Python

pyhon介绍

Python是一种功能强大且通用的编程语言，因其易用性而广受好评，它具有非常清晰的语法特点，适用于多种操作系统，目前在国际上非常流行，正在得到越来越多的应用，比如web开发，游戏开发，机器学习等。同时python也被称为胶水语言，可以通过编译的C语言来扩展，使得其有着非常好的扩展和应用范围。

python的在科学计算领域的应用

Python因其简单的、解释性的、交互式的、可移植的、面向对象的特点，非常易与初学者学习，同时有很多流行的科学计算工具包的支持，比如numpy，scipy，matplotlib等，使得其在科学计算领域变得非常强大且流行。

如果你有相关的python和numpy的编程经验，你可以跳过这一个部分。对于没有这方面经验的同学，这个部分将帮助你快速的入门python这个编程语言，让你知道在科学计算领域python的应用。

如果你们之前使用过Matlab，那么推荐你们看看numpy for Matlab users这个页面，这能够帮助你们从Matlab非常快地迁移到python当中。

下面就让我们进入到python这个神奇的世界，我们会一步一步教你如何熟练应用python解决问题。

你的第一个python程序

python非常简单，可以直接当一个小的计算器，比如我们希望计算100加200加300的结果，那么可以像下面这样。

```
100 + 200 + 300
```

```
600
```

使用print可以打印出任何你希望打出的结果

```
print('hello, world and hello deep learning!')
```

```
hello, world and hello deep learning!
```

基本的数据结构

和大多数其他的编程语言一样，Python拥有基本的数据类型，比如整形，浮点型，布尔型和字符型，这些数据类型我们下面会一一介绍。

空值和变量

空值是python里面一个特殊的值，用`None`表示。`None`不能理解为0，因为0是有意义的。

变量在程序中使用一个变量名表示，变量名必须是大小写英文、数字和_的组合，且不能数字开头，同时变量还可以重新赋值。

比如

```
x = 10
print(x)
x = x + 2
print(x)
```

```
10
12
```

整数

python可以处理任意大小的整数，包括负数，在程序中表现的方式和数学的写法一模一样，比如3或者-1等等。

我们可以用type来显示一个数据的类型，比如我们可以打印出3的数据类型。

```
print(type(3))
```

```
<class 'int'>
```

浮点型

所谓的浮点数，就是小数，比如1.23，3.24，-9.01等等。对于很大或者很小的浮点数，必须用科学计数法表示，把10用e替代，比如1.23×10^9就是`1.23e9`或者`12.3e8`,而0.012可以写成`1.2e-2`等等。

```
1.23e9
```

```
1230000000.0
```

```
12.3e8
```

```
1230000000.0
```

```
1.2e-2
```

```
0.012
```

字符型

字符串是以''或者""括起来的任意文本，比如

```
s = 'abc'
print(s)
print(len(s))
```

```
abc
3
```

如果字符串内部包含'或者"怎么办呢？我们可以使用\来表示转义，比如

```
print('I\'m \"OK\"')
```

```
I'm "OK"
```

同时\可以转义很多字符，比如\n表示换行，\t表示制表符等等。

```
a = 'hello'
b = 'world'
c = a + ' like ' + b
print(c)
```

```
hello like world
```

```
d = 'a: {}, b: {}'.format(23, 34)
print(d)
```

```
a: 23, b: 34
```

字符串有很多实用的方法，比如

```
s = 'hello'
print(s.capitalize())
print(s.upper())
print(s.rjust(7))
print(s.center(7))
print(s.replace('e', 'a'))
print('  world  '.strip())
```

```
Hello
HELLO
  hello
 hello
hallo
world
```

你可以在这个文档中找到更多的字符串方法。

布尔值

布尔值和布尔代数表示的含义完全相同，一个布尔值只有True和False两种值，要么是True，要么是False，在python中可以直接使用True和False来表示布尔值，注意大小写。

```
print(type(t))
```

```
<class 'bool'>
```

```
True
```

```
True
```

```
False
```

```
False
```

```
3 > 2
```

```
True
```

```
3 > 8
```

```
False
```

同时布尔值可以用and、or和not运算，比如

```
t = True
f = False
print(t and f)
print(t or f)
print(not t)
print(t != f)
```

```
False
True
False
True
```

