Python语言基础与应用-03

陈斌 gischen@pku.edu.cn

目录

- 面向对象
- 异常
- 迭代器和生成器
- 基本模块介绍
  （PIL / Flask / numpy / matplotlib / raspberry-gpio）

面向对象：什么是对象？

- Python中的所有事物都是以对象形式存在，从简单的数值类型，到复杂的代码模块，都是对象。
- 对象以id作为标识，既包含数据（属性），也包含代码（方法）

  赋值语句给予对象以名称，对象可以有多个名称（变量引用），但只有一个id
- 同一类（class）的对象具有相同的属性和方法，但属性值和id不同
- 对象实现了属性和方法的封装，是一种数据抽象机制

```
>>> id(1)
4297537952
>>> type(1)
<class 'int'>
>>> dir(1)
['__abs__', '__add__', '__class__', '__contains__', '__
r__', '__dir__', '__divmod__', '__doc__', '__eq__', '__format__', '__ge__', '__getattrib
_floordiv__', '__ge__', '__getitem__', '__gt__', '__hash__', '__init__', '__it
t__', '__hash__', '__index__', '__init__', '__int__', '__invert__', '__le__', '__len__',
_lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__neg__',
r__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__re
 '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__',
l__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__
', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__
, '__subclasshook__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag',
ugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

```
>>> id('a')
4300773280
>>> type('a')
<class 'str'>
>>> dir('a')
['__add__', '__class__', '__contains__', '__
_eq__', '__format__', '__ge__', '__getattrib
__', '__gt__', '__hash__', '__init__', '__it
__', '__mod__', '__mul__', '__ne__', '__new__',
epr__', '__rmod__', '__rmul__', '__setattr__
asshook__', 'capitalize', 'casefold', 'cente
expandtabs', 'find', 'format', 'format_map',
ecimal', 'isdigit', 'isidentifier', 'islower
ace', 'istitle', 'isupper', 'join', 'ljust',
                                            'rju
                                            'str
```

```
>>> abs(-1)
1
>>> id(abs)
4298931872
>>> type(abs)
<class 'builtin_function_or_method'>
>>> dir(abs)
['__call__', '__class__', '__delattr__', '__
at__', '__ge__', '__getattribute__', '__gt__
 '__lt__', '__module__', '__name__', '__ne__
uce__', '__reduce_ex__', '__repr__', '__self
_str__', '__subclasshook__', '__text_signatu
```

面向对象：类的定义与调用

- 类是对象的模版，封装了对应现实实体的性质和行为
- 定义类：class语句；

```
class <类名>:
    def __init__(self, <参数表>):
    def <方法名>(self, <参数表>):
```

- 调用类：<类名>（<参数>）

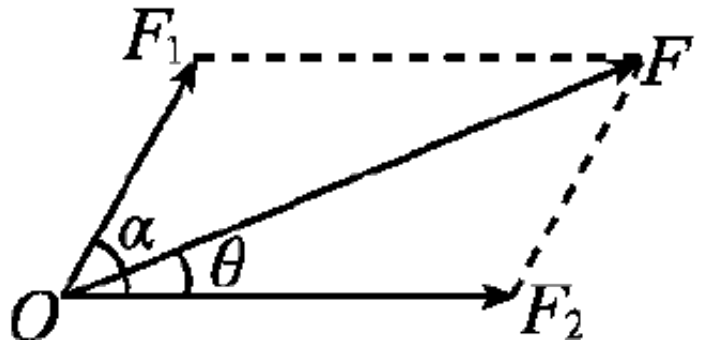
调用类会创建一个对象，（注意括号！）

```
obj = <类名> (<参数表>)
```

返回一个对象实例，

类方法中的self指这个对象实例！

```
1  class Force:  # 力
2      def __init__(self, x, y):  # x,y方向分量
3          self.fx, self.fy = x, y
4
5      def show(self):  # 打印出力的值
6          print("Force<%s,%s>" % (self.fx, self.fy))
7
8      def add(self, force2):  # 与另一个力合成
9          x = self.fx + force2.fx
10         y = self.fy + force2.fy
11         return Force(x, y)
12
13
14 # 生成一个力对象
15 f1 = Force(0, 1)
16 f1.show()
17
18 # 生成另一个力对象
19 f2 = Force(3, 4)
20 # 合成为新的力
21 f3 = f1.add(f2)
22 f3.show()
```




```
Force<0,1>
Force<3,5>
```

对象属性和方法的引用

- 通过<对象名>.<属性名>的形式引用，可以跟一般的变量一样用在赋值语句和表达式中
- Python语言动态的特征，使得对象可以随时**增加**或者**删除**属性或者方法。

```
44  print(f3.fx, f3.fy)
45  f3.fz = 3.4
46  print(f3.fz)
47  del f3.fz
```

```
0.0 4.5
3.4
```

类定义中的特殊方法

- 在类定义中实现一些特殊方法，可以方便地使用python一些内置操作

  所有特殊方法以两个下划线开始结束

  __str__(self): 自动转换为字符串

  __add__(self, other): 使用+操作符

  __mul__(self, other): 使用*操作符

  __eq__(self, other): 使用==操作符

- 其它特殊方法参见课程网站

  http://gis4g.pku.edu.cn/python-magic-method/

```
13      __add__ = add
14
15      def __str__(self):
16          return "F<%s,%s>" % (self.fx, self.fy)
17
18      def __mul__(self, n):
19          x, y = self.fx * n, self.fy * n
20          return Force(x, y)
21
22      def __eq__(self, force2):
23          return (self.fx == force2.fx) and \
24                 (self.fy == force2.fy)
```

```
37  # 操作符使用
38  f3 = f1 + f2
39  print("Fadd=%s" % (f3,))
40  f3 = f1 * 4.5
41  print("Fmul=%s" % (f3,))
42  print("%s==%s? -> %s" % (f1, f2, f1 == f2))
```

```
Fadd=F<3,5>
Fmul=F<0.0,4.5>
F<0,1>==F<3,4>? -> False
```

自定义对象的排序

- **Python列表类型的sort方法和内置排序函数sorted()**

  由于Python的可扩展性，每种数据类型可以定义特殊方法def __lt__(self, y)，返回True视为比y“小”，排在前，而返回False视为比y“大”，排在后

  任何自定义类都可以使用x<y这样的比较，只要类定义中定义了特殊方法__lt__

- **例子：Student**

  姓名，成绩

- **按照成绩排序**

  由高到低

- **用内置sort**

```
class Student:
    def __init__(self, name, grade):
        self.name, self.grade = name, grade

    # 内置sort函数只引用 < 比较符来判断前后
    def __lt__(self, other):
        # 成绩比other高的，排在他前面
        return self.grade > other.grade

    # Student的易读字符串表示
    def __str__(self):
        return "(%s,%d)" % (self.name, self.grade)

    # Student的正式字符串表示，我们让它跟易读表示相同
    __repr__ = __str__
```

Python可扩展的“大小”比较及排序

- 我们构造一个Python列表
- 在列表中加入Student对象
- 直接调用列表的sort方法
- 可以看到已经根据__lt__定义排序
- 直接检验Student对象的大小

  <
- 另外可以定义其它比较符

  __gt__等

```
# 构造一个Python List对象
s = list()

# 添加Student对象到List中
s.append(Student("Jack", 80))
s.append(Student("Jane", 75))
s.append(Student("Smith", 82))
s.append(Student("Cook", 90))
s.append(Student("Tom", 70))
print("Original:", s)

# 对List进行排序，注意这是内置sort方法
s.sort()

# 查看结果，已经按照成绩排好序
print("Sorted:", s)
```

```
============== RESTART: /Users/chenbin/Documents/homework/stu.py ==
Original: [(Jack,80), (Jane,75), (Smith,82), (Cook,90), (Tom,70)]
Sorted: [(Cook,90), (Smith,82), (Jack,80), (Jane,75), (Tom,70)]
>>> s[0]<s[1]
True
>>>
```

Python可扩展的“大小”比较及排序

- 我们可以把__lt__方法重新定义，改为比较姓名
- 这样sort方法就能按照姓名来排序

```
class Student:
    def __init__(self, name, grade):
        self.name, self.grade = name, grade

