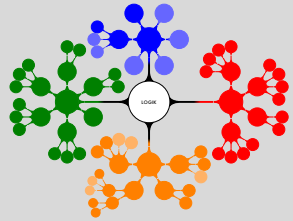
Lecture 2

Introduction to Python

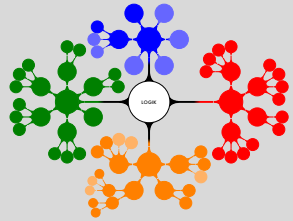
Scientific Packages

IKC-MH.55 *Scientific Computing with Python* at October 20, 2023

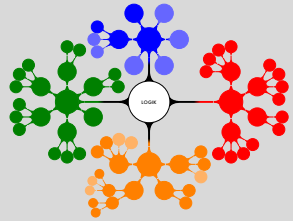
Dr. Cem Özdoğan
Engineering Sciences Department
İzmir Kâtip Çelebi University



Contents



Dr. Cem Özdoğan



Introduction to Python I

- Running a Computer Program.

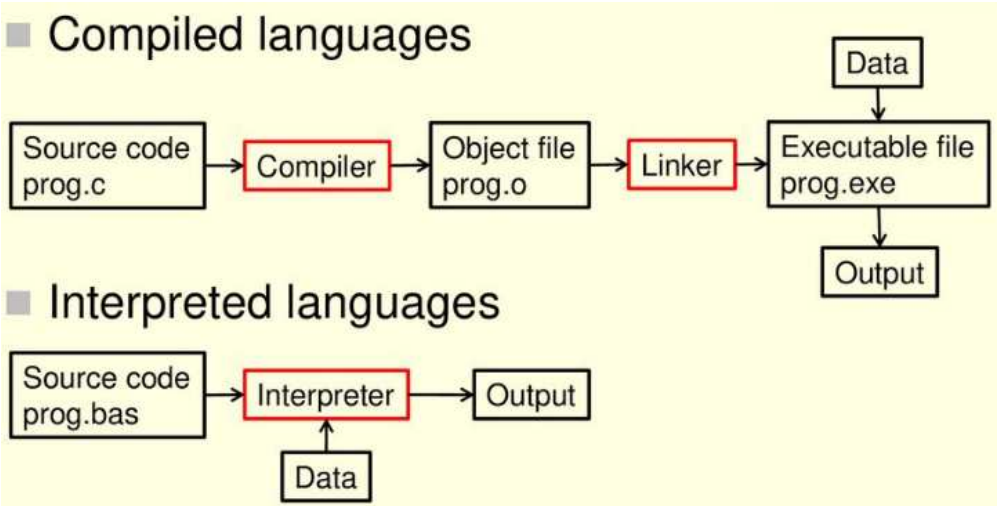
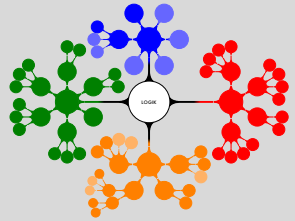


Figure: Running a Computer Program.

- Python is an interpreted language.
  - The code is pre-processed to produce a bytecode (similar to machine language) and then executed by the interpreter (virtual machine).
- **Code portability:** Runs on hardware/software platforms different from which used to develop the code.
  - Python is portable if the interpreter is available on the target platform.



Introduction to Python II

- **Variables:**

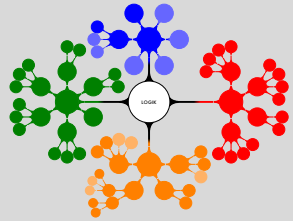
A variable stores a piece of data and gives it a name.

• a variable name must start with a letter or the underscore character;
• a variable name cannot start with a number;
• a variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _);
• white spaces and signs with special meanings, as "+" and "-" are not allowed;
• variable names are case-sensitive (age, Age and AGE are three different variables).

- **Built-in data types:**

Built-in data types

• **Text Type:** str • **Numeric Types:** int, float, complex
• **Sequence Types:** list, tuple, range • **Mapping Type:** dict
• **Set Types:** set, frozenset • **Boolean Type:** bool
• **Binary Types:** bytes, bytearray, memoryview



Introduction to Python III

Variables:

A variable stores a piece of data and gives it a name

Lists:

What if we want to store many integers? We need a list!

