Introduction to important modules (libraries)

- Read (and write) netCDF
- Work with array
- Data analysis
- Plotting

Read (and write netCDF)

The netCDF4 module

Read a netCDF

In [3]:
```
# import the module

import netCDF4
```

In [4]:
```
# open the netCDF file
ncfile = netCDF4.Dataset(filename)
```

In [5]:
```
# get global attributes
print ncfile.ncattrs()
```

```
[u'CDI', u'Conventions', u'history', u'source', u'institution', u'title', u'project_id'
, u'experiment_id', u'realization', u'conventionsURL', u'contact', u'references', u'ncc
f_version', u'creation_date', u'CDO']
```

In [6]:
```
# get the variables
print type(ncfile.variables)
print ncfile.variables.keys()
```

```
<class 'collections.OrderedDict'>
[u'lon', u'lat', u'lev', u'time', u'time_bnds', u'vcoord', u'HHL', u'HSURF', u'FIS', u'
FC', u'RLAT', u'RLON', u'FR_LAND', u'SOILTYP', u'VIO3', u'HMO3', u'PLCOV', u'LAI', u'RO
OTDP']
```

In [7]:
```
# close the file
ncfile.close()
```

Write a netCDF

In [9]:
```
# open the dataset, r+ indicate read and write rights
ncfile = netCDF4.Dataset(filename, 'r+') # or 'w'

# create a new dimension
ncfile.createDimension('newdim', 1)

# create a new variable
ncfile.createVariable('newvar', 'f8', ('newdim', ))

# save the changes
ncfile.sync()

# close the files
ncfile.close()
```

Manipulate data using array/matrix

The numpy module

- Homogeneous multidimensional array
- *It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy dimensions are called axes.*

[http://wiki.scipy.org/Tentative_NumPy_Tutorial]

Basics

In [11]:
```
# load the module
import numpy as np
```

In [12]:
```
# create a 3x2 array of zeros
array = np.arange(6).reshape(3, 2)
print array
```

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

In [13]:
```
print array.shape
```

```
(3, 2)
```

In [14]:
```
print array.size
```

```
6
```

In [15]:
```
print array.ndim
```

```
2
```

Create Array

In [16]:
```
print np.array([10,20,30])
```

```
[10 20 30]
```

In [17]:
```
print np.array([[10, 20, 30], [1, 2, 3]])
```

```
[[10 20 30]
 [ 1  2  3]]
```

In [18]:
```
print np.arange(0, 10, 1)
```

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

In [19]:
```
print np.linspace(0,10,11)
```

```
[  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
```

In [20]:
```
print np.zeros((10,10))  # or use empty() or ones()
```

```
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

Basic Operations

In [21]:
```
array = np.arange(6).reshape(3,2)
b = array + 12
print b
```

```
[[12 13]
 [14 15]
 [16 17]]
```

In [22]:
```
print b.min()
print b.max()
```

```
12
17
```

In [23]:
```
print b - array
```

```
[[12 12]
 [12 12]
 [12 12]]
```

In [24]:
```
print 10 * np.sin(b)
```

```
[[-5.36572918  4.20167037]
 [ 9.90607356  6.5028784 ]
 [-2.87903317 -9.61397492]]
```

In [25]:
```
print b.mean(axis=1)
```

```
[ 12.5  14.5  16.5]
```

Slicing and indexing

In [26]:
```
a = np.arange(10)**3
print a
```

```
[  0   1   8  27  64 125 216 343 512 729]
```

In [27]:
```
print a[2], a[2:5]
print a[::-1]
```

```
8 [ 8 27 64]
[729 512 343 216 125  64  27   8   1   0]
```

In [28]:
```
a[:6:2] = - 1000
print a
```

```
[-1000     1 -1000    27 -1000   125   216   343   512   729]
```

In [29]:
```
a.shape = (5,2)
print a[:, 0]
```