小练习:
打印出如下的字符
I'm
"OK"

```
# 答案
print('I\'m\n\"OK\"')
```

```
I'm
"OK"
```

小练习
观察下面的代码，看看输出的结果是什么，然后运行一下进行检查。

```
a = 'ABC'
b = a
a = 'XYZ'
print(b)
```

容器

python中有几种容器类型，列表，字典，集合，元组。下面我们来依次进行介绍。

列表

列表是Python中非常常见的容器，可以看做一个序列，序列中的每个元素都可以修改，且可以是不同的数据类型，同时序列的长度也能够进行修改。

```
xs = [3, 1, 2]    # 创建一个序列
print(xs, xs[2])  # 打印出 "[3, 1, 2]"和序列中的第3个元素
print(xs[-1])     # 打印出序列的最后一个元素
xs[2] = 'foo'     # 将序列中第三个元素进行重新赋值
print(xs)
xs.append('bar')  # 在列表的最后加一个新元素
print(xs)
x = xs.pop()      # 移除列表的最后一个元素，同时将这个元素返回到一个变量中
print(x, xs)
```

```
[3, 1, 2] 2
2
[3, 1, 'foo']
[3, 1, 'foo', 'bar']
bar [3, 1, 'foo']
```

你可以找到更多的列表使用方法，在以下文档中。

小练习
用索引取出下面的指定元素

```
L = [
    ['Apple', 'Google', 'Microsoft'],
    ['Java', 'Python', 'Ruby', 'PHP'],
    ['Adam', 'Bart', 'Lisa']
]
# 'Apple'

# 'Python'

# 'Lisa'
```

```
# 答案
print(L[0][0]) # Apple
print(L[1][1])
print(L[2][2])
```

```
Apple
Python
Lisa
```

切片

除了一次访问列表中的一个元素，Python还提供了一种更高效的使用方式叫做切片。
下面我们举几个例子。

```
nums = [0, 1, 2, 3, 4]
print(nums)
print(nums[2:4]) # 得到一个切片，下标从2到4，但是不包括4
print(nums[2:]) # 得到一个切片，下标从2到列表结束
print(nums[:2]) # 得到一个切片，下标从开始到2，不包括2
print(nums[:]) # 得到整个列表的元素
print(nums[:-1]) # 得到一个切片，下标从开始到最后一个元素，不包括最后一个元素
nums[2:4] = [8, 9] # 将列表中下标从2到4(不包括4)的切片替换成新的元素
print(nums)
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

小练习
从列表nums中取出`[0, 1, 3, 4]`

```
nums = [0, 1, 2, 3, 4]
# 得到[0, 1, 3, 4]
```

```
# 答案
print(nums[:2] + nums[3:])
```

```
[0, 1, 3, 4]
```

循环

你可以对列表中的元素进行循环，就是从列表的开始遍历到列表的结束，比如

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```

```
cat
dog
monkey
```

如果想取得每个元素的下标，可以通过调用内置函数`enumerate`来实现。

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#{}: {}'.format(idx + 1, animal))
```

```
#1: cat
#2: dog
#3: monkey
```

如果我们需要改变列表中元素的值，我们可以通过循环遍历去访问到每个元素，然后进行修改，比如

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

```
[0, 1, 4, 9, 16]
```

选看
我们有一种更高级的列表循环方法,可以避免我们创建一个空列表进行元素的添加，上面的例子可以等价于

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)
```

```
[0, 1, 4, 9, 16]
```

练习
求$1+2+3\cdots+100$的结果
提示：使用range很方便的创建长的序列

```
# 答案
total = 0
for i in range(101):
    total += i
print(total)
```

```
5050
```

集合

集合里面的元素没有顺序关系，同时里面也没有重复的元素，我们可以看看下面的例子。

```
animals = {'cat', 'dog'}
print('cat' in animals)   # 验证该元素是否在集合中
print('fish' in animals)
animals.add('fish')       # 在集合里面添加元素
print('fish' in animals)
print(len(animals))       # 打印出集合的元素个数
animals.add('cat')
print(len(animals))
animals.remove('cat')     # 从集合里面移除某个元素
print(len(animals))
```

```
True
False
True
3
3
2
```

可以在以下文档查看关于集合的所有信息。

字典

字典里面存在的都是数据对，这些数据都是以(key, value)的形式存在，且存在映射关系，可以看看下面的例子。

```
d = {'cat': 'cute', 'dog': 'furry'}  # 创造一个字典
print(d['cat'])       # 取得字典中的某个元素
print('cat' in d)     # 判断字典中key是否有该元素
```