Loops:

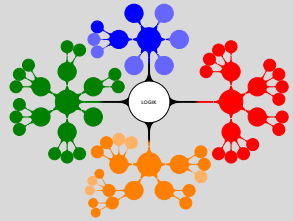
Repeat code until a conditional statement ends the loop.

Conditionals:

Sometimes you want to execute code only in certain circumstances.

Functions:

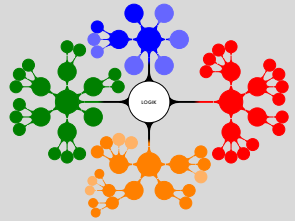
We can separate off code into functions, that can take input and can give output. They serve as black boxes from the perspective of the rest of our code.



Python Libraries for Data Science I

Extensive first and third party libraries. Top Python Libraries for Data Science.

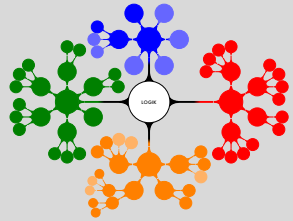
- **NumPy** (aka Numerical Python) is the core numeric and scientific computation library in Python. General-purpose array-processing package.
- **SciPy** (aka Scientific Python) is extensively used for scientific and technical computations (extends NumPy).
- **Matplotlib** is an essential library in Python for data visualization in data science. A plotting library.
- **Seaborn** is another library in Python for data visualization. Extension of Matplotlib. Statistical and graphical analysis in data science.
- **Pandas** (Python data analysis) is a foundational Python library for data analysis in data science. Data cleaning, data handling, manipulation, and modeling.



Python Libraries for Data Science II

Top Python Libraries for Data Science.

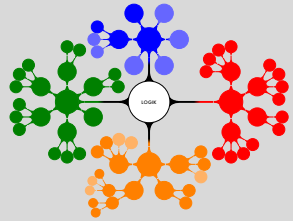
- **SciKit-Learn** is a robust machine learning library in Python. Data mining, feature engineering, training and deploying machine learning models.
- **Statsmodels** - provides functionalities for descriptive and inferential statistics for statistical models.
- **TensorFlow** - a framework for defining and running computations that involve tensors. Machine learning and deep learning framework.
- **Keras** is a neural network Python library for deep learning model development, training, and deployment.
- **PyTorch** - scientific computing package that uses the power of graphics processing units.
- **Scrapy** - for web crawling frameworks.
- **BeautifulSoup** - for web crawling and data scraping.
- **NLTK** (Natural Language Tool Kit) is a Python package essentially for natural language processing.



NumPy

Numpy (Numerical Python) - The Fundamental Package for Scientific Computing with Python. https://numpy.org/

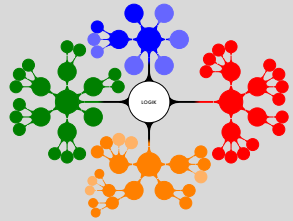
- NumPy offers high-quality mathematical functions and supports logical operations on built-in *multi-dimensional array objects*.
- NumPy arrays are significantly faster than traditional Python lists and way more efficient in performance.
- Some of the features provided by NumPy
  - Basic array operations such as addition and multiplication
  - Mathematical, logical, shape manipulation operations
  - Indexing, slicing, flattening, and reshaping the arrays
  - Stacking, splitting, and broadcasting arrays
  - I/O Operations
  - Fourier transform capabilities
  - Basic linear algebra
  - Basic statistical operations
  - Random number generation
  - . . .



NumPy Module Organization

Sub-Packages	Purpose	Comments
core	basic objects	all names exported to numpy
lib	Addintional utilities	all names exported to numpy
linalg	Basic linear algebra	LinearAlgebra derived from Numeric
fft	Discrete Fourier transforms	FFT derived from Numeric
random	Random number generators	RandomArray derived from Numeric
distutils	Enhanced build and distribution	improvements built on standard distutils
testing	unit-testing	utility functions useful for testing
f2py	Automatic wrapping of Fortran code	a useful utility needed by SciPy

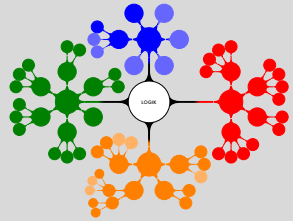
Figure: NumPy Module Organization.