```
[-1000 -1000 -1000   216   512]
```

In [30]:
```
print a[2:4, -1]
```

```
[125 343]
```

In [31]:
```
a.shape = (2, 1, 5)
print a[1, ...]
```

```
[[125 216 343 512 729]]
```

Data analysis

The scipy module

module	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 distributions and functions
weave	C/C++ integration

Small example

In [32]:
```
from scipy import integrate
```

In [33]:
```
integrate.trapz([1,2,3])
```

Out[33]:
```
4.0
```

In [34]:
```
integrate.trapz([1,2,3], x=[4,6,8])
```

Out[34]:
```
8.0
```

Plotting

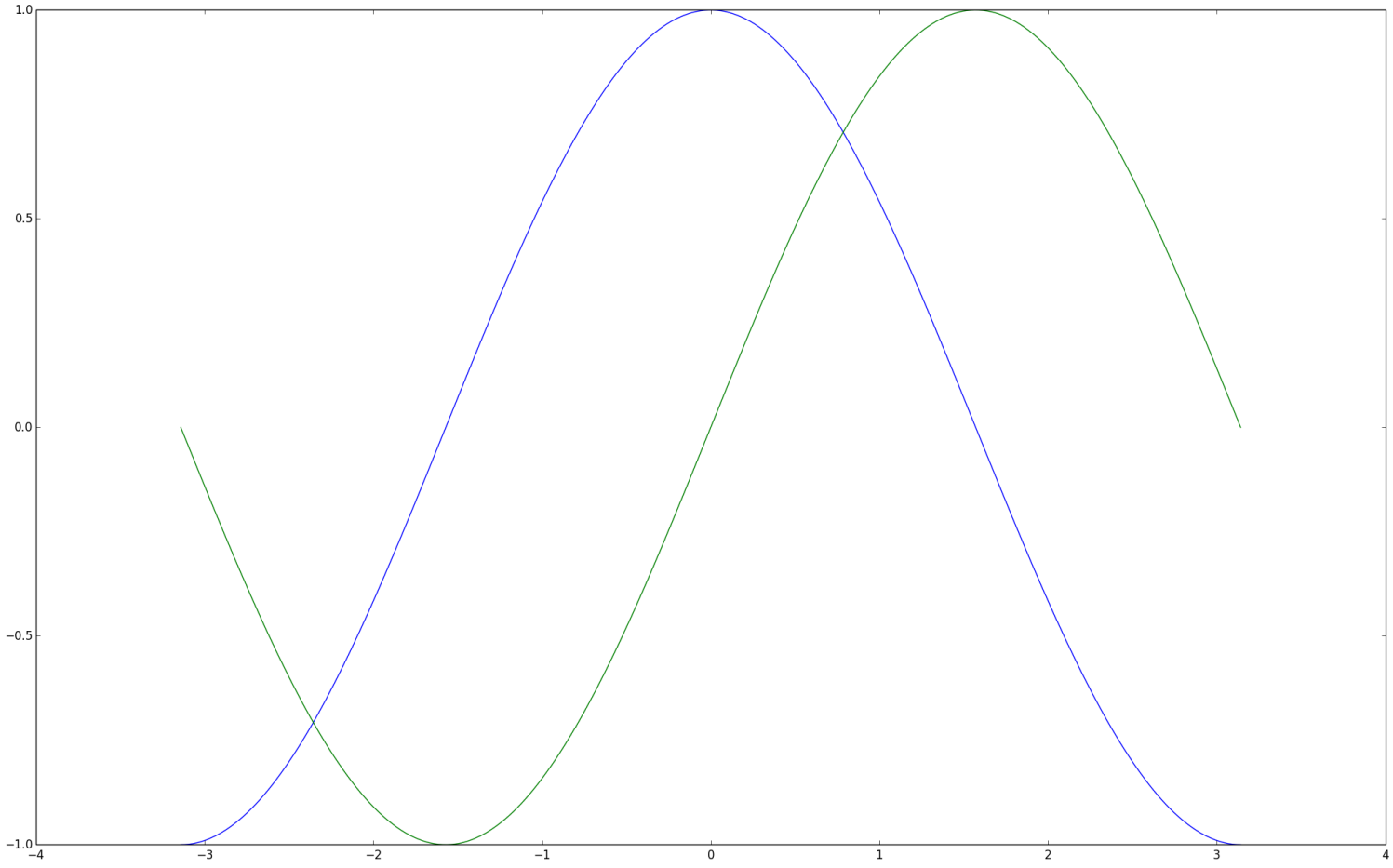
The matplotlib module

based on http://www.loria.fr/~rougier/teaching/matplotlib/

In [36]:
```
# import the module
import matplotlib.pyplot as plt
