In [ ]:

```
# Initialize Otter
import otter
grader = otter.Notebook("COGS118B_D1.ipynb")
```

Discussion Notebook Week 1

In this notebook, we are going to do some basic reviews about python, numpy, scipy, along with some basic review with probability theory and estimation.

In [1]:

```
# According to https://github.com/jmshea/jupyterquiz/issues/20
# Restart the kernel if needed
!pip install -q jupyterquiz==2.7.0a1
from jupyterquiz import display_quiz
```

Different Types of Errors

In [2]:

```
display_quiz("data/prob.json")
```

Randon Variables and Probability Events

In [3]:

```
display_quiz("data/rv.json")
```

Expected Values of Discrete / Continous R.V.

In [4]:

```
display_quiz("data/EX.json")
```

Numpy

Import Numpy

In [6]:

```
import numpy as np
```

Numpy Array Creation

Create an one-dimensional array of $\begin{bmatrix}1, 2, 3\end{bmatrix}$ and a two dimensional array of $\begin{bmatrix}1\\2\\3\end{bmatrix}$

In [12]:

```
array_1d = np.array([1, 2, 3])         # SOLUTION

print("1D array:", array_1d, "Shape:", array_1d.shape)

array_2d = np.array([[1], [2], [3]])  # SOLUTION

print("2D Array:")
print(array_2d)
print("Shape:", array_2d.shape)
```

```
1D array: [1 2 3] Shape: (3,)
2D Array:
[[1]
 [2]
 [3]]
Shape: (3, 1)
```

Array Indexing

In [14]:

```
# Given a 3x3 matrix
print("Matrix")
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 3x3 array.
print(m)
```

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

In [15]:

```
# Extract the 2nd row from the Matrix m
second_row = m[1] # SOLUTION
print(second_row)
```

```
[4 5 6]
```

In [16]:

```
# Extract the 3rd column from the Matrix m
third_column = m[:, 2] # SOLUTION
print(third_column)
```

```
[3 6 9]
```

In [17]:

```
# Extract the element in the 1st row and 3rd column from the Matrix m
ele = m[0,2] # SOLUTION
print(ele)
```

```
3
```

Modify an Array

Given the matrix `m` , modify the 1st row of the following 3x3 matrix to [100, 100, 100] inplace.

In [18]:

```
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 3x3 array.
print('Before Modification:')
print(m)
```

```
Before Modification:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [21]:

```
# BEGIN SOLUTION
m[0] = 100
# END SOLUTION

print('After Modification:')
print(m)
```

```
After Modification:
[[100 100 100]
 [  4   5 100]
 [  7   8 100]]
```

In [22]:

```
# Change the last column of the following 3x3 matrix to [100, 100, 100]
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 3x3 array.
print('Before Modification:')
print(m)
```

```
Before Modification:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [23]:

```
# Solution
# BEGIN SOLUTION
m[:, 2] = 100
# END SOLUTION
print('After Modification:')
print(m)
```

```
After Modification:
[[  1   2 100]
 [  4   5 100]
 [  7   8 100]]
```

Math Operations

In [25]:

```
a = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.float64)
```

given the matrix `a` , multiply each element by 3.

In [26]:

```
multiply_3 = a * 3 # SOLUTION
print(multiply_3)
```

```
[[ 3.  6.  9.]
 [12. 15. 18.]]
```

The following is a list of availabe matrix operations when you have two matrices.