SciPy

SciPy is a scientific computation library in Python. A collection of mathematical functions and algorithms built on Python's extension NumPy https://scipy.org/.

- It provides the user with high-level commands and classes for manipulating and visualizing data.
- It is widely used in machine learning and scientific programming and comes with integrated support for linear algebra and statistics.
- Some of the features provided by SciPy
  - Search for minima and maxima of functions
  - Calculation of function integrals
  - Support for special functions
  - Signal processing
  - Multi-dimensional image processing
  - Work with genetic algorithms
  - Fourier transform capabilities
  - Solving ordinary differential equations
  - . . .
- The scipy namespace itself only contains functions imported from numpy.

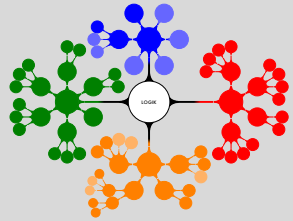


SciPy Modules

Therefore, importing only the scipy base package does only provide numpy content, which could be imported from numpy directly (NOT USED as *import scipy*).
i.e., from scipy import linalg, io

Subpackage	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distribution and function

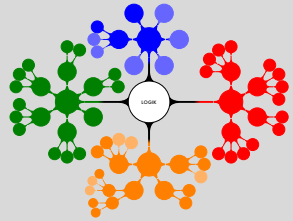
Figure: SciPy Modules.



Matplotlib

Matplotlib is the core plotting and data visualization package in Python https://matplotlib.org/.

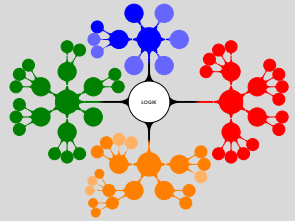
- A 2D graphical Python library which produces publication quality figures. However, it also supports 3D graphics (mplot3d toolkit), but this is very limited.
- Matplotlib is capable of producing high-quality figures in various formats. It offers interactive cross-platform environments for plotting.
- It provides a MATLAB/Mathematica-like interface for simple plotting pyplot submodule with secondary x-y axis support, and facilitates the creation of subplots, labels, grids, legends, use a logarithmic scale or polar coordinates etc.
  - Matplotlib also allows full control of axes properties, font styles, line and marker styles, and some more formatting entities.
- You can generate line plots (Charts), bar charts, histograms, power spectra, pie charts, error charts, box plots, scatter plots, stem plots, contour plots, etc., with just a few lines of codes in Matplotlib.



Programming in Python I

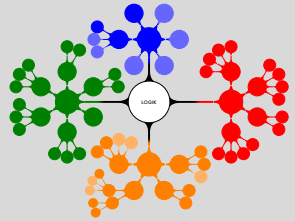
week2_HandsOn.py:

```
#!/usr/bin/python3
######### Variables #########
# Each variable in python has a "type". The variable type is not
    pre-defined, it is "DYNAMICALLY" resolved at run-time
answer = 42   # "answer" contained an integer because we gave it
print(answer) # as an integer!
# 42
is_it_thursday = True   # These both are 'booleans' or true/false
is_it_wednesday = False #  values
pi_approx = 3.1415   # This will be a floating point number
my_string = "Value of pi number" # This is a string datatype
print(pi_approx, my_string)
# 3.1415 Value of pi number
print ("my_string[0]: ", my_string[0]) # Access substrings use []
# my_string[0]:  V
print ("my_string[1:5]: ", my_string[1:5]) # or indices
# my_string[1:5]:  alue
# print(pi_approx + my_string)
## TypeError: unsupported operand type(s) for +: 'float' and 'str'
print(my_string + " in four digits after .")
# Value of pi number in four digits after .
print(type(pi_approx)) # You can get the data type of any object
# <class 'float'>
# Addition, subtraction, multiplication, division are as you
    expect
float1 = 5.75; float2 = 2.25
print(float1 + float2); print(float1 - float2); print(float1 *
    float2); print(float1 / float2)
print(5 % 2) # Modulus
```

