2017年11月24日17:08:49

1. python的输出使用print

print（“hello world”）

1. 输入 input

x=input（）

name=input（“please input your name”）

1. Python的注释

用#号

以#开头的语句是注释，注释是给人看的，可以是任意内容，解释器会忽略掉注释。其他每一行都是一个语句，当语句以冒号:结尾时，缩进的语句视为代码块。

python程序是大小写敏感的，如果写错了大小写，程序会报错

1. 数据类型

计算机顾名思义就是可以做数学计算的机器，因此，计算机程序理所当然地可以处理各种数值。但是，计算机能处理的远不止数值，还可以处理文本、图形、音频、视频、网页等各种各样的数据，不同的数据，需要定义不同的数据类型。在Python中，能够直接处理的数据类型有以下几种：

整数

Python可以处理任意大小的整数，当然包括负整数，在程序中的表示方法和数学上的写法一模一样，例如：1，100，-8080，0，等等。

计算机由于使用二进制，所以，有时候用十六进制表示整数比较方便，十六进制用0x前缀和0-9，a-f表示，例如：0xff00，0xa5b4c3d2，等等。

浮点数

浮点数也就是小数，之所以称为浮点数，是因为按照科学记数法表示时，一个浮点数的小数点位置是可变的，比如，1.23x109和12.3x108是完全相等的。浮点数可以用数学写法，如1.23，3.14，-9.01，等等。但是对于很大或很小的浮点数，就必须用科学计数法表示，把10用e替代，1.23x109就是1.23e9，或者12.3e8，0.000012可以写成1.2e-5，等等。

整数和浮点数在计算机内部存储的方式是不同的，整数运算永远是精确的（除法难道也是精确的？是的！），而浮点数运算则可能会有四舍五入的误差。

字符串

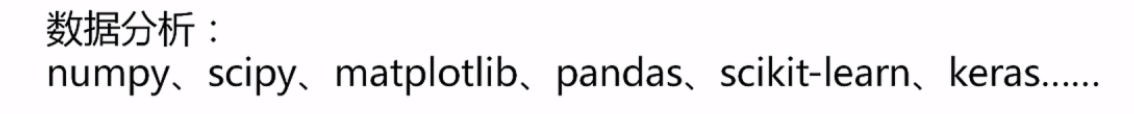
**字符串是以单引号'或双引号"括起来的任意文本**，比如'abc'，"xyz"等等。请注意，''或""本身只是一种表示方式，不是字符串的一部分，因此，字符串'abc'只有a，b，c这3个字符。如果'本身也是一个字符，那就可以用""括起来，比如"I'm OK"包含的字符是I，'，m，空格，O，K这6个字符。

