介绍

张晓平

1 Python 介绍

Python 是一种高级的、动态的多范型编程语言。很多时候，大家会说 Python 看起来简直和伪代码一样，因为你可以使用很少几行可读性很高的代码来表达很有力的想法。举个例子，下面是经典快速排序算法的 Python 实现：

```
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]))
# Prints "[1, 1, 2, 3, 6, 8, 10]"
```

1.1 Python versions

目前 Python 有两个支持的版本，分别是 2.7 和 3.4。这有点让人迷惑，Python 3.0 引入了很多不可向下兼容的变化，2.7 下的代码有时候在 3.4 下是行不通的。

如何查看版本呢？可使用如下命令：

```
python --version
```

1.2 基本数据类型

和大多数编程语言一样，Python 拥有一系列的基本数据类型，比如整型、浮点型、布尔型和字符串等。这些数据类型的使用方式和在其他语言中的使用方式是类似的。

1.2.1 Numbers

整型和浮点型的使用和其他语言类似。

```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
```

```
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

注意：Python 没有类似于 ++ 和 -- 的一元运算的操作。Python 也有内置的长整型和复数类型，具体细节可查看文档。

1.2.2 Booleans

Python 实现了所有的布尔逻辑，但用的是英语，而不是我们习惯的操作符 (如&&, || 等):

```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

1.2.3 Strings

Python 对字符串的支持非常强大。

```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello)       # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"
```

字符串对象有一系列常用的方法，例如:

```
s = "hello"
print(s.capitalize())  # Capitalize a string; prints "Hello"
print(s.upper())       # Convert a string to uppercase; prints "HELLO"
print(s.rjust(7))      # Right-justify a string, padding with spaces; prints "
hello"
print(s.center(7))     # Center a string, padding with spaces; prints " hello "
print(s.replace('l', '(ell)'))  # Replace all instances of one substring with
another;
                               # prints "he(ell)(ell)o"
print('  world '.strip())  # Strip leading and trailing whitespace; prints "world"
```

如果想详细查看字符串方法，请查看文档。

1.3 容器（Containers）

Python 包含了几个内置的容器类型：列表（lists）、字典（dictionaries）、集合（sets）、元组（tuples）

1.3.1 列表（list）

列表就是 Python 中的数组，但列表长度可变，且能包含不同类型的元素。

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)         # Prints "[3, 1, 'foo']"
xs.append('bar')  # Add a new element to the end of the list
print(xs)         # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)      # Prints "bar [3, 1, 'foo']"
```

列表的细节，可以查阅文档。

切片（Slicing）：为了一次性地获取列表中的多个元素，Python 提供了一种简洁的语法，这就是切片。

```
nums = list(range(5))     # range is a built-in function that creates a list of
integers
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, 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]"
```

在 Numpy 数组的内容中，我们会再次看到切片语法。

循环（Loops）：我们可以这样遍历列表中的每一个元素：

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

如果你想在循环体内访问每个元素的索引，可使用内置的 enumerate 函数：

```
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
```

列表推导：在编程的时候，我们经常会想要将一种数据类型转换成另一种。下面是一个简单例子，将列表中的每个元素变成它的平方：

```
nums = [0, 1, 2, 3, 4]
squares = []
```

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

使用列表推导，你可以让代码简化很多：

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

列表推导还可以包含条件：

```
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]"
```

1.3.2 字典（Dictionaries）

字典用来存储（键，值）对。你可以这样使用它：

```
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"
d['fish'] = 'wet'     # Set an entry in a dictionary
print(d['fish'])      # Prints "wet"
# print(d['monkey'])  # KeyError: 'monkey' not a key of d
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"
del d['fish']         # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

想要知道字典的其他特性，请查阅文档。

循环（Loops）：在字典中，用键来迭代更加容易。

```
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"
```

如果你想要访问键和对应的值，那就使用 items 方法：

```
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"
```

字典推导（Dictionary comprehensions）：和列表推导类似，但允许你方便地构建字典。

```
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}"
```

1.3.3 集合（Sets）

sets 是离散元的无序集合。示例如下：

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

和前面一样，要知道更详细的，查看文档。

循环（Loops）：在集合中循环的语法和在列表中一样，但是集合是无序的，所以你在访问集合的元素的时候，不能做关于顺序的假设。

```
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: fish", "#2: dog", "#3: cat"
```

集合推导（Set comprehensions）：和列表推导及字典推导类似，可以很方便地构建集合：

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

1.3.4 元组（Tuples）

元组是一个值的有序列表（不可改变）。从很多方面来说，元组和列表都很相似。和列表最重要的不同在于，元组可以在字典中用作键，还可以作为集合的元素，而列表不行。例子如下：

```
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"
```

