Official Chinese Tutorial [here (https://docs.python.org/zh-cn/3/tutorial/index.html)](https://docs.python.org/zh-cn/3/tutorial/index.html)

# Basic of Python data structure

## List

In [83]:

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
```

```
[3, 1, 2] 2
```

In [5]:

```
xs = [3, 'w', 'sada']
xs
```

Out[5]:

```
[3, 'w', 'sada']
```

In [6]:

```
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
```

```
sada
```

In [7]:

```
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)         # Prints "[3, 1, 'foo']"
```

```
[3, 'w', 'foo']
```

In [9]:

```
xs.append('bar')  # Add a new element to the end of the list
print(xs)         # Prints "[3, 1, 'foo', 'bar']"
```

```
[3, 'w', 'foo', 'bar']
```

In [10]:

```
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)      # Prints "bar [3, 1, 'foo']"
```

```
bar [3, 'w', 'foo']
```

# Slicing of lists

In [11]:

```
nums = list(range(5))     # range is a built-in function that creates a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
```

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

In [12]:

```
nums[1]
```

Out[12]:

```
1
```

In [7]:

```
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])           # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])            # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]"
```

```
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

In [14]:

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
# Prints "cat", "dog", "monkey", each on its own line.
```

```
cat
dog
monkey
```

In [16]:

```
for animal in animals[1:]:
    print(animal)
```

```
dog
monkey
```

In [9]:

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: cat", "#2: dog", "#3: monkey", each on its own line
```

```
#1: cat
#2: dog
#3: monkey
```

## List comprehensions

In [17]:

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)   # Prints [0, 1, 4, 9, 16]
```

```
[0, 1, 4, 9, 16]
```

In [18]:

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Prints [0, 1, 4, 9, 16]
```

```
[0, 1, 4, 9, 16]
```

In [19]:

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

```
[0, 4, 16]
```

# Dictionaries

In [21]:

```
d = {'cat': 'cute',
     'dog': 'furry',
     'fish': 'wet'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
print('cat' in d)     # Check if a dictionary has a given key; prints "True"
```

```
cute
True
```

In [22]:

```
d['fish'] = 'wet'      # Set an entry in a dictionary
print(d['fish'])       # Prints "wet"
```

```
wet
```

In [23]:

```
print(d['monkey'])  # KeyError: 'monkey' not a key of d
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-23-78fc9745d9cf> in <module>
----> 1 print(d['monkey'])  # KeyError: 'monkey' not a key of d

KeyError: 'monkey'
```

In [25]:

```
print(d.get('monkey', 'N/A'))  # Get an element with a default; prints "N/A"
print(d.get('fish', 'N/A'))    # Get an element with a default; prints "wet"
```

```
N/A
wet
```

In [26]:

```
del d['fish']         # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

```
N/A
```

In [27]:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

```
A person has 2 legs
A cat has 4 legs
A spider has 8 legs
```

In [28]:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

```
A person has 2 legs
A cat has 4 legs
A spider has 8 legs
```

## Dictionary comprehensions

In [29]:

```
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)  # Prints "{0: 0, 2: 4, 4: 16}"
```

```
{0: 0, 2: 4, 4: 16}
```

## Set

In [31]:

```
a= [1,2,3,3]
```

In [32]:

```
a
```

Out[32]:

```
[1, 2, 3, 3]
```

In [ ]:

```

```

In [33]:

```
animals = {'cat', 'dog'}
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals)  # prints "False"
```

```
True
False
```

In [34]:

```
animals.add('fish')       # Add an element to a set
print('fish' in animals)  # Prints "True"
```

```
True
```

In [35]:

```
print(len(animals))       # Number of elements in a set; prints "3"
animals.add('cat')        # Adding an element that is already in the set does nothing
print(len(animals))       # Prints "3"
animals.remove('cat')     # Remove an element from a set
print(len(animals))       # Prints "2"
```

```
3
3
2
```

## Set comprehensions

In [38]:

```
[int(sqrt(x)) for x in range(30)]
```

Out[38]:

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

In [37]:

```
from math import sqrt
nums = {int(sqrt(x)) for x in range(30)}
print(nums)  # Prints "{0, 1, 2, 3, 4, 5}"
```

```
{0, 1, 2, 3, 4, 5}
```

## Tuple

A tuple is an (immutable) ordered list of values. A tuple is in many ways similar to a list; one of the most important differences is that tuples can be used as keys in dictionaries and as elements of sets, while lists cannot. Here is a trivial example:

In [44]:

```
d = {(x, x + 1): x for x in range(10)}  # Create a dictionary with tuple keys
t = (5, 6)        # Create a tuple
print(type(t))    # Prints "<class 'tuple'>"
print(d[t])       # Prints "5"
print(d[(1, 2)])  # Prints "1"
```

```
<class 'tuple'>
5
1
```

# Function

In [45]:

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
# Prints "negative", "zero", "positive"
```

```
negative
zero
positive
```

