```
# We will now learn how to make figures in Python.
# It is useful to know that
# there are many useful softwares to make figures, such as
# gnuplot (for free), xmgrace (for free),
# Mathematica and Matlab (which require licenses).


# matplotlib is a plotting library for Python
# pyplot is a module that provides a MATLAB-like interface
# matplotlib.pyplot makes matplotlib work like MATLAB


# ------------------------------------------------
# MORE INFO
#  https://github.com/rougier/matplotlib-tutorial
#  https://matplotlib.org/tutorials/index.html
#  https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python
# ------------------------------------------------


# ------------------------------------------------
# ANIMATION:
# https://brushingupscience.com/2016/06/21/matplotlib-animations-the-easy-way/
# http://www.roboticslab.ca/wp-content/uploads/2012/11/
 robotics_lab_animation_example.txt
# ------------------------------------------------


# ------------------------------------------------
# DRAWINGS:
# https://nickcharlton.net/posts/drawing-animating-shapes-matplotlib.html
# ------------------------------------------------


# ------------------------------------------------
# IF ANIMATION DOES NOT SHOW in SPYDER
# https://stackoverflow.com/questions/35856079/animation-from-matplotlib-not-
 working-in-spyder

# You can go to:  python > Preferences > IPython Console > Graphics > Backend
# and change it from "Inline" to "Automatic"

# After changing values, do not forget to restart IDE (Spyder, PyCharm, etc.)
# ------------------------------------------------




# ------------------------------------------------
# Before anything, let us import
# numpy
```

```
# and
# matplolib.pyplot
# and
# math -- just in case we need it also

import matplotlib.pyplot as plt
import numpy as np
import math
# ---------------------------------------------




# ---------------------------
# Example 1
# ---------------------------
# This plot makes a straigth line, where the points are
# (x,y) = (0,1), (1,2), (2,3), (3, 4)
# The y values are given with the list below
# Since the x values were not given,
# they are automatically generated by matplotlib
print('EXAMPLE 1')
plt.plot([1,2,3,4])
plt.ylabel('Label the y-axis')
# Show the plot
plt.show()


# ---------------------------
# Example 2
# ---------------------------
print()
print('EXAMPLE 2')
# The default choice for the plot in Example 1
# is a blue straight line, denoted by 'b-'
# If we wanted, red circles instead, we would need to
# explicitly write 'ro'
plt.plot([1,2,3,4],'ro')
# 'ro' is equivalent to color='red' and linestyle='--'
# so you could also USE:
# ---------------------------------------------
# plt.plot([1,2,3,4],color='red',linestyle='--')
# ---------------------------------------------
plt.ylabel('Label the y-axis')
# Show the plot
plt.show()


# ---------------------------------------------
```

```
# --------------------------
# Example 3
# --------------------------
print()
print('EXAMPLE 3')
# Here, we specify
# --) x and y values
# --) color and line type
plt.plot([2,3,4,5], [4,9,16,25], 'ro')
plt.plot([2,3,4,5], [4,9,16,25], 'b--')
# --) range of the plot
plt.axis([0, 6, 0, 30])
# --) axes labels
plt.xlabel('Values of x')
plt.ylabel('Values of y')
# Show the plot
plt.show()
#
# SHORTER ALTERNATIVE
print()
print('Or equivalently')
x = [2,3,4,5]
y = [4,9,16,25]
plt.plot(x,y, 'ro', x,y,'b--')
plt.axis([0, 6, 0, 30])
plt.xlabel('Values of x')
plt.ylabel('Values of y')
plt.show()
#


# ---------------------------------------------
#          COMMENTS
# ---------------------------------------------
# From the plots above, we conclude that we just need
# the data for the x- and for y-axis in the form of LISTS.
# We can also use ARRAYS and the result is the same.
# Remember that elements can be appended to a LIST,
# but when using an ARRAY, the size has to be decided beforehand.

