Python 和相关的包简介

Dengwen Zhou

控制与计算机工程学院

September 27, 2020

Python

- 高级动态类型的多范式编程语言
- 几乎类似于伪代码
- 很少的几行代码可表达强大的思想，又很易读

Basic data types I

Numbers: *Integers* and *floats* work as other languages:

```
x = 3
print(type(x)) # ==> <class 'int'>
print(x)       # ==> 3
print(x + 1)   # 加 ==> 4
print(x - 1)   # 减 ==> 2
print(x * 2)   # 乘 ==> 6
print(x ** 2)  # 指数 ==> 9
x += 1
print(x)  # ==> 4
x *= 2; print(x)  # ==> 8
y = 2.5
print(type(y)) # ==> <class 'float'>
print(y, y + 1, y * 2, y ** 2) # ==> 2.5 3.5 5.0 6.25
```

Basic data types II

- Python 没有单增 (x++) or 单减 (x--) 运算符
- Python 支持复数类型: `x + yj`, `3 + 6j`

Booleans: Python 布尔运算符用单词 (非 &&, || 等符号):

```python
t = True
f = False
print(type(t))  # ==> <class 'bool'>
print(t and f)  # 逻辑与 ==> False
print(t or f)   # 逻辑或 ==> True
print(not t)    # 逻辑非 ==> False
print(t != f)   # 逻辑异或 ==> True
```

Strings: Python has great support for strings:

Basic data types III

```
hello = 'hello' # 字符串可以使用单引号
world = "world" # 或双引号
print(hello) # ==> hello
print(len(hello)) # 字符串长度 ==> 5
hw = hello + ' ' + world  # 字符串连接
print(hw) # ==> hello world
hw12 = '%s %s %d' % (hello, world, 12) # sprintf风格的字符串格式化
print(hw12) # ==> hello world 12
```

String objects have a bunch of *useful methods*; for example:

Basic data types IV

```
s = "hello"
print(s.capitalize()) # 字符串首字母大写 ==> Hello
print(s.upper()) # 字符串转换为大写 ==> HELLO
print(s.rjust(7)) # 右对齐字符串，并用空格填充 ==> "  hello"
print(s.center(7)) # 字符串居中，并用空格填充 ==> " hello "
print(s.replace('l', '(ell)')) # 一个子串替换另一个子串 ==> he(ell)(ell)o
print('  world '.strip()) # 删除前和尾部空格 ==> "world"
```

Containers(容器)

Python includes several built-in ***container*** (容器) types:

Lists (表) I

Python 表相当于数组, 但可调整大小, 及包含不同类型的元素:

```
xs = [3, 1, 2] # 创建一个表
print(xs, xs[2]) # ==> [3, 1, 2] 2
print(xs[-1]) # 负索引从表的末尾计数 ==> 2
xs[2] = 'foo' # 表可以包含不同类型的元素
print(xs) # ==> [3, 1, 'foo']
xs.append('bar') # 表尾添加一个新元素
print(xs) # ==> [3, 1, 'foo', 'bar']
x = xs.pop() # 删除并返回表的最后一个元素
print(x, xs) # ==> bar [3, 1, 'foo']
```

Slicing (切片): 访问子表 *(sublist)*, 称之为切片 *(slicing)*:

Lists (表) II

```python
nums = list(range(5)) # range是创建整数表的内置函数
print(nums) # ==> [0,1,2,3,4]
print(nums[2:4]) # 从索引2到4(不包括)的切片: [2, 3]
print(nums[2:]) # 从索引2到末尾的切片: [2,3,4]
print(nums[:2]) # 从开始到索引2(不包含)的切片: [0,1]
print(nums[:]) # 整个表的切片: [0,1,2,3,4]
print(nums[:-1]) # 切片索引可为负: [0,1,2,3]
nums[2:4] = [8, 9] # 切片赋值为一个新子表
print(nums) # ==> [0,1,8,9,4]
```

Loops (循环): 遍历表元素:

Lists (表) III

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
# 分行打印: "cat", "dog", 和 "monkey"
```

List comprehensions (表解析):

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

更简单的表解析代码:

Lists (表) IV

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

表解析可以包含条件:

```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]
```

Dictionaries (字典) I

A `dictionary` stores (`key`, `value`) pairs, similar to a `Map` in Java:

```python
d = {'cat': 'cute', 'dog': 'furry'} # 创建一个有些数据的新字典
print(d['cat']) # 读一个字典项 ==> cute
print('cat' in d) # 检查字典是否有给定的键 ==> True
d['fish'] = 'wet' # 设置一个字典项
print(d['fish']) # ==> wet
# print(d['monkey']) # KeyError: 'monkey'不是d的键
print(d.get('monkey', 'N/A')) # 读有默认值的元素 ==> N/A
print(d.get('fish', 'N/A')) # 读有默认值的元素 ==> wet
del d['fish'] # 删除字典元素
```

Dictionaries (字典) II

```python
print(d.get('fish', 'N/A')) # "fish" 不再是键 ==> N/A
```

Loops (循环): It is easy to iterate over the *keys* in a *dictionary*:

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

访问 keys 和对应 values, use the `items` method:

Dictionaries (字典) III

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

Dictionary comprehensions (字典解析): 类似于表解析, 易于构造字典:

```
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)  # ==> {0: 0, 2: 4, 4: 16}
```

Sets (集合) I

A **set** is an unordered collection of distinct elements. A simple example:

```
animals = {'cat', 'dog'}
print('cat' in animals) # 检查元素是否在集合中 ==> True
print('fish' in animals) # ==> False
animals.add('fish') # 在集合中增加一个元素
print('fish' in animals) # ==> True
print(len(animals)) # 集合中元素数目 ==> 3
animals.add('cat') # 添加已在集合中的元素 ==> 不执行任何操作
print(len(animals)) # ==> 3
animals.remove('cat') # 删除集合中一个元素
print(len(animals)) # ==> 2
```

Sets (集合) II

Loops (循环): 语法与表相同, 但是, 集合是无序的:

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

Set comprehensions (集合解析): 类似表和字典, 利用集合解析, 易于构造集合:

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

Tuples (元组)

A `tuple` is an (immutable 不可变的) ordered list of values, 与表类似. 最重要的差别: 元组可用作字典键和集合元素, 而表不可以

```
d = {(x, x + 1): x for x in range(10)}  # 用元组键创建字典
t = (5, 6)              # 创建一个元组
print(type(t))          # ==> <class 'tuple'>
print(d[t])             # ==> 5
print(d[(1, 2)])        # ==> 1
```

Functions (函数) I

Python *functions* are defined using the `def` keyword:

```
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"
```

允许可选的关键字参数:

Functions (函数) II

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

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

Classes (类)

```python
class Greeter(object):

    def __init__(self, name): # 构造器(函数)
        self.name = name # 创建一个实例变量

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

g = Greeter('Fred') # 构造一个Greeter类的实例
g.greet() # 调用实例方法 ==> Hello, Fred
g.greet(loud=True) # 调用实例方法 ==> HELLO, FRED!
```

Numpy

- Python 科学计算的核心库
- 提供了高性能的多维数组对象，以及数组处理工具
- 类似于 MATLAB

Arrays I

- A numpy array is a *grid of values*
- Numpy 数组元素类型均相同
- Numpy 数组元素的索引是非负整数元组
- Numpy 数组维数称为数组的秩 (rank)
- 整数元组给出数组形状 (即每一维的大小)

可用嵌套的 Python 表初始化 numpy 数组，方括号访问元素:

Arrays II

```python
import numpy as np

a = np.array([1, 2, 3]) # 创建一个1维数组
print(type(a)) # ==> <class 'numpy.ndarray'>
print(a.shape) # ==> (3,)
print(a[0], a[1], a[2]) # ==> 1 2 3
a[0] = 5 # 修改数组元素值
print(a) # ==> [5, 2, 3]

b = np.array([[1,2,3],[4,5,6]]) # 创建一个2维数组
print(b.shape) # ==> (2, 3)
print(b[0, 0], b[0, 1], b[1, 0]) # ==> 1 2 4
```

Numpy also provides many *functions* to create arrays:

Arrays III

```python
import numpy as np

a = np.zeros((2,2)) # 创建零数组
print(a) # => [[ 0.   0.]
         #     [ 0.   0.]]

b = np.ones((1,2)) # 创建1数组
print(b) # => [[ 1.   1.]]

c = np.full((2,2), 7) # 创建常量数组
print(c) # => [[ 7.   7.]
         #     [ 7.   7.]]

d = np.eye(2) # 创建一个2×2的单位矩阵

```

Arrays IV

```
print(d) # ==> [[ 1.   0.]
         #      [ 0.   1.]]

e = np.random.random((2,2)) # 创建随机值数组
print(e) # 可能==>[[ 0.91940167   0.08143941]
         #         [ 0.68744134   0.87236687]]
```

Array indexing(数组索引) I

Numpy offers several ways to index into arrays.

Slicing(切片): 与 Python 表类似, 必须为数组的每个维指定一个切片:

```
import numpy as np

# 创建(3, 4)的2维数组
# [[ 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]])

# 切片提取数组前2行和1、2列 ==> (2, 2)的b数组:
# [[2  3]
#  [6  7]]
```

Array indexing(数组索引) II

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

# 数组切片是相同数据视图，修改视图将修改原数组
print(a[0, 1])     # ==> 2
b[0, 0] = 77       # b[0, 0]与a[0, 1]数据相同
print(a[0, 1])     # ==> 77
```

整数索引与切片索引可以混合使用:

```
import numpy as np

# 创建(3, 4)的2维数组
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
```

Array indexing(数组索引) III

```python
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# 两种方式访问数组中间行的数据
# 整数索引与切片混合(比原数组维数低)
# 仅使用切片(与原数组维数相同):
row_r1 = a[1, :]    # a的第二行视图(1维)
row_r2 = a[1:2, :]  # a的第二行视图(2维)
print(row_r1, row_r1.shape) # ==> [5 6 7 8] (4,)
print(row_r2, row_r2.shape) # ==> [[5 6 7 8]] (1, 4)

# 类似地，访问数组的列
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
```

Array indexing(数组索引) IV

```
print(col_r1, col_r1.shape) # ==>[ 2  6 10] (3,)
print(col_r2, col_r2.shape) # ==>[[ 2]
                            #     [ 6]
                            #     [10]] (3, 1)
```

Integer array indexing(整数数组索引): 切片视图是与原数组维数相同的子数组, 整数数组索引可以构造任意数组:

```
import numpy as np

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

# 整数数组索引实例: 返回的数组形状为(3,)
print(a[[0, 1, 2], [0, 1, 0]]) # ==> [1 4 5]

```

Array indexing(数组索引) V

```
# 上面的整数数组索引示例等价于:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
    # ==> [1 4 5]

# 使用整数数组索引, 可重用源数组中的相同元素:
print(a[[0, 0], [1, 1]]) # ==> [2 2]

# 等价于前面的整数数组索引示例
print(np.array([a[0, 1], a[0, 1]])) # ==>[2 2]
```

整数数组索引的一个技巧: 选择或修改矩阵每一行中的一个元素:

Array indexing(数组索引) VI

```python
import numpy as np

# 创建一个数组
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print(a) # ==> array([[ 1,  2,  3],
         #            [ 4,  5,  6],
         #            [ 7,  8,  9],
         #            [10, 11, 12]])

# 创建索引数组
b = np.array([0, 2, 0, 1])

# 使用b中的索引，从a的每一行中选择一个元素
```

Array indexing(数组索引) VII

```
print(a[np.arange(4), b])  # ==> [ 1  6  7 11]

# 使用b中的索引，修改a的每一行中的一个元素
a[np.arange(4), b] += 10

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

Boolean array indexing(布尔数组索引): 布尔数组索引可选择数组的任意元素:

Array indexing(数组索引) VIII

```python
import numpy as np

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

bool_idx = (a > 2)  # 查找大于2的元素；
                    # 返回一个布尔数组，形状同a

print(bool_idx)  # ==> [[False False]
                 #      [ True  True]
                 #      [ True  True]]

# 使用布尔数组索引构建1维数组，
# 由与bool_idx的True值对应的元素组成
print(a[bool_idx])  # ==> [3 4 5 6]

```

Array indexing(数组索引) IX

```
# 可在一个简洁的语句中完成上述所有操作:
print(a[a > 2]) # ==> [3 4 5 6]
```

Datatypes

在创建数组时, Numpy 会猜测一个数据类型, 但是, 构造数组的函数, 包括一个可选参数, 以显式指定数据类型:

```
import numpy as np

x = np.array([1, 2]) # 让numpy选择数据类型
print(x.dtype) # ==> int64

x = np.array([1.0, 2.0]) # 让numpy选择数据类型
print(x.dtype) # ==> float64

x = np.array([1, 2], dtype=np.int64) # 强制使用特定数据类型
print(x.dtype) # ==> int64
```

Array math(数组数学) I

- 基本数学函数在数组上逐元素运算
- 可用作运算符重载和 numpy 模块中的函数

```
import numpy as np

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

# 逐元素求和，生成数组：
# [[ 6.0   8.0]
#  [10.0  12.0]]
print(x + y)
print(np.add(x, y))

# 逐元素减，生成数组：
```

Array math(数组数学) II

```
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# 逐元素乘，生成数组：
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# 逐元素除，生成数组：
# [[ 0.2         0.33333333]
#  [ 0.42857143 0.5       ]]
print(x / y)
print(np.divide(x, y))
```

Array math(数组数学) III