In [29]:

```
b = np.array([[1, 1, 1], [2, 2, 2]], dtype=np.float64)

print(a + b)                                            # Elementwise sum
print(a - b)                                            # Elementwise difference
print(a * b)                                            # Elementwise product
print(a / b)                                            # Elementwise division
print(a == b)                                           # Elementwise comparison
```

```
[[2. 3. 4.]
 [6. 7. 8.]]
[[0. 1. 2.]
 [2. 3. 4.]]
[[ 1.  2.  3.]
 [ 8. 10. 12.]]
[[1.  2.  3. ]
 [2.  2.5 3. ]]
[[ True False False]
 [False False False]]
```

In [30]:

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

# Sum of all array elements
print(np.sum(m))
# Sum of each column
print(np.sum(m, axis=0))
# Sum of each row
print(np.sum(m, axis=1))
```

```
45
[12 15 18]
[ 6 15 24]
```

Matrix Operations

In [31]:

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[2, 1], [3, 1]])
# Multiply matrix a with matrix b
ab = a@b # SOLUTION
print(ab)

# Multiply matrix a with a vector x
x = np.array([4, 2])
ax = a@x # SOLUTION
print(ax)
```

```
[[ 8  3]
 [18  7]]
[ 8 20]
```

Load Data with Numpy

In [19]:

```
# The Iris dataset(with no labels) is stored in 'data/iris.txt'
# Load the dataset with Numpy
iris_dataset = np.loadtxt('data/iris.txt') # SOLUTION
```

In [20]:

```
# Each row represents features of sample
# Each column represents one type of feature
print(iris_dataset[:10])
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

Matplotlib

In [32]:

```
# Import matplotlib
import matplotlib.pyplot as plt
```

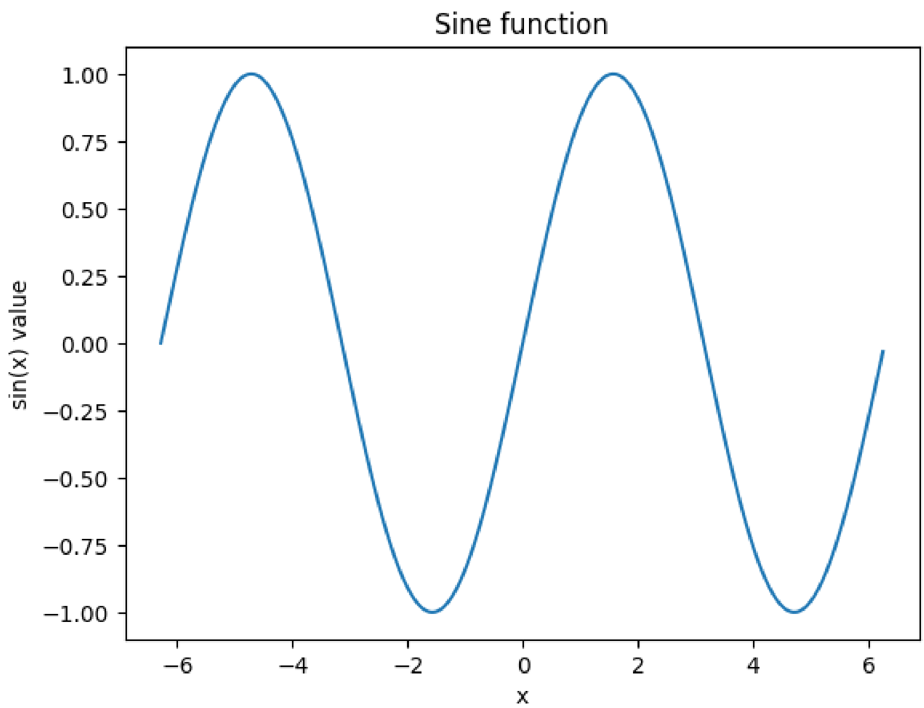
Plot a sine function over the domain of $[-2\pi, 2\pi]$

In [15]:

```
# BEGIN SOLUTION
x = np.arange(-2., 2., 0.01) * np.pi

# Plot sin(x)
plt.plot(x, np.sin(x))
# Set x label
plt.xlabel('x')
# Set y label
plt.ylabel('sin(x) value')
# Set plot title
plt.title('Sine function')

plt.show()
# END SOLUTION
```



In [ ]:

```
grader.check("q6")
```

End of D1 :-)