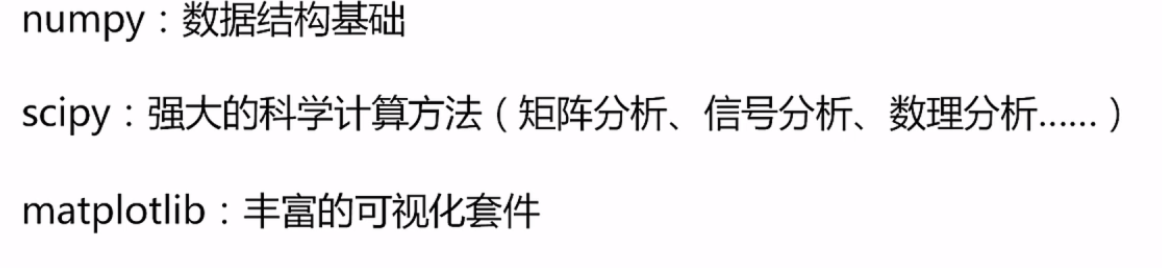
**2018年11月19日17:13:59**

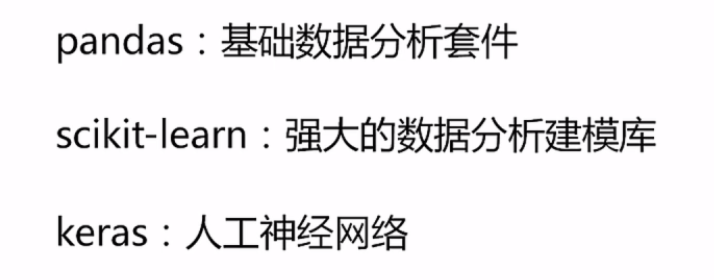
1. python视频教程

<https://www.imooc.com/video/15001>

1. python数据分析：







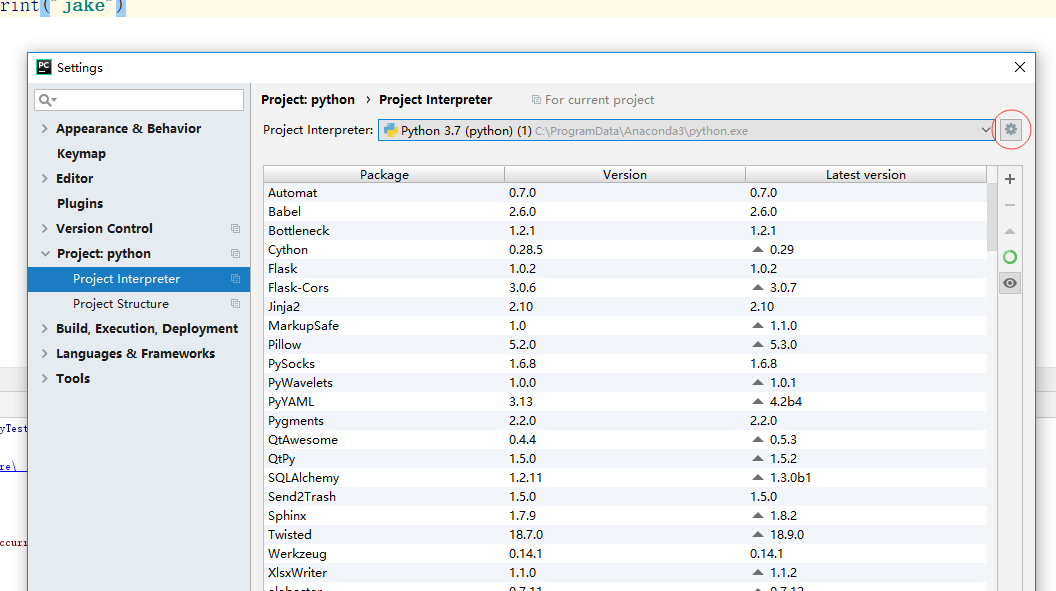
1. pycharm中使用numpy

安装好pycharm后，要设置python的解析器。也就是设置python的安装路径。

python可以直接从官网安装，但是这个版本缺乏很多库。也可以使用anaconda来安装。

anaconda的python安装路径是C:\ProgramData\Anaconda3

在pycharm里面点击file->settings -> project->project interpreter

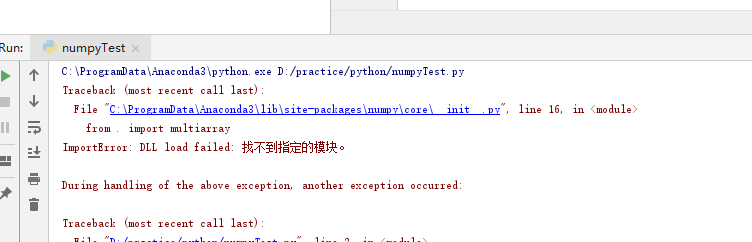


选择非anaconda的python，点击齿轮按钮添加，选择existing environment。

上面那个图里面，点击加号后，输入numpy，点击安装numpy。就可以用了。

python 代码里面用法：import numpy as np

不过上面的方法若用在anaconda还是不行。



2018年11月22日22:39:01

1. 使用numpy的数组

python的默认数组里面可以存放任何类型的数据，而numpy的数组只能存放一种类型的数据，使用numpy.array来使用

import numpy as np

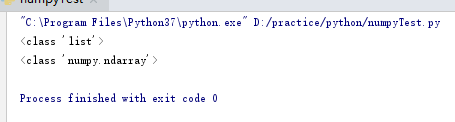
lst=[[1,3,5],[2,4,6]]

print(type(lst))

np\_lst=np.array(lst)

print(type(np\_lst))

输出：



1. 定义数组的类型

np\_lst=np.array(lst,dtype=np.float)

数据类型：bool, int, int8, int16, int32, int64, int128, uint8, uint16,uint32, uint64,uint128, float,float16,float32

1. 输出数组的形状

print(np\_lst.shape)

输出为：(2,3) 表述这个数组是两行三列

1. 输出数组的维度

print(np\_lst.ndim)

结果为：2 表示两维

1. 输出数据类型

print(np\_lst.dtype)

结果为：float64 ，float的默认类型就是float64

6．输出每个元素的大小

print(np\_lst.dtype)

结果为：8 ，表明是8个字节。因为float64是8字节的。

7. 输出数组的元素个数

print(np\_lst.size)

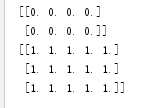
结果为：6

2018年11月22日22:54:44

1. 输出特殊的数组

print(np.zeros([2,4]))

print(np.ones([3,5]))



1. 生成随机数
2. 输出两行四列是0-1的随机数

print(np.random.rand(2,4))



1. 输出一个随机数

print(np.random.rand())

0.07455894346709724

1. 生成一个随机的整数：

要设置一个范围，比如1-10

print(np.random.randint(1,10))

也可设置生成的个数，比如3个

print(np.random.randint(1,10,3))



1. 生成一个标准正态分布的随机数

print(np.random.randn())

0.2545596654020659

也可以说设置尺寸，两行四列：

print(np.random.randn(2,4))

结果：[[-0.20550518 2.02257755 -2.5338618 -0.97472658]

[-0.7055419 -1.5865443 -0.02796797 0.4979502 ]]

1. 使用choice生成指定数组中的随机数，也就是从已有的数组中随机选择一个

print(np.random.choice([10,20,30]))

这里随机从10,20,30中随机选择一个

1. 输出beta分布的随机数

print(np.random.beta(1,10,100))

这里输出100个1-10的复合beta分布的随机数

1. 还可以使用numpy生成很多别的分布
2. 使用arange生成一个等差数列：

print(np.arange(1,11))

这里生成1到11的等差数列，其中包括1，但不包括11.

结果：[ 1 2 3 4 5 6 7 8 9 10]

这里输出的是一维的数组，可以使用reshape改成二维的数组

print(np.arange(1,11).reshape([2,5])) 这个5可以可以写成默认的-1



1. 使用exp输出e的指数

a=1

print(np.exp(a))

输出：2.718281828459045

也可以传入一个数组

print(np.exp(np.array([1,2,3])))

输出：[ 2.71828183 7.3890561 20.08553692]

1. exp2函数

求2为底数的指数：

比如np.exp2(2) 结果为：4

1. 其余函数：

sqrt：开方

sin：正弦

log：对数

1. 两个数组相加

ls1=np.array([1,2,3])

ls2=np.array([4,5,6])

print(ls1+ls2)

结果：[5 7 9]

1. 两个数组相减

ls1=np.array([1,2,3])

ls2=np.array([4,5,6])

print(ls1-ls2)

结果: [-3 -3 -3]

1. 其余操作

print(ls1/ls2) 除法

print(ls1\*ls2) 乘法

print(ls1\*\*ls2) 乘方 结果：[ 1 32 729]

1. 点乘，即矩阵相乘

ls1=np.array([1,2,3,4])

ls2=np.array([4,5,6,7])

print(np.dot(ls1.reshape([2,2]),ls2.reshape([2,2])))

结果：[[16 19]

[36 43]]

1. 数组拼接：

print(np.concatenate((ls1,ls2),axis=0))

结果：[1 2 3 4 4 5 6 7]

print(np.vstack((ls1,ls2))) 注意括号个数

结果：[[1 2 3 4]

[4 5 6 7]]

print(np.hstack((ls1,ls2)))

结果：[1 2 3 4 4 5 6 7]

1. 数组的分离：

print(np.split(ls1,2))

结果：[array([1, 2]), array([3, 4])]

1. 数组的复制：

print(np.copy(ls1))

结果：[1 2 3 4]

2018年11月23日16:13:39

1. 线性方程组和矩阵

引入模块

**from numpy.linalg import \***

1. 单位矩阵

print(np.eye(3))

结果: [[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]]

3. 求矩阵的逆

ls3=np.array([[1,2],[3,4]])

print(inv(ls3))

结果：[[-2. 1. ]

[ 1.5 -0.5]]

4. 求矩阵的转置

print(ls3.transpose())

结果：

[[1 3]

[2 4]]

5. 求矩阵的行列式

print(det(ls3))

结果：-2.0000000000000004

6. 求特征值和特征向量

print(eig(ls3))

结果：(array([-0.37228132, 5.37228132]), array([[-0.82456484, -0.41597356],

[ 0.56576746, -0.90937671]]))

7. 解方程组

y=np.array([[5.],[7.]])

print(solve(ls3,y))

结果：[[-3.]

[ 4.]]