```
# 逐元素平方根，生成数组：
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

- `*`：**Numpy** ==> 逐元素乘; **Matlab** ==> 矩阵乘
- Numpy 中, `dot`函数 ==> 向量内积、向量与矩阵乘, 以及矩阵乘
- `dot`可用作 *Numpy* 函数, 也可用作数组对象的实例方法

Array math(数组数学) IV

```
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])

# 向量内积 ==> 219
print(v.dot(w))
print(np.dot(v, w))

# 矩阵/向量积 ==> 1维数组[29 67]
print(x.dot(v))
print(np.dot(x, v))
```

Array math(数组数学) V

```python

# 矩阵/矩阵 ==> 2维数组
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

Numpy 有许多有用的数组计算函数，最有用的一个是 `sum`:

```python
import numpy as np

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

print(np.sum(x)) # 计算所有元素的和 ==> 10
print(np.sum(x, axis=0)) # 求各列和 ==> [4 6]
print(np.sum(x, axis=1)) # 求各行和 ==> [3 7]
```

Array math(数组数学) VI

转置矩阵: 使用数组对象的 `T` 属性:

```
import numpy as np

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

# 1维数组转置, 不执行任何运算
v = np.array([1,2,3])
print(v)    # ==> [1 2 3]
print(v.T)  # ==> [1 2 3]
```

Broadcasting(广播) I

广播 (Broadcasting) 是一种强大的机制, 允许 numpy 对不同形状的数组运算

向矩阵的每一行添加一个常数向量:

```
import numpy as np

# 向量v添加到矩阵x的每一行, 结果存储在矩阵y中
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) # 创建与x形状相同的空矩阵

# 通过显式循环, 向量v加到矩阵x的每一行
for i in range(4):
    y[i, :] = x[i, :] + v

```

Broadcasting(广播) II

```
# 现在y是
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]
print(y)
```

Python 显式循环很慢, 替代实现:

```
import numpy as np

# 向量v添加到矩阵x的每一行, 结果存储在矩阵y中
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))   # v堆积4次
```

Broadcasting(广播) III

```
print(vv)                        # ==> [[1 0 1]
                                 #      [1 0 1]
                                 #      [1 0 1]
                                 #      [1 0 1]]
y = x + vv  # x和vv逐元素相加
print(y)  # ==> [[ 2  2  4
          #       [ 5  5  7]
          #       [ 8  8 10]
          #       [11 11 13]]
```

Numpy 广播, 更好的实现:

Broadcasting(广播) IV

```python
import numpy as np

# 向量v添加到矩阵x的每一行，结果存储在矩阵y中
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10,
    11, 12]])
v = np.array([1, 0, 1])
y = x + v # 使用广播，v加到x的每一行
print(y)   # ==> [[ 2  2  4]
           #      [ 5  5  7]
           #      [ 8  8 10]
           #      [11 11 13]]
```

广播规则:

Broadcasting(广播) V

- **规则 1**: 如果两个数组的维数不同, 则维数较小的数组, 在其形状左侧扩充尺寸为 *1* 的维度, 使之与维数较大的数组有相同的维数
- **规则 2**: 如果两个数组的形状在任何维度上都不匹配, 则将在该维度上拉伸形状等于 1 的数组, 以匹配另一个数组的形状
- **规则 3**: 如果两个数组尺寸在任何维度上都不相同, 且没有形状等于 1 的尺寸, 则不能广播 (也就是说, 两个数组对应的维度: 形状尺寸要么相等, 要么其中一个为 1)

支持广播的函数称为通用函数

广播的一些应用:

Broadcasting(广播) VI

```
import numpy as np

# 计算向量的外积
v = np.array([1,2,3]) # v的形状(3,)
w = np.array([4,5]) # w的形状(2,)
# 要计算外部乘积，首先将v调整为形状(3,1)的列向量. 生成形状是(3,2)的v和w的外积:
# [[ 4  5]
#  [ 8 10]
#  [12 15]]
print(np.reshape(v, (3, 1)) * w)

# 向量加到矩阵的每一行
x = np.array([[1,2,3], [4,5,6]])
```

Broadcasting(广播) VII

```
# x的形状为(2,3)，而v的形状为(3,). 它们广播到(2,3):
# [[2 4 6]
#  [5 7 9]]
print(x + v)

# 向量加到矩阵的每一列
# x的形状为(2,3)，w的形状为(2,)
# 如果对x进行转置，则它的形状为(3,2)，广播结果形状(3,2)
# 转置此结果 ==> 向量w被加到矩阵x每一列、形状为(2, 3)的最终结果:
# [[ 5  6  7]
#  [ 9 10 11]]
print((x.T + w).T)
```

Broadcasting(广播) VIII

