

1.1.1 Tutorial One: Using Python as a Powerful Calculator

Copy the commands after the chevron prompt `>>>` in the IDLE Editor window (see Figure 1.2). There is no need to copy the comments, they are there to help you. The output is not displayed. New users should type the commands line by line while experienced users can use the tutorial for reference. For help on the `math` module type `>>>import math`, then `>>>help(math)`, in the Python Editor Window (Python Shell).

Python Command Lines

Comments

<code>>>> # This is a comment.</code>	<code># Helps when writing programs.</code>
<code>>>> 2 + 3 - 36 / 2 - 5</code>	<code># Simple arithmetic.</code>
<code>>>> 2**5</code>	<code># Exponentiation.</code>
<code>>>> 2**0.5</code>	<code># Fractional powers.</code>
<code>>>> x = 3</code>	<code># Assign a variable.</code>
<code>>>> x**2 + 1</code>	<code># Work with the variable.</code>
<code>>>> type(x)</code>	<code># x is type (class) integer.</code>
<code>>>> x1 = 4.2; x2 = 2.675;</code>	<code># Assign x1 and x2.</code>
<code>>>> type(x1)</code>	<code># x1 is float.</code>
<code>>>> int(x1)</code>	<code># Truncates to integer.</code>
<code>>>> -7 // 3</code>	<code># Floor division.</code>
<code>>>> round(x2, 2)</code>	<code># Floating point arithmetic.</code>
<code>>>> # Complex Numbers.</code>	
<code>>>> z1=2 + 3j; z2 = 3 - 1j</code>	<code># Assign z1 and z2.</code>
<code>>>> type(z1)</code>	<code># z1 is class complex.</code>
<code>>>> z1**2 - 2 * z2</code>	<code># Complex arithmetic.</code>
<code>>>> abs(z1)</code>	<code># The modulus.</code>
<code>>>> z1.real</code>	<code># The real part of z1.</code>
<code>>>> z1.imag</code>	<code># Imaginary part of z1.</code>
<code>>>> z1.conjugate()</code>	<code># The complex conjugate.</code>
<code>>>> # Lists.</code>	
<code>>>> a=[1, 2, 3, 4, 5]</code>	<code># A list of integers.</code>

```
>>> type(a) # Determine a is a list.
>>> a[0] # 1st element, 0 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 the list 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 found.
>>> print(a) # Prints the list
               a=[1,2,3,4,5].

>>> a[1 : 3] # Slice to get the list [2,3].
>>> range(10) # A range object zero to nine.
>>> list(range(5)) # A list [0,1,2,3,4].
>>> list(range(4, 9)) # A list [4,5,6,7,8].
>>> list(range(2, 10, 2)) # A list [2,4,6,8].
>>> list(range(10, 5, -2)) # A list [10,8,6].
>>> A=[[1, 2], [3, 4]] # A list of lists.
>>> A[0] # The first list [1,2].
>>> A[0][1] # Second element in list 1.
>>> import math # Import all under name
                 space math.

>>> from math import sin # Import sine command only.
>>> from math import * # Import all math commands.
>>> sin(pi) # Sine function.
>>> acos(0) # Inverse cosine.
>>> exp(0.3) # Exponential function.
>>> log10(0.3) # Log base 10.
>>> floor(2.35) # Return floor as integer.
```

1.1.2 Tutorial Two: Simple Programming with Python

Tutorial One demonstrated how one may use the IDLE Editor window as a powerful calculator using Python commands. In this subsection, the reader will be shown how to construct simple Python programs. In the IDLE Editor window, click on the File tab and then New File. An Untitled window opens and the reader types in Python command lines as illustrated in Figure 1.3.

In this section, the author has decided to concentrate on three programming structures: (i) defining functions, (ii) using for and while loops, and (iii) if, elif, else constructs. These three structures are commonly taught to new programmers and readers will see that they are used extensively in this book.

```

# The logistic function - save file as f_mu.py.
# Run the Module (or type F5).
"""
You can write your own text here.
Created on Thur May 24 09:23:47 2018
@author: sladmin
"""
def f_mu(mu, x):
    return mu * x * (1 - x)

```

Ln: 1 Col: 0

Figure 1.3: File Editor window displaying a Python program defining the logistic function.

(i) Defining Functions. Examples 1 and 2 illustrate how functions are defined in Python.

Example 1. Write a Python program that defines the logistic function given by $f_\mu(x) = \mu x(1 - x)$. Once defined and executed, the function gives an extra command within IDLE.

Solution. The first program is shown in Figure 1.3 and defines the logistic function saved as `f_mu.py`. All Python programs should be saved with `.py` at the end of the filename. Note that comments appear as red text, and non-executable author text can be inserted between the triple quotes (green text). The `def` command defines the function, which in this case is a function of two variables, μ and x . At the end of the `def` line one types a colon and then IDLE automatically indents the next line after you type the ENTER key. In the final line one types `return` followed by your choice of function. The program then returns that function.

To run the program, click on Run and Run Module or click the F5 function button. You will see that the program has executed in the IDLE Editor window. One can then call this function in the IDLE Editor window as shown below. The output has also been included.

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

Thus, Python calculates $f_2(0.8)$ as 0.31999999999999995, this is as a result of floating point arithmetic.

Example 2. Write a Python Program that converts degrees Fahrenheit in to Kelvin.