# In the example below, we will use ARRAYS.
# Two very convenient functions from NUMPY are
# 1)              LINSPACE
# linspace = returns evenly spaced numbers over a specified interval
# np.linspace(first number, last number, total number of numbers)
# For example
# np.linspace(0,2,11) gives
# array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ])
# 2)              ARANGE
```

```
# arange = Similar to linspace, but it uses a step size
# np.arange(first number, last number + dx, dx =step size, increment)
# For example
# np.arange(0,2,0.2) gives
# array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8])
# NOTICE that above we did NOT get the final value 2.
# ---------------------------------------------



# ---------------------------
# Example 4  (USING NUMPY)
# ---------------------------
print()
print('EXAMPLE 4: using numpy')
# Here, we use numpy to generate the data
# linspace = returns evenly spaced numbers over a specified interval
x = np.linspace(0, 2, 100)
# print(x)
# SIMILAR
# arange = Similar to linspace, but uses a step size
# x = np.arange(0, 2, 0.02)
# print(x)
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')

plt.xlabel('x label')
plt.ylabel('y label')

plt.title('Three-Curve Plot')

# Add a legend (note that in plt.plot, we have the curves names)
plt.legend()

plt.show()



# ---------------------------
# Example 5  (SINE and COSINE)
# ---------------------------
# SAVE AS PDF or PNG
# ---------------------------
print()
print('EXAMPLE 5: SAVE')
# To Save, we give a name to the figure.
# In the case below, it is called "outFig".
# Many online examples, use simply "fig"
outFig = plt.figure()
# values of x, y1, and y2
```

```
x = np.arange(0, 10, 0.2)
y1 = np.sin(x)
y2 = np.cos(x)
# sine plot: red dashed line
plt.plot(x, y1, 'r--',label = 'sin')
# cosine plot: black solid line
plt.plot(x, y2, color='black',linestyle='-',linewidth=3.0,label = 'cos')
# Show the legend
plt.legend()
# x and y labels (colors and fontsizes)
plt.xlabel('Values of x', fontsize='12')
plt.ylabel('Values of y',fontsize='18',color='red')
# x and y TICKS (color, fontsize, weight, selection of values)
plt.xticks(fontsize='12',weight='bold')
plt.yticks([-1,0,1],[r'$-1.000$', r'$zero$',r'$one$'],color='blue')
# Range of the plot
plt.axis([0, 12, -1.5, 1.5])
# TITLE
plt.title('Sin-Cos Plot')
# Adding a GRID
plt.grid(True)
# Show the plot
plt.show()
# Save the figure:
# bbox_inches='tight' guarantees that the figure is not chopped
outFig.savefig("MyFig01.pdf",bbox_inches='tight')
outFig.savefig("MyFig01.png",bbox_inches='tight')


# --------------------------
# Example 6  LOG-PLOT
# --------------------------
# and FIGURE SIZE
# --------------------------
print()
print('EXAMPLE 6: LOG')
# Data
x = np.arange(0, 5, 0.2)
y = np.exp(x)
plt.plot(x, y,'ro')
plt.plot(x, y, color='black')
plt.title('Linear Plot')
plt.show()

# LOG plot
# figsize=(6,2) = size of the panel in inches
# width is 6in and height is 2in
plt.figure(figsize=(6,2))
plt.yscale('log')
plt.plot(x, y,'ro')
plt.plot(x, y, color='black')
```

```
plt.title('Log Plot')
plt.show()

# LOG plot
# figsize=(6,2) = size of the panel in inches
# width is 6in and height is 2in
# dpi is for pixels (resolution in dots per inch)
# facecolor = the background color
# edgecolor = the border color
# linewidth -- for the edgecolor
plt.figure(figsize=(6,2),dpi=80, facecolor='y', edgecolor='k', linewidth='4')
plt.yscale('log')
plt.plot(x, y,'ro')
plt.plot(x, y, color='black')
plt.title('Log Plot')
plt.show()


# ---------------------------
# Example 7  FRAME and TICKS
# ---------------------------
print()