```
# 另一个解决方案是将w形状调整为(2,1)的列向量，
    可生成相同的输出
print(x + np.reshape(w, (2, 1)))

# 矩阵乘以常数：
# x的形状为(2,3)，产生以下数组：
# [[ 2  4  6]
#  [ 8 10 12]]
print(x * 2)
```

SciPy

SciPy 提供了许多在 *numpy* 数组上运行的有用的函数

Image operations I

SciPy 提供了一些处理图像的基础函数:

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

# JPEG图像读入numpy数组
img = imread('assets/cat.jpg')
print(img.dtype, img.shape)  # ==> uint8 (400, 248, 3)

# 可以使用不同的标量常数缩放每个颜色通道，为图像着色. 图像形状为(400,248,3)，将其乘以形状(3,)的数组[1,0.95,0.9]
# numpy广播：保持红色通道不变，绿色和蓝色通道分别乘以0.95和0.9
img_tinted = img * [1, 0.95, 0.9]

```

Image operations II

```
# 着色图像的大小调整为300 × 300像素
img_tinted = imresize(img_tinted, (300, 300))

# 着色图像写到磁盘
imsave('assets/cat_tinted.jpg', img_tinted)
```

MATLAB files

The functions `scipy.io.loadmat` and `scipy.io.savemat`, 可以读写 MATLAB 文件

Distance between points I

函数 `scipy.spatial.distance.pdist` 计算给定点集中，所有点对之间的距离:

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

# 创建以下数组，每一行是2D空间中的一个点:
# [[0 1]
#  [1 0]
#  [2 0]]
x = np.array([[0, 1], [1, 0], [2, 0]])
print(x)

# 计算x所有行之间的欧几里得距离
# d[i,j]是x[i,:]和x[j,:]之间的欧几里得距离，d数组如下:
```

Distance between points II

```
# [[ 0.          1.41421356  2.23606798]
#  [ 1.41421356  0.          1.        ]
#  [ 2.23606798  1.          0.        ]]
d = squareform(pdist(x, 'euclidean'))
print(d)
```

类似的函数 `scipy.spatial.distance.cdist`, 计算两个点集之间, 所有点对之间的距离

Matplotlib

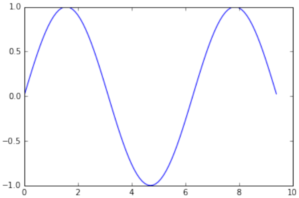
Matplotlib 是一个绘图库: *matplotlib.pyplot* 模块, 与 MATLAB 的绘图系统类似

Plotting I

plot 是 matplotlib 中最重要绘图函数，能够绘制 2D 数据:

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

# 计算正弦曲线上点的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# 用matplotlib绘制点
plt.plot(x, y)
plt.show()  # 必须调用plt.show()才能显示图形
```

Plotting II

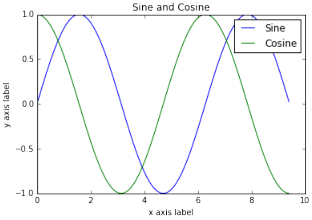


绘制多条线, 并添加标题、图例和轴标签:

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

# 计算正弦和余弦曲线上的点的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

```

Plotting III

```
# 用matplotlib绘制点
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()
```

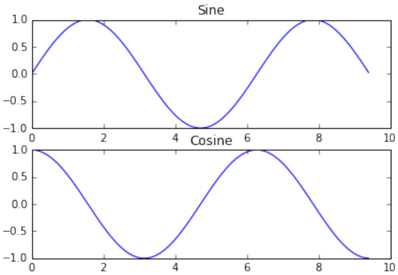


Subplots(子图) I

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

# 计算正弦和余弦曲线上的点的x和y坐标
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# 设置高为2、宽为1的子图网格，并将第1个子图设置为活动状态
plt.subplot(2, 1, 1)

# 绘制第一个子图
plt.plot(x, y_sin)
plt.title('Sine')

```

Subplots(子图) II

```
# 第2个子图设置为活动状态，并绘制第2个子图
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# 显示图形
plt.show()
```



```python
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]

# 显示原图像
plt.subplot(1, 2, 1)
plt.imshow(img)

# 显示着色图像
plt.subplot(1, 2, 2)

# 显示图像之前，图像显式转换为uint8
plt.imshow(np.uint8(img_tinted))
```

Images II

```
17   plt.show()
```