    # 内置sort函数只引用 < 比较符来判断前后
    def __lt__(self, other):
        # 姓名字母顺序在前，就排在他前面
        return self.name < other.name

    # Student的易读字符串表示
    def __str__(self):
        return "(%s,%d)" % (self.name, self.grade)

    # Student的正式字符串表示，我们让它跟易读表示相同
    __repr__ = __str__
```

```
============= RESTART: /Users/chenbin/Documents/homework/stu2.py =
Original: [(Jack,80), (Jane,75), (Smith,82), (Cook,90), (Tom,70)]
Sorted: [(Cook,90), (Jack,80), (Jane,75), (Smith,82), (Tom,70)]
>>> s[0]<s[1]
True
>>>
```

类的继承机制：代码复用

- 如果两个类具有“一般-特殊”的逻辑关系，那么特殊类就可以作为一般类的“子类”来定义，从“父类”继承属性和方法

```
class <子类名>(<父类名>):
    def <重定义方法>(self,…):
```

- 子类对象可以调用父类方法，除非这个方法在子类中重新定义了（覆盖override）

```
45  class Car:
46      def __init__(self, name):
47          self.name = name
48          self.remain_mile = 0
49
50      def fill_fuel(self, miles):  # 加燃料里程
51          self.remain_mile = miles
52
53      def run(self, miles):  # 跑miles英里
54          print (self.name, end=': ')
55          if self.remain_mile >= miles:
56              self.remain_mile -= miles
57              print("run %d miles!" % (miles,))
58          else:
59              print("fuel out!")
60
61
62  class GasCar(Car):
63      def fill_fuel(self, gas):  # 加汽油gas升
64          self.remain_mile = gas * 6.0  # 每升跑6英里
65
66
67  class ElecCar(Car):
68      def fill_fuel(self, power):  # 充电power度
69          self.remain_mile = power * 3.0  # 每度电3英里
```

```
71  gcar=GasCar("BMW")
72  gcar.fill_fuel(50.0)
73  gcar.run(200.0)
```

```
75  ecar=ElecCar("Tesla")
76  ecar.fill_fuel(60.0)
77  ecar.run(200.0)
```

```
BMW: run 200 miles!
Tesla: fuel out!
```

子类与父类

- 子类可以添加父类中没有的方法和属性
- 如果子类同名方法覆盖了父类的方法，仍然还可以调用父类的方法

```
class GasCar(Car):
    def __init__(self, name, capacity):  # 名称和排量
        super().__init__(name)  # 父类初始化方法，只有名称
        self.capacity = capacity  # 增加了排量属性
```

关于self

- 在类定义中，所有方法的首个参数一般都是self
- self实际上代表对象实例
- <对象>.<方法>(<参数>)
- 等价于：
- <类>.<方法>(**<对象>**, <参数>)
- 这里的对象就是self了
- 如右图Line81和82

```
79  gcar = GasCar("BMW")
80  gcar.fill_fuel(50.0)
81  gcar.run(200.0)
82  GasCar.run(gcar, 200.0)
```

上机练习

- **创建一个类People**
  - 包含属性name，city
  - 可以转换为字符串形式（__str__）
  - 包含方法moveto(self, newcity)
  - 可以按照city排序
  - 创建4个人对象，放到列表进行排序
- **创建一个类Teacher**
  - 是People的子类，新增属性school
  - moveto方法改为newschool
  - 按照school排序
  - 创建4个教师对象，放到列表进行排序
- **创建一个mylist类，继承自内置数据类型list（列表）**
  - 增加一个方法“累乘”product
  - def product(self):
  - 返回所有数据项的乘积。

例外处理Exception Handling

- 代码运行可能会意外各种错误：

  语法错误：Syntax Error

  除以0错误：ZeroDivisionError

  列表下标越界：IndexError

  类型错误：TypeError…

- 错误会引起程序中止退出，如果希望掌控意外，就需要在可能出错误的地方设置陷阱捕捉错误

  try: # 为缩进的代码设置陷阱

  except: # 处理错误的代码

  else: # 没有出错执行的代码

  finally: # 无论出错否，都执行的代码

```
1   try:
2       print('try...')
3       r = 10 / 'xyz'
4       print('result:', r)
5   except TypeError as e:
6       print('TypeError:', e)
7   except ZeroDivisionError as e:
8       print('ZeroDivisionError:', e)
9   else:
10      print('no error!')
11  finally:
12      print('finally...')
13  print('END')
```

```
try...
TypeError: unsupported operand type(s) for /: 'int' and 'str'
finally...
END
```

推导式

- 可以用来生成列表、字典和集合的语句

[<表达式> for <变量> in <可迭代对象> if <逻辑条件>]

{<键值表达式>:<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>}

{<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>}

```
>>> [x*x for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
>>> {'K%d'%(x,):x**3 for x in range(10)}
{'K2': 8, 'K8': 512, 'K5': 125, 'K6': 216, 'K3': 27, 'K9': 729, 'K0': 0,
'K7': 343, 'K1': 1, 'K4': 64}
>>>
>>> {x*x for x in range(10)}
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
>>>
>>> {x+y for x in range(10) for y in range(x)}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17}
```

推导式


```
>>> [x+y for x in range(10) for y in range(x)]
[1, 2, 3, 3, 4, 5, 4, 5, 6, 7, 5, 6, 7, 8, 9, 6, 7, 8, 9, 10, 11, 7, 8, 9
, 10, 11, 12, 13, 8, 9, 10, 11, 12, 13, 14, 15, 9, 10, 11, 12, 13, 14, 15
, 16, 17]
>>>
>>> [x*x for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
>>>
>>> [x.upper() for x in [1, 'abc', 'xyz', True] if isinstance(x, str)]
['ABC', 'XYZ']
```

生成器推导式

- 与推导式一样语法：

  (<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>)

- 返回一个生成器对象，也是可迭代对象

- 但生成器并不立即产生全部元素，仅在要用到元素的时候才生成，可以极大节省内存

```
>>> agen = (x*x for x in range(10))
>>> agen
<generator object <genexpr> at 0x1078f5620>
>>> for n in agen:
        print (n)


0
1
4
9
16
25
36
```

生成器函数

- 如果生成器比较复杂，一行表达式无法容纳，可以定义生成器函数
- 生成器函数的定义与普通函数相同，只是将return换成了yield
- yield会立即返回一个值
- 但在下一次迭代生成器函数的时候，会从yield语句后的语句**继续执行**，直到再次yield返回，或终止
- return语句则不同，它会终止函数的执行，下次调用会重新执行函数

```
def even_number(max):
    n = 0
    while n < max:
        yield n
        n += 2

for i in even_number(10):
    print (i)
```

```
============== RESTART:
0
2
4
6
8
>>>
```

上机练习

- 编写程序，输入两个数，输出它们的商，采用例外处理来处理两种错误，给出用户友好的提示信息
  1）除数为0
  2）输入了非数值
- 编写一个推导式，生成包含100以内所有勾股数(i,j,k)的列表
- 编写一个生成器函数，能够生成斐波那契数列

```
def fib():
    ….
for fn in fib():
    print (fn)
    if fn>1000:
        break
```

PIL：图像处理库

- Python 3安装Pillow
- Python上事实标准库
- 功能强大，可以对图像做各种处理
- 如：缩放、裁剪、旋转、滤镜、文字、调色板等等

PIL缩放图像操作


```
from PIL import Image

# 打开一个jpg图像文件，注意是当前路径:
im = Image.open('test.jpg')
# 获得图像尺寸:
w, h = im.size
print('Original image size: %sx%s' % (w, h))
# 缩放到50%:
im.thumbnail((w//2, h//2))
print('Resize image to: %sx%s' % (w//2, h//2))
# 把缩放后的图像用jpeg格式保存:
im.save('thumbnail.jpg', 'jpeg')
```

```
============== RESTART: /Users
Original image size: 900x600
Resize image to: 450x300
```

PIL模糊效果