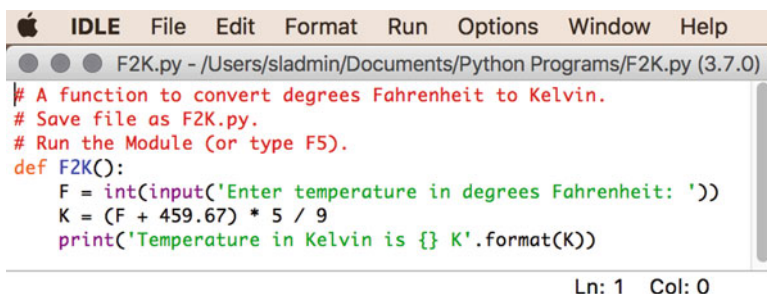


Figure 1.4: File Editor window showing a Python program for converting degrees Fahrenheit into Kelvin, saved as F2K.py.

Solution. The Python program to convert Fahrenheit to Kelvin is shown in Figure 1.4 and the IDLE command line and output is listed below. Run the module before entering the IDLE command line.

```
>>> F2K()
Enter temperature in degrees Fahrenheit: 68
Temperature in Kelvin is: 293.15000000000003 K
```

(ii) **Using Loops.** Examples 3 and 4 illustrate how for and while loops can be used for repetitive tasks.

Example 3. Use the for statement to write a Python Program that lists the first n terms of the Fibonacci sequence.

Solution. The Python program for listing the first n terms of the Fibonacci sequence is shown in Figure 1.5 and the IDLE command line and output is listed below. Run the module before entering the IDLE command line.

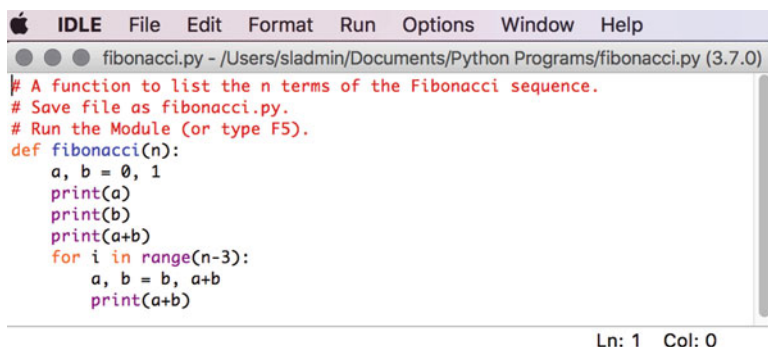


Figure 1.5: File Editor window showing a Python program for listing the first n terms of the Fibonacci sequence.

```
>>> fibonacci(20)
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181
```

Example 4. Use the `while` command to write a Python Program that sums the natural numbers to n .

Solution. The Python program for summing the first n natural numbers is shown in Figure 1.6 and the IDLE command line and output is listed below. Run the module before entering the IDLE command line.

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

(iii) **If, elif, else.** Examples 5 and 6 illustrate how to use conditional statements in programming.

Example 5. Write a Python program that grades students' results.

Solution. A Python program for grading students' results is shown in Figure 1.7 and the IDLE command line and output is listed below. Run the module before entering the IDLE command line.

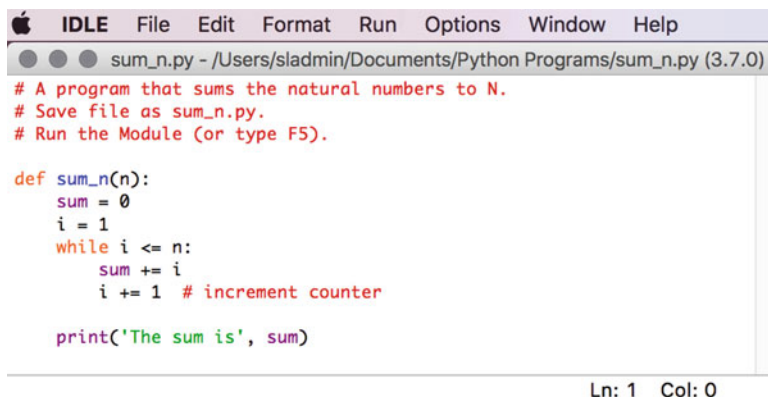


Figure 1.6: File Editor window showing a Python program for listing the sum of the first n natural numbers.

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

Example 6. Write an interactive Python program to play a “guess the number” game. The computer should think of a random integer between 1 and 20 and the user (player) has to try to guess the number within six attempts. The program should let the player know if the guess is too high or too low.

Solution. The Python program for playing the guess the number game is shown in Figure 1.8.

The program for Example 6 is the first program that has imported a module. In order to use Python to study Dynamical Systems more modules and libraries have to be imported as demonstrated in the following sections and subsections.

Figures 1.3–1.8 were screen shots of the IDLE file editor window and the reader should now be familiar with the color command codes used by Python; the remaining program files are listed in the text between horizontal lines and the color coding has been omitted.

1.1.3 Tutorial Three: Simple Plotting Using the Turtle Module

Python comes with the turtle module (turtle.py) already built in and functions within the module enable users to move the turtle around the screen to

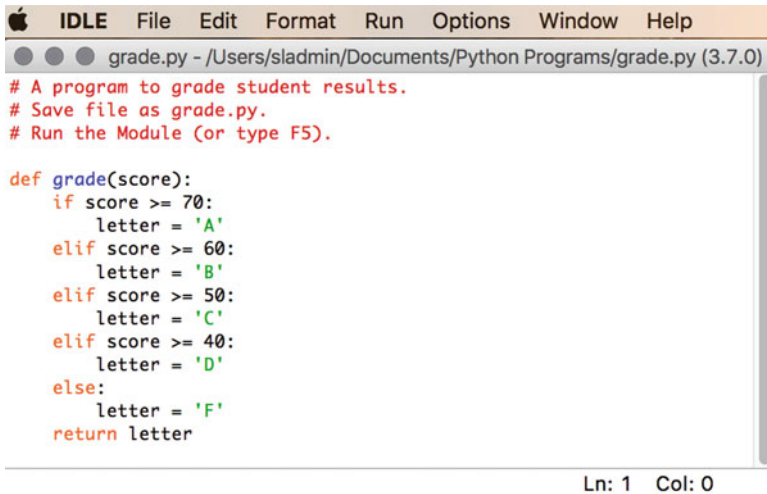


Figure 1.7: File Editor window showing a Python program for grading student results.

create graphics. In order to import the turtle module and its files one simply types `>>> from turtle import *` in the IDLE Editor window. Readers interested in more detail on the Turtle module are directed to the Kindle books [14] and [16].