```
cute
True
```

```
d['fish'] = 'wet'     # 字典中添加一个新元素
print(d['fish'])      # 打印出字典中的元素
```

```
wet
```

```
print(d.get('monkey', 'N/A'))  # 如果元素在字典的key中，返回其对应的值，否则返回get中的值
print(d.get('fish', 'N/A'))
```

```
N/A
wet
```

```
del d['fish']         # 从字典中移除某个元素
print(d.get('fish', 'N/A')) # fish被移除之后，不能显示出'wet'
```

```
N/A
```

```
print(d['pig'])
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-83-ad579f5a4027> in <module>()
----> 1 print(d['pig'])
```

```
KeyError: 'pig'
```

元组

一个元组和列表类似，但是元组是无法改变的。同时一个最大的不同是元组可以作为字典中的key，而列表不可以。

```
d = {(x, x + 1): x for x in range(10)} # 使用元组创造一个字典
t = (5, 6)        # 创造一个元组
print(type(t))    # 打印出元组的类型
print(d[t])       # 打印出字典中对应于(5, 6)的元素
print(d[(1, 2)])  # 打印出字典中对应于(1, 2)的元素
```

```
<class 'tuple'>
5
1
```

下面的文档有更多关于元素的信息。

函数

函数通过`def`来定义，将一些重复使用的操作封装在一起，每次不需要重复写相同的代码，只需要调用函数就可以了，比如

```
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))
```

关于函数的更多信息，可以查阅一下文档。

小练习
写一个函数，传入x，返回他的绝对值。

```
## 答案
def abs_fun(x):
    if x >= 0:
        return x
    else:
        return -x

print(abs_fun(3))
print(abs_fun(-2))
```

```
3
2
```

类

类和实例是面向对象中重要的概念，必须牢记类是抽象的模板，而实例是根据类创建出来的一个个具体的“对象”，每个对象都拥有相同的方法，但各自的数据可能不同，下面是一个定义类的简单例子

```
class Student(object):

    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_score(self):
        print('{}: {}'.format(self.name, self.score))

a = Student('mike', 99)
b = Student('lisa', 80)
```

```
a.print_score()
b.print_score()
```

```
mike: 99
lisa: 80
```

下面的文档有更多关于类的信息。

小练习
给这个类增加一个函数(method)，类似print_score，使得这个函数能够输出学生的等级

```
class Student(object):

    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_score(self):
        print('{}: {}'.format(self.name, self.score))

    def get_grade(self):
        '''
        90以上的是A，60到90分的是B，60分以下的是C
        '''
        pass
```

```
# 答案
class Student(object):

    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_score(self):
        print('{}: {}'.format(self.name, self.score))

    def get_grade(self):
        if self.score >= 90:
            return 'A'
        elif self.score >= 60:
            return 'B'
        else:
            return 'C'
```

Numpy

numpy是Python中科学计算的核心库，其提供了一个高表现的、高位矩阵计算工具，如果你对MATLAB熟悉，你可以找到这个教程，帮助你从MATLAB转到Numpy。

数组(Arrays)

数组就是一个任意维度的矩阵，数组的维度就是矩阵的秩，数组内部的数据类型和python的数据类型一致，我们可以看看下面的例子。

```
import numpy as np
```

```
a = np.array([1, 2, 3])   # 创建一个一维数组
print(type(a))            # 打印出元素类型
print(a.dtype)
print(a.shape)            # 打印出元素的维度
```

```
<class 'numpy.ndarray'>
int64
(3,)
```

```
print(a[0], a[1], a[2])   # 访问数组中的元素
a[0] = 5                  # 改变元素中某个位置的值
print(a)
```

```
1 2 3
[5 2 3]
```

```
b = np.array([[1,2,3],[4,5,6]])    # 创造一个二维数组
print(b.shape)                     # 打印出数组的维度
print(b[0, 0], b[0, 1], b[1, 0])   # 访问其中的元素
```

```
(2, 3)
1 2 4
```

数组索引

Numpy也支持多种类型的数据索引，比如切片，整数值索引和布尔值索引，下面可以看几个例子。

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

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
b = a[:2, 1:3]
print(b)
```

```
[[2 3]
 [6 7]]
```

```
row_r1 = a[1, :]
print(row_r1)
```

```
[5 6 7 8]
```

```
print(a[0, 0], a[1, 1])
```

```
1 6
```

```
bool_idx = a > 6
print(bool_idx)
```

```
[[False False False False]
 [False False  True  True]
 [ True  True  True  True]]
```

```
print(a[bool_idx])
```

```
[ 7  8  9 10 11 12]
```

