Python语言基础

本章是python语言的基础部分，也是后续内容的基础。

1 Python数据类型

1.1 字符串

在Python中用引号引起来的字符集称之为字符串，比如：'hello'、"my Python"、"2+3"等都是字符串
Python中字符串中使用的引号可以是单引号、双引号跟三引号

In [ ]:
```
print ('hello world!')
```

In [ ]:
```
c = 'It is a "dog"!'
print (c)
```

In [ ]:
```
c1= "It's a dog!"
print (c1)
```

In [ ]:
```
c2 = """hello
world
!"""
print (c2)
```

- 转义字符''

转义字符\可以转义很多字符，比如\n表示换行，\t表示制表符，字符\本身也要转义，所以\表示的字符就是\

In [ ]:
```
print ('It\'s a dog!')
print ("hello world!\nhello Python!")
print ('\\\t\\')
```

原样输出引号内字符串可以使用在引号前加r

In [ ]:
```
print (r'\\\t\\')
```

- 子字符串及运算

In [ ]:
```
s = 'Python'
print( 'Py' in s)

print( 'py' in s)
```

取子字符串有两种方法，使用[]索引或者切片运算法[:]，这两个方法使用面非常广

In [ ]:
```
print (s[2])
```

In [ ]:
```
print (s[1:4])
```

- 字符串连接与格式化输出

In [ ]:
```
word1 = '"hello"'
word2 = '"world"'
```

```
sentence = word1.strip('"') + ' ' + word2.strip('"') + '!'

print( 'The first word is %s, and the second word is %s' %(word1, word2))
print (sentence)
```

1.2 整数与浮点数

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

In [ ]:
```
i = 7
print (i)
```

In [ ]:
```
7 + 3
```

In [ ]:
```
7 - 3
```

In [ ]:
```
7 * 3
```

In [ ]:
```
7 ** 3
```

In [ ]:
```
7 / 3#Python3之后，整数除法和浮点数除法已经没有差异
```

In [ ]:
```
7 % 3
```

In [ ]:
```
7//3
```

浮点数

In [ ]:
```
7.0 / 3
```

In [ ]:
```
3.14 * 10 ** 2
```

其它表示方法

In [ ]:
```
0b1111
```

In [ ]:
```
0xff
```

In [ ]:
```
1.2e-5
```

更多运算

In [ ]:
```
import math

print (math.log(math.e)) # 更多运算可查阅文档
```

1.3 布尔值

In [ ]:
```
True
```

In [ ]:
```
False
```

In [ ]:
```
True and False
```

In [ ]:
```
True or False
```

```
not True
```

```
True + False
```

```
18 >= 6 * 3 or 'py' in 'Python'
```

```
18 >= 6 * 3 and 'py' in 'Python'
```

```
18 >= 6 * 3 and 'Py' in 'Python'
```

1.4 日期时间

```
import time

now = time.strptime('2016-07-20', '%Y-%m-%d')
print (now)
```

```
time.strftime('%Y-%m-%d', now)
```

```
import datetime

someDay = datetime.date(1999,2,10)
anotherDay = datetime.date(1999,2,15)
deltaDay = anotherDay - someDay
deltaDay.days
```

还有其他一些datetime格式

- 查看变量类型

```
type(None)
```

```
type(1.0)
```

```
type(True)
```

```
s="NoneType"
type(s)
```

- 类型转换

```
str(10086)
```

```
?float()
```

```
float(10086)
```

```
int('10086')
```

In [ ]:
```
complex(10086)
```

2 Python数据结构

列表（list）、元组（tuple）、集合（set）、字典（dict）

用来存储一连串元素的容器，列表用[]来表示，其中元素的类型可不相同。

```
mylist= [0, 1, 2, 3, 4, 5]
print (mylist)
```

列表索引和切片

In [ ]:
```
# 索引从0开始，含左不含右
print ('[4]=', mylist[4])
print ('[-4]=', mylist[-4])
print ('[0:4]=', mylist[0:4])
print ('[:4]=', mylist[:4])#dddd
print( '[4:]=', mylist[4:])
print ('[0:4:2]=', mylist[0:4:2])
print ('[-5:-1:]=', mylist[-5:-1:])
print ('[-2::-1]=', mylist[-2::-1])
```

修改列表

In [ ]:
```
mylist[3] = "小月"
print (mylist[3])

mylist[5]="小楠"
print (mylist[5])

mylist[5]=19978
print (mylist[5])
```

In [ ]:
```
print (mylist)
```

插入元素

In [ ]:
```
mylist.append('han') # 添加到尾部
mylist.extend(['long', 'wan'])
print (mylist)
```

In [ ]:
```
scores = [90, 80, 75, 66]
mylist.insert(1, scores) # 添加到指定位置
mylist
```

In [ ]:
```
a=[]
```

删除元素

In [ ]:
```
print (mylist.pop(1)) # 该函数返回被弹出的元素，不传入参数则删除最后一个元素
print (mylist)
```

判断元素是否在列表中等

In [ ]:
```
print( 'wan' in mylist)
print ('han' not in mylist)
```

In [ ]:
```
mylist.count('wan')
```

In [ ]:
```
mylist.index('wan')
```

range函数生成整数列表

In [ ]:
```
print (range(10))
print (range(-5, 5))
print (range(-10, 10, 2))
print (range(16, 10, -1))
```

2.2 元组(tuple)

元组类似列表，元组里面的元素也是进行索引计算。列表里面的元素的值可以修改，而元组里面的元素的值不能修改，只能读取。元组的符号是()。

In [ ]:
```
studentsTuple = ("ming", "jun", "qiang", "wu", scores)
studentsTuple
```

In [ ]:
```
try:
    studentsTuple[1] = 'fu'
except TypeError:
    print ('TypeError')
```

In [ ]:
```
scores[1]= 100
studentsTuple
```

In [ ]:
```
'ming' in studentsTuple
```

In [ ]:
```
studentsTuple[0:4]
```

In [ ]:
```
studentsTuple.count('ming')
```

In [ ]:
```
studentsTuple.index('jun')
```

In [ ]:
```
len(studentsTuple)
```

2.3 集合(set)

Python中集合主要有两个功能，一个功能是进行集合操作，另一个功能是消除重复元素。 集合的格式是：set()，其中()内可以是列表、字典或字符串，因为字符串是以列表的形式存储的

In [ ]:
```
studentsSet = set