```

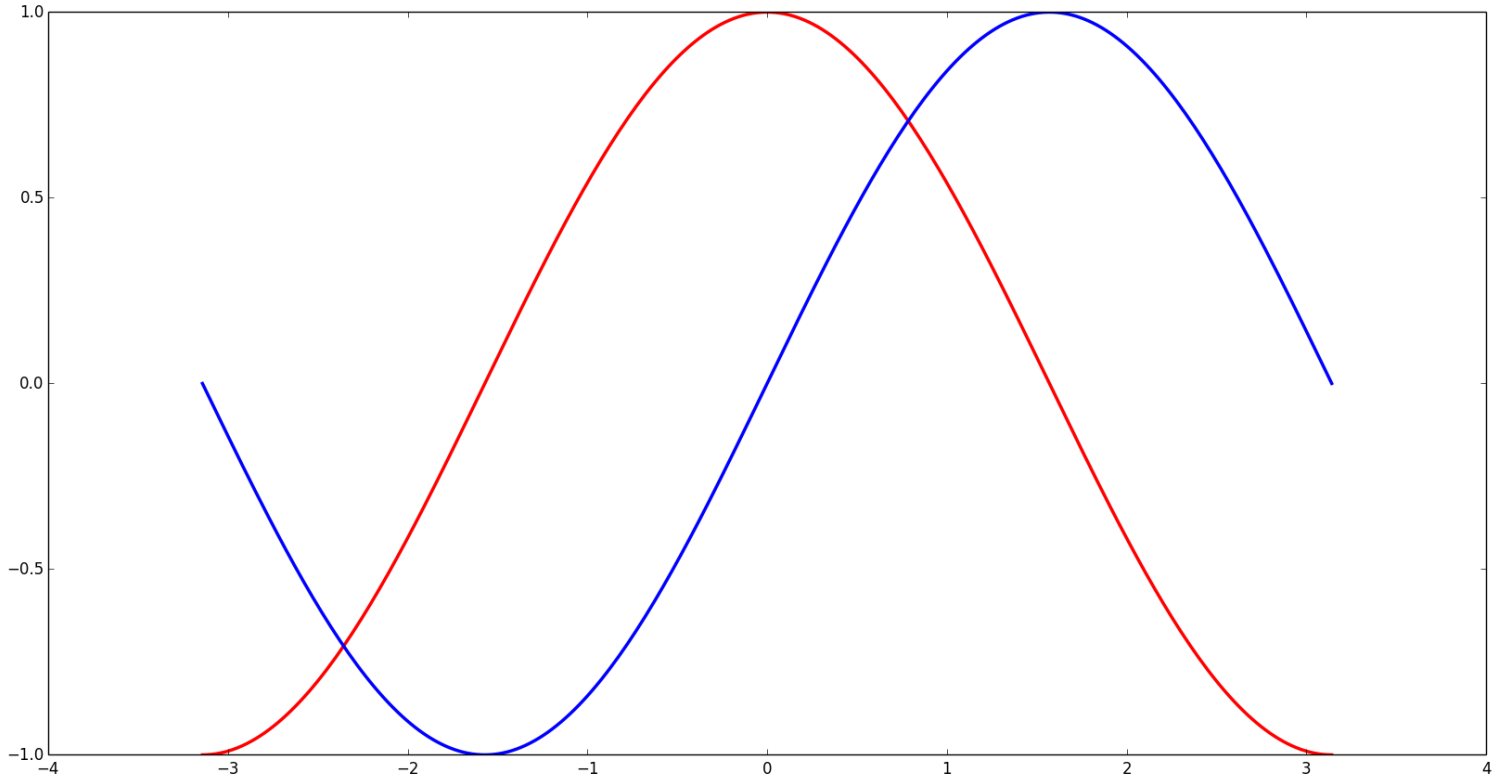
In [37]:
```
X = np.linspace(-np.pi, np.pi, 256,endpoint=True)
C,S = np.cos(X), np.sin(X)
```

In [39]:
```
plt.plot(X,C);
plt.plot(X,S);
```

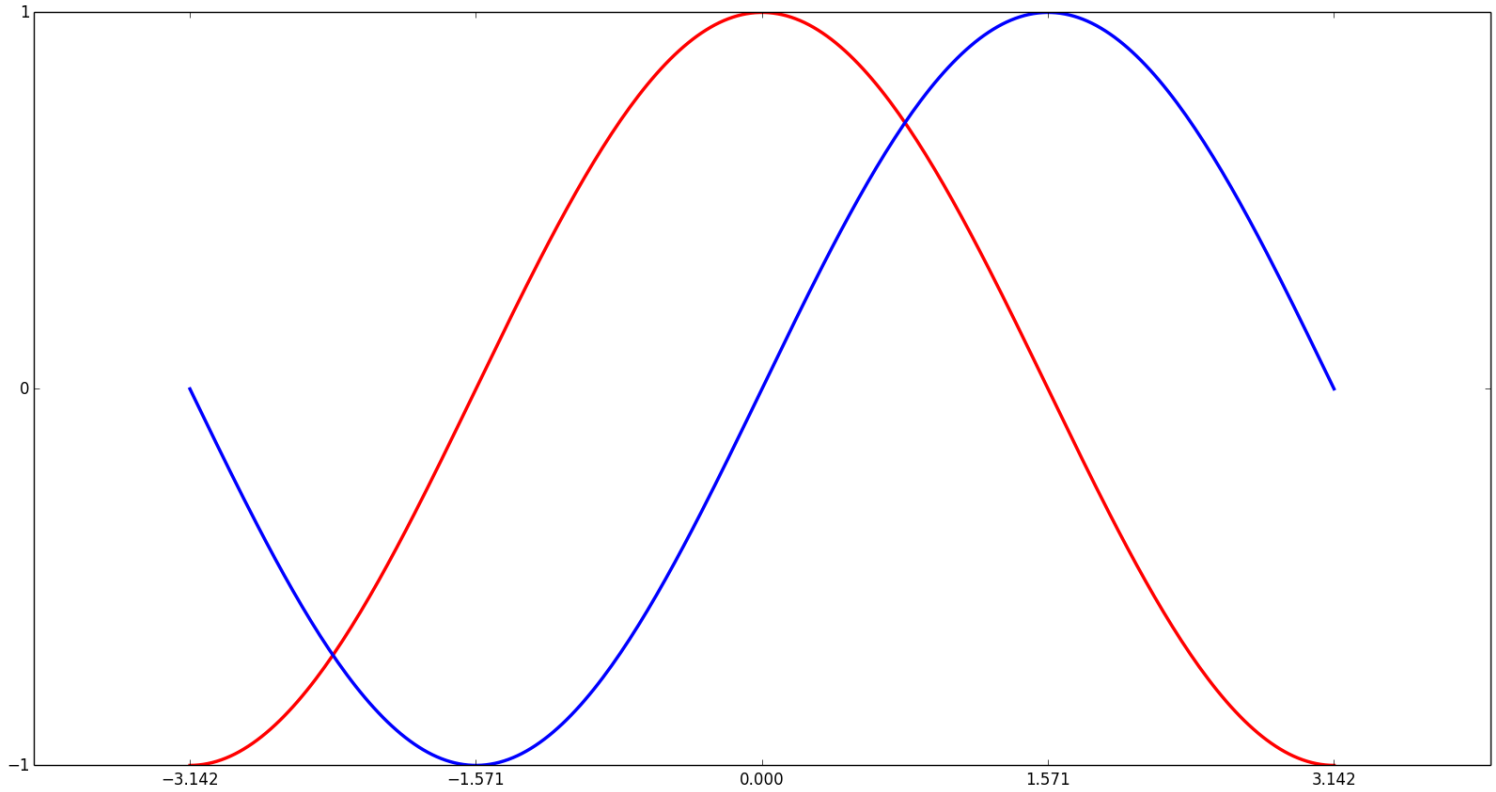


In [41]:

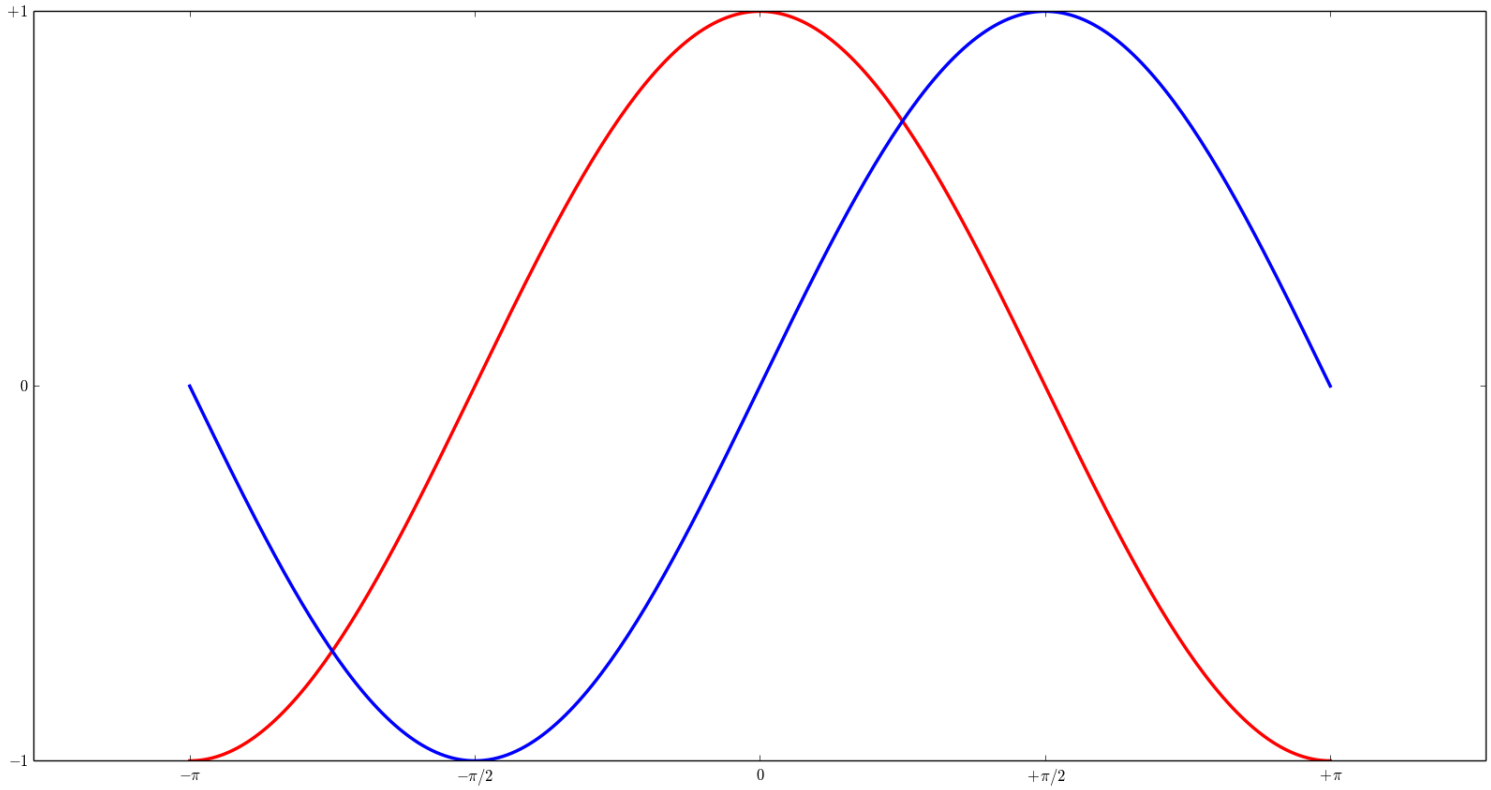
```
fig = plt.figure(figsize=(20,10), dpi=150)

plt.plot(X, C, color="red", linewidth=2.5, linestyle="-")
plt.plot(X, S, "-b",  linewidth=2.5);
```



In [43]:
```
plt.xticks( [-np.pi, -np.pi/2, 0, np.pi/2, np.pi]);
plt.yticks([-1, 0, +1]);
```

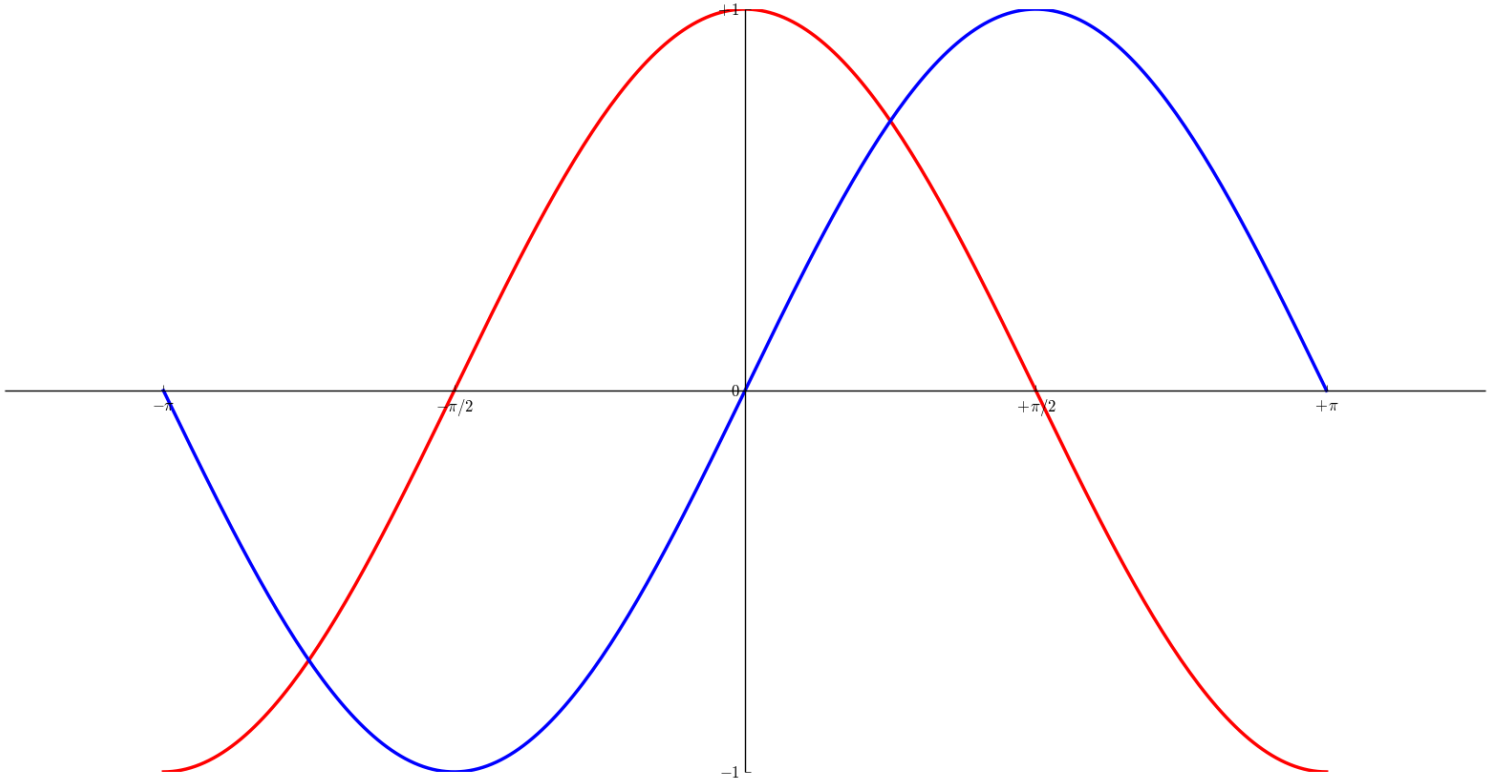


In [45]:
```
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
       [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$']);