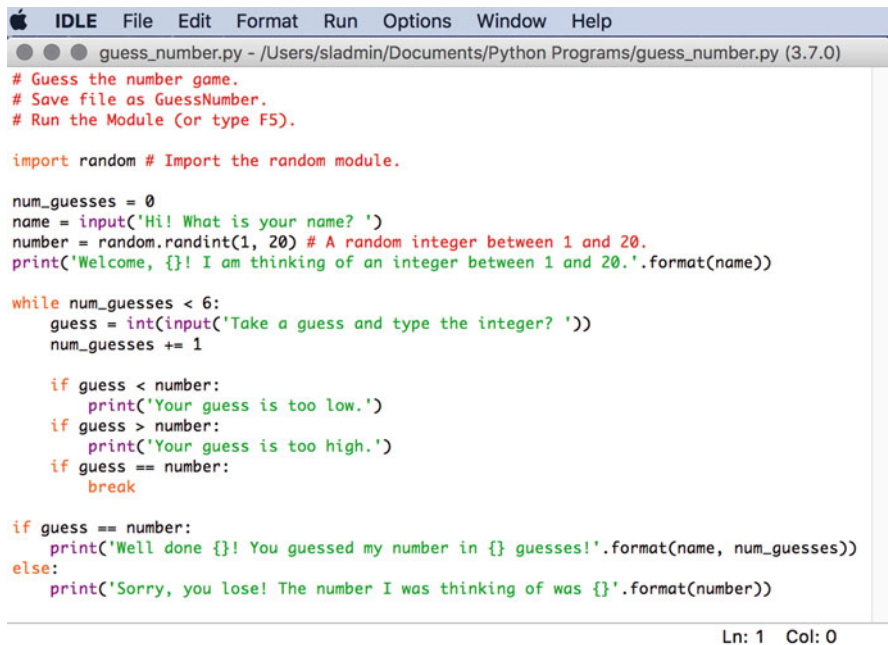
The Turtle module is an excellent tool for plotting fractals as the next three examples demonstrate. Fractals are discussed in more detail in Chapter 17.

Example 7. Write a Python program that plots a fractal tree.

Solution. A Python program for plotting fractal trees is listed below. See Figure 1.9.

```
# A program for plotting fractal trees.
# Save file as fractal_tree_color.py.
# Remember to run the Module (or type F5).
# Run Module and type >>> fractal_tree_color(200,10) in the Python Shell.
from turtle import *
setheading(90)           # The turtle points straight up.
penup()                 # Lift the pen.
setpos(0, -250)          # Set initial point.
pendown()               # Pen down.

def fractal_tree_color(length, level):
    """
    Draws a fractal tree
    """
```



The screenshot shows the IDLE Python IDE interface. The menu bar includes Apple icon, IDLE, File, Edit, Format, Run, Options, Window, and Help. The title bar reads 'guess_number.py - /Users/sladmin/Documents/Python Programs/guess_number.py (3.7.0)'. The editor contains the following Python code:

```
# Guess the number game.
# Save file as GuessNumber.
# Run the Module (or type F5).

import random # Import the random module.

num_guesses = 0
name = input('Hi! What is your name? ')
number = random.randint(1, 20) # A random integer between 1 and 20.
print('Welcome, {}! I am thinking of an integer between 1 and 20.'.format(name))

while num_guesses < 6:
    guess = int(input('Take a guess and type the integer? '))
    num_guesses += 1

    if guess < number:
        print('Your guess is too low.')
    if guess > number:
        print('Your guess is too high.')
    if guess == number:
        break

if guess == number:
    print('Well done {}! You guessed my number in {} guesses!'.format(name, num_guesses))
else:
    print('Sorry, you lose! The number I was thinking of was {}'.format(number))
```

The status bar at the bottom right indicates 'Ln: 1 Col: 0'.

Figure 1.8: File Editor window showing a Python program for playing the guess the number game.

```
pensize(length/10)          # Thickness of lines.
if length < 20:
    pencolor("green")
else:
    pencolor("brown")

speed(0)                    # Fastest speed.
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)                  # Right turn 60 degrees.
    penup()
    bk(length)              # Backward.
    pendown()

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

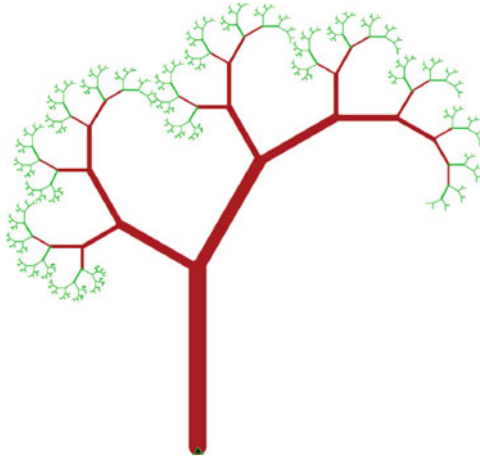



Figure 1.9: [Python] A color fractal tree when length=200 and level=10.

Example 8. Write a Python program that plots a Koch square fractal curve.

Solution. A Python program for plotting a Koch square curve is listed below. See Figure 1.10.

