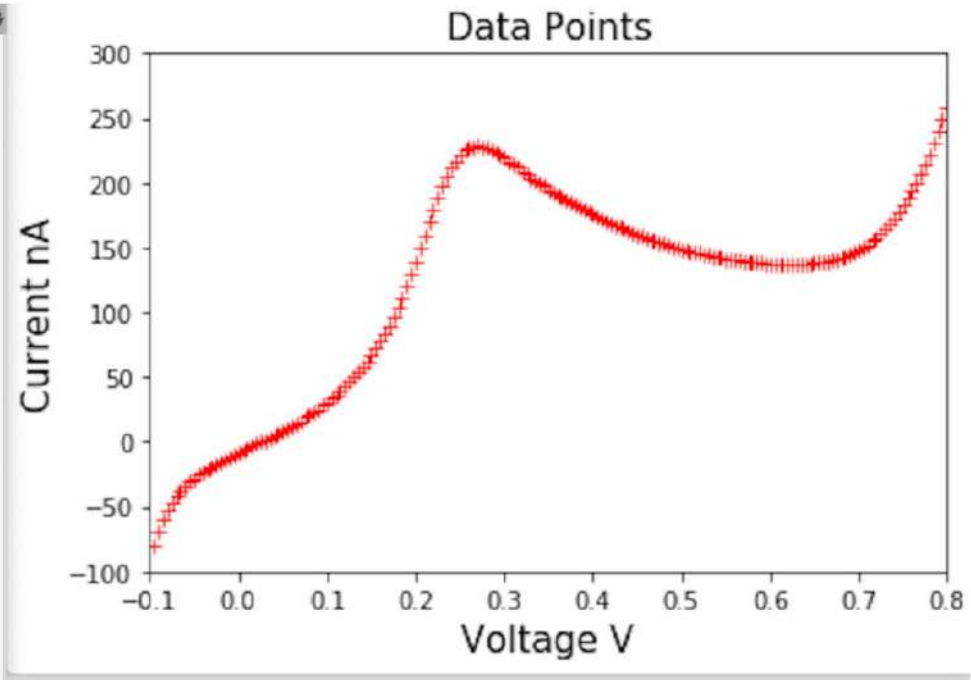
```
1 # Save as plot4.py.
2 # Subplots.
3 import numpy as np
4 import matplotlib.pyplot as plt
5 def f(t):
6     return np.exp(-t) * np.cos(2 * np.pi * t)
7 t = np.arange(0 , 5 , 0.1)
8 plt.figure(1)
9 plt.subplot(211) # subplot(num rows, num cols, fig num)
10 plt.plot(t,f(t),"bo",t,f(t),"k",label="damping")
11 plt.xlabel("Time (s)")
12 plt.ylabel("Amplitude (cm)")
13 plt.title("Damped Pendulum")
14 plt.subplot(212) # subplot(num rows, num cols, fig num)
15 plt.plot(t,np.cos(2*np.pi*t),"g--",linewidth=2)
16 plt.xlabel("Time (s)")
17 plt.ylabel("Amplitude (cm)")
18 plt.title("Undamped Pendulum")
19 plt.subplots_adjust(hspace = 0.8)
20 plt.show()
```



Matplotlib (MATrix PLOTting LIBrary) End Session 4

plot5.py

```
# Save as plot5.py.
# The data file Kam_Data_nA.xlsx must be in the same directory.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
mydata=pd.read_excel("Kam_Data_nA.xlsx",sheet_name="Sheet1")
df = pd.DataFrame(mydata)
xs = (df["Volt"])                  # x values
ys = (df["nAJ"])                   # y values
z = np.polyfit(xs , ys , 20)       # Fit a polynomial of degree 20.
p = np.poly1d(z)
print("polynomial = " , p)
plt.axis([-0.1 , 0.8 , -100 , 300])
plt.xlabel("Voltage V" , fontsize = 15)
plt.ylabel("Current nA" , fontsize = 15)
plt.title("Data Points" , fontsize = 15)
plt.plot(xs , ys , "r+")
plt.show()
```



SymPy: Symbolic Computation, Start Session 5

Python Command Lines	Comments
`In[1]: from sympy import *`	`# Import all functions.`
`In[2]: n , x , y = symbols("n x y")`	`# Declare n, x, y symbolic.`
`In[3]: factor(x**2 - y**2)`	`# Factorize.`
`In[4]: solve(x**2 - 4 * x - 3 , x)`	`# Solve an algebraic equation.`
`In[5]: apart(1 / ((x+2) * (x+1)))`	`# Partial fractions.`
`In[6]: trigsimp(cos(x) - cos(x)**3)`	`# Simplify trig expressions.`
`In[7]: limit(x / sin(x) ,x , 0)`	`# Limits.`
`In[8]: diff(x**2 - 7 * x + 8 , x)`	`# Differentiation.`
`In[9]: diff(5 * x**7 , x , 3)`	`# The third derivative.`
`In[10]: diff(3*x**4*y**7,x,2,y,1)`	`# Partial differentiation.`
`In[11]: (exp(x)*cos(x)).series(x,0,10)`	`# Taylor series expansion.`
`In[12]: integrate(x**4 , x)`	`# Indefinite integration.`
`In[13]: integrate(x**4 , (x,1,3))`	`# Definite integration.`
`In[14]: integrate(1 / x**2, (x,1,oo))`	`# Improper integration.`
`In[15]: summation(1/n**2,(n,1,oo))`	`# Infinite sum.`

SymPy: Symbolic Computation

```
In[16]: solve([x-y,x+2*y-5],[x,y])      # Linear simultaneous equations.
In[17]: solve([x-y,x**2+y**2-1],[x,y]) # Nonlinear simultaneous equations.
In[18]: N(pi , 500)                     # π to 500 decimal places.
In[19]: A = Matrix([[1,-1],[2,3]])      # A 2 × 2 matrix.
In[20]: B = Matrix([[0,2],[3,3]])       # A matrix.
In[21]: 2 * A + 3 * B                   # Matrix algebra.
In[22]: A * B                           # Matrix multiplication.
In[23]: A.row(0)                        # Access row 1 of A.
In[24]: A.col(1)                        # Access column 2 of A.
In[25]: A[0 , 1]                        # The element in row 1, column 2.
In[26]: A.T                             # The transpose matrix.
In[27]: A.inv()                         # The inverse matrix, if it exists.
In[28]: A.det()                         # The determinant.
In[29]: zeros(5,5)                      # A 5 × 5 matrix of zeros.
In[30]: ones(1 , 5)                     # A 1 × 5 matrix of ones.
```

SymPy: Symbolic Computation: End Session 5

```
In[31]: eye(10)                              # The 10 × 10 identity matrix.
In[32]: M = Matrix([[1,-1],[2,4]])           # A 2x2 matrix.
In[33]: M.eigenvals()                        # Eigenvalues of M are {2 : 1, 3 : 1}.
In[34]: z1 , z2 = 3 + 1j , 5 - 4j            # Complex numbers. See notes below.
In[35]: 2 * z1 + z1 * z2                     # Complex algebra.
In[36]: abs(z1)                              # The modulus.
In[37]: re(z1) , im(z1)                      # Real and imaginary parts.
In[38]: arg(z1)                              # The argument.
In[39]: exp(1j*z1).expand(complex=True) # Express in the form x+jy.
In[40]: quit                                 # Restart the kernel.
```

End Day 1 Summary

Day 1			
Using Python as a Calculator	10am-11am	Numpy and Matplotlib	2pm-3pm
Simple Programming	11am-12pm	Sympy – Symbolic Computation	3pm-4pm
Simple Plots using Turtle	12pm-1pm		

Download all files from GitHub:

https://github.com/proflynch/CRC-Press/

Solutions to the Exercises in Section 1:

https://drstephenlynch.github.io/webpages/Solutions_Section_1.html