In [28]:

```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)  # Prints "HELLO, FRED!"
```

```
Hello, Bob
HELLO, FRED!
```

# Class

In [29]:

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)

g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

```
Hello, Fred
HELLO, FRED!
```

## Basic of numpy

In [46]:

```
import numpy as np
```

In [47]:

```
from numpy import array
```

In [48]:

```
array
```

Out[48]:

```
<function numpy.array>
```

In [49]:

```
np.array
```

Out[49]:

```
<function numpy.array>
```

## Create array from list

In [34]:

```
a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"
```

```
<class 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
```

In [50]:

```
b = np.array([
    [1,2,3],
    [4,5,6]
])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

```
(2, 3)
1 2 4
```

In [51]:

```
print(b)
```

```
[[1 2 3]
 [4 5 6]]
```

## Use np function to create array

In [52]:

```
a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"
```

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

In [53]:

```
b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"
```

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

In [54]:

```
c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"
```

```
[[7 7]
 [7 7]]
```

In [55]:

```
d = np.eye(6)          # Create a 2x2 identity matrix
print(d)               # Prints "[[ 1.  0.]
                       #          [ 0.  1.]]"
```

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

In [57]:

```
e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #               [ 0.68744134  0.87236687]]"
```

```
[[0.96758346 0.50924067]
 [0.32057946 0.35888028]]
```

# Array Indexing

In [58]:

```
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

In [59]:

```
# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]
print(b)
```

```
[[2 3]
 [6 7]]
```

In [60]:

```
# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

```
2
77
```

In [61]:

```
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

In [46]:

```
# Two ways of accessing the data in the middle row of the array.
# Mixing integer indexing with slices yields an array of lower rank,
# while using only slices yields an array of the same rank as the
# original array:
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)  # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape)  # Prints "[[5 6 7 8]] (1, 4)"
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
```

In [47]:

```
# We can make the same distinction when accessing columns of an array:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)  # Prints "[ 2  6 10] (3,)"
print(col_r2, col_r2.shape)  # Prints "[[ 2]
                             #          [ 6]
                             #          [10]] (3, 1)"
```

```
[ 2  6 10] (3,)
[[ 2]
 [ 6]
 [10]] (3, 1)
```

## Bool indexing

In [62]:

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

```
[[1 2]
 [3 4]
 [5 6]]
```

In [63]:

```
bool_idx = (a > 2)   # Find the elements of a that are bigger than 2;
                     # this returns a numpy array of Booleans of the same
                     # shape as a, where each slot of bool_idx tells
                     # whether that element of a is > 2.
print(bool_idx)
```

```
[[False False]
 [ True  True]
 [ True  True]]
```

In [64]:

```
# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])  # Prints "[3 4 5 6]"
```

```
[3 4 5 6]
```

In [65]:

```
# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

```
[3 4 5 6]
```

## Data type

In [66]:

```
x = np.array([1, 2])   # Let numpy choose the datatype
print(x.dtype)         # Prints "int64"

x = np.array([1.0, 2.0])   # Let numpy choose the datatype
print(x.dtype)             # Prints "float64"

x = np.array([1, 2], dtype=np.float64)   # Force a particular datatype
print(x.dtype)                          # Prints "int64"
```

```
int32
float64
float64
```

## Array math

In [67]:

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

In [68]:

```
# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

In [69]:

```
# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

In [70]:

```
# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

In [71]:

```
# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))
```

```
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

In [72]:

```
# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

```
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

In [73]:

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

v = np.array([9,10])
w = np.array([11, 12])
```

In [74]:

```
# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))
# 9 * 11 + 10 * 12
```

```
219
219
```

In [75]:

```
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))
```

```
[29 67]
[29 67]
```

In [76]:

```
# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

```
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
```

In [77]:

```
# You can also use @ for product
x @ y
```

Out[77]:

```
array([[19, 22],
       [43, 50]])
```

In [78]:

```
x = np.array([[1,2,3],[3,4,5]])
print(x)
```

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

In [65]:

```
print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

```
18
[4 6 8]
[ 6 12]
```

In [79]:

```
x = np.array([[1,2,3], [3,4,5]])
print(x)    # Prints "[[1 2]
            #          [3 4]]"
print(x.T)  # Prints "[[1 3]
            #          [2 4]]"
```

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

In [80]:

```
# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])
print(v)    # Prints "[1 2 3]"
print(v.T)  # Prints "[1 2 3]"
```

```
[1 2 3]
[1 2 3]
```

## Broadcasting

In [82]:

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x
```

In [83]:

```
# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

# Now y is the following
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]
print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [70]:

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))   # Stack 4 copies of v on top of each other
print(vv)                 # Prints "[[1 0 1]
                          #          [1 0 1]
                          #          [1 0 1]
                          #          [1 0 1]]"
y = x + vv  # Add x and vv elementwise
print(y)  # Prints "[[ 2  2  4
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

```
[[1 0 1]
 [1 0 1]
 [1 0 1]
 [1 0 1]]
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [84]:

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)  # Prints "[[ 2  2  4]
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

# PIL for image operation

In [85]:

```
from PIL import Image
```

In [86]:

```
img = Image.open('pics/cat.jpg')
```

In [87]:

```
img
```

Out[87]:

In [88]:

```
print(img.size)
```

(345, 230)

In [89]:

```
img_array = np.array(img)
```

In [90]:

```
img_array.shape
```

Out[90]:

(230, 345, 3)

In [91]:

```
img.convert(mode='L')
```

Out[91]:

In [92]:

```
img.crop([120, 130, 240, 230])
```

Out[92]:

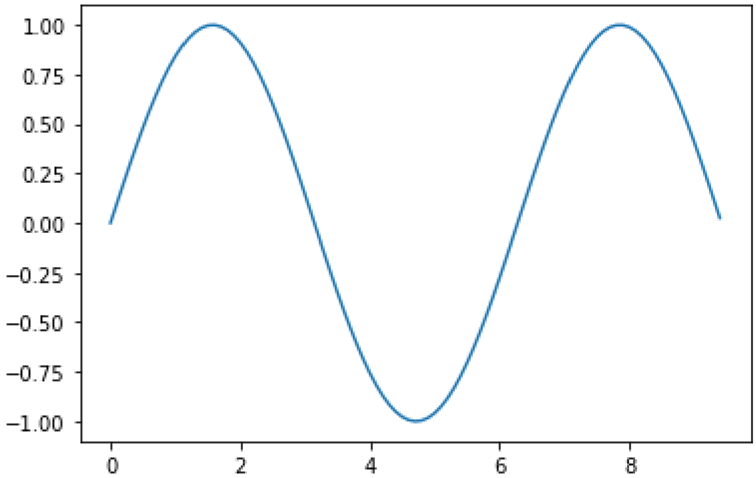
## Matplotlib

In [96]:

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to make graphics appear.
```




In [95]:

```
x
```

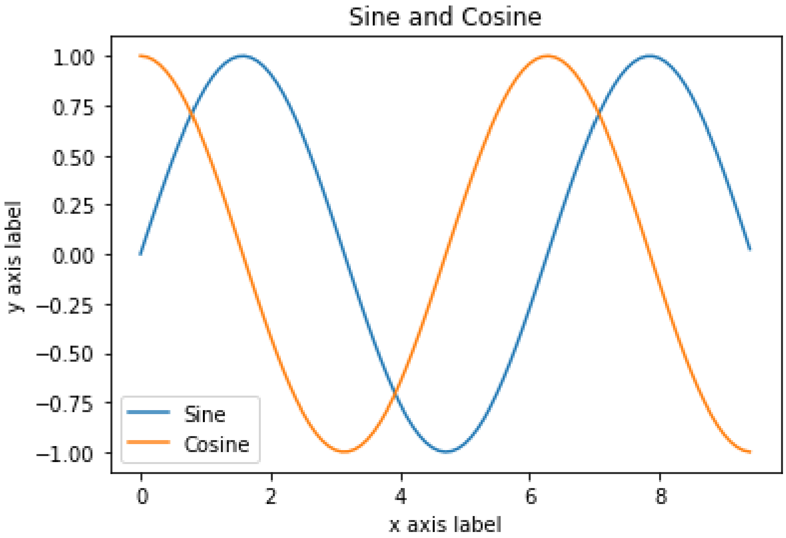
Out[95]:

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
       3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. , 5.1,
       5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4,
       6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
       7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9. ,
       9.1, 9.2, 9.3, 9.4])
```

In [81]:

```
# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



In [82]:

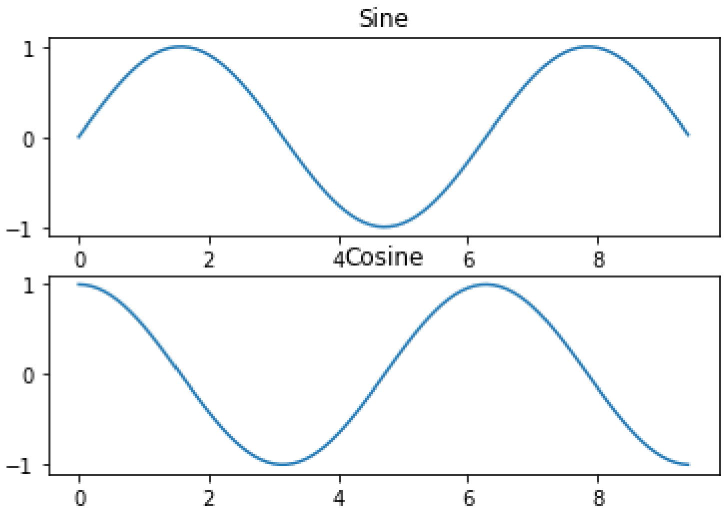
```
# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```




In [ ]:

```
```

In [ ]:

```
```