print('EXAMPLE 7: FRAME and TICKS')

x = np.arange(0, 5, 0.2)
y = np.exp(x)

# Log plot
plt.yscale('log')
plt.plot(x, y,'ro')
plt.plot(x, y, color='black')

# Size and color of the numbers on the axes
plt.xticks(fontsize='13')
plt.yticks(fontsize='13',color='blue',weight='bold')

# Thickness, color, and which lines are kept in the FRAME
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['left'].set_color('blue')
ax.spines['left'].set_linewidth('3')
ax.spines['bottom'].set_linewidth('3')

# Show also small ticks in x-axis (in y there are shown, because it is log)
ax.minorticks_on()
# Size of the ticks
ax.tick_params('both', length=20, width=2, which='major')
ax.tick_params('both', length=10, width=1, which='minor')

plt.title('Log Plot')
```

```
plt.show()



# ---------------------------
# Example 8  PLOT from DATA
# ---------------------------
print()
print('EXAMPLE 8: PLOT from DATA')

data = np.loadtxt("Lec08_ValuesPlot.txt",float)
x = data[:,0]
y = data[:,1]
plt.plot(x,y)
plt.show()


# ---------------------------------------------------------
# ---------------------------------------------------------
#                    EXERCISES
# ---------------------------------------------------------
# ---------------------------------------------------------

# ---------------------------
# Exercise 1  in class
# ---------------------------
# Plot simultaneously x^2 + x + 1 and x/2-1 from x=-2 to x=2.
# Choose different colors for the curves (one black and one red)
# and use thickness "2"
# Label the axes: "Y Values" and "X Values" - use fontsize='14'
# Use fontsize='12' for the numbers in the x and y axes
# Use length='10' and width='2' for the major ticks
# Use length='5' for the minor ticks
# Have legend "quadratic" and "linear" for the curves



# ---------------------------
# Exercise 2  in class
# ---------------------------
# Plot simultaneously the functions
# y=-x (black dashed), y=x (red dotted), and y=x*sinx (green solid)
# on the interval [-6 Pi,6 Pi].
# Label the axes: "Y Values" and "X Values" - use fontsize='14'
# Use fontsize='12' for the numbers in the x and y axes
# Give a title to your plot



# ----------------------------
# Some Info about RANDOM NUMBERS
```

```
# ----------------------------

# 6 random integers between 1 (including 1) and 5 (not including 5)
print(np.random.randint(1,5,6) )

print()
# 6 real random numbers from a uniform distribution
# between -2 and 5
print(  np.random.uniform(-2,5,6) )
print(  np.random.uniform(-2,5,size=6) )

print()
# a matrix with 2 rows and 4 columns
# where all numbers are real random from a uniform distribution
# between -2 and 5
print(np.random.uniform(-2,5,(2,4) ) )

print()
# 1 random number from a normal (Gaussian) distribution
# with mean=0 and variance=1
print( np.random.normal() )

print()
# 4 random numbers from a normal (Gaussian) distribution
# with mean=0 and variance=1
print( np.random.normal(size=4) )




# ---------------------------
# Exercise 3  in class
# ---------------------------
# Real random numbers between 0 and 1 can be generated with
# x=np.random.rand(3)   ---- where 3 means three numbers
# or
# np.random.uniform(-2,5) ---- where -2 is the minimum value
# it can have and 5 is the maximum value it can have
#
# Make a plot where the x values are integers from 0 to 100
# and y values are obtained from real random numbers as follows
# y[0] = is a real random number between -2 and 2
# y[1] = y[0] + a new real random number also in [-2,2]
# y[2] = y[1] + a new real random number also in [-2,2]
#...