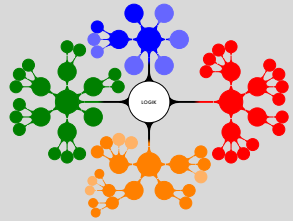
Programming in Python II

```
## More Complicated Data Types
# LIST. What if we want to store many integers? We need a list.
prices = [10, 20, 30, 40, 50] # A way to define a list in place
colors = [] #  We can also make an empty list and add to it
colors.append("Green")
colors.append("Blue")
colors.append("Red")
print(colors)
prices.append("Sixty") # We can also add unlike data to a list
print(prices) # Items in a list can be of different type
print(colors[0]) # Single list elements can be accessed
print(colors[2]) # with the operator [ ]
ourlist = [1, 2, 3, 4, 5] # Basic List Operations
print(ourlist+ourlist) # Concatenation
print(3*ourlist)       # Repetition
multiplied_ourlist = [value * 3 for value in ourlist] # Membership
print(multiplied_ourlist)  # Iteration
#
# TUPLE. A tuple is a sequence ordered data enclosed between ().
tuple1 = "a", "b", "c", "d"
tuple2 = ('physics', 'chemistry', 2022, 2023) # Data heterogeneous
tuple3 = (1, 2, 3, 4, 5 )
ikc_muh_55_info = ("IKC-MH", "55", 2023, "February", 28)
print("tuple1[0]: ", tuple1[0])     # access to single element
print("tuple2[1:5]: ", tuple2[1:5]) # access to slice
print(ikc_muh_55_info[0] + "." + ikc_muh_55_info[1])
print(tuple3)
```

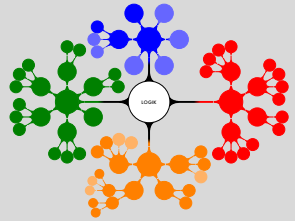
Programming in Python III

```
# DICTIONARY. An unordered collection of items
person = {"name": "Mehmet", "age": 19}
print(f"{person['name']} is {person['age']} years old.")
print(person["name"])
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
    print(squares[i])
#
######### Loops in Python #########
# Repeat code until a conditional statement ends the loop
# WHILE.
list = [1, 1, 2, 3, 5, 8]
print(list)
print("i", "list[i]")
i = 0
while (i < len(list)): # While loops are the basic type
    print(i, list[i])
    i = i + 1
#
# FOR. The 'for' loop is the way to write it faster.
for i in range(0, len(list)):
    print(i, list[i])
# Or you can do so even neater
for e in list:
    print(e)
```

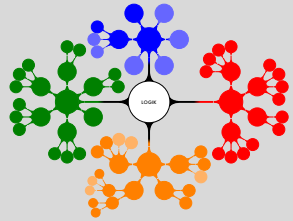
Programming in Python IV

```
######### Conditionals in Python #########
# Sometimes you want to execute code only in certain circumstances
answer = 42 # Change answer and see what code is executed
if answer == 42:
    print('This is the answer to the ultimate question')
elif answer < 42:
    print('This is less than the answer to the ultimate question')
else:
    print('This is more than the answer to the ultimate question')
print('This print statement is run no matter what because it is
    not indented!')
# Using boolean operations. Question: How long does it take me to
    get to work?
snowy = True
day = "Monday"
rainy = True
if (snowy == False) and (day != "Monday"): # "and" is boolean and.
     True only if both are true. False otherwise
    time = 7
elif (snowy == True) and (day == "Monday"):
    time = 11
elif (rainy == True) or (day == "Monday"):
    time = 9
print("It takes me %d minutes" % (time))
```

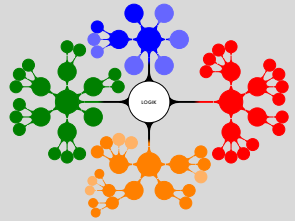
Programming in Python V

```
# while & if statements example
number = 23
running = True
while running:
    guess = int(input('Enter an integer : '))
    if guess == number:
        print('Congratulations, you guessed it.')
        running = False # this causes the while loop to stop
    elif guess < number:
        print('No, it is a little higher than that.')
    else:
        print('No, it is a little lower than that.')
else:
    print('The while loop is over.')
print('Done') # Do anything else you want to do here
#
######### Functions in Python #########
# A function is a block of code which only runs when it is called
    and can be run repetitively.
# use the def keyword, and indent because this creates a new block
def print_me(string): # Function definition is here
    "This prints a passed string into this function" # The first
    statement of a function can be an optional statement
    print(string)
    return      # End with the "return" keyword
print_me("I'm first call to user defined function!") # Function
    call
print_me("Again second call to the same function") # Function call
```

