Python基础以及数学库使用

1：类/继承类：

class People:

# 构造函数

def \_\_init\_\_(self,param1,param2):

self.name=param1

self.age=param2

#类的私有属性，python对属性的访问权限是通过名称来控制，一个属性由双下划线开头(\_\_)，该属性就无法被外部访问

self.\_\_weight = 50

#类中的方法，需要加上self

def say(self):

print('hello')

#继承：class subClassName(supClassName)

class Student(People):

#Python是动态语言，任何实例在运行期都可以动态地添加属性

#要限制添加的属性，就需要使用\_\_slots\_\_来实现

\_\_slots\_\_=('name','age','school') #限制student对象只有这三个属性

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

super(Student, self).\_\_init\_\_(name, age)

self.school = school

def say(self):

#调用父类的方法

super(Student,self).say()

print(self.name)

print(self.school)

2：numpy的一些使用：

import numpy as np

# 标准python的列表（list）中，元素的本质是对象,对于数值的运算比较浪费内存和cpu

#1：使用array创建

#通过array函数传递list对象

L = [1,2,3,4,5,6]# python里面的list

print(L)

a = np.array(L) #numpy提供的数组类型

print(a)

# 也可以转换嵌套的list

a1 = np.array([[1,2,3],[4,5,6]])

print(a1)

#数组大小可以通过shape属性获得

print(a.shape)

print(a1.shape)

#强制修改shape,并不是对数组进行了转置，而是改变每个方向的大小

a1.shape=3,2

print(a1)

# -1就是可以自动计算每个方向的大小

a1.shape = 2,-1

print(a1)

# reshape创建规定尺寸的新数组，但是两个对象是使用的同一个内存，会互相影响

a2 = a1.reshape((3,-1))

print(a2)

a2[0][1] = -1

# 数组元素可通过dtype获取数据类型

print(a2.dtype)

#转换时可指定类型

a3 = np.array([[1,2,3,4],[5,6,7,8]],dtype=np.float)

print(a3.dtype)

#astype进行数据类型的安全转换

a4 = a3.astype(np.int)

print(a4.dtype)

#不要强制修改元素类型，会存在精度问题

# 2：使用函数创建array

# 如果生成一定规则的数据，可使用np所提供的函数：

#arange函数，指定起始值，终止值（生成时不包含）和步长来创建数组

a5 = np.arange(2,5,0.5)

print(a5)

#linspace函数创建等差数列，指定起始值，终止值（生成时通过endpoint参数来控制是否包含；false：不包含；true：包含；）和元素个数来创建数组

a6 = np.linspace(2,5,10,endpoint=False)

print(a6)

#logspace函数创建等比数列

#frombuffer,fromstring,fromfile等函数可以从字节序列创建数组

# a7 = np.fromstring('abcd',dtype=np.int8)

#3：存取

a8 = np.arange(10)

#数组获取片段：[startIndex,endIndex)，与原来的那个数组共享一个内存，修改会有影响；但是通过数组下标获取的数据不会和原来的那个数组共享内存

print(a8[3:7])

#省略startIndex：表示从0开始；不包含endIndex

print(a8[:3])

#省略endIndex：表示从startIndex开始，一直到最后

print(a8[3:])

#步长step:[startIndex:endIndex:step],不包含step；{::-1]:表示翻转

print(a8[1:9:2])

#整数/布尔数组存取

a9 = np.logspace(0,9,10,base=2)

b = np.arange(0,10,2)

print(b)

#通过下标来取出数据，此时内存独立

b1 = a9[b]

print(b1)

#通过条件来取出数组，此时内存也是独立，修改一个不会影响另一个

b2 = a9[a9>20]

#二维数组

#生成二维数组

b3 = np.arange(0,60,10).reshape((-1,1))+np.arange(0,6) #行+列

print(b3)

#二维数组获取片段

## arr[(xIndex,xIndex2,...,xIndexn),(yIndex,yIndex2,...yIndexn)]：获取的是一个长度为n的数组，每一个元素的在二维数组中的位置为(xIndex,yIndex)

b4 = b3[(1,2,3),(2,3,4)]

print(b4)

## arr[xStartIndex:,[yIndex,yIndex2,...yIndexn]]：获取的是一个[行数-xStartIndex \* yIndex个数]的二维数组

b5 = b3[3:,[2,3,4]]

print(b5)

2：c或c++扩展python

* 错误或异常

Python解释器中一个重要的约定是：当函数失败是，它应该设置一个异常条件并返回错误值（NULL指针）。异常存储在解释器内部的静态全局变量中，如果为null，则不会发生异常。第二个全局变量存储异常的“关联值”（的第二个参数[raise](https://docs.python.org/2/reference/simple_stmts.html#raise)）。第三个变量包含堆栈追溯，以防错误源于Python代码。这三个变量是变量的Python的C-当量sys.exc\_type， sys.exc\_value和sys.exc\_traceback（见模块的部分 [sys](https://docs.python.org/2/library/sys.html#module-sys)在Python库参考）。

Python API来设置各种类型的异常函数：

* PyErr\_SetString()：

参数是一个异常对象和一个C字符串。异常对象通常是预定义的对象。C字符串只是错误的原因，并被转换为python字符串对象存储为异常的关联值。

* PyErr\_SetFromEeeno：

仅接受异常参数，并通过检查全局变量构建关联的值errno。最通用的函数是 [PyErr\_SetObject()](https://docs.python.org/2/c-api/exceptions.html#c.PyErr_SetObject)，它带有两个对象参数，即异常及其相关值。不需要[Py\_INCREF()](https://docs.python.org/2/c-api/refcounting.html#c.Py_INCREF)将对象传递给任何这些函数。

* 方法表和初始化函数

static PyMethodDef SpamMethods[] = {

...

{"system", spam\_system, METH\_VARARGS,

"Execute a shell command."},

...

{NULL, NULL, 0, NULL} /\* Sentinel \*/

};

METH\_VARARGS：这是一个标志，告诉编译器要用于C函数的调用约定。通常应始终为METH\_VARARGS或METH\_KEYWORDS；值为0表示使用了PyArg\_ParseTuple()废弃的变体

3：在其他应用程序嵌入python：

答：

嵌入应用程序能够使C/C++程序拥有python的某些功能；

嵌入python和扩展python类似，但是不完全相同。区别在于：扩展python时，应用程序的主程序依然时python解释器；而嵌入python，则主程序可能与python无关，即应用程序某些部分有时会调用python解释器运行一些python代码。

如果要嵌入python，那么就需要有一个自己的主程序。这个主程序要做的是初始化python解释器，至少必须调用Py\_Initialize()。一些可选的调用可以将命令行参数传递给python。然后就可以在应用程序的任何部分调用解释器。

Py\_SetProgramName()