# y[101] = y[100]+a new real random number also in [-2,2]
# Label the axes: "Y Values" and "X Values" - use fontsize='14'
# Use fontsize='12' for the numbers in the x and y axes
# Use fontsize='12' for the numbers in the x and y axes
# Use figsize=(5,5) = size of the panel in inches
# Use background color yellow
```

```
# -------------------------------------------
# -------------------------------------------
# -------------------------------------------
#                 SUBPLOTS
# -------------------------------------------
# -------------------------------------------
# -------------------------------------------
#

# ---------------------------------------
# Example 9  Subplots of different sizes
# ---------------------------------------

#
# create some data to use for the plot
dt = 0.001
t = np.arange(0.0, 10.0, dt)
r1 = np.exp(-t)
r2 = np.cos(t)
r3 = np.sin(t)

# the main axes (=panel) is subplot(111) by default
plt.plot(t, r1)
plt.xlabel('time (s)')
plt.ylabel('Decay')
plt.title('Decaying Function')

# AXES has [left, bottom, width, height]
# this is an inset axes over the main axes (main axes = main panel)
plt.axes([0.3, 0.6, 0.2, 0.2], facecolor='y')
plt.plot(t, r3)
plt.title('Sin')

# this is another inset axes over the main axes
plt.axes([0.65, 0.6, 0.2, 0.2], facecolor='y')
plt.plot(t, r2)
plt.title('Cos')
# CAREFUL! CAREFUL! CAREFUL! CAREFUL!
# plt.axes is different from plt.axis
plt.axis([0, 15, -1.5, 1.5])

plt.show()
```

```
# ---------------------------------------
# Example 10  ADDITIONAL DETAILS
# ---------------------------------------

# create some data to use for the plot
dt = 0.001
t = np.arange(0.0, 10.0, dt)
r1 = np.exp(-t)
r2 = np.cos(t)
r3 = np.sin(t)

# the main axes is subplot(111) by default
plt.plot(t, r1)
# AXES has [left, bottom, width, height]
# NOTE THE DETAIL BELOW for the MAXIMUM VALUE OF THE y-axis
plt.axis([0, 11, 0, 2*np.amax(r1)])
plt.xlabel('time (s)')
plt.ylabel('Decay')
plt.title('Decaying Function')


# this is another inset axes over the main axes
plt.axes([0.3, 0.6, 0.2, 0.2], facecolor='y')
plt.plot(t, r3)
plt.title('Sin')
# NOTE THE DETAIL BELOW for the LIMITS of the x-axis
plt.xlim(0, 15)
# the limits above is equivalent to using plt.axis
# but NOT NOT NOT plt.axes


# this is an inset axes over the main axes
plt.axes([0.65, 0.6, 0.2, 0.2], facecolor='y')
plt.plot(t, r2)
plt.title('Cos')
# NOTE THE DETAIL BELOW for NO TIC LABELS in the x-axis
plt.xticks([])


plt.show()



# ---------------------------
# Example 11  BEAT
# ---------------------------
# https://www.youtube.com/watch?v=IQ1q8XvOW6g
#
# create some data to use for the plot
dt = 0.01
t = np.arange(0.0, 100.0, dt)
```

```
r1 = np.cos(2.*t)
r2 = np.cos(2.1*t)
r3 = r1+r2

# the main axes is subplot(111) by default
# AXES has [left, bottom, width, height]
plt.axes([0.2, 0.1, 0.6, 0.5])
plt.plot(t, r3)
plt.xlabel('time (s)')
plt.ylabel('Sum of Waves')
plt.title('Beat')


# this is another inset axes over the main axes
plt.axes([0.2, 0.67, 0.2, 0.2], facecolor='y')
plt.plot(t, r1)
plt.title('One Wave')


# this is an inset axes over the main axes
plt.axes([0.6, 0.67, 0.2, 0.2], facecolor='y')
plt.plot(t, r2)
plt.title('Another Wave')


plt.show()



# ---------------------------
# Example 12  Just TWO Panels
# of EQUAL size
# ---------------------------

dt = 0.01
t = np.arange(0.0, 100.0, dt)
r1 = np.cos(2.*t)
r2 = np.cos(2.1*t)
r3 = r1+r2

# this is the BOTTOM panel
# AXES has [left, bottom, width, height]
plt.axes([0.2, 0.1, 0.6, 0.35])
plt.plot(t, r3)
plt.xlabel('time (s)')
plt.ylabel('Sum of Waves')
plt.title('Beat')


# this is the TOP panel
plt.axes([0.2, 0.55, 0.6, 0.35])
```

```
plt.plot(t, r1)
plt.ylabel('Sin')
plt.title('One Wave')


plt.show()