数学计算

numpy中支持很多线性代数的数学运算，下面举几个简单的例子。

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

```
print(x)
```

```
[[ 1.  2.]
 [ 3.  4.]]
```

```
print(y)
```

```
[[ 5.  6.]
 [ 7.  8.]]
```

```
print(x + y)
```

```
[[  6.   8.]
 [ 10.  12.]]
```

```
print(x * y)
```

```
[[  5.  12.]
 [ 21.  32.]]
```

```
print(x / y)
```

```
[[ 0.2         0.33333333]
 [ 0.42857143  0.5       ]]
```

```
print(np.dot(x, y)) # 矩阵乘法
```

```
[[ 19.  22.]
 [ 43.  50.]]
```

一些有用的函数

numpy中有很多有用的函数，我们举几个简单的例子，更多的信息可以查阅下面的文档。

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

```
[[1 2]
 [3 4]]
```

```
print(np.sum(x))
```

```
10
```

```
print(np.sum(x, axis=0))
```

```
[4 6]
```

```
print(np.sum(x, axis=1))
```

```
[3 7]
```

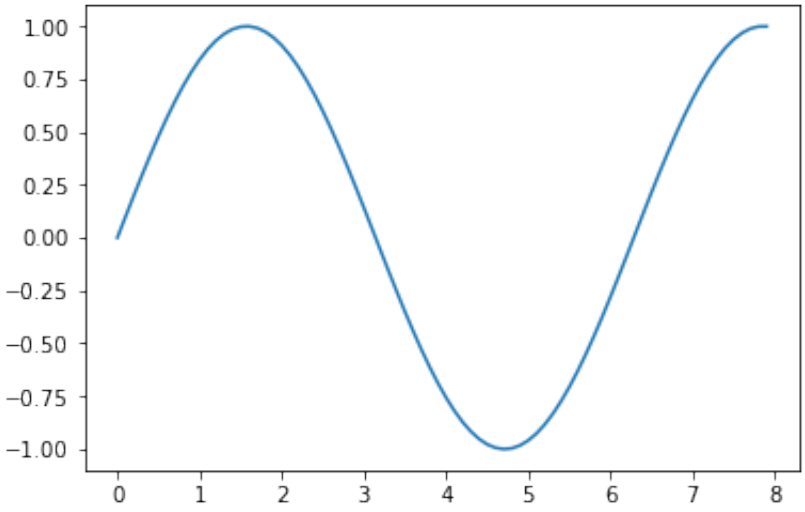
Matplotlib

这是Python中的画图工具，可以画2D的图像，也可以画3D的图像，下面举一个简单的例子。

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

```
x = np.arange(0, 8, 0.1)
y = np.sin(x)
```

```
plt.plot(x, y)
plt.show()
```



Pillow

pillow是Python中官方的图像处理库，有很多对图片的操作，使用`pip install Pillow`进行安装，更多的信息，可以通过官方文档查看，下面举一个小例子进行展示。

```
from PIL import Image
```

```
img = Image.open('./jim.jpg')
```

```
img
```


```
img_array = np.array(img)
print(img_array)
```

```
[[[ 76 121 162]
  [ 82 125 168]
  [ 81 124 167]
  ...,
  [ 64 111 155]
  [ 61 108 152]
  [ 61 105 144]]

 [[ 71 114 156]
  [ 64 105 149]
  [ 65 108 151]
  ...,
  [ 63 110 152]
  [ 60 109 152]
  [ 62 106 145]]

 [[ 73 118 159]
  [ 65 108 150]
  [ 69 112 155]
  ...,
  [ 70 115 157]
  [ 68 115 159]
```

```
  [ 66 110 149]]

 ...,
 [[116 117  51]
  [127 132  29]
  [132 134  35]
  ...,
  [135 135  23]
  [131 133  23]
  [130 132  25]]

 [[122 123  55]
  [128 133  31]
  [130 132  31]
  ...,
  [134 133  24]
  [134 133  24]
  [130 132  23]]

 [[124 123  67]
  [122 126  29]
  [125 126  32]
  ...,
  [130 129  20]
  [131 130  21]
  [132 131  23]]]
```

```
img.transpose(Image.FLIP_LEFT_RIGHT)
```

```
from PIL import ImageEnhance
```

```
enhancer = ImageEnhance.Brightness(img)
new_img = enhancer.enhance(1.5)
```

```
new_img
```


J.W.