```
from PIL import Image, ImageFilter

# 打开一个jpg图像文件，注意是当前路径:
im = Image.open('test.jpg')
# 应用模糊滤镜:
im2 = im.filter(ImageFilter.BLUR)
im2.save('blur.jpg', 'jpeg')
```

PIL生成验证码

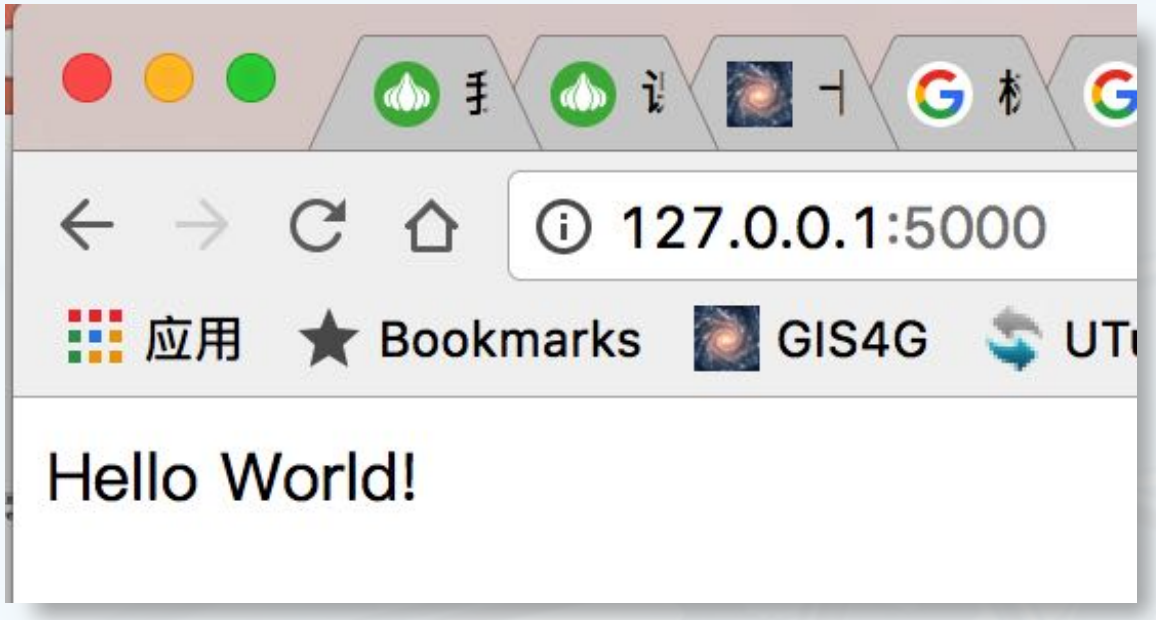
```python
from PIL import Image, ImageDraw, ImageFont, ImageFilter

import random

# 随机字母:
def rndChar():
    return chr(random.randint(65, 90))

# 随机颜色1:
def rndColor():
    return (random.randint(64, 255), \
            random.randint(64, 255), \
            random.randint(64, 255))

# 随机颜色2:
def rndColor2():
    return (random.randint(32, 127), \
            random.randint(32, 127), \
            random.randint(32, 127))

# 240 x 60:
width = 60 * 4
height = 60
image = Image.new('RGB', (width, height), (255, 255, 255))
# 创建Font对象:
font = ImageFont.truetype('Arial.ttf', 36)
# 创建Draw对象:
draw = ImageDraw.Draw(image)
# 填充每个像素:
for x in range(width):
    for y in range(height):
        draw.point((x, y), fill=rndColor())
# 输出文字:
for t in range(4):
    draw.text((60 * t + 10, 10), rndChar(), font=font, fill=rndColor2())
# 模糊:
image = image.filter(ImageFilter.BLUR)
image.save('code.jpg', 'jpeg')
```

Flask

- Web应用已经成为目前最热门的应用软件形式
- Web应用通过Web服务器提供服务，客户端采用浏览器或者遵循HTTP协议的客户端
- 由于需要处理HTTP传输协议，很多web开发框架涌现
- flask是一种非常容易上手的Python web开发框架，只需要具备基本的python开发技能，就可以开发出一个web应用来

Flask小例子


```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

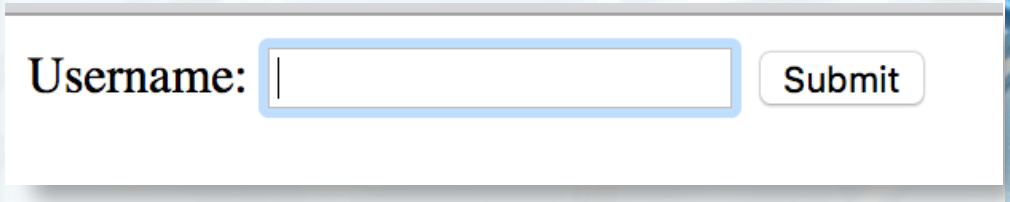



```
============= RESTART: /Users/chenbin/Documents/homework/flsk.py ===========
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [31/Mar/2017 02:47:59] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [31/Mar/2017 02:48:00] "GET /favicon.ico HTTP/1.1" 404 -
```

更复杂一些的例子：表单插件Flask-WTF


```python
from flask_wtf import Form
from wtforms import StringField
from wtforms.validators import DataRequired

class MyForm(Form):
    user = StringField('Username', validators=[DataRequired()])

from flask import Flask, render_template

app = Flask(__name__)
app.secret_key = '1234567'

@app.route('/login', methods=('GET', 'POST'))
def login():
    form = MyForm()
    if form.validate_on_submit():
        # if form.user.data == 'admin':
        if form.data['user'] == 'admin':
            return 'Admin login successfully!'
        else:
            return 'Wrong user!'
    return render_template('login.html', form=form)

if __name__ == "__main__":
    app.run()
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登录表单</title>
</head>
<body>
<form method="POST" action="{{ url_for('login') }}">
    {{ form.hidden_tag() }}
    {{ form.user.label }}: {{ form.user(size=20) }}
    <input type="submit" value="Submit">
</form>
</body>
</html>
```



numpy

- numpy是Python用于处理大型矩阵的一个速度极快的数学库。
- 它可以在Python中做向量和矩阵的运算，而且很多底层的函数都是用C写的，可以得到在普通Python中无法达到的运行速度
- 包括各种创建矩阵的方法，以及一般的矩阵运算、求逆、求转置

```
>>> import numpy as np
>>> a= np.matrix([[1,2],[3,4]])
>>> a.I
matrix([[-2. ,  1. ],
        [ 1.5, -0.5]])
>>> a.T
matrix([[1, 3],
        [2, 4]])
>>> a.I * a
matrix([[  1.00000000e+00,   0.00000000e+00],
        [  1.11022302e-16,   1.00000000e+00]])
>>> b= np.matrix([[7,6],[5,4]])
>>> a*b
matrix([[17, 14],
        [41, 34]])
```

```
>>> a.shape
(2, 2)
>>> a.size
4
>>> a.dtype
dtype('int64')
```