```
# A program for plotting a Koch square curve.
# Save file as koch_square.py.
# Remember to run the Module (or type F5).
from turtle import *
def koch_square(length, level): # Koch square function.
    speed(0) # Fastest speed.
    for i in range(4):
        plot_side(length, level)
        rt(90)

def plot_side(length, level): # Plot side function.
    if level==0:
        fd(length)
        return
    plot_side(length/3, level - 1)
    lt(90)
    plot_side(length/3, level - 1)
    rt(90)
    plot_side(length/3, level - 1)
    rt(90)
    plot_side(length/3, level - 1)
    lt(90)
    plot_side(length/3, level - 1)
```

```
>>> koch_square(200, 4)
```

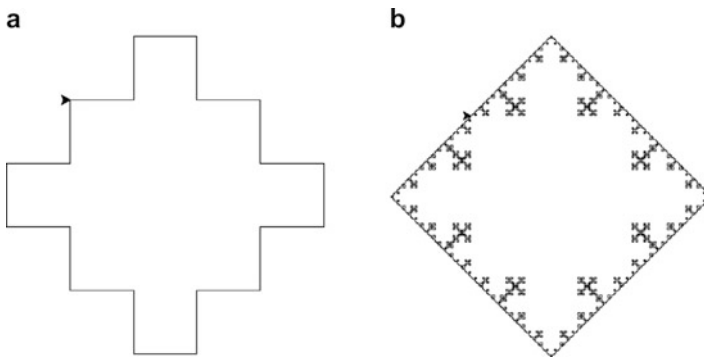


Figure 1.10: [Python] (a) Koch square fractal at level 1; (b) Koch square fractal at level 4. Fractals are discussed in more detail in Chapter 17.

Example 9. Write a Python program that plots a Sierpinski triangle fractal.

Solution. The Python program for plotting a Sierpinski triangle fractal is listed below. See Figure 1.11.

```
# A program that plots the Sierpinski fractal.
# Save file as sierpinski.py.
# Remember to run the Module (or type F5).
from turtle import *
def sierpinski(length, level): # Sierpinski function.
    speed(0)                  # Fastest speed.
    if level==0:
        return
    begin_fill()               # Fill shape.
    color('red')
    for i in range(3):
        sierpinski(length/2, level-1)
        fd(length)
        lt(120)
    end_fill()
```