即解方程组： 5=x1+2x2 7=3x1+4x2

2018年11月23日16:54:36

1. numpy在其他领域的应用
2. FFT 快速傅里叶变换，用在信号处理方面

print(np.fft.fft(np.array([1,1,1,1,1,1,1,1,1])))

结果：[9.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j 0.+0.j]

1. 求相关系数

print(np.corrcoef([1,0,1],[0,2,1]))

结果：

[[ 1. -0.8660254]

[-0.8660254 1. ]]

4. 生成一元多次函数

print(np.poly1d([2,1,3]))

结果：

这个结果实际上是2x2+x+3

2018年11月23日17:10:41

1. matplotlib的使用

用来绘制图标

首先要安装matplotlib的包，在pycharm里面settings安装即可。

导入包 import matplotlib.pyplot as plt

1. 画正弦和余弦函数

import numpy as np

import matplotlib.pyplot as plt

x=np.linspace(-np.pi,np.pi,256,endpoint=True)

c,s=np.cos(x),np.sin(x)

plt.figure(1)

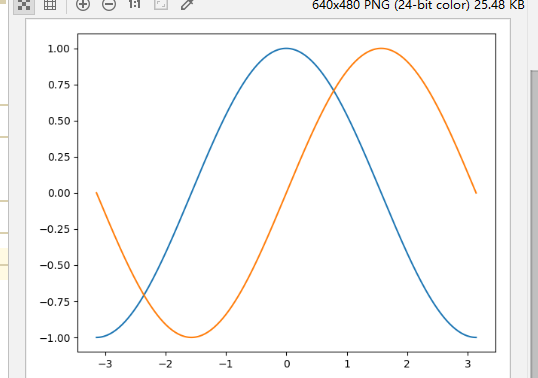
plt.plot(x,c)

plt.plot(x,s)

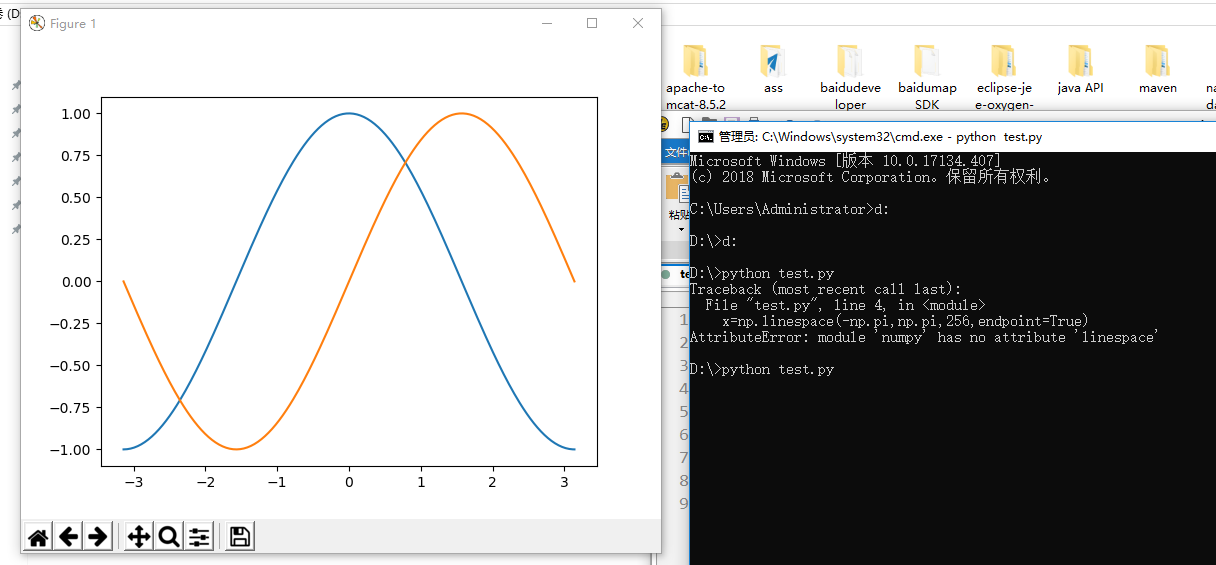
plt.show()

上面自变量从-π到π的256个数，包括最后一个数

结果：



在命令行里面运行代码也可以：



还可以设置颜色，线宽，样式等

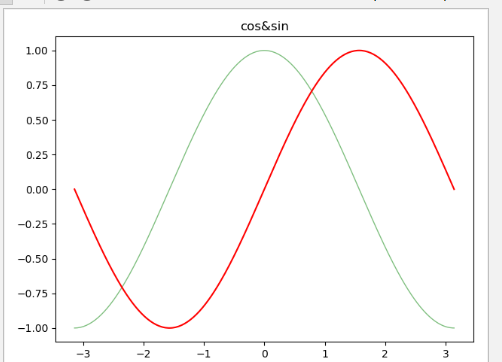
plt.plot(x,c,color="green",linewidth=1.0,linestyle="-")

还可以设置标签，透明度等

plt.plot(x,c,color="green",linewidth=1.0,linestyle="-",label="COS",alpha=0.5)

设置标题

plt.title("cos&sin")



设置坐标轴样式，隐藏上和右边框，坐标轴居中。

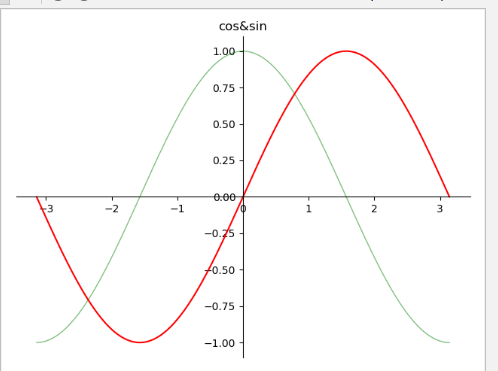
ax=plt.gca()

ax.spines["right"].set\_color("none")

ax.spines["top"].set\_color("none")

ax.spines["left"].set\_position(("data",0))

ax.spines["bottom"].set\_position(("data",0))



2018年11月23日22:44:11

1. 画散点图

fig=plt.figure()

fig.add\_subplot(3,3,1)

n=128

X=np.random.normal(0,1,n)

Y=np.random.normal(0,1,n)

T=np.arctan2(Y,X)

plt.axes([0.025,0.025,0.95,0.95])

plt.scatter(X,Y,s=75,c=T,alpha=.5)

plt.xlim(-1.5,1.5),plt.xticks([])

plt.ylim(-1.5,1.5),plt.yticks([])

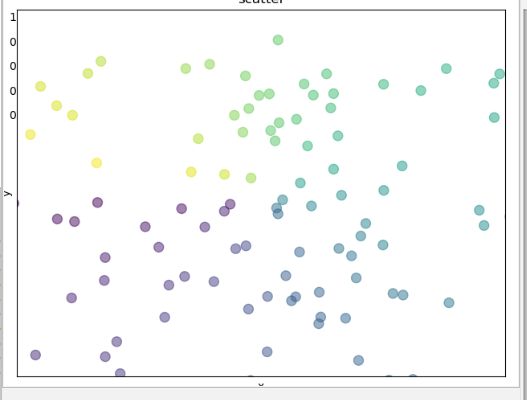
plt.axis()

plt.title("scatter")

plt.xlabel("x")

plt.ylabel("y")

plt.show()



2018年11月26日23:17:30

1. 使用scipy进行积分运算

首先还是得在python里面安装scipy的包。

import numpy as np

#integrate

from scipy.integrate import quad,dblquad

print(quad(lambda x:np.exp(-x),0,np.inf))