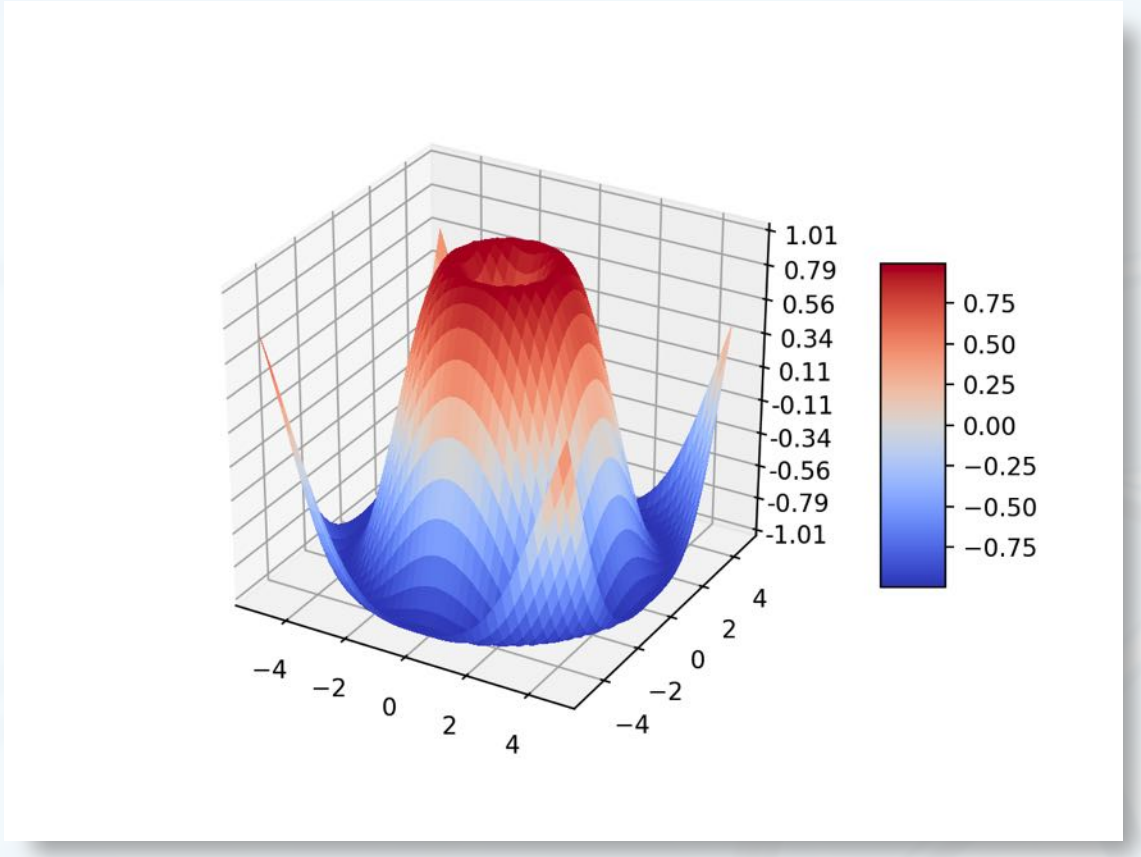
matplotlib

- **matplotlib 是 Python 的一个绘图库。它包含了大量的工具，可以使用这些工具创建各种图形**

  包括简单的散点图，折线图，甚至是三维图形、动画等，Python 科学计算社区经常使用它完成数据可视化的工作。

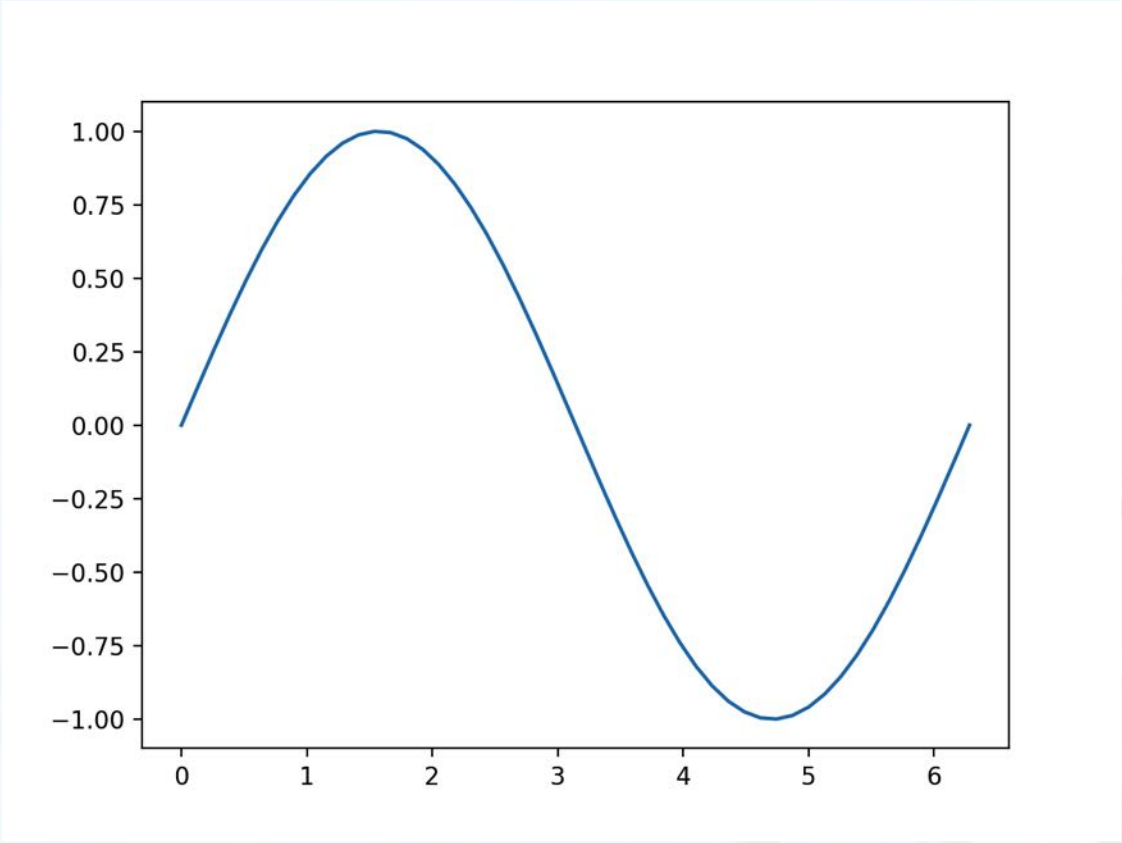
- **功能异常强大**

  http://matplotlib.org/gallery.html



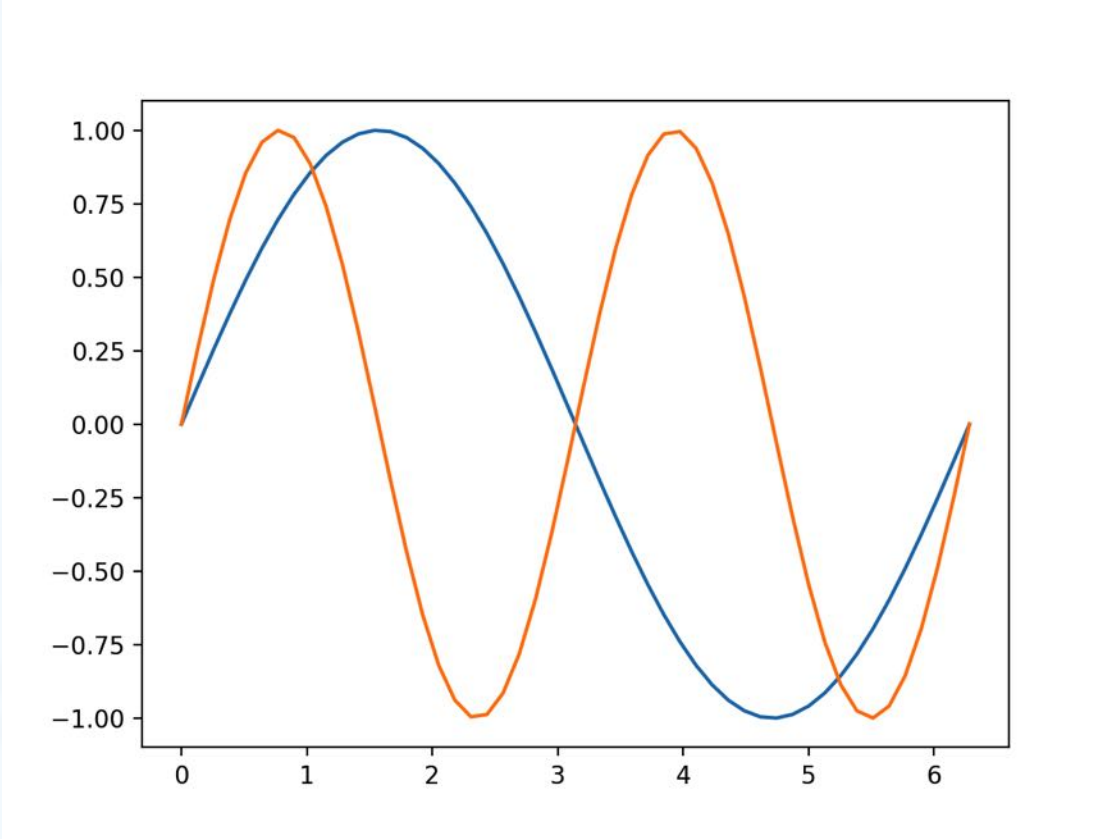
matplotlib：简单图形


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

# 简单的绘图
x = np.linspace(0, 2 * np.pi, 50)

# 如果没有第一个参数 x，图形的 x 坐标默认为数组的索引
plt.plot(x, np.sin(x))

plt.show() # 显示图形
```



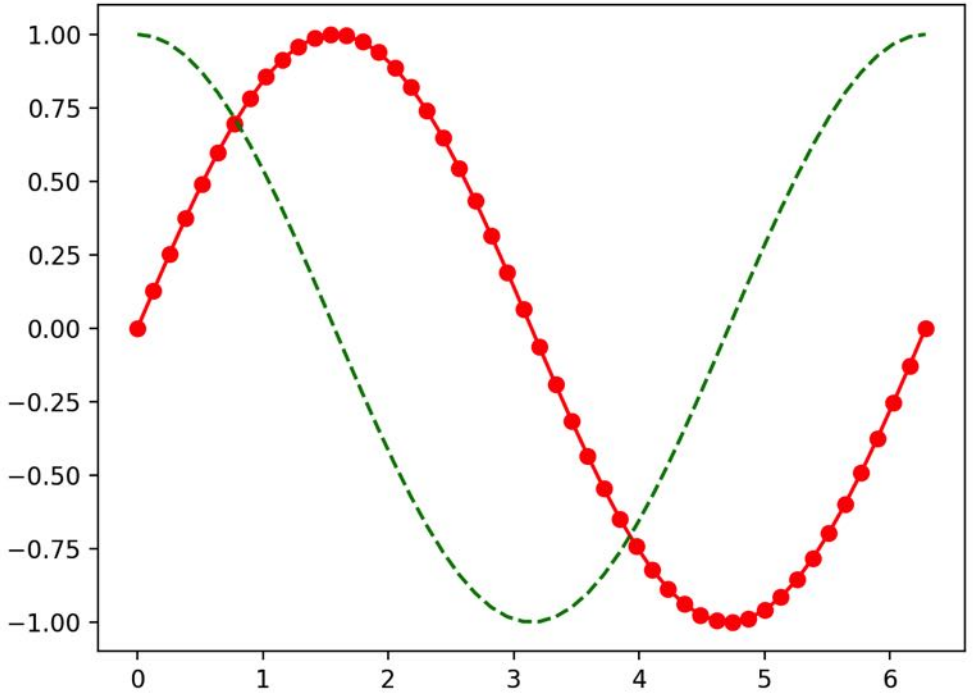
matplotlib：多个简单图形


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

x = np.linspace(0, 2 * np.pi, 50)
plt.plot(x, np.sin(x),
         x, np.sin(2 * x))
plt.show()
```



