In [1]:

```
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
#Set array for one side of grid
points = np.arange(-5,5,1)
points
```

Out[2]:

```
array([-5, -4, -3, -2, -1,  0,  1,  2,  3,  4])
```

In [3]:

```
points.shape
```

Out[3]:

```
(10,)
```

In [4]:

```
#Create the grid
dx,dy=np.meshgrid(points,points)
```

In [5]:

```
#Show what one side looks like
dx
```

Out[5]:

```
array([[-5, -4, -3, -2, -1,  0,  1,  2,  3,  4],
       [-5, -4, -3, -2, -1,  0,  1,  2,  3,  4],
       [-5, -4, -3, -2, -1,  0,  1,  2,  3,  4],
       [-5, -4, -3, -2, -1,  0,  1,  2,  3,  4],
       [-5, -4, -3, -2, -1,  0,  1,  2,  3,  4],
       [-5, -4, -3, -2, -1,  0,  1,  2,  3,  4],
       [-5, -4, -3, -2, -1,  0,  1,  2,  3,  4],
       [-5, -4, -3, -2, -1,  0,  1,  2,  3,  4],
       [-5, -4, -3, -2, -1,  0,  1,  2,  3,  4],
       [-5, -4, -3, -2, -1,  0,  1,  2,  3,  4]])
```

In [6]:

```
dx.shape
```

Out[6]:

```
(10, 10)
```

In [7]:

```
dy
```

Out[7]:

```
array([[-5, -5, -5, -5, -5, -5, -5, -5, -5, -5],
       [-4, -4, -4, -4, -4, -4, -4, -4, -4, -4],
       [-3, -3, -3, -3, -3, -3, -3, -3, -3, -3],
       [-2, -2, -2, -2, -2, -2, -2, -2, -2, -2],
       [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
       [ 2,  2,  2,  2,  2,  2,  2,  2,  2,  2],
       [ 3,  3,  3,  3,  3,  3,  3,  3,  3,  3],
       [ 4,  4,  4,  4,  4,  4,  4,  4,  4,  4]])
```

In [8]:

```
dy.shape
```

Out[8]:

```
(10, 10)
```

In [9]:

```
# Evaluating Function
z = (np.sin(dx) + np.sin(dy))
```

In [10]:

```
#Lets take a look at the z result
z
```

Out[10]:

```
array([[ 1.91784855,  1.71572677,  0.81780427,  0.04962685,  0.11745329,
         0.95892427,  1.80039526,  1.8682217 ,  1.10004428,  0.20212178],
       [ 1.71572677,  1.51360499,  0.61568249, -0.15249493, -0.08466849,
         0.7568025 ,  1.59827348,  1.66609992,  0.8979225 ,  0.        ],
       [ 0.81780427,  0.61568249, -0.28224002, -1.05041743, -0.98259099,
        -0.14112001,  0.70035098,  0.76817742,  0.        , -0.8979225 ],
       [ 0.04962685, -0.15249493, -1.05041743, -1.81859485, -1.75076841,
        -0.90929743, -0.06782644,  0.        , -0.76817742, -1.66609992],
       [ 0.11745329, -0.08466849, -0.98259099, -1.75076841, -1.68294197,
        -0.84147098,  0.        ,  0.06782644, -0.70035098, -1.59827348],
       [ 0.95892427,  0.7568025 , -0.14112001, -0.90929743, -0.84147098,
         0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ],
       [ 1.80039526,  1.59827348,  0.70035098, -0.06782644,  0.        ,
         0.84147098,  1.68294197,  1.75076841,  0.98259099,  0.08466849],
       [ 1.8682217 ,  1.66609992,  0.76817742,  0.        ,  0.06782644,
         0.90929743,  1.75076841,  1.81859485,  1.05041743,  0.15249493],
       [ 1.10004428,  0.8979225 ,  0.        , -0.76817742, -0.70035098,
         0.14112001,  0.98259099,  1.05041743,  0.28224002, -0.61568249],
       [ 0.20212178,  0.        , -0.8979225 , -1.66609992, -1.59827348,
        -0.7568025 ,  0.08466849,  0.15249493, -0.61568249, -1.51360499]])
```

In [11]:

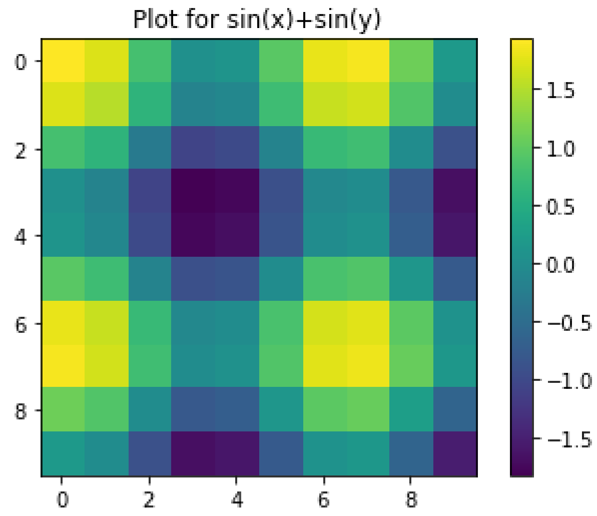
```
#Plot out the 2d array
plt.imshow(z)

#Plot with a colorbar
plt.colorbar()

#Give the plot a title
plt.title("Plot for sin(x)+sin(y)")
```

Out[11]:

```
Text(0.5, 1.0, 'Plot for sin(x)+sin(y)')
```



In [12]:

```
#Lets learn how to use the numpy where

#First the slow way to do things

A = np.array([1,2,3,4])

B= np.array([100,200,300,400])

#Now a boolean array
condition = np.array([True,True,False,False])

#Using a list comprehension
answer = [(A_val if cond else B_val) for A_val,B_val,cond in zip(A,B,condition)]

#Show the answer
answer

#Problems include speed issues and multi-dimensional array issues
```

Out[12]:

```
[1, 2, 300, 400]
```

In [13]:

```
#Now using numpy.where

answer2 = np.where(condition,A,B)

#Show
answer2
```

Out[13]:

```
array([  1,   2, 300, 400])
```

In [14]:

```
#Can use np.where  on 2d for manipulation

from numpy.random import randn

arr = randn(5,5)

#Show arr
arr
```

Out[14]:

```
array([[-0.37021698, -0.33956618,  0.82883049,  0.34665519, -0.98079637],
       [ 0.06921009,  1.0004979 ,  0.64052136,  0.11209235,  1.75189651],
       [-0.11035345,  1.56437168,  0.4142647 , -2.546081  ,  0.07101351],
       [-0.34981608,  0.43118326, -1.43787097, -0.51617652, -0.57059934],
       [ 0.03782089,  0.01353163, -1.55307154, -0.54691713, -1.20821219]])
```

In [15]:

```
# Where array is less than zero, make that value zero, otherwise leave it as the array valu
np.where(arr < 0,0,arr)
```

Out[15]:

```
array([[0.        , 0.        , 0.82883049, 0.34665519, 0.        ],
       [0.06921009, 1.0004979 , 0.64052136, 0.11209235, 1.75189651],
       [0.        , 1.56437168, 0.4142647 , 0.        , 0.07101351],
       [0.        , 0.43118326, 0.        , 0.        , 0.        ],
       [0.03782089, 0.01353163, 0.        , 0.        , 0.        ]])
```

In [16]:

```
#Other Statistical Processing
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])

arr
```

Out[16]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [17]:

```
#SUM
arr.sum()
```

Out[17]:

```
45
```

In [18]:

```
#Can also do along an axis (we shold expect a 3 diff between the columns)
arr.sum(0)
```

Out[18]:

```
array([12, 15, 18])
```

In [19]:

```
#Mean
arr.mean()
```

Out[19]:

```
5.0
```

In [20]:

```
#Standard Deviation
arr.std()
```

Out[20]:

```
2.581988897471611
```

In [21]:

```
#Variance
arr.var()
```

Out[21]:

```
6.666666666666667
```

In [22]:

```
#Also any and all for processing boolean arrays

bool_arr = np.array([True,False,True])

#For any True
bool_arr.any()
```

Out[22]:

```
True
```

In [23]:

```
# For all True
bool_arr.all()
```

Out[23]:

```
False
```

In [24]:

```
# Finally sort array

#Create a random array
arr = randn(5)
#show
arr
```

Out[24]:

```
array([-1.06136653,  1.6536938 ,  0.62660528, -1.15327229,  0.52436674])
```

In [25]:

```
#Sort it
arr.sort()
#show
arr
```

Out[25]:

```
array([-1.15327229, -1.06136653,  0.52436674,  0.62660528,  1.6536938 ])
```

In [26]:

```
#Lets learn about unique
countries = np.array(['France', 'Germany', 'USA', 'Russia','USA','Mexico','Germany'])

np.unique(countries)
```

Out[26]:

```
array(['France', 'Germany', 'Mexico', 'Russia', 'USA'], dtype='<U7')
```

In [27]:

```
# in1d test values in one array
np.in1d(['France','USA','Sweden'],countries)
```

Out[27]:

```
array([ True,  True, False])
```