# ------------------------------------------------
# Example 13  Subplots
# an equivalent way to get TWO panels of EQUAL size
# ------------------------------------------------
# The figure() command here is optional because figure(1) will be created
# by default, just as a subplot(111) will be created by default if you don't
# manually specify the axes (=subplot=panel).
# The subplot() command specifies
# numrows, numcols, fignum where fignum ranges from 1 to numrows*numcols.
# The commas in the subplot command are optional if numrows*numcols<10.
# So subplot(211) is identical to subplot(2,1,1).

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)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()


# --------------
# ALTERNATIVE
# --------------
# DEFINE the function
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

# Get data for the x-axis
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

# Give a name to the entire figure
# and choose its size
```

```
fig = plt.figure(figsize=(6, 4))

# Make each panel and give a name to each one
ax1 = fig.add_subplot(211)
ax1 = plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

ax2 = fig.add_subplot(212)
ax2 = plt.plot(t2, np.cos(2*np.pi*t2), 'r--')

plt.show()
# -------------------------
# NOTES TO WHAT WE SEE ABOVE
# -------------------------
# 1)
# A tuple is a sequence of immutable Python objects.
# Tuples are sequences, just like lists.
# The differences between tuples and lists are:
# the tuples cannot be changed unlike lists and
# tuples use parentheses, whereas lists use square brackets.

# Creating a tuple is as simple as putting different comma-separated values.
# Optionally you can put these comma-separated values between parentheses also.
# For example -
# tup1 = ('physics', 'chemistry', 1997, 2000);
# tup2 = (1, 2, 3, 4, 5 );


# 2)
# plt.subplots() is a function that returns a tuple containing a
# figure and axes object(s).
#
# The line below
# fig, ax = plt.subplots()
#
# is a concise way to write this:
# fig = plt.figure()
# ax = fig.add_subplot(111)
#
# We need to define "fig" (or any other name we prefer, such as OutFig)
# if we later want to SAVE it.
# "ax" is used to define each panel. In python a panel is an "ax".

# Thus when using fig, ax = plt.subplots()
# you unpack this tuple into the variables fig and ax.



# ------------------------------------------
# ------------------------------------------
# ------------------------------------------
#                 ANIMATION
```

```
# ------------------------------------------
# ------------------------------------------
# ------------------------------------------
#
# How to make an animation?
# We need to combine many plots, each slightly different from the previous one
# For example, let us make 5 plots of x vs cos(t+x)
# where t changes slightly from one plot to the other
# As we move to one plot to the other, we get the impression that
# the cosine function is moving to the left
x = np.linspace(0, 6, 101)
for n in range(5):
    t=n*0.1
    plt.plot(x,np.cos(x+t), 'b-')
    plt.show()

# NOTES