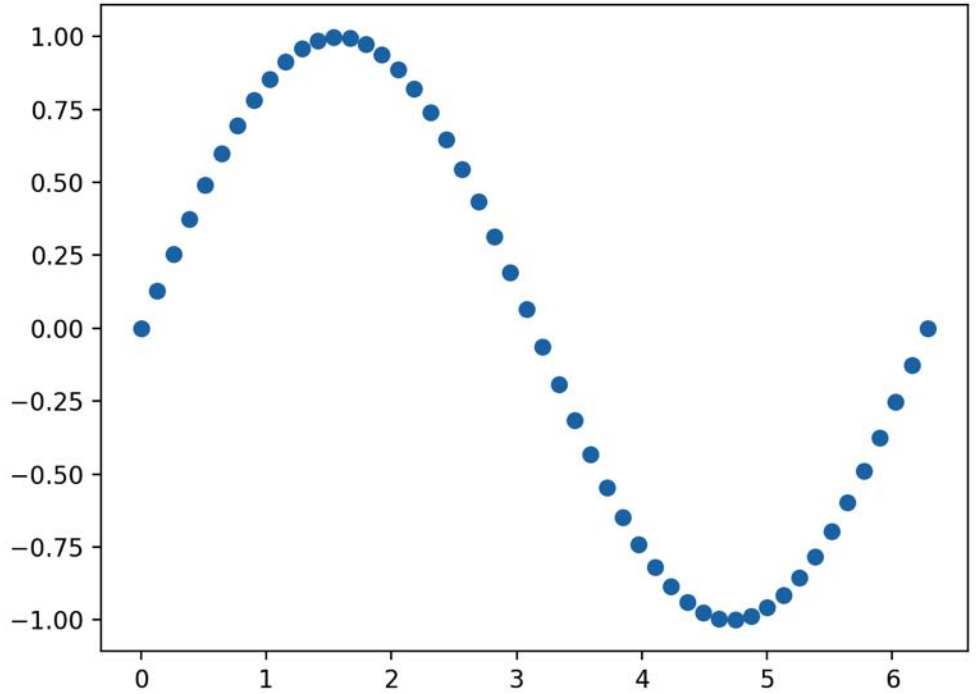
matplotlib：定制线型


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

# 自定义曲线的外观
x = np.linspace(0, 2 * np.pi, 50)
plt.plot(x, np.sin(x), 'r-o',
         x, np.cos(x), 'g--')
plt.show()
```



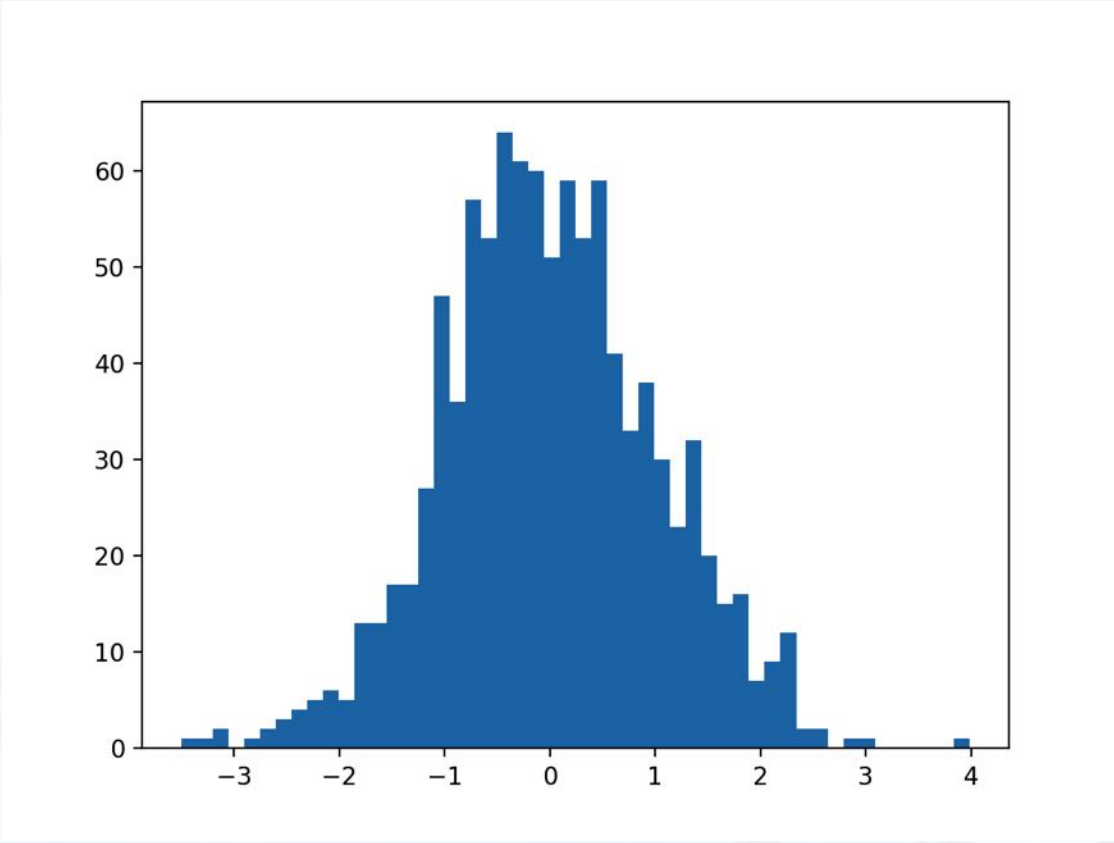
matplotlib：散点图


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

# 简单的散点图
x = np.linspace(0, 2 * np.pi, 50)
y = np.sin(x)
plt.scatter(x,y)
plt.show()
```



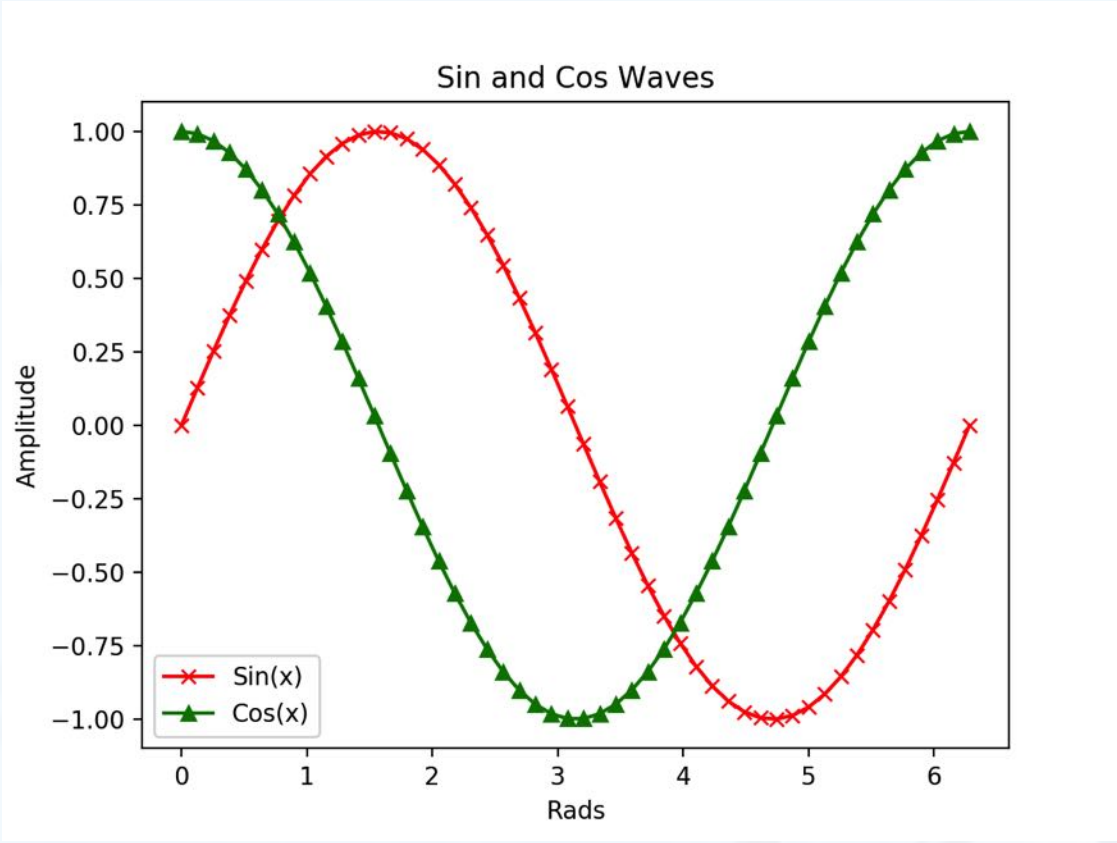
直方图


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

# 直方图
x = np.random.randn(1000)
plt.hist(x,50)
plt.show()
```



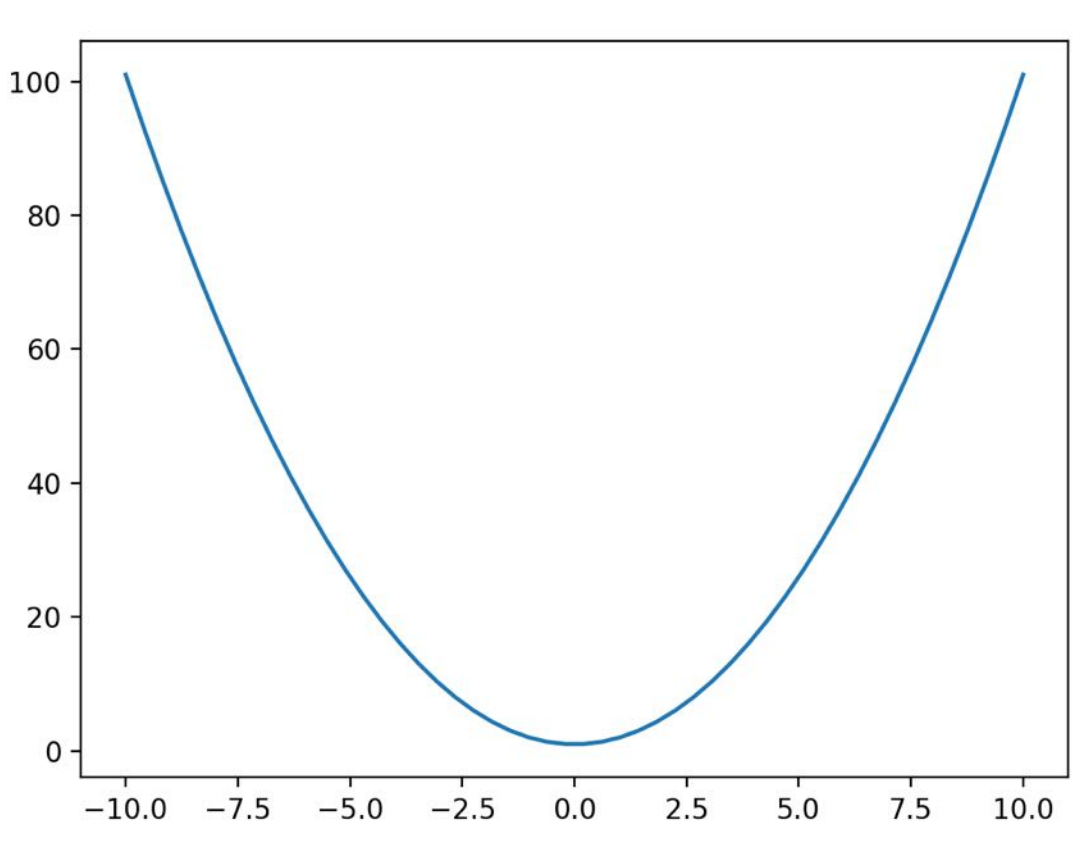
标题，标签和图例


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

# 添加标题，坐标轴标记和图例
x = np.linspace(0, 2 * np.pi, 50)
plt.plot(x, np.sin(x), 'r-x', label='Sin(x)')
plt.plot(x, np.cos(x), 'g-^', label='Cos(x)')
plt.legend() # 展示图例
plt.xlabel('Rads') # 给 x 轴添加标签
plt.ylabel('Amplitude') # 给 y 轴添加标签
plt.title('Sin and Cos Waves') # 添加图形标题
plt.show()
```



自定义函数


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

x = np.linspace(-10, 10, 50)

def y(x):
    return x**2+1

plt.plot(x, list(map(y, list(x))))

plt.show() # 显示图形
```



上机练习

- 在pycharm中安装模块：

  Pillow，flask，

  numpy，matplotlib

- 利用PIL生成一批图片的统一大小(200*100)的缩略图

  0.jpg，1.jpg，2.jpg……

  生成s0.jpg，s1.jpg，s2.jpg……

- 编写一个flask服务器，保存两个网页，URL分别是/, /city

  在/下面的网页带链接：

  ```
  <a href="/city">城市</a>
  ```

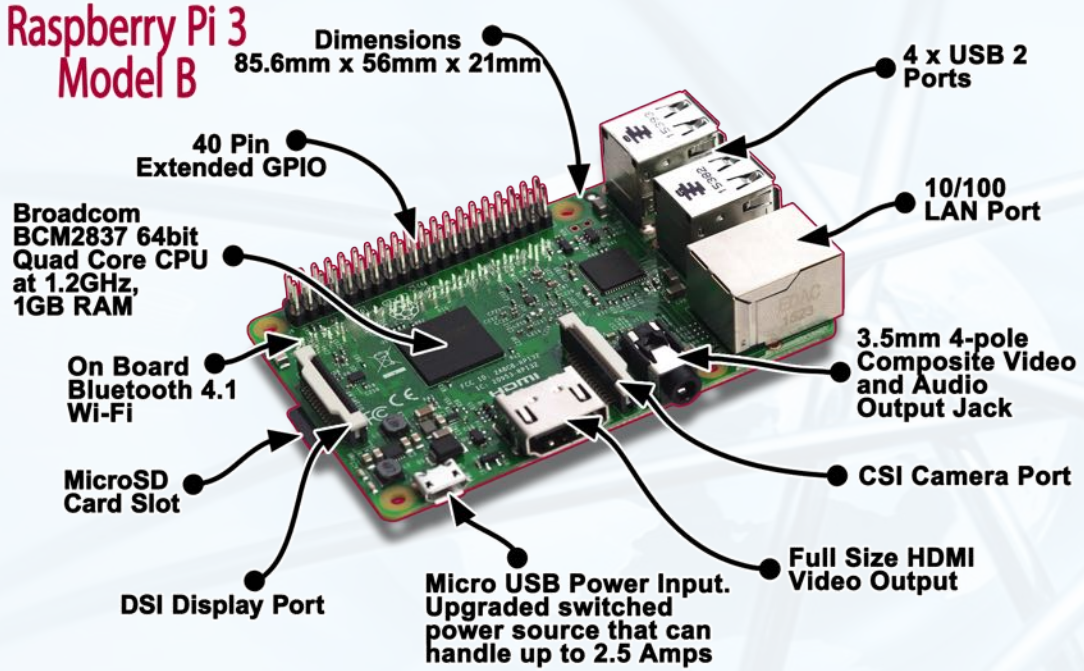
- 用matplotlib绘制一个带标题、标签和图例的函数图像(x=-10..10)：

  y=x**2-2*x

  y=2*x+3

树莓派及扩展板

- 树莓派（Raspberry Pi），是一款仅有信用卡大小的单板机电脑
- 它由英国的树莓派基金会所开发，目的是以低价硬件及自由软件促进学校的基本计算机科学教育
- 官方操作系统Raspbian（基于Debian Linux），推广Python语言
- 还可以运行很多Linux、Android
- Windows 10 IoT版



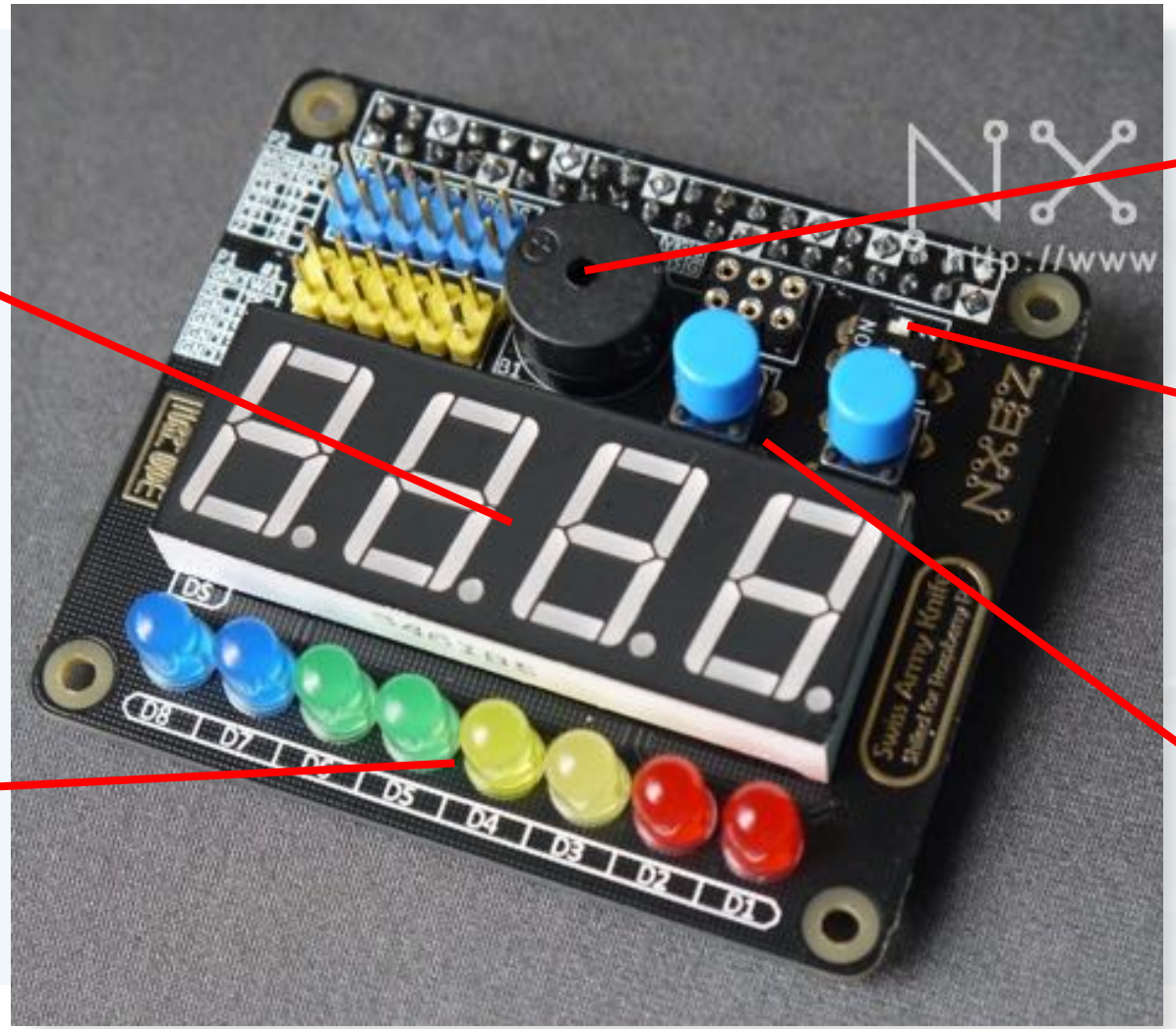
项目	Raspberry Pi B	Raspberry Pi B+	Raspberry Pi A+	Raspberry Pi 2 Model B	Raspberry Pi Zero	Raspberry Pi 3 Model B
发布时间	2011-12	2014-07-14	2014-11-11	2015-02-02	2015-11-26	2016-02-29
SoC	BCM2835	BCM2835	BCM2835	BCM2836	BCM2835	BCM2837
CPU	ARM1176JZF-S核心 700MHz 单核	ARM1176JZF-S核心 700MHz 单核	ARM1176JZF-S核心 700MHz 单核	ARM Cortex-A7 900MHz 四核	ARM1176JZF-S核心 700MHz 单核	ARM Cortex-A53 1.2GHz 四核
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, 1080p 30 h.264/MPEG-4 AVC 高清解码器（本表格由树莓派实验室绘制 http://shumeipai.nxez.com 若有疏漏请联系我们更正）					
RAM	512MB	512MB	256MB	1GB	512MB	1GB
USB接口	USB2.0 × 2	USB2.0 × 4	USB2.0 × 1	USB2.0 × 4	Micro USB2.0 × 1	USB2.0 × 4
视频接口	RCA视频接口输出，支持PAL和NTSC制式，支持HDMI (1.3和1.4)，分辨率为640 x 350 至 1920 x 1200 支持PAL 和 NTSC制式。	支持PAL和NTSC制式，支持HDMI (1.3和1.4)，分辨率为640 x 350 至 1920 x 1200 支持PAL 和 NTSC制式。	支持PAL和NTSC制式，支持HDMI (1.3和1.4)，分辨率为640 x 350 至 1920 x 1200 支持PAL 和 NTSC制式。	支持PAL和NTSC制式，支持HDMI (1.3和1.4)，分辨率为640 x 350 至 1920 x 1200 支持PAL 和 NTSC制式。	支持PAL和NTSC制式，支持HDMI (1.3和1.4)，分辨率为640 x 350 至 1920 x 1200 支持PAL 和 NTSC制式。	支持PAL和NTSC制式，支持HDMI (1.3和1.4)，分辨率为640 x 350 至 1920 x 1200 支持PAL 和 NTSC制式。
音频接口	3.5mm 插孔，HDMI（高清晰度多音频/视频接口）	3.5mm 插孔，HDMI（高清晰度多音频/视频接口）	3.5mm 插孔，HDMI（高清晰度多音频/视频接口）	3.5mm 插孔，HDMI（高清晰度多音频/视频接口）	HDMI（高清晰度多音频/视频接口）	3.5mm 插孔，HDMI（高清晰度多音频/视频接口）
SD卡接口	标准SD卡接口	Micro SD卡接口	Micro SD卡接口	Micro SD卡接口	Micro SD卡接口	Micro SD卡接口
网络接口	10/100 以太网接口（RJ45接口）	10/100 以太网接口（RJ45接口）	无	10/100 以太网接口（RJ45接口）	无	10/100 以太网接口（RJ45接口），内置WiFi、蓝牙。
GPIO接口	26PIN	40PIN	40PIN	40PIN	40PIN	40PIN
额定功率	700毫安(为3.5W)	600毫安(为3.0W)	未知，但更低	1000毫安(为5.0W)	未知，但更低	未知，但更高
电源接口	MicroUSB 5V	MicroUSB 5V	MicroUSB 5V	MicroUSB 5V	MicroUSB 5V	MicroUSB 5V
尺寸	85.60 × 53.98 mm	85 x 56 x 17 mm	65 × 56 mm	85 x 56 x 17 mm	65 × 30 x 5 mm	85 x 56 x 17 mm
官方定价	35美元	35美元	20美元	35美元	5美元	35美元

“瑞士军刀”扩展板SAKS

- 树莓派瑞士军刀扩展板（SAKS for RPi）是由一系列常用电气元件经过精心构建而成的多功能扩展板
- 适用于40Pin GPIO口的树莓派系列产品
- 可以基于树莓派主机和扩展板开发出丰富的上层软件，软硬件结合，研发出功能丰富的应用。

TM1637芯片控制4位数码管

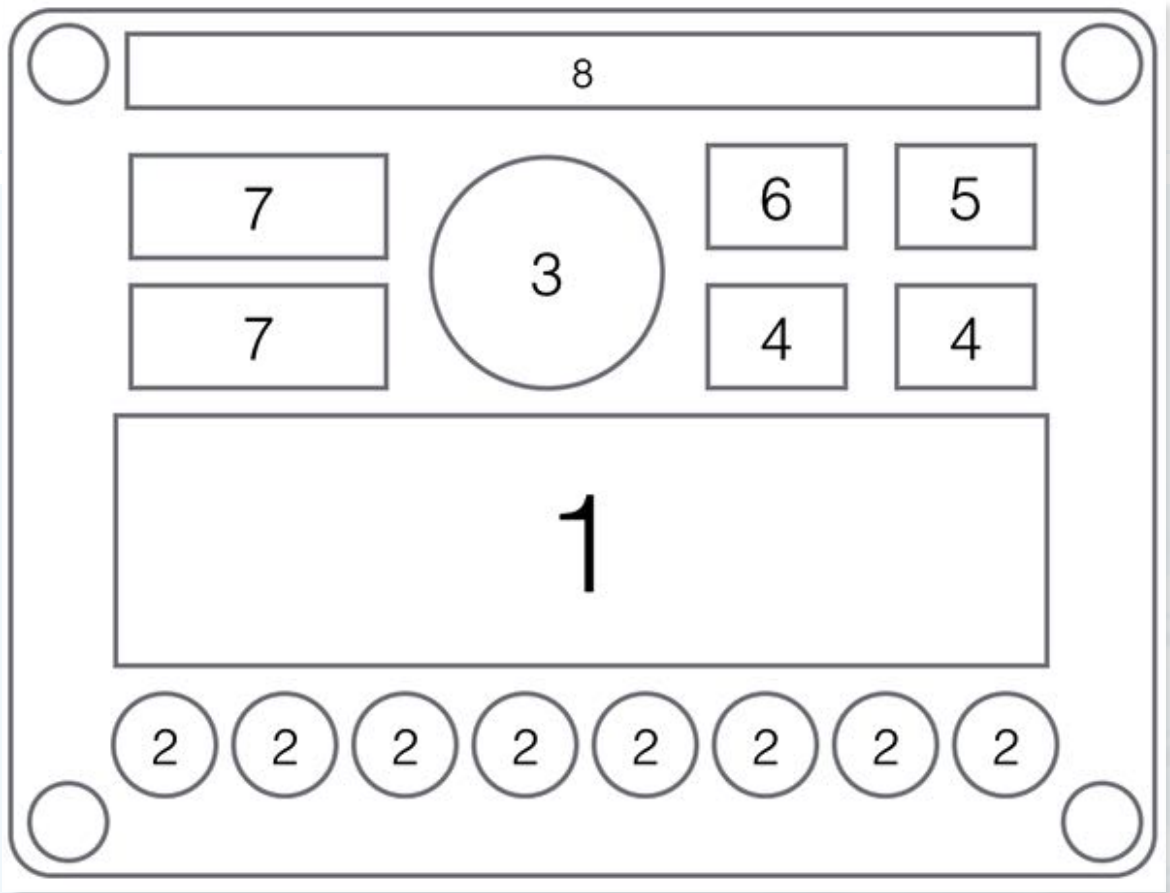
74HC595芯片控制LED8个



有源蜂鸣器

2位拨码开关

轻触开关2个

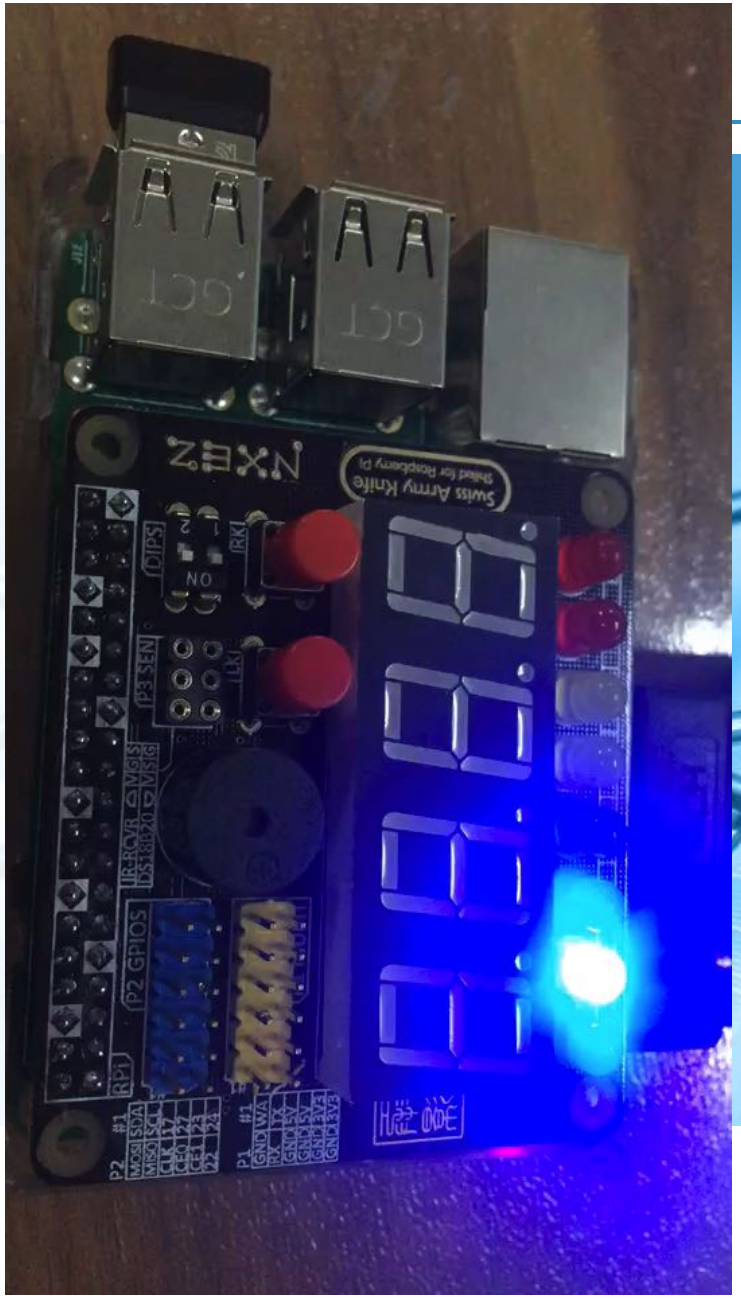


1. 4位数码管1个
2. LED 8个
3. 有源蜂鸣器1个
4. 轻触开关2个
5. 2位拨码开关1个
6. 2×3Pin 排座1个（专用传感器接口）
7. 2×6Pin 排针2个（功能扩展接口）
8. 40Pin 排座1个 或 40Pin 排针1个

RPi.GPIO模块开发示例