文档有更多关于元组的信息。

1.4 函数

Python 中使用 def 关键字来定义函数，如：

```
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"
```

我们常常使用可选参数来定义函数，例如：

```
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!"
```

1.5 类

Python 中，对类的定义是简单直接的：

```
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!"
```

更多类的信息请查阅文档。

2 Numpy

Numpy 是 Python 中用于科学计算的核心库。它提供一个高性能的多维数据对象，以及相关工具。

2.1 数组

一个 numpy 数组是一个由不同数值组成的网格。网格中的数据都是同一种数据类型，可以通过非负整型数的元组来访问。维度的数量被称为数组的阶，数组的大小是一个由整型数构成的元组，可以描述数组不同维度上的大小。

我们可以从列表创建数组，然后利用方括号访问其中的元素：

```
import numpy as np

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]"

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"
```

Numpy 还提供了很多其他创建数组的方法：

```
import numpy as np

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

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

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

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

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]]"
```

2.2 访问数组

Numpy 提供了多种访问数组的方法。

切片：和 Python 列表类似，numpy 数组也是可以使用切片语法。因为数组可以是多维的，所以你必须为每个维度指定好切片。

```
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]

# 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"
```

你可以同时使用整型和切片语法来访问数组。但是，这样做会产生一个比原数组低阶的新数组。需要注意的是，这里和 MATLAB 中的情况是不同的：

```
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]])

# 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)"

# 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)"
```

整型数组访问：当我们使用切片语法访问数组时，得到的总是原数组的一个子集。整型数组访问允许我们利用其它数组的数据构建一个新的数组：

```
import numpy as np

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]])  # Prints "[1 4 5]"
```

```
# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

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

整型数组访问语法还有个有用的技巧，可以用来选择或者更改矩阵中每行中的一个元素：

```
import numpy as np

# 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)  # prints "array([[ 1,  2,  3],
          #                [ 4,  5,  6],
          #                [ 7,  8,  9],
          #                [10, 11, 12]])"

# 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]"

# Mutate one element from each row of a using the indices in b
a[np.arange(4), b] += 10

print(a)  # prints "array([[11,  2,  3],
          #                [ 4,  5, 16],
          #                [17,  8,  9],
          #                [10, 21, 12]])
```

布尔型数组访问：布尔型数组访问可以让你选择数组中任意元素。通常，这种访问方式用于选取数组中满足某些条件的元素，举例如下：

```
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)      # Prints "[[False False]
                     #          [ True  True]
```

```
                    #           [ True  True]]"

# 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]"

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

为了教程的简洁，有很多数组访问的细节我们没有详细说明，可以查看文档。

2.3 数据类型

每个 Numpy 数组都是数据类型相同的元素组成的网格。Numpy 提供了很多的数据类型用于创建数组。当你创建数组的时候，Numpy 会尝试猜测数组的数据类型，你也可以通过参数直接指定数据类型，例子如下：

```
import numpy as np

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.int64)   # Force a particular datatype
print(x.dtype)                         # Prints "int64"
```

2.4 Array math

基本数学函数会对数组进行逐元计算，既可以利用操作符重载，也可以使用函数方式：

```
import numpy as np

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
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

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

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

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

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

不同于 MATLAB，* 是逐元乘法，而不是矩阵乘法。在 numpy 中，使用 dot 函数计算向量的的内积、矩阵向量乘法、矩阵乘法。

```
import numpy as np

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))

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

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

Numpy 提供很多计算数组的函数；其中最常用的一个是 sum：

```
import numpy as np

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]"
```

其他一些数学函数：

```
import numpy as np

# absolute value, print 1
a = np.abs(-1)

# sin function, print 1.0
b = np.sin(np.pi/2)

# arctanh function, print 0.50000107157840523
c = np.arctanh(0.462118)

# exponential function, print 20.085536923187668
d = np.exp(3)

# cubic of 2, print 8
f = np.power(2, 3)

# dot of [1,2] and [3,4], print 11
g = np.dot([1, 2], [3, 4])

# sqrt function, print 5.0
h = np.sqrt(25)

# mean of [4,5,6,7], print 5.5
m = np.mean([4, 5, 6, 7])

# standard deviation, print 0.96824583655185426
p = np.std([1, 2, 3, 2, 1, 3, 2, 0])
```

想要了解更多数学函数，可以查看文档。

除了计算，我们还常常改变数组或者操作其中的元素。其中将矩阵转置是常用的一个，在 Numpy 中，使用 T 来转置矩阵：

```
import numpy as np

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

# 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]"
```