# The purpose of meshgrid is to create a rectangular grid out of an
# array of x values and an array of y values.
# In our example below,
# we have for each t, all values of x and for each (t+x) a value of y
# That is
# t = 0. 0.03 0.06 0.09 0.12 0.15 0.18 0.21 0.24...3
# x = 0. 0.06 0.12 0.18 0.24 0.3  0.36 ... 6
#
# Plot (x,y) for t=0
# (x+t) = 0 0.06 0.12 0.18 0.24 0.3  0.36 ... 6
# y = cos(0), cos(0.06), cos(0.12)... cos(6)
#
# Plot (x,y) for t=0.03
# (x+t) = 0.03 0.09 0.15 0.21 0.27 0.33  0.39 ... 6.03
# y = cos(0.03) cos(0.09) cos(0.15) cos(0.21) ... cos(6.03)
#
# We combine all plots to make an animation.


# --------------------------
#      MESHGRID
# --------------------------
# Understanding the meshgrid
# --------------------------
# STEP by STEP

x = np.linspace(0, 6, 5)
t = np.linspace(0,3,4)
X2, T2 = np.meshgrid(x, t)

F = np.cos(T2+X2)

print(X2[0])
```

```
print(T2[0])
print(F[0])

print()
print(X2[1])
print(T2[1])
print(F[1])

#----------------------

x = np.linspace(0, 6, 5)
t = np.linspace(0,3,4)
X2, T2 = np.meshgrid(x, t)

F = np.cos(T2+X2)

# F is a matrix
# with Nt rows
# and Nx columns
print(F)
print()
# F[0,:] selects the first row of F
print(F[0,:])


#----------------------

x = np.linspace(0, 6, 11)
t = np.linspace(0,3,11)
X2, T2 = np.meshgrid(x, t)

F = np.cos(T2+X2)

# F[0,:] selects the first row of F
plt.plot(x, F[0,:], color='b', lw=2)
plt.show()



# ----------------------------
#     EXAMPLE
################################
# wave moving
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

fig, ax = plt.subplots(figsize=(5, 3))
ax.set(xlim=(-.5, 6.5), ylim=(-1, 1))

x = np.linspace(0, 6, 101)
```

```
t = np.linspace(0,3,101)
X2, T2 = np.meshgrid(x, t)

F = np.cos(T2+X2)

line = ax.plot(x, F[0, :], color='b', lw=2)[0]

def animate(i):
    line.set_ydata(F[i, :])
    return line

# interval sets the speed of the movie
# large interval = slow movie
# small interval = fast movie
# frames -- not necessary
anim = FuncAnimation(fig, animate, interval=200, frames=len(t)-1)

plt.draw()
plt.show()



# ----------------------------
#     EXAMPLE
################################
# frequency w increasing
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
# plt.style.use('ggplot')

fig, ax = plt.subplots(figsize=(5, 3))
ax.set(xlim=(-3, 3), ylim=(-1, 1))

t = np.linspace(-3, 3, 91)
w = np.linspace(1,2,50)
T2, W2 = np.meshgrid(t, w)

F = np.sin(W2*T2)

line = ax.plot(t, F[0, :], color='b', lw=2)[0]

def animate(i):
    line.set_ydata(F[i, :])
    return line

anim = FuncAnimation(
    fig, animate, interval=200, frames=len(t)-1)

plt.draw()
plt.show()
```

```
# ------------------------------------------------------
# ------------------------------------------------------
#                    EXERCISES
# ------------------------------------------------------
# ------------------------------------------------------


# ---------------------------
# Exercise 4  in class
# ---------------------------

# Use the BEAT example to make an animation, where
# the frequency of the r2 goes from 2 to 5 in increments of 0.01.
# Plot only the SUM of the TWO WAVES, that is, only r3.




# ---------------------------------------------
# ---------------------------------------------
# ---------------------------------------------
#             ANIMATED SUBPLOTS
# ---------------------------------------------
# ---------------------------------------------
# ---------------------------------------------



# I AM SKIPPING ALL THESE DETAILS AND GOING STRAIGHT TO
# EXAMPLE 15




#
# -----------------------------------------
# Comment from (there is just one site below, you need
# to copy both lines in one)
# https://stackoverflow.com/questions/34162443/why-do-
# many-examples-use-fig-ax-plt-subplots-in-matplotlib-pyplot-python
#
# plt.subplots() is a function that returns a tuple
# containing a figure and axes object(s). Thus when using
# fig, ax = plt.subplots() you unpack this tuple into the variables
# fig and ax. Having fig is useful if you want to change figure-level
```

