In [1]:
```python
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]

    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)
print(quicksort([3,6,8,10,1,2,1]))
```
```
[1, 1, 2, 3, 6, 8, 10]
```

In [2]:
```python
x = 3
print(x, type(x))
```
```
3 <class 'int'>
```

In [3]:
```python
print(x + 1)   # Addition
print(x - 1)   # Subtraction
print(x * 2)   # Multiplication
print(x ** 2)  # Exponentiation
```
```
4
2
6
9
```

In [4]:
```python
x += 1
print(x)
x *= 2
print(x)
```
```
4
8
```

In [5]:
```python
y = 2.5
print(type(y))
print(y, y + 1, y * 2, y ** 2)
```
```
<class 'float'>
2.5 3.5 5.0 6.25
```

In [6]:
```python
t, f = True, False
print(type(t))
```
```
<class 'bool'>
```

In [7]:
```python
print(t and f) # Logical AND;
print(t or f)  # Logical OR;
print(not t)   # Logical NOT;
print(t != f)  # Logical XOR;
```
```
False
True
False
True
```

In [8]:
```python
hello = 'hello'   # String literals can use single quotes
world = "world"   # or double quotes; it does not matter
print(hello, len(hello))
```
```
hello 5
```

In [9]:
```python
hw = hello + ' ' + world  # String concatenation
print(hw)
```
```
hello world
```

In [10]:
```python
hw12 = '{} {} {}'.format(hello, world, 12)  # string formatting
print(hw12)
```
```
hello world 12
```

In [11]:
```python
s = "hello"
print(s.capitalize())  # Capitalize a string
print(s.upper())       # Convert a string to uppercase; prints "HELLO"

print(s.rjust(7))      # Right-justify a string, padding with spaces
print(s.center(7))     # Center a string, padding with spaces
print(s.replace('l', '(ell)'))  # Replace all instances of one substring with anoth
print('  world '.strip())  # Strip leading and trailing whitespace
```
```
Hello
HELLO
  hello
 hello
he(ell)(ell)o
world
```

In [12]:
```python
xs = [3, 1, 2]   # Create a list
print(xs, xs[2])
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
```
```
[3, 1, 2] 2
2
```

In [13]:
```python
xs[2] = 'foo'    # Lists can contain elements of different types
print(xs)
```
```
[3, 1, 'foo']
```

In [14]:
```python
xs.append('bar') # Add a new element to the end of the list
print(xs)
```
```
[3, 1, 'foo', 'bar']
```

In [15]:
```python
x = xs.pop()     # Remove and return the last element of the list
print(x, xs)
```
```
bar [3, 1, 'foo']
```

In [16]:
```python
nums = list(range(5))    # range is a built-in function that creates a list of inte
print(nums)         # Prints "[0, 1, 2, 3, 4]"
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
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]"
```
```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

In [17]:
```python
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```
```
cat
dog
monkey
```

In [18]:
```python
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#{}: {}'.format(idx + 1, animal))
```
```
#1: cat
#2: dog
#3: monkey
```

In [19]:
```python
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)
```
```
[0, 1, 4, 9, 16]
```

In [20]:
```python
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)
```
```
[0, 1, 4, 9, 16]
```

In [21]:
```python
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)
```
```
[0, 4, 16]
```

In [22]:
```python
d = {'cat': 'cute', 'dog': 'furry'}  # 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 [23]:
```python
d['fish'] = 'wet'    # Set an entry in a dictionary
print(d['fish'])      # Prints "wet"
```
```
wet
```

In [24]:
```python
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 [25]:
```python
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 [26]:
```python
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A {} has {} legs'.format(animal, legs))
```
```
A person has 2 legs
A cat has 4 legs
A spider has 8 legs
```

In [27]:
```python
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)
```
```
{0: 0, 2: 4, 4: 16}
```

In [28]:
```python
animals = {'cat', 'dog'}
True
False
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals)  # prints "False"
```
```
True
False
```

In [29]:
```python
animals.add('fish')      # Add an element to a set
print('fish' in animals)
print(len(animals))       # Number of elements in a set;
```
```
True
3
```

In [30]:
```python
animals.add('cat')       # Adding an element that is already in the set does nothin
print(len(animals))
animals.remove('cat')    # Remove an element from a set
print(len(animals))
```
```
3
2
```

In [31]:
```python
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
    print('#{}: {}'.format(idx + 1, animal))
```
```
#1: fish
#2: dog
#3: cat
```

In [32]:
```python
from math import sqrt
print({int(sqrt(x)) for x in range(30)})
```
```
{0, 1, 2, 3, 4, 5}
```

In [33]:
```python
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))
print(d[t])
print(d[(1, 2)])
```
```
<class 'tuple'>
5
1
```

In [35]:
```python
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))
```
```
negative
zero
positive
```

In [36]:
```python
def hello(name, loud=False):
    if loud:
        print('HELLO, {}'.format(name.upper()))
    else:
        print('Hello, {}!'.format(name))
hello('Bob')
hello('Fred', loud=True)
```
```
Hello, Bob!
HELLO, FRED
```

In [37]:
```python
class Greeter:
    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable
    # Instance method
    def greet(self, loud=False):
        if loud:
          print('HELLO, {}'.format(self.name.upper()))
        else:
          print('Hello, {}!'.format(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
```

In [38]:
```python
import numpy as np
```

In [39]:
```python
a = np.array([1, 2, 3])  # Create a rank 1 array
print(type(a), a.shape, a[0], a[1], a[2])
a[0] = 5                 # Change an element of the array
print(a)
```
```
<class 'numpy.ndarray'> (3,) 1 2 3
[5 2 3]
```

In [40]:
```python
b = np.array([[1,2,3],[4,5,6]])   # Create a rank 2 array
print(b)
```
```
[[1 2 3]
 [4 5 6]]
```

In [41]:
```python
print(b.shape)
print(b[0, 0], b[0, 1], b[1, 0])
```
```
(2, 3)
1 2 4
```

In [42]:
```python
a = np.zeros((2,2))  # Create an array of all zeros
print(a)
```
```
[[0. 0.]
 [0. 0.]]
```

In [43]:
```python
b = np.ones((1,2))   # Create an array of all ones
print(b)
```
```
[[1. 1.]]
```

In [44]:
```python
c = np.full((2,2), 7) # Create a constant array
print(c)
```
```
[[7 7]
 [7 7]]
```

In [45]:
```python
d = np.eye(2)        # Create a 2x2 identity matrix
print(d)
```
```
[[1. 0.]
 [0. 1.]]
```

In [46]:
```python
e = np.random.random((2,2)) # Create an array filled with random values
print(e)
```
```
[[0.46557281 0.70459386]
 [0.67720505 0.52964318]]
```

In [47]:
```python
import numpy as np
# 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]])
# 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 [48]:
```python
print(a[0, 1])
b[0, 0] = 77    # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])
```
```
2
77
```

In [49]:
```python
# Create the following rank 2 array with shape (3, 4)
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 [50]:
```python
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
row_r3 = a[[1], :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)
print(row_r3, row_r3.shape)
```
```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
[[5 6 7 8]] (1, 4)
```

In [51]:
```python
# 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)
print()
print(col_r2, col_r2.shape)
```
```
[ 2  6 10] (3,)

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

In [52]:
```python
a = np.array([[1,2], [3, 4], [5, 6]])
# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])
# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
```
```
[1 4 5]
[1 4 5]
```

In [53]:
```python
# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]])

# Equivalent to the previous integer array indexing example
print(np.array([a[0, 1], a[0, 1]]))
```
```
[2 2]
[2 2]
```

In [54]:
```python
# Create a new array from which we will select elements
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 [55]:
```python
# Create an array of indices
b = np.array([0, 2, 0, 1])
# Select one element from each row of a using the indices in b
print(a[np.arange(4), b])  # Prints "[ 1  6  7 11]"
```
```
[ 1  6  7 11]
```

In [56]:
```python
# Mutate one element from each row of a using the indices in b
a[np.arange(4), b] += 10
print(a)
```
```
[[11  2  3]
 [ 4  5 16]
 [17  8  9]
 [10 21 12]]
```

In [57]:
```python
import numpy as np
a = np.array([[1,2], [3, 4], [5, 6]])
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 [58]:
```python
# 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])
# We can do all of the above in a single concise statement:
print(a[a > 2])
```
```
[3 4 5 6]
[3 4 5 6]
```

In [59]:
```python
x = np.array([1, 2])  # Let numpy choose the datatype
y = np.array([1.0, 2.0])  # Let numpy choose the datatype
z = np.array([1, 2], dtype=np.int64)  # Force a particular datatype
print(x.dtype, y.dtype, z.dtype)
```
```
int32 float64 int64
```

In [60]:
```python
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
# Elementwise sum; both produce the array
print(x + y)
print(np.add(x, y))
```
```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

In [61]:
```python
# Elementwise difference; both produce the array
print(x - y)
print(np.subtract(x, y))
```
```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

In [62]:
```python
# Elementwise product; both produce the array
print(x * y)
print(np.multiply(x, y))
```
```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

In [63]:
```python
# 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 [64]:
```python
# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```
```
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

In [67]:
```python
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])

w = np.array([11, 12])
# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))
```
```
219
219
```

In [68]:
```python
print(v @ w)
```
```
219
```

In [69]:
```python
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))
print(x @ v)
```
```
[29 67]
[29 67]
[29 67]
```

In [70]:
```python
# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
print(x @ y)
```
```
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
```

In [71]:
```python
x = np.array([[1,2],[3,4]])
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]"
```
```
10
[4 6]
[3 7]
```

In [72]:
```python
print(x)
print("transpose\n", x.T)
```
```
[[1 2]
 [3 4]]
transpose
 [[1 3]
 [2 4]]
```

In [73]:
```python
v = np.array([[1,2,3]])
print(v )
print("transpose\n", v.T)
```
```
[[1 2 3]]
transpose
 [[1]
 [2]
 [3]]
```

In [74]:
```python
# 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
# 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
print(y)
```
```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [75]:
```python
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]]"
```
```
[[1 0 1]
 [1 0 1]
 [1 0 1]
 [1 0 1]]
```

In [76]:
```python
y = x + vv  # Add x and vv elementwise
print(y)
```
```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [77]:
```python
import numpy as np
# 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)
```
```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [78]:
```python
# Compute outer product of vectors
v = np.array([1,2,3])  # v has shape (3,)
w = np.array([4,5])    # w has shape (2,)
# To compute an outer product, we first reshape v to be a column
# vector of shape (3, 1); we can then broadcast it against w to yield
# an output of shape (3, 2), which is the outer product of v and w:
print(np.reshape(v, (3, 1)) * w)
```
```
[[ 4  5]
 [ 8 10]
 [12 15]]
```

In [79]:
```python
# Add a vector to each row of a matrix
x = np.array([[1,2,3], [4,5,6]])
# x has shape (2, 3) and v has shape (3,) so they broadcast to (2, 3),
# giving the following matrix:
print(x + v)
```
```
[[2 4 6]
 [5 7 9]]
```

In [80]:
```python
# Add a vector to each column of a matrix
# x has shape (2, 3) and w has shape (2,).
# If we transpose x then it has shape (3, 2) and can be broadcast
# against w to yield a result of shape (3, 2); transposing this result
# yields the final result of shape (2, 3) which is the matrix x with
# the vector w added to each column. Gives the following matrix:
print((x.T + w).T)
```
```
[[ 5  6  7]
 [ 9 10 11]]
```

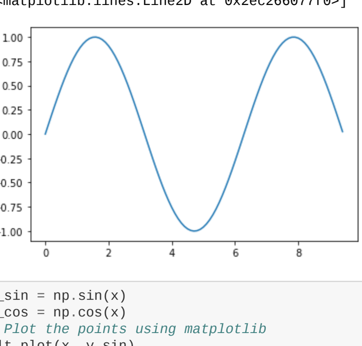
In [81]:
```python
# Another solution is to reshape w to be a row vector of shape (2, 1);
# we can then broadcast it directly against x to produce the same
# output.
print(x + np.reshape(w, (2, 1)))
```
```
[[ 5  6  7]
 [ 9 10 11]]
```

In [82]:
```python
# Multiply a matrix by a constant:
# x has shape (2, 3). Numpy treats scalars as arrays of shape ();
# these can be broadcast together to shape (2, 3), producing the
# following array:
print(x * 2)
```
```
[[ 2  4  6]
 [ 8 10 12]]
```

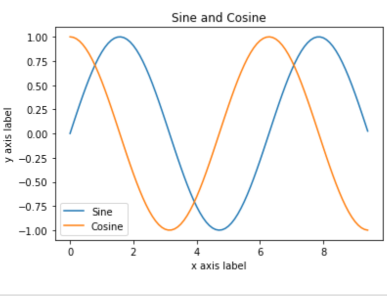
In [83]:
```python
import matplotlib.pyplot as plt
```

In [84]:
```python
%matplotlib inline
```

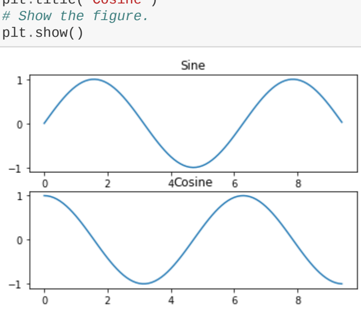
In [85]:
```python
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
# Plot the points using matplotlib
plt.plot(x, y)
```
Out[85]:
```
[<matplotlib.lines.Line2D at 0x2ec266077f0>]
```



In [86]:
```python
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'])
```
Out[86]:
```
<matplotlib.legend.Legend at 0x2ec266b6dc0>
```



In [87]:
```python
# 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 [ ]: