数据结构与算法

引言

张晓平

武汉大学数学与统计学院

2017 年 10 月 11 日

目录

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

目标

- 回顾计算机科学的思想，提高编程和解决问题的能力。
- 理解抽象化以及它在解决问题过程中发挥的作用
- 理解和实现抽象数据类型的概念

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

快速开始

- 计算技术的变化为计算机科学家提供了越来越多的工具和平台。
- 计算机的快速发展，诸如快速处理器、高速网络和大容量存储器，已经让计算机科学家陷入高度复杂螺旋中。

在所有这些快速演变中，有一些基本原则会保持不变。计算机科学关注用计算机来解决问题。

快速开始

问题以及解决方案的复杂性可能会掩盖求解过程中的一些基本思想。

快速开始

简要回顾计算机科学、算法和数据结构所必须适应的框架，研究其原因，以及如何理解它们以有助于更好的解决问题。

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

什么是计算机科学

所谓计算机科学，即

- 研究问题
- 解决问题
- 生成解决问题的方案

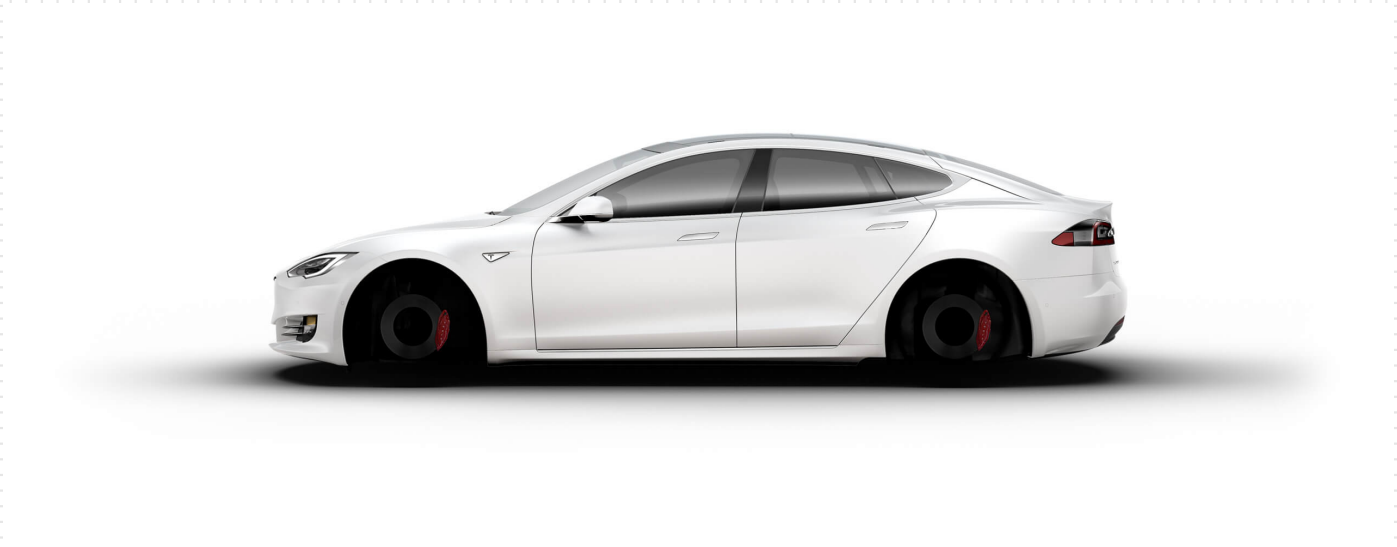
给定一个问题，目标是开发算法，编写程序，用于解决可能出现的问题。

什么是计算机科学

计算机科学，因涉及问题解决过程本身，是关于抽象的研究。抽象使我们能从逻辑视角和物理视角来分别看待问题及其解决方案。

什么是计算机科学

什么是计算机科学



- 作为司机，你会与车有一些互动，如上车、插钥匙、点火、换挡、制动、加速、转向、下车等。你只需要了解车一些基本功能，就可以操纵它。

  此时，你所看到的是汽车的“逻辑视角”，这些功能通常称为“接口”。

什么是计算机科学

- 作为司机，你会与车有一些互动，如上车、插钥匙、点火、换挡、制动、加速、转向、下车等。你只需要了解车一些基本功能，就可以操纵它。

  此时，你所看到的是汽车的“逻辑视角”，这些功能通常称为“接口”。

- 作为修车师傅，看待汽车的视角就会截然不同。你不仅需要知道如何开车，还必须知道汽车的一些内部细节，比如说发动机如何工作、变速箱如何变速、温度如何控制等等。

  这就是“物理视角”，细节发生在“引擎盖下”。

什么是计算机科学

什么是计算机科学

- 作为用户，你可以编写文档、收发邮件、上网冲浪、播放音乐、存储图像和玩游戏等，但你并不知道这些 APP 工作的细节。

  此时，你是从逻辑或用户角度看待计算机。

什么是计算机科学

- 作为用户，你可以编写文档、收发邮件、上网冲浪、播放音乐、存储图像和玩游戏等，但你并不知道这些 APP 工作的细节。

  此时，你是从逻辑或用户角度看待计算机。

- 作为计算机科学家、程序员、技术支持人员和系统管理员，你看待计算机的角度会截然不同。你必须知道操作系统如何工作、如何配置网络协议、如何编写控制功能的各种脚本。

  总之，你必须能够控制底层的细节。

什么是计算机科学

作为用户，你不需要知道细节，只需了解接口的工作方式。接口是用户与底层沟通的窗口。

什么是计算机科学

看一个抽象的例子—Python 数学模块。一旦导入模块，就可以执行计算

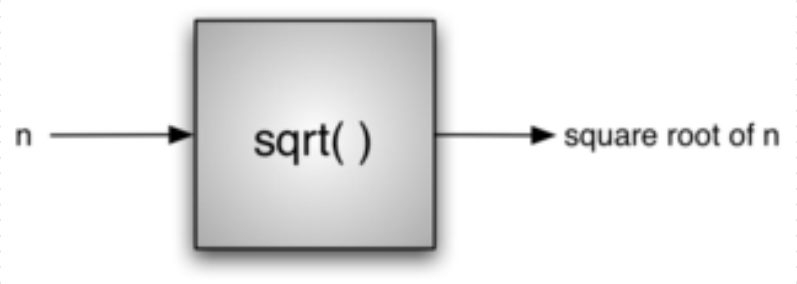
```
>>> import math
>>> math.sqrt(16)
4.0
```

什么是计算机科学

看一个抽象的例子—Python 数学模块。一旦导入模块，就可以执行计算

```
>>> import math
>>> math.sqrt(16)
4.0
```

你无需知道计算平方根的细节，只需知道 sqrt 函数的功能及其使用方式。这就像一个“黑盒子”，其接口可描述为：函数名、参数、返回值，其细节隐藏在内部。



1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

什么是编程

编程是将算法转换为程序语言的过程，以便能被计算机所执行。

- 首先要有解决方案，亦即算法；
- 然后在选择合适的编程语言实现算法。

计算机科学不研究编程，但编程却是计算机科学家的重要能力。编程通常是解决方案的表达方式。

什么是编程

- 算法描述了依据实际问题所生成的解决方案和产生预期结果所需要的一套步骤。
- 编程语言必须提供一种表示方法来表示对应的过程和数据。为此，它提供了控制结构和数据类型。

什么是编程

控制结构允许以方便而明确的方式表示算法步骤。至少，算法需要执行顺序处理、决策选择和控制迭代。只要语言提供这些基本语句，它就可以表达算法。

什么是编程

在计算机中，所有数据项都由一串一串的二进制数表示。为了让这些二进制串有意义，就需要有数据类型。

- 数据类型为二进制数据提供解释，以便我们能够根据实际问题来思考数据。这些底层的内置数据类型（有时称为原始数据类型）为算法开发提供了基础。

什么是编程

在计算机中，所有数据项都由一串一串的二进制数表示。为了让这些二进制串有意义，就需要有数据类型。

- 数据类型为二进制数据提供解释，以便我们能够根据实际问题来思考数据。这些底层的内置数据类型（有时称为原始数据类型）为算法开发提供了基础。

  例如，大多数编程语言提供整数类型。内存中的二进制数据可解释为整数，并且给予一个与整数（如 23, 654 和 -19）相关联的含义。

什么是编程

在计算机中，所有数据项都由一串一串的二进制数表示。为了让这些二进制串有意义，就需要有数据类型。

- 数据类型为二进制数据提供解释，以便我们能够根据实际问题来思考数据。这些底层的内置数据类型（有时称为原始数据类型）为算法开发提供了基础。

  例如，大多数编程语言提供整数类型。内存中的二进制数据可解释为整数，并且给予一个与整数（如 23, 654 和 -19）相关联的含义。

- 此外，数据类型还提供数据项所参与操作的描述。对于整数，提供诸如加法、减法和乘法的操作。

什么是编程

然而，通常我们遇到的困难是问题及其解决方案非常复杂。由语言提供的简单的结构和数据类型，虽然可以表示复杂的解决方案，但在实际中却不好用。我们需要一些方法控制这种复杂性，以助于形成更好的解决方案。

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

为什么要学习数据结构和抽象数据类型

为管理问题的复杂性及解决过程，计算机科学家使用抽象使他们能够专注于“大局”而不会迷失在细节中。

通过对问题进行建模，我们能够更好和更有效地解决问题。

这些模型允许我们以更加一致的方式来描述我们的算法。

为什么要学习数据结构和抽象数据类型

- 过程抽象：隐藏特定函数的细节，以允许用户或客户端在高层查看它。

为什么要学习数据结构和抽象数据类型

- 过程抽象：隐藏特定函数的细节，以允许用户或客户端在高层查看它。
- 数据抽象，即抽象数据类型 (ADT)：对数据和允许操作的逻辑描述，不用考虑如何实现它们。

  这意味着我们只关心数据表示什么，而不关心它最终将如何构造。通过提供这种级别的抽象，我们围绕数据创建一个封装。通过封装实现细节，我们将它们从用户的视图中隐藏。这称为信息隐藏。

为什么要学习数据结构和抽象数据类型

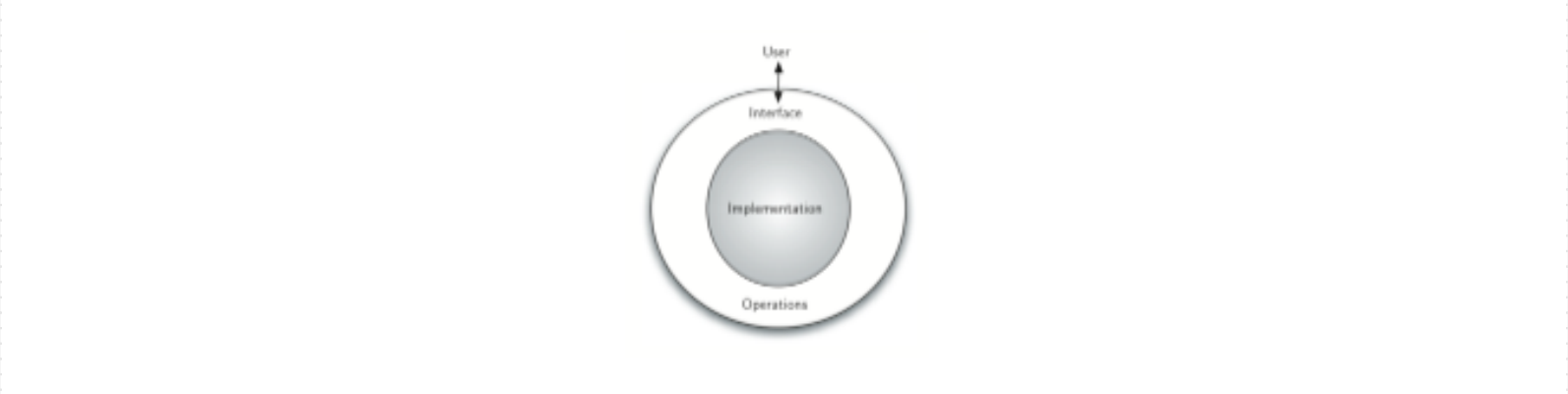


图: 展示了 ADT 是什么以及如何操作。用户与接口交互，使用抽象数据类型指定的操作。抽象数据类型是用户与之交互的 shell。实现隐藏在更深的底层。用户不关心实现的细节。

为什么要学习数据结构和抽象数据类型

ADT 的实现要求我们使用一些程序构建和原始数据类型的集合来提供数据的物理视图。

“逻辑”与“物理”两个视角的分离，允许我们将问题定义复杂的数据模型，而不给出关于模型如何实际构建的细节。这提供了独立于实现的数据视图。

由于通常有许多不同的方法来实现抽象数据类型，所以这种实现独立性允许程序员在不改变数据的用户与其交互的方式的情况下切换实现的细节。

用户可以继续专注于解决问题的过程。

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

为什么要学习算法

计算机科学家经常通过经验学习。我们通过看别人解决问题和自己解决问题来学习。接触不同的问题解决技术，看不同的算法设计有助于我们面对下一个具有挑战性的问题。通过思考许多不同的算法，我们可以开始开发模式识别，以便下一次出现类似的问题时，我们能够更好地解决它。

为什么要学习算法

算法通常彼此完全不同。例如，对于 sqrt，完全可能存在很多不同的方式来实现其细节。

自然地，我们建议使用一个更高效，或者一个只是工作更快或使用更少的内存的算法。

为什么要学习算法

作为计算机科学家，除了需要具备解决问题的能力，还需要掌握解决方案的评估技术。

通常有很多方法来解决问题，所以当我们找到一个解决方案时，需要一遍又一遍地进行比较，然后决定它是否是一个好的方案。

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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。

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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)               # print "[[ 0.91940167  0.08143941]
                       #          [ 0.68744134  0.87236687]]"
```

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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。

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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"
```

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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 0.9682458365518542 6
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]"
```

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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]]"
```

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,)，`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。

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。

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!

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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。

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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)
```

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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。

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

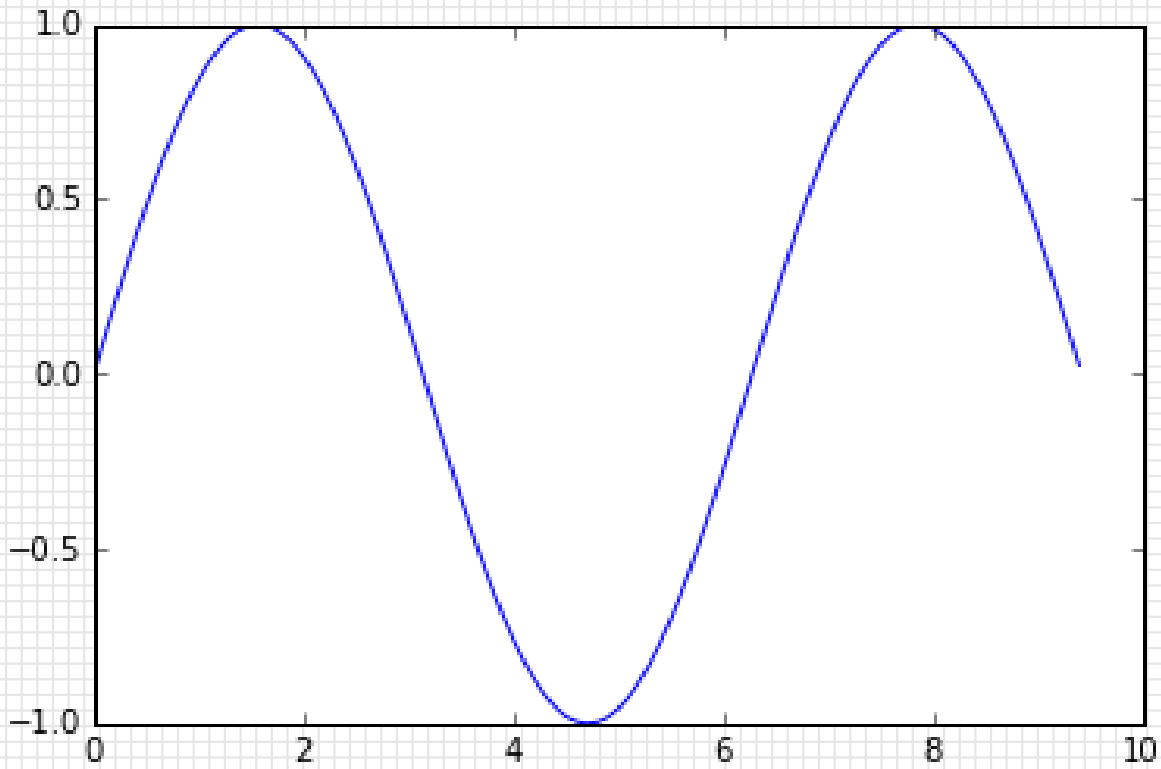
Matplotlib 是一个绘图库。这里简要介绍 matplotlib.pyplot 模块，功能和 MATLAB 的绘图功能类似。

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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：

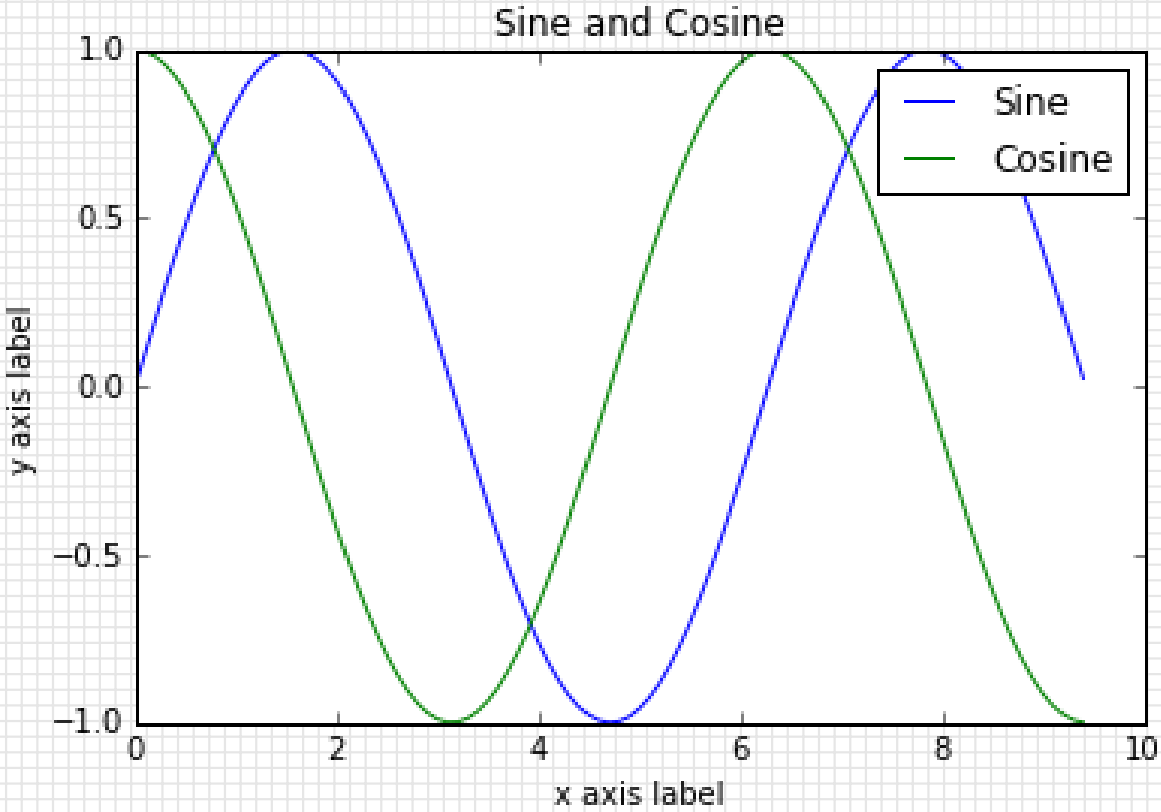
```
import numpy as np
import matplotlib.pyplot as plt

# 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：

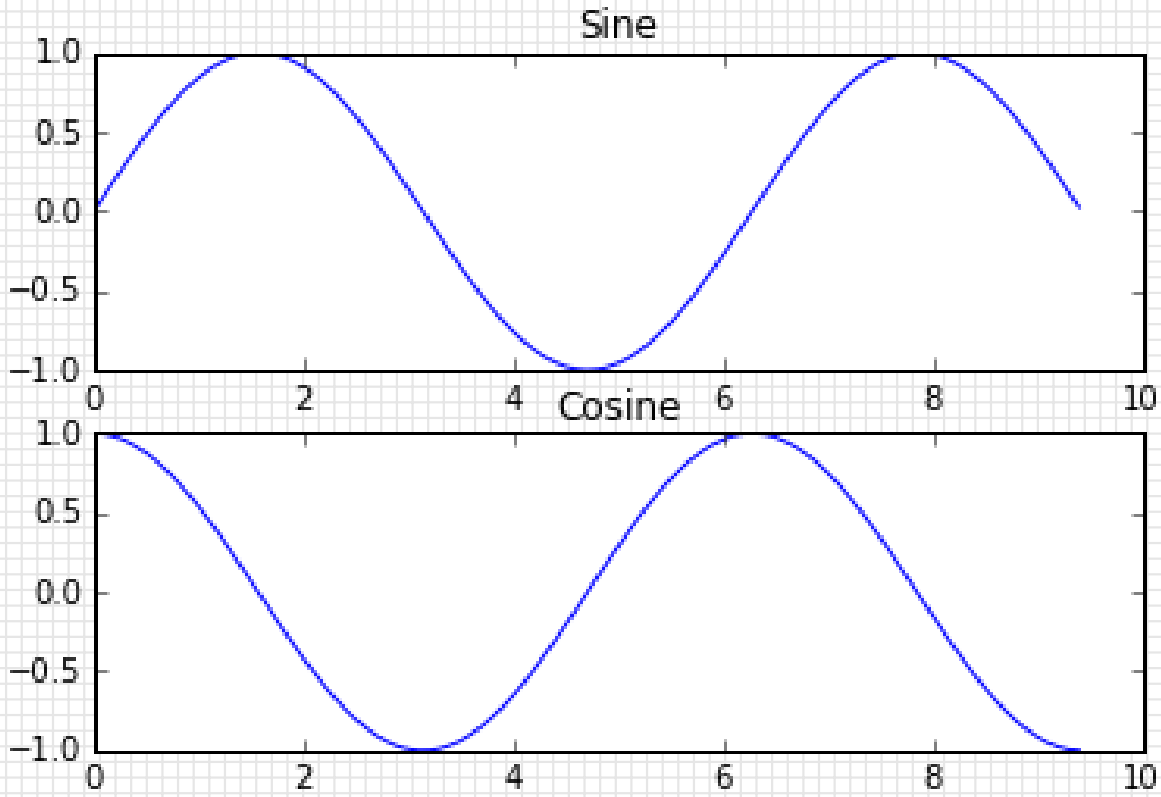
```
import numpy as np
import matplotlib.pyplot as plt

# 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()
```


Sine and Cosine
Sine
Cosine
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
y axis label
x axis label

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - Images

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()
```


Sine
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
Cosine
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10

1. 目标
2. 快速开始
3. 什么是计算机科学
4. 什么是编程
5. 为什么要学习数据结构和抽象数据类型
6. 为什么要学习算法
7. Numpy
   - 数组
   - 访问数组
   - 数据类型
   - Array math
   - 广播机制（Broadcasting）
8. SciPy
   - 图像操作
   - MATLAB 文件
   - 点之间的距离
9. Matplotlib
   - Plotting
   - Subplots
   - 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