结果：(1.0000000000000002, 5.842606703608969e-11)

第二个为误差范围

1. 二元积分

print(dblquad(lambda t,x:np.exp(-x\*t)/t\*\*3,0,np.inf,lambda x:1,lambda x:np.inf))

结果：(0.33333333325010883, 1.3888461883425516e-08)

1. n元积分

def f(x,y):

return x\*y

def bound\_y():

return [0,0.5]

def bound\_x(y):

return [0,1-2\*y]

print(nquad(f,[bound\_x,bound\_y]))

结果：(0.010416666666666668, 4.101620128472366e-16)

1. optimize

2018年11月27日15:23:18

1. python的类

定义类使用class，比如：

class Student（object）：

pass

class后面跟的是类名，括号里面是继承的类  
 定义好类后，就可以创建对象，

bart=Student()

前面是对象名，可以任意，后面就是名了。

1. 在创建类的时候，可以通过一个特殊的方法\_\_init\_\_()把类的属性传递过去。类似于构造方法。

class Student（object）：

def \_\_init\_\_(self,name,socre)

self.name=name

self.score=score

注意：self这个关键字是固定的，可以吧各种属性绑定到self上面。。并且init的前后都有两个下划线。

这时创建对象的方法：

bart=Student（’Bart Simpson’,59）

1. python 中from的用法

经常看到别的程序使用了from、import关键字

发现from是用来引用py文件的，import才是导入类的

比如定义了一个KNN.py文件，里面有个Student类。

在一个测试类里面需要使用这个Student类。那么导入方法是：

from KNN import Student

1. 数据封装

在KNN.py中定义一个方法

def print\_score(self):

print('%s: %s' %(self.name,self.age))

调用方法：

tom=Student("tom",44)

tom.print\_score()

2018年11月27日16:09:24

1. 使用sys模块，获取命令行参数

python程序的运行都是：python test.py的形式

第一个参数的python文件，其实后面还可以跟别的参数。这时候就要用到sys了。

import sys

def test():

args = sys.argv

if len(args)==1:

print('Hello, world!')

elif len(args)==2:

print('Hello, %s!' % args[1])

else:

print('Too many arguments!')

if \_\_name\_\_=='\_\_main\_\_':

test()

导入sys模块后，我们就有了变量sys指向该模块，利用sys

这个变量，就可以访问sys模块的所有功能。

sys模块有一个argv变量，用list存储了命令行的所有参数。argv至少有一个元素，因为第一个参数永远是该.py文件的名称，例如：

运行python3 hello.py获得的sys.argv就是['hello.py']；

运行python3 hello.py Michael获得的sys.argv就是['hello.py', 'Michael]。

2.



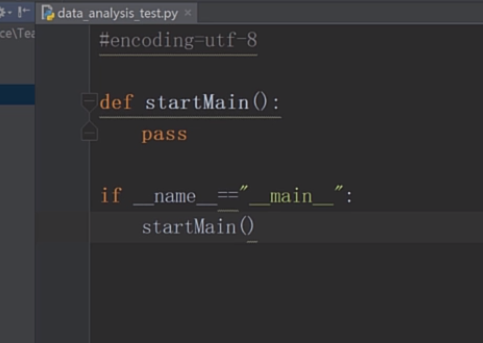
在一个模块里面使用 if \_\_name\_\_=’\_\_main\_\_’: 的时候，是为了方便测试用的

在这句话的下面加上测试方法。

如果是在命令行里面直接运行这个模块，比如 python hello.py，那么就会运行这个模块的test（）方法。如果是别别的模块引用了该模块，那么if判断会失败，即不会运行test()方法。

简单说，如果直接运行该模块，则会把它当做主方法，如果是被别的模块引用的，就不会把它当作主方法。

比如慕课网上的python数据分析视频教程就是这么做的。



这是他新建一个模块时的默认代码。

廖雪峰的讲解连接是：

<https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000/001431845183474e20ee7e7828b47f7b7607f2dc1e90dbb000>

这样做的好处就是为了方便测试，如果不需要测试，那么不这么做也是可以的。

比如新建一个test2.py，内容如下：

def test2():

print("hello world")

那么直接运行它： python test2.py 是不出任何结果的

3. 作用域

\_\_author\_\_，\_\_name\_\_等是特殊变量。前后是两个下划线

而 \_xxx或者\_\_xxx这样的函数或者变量就是非公开的，也就是说别的模块不能直接引用。

正常变量是公开的，在正常变量前加一条或者两条下划线就是非公开的。

在一个模块里面引用别的模块里面的变量或者方法，必须要用import来指定

比如在test2.py里面有个msg变量

\_\_name="guozhang"

msg="hello"

def test2():

print("hello world %s" %\_\_name)

def test3():

print("i have a dream")

在test.py里面引用：

from test2 import msg

print(msg)

这样就可以直接使用别的模块里面的msg变量了。

如果要引用的内容过多，可以使用\* 来引用所有内容。

from test2 import \*

如果test2.py里面的msg是非公开的，即变为 \_\_msg=”hello world”

那么即使在test.py里面引用了，也无法访问。

1. 隐藏类的属性

默认的类里面的属性是公开的。比如之前的student类。

class Student(object):

def \_\_init\_\_(self,name,age):

self.name=name

self.age=age

这个类有两个属性：name和age。

那么创建对象后，可以直接访问它：

比如：tom=Student("tom",22)

print(tom.name)

这样能直接输出name属性

如何隐藏类的属性呢？还是使用加下划线的方式

class Student(object):

def \_\_init\_\_(self,name,age):

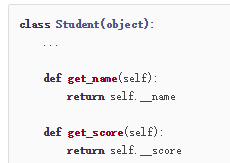
self.\_\_name=name

self.\_\_age=age

这样调用的时候print(tom.name)和print(tom.\_\_name)都不能用

只能通过Student里面的别的方法间接访问。

改为私有变量后，可以增加get方法来获取它



1. 继承和多态

在定义类的时候，括号里面设置父类

class Animal(object):

def run(self):

print('Animal is running...')

class Dog(Animal):

pass

class Cat(Animal):

pass

这样Dog类和Cat类都继承自Animal类

使用了继承，就可以使用父类的方法和数据了。

当子类和父类都存在相同的run()方法时，我们说，子类的run()覆盖了父类的run()，在代码运行的时候，总是会调用子类的run()。这样，我们就获得了继承的另一个好处：多态。

1. 使用type来判断类型：

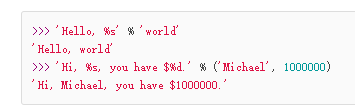
a=Dog()

print(type(a))