```
# attributes or save the figure as an image file later (e.g. with
# fig.savefig('yourfilename.png'). You certainly don't have to use
# the returned figure object but many people do use it later so it's
# common to see. Also, all axes objects (the objects that have plotting
# methods), have a parent figure object anyway, thus:
# fig, ax = plt.subplots()
# is more concise than this:
# fig = plt.figure()
# ax = fig.add_subplot(111)
#
# We can use
# fig = plt.subplots(nrows=2, ncols=2) to set a group of subplots
# with grid(2,2) in one figure object.
# We should then write as
# fig, [[ax1, ax2], [ax3, ax4]] = plt.subplots(nrows=2, ncols=2)
# OR as more evidently done in the Example below.
# Of course, if you use parameters as (nrows=1, ncols=4), then the
# format should be:
# fig, [ax1, ax2, ax3, ax4] = plt.subplots(nrows=1, ncols=4)
# So just remember to keep the construction of the list
# as the same as the subplots grid we set in the figure.

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation



# ---------------------------
# Example 14
# ---------------------------------------------------
# The animation below, shows the BEAT plots in three
# different panels.
# ---------------------------------------------------

fig = plt.figure(figsize=(6,4))


ax01 =fig.add_subplot(221)
ax02 =fig.add_subplot(222)
ax03 =fig.add_subplot(212)
# NOTICE THAT ABOVE,
# if we use for ax03 subplot(2,2,3),
# we get a plot with the same size as the others

ax01.set(xlim=(0, 101), ylim=(-2, 2))
ax02.set(xlim=(0, 101), ylim=(-2, 2))
ax03.set(xlim=(0, 101), ylim=(-2, 2))
```

```
dt = 0.1
dw = 0.01
t = np.arange(0.0, 100.0, dt)
freq = np.arange(2.0, 2.5, dw)
T2, F2 = np.meshgrid(t, freq)

func1 = np.cos(2.*T2)+np.cos(F2*T2)
func2 = np.cos(2.*T2)+np.cos(F2*T2)
func3 = np.cos(2.*T2)+np.cos(F2*T2)

p01 = ax01.plot(t,func1[0,:])[0]
p02 = ax02.plot(t,func2[0,:])[0]
p03 = ax03.plot(t,func3[0,:])[0]


def animate(i):
    p01.set_ydata(func1[i, :])
    p02.set_ydata(func2[i, :])
    p03.set_ydata(func3[i, :])
    return p01, p02, p03

anim = FuncAnimation(fig, animate, interval=200, frames=len(t)-1)


plt.draw()
plt.show()

#
# ---------------------------
# Example 15
# ---------------------------
# ------------------------------------------------
# This is again the BEAT
# but with choices of sizes for the PANELS
# ------------------------------------------------

fig = plt.figure(figsize=(6,4))

# AXES has [left, bottom, width, height]
ax01 =plt.axes([0.1, 0.65, 0.2, 0.2])
ax02 =plt.axes([0.6, 0.65, 0.2, 0.2])
ax03 =plt.axes([0.3, 0.1, 0.3, 0.4])


ax01.set(xlim=(0, 101), ylim=(-2, 2))
ax02.set(xlim=(0, 101), ylim=(-2, 2))
ax03.set(xlim=(0, 101), ylim=(-2, 2))


dt = 0.1
dw = 0.01
```

```
t = np.arange(0.0, 100.0, dt)
freq = np.arange(2.0, 2.5, dw)
T2, F2 = np.meshgrid(t, freq)

func1 = np.cos(2.*T2)+np.cos(F2*T2)
func2 = np.cos(2.*T2)+np.cos(F2*T2)
func3 = np.cos(2.*T2)+np.cos(F2*T2)

p01 = ax01.plot(t,func1[0,:],'r-')[0]
p02 = ax02.plot(t,func2[0,:],'b-')[0]
p03 = ax03.plot(t,func3[0,:],'k-')[0]


def animate(i):
    p01.set_ydata(func1[i, :])
    p02.set_ydata(func2[i, :])
    p03.set_ydata(func3[i, :])
    return p01, p02, p03

anim = FuncAnimation(fig, animate, interval=200, frames=len(t)-1)


plt.draw()
plt.show()



# ---------------------------
# Example 16
# ---------------------------
# ------------------------------------------------------
# The animation below, puts TWO curves on the same panel
#
# The size of the panel is also chosen and its title
# ------------------------------------------------------

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import math