```python
def main():
    try:
        init()
        while True:
            #以下一组8个编码由一组二进制转换而成:
            #00000001,00000010,00000100,00001000,00010000,00100000,01000000,10000000
            #分别对应8个LED点亮状态
            for i in [0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80]:
                writeByte(i)
                time.sleep(0.1)
            #LED组全开
            #writeByte(0xff)
            #time.sleep(0.1)

    except KeyboardInterrupt:
        print "except"
        #LED组全关
        writeByte(0x00)
        GPIO.cleanup()
    return 0

if __name__ == '__main__':
    main()
```

```python
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
DS = 6
SHCP = 19
STCP = 13
def init():
    GPIO.setup(DS, GPIO.OUT)
    GPIO.setup(SHCP, GPIO.OUT)
    GPIO.setup(STCP, GPIO.OUT)
    GPIO.output(DS, GPIO.LOW)
    GPIO.output(SHCP, GPIO.LOW)
    GPIO.output(STCP, GPIO.LOW)

def writeBit(data):
    GPIO.output(DS, data)
    GPIO.output(SHCP, GPIO.LOW)
    GPIO.output(SHCP, GPIO.HIGH)

#写入8位LED的状态
def writeByte(data):
    for i in range (0, 8):
        writeBit((data >> i) & 0x01)
    #状态刷新信号
    GPIO.output(STCP, GPIO.LOW)
    GPIO.output(STCP, GPIO.HIGH)
```



SAKS SDK开发示例


```python
from sakshat import SAKSHAT
import time

# Declare the SAKS Board
SAKS = SAKSHAT()

def main():
    b = SAKS.buzzer  # 蜂鸣器
    b.beep(1)
    # 定义亮灯状态和数字显示
    alloff = list((False,) * 8)
    onone = [alloff[:] for i in range(8)]
    for i in range(8):
        onone[i][i] = True
    nums = {0: '1000', 1: '2000', 2: '0100', 3: '0200', 4: '0010', 5: '0020', 6: '0001', 7: '0002'}

    SAKS.ledrow.off()
    time.sleep(3)
    SAKS.ledrow.set_row([True, False, True, False, True, False, True, False])
    time.sleep(2)
    for i in range(8):
        SAKS.digital_display.show(nums[i])
        SAKS.ledrow.set_row(onone[i])
        time.sleep(0.5)
    SAKS.ledrow.off()

    SAKS.digital_display.show("2.3.3.3.") # 将显示“2333”4位数字，并且每一位右下角的小点点亮
    time.sleep(1)
    SAKS.digital_display.show("66.66") # 将显示“6666”4位数字，并且数字2后面的小点点亮
    time.sleep(1)
    SAKS.digital_display.show("###1") # 在第4位数码管显示“1”，其他3位数码管不显示
    time.sleep(1)
    SAKS.digital_display.off()

if __name__ == '__main__':
    main()
```