结果：<class 'Dog.Dog'>

1. python字符串的格式化

和c语言类似，但有区别



使用%来格式化，前面是通配符。可以有%d %f %s %x

后面是真正的数据，如果只有一个变量，直接在%后面加上变量即可，如果有多个建立，要在%（）的括号内增加变量。

如果以字符串里面本身就是百分号%, 就需要转义，方法是两个百分号%%

1. 使用format来格式化字符串

有点和C#类似。

>>> 'Hello, {0}, 成绩提升了 {1:.1f}%'.format('小明', 17.125)

'Hello, 小明, 成绩提升了 17.1%'

直接对带通配符的字符串调用format方法，参数对应前面的通配符。

注意：使用%的时候，是没有逗号的，

使用format是有点的，是调用的format方法

print(**"hello %s"** % **"guozhang"**)  
print(**"hello %s,%s"** %(**"guo"**,**"zhang"**))  
print(**"hello {0}"**.format(**"guozhang"**))

10． 使用requests来进行网络访问

注意是requests，而不是request，这两个都可以在python中安装，不是同一个东西。

import requests

r=requests.get('http://www.guozhang.top/')

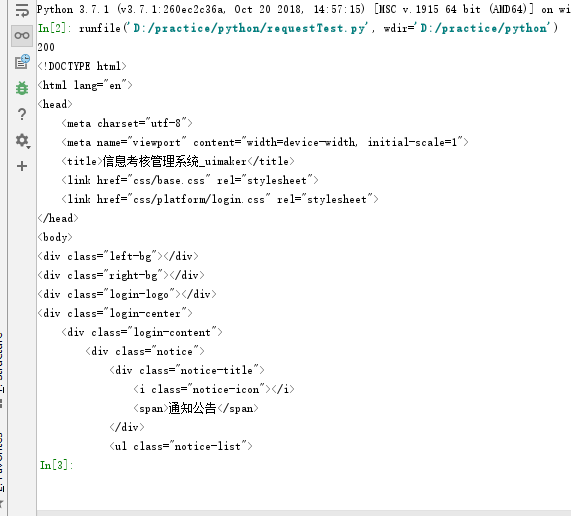
print(r.status\_code)

结果：200，表明访问正常。

如果打印返回内容：

print(r.text)

结果为html内容：

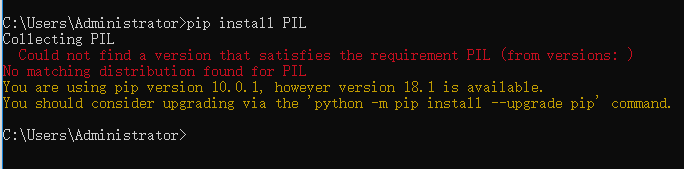


11. PIL库的使用

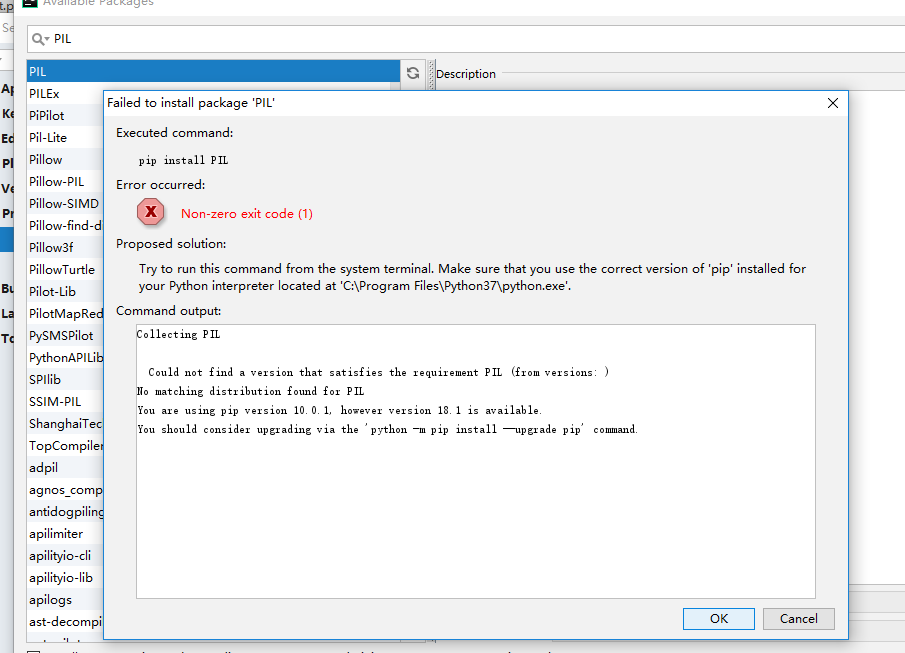
PIL：Python Imaging Library，已经是Python平台事实上的图像处理标准库了。

在之前做hexo的博客的时候，对于图片的处理就用了PIL。

PIL是python2的工具，不能在python3使用。如果使用pip install PIL 是会报错的。

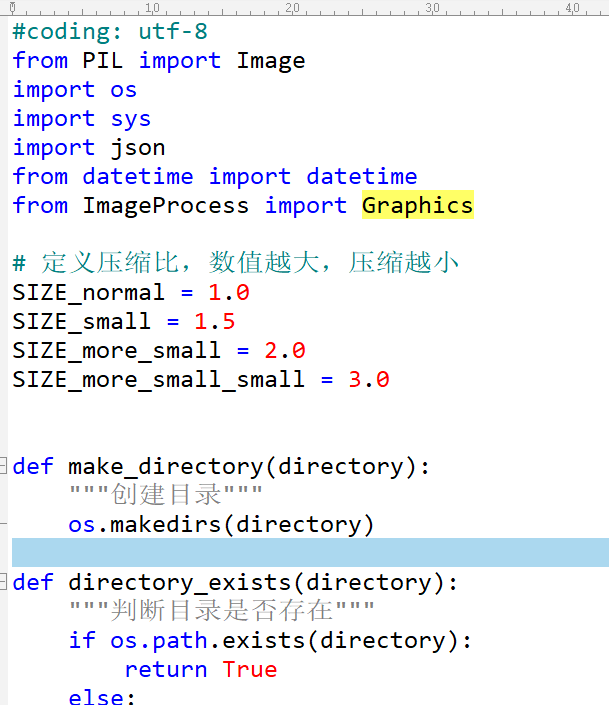


在pycharm里面安装也会报错



正确方法应该是安装Pillow。

如果没有安装Pillow，之前博客的图片压缩也不能用。提示没有PIL模块。



安装方法是在pycharm里面直接搜索pillow，安装即可。

2. 获取图像的尺寸

from PIL import Image

im=Image.open("2018-09-24\_321171.jpg")

w,h=im.size

print("Original image size: %sx%s"%(w,h))

3. 图像压缩使用 thumbnail方法

im.thumbnail((w//2,h//2))

im.save("thumbnail.jpg","jpeg")

注意是两个括号，两个斜线

4. Python中的除法

（1）真正的除法：/ 返回真正的商，不管除数与被除数的数据类型，返回的都是浮点型

比如 9/3 得到 3.0

(2) 地板除法： // 不管结果中有没有小数，都直接舍去，保留整数部分。

比如：9//2 得到 4

（3）取余 ：% 返回余数，没有余数返回0

比如：9%2 得到1

5.  **python创建一个类的实例后，还可以给他增加属性和方法**。这是动态语言的优势

比如创建一个Student类，在Student.py的内容：

class Student(object):

pass

内容很简单，几乎什么都没有

然后在测试类里面调用：

from Student import \*

s=Student()

s.name="guozhang"

print(s.name)

竟然能正确运行

还可以绑定一个方法

def set\_age(self,age):

self.age=age

from types import MethodType

s.set\_age=MethodType(set\_age,s)

s.set\_age(25)

print(s.age)

也能正常运行。

但是给一个实例绑定方法后，另一个对另一个实例却不起作用。

比如：s2=Student（）

不能调用s2.set\_age(23)

如何给所有实例绑定方法呢，使用class

def set\_score(self,score):

self.score=score

Student.set\_score=set\_score

s2=Student()

s2.set\_score(44)

print(s2.score)

6. 使用 \_\_slots\_\_限制实例的属性，比如只想要student添加name和age属性

class Teacher(object):

\_\_slots\_\_ = ('name','age')

from teacher import \*

t=Teacher()

t.name="jack"

t.age=22

print(t.name)

print(t.age)

如果增加一个属性，t.height=99

则会报错

使用\_\_slots\_\_要注意，\_\_slots\_\_定义的属性仅对当前类实例起作用，对继承的子类是不起作用的：

7． 为了限制属性的值不被随意修改，可以将其设置为私有的，并且增加get和set方法，在set方法里面判断类型和设置范围。

但这样做有点复杂。可以使用@property来设置。

class Teacher(object):

@property

def score(self):

return self.\_\_score

@score.setter

def score(self,value):

if not isinstance(value,int):

raise ValueError("score must be an integer")

if value<0 or value>100:

raise ValueError("socre must between 0~100")

self.\_\_score=value

注意要写成 \_\_score, 否则调用的时候print打印不出来

调用方法：

from teacher import \*

t=Teacher()

t.score=100

print(t.score)

调用的时候就是score，而不是\_\_score

2018年11月27日21:32:07

1. 多重继承

class Runnable(object):

def run(self):

print('Running...')

class Flyable(object):

def fly(self):

print('Flying...')

class Mammal(Animal):

pass

class Dog(Mammal, Runnable):

pass

狗这个类即继承哺乳动物类，又继承能跑这个类。

通过多重继承，一个子类就可以同时获得多个父类的所有功能。

1. \_\_str\_\_的使用

默认print打印一个对象的是时候，输出很多比较复杂的内容，可以使用\_\_str\_\_来修改输出内容。

比如在teacher.py里面添加：

def \_\_str\_\_(self):

return 'hello guozhang'

然后调用：print（Teacher()） 即可输出上面的话了。

1. \_\_iter\_\_的使用

如果一个类想要被用于for…in 循环，则必须实现\_\_iter\_\_方法。

class Teacher(object):

def \_\_init\_\_(self):

self.a,self.b=0,1

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

self.a,self.b=self.b,self.a+self.b

if self.a>100000:

raise StopIteration()

return self.a

调用方法：

for n in Teacher():

print(n)

结果：

1

1

2

3

5

8

13

21

34

55

89

144

233

377

610

987

1597

2584

4181

6765

10946

17711

28657

46368

75025

1. \_\_getitem\_\_的使用

使类像数组一样使用

class Fib(object):

def \_\_getitem\_\_(self, n):

a, b = 1, 1

for x in range(n):

a, b = b, a + b

return a

>>> f = Fib()

>>> f[0]

1

>>> f[1]

1

>>> f[2]

2

>>> f[3]

3

>>> f[10]

89

>>> f[100]

573147844013817084101

2018年11月27日22:45:35

1. range函数

创建一个整数列表，用在for循环中。

range(start, stop[, step])

参数说明：

start: 计数从 start 开始。默认是从 0 开始。例如range（5）等价于range（0， 5）;

stop: 计数到 stop 结束，但不包括 stop。例如：range（0， 5） 是[0, 1, 2, 3, 4]没有5

step：步长，默认为1。例如：range（0， 5） 等价于 range(0, 5, 1)

例子：

x=[i\*2 for i in range(1,11)]

print(x)

结果：[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

类似于numpy的arange

2018年11月27日23:47:27

1. numpy 使多次生成的随机数相同

比如一个方法里面调用了random方法，默认情况下每次调用这个方法生成的随机数都不相同。

如何相同呢？可以使用numpy.random.seed()

如果在seed()中传入的数字相同，那么接下来使用random()或者rand()方法所生成的随机数序列都是相同的（仅限使用一次random()或者rand()方法，第二次以及更多次仍然是随机的数字）

2018年12月6日21:31:10

1. 使用graphviz画决策树时要安装graphviz，这个不是用pip或者在pychram里面简单就能安装上的，要去官网下载一个安装包msi，然后安装好后要配置环境变量。

下载地址

<https://graphviz.gitlab.io/_pages/Download/Download_windows.html>

安装目录：C:\Program Files (x86)\Graphviz2.38\bin

将其配置到系统环境变量的path里面，

判断是否安装好：命令行输入：dot -version

然后在pycharm里面再次安装graphviz，接着运行代码，发现一直报错，提示问你是否确定安装了graphviz的程序，真实无语。

后来看到网友的提示，把pycharm重新启动一下即没问题了。

# coding:utf-8

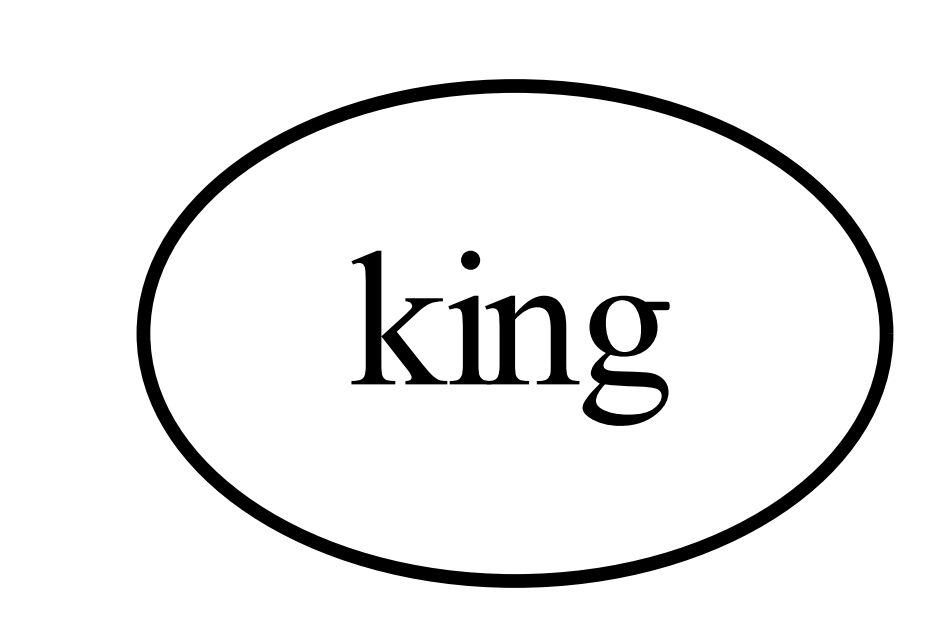
from graphviz import Digraph

dot = Digraph(comment='The Round Table')

dot.node('A', 'king')

dot.view()

运行后会将结果输出到pdf文件里面，并打开pdf



2018年12月8日23:01:37

1. python类的构造方法的特殊用法

之前使用 \_\_init\_\_来设计类的构造方法，只是用了最基本的用法，然而看了机器学习上的构造方法的使用，似乎和我之前的用法不一样。

别人定义决策树的方法是：

clf=tree.DecisionTreeClassifier(criterion="entropy",random\_state=30)

他在定义决策树时，构造方法里面对参数名也定义了，让参数等于某个对应的参数。而不是直接按顺序输入参数。

之前按顺序输入参数的用法：

class Student(object):

def \_\_init\_\_(self,

name,

age,

score):

self.name=name

self.age=age

self.score=score

调用：st=Student(“guozhang”,19,30)

这样明显和决策树定义不同。通过研究发现可以在构造方法里面输入参数名。并且参数顺序可以打乱。

st=Student(age=44,name="guozhang"，score=88)

这样是没有问题的，

另外别的决策树定义的时候只定义了几个参数而已，实际上决策树有很多个参数。那么如何实现的呢？实际上在类的构造方法设计的时候就可以给参数设计一个默认值，那么这个带默认值的参数在创建对象的时候就可以不用输入了。

class Student(object):

def \_\_init\_\_(self,

name,

age,

score=90):

self.name=name

self.age=age

self.score=score

调用：st=Student(age=44,name="guozhang")

这样也是可以的。但是如果某个参数没有默认值，那么在定义对象的时候就必须赋值，否则报错。

2018年12月11日14:40:37

1. numpy中的linespace是生成从起点到终点的指定数量的数据

而numpy中的arange是生成从起点到终点的指定步长的等差数列，默认步长为1

1. numpy的sort进行排序

可以通过设置axis属性进行选择是列排序还是行排序

import numpy as np

# a=[[14,13,12,11],[10,9,8,7],[6,5,4,3]]

# print(a)

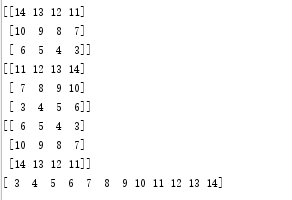
a=np.arange(14,2,-1).reshape(3,4)

print(a)

print(np.sort(a))

print(np.sort(a,axis=0))

print(np.sort(a,axis=None))



可以看出，sort没有axis属性时默认排序是对行维度进行排序，

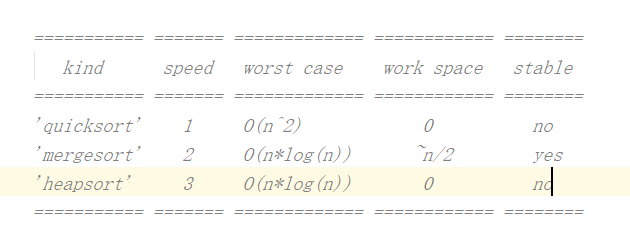
当axis=0时，是对列维度进行排序

而axis=None是对所有数据进行排序。

转到sort的定义：



可以看到axis默认值是-1，而且还有排序算法设置，kind默认为quicksort，即快排序。



可以看出除了快速排序还有归并排序和堆排序。

1. numpy的rand方法是生成指定行和列数的随机数。比如rand(80,1)是生成80行1列的随机数。
2. numpy的ravel方法

这个方法是将多维数组降维成1维数组。

import numpy as np

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

print(np.ravel(x))

输出：[1 2 3 4 5 6]

1. numpy的ravel（）和flatten（）函数比较

功能

两个函数的功能都是将多维数组降为一维。

用法

import numpy as np

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

print(np.ravel(x))

print(x.ravel())

print(x.flatten())区别

ravel（）返回的是视图，意味着改变元素的值会影响原始数组；

flatten（）返回的是拷贝，意味着改变元素的值不会影响原始数组。

注意：ravel可以用对象调用，也可以用类名调用。而flatten只能用对象调用。



numpy的rand方法生成的随机数要设置每个维度的大小。

import numpy as np

a=np.random.rand(10,1)

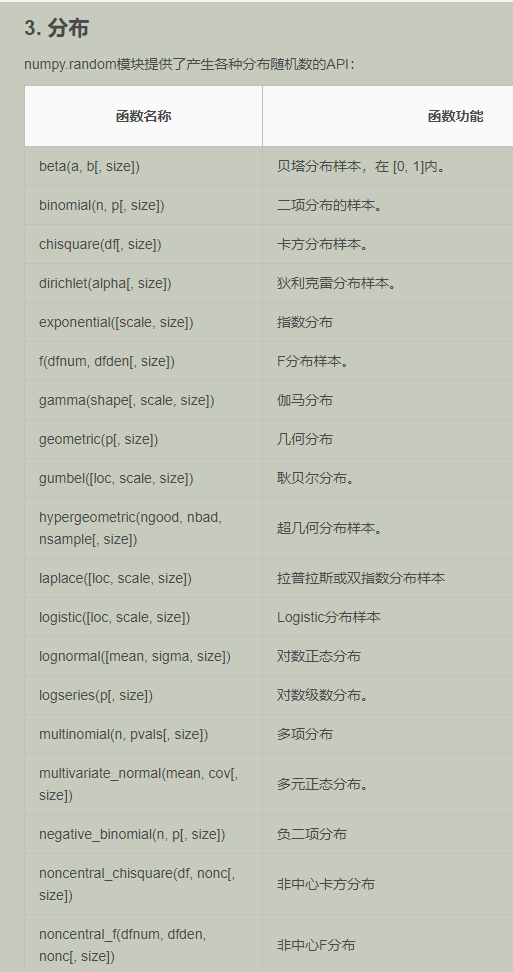
b=np.random.rand(10)