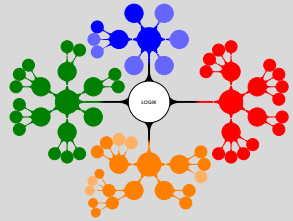
Programming in Python VI

```
1  def changelist( mylist ):
2      """This changes a passed list into this function"""
3      mylist = [1,20,3,4] # Call by  Value in Python. This assigns
        new reference in mylist
4                          # Call by Reference when commenting this line
5      print("Address inside the function: ", id(mylist))
6      mylist.append(9)
7      mylist[0]=7
8      mylist.remove(20)
9      print("Values inside the function: ", mylist)
10 mylist = [10,20,30] # Function call
11 print("Initial values outside the function: ", mylist)
12 print("Address outside the function: ", id(mylist))
13 changelist( mylist )
14 print("Values outside the function: ", mylist)
15 # Initial values outside the function:  [10, 20, 30]
16 # Address outside the function:  140390909223488 # Call by  Value
17 # Address inside the function:  140390919434048  # Call by  Value
18 # Values inside the function:  [7, 3, 4, 9]  # Call by  Value
19 # Values outside the function:  [10, 20, 30] # Call by  Value
20 #
21 # Initial values outside the function:  [10, 20, 30]
22 # Address outside the function:  140390919434048 # Call by Ref.
23 # Address inside the function:  140390919434048  # Call by Ref.
24 # Values inside the function:  [7, 30, 9]  # Call by Reference
25 # Values outside the function:  [7, 30, 9] # Call by Reference
```

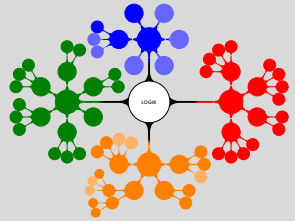
Programming in Python VII

```
1  def step(x): # Your functions can return data if you so choose
2      if (x < 0):
3          return -1
4      elif (x > 0):
5          return 1
6  print(step(-1)) # call functions by repeating their name, and
       putting your variable in the parenthesis
7  print(step(1))  # Your variable need not be named the same thing,
       but it should be the right type
8  # what happens for x = 0?
9  print(step(0)) # Python automatically adds in a "return none"
       statement if you are missing one.
10 #
11 # Fix the return none issue
12 def step_v2(x):
13     if (x < 0):
14         return -1
15     elif (x >= 0):
16         return 1
17 print(step_v2(0))
```

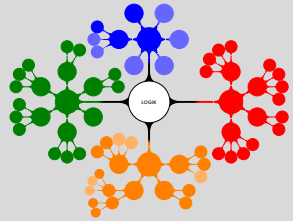
Programming in Python VIII

```
######### Importing in Python #########
# Just about every standard math function on a calculator has a
    python equivalent pre-made.
import math
float1 = 5.75
float2 = 2.25
print(math.log(float1)); print(math.exp(float2)); print(math.pow
    (2,5))
print(2.0**5.0) # There is a quicker way to write exponents
#
######### Numpy - "The Fundamental Package for Scientific
    Computing with Python" #########
# numpy has arrays, which function similarly to Python lists.
#
# Generally, it is used a convention on names used to import
    packages (such as numpy, scipy, and matplotlib)
# import [package] as [alias]
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
#
# Generally scipy is not imported as module because interesting
    functions in scipy are actually located in the submodules, so
     submodules or single functions are imported
# from [package] import [module] as [alias]
from scipy import fftpack
from scipy import integrate
```

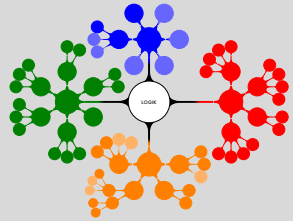
Programming in Python IX

```
1  import numpy as np # Here, we grab all of the functions and tools
       from the numpy package and store them in a local variable
       called np.
2  l = [1,2,3,4] # python list
3  print(l)
4  l_np = np.array(l)
5  print(l_np)
6  print(l*5)    # multiplying a python list replicates it
7  print(l_np*5) # numpy applies operation elementwise
8  #
9  ## 1D array
10 a1 = np.array([1,2,3,4]) # initialized with a numpy list. Be
       careful with syntax. The parentheses and brackets are both
       required
11 print(a1)
12 print(a1.shape)  # shape indicates the rank of the array
13 #
14 ## Rank 2 array
15 # row vector
16 a2 = np.array([[1,2,3,4]])
17 print(a2)
18 print(a2.shape)  # shape indicates the rank of the array. this
       looks more like a row vector
19 # column vector
20 a3 = np.array([[1],
21                [2],
22                [3],
23                [4]])
24 print(a3)
25 print(a3.shape)  # this looks more like a column vector
```

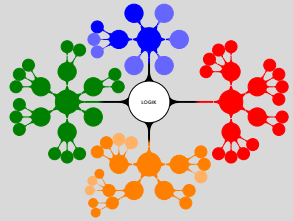
Programming in Python X

```
1  import numpy as np
2  a = np.array([0, 10, 20, 30, 40])
3  print(a)
4  print(a[:])
5  print(a[1:3])
6  a[1] = 15
7  print(a)
8  b = np.arange(-5, 5, 0.5)
9  print(b)
10 print(b**2)
11 1/b
12 # <input>:1: divide by zero encountered in true_divide
13 1/b[10]
14 # <input>:1: divide by zero encountered in double_scalars
15 #
16 ## Element-wise operations
17 a = np.array([1, 2, 3])
18 b = np.array([9, 8, 7])
19 print(a)
20 print(a.shape); print(b.shape)
21 print(a[0]) # Access elements from them just like a list
22 #
23 # Element-wise operations. This is different from MATLAB where you
       add a dot to get element wise operators.
24 c = a + b
25 d = a - b
26 e = a * b
27 f = a / b
28 print(c); print(d); print(e); print(f)
```

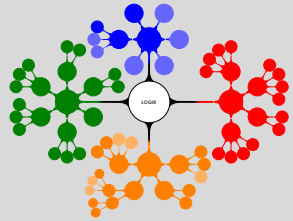
Programming in Python XI

```
1  import numpy as np
2  # What about multi-dimensional arrays? Matrices! You just nest
       lists within lists
3  A = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]]); print(A)
4  # [[1 2 3]
5  #  [4 5 6]
6  #  [7 8 9]]
7  print(A.shape)
8  # (3, 3)
9  B = np.array([[1, 1, 1],[2, 2, 2],[3, 3, 3]]); print(B)
10 C = np.matmul(A, B); print(C) # Then matrix multiplication
11 print(np.linalg.det(A)) # Or determinants
12 #
13 import numpy as np
14 p = np.poly1d([3,4,5])
15 print(p)
16 #    2
17 # 3 x + 4 x + 5
18 print(p*p)
19 #    4      3      2
20 # 9 x + 24 x + 46 x + 40 x + 25
21 print(p.integ(k=6))
22 #    3     2
23 # 1 x + 2 x + 5 x + 6
24 print(p.deriv())
25 # 6 x + 4
26 p([4, 5])
27 # array([ 69, 100])
```

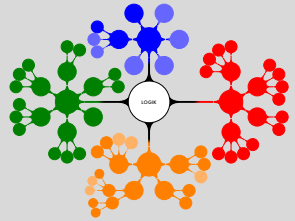
Programming in Python XII

```
1  ##### SciPy is a scientific computation library in Python #####
2  # A collection of functions to perform basic scientific
       programming and data analysis
3  # Integrate a list of numbers using scipy you might use a function
       called trapz from the integrate package
4  import scipy.integrate as integ
5  # from scipy import integrate as integ
6  result = integ.trapz([0, 1, 2, 3, 4, 5])
7  print(result)
8  # Integrate sin(x) from 0 to Pi you could use the quad function
9  import numpy as np
10 import scipy.integrate as integ
11 result = integ.quad(np.sin, 0, np.pi)
12 print(result)
13 #
14 ######### Matplotlib #########
15 # Matplotlib, like many Python packages, is organized into a
       number of "modules" (essentially subsets of functions).
16 import matplotlib.pyplot as plt # import packages with alias
17 # from matplotlib  import pyplot as plt
18 x_vals = [-2, -1, 0, 1, 2]
19 y_vals = [-4, -2, 0, 2, 4]
20 print(x_vals,y_vals)
21 plt.xlabel('abscissa') # add a label to the x axis
22 plt.ylabel('ordinate') # add a label to the y axis
23 plt.title('A practice plot') # add a title
24 plt.plot(x_vals, y_vals, marker="o")
25 plt.savefig('plot_0.png') # save the figure to the current
       diretory as a png file
26 plt.show()
```