2.5 广播机制（Broadcasting）

广播是一种强有力的机制，它允许 numpy 让不同大小的矩阵在一起进行数学计算。我们常常会有一个小的矩阵和一个大的矩阵，然后我们会需要用小的矩阵对大的矩阵做一些计算。

举个例子，如果我们想要把一个向量加到矩阵的每一行，我们可以这样做：

```
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 = 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

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

这样是行得通的，但是当 x 矩阵非常大，利用循环来计算就会变得很慢很慢。我们可以换一种思路：

```
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])
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]]"
```

Numpy 广播机制可以让我们不用创建 vv，就能直接运算，看看下面例子：

```
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)  # Prints "[[ 2  2  4]
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

由于广播机制，即使 x 有 shape (4, 3), v 有 shape (3,)，行y = x + v 仍可工作；这就如同 v 有 shape (4, 3)，其中每一行为 v 的拷贝，求和按逐元进行。

Broadcasting two arrays together follows these rules:

1. If the arrays do not have the same rank, prepend the shape of the lower rank array with 1s until both shapes have the same length.
2. The two arrays are said to be compatible in a dimension if they have the same size in the dimension, or if one of the arrays has size 1 in that dimension.
3. The arrays can be broadcast together if they are compatible in all dimensions.
4. After broadcasting, each array behaves as if it had shape equal to the elementwise maximum of shapes of the two input arrays.
5. In any dimension where one array had size 1 and the other array had size greater than 1, the first array behaves as if it were copied along that dimension
6. If this explanation does not make sense, try reading the explanation from the documentation or this explanation.

广播两个数组遵循以下原则：

1. 如果数组的秩不同，使用 1 来将秩较小的数组进行扩展，直到两个数组的尺寸的长度都一样。
2. 如果两个数组在某个维度上的长度是一样的，或者其中一个数组在该维度上长度为 1，那么我们就说这两个数组在该维度上是相容的。
3. 如果两个数组在所有维度上都是相容的，他们就能使用广播。
4. 如果两个输入数组的尺寸不同，那么注意其中较大的那个尺寸。因为广播之后，两个数组的尺寸将和那个较大的尺寸一样。
5. 在任何一个维度上，如果一个数组的长度为 1，另一个数组长度大于 1，那么在该维度上，就好像是对第一个数组进行了复制。

如果上述解释看不明白，可以读一读文档和这个解释。

支持广播机制的函数是全局函数。哪些是全局函数可以在文档中查找。

下面是一些广播机制的使用：

```
import numpy as np

# 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:
```

```
# [[ 4  5]
#  [ 8 10]
#  [12 15]]
print(np.reshape(v, (3, 1)) * w)

# 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:
# [[2 4 6]
#  [5 7 9]]
print(x + v)

# 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:
# [[ 5  6  7]
#  [ 9 10 11]]
print((x.T + w).T)
# Another solution is to reshape w to be a column 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)))

# 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:
# [[ 2  4  6]
#  [ 8 10 12]]
print(x * 2)
```

广播机制能够让你的代码更简洁更迅速，能够用的时候请尽量使用！

3 SciPy

Numpy 提供了高性能的多维数组，以及计算和操作数组的基本工具。Scipy 基于 Numpy，提供了大量的处理 numpy 数组的函数，这些函数对于不同类型的科学和工程计算非常有用。

熟悉 SciPy 的最好方法就是阅读文档。

3.1 图像操作

Scipy 提供了一些操作图像的基本函数。例如，它提供了将图像从硬盘读入到数组的函数，也提供了将数组中数据写入硬盘成为图像的函数，还提供了调整图像大小的函数。下面是一个简单的例子：

```
from scipy.misc import imread, imsave, imresize
```

```
# Read an JPEG image into a numpy array
img = imread('assets/cat.jpg')
print(img.dtype, img.shape)  # Prints "uint8 (400, 248, 3)"

# We can tint the image by scaling each of the color channels
# by a different scalar constant. The image has shape (400, 248, 3);
# we multiply it by the array [1, 0.95, 0.9] of shape (3,);
# numpy broadcasting means that this leaves the red channel unchanged,
# and multiplies the green and blue channels by 0.95 and 0.9
# respectively.
img_tinted = img * [1, 0.95, 0.9]

# Resize the tinted image to be 300 by 300 pixels.
img_tinted = imresize(img_tinted, (300, 300))

# Write the tinted image back to disk
imsave('assets/cat_tinted.jpg', img_tinted)
```