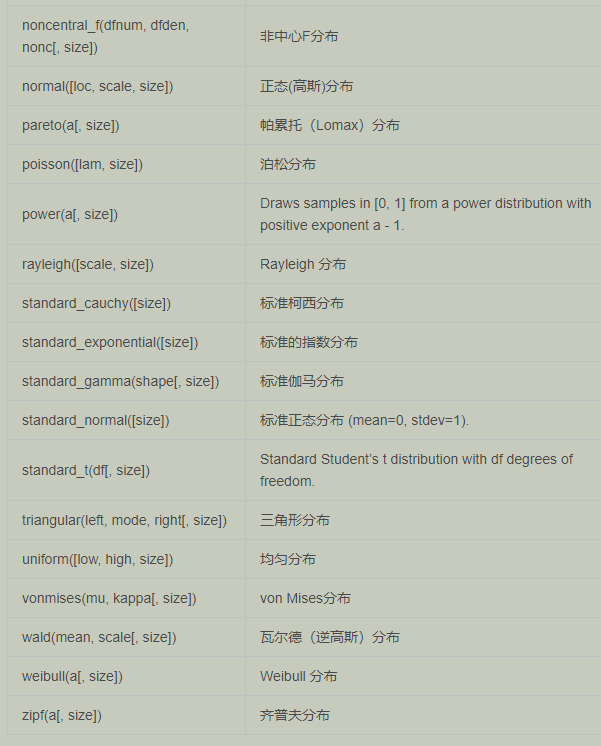
print(a)

如果rand()方法只有一个参数，那么生成的是一维数组，如果有两个参数，那么生成的是二维的数组。

菜菜的决策树那一章里面用了rand（80,1）和ravel来降维是用复杂了，可以直接用rand(80)生成一维数组，就不用降维了。

numpy中指定了随机分布的模型的方法可以直接设置随机数个数。比如正态分布。





import numpy as np

import matplotlib.pyplot as plt

mu = 1 #期望为1

sigma = 3 #标准差为3

num = 10000 #个数为10000

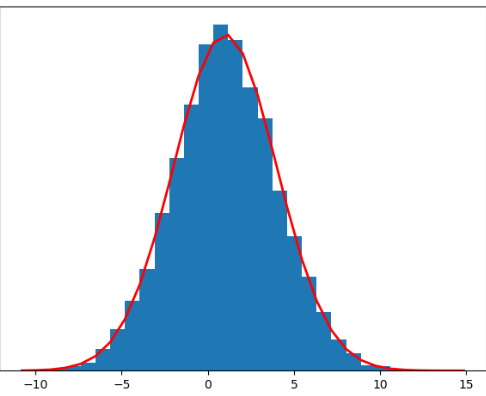
rand\_data = np.random.normal(mu, sigma, num)

count, bins, ignored = plt.hist(rand\_data, 30, normed=True)

plt.plot(bins, 1/(sigma \* np.sqrt(2 \* np.pi)) \*np.exp( - (bins - mu)\*\*2 / (2 \* sigma\*\*2)), linewidth=2, color='r')

plt.show()

正态分布要指定的参数有：方差，标准差和个数。



1. python的range的用法

numpy里面有个arange是创建等差数列的。而python自带的range也是类似的。

python range() 函数可创建一个整数列表，一般用在 for 循环中。

函数语法

range(start, stop[, step])

参数说明：

start: 计数从 start 开始。默认是从 0 开始。例如range（5）等价于range（0， 5）;

stop: 计数到 stop 结束，但不包括 stop。例如：range（0， 5） 是[0, 1, 2, 3, 4]没有5

step：步长，默认为1。例如：range（0， 5） 等价于 range(0, 5, 1)

不能直接用print输出range

比如：a=range(1,20,2)

print(a)

结果仍为：range(1, 20, 2)

应该输出下标：print(a[3])

for i in range(len(x)) :

print(x[i])

range不能直接打印，可以转换为list，而numpy中的arange可以直接输出。

a=range(1,20,2)

print(a[3])

b=list(a)

print(b)

输出：[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

1. python中的冒号的用法：
   1. 单冒号

[m : ] 代表列表中的第m+1项到最后一项

[ : n] 代表列表中的第一项到第n项

[m:n]代表从第m+1项到第n项

import numpy as np

m=3

n=6

a=np.arange(1,10)

print(a)

print(a[m:])

print(a[:n])

print(a[m:n])

结果：

[1 2 3 4 5 6 7 8 9]

[4 5 6 7 8 9]

[1 2 3 4 5 6]

[4 5 6]

7.2 双冒号的使用

双冒号其实是单冒号的简写。

s[i:j:k]是，根据该“片第从i到j与第k步”。何时i和j缺席，整个序列是和s[::k]意思是“每k个项目”。 示例 首先，让我们来初始化一个列表：

比如：（1）

c=range(10)[::2]

print(list(c))

结果：[0, 2, 4, 6, 8]

range(10)本身是0-9的步长为1的等差数列。加上[::2]就是步长为2的数列了。

（2）

c=range(10)[2:5]

print(list(c))

输出：[2, 3, 4]

这个就是单冒号的例子

（3）

c=range(10)[2:8:2]

print(list(c))

输出：[2, 4, 6]

这里是双冒号的例子，只不过起点、终点、步长都设置了。这里步长为2

（4）

c=range(10)[2::2]

print(list(c))

输出：[2, 4, 6, 8]

（5）

c=range(10)[2::]

print(list(c))

输出：[2, 3, 4, 5, 6, 7, 8, 9]

这里[2::]和[2:]是一样的

1. 总结下冒号容易混淆的地方

（1）[:n]和[::n]

前者是从一个列表的起点到第n个数

后者是从列表的起点到终点，步长为n取数。

（2）对于上面的冒号，不管是单冒号还是双冒号，基本模型都是**s[i:j:k]**

然后不管是I,j,k还是冒号都可以省略的 。看到冒号时把它还原成基本模型来分析。

1. python里面的很多略写的地方都要把它还原成基本模型来思考，否则单独记忆每个略写模块，很容易出错。

比如numpy.random.rand(10)不要仅仅把它想成输出10个随机数，虽然它确实是这样的。而且它是表明输出的一维数组的随机数，数组大小为10。应该把它想成rand(d0，d1…)，它可以设置很多维度。

如果不这么记，那么下次看到rand（2,3）就不知道是什么意思了。

2018年12月11日20:46:47

1. 使用numpy的newaxis来给数组增加维度

import numpy as np

x=np.arange(3,9)

print(x.shape)

y=x[:,np.newaxis]

print(y.shape)

输出：(6,)

(6, 1)

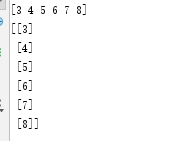
可以看出数组从一维变成二维了。

x=np.arange(3,9)

print(x)

y=x[:,np.newaxis]

print(y)



感觉是将行列式转置了。

1. 索引多维数组中的某一列

使用a[:,1]的格式；

a=np.arange(1,10).reshape(3,3)

b=a[:,1]

print(a)

print(b)

结果：[[1 2 3]

[4 5 6]

[7 8 9]]

[2 5 8]

可以看到输出原来数组的第二列

这时候把原来的第二列转换成行输出来了，如果要保留列的样式，使用newaxis方式

a=np.arange(1,10).reshape(3,3)

b=a[:,1][:,np.newaxis]

print(a)

print(b)