```
>>> sierpinski(200, 4)
```

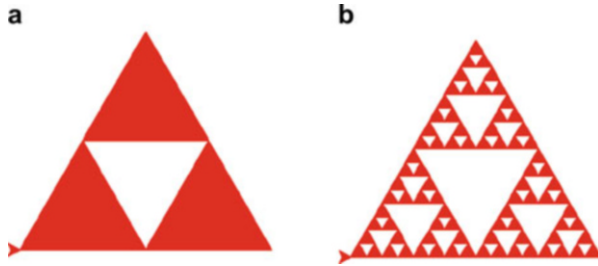


Figure 1.11: [Python] (a) Sierpinski triangle fractal at level 1; (b) Sierpinski triangle fractal at level 4. Fractals are discussed in more detail in Chapter 17.

1.2 Anaconda, Spyder and the Libraries, Sympy, Numpy, and Matplotlib

The first section introduces the reader to Python using the IDLE editor; however, in order to perform scientific computing and computational modeling additional libraries (or packages) that are not part of the Python standard library are required. The additional libraries required for this book include sympy (SYMbolic PYthon) for symbolic computation, numpy (NUMeric PYthon) for numerical routines, and matplotlib (PLOTting LIBrary) for creating plots. Python has a number of interpreters along with packages and editors and the author has found that the Anaconda free package manager, environment manager and Python distribution is one of the best for dynamical systems work. Readers may also be interested in alternatives to Anaconda such as WinPython and Enthought Canopy. The Anaconda Python distribution is available for download for Windows, Mac OS, and the Linux operating systems. The current URL to download Anaconda is at:

<https://www.continuum.io/downloads>

Readers should click on the Anaconda Navigator icon and a window opens as displayed in Figure 1.12.

By clicking on the Launch button under the Spyder icon (see Figure 1.12) an Integrated Development Environment (IDE) or notebook opens as displayed in Figure 1.13. The three windows are described below:

1. The editor window is used to write code and save to file.
2. The variable/file explorer window can display detailed help on variables or files and is useful when debugging.
3. The console window is where one can work interactively and where output results and error messages are displayed. The console can be used in the same way as the IDLE editor window - see Subsection 1.1.1.

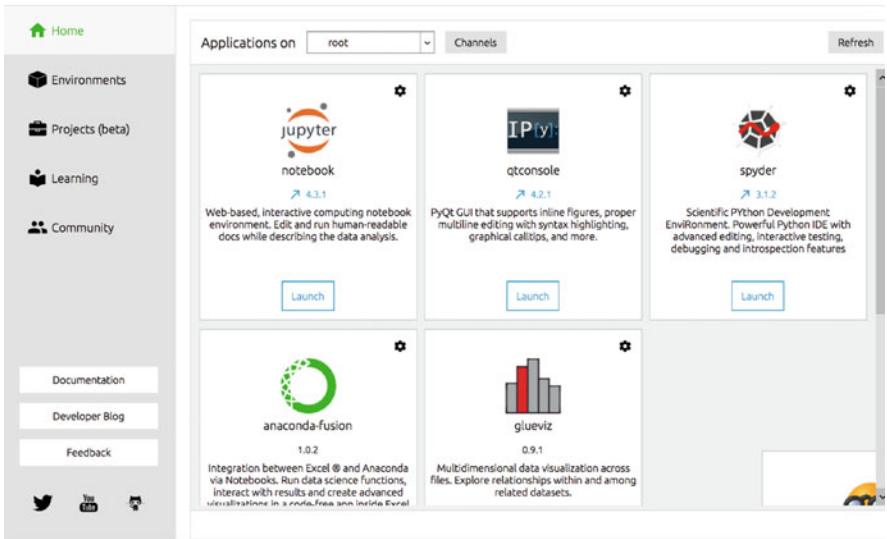


Figure 1.12: The Anaconda Navigator window on a Mac OS.

Note that when saving a file and running the code the file should be located in the working directory indicated in the top right corner of the IDE.

1.2.1 Tutorial One: A Tutorial Introduction to SymPy

Sympy is a computer algebra system and a Python library for symbolic mathematics written entirely in Python. The following tutorial has been designed to allow new users to become familiar with the commands by means of example. For more detailed information please refer to sympy’s document pages at:

<http://docs.sympy.org/latest/index.html>

The following command lines should be typed in the console window. There is no need to copy the comments, they are there to help you.

Python Commands	Comments
<code>In[1]: 2 / 3 + 4 / 5</code>	<code># Approximate decimal.</code>
<code>In[2]: from fractions import Fraction</code>	<code># To work with fractions.</code>
<code>In[3]: Fraction(2, 3)+Fraction(4, 5)</code>	<code># Symbolic answer.</code>
<code>In[4]: sqrt(16)</code>	<code># Square root.</code>
<code>In[5]: sin(pi)</code>	<code># Trigonometric function.</code>

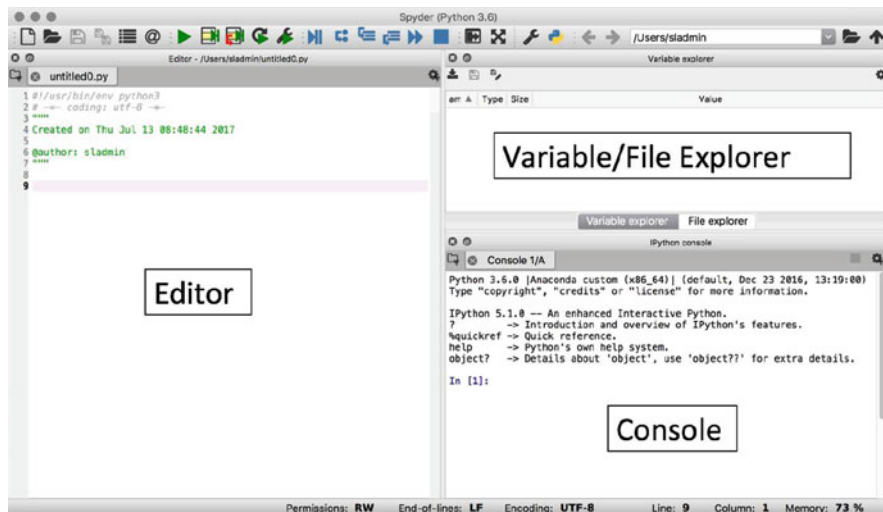


Figure 1.13: A Spyder IDE (notebook) showing the Editor window, the Variable/File Explorer window, and the Console window.

```
In[6]: from sympy import *                # Import everything from sympy
                                           # library into global scope.

In[7]: x,y=symbols('x y')                # Declare x and y symbolic.

In[8]: factor(x**3 - y**3)                # Factorize.

In[9]: expand(Out[8])                     # Expand the last result.

In[10]: factor((x**3 - y**3)/(x - y))     # Simplify an expression.

In[11]: apart(1/((x + 2)*(x + 1)))        # Partial fractions.

In[12]: trigsimp(cos(x) - cos(x)**3)      # Simplify a trig expression.

In[13]: limit(1/x, x, oo)                 # Limits.

In[14]: diff(x**2 - 3*x + 6,x)            # Total differentiation.

In[15]: diff(x**3*y**5, x, y, 3)          # Partial differentiation.

In[16]: integrate(sin(x)*cos(x),x)        # Indefinite integration.

In[17]: integrate(exp(-x**2 - y**2),     # Definite integration.
                  (x, 0, oo),(y, 0, oo))

In[18]: (exp(x)*cos(x)).series(x,0,10)    # Taylor series expansion.