3.2 MATLAB 文件

函数 scipy.io.loadmat 和 scipy.io.savemat 允许你读写 MATLAB 函数。具体请查看文档。

3.3 点之间的距离

Scipy 定义了一些有用的函数，可计算集合中点之间的距离。

函数 scipy.spatial.distance.pdist 计算给定集合中所有两点之间的距离：

```
import numpy as np
from scipy.spatial.distance import pdist, squareform
```

```
# Create the following array where each row is a point in 2D space:
# [[0  1]
#  [1  0]
#  [2  0]]
x = np.array([[0, 1], [1, 0], [2, 0]])
print(x)

# Compute the Euclidean distance between all rows of x.
# d[i, j] is the Euclidean distance between x[i, :] and x[j, :],
# and d is the following array:
# [[ 0.          1.41421356  2.23606798]
#  [ 1.41421356  0.          1.        ]
#  [ 2.23606798  1.          0.        ]]
d = squareform(pdist(x, 'euclidean'))
print(d)
```

具体细节请阅读文档。

函数 scipy.spatial.distance.cdist 可以计算不同集合中点的距离。

4 Matplotlib

Matplotlib 是一个绘图库。这里简要介绍 matplotlib.pyplot 模块，功能和 MATLAB 的绘图功能类似。

4.1 Plotting

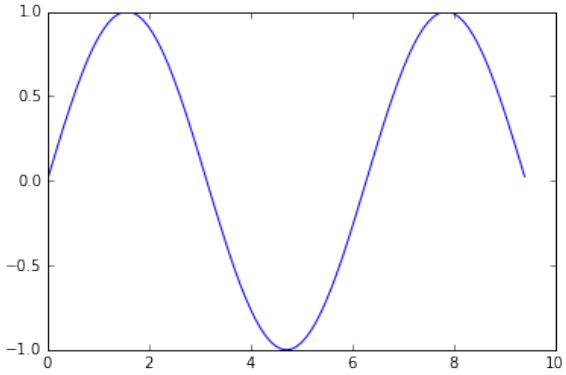
matplotlib 中最重要的函数是 plot，该函数允许你绘制 2D 图形。这是一个简单的例子：

```
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.
```

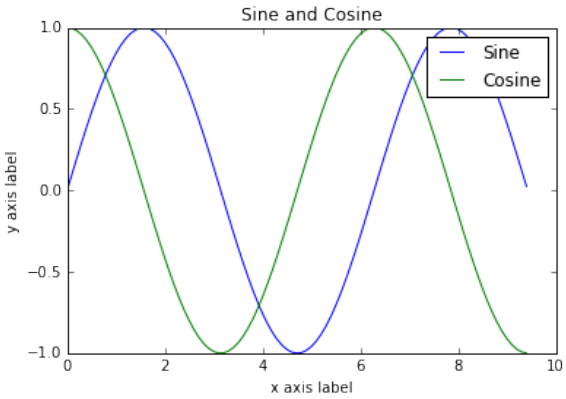
运行该代码可生成下图：

只需少量工作，就可以一次绘制多条曲线，并添加标题、图示和轴标：

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

# 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()
```

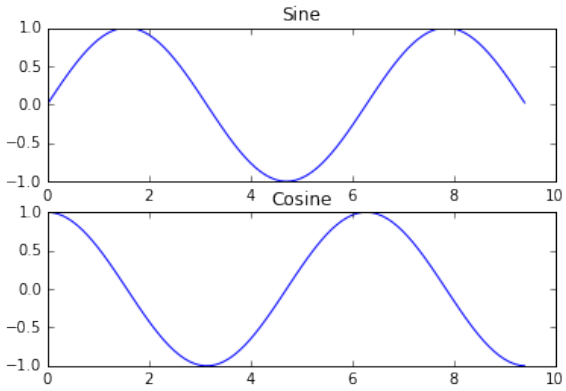


4.2 Subplots

可以使用 subplot 函数来在一幅图中画不同的东西:

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

# 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()
```



4.3 Images

你可以使用 imshow 函数来显示图像。例如:

```
import numpy as np
from scipy.misc import imread, imresize
import matplotlib.pyplot as plt

img = imread('assets/cat.jpg')
img_tinted = img * [1, 0.95, 0.9]

# Show the original image
plt.subplot(1, 2, 1)
plt.imshow(img)

# Show the tinted image
plt.subplot(1, 2, 2)

# A slight gotcha with imshow is that it might give strange results
# if presented with data that is not uint8. To work around this, we
# explicitly cast the image to uint8 before displaying it.
plt.imshow(np.uint8(img_tinted))
plt.show()
```


0
50
100
150
200
250
300
350
400
0
50
100
150
200
0
50
100
150
200
250
300
350
400
0
50
100
150
200