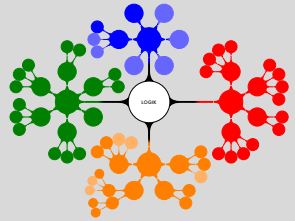
Programming in Python XIII

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  t = np.arange(0.0, 2.0, 0.01)
4  print(t)
5  print(t.shape)
6  print(len(t))
7  s = 1 + np.sin(2*np.pi*t) # Degree to Radian conversion
8  print(s)
9  plt.plot(t, s)
10 plt.xlabel('Time (s)'); plt.ylabel('Voltage (mV)')
11 plt.title('Voltage vs Time')
12 plt.grid(True)
13 plt.savefig("test.png")
14 plt.show()
15 #
16 # Multiplotting
17 import numpy as np
18 import matplotlib.pyplot as plt
19 x1 = np.linspace(0.0, 5.0); x2 = np.linspace(0.0, 2.0)
20 y1 = np.cos(2*np.pi*x1)*np.exp(-x1); y2 = np.cos(2*np.pi*x2)
21 plt.subplot(2, 1, 1) # use the subplot function to generate
       multiple panels within the same plotting window
22 plt.plot(x1, y1, 'o-')
23 plt.title(' 2 subplots')
24 plt.ylabel('Damped oscillation')
25 plt.subplot(2, 1, 2)
26 plt.plot(x2, y2, '.-')
27 plt.xlabel('time (s)'); plt.ylabel('Undamped')
28 plt.show()
```

Programming in Python XIV

```
# importing all functions from pylab module
from pylab import * # Not the preferred methodology. Means to
    bring everything in to the top level name space
x = arange(1, 10, 0.5); print(x)
xsquare = x**2; print(xsquare)
xcube = x**3; print(xcube)
xsquareroot = x**0.5; print(xsquareroot)
figure(1)           # open figure 1
plot(x, xsquare)    # basic plot
xlabel('abscissa') # add a label to the x axis
ylabel('ordinate') # add a label to the y axis
title('A practice plot') # add a title
savefig('plot_1.png') # save the figure to the current diretory
figure(2)           # open a second figure
plot(x, xsquare,'ro',x,xcube,'g+--') # Two plots. Red circles with
    no line. Green plus signs joined by a dashed curve
xlabel('abscissa') # x and y labels, title
ylabel('ordinate') # x and y labels, title
title('More practice') # x and y labels, title
legend(('squared', 'cubed')) # add a legend
savefig('plot_2.png') # save the figure
figure(3)           # open a third figure
subplot(3,1,1); plot(x,xsquareroot,'k*:')# Black stars+dotted line
title('square roots') # add a title
subplot(3,1,2); plot(x, xsquare,'r>-') # Red triangles+dashed line
title('squares') # add a title
subplot(3,1,3); plot(x, xcube,'mh-') # Magenta hexagons+solid line
title('cubes') # add a title
savefig('plot_3.png') # save the figure
show()
```

Programming in Python XV

```
1  ######### How to find documentation #########
2  # The dir(module) function can be used to look at the namespace of
       a module or package, i.e. to find out names that are defined
       inside the module
3  # The help(function) function is available for each module/object
     and allows to know the documentation for each module or
     function
4  #
5  import math
6  dir()
7  dir(math.acos)
8  help(math.acos)
9  import matplotlib.pyplot
10 dir(matplotlib.pyplot)
```

Some links to study python.

- https://python-course.eu/
- https://www.codecademy.com/catalog/language/python
- https://scipy-lectures.org/
- https://computation.physics.utoronto.ca/tutorials/
- https://moodle2.units.it/course/view.php?id=6837
- https://jckantor.github.io/CBE30338/
- https://matplotlib.org/stable/tutorials/index.html