In[19]: summation(1 / x**2,(x,1,oo))      # An infinite sum.
```

```
In[20]: solve(x**5 - 1, x)           # Solving equations. Roots
In[21]: solve([x+5*y-2, -3*x+6*y-15], # Solving simultaneous equations.
            [x, y])
In[22]: z1 = 3 + 1*I; z2 = 5 - 4*I   # Note that 1j=I can be used too.
In[23]: 2 * z1 + z1 * z2             # Simple complex arithmetic.
In[24]: conjugate(z1)               # Complex conjugate.
In[25]: arg(z1)                    # The argument of z1.
In[26]: abs(z1)                   # The modulus of z1.
In[27]: re(z1)                    # The real part.
In[28]: im(z1)                    # The imaginary part.
In[29]: exp(1*z1).expand(complex=True) # Express in form x+iy.
In[30]: A=Matrix([[1, -1], [2, 3]]); # Two 2x2 matrices.
        B=Matrix([[0, 2], [3, 3]]);
In[31]: 2 * A + 3 * A * B          # Matrix algebra.
In[32]: A.row(0)                   # Access the first row.
In[33]: A.T                       # The transpose of a matrix.
In[34]: A.T.row(1)                 # Access the second column of A.
In[35]: A[0, 1]                   # The element in row 1, column 2.
In[36]: A**(-1)                   # The inverse of a matrix.
In[37]: A**5                      # The power of a matrix.
In[38]: A.det()                   # The determinant of A.
In[39]: zeros(3, 3)               # A 3x3 matrix of zeros.
In[40]: ones(1, 5)                # A 1x5 matrix of ones.
In[41]: C=Matrix([[2,1,0], [-1,4,0], # A 3x3 matrix.
                [-1,3,1]])
In[42]: C.rref()                  # The row reduced echelon form.
In[43]: C.eigenvals()             # The eigenvalues of C.
In[44]: C.eigenvects()           # The eigenvectors of C.
In[45]: s, t, w = symbols('s t w') # Declare s,t,w symbolic.
In[46]: laplace_transform(t**3,t,s) # Laplace transform.
In[47]: inverse_laplace_transform  # Inverse transform.
        (6/s**4, s, t)
```

```

In[48]: fourier_transform(-2 * pi *      # Fourier transform.
        abs(t), t, w)
In[49]: inverse_fourier_transform(1/     # Inverse transform.
        (pi * w**2), w, t)
In[50]: quit                            # Quits IPython console.

```

1.2.2 Tutorial Two: A Tutorial Introduction to Numpy and Matplotlib

Numpy's main object is the homogeneous multidimensional array and allows Python to compute with vectors and matrices. Matplotlib is a Python 2-dimensional plotting library used to generate bar charts, error charts, histograms, plots, power spectra, and scatterplots, for example. When combining the pyplot module with IPython a MATLAB-like interface is provided and the user can control axes properties, font properties, and line styles via an object oriented interface. For a more detailed reference guide to numpy, readers are directed to:

<https://docs.scipy.org/doc/numpy/reference/>

and an introduction to matplotlib is given here:

<https://matplotlib.org>

The following command lines provide a concise introduction to numpy and matplotlib by means of example. The following command lines should be typed in the IPython console window. There is no need to copy the comments, they are there to help you.

Python Commands

Comments

In[1]: import numpy as np	# Import numpy into the np # namespace.
In[2]: a = np.arange(5)	# A 1d array [0 1 2 3 4].
In[3]: b = np.arange(6).reshape(2,3)	# A 2d array [[0 1 2],[3 4 5]].
In[4]: A = np.array([[1, 1], [0, 1]])	# A 2d array.
In[5]: B = np.array([[2, 0], [3, 4]])	# A 2d array.
In[6]: A * B	# Elementwise product [[2 0], # [0 4]].
In[7]: A.dot(B)	# Matrix product [[5 4],[3 4]].

```
In[8]: np.dot(A, B)                # Matrix product [[5 4],[3 4]].
In[9]: c=np.arange(12).reshape(3, 4)  # A 2d array.
In[10]: c.sum(axis = 0)              # Sum each column.
In[11]: c.min(axis = 1)              # Minimum of each row.
In[12]: c.cumsum(axis = 1)           # Cumulative sum for each row.
In[13]: np.linspace(0, 2, 5)         # Gives array([ 0.,0.5,1.,
                                     # 1.5,2.]).

In[14]: # Simple plots with Python and matplotlib
In[15]: from numpy import *          # Import numpy.
In[16]: import matplotlib.pyplot as plt # Import pyplot.
In[17]: x = np.linspace(-2, 2, 50)   # Set up the domain.
In[18]: y = x**2                     # A function of x.
In[19]: plt.plot(x, y)               # A basic plot.
In[20]: # Two plots on one graph
In[21]: t = np.linspace(0, 100, 1000); # Set the domain.
        p1 = np.exp(-.1*t) * np.cos(t); # Two functions.
        p2 = np.cos(t);
        plt.plot(t, p1);plt.plot(t, p2); # Plot two curves.

In[22]: # Plot with a legend
In[23]: plt.plot(t, p1, label = 'p1'); plt.plot(t, p2, label = 'p2');
        plt.legend();

In[24]: # Plot with labels and title
In[25]: plt.xlabel('x');plt.ylabel('y');plt.title('y=x^2');
        plt.plot(x,y)

In[26]: # Change line width and colour
In[27]: plt.plot(x,y,color = 'green', linewidth = 4)

In[28]: # Plotting implicit functions
In[29]: x,y=np.mgrid[-5:5:100j,-5:5:100j] # A grid of x,y values.
        z = x**2 / 4 + y**2                # A function of two variables.
        plt.contour(x,y,z,levels=[1])      # The curve x**2/4+y**2=1.

In[30]: # A parametric plot
In[31]: t=np.linspace(0, np.pi, 100);    # A set of t values.
        x = 0.7*np.sin(t+1)*np.sin(3*t); # A set of x values.
        y = 0.7*np.cos(t+1)*np.sin(3*t); # A set of y values.
        plt.plot(x, y)                    # The 2D parametric plot.

In[32]: quit                             # Quits IPython console.
```


1.2.3 Tutorial Three: Simple Programming, Solving ODEs, and More Detailed Plots

In order to solve ordinary differential equations (ODEs) and produce more detailed plots the reader is advised to write short programs rather than using the console window. Examples are listed below, where each file is listed between horizontal lines and the output is also included in the indicated figures.

Readers can get help from within the Python console using the help command. For example, by typing `>>>help(dsolve)`, information and examples are listed in the console.

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.

Example 10. Write a Python program that solves the ODE: $\frac{dx}{dt} + x = 1$.

Solution. The Python program for solving the ODE is listed below.

```
# Program 01a: A program that solves a simple ODE.
from sympy import dsolve, Eq, symbols, Function
t = symbols('t')
x = symbols('x', cls=Function)
deqn1 = Eq(x(t).diff(t), 1 - x(t))
sol1 = dsolve(deqn1, x(t))
print(sol1)
```

$\text{Eq}(x(t), C1*\exp(-t) + 1)$

Example 11. Write a Python program that solves the ODE: $\frac{d^2y}{dt^2} + \frac{dy}{dt} + y = e^t$.

Solution. The Python program for solving the second order ODE is listed below.

```
# Program 01b: A program that solves a second order ODE.
from sympy import dsolve, Eq, exp, Function, symbols
t = symbols('t')
y = symbols('y', cls=Function)
deqn2 = Eq(y(t).diff(t,t) + y(t).diff(t) + y(t), exp(t))
sol2 = dsolve(deqn2, y(t))
print(sol2)
```

$\text{Eq}(y(t), (C1*\sin(\sqrt{3}*t/2) + C2*\cos(\sqrt{3}*t/2))/\sqrt{\exp(t)} + \exp(t)/3)$

Example 12. Write a Python program that plots two curves on one graph.

Solution. The Python program for plotting Figure 1.14 is listed below.

```
# Program 01c: A program that plots two curves on one graph.
# Remember to run the Module (or type F5).
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0.0, 2.0, 0.01)
c = 1 + np.cos(2*np.pi*t)
s = 1 + np.sin(2*np.pi*t)

plt.plot(t, s, 'r--', t, c, 'b-.')
plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('Voltage-time plot')
plt.grid(True)
plt.savefig("Voltage-Time Plot.png")
plt.show()
```

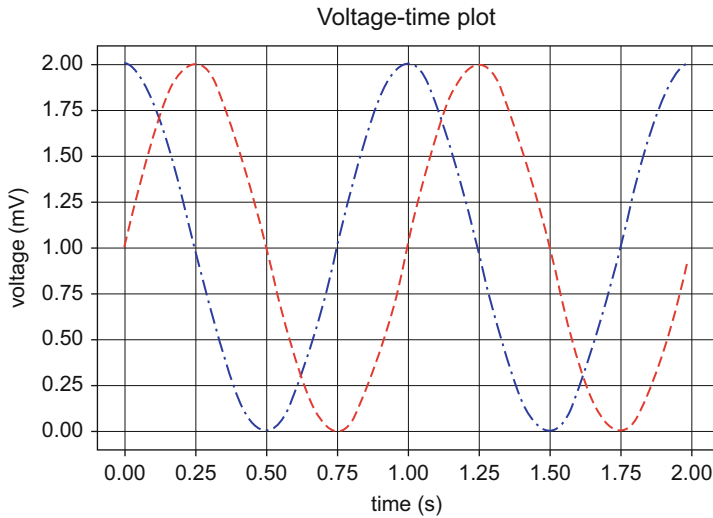


Figure 1.14: [Python] A voltage time plot. Note that 'r-' gives a red dashed curve and 'b-' gives a blue dash-dot curve. Using savefig, the figure is saved in the same folder where the python program is stored.

Example 13. Write a Python program that plots subplots.

Solution. The Python program for plotting Figure 1.15 is listed below. Note that the syntax for the subplot command is subplot(number of rows, number of columns, figure number).

```
# Program 01d: A program that plots subplots.
# Remember to run the Module (or type F5).
import matplotlib.pyplot as plt
import numpy as np

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211) #subplot(num rows,num cols,fig num)
plt.plot(t1,f(t1), 'bo', t2, f(t2), 'k', label='damping')
plt.xlabel('time (s)')
plt.ylabel('amplitude (m)')
plt.title('Damped pendulum')
legend = plt.legend(loc='upper center', shadow=True)
```

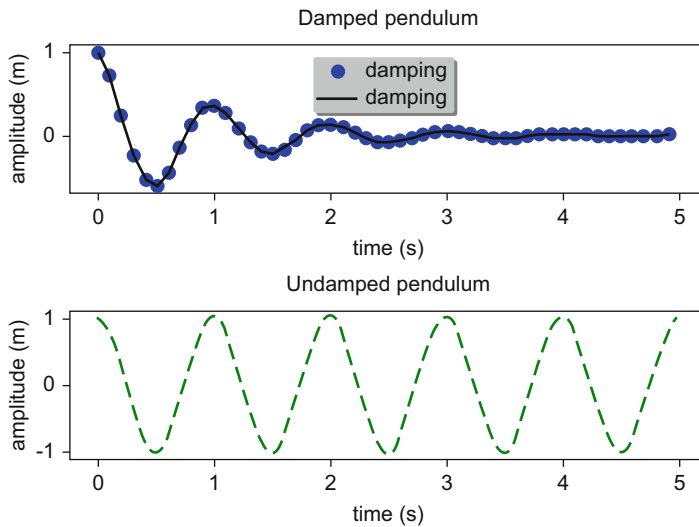


Figure 1.15: [Python] Two subplots for a damped and undamped pendulum. The upper plot also has a figure legend.

```
plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'g--', linewidth=4)
plt.xlabel('time (s)')
plt.ylabel('amplitude (m)')
plt.title('Undamped pendulum')
plt.subplots_adjust(hspace=0.8)
plt.show()
```

Example 14. Write a Python program that plots a surface and corresponding contour plots in 3D.

Solution. The Python program for plotting Figure 1.16 is listed below.