plt.yticks([-1, 0, +1],
       [r'$-1$', r'$0$', r'$+1$']);
```

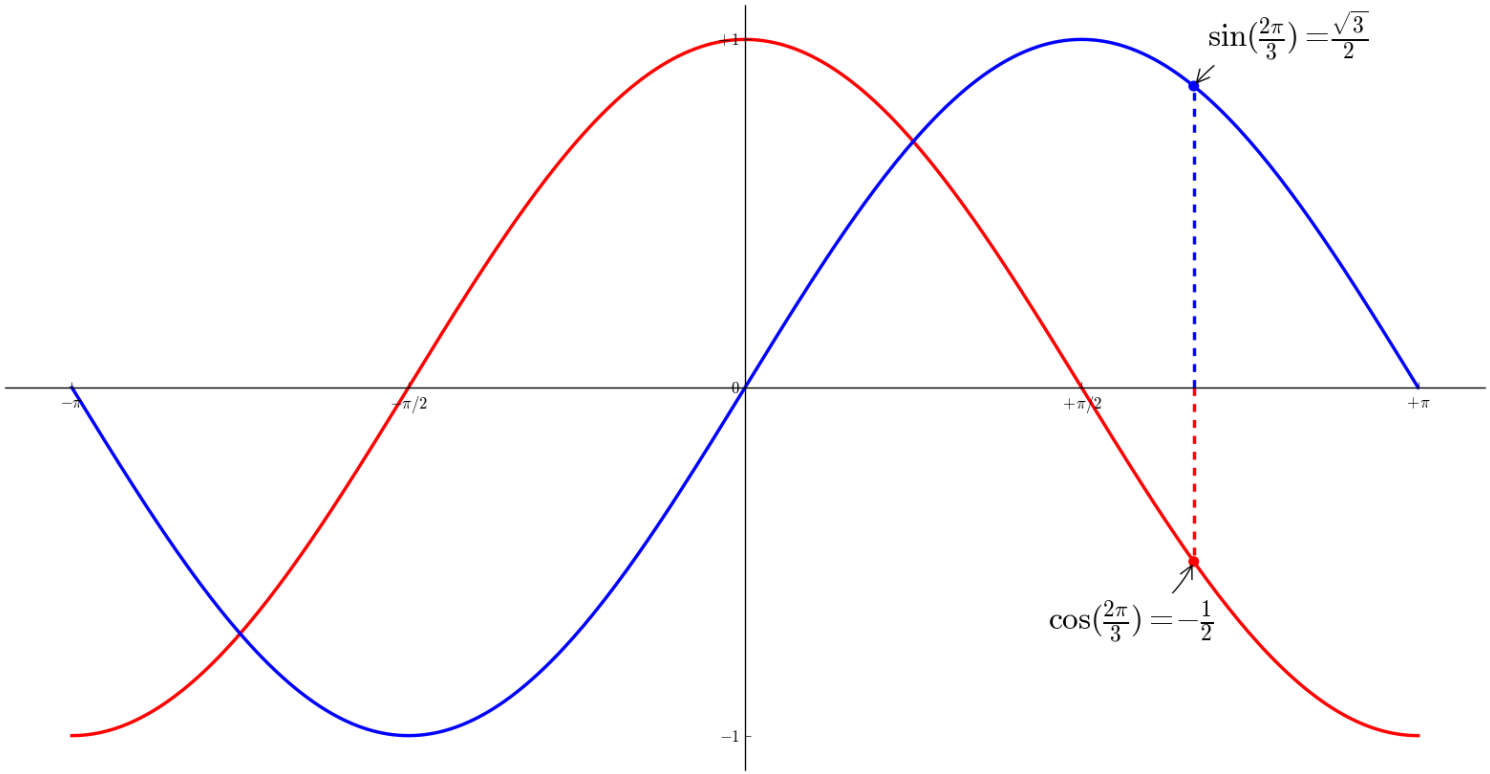


In [47]:

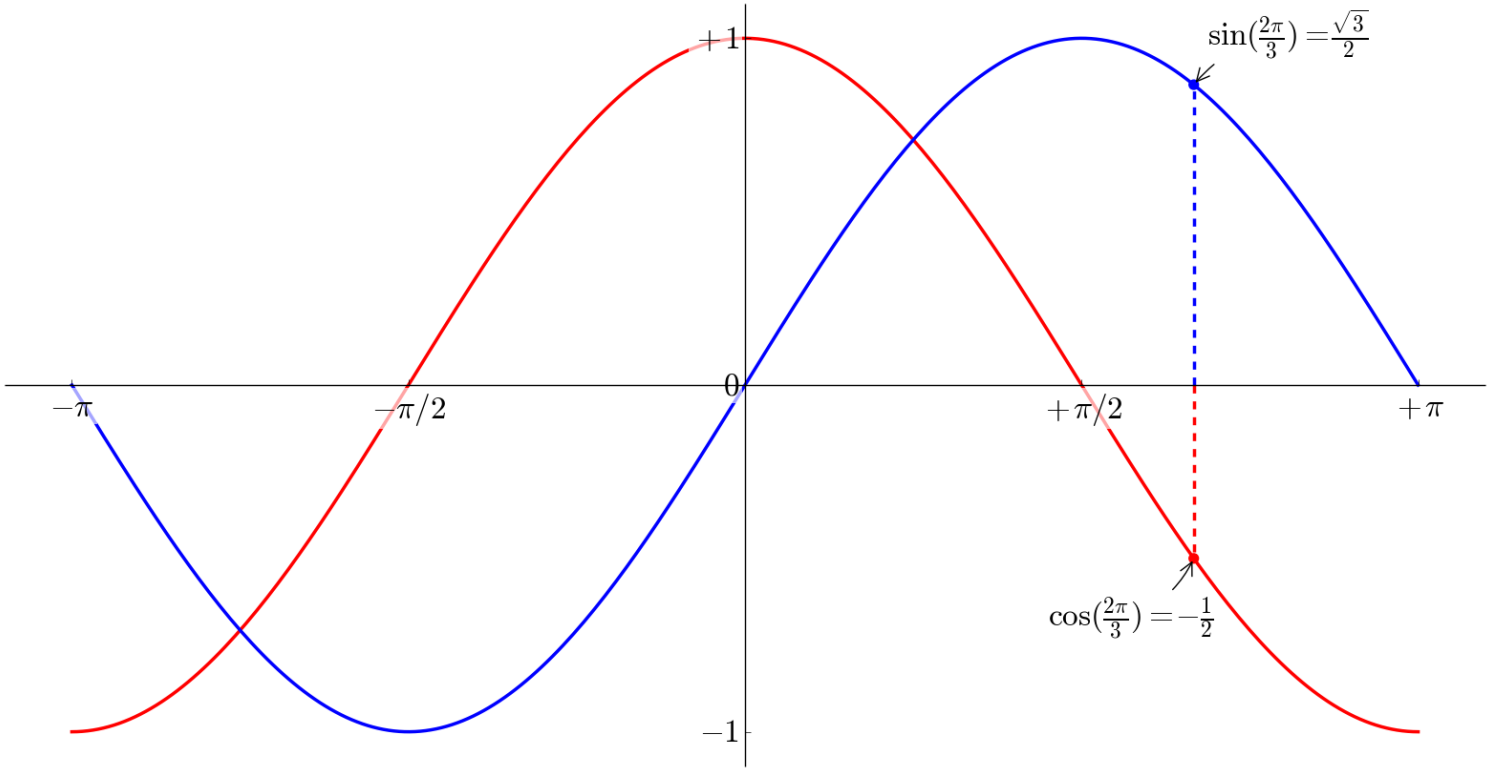
```
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
```



In [50]:
```
t = 2*np.pi/3
plt.plot([t,t],[0,np.cos(t)], color ='red', linewidth=2.5, linestyle="--")
plt.scatter([t,],[np.cos(t),], 50, color ='red')

plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
         xy=(t, np.sin(t)), xycoords='data',
         xytext=(+10, +30), textcoords='offset points', fontsize=24,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t,t],[0,np.sin(t)], color ='blue', linewidth=2.5, linestyle="--")
plt.scatter([t,],[np.sin(t),], 50, color ='blue')

plt.annotate(r'$\cos(\frac{2\pi}{3})=-\frac{1}{2}$',
         xy=(t, np.cos(t)), xycoords='data',
         xytext=(-110, -50), textcoords='offset points', fontsize=24,
         arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"));
```



In [52]:
```
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontsize(24)
    label.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.65 ))
```

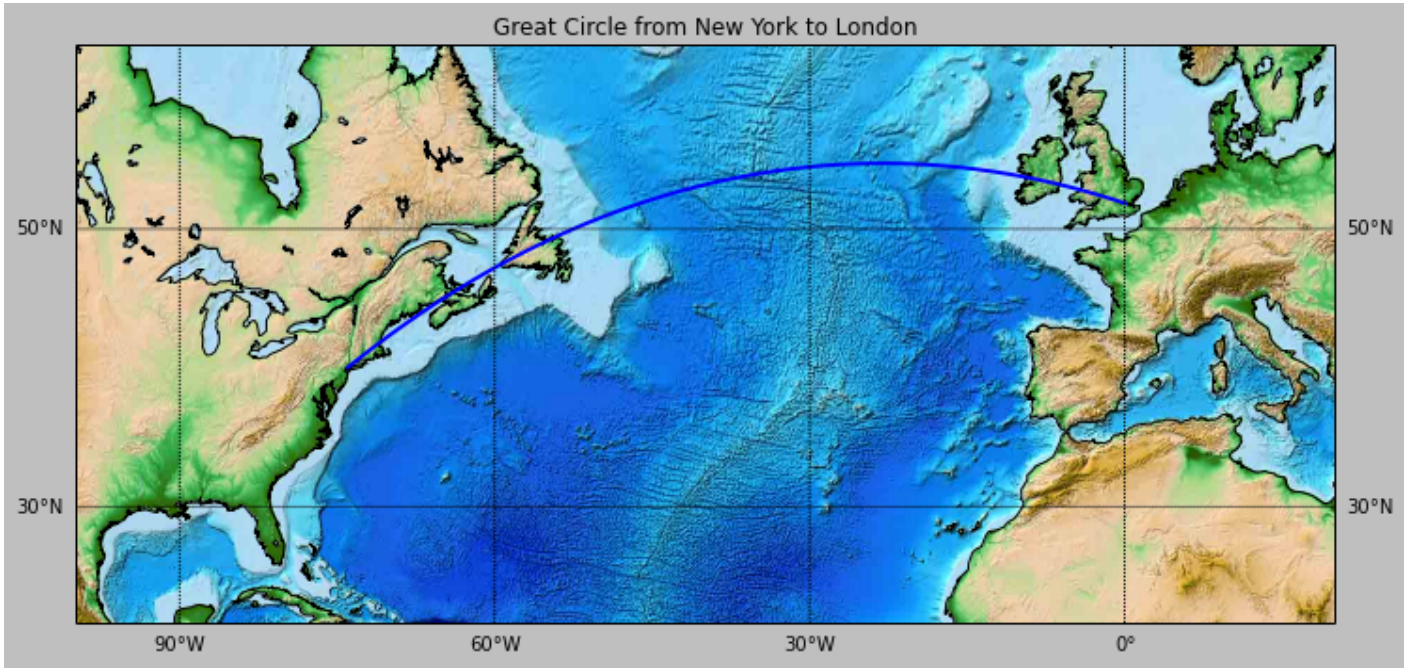


Plotting on a map

The basemap module

In [54]:
```
from mpl_toolkits.basemap import Basemap
m = Basemap(llcrnrlon=-100.,llcrnrlat=20.,urcrnrlon=20.,urcrnrlat=60, resolution='l',
            projection='merc',lat_0=40.,lon_0=-20.,lat_ts=20.)
```

In [61]:
```
# lon / lat of New York and London
(nylat, nylon),  (lonlat, lonlon) = (40.78, -73.98), (51.53, 0.08),

# draw great circle route between NY and London
m.drawgreatcircle(nylon,nylat,lonlon,lonlat,linewidth=2,color='b')
m.drawcoastlines()

# draw parallels
m.drawparallels(np.arange(10,90,20),labels=[1,1,0,1])

# draw meridians
m.drawmeridians(np.arange(-180,180,30),labels=[1,1,0,1])
plt.title('Great Circle from New York to London')
topo = m.etopo()
```



Other plotting possibilities

(some will be used in the next exercise)

In []:
```
m.plot
m.contour
m.contourf
m.scatter
m.barbs
m.quiver
```

Question ?