Introduction to NumPy

`NumPy` stands for Numeric Python or Numerical Python

- Fundamental package for scientific computing in Python
- Provides efficient storage & mathematical operation for multi-dimensional data

Applicatoin of NumPy

Numpy is widely used in scientific works. Check out Ecosystem and Cast Studies at https://numpy.org/ (https://numpy.org/).

In [1]:
```
!pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\isszq\anaconda3\lib\site-p
ackages (1.18.5)
```

In [2]:
```
import numpy as np
np.__version__
```

Out[2]: `'1.18.5'`

NumPy `ndarray` Structure

A fixed-size multidimensional array object efficient data storage.

Fixed Size

- Unlike Python lists which can grow dynamically
- Changing the size of an ndarray will create a new array and delete the original.

Homogeneous Data Type

- All elements must be of same data type

Why Not List?

- List is the most frequently used data type to work with collections in Python
  - Flexible to contain different data types
  - Easy to manipulate data stored in lists
- But...
  - Uses more memory to store data
  - Cumbersome in working with multi-dimensional data

Creating Array

Create array with 0 or 1 values.

In [3]:
```
x = np.zeros(5)
print(x)
y = np.ones([2,4])
print(y)
```

```
[0. 0. 0. 0. 0.]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Create array from existing list.

In [4]:
```
a = np.array([10, 20, 30, 40, 50])
type(a)
```

Out[4]:
```
numpy.ndarray
```

Create a range of value, which is similiar to `range()` function.

In [5]:
```
b = np.arange(1, 11, 2)
b
```

Out[5]:
```
array([1, 3, 5, 7, 9])
```

Create an array with 5 points between 2 and 10 using `linspace()` function.

In [6]:
```
c = np.linspace(2, 10, 5)
c
```

Out[6]:
```
array([ 2.,  4.,  6.,  8., 10.])
```

Basic Operations

Minus 1 from every element in array `c` . This is also known as **Broadcasting**.

In [7]:
```
print(c)
d = c - 1
print(d)
```

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

Compare elements of same index in `b` and `d` . This is also known as **Vectorization**.

In [8]:
```
print(b)
print(d)
b == d
```

```
[1 3 5 7 9]
[1. 3. 5. 7. 9.]
```

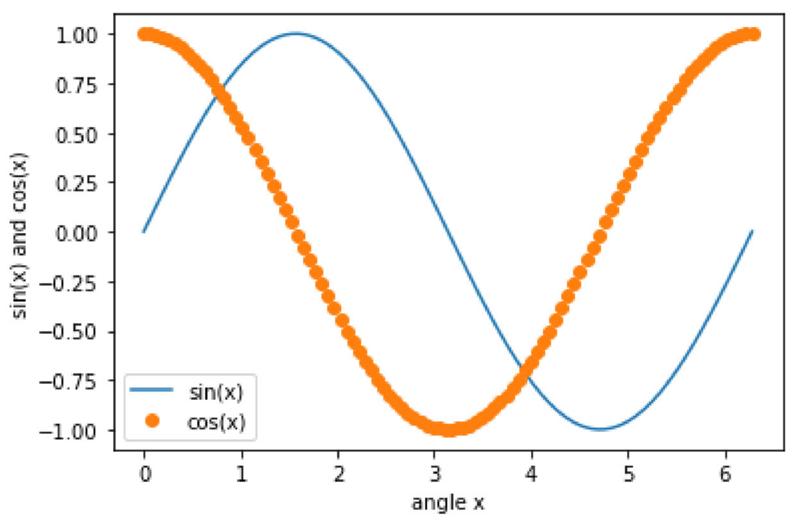
Out[8]:
```
array([ True,  True,  True,  True,  True])
```

Maths Functions

Numpy provides a rich set of maths functions. Check out [https://numpy.org/doc/stable/reference/routines.math.html (https://numpy.org/doc/stable/reference/routines.math.html)](https://numpy.org/doc/stable/reference/routines.math.html).

In [9]:
```
a = np.linspace(0, 2*np.pi, 100)
s = np.sin(a)
c = np.cos(a)

import matplotlib.pyplot as plt
plt.plot(a,s,'-')
plt.plot(a,c,'o')
plt.xlabel('angle x')
plt.ylabel('sin(x) and cos(x)')
plt.legend(['sin(x)', 'cos(x)'])
plt.show()
```



Random Numbers

The `np.random.rand(n)` function generates `n` number of random values uniformly distributed over `[0,1)` .

- Generating of random numbers are much more efficient using Numpy.
- Use `np.random.seed(0)` to fix the seed value to `0` so that the randomly generated values are reproducible.

In [10]:

```
np.random.seed(0)
arr = np.random.rand(3)
print(type(arr))
print(arr)
```

```
<class 'numpy.ndarray'>
[0.5488135  0.71518937 0.60276338]
```

Numpy can also generate multidimensional random integers.

In [11]:

```
x = np.random.randint(0,10,(3,5))
print(x)
print(x.ndim)
print(x.shape)
print(x.size)
print(x.dtype)
```

```
[[3 7 9 3 5]
 [2 4 7 6 8]
 [8 1 6 7 7]]
2
(3, 5)
15
int32
```

Reshape an Array

An ndarray object can be reshape into other dimensions using either `reshape()` or `resize()` .

- The `reshape()` method returns a ndarray of a modified shape
- The `resize()` method modifies the array itself

In [12]:

```
r = np.random.rand(12)
print(r)
arr = r.reshape(3,4)
print(arr)
```

```
[0.47997717 0.3927848  0.83607876 0.33739616 0.64817187 0.36824154
 0.95715516 0.14035078 0.87008726 0.47360805 0.80091075 0.52047748]
[[0.47997717 0.3927848  0.83607876 0.33739616]
 [0.64817187 0.36824154 0.95715516 0.14035078]
 [0.87008726 0.47360805 0.80091075 0.52047748]]
```

Indexing

A `ndarray` behaves like a list. You can access elements by their indexes. NdArray supports **negative index**.

For single array, indexing and slicing is similar to lists.

In [13]:
```
print(r[0])
print(r[-2:])
```

```
0.4799771723750573
[0.80091075 0.52047748]
```

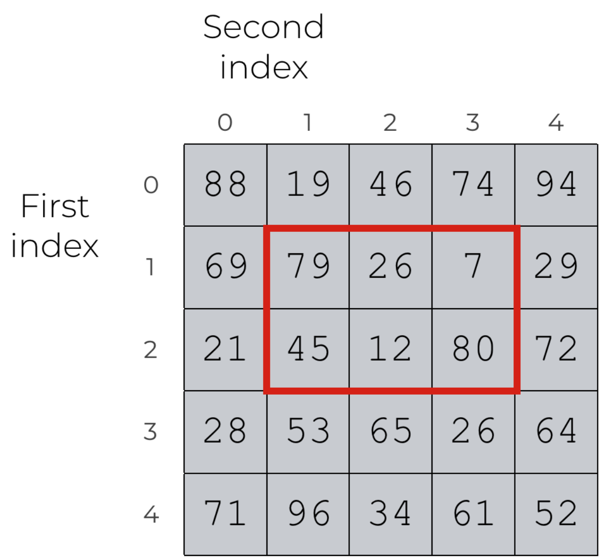
For multi-dimension array, individual element can be accessed by `arr[row, col]` .

In [14]:
```
print(arr[1][2])
```

```
0.9571551589530464
```

Slicing and View

Slicing can be done using `arr[start:end,start:end]` .



In [15]:
```
print(arr)
arr[1:, 1:]
```

```
[[0.47997717 0.3927848  0.83607876 0.33739616]
 [0.64817187 0.36824154 0.95715516 0.14035078]
 [0.87008726 0.47360805 0.80091075 0.52047748]]
```

Out[15]:
```
array([[0.36824154, 0.95715516, 0.14035078],
       [0.47360805, 0.80091075, 0.52047748]])
```

View is a View

Slicing a Numpy array does NOT creates another array. It creates a view pointing to the original ndarray data.

- Modification to the view affects original ndarray.

In [16]:

```
b = arr[1:, 1:]
b[:] = 0
arr
```

Out[16]:

```
array([[0.47997717, 0.3927848 , 0.83607876, 0.33739616],
       [0.64817187, 0.        , 0.        , 0.        ],
       [0.87008726, 0.        , 0.        , 0.        ]])
```