```
# Program 01e: A program that plots a surface and contour plots in 3D.
# Remember to run the Module (or type F5).
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

alpha = 0.7
phi_ext = 2 * np.pi * 0.5
```

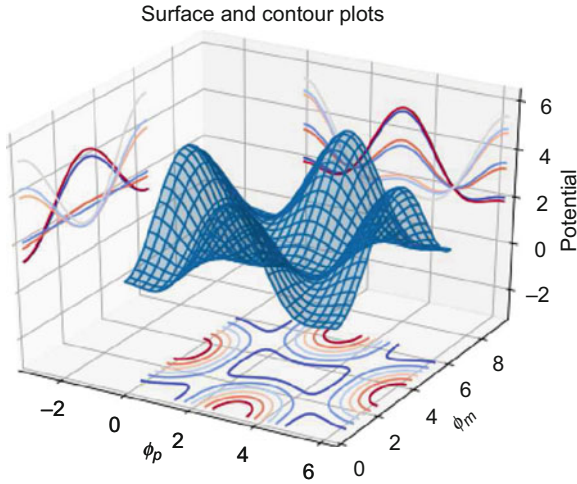


Figure 1.16: [Python] A surface and contour plot. Note that the font size of ticks and axis labels have also been set. In this case the axis labels are generated with LaTeX code.

```
def flux_qubit_potential(phi_m, phi_p):
    return 2*alpha-2*np.cos(phi_p)*np.cos(phi_m)-alpha*np.cos(
        phi_ext-2*phi_p)

phi_m = np.linspace(0, 2 * np.pi, 100)
phi_p = np.linspace(0, 2 * np.pi, 100)
X,Y = np.meshgrid(phi_p, phi_m)
Z = flux_qubit_potential(X, Y).T

fig = plt.figure(figsize = (8, 6))

ax=fig.add_subplot(1, 1, 1, projection='3d')
p=ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4)
ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
cset=ax.contour(X,Y,Z,zdir='z', offset=-np.pi, cmap=plt.cm.coolwarm)
cset=ax.contour(X,Y,Z,zdir='x', offset=-np.pi, cmap=plt.cm.coolwarm)
cset=ax.contour(X,Y,Z,zdir='y', offset=3*np.pi, cmap=plt.cm.coolwarm)

ax.set_xlim3d(-np.pi, 2*np.pi);
ax.set_ylim3d(0, 3*np.pi);
ax.set_zlim3d(-np.pi, 2*np.pi);
ax.set_xlabel('$\phi_p$', fontsize=15)
ax.set_ylabel('$\phi_m$', fontsize=15)
```

```
ax.set_zlabel('Potential', fontsize=15)
plt.tick_params(labels=15)
ax.set_title("Surface and contour plots",font=15)
plt.show()
```

Example 15. Write a Python program that plots a parametric plot in 3D.

Solution. The Python program for plotting Figure 1.17 is listed below.

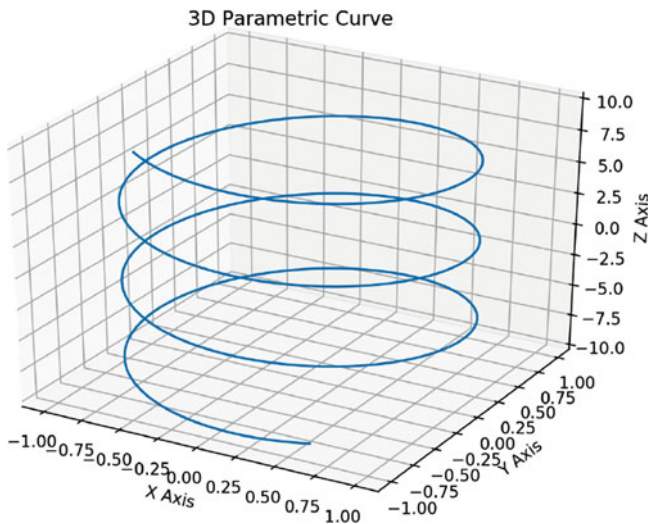


Figure 1.17: [Python] A parametric plot in 3D.

```
# Program 01f: A program that plots a parametric curve in 3D.
# Remember to run the Module (or type F5).
import numpy as np
import matplotlib.pyplot as plt
#from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(1, 1, 1, projection='3d')
t = np.linspace(-10, 10, 1000)
x = np.sin(t)
y = np.cos(t)
z = t
ax.plot(x, y, z)
ax.set_xlabel("X Axis")
```

```

ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("3D Parametric Curve")
plt.show()

```

Example 16. Write a Python program that animates a simple curve.

Solution. The Python program for producing an animation is listed below.

```

# Program 01g: A program that animates a simple curve.
# Remember to run the Module (or type F5).
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation
fig = plt.figure()
ax = plt.axes(xlim=(0, 2), ylim=(-2, 2))
line, = ax.plot([], [], lw=2)
plt.xlabel('time')
plt.ylabel('sin( $\omega t$ )')

def init():
    line.set_data([], [])
    return line,

# The function to animate.
def animate(i):
    x = np.linspace(0, 2, 1000)
    y = np.sin(2 * np.pi * (0.1 * x * i))
    line.set_data(x, y)
    return line,

# Note: blit=True means only re-draw the parts that have changed.
anim=animation.FuncAnimation(fig, animate, init_func=init, \
                             frames=100, interval=200, blit=True)

plt.show()

```

Readers may be interested in my other Dynamical Systems books based on Mathematica, MATLAB, and Maple, [7, 8, 9], where introductory chapters provide tutorial guides to those packages.