fig = plt.figure(num = 0, figsize=(6, 4))

ax01 =fig.add_subplot(111)

# [left, bottom, width, height]
pos1=[0.3, 0.5, 0.4, 0.4]
ax01.set_position(pos1)

ax01.set(xlim=(0, 4.), ylim=(-2, 2))
#ax02.set(xlim=(0, 4.), ylim=(-2, 2))
```

```
dt = 0.1
dv = 0.1
t = np.arange(0, 4., dt)
v = np.arange(-4., 4., dv)


T2, V2 = np.meshgrid(t, v)

func1 = np.cos(T2*V2)
func2 = np.sin(T2*V2)


# Figure with TWO curves
p01 = ax01.plot(t,func1[0,:],'r-')[0]
p02 = ax01.plot(t,func2[0,:])[0]
ax01.set_title('Two Curves')




def animate(i):
    p01.set_ydata(func1[i, :])
    p02.set_ydata(func2[i, :])
    return p01, p02

anim = FuncAnimation(
    fig, animate, interval=200, frames=len(t)-1)


plt.draw()
plt.show()


# ------------------------------------------------
# ------------------------------------------------
# ------------------------------------------------
#                  DRAWING
# ------------------------------------------------
# ------------------------------------------------
# ------------------------------------------------

# -------
# CIRCLES
# -------

import numpy as np
from matplotlib import pyplot as plt

# alternatives to determine the size of the figure
#fig = plt.figure()
```

```
#fig.set_dpi(200)

#fig = plt.figure()
#fig.set_size_inches(7, 6.5)

#fig, ax = plt.subplots(figsize=(7, 6.5))

fig = plt.figure(figsize=(7, 6.5))
ax = plt.axes(xlim=(0, 10), ylim=(0, 10))

# Circle(xy,radius)
patch=plt.Circle((1, 1), 0.5, color='r')
# equivalent below
#ax.add_artist(patch)
ax.add_patch(patch)

patch2=plt.Circle((5, 5), 0.5, color='b')
ax.add_patch(patch2)

patch3=plt.Circle((9, 9), 0.5, color='g')
ax.add_patch(patch3)

plt.show()


# ------------------------------------
# EXAMPLE FROM
# https://nickcharlton.net/posts/drawing-animating-shapes-matplotlib.html


import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation

fig = plt.figure(figsize=(7, 6.5))
ax = plt.axes(xlim=(0, 10), ylim=(0, 10))
circ = plt.Circle((5, 5), 0.75, color='r')

def animate(i):
    ax.add_patch(circ)
    x, y = circ.center
    x = 5 + 3 * np.sin(np.radians(i))
    y = 5 + 3 * np.cos(np.radians(i))
    circ.center = (x, y)
    return circ,

# frame=360: to use 360 points in animate
#anim = animation.FuncAnimation(fig, animate, frames=360)
# interval=20: how fast the frames should appear
#anim = animation.FuncAnimation(fig, animate, frames=60,interval=20)
# blit = to give a sense of continuous motion
```

```
anim = animation.FuncAnimation(fig, animate, frames=360,interval=20,blit=True)

plt.show()

# There are many other possibilities
# SCATTER
# BAR
# CONTOUR PLOT - DENSITY PLOT
# PIE CHART
# 3D PLOT

# ----------------------------------------
# STUDENTS FIND OUT IN CLASS IN SMALL GROUPS
# ----------------------------------------

#
# --------------------------
# Exercise 5 in class
#    SCATTER PLOT
# --------------------------
# Reproduce Figure 3.4 from Chapter 3 of Newman's book
# Use data file Lec08_stars.txt


# --------------------------
# Exercise 6 in class
#    DENSITY PLOT
# --------------------------
# Reproduce Figure 3.6(a) from Chapter 3 of Newman's book (with COLORS)
# Use data file Lec08_circular.txt
#
# NOTE: the data file contains only the values of "z",
# while the values of x and y are integers ranging from 0 to the
# total number of points "z"

# Bright color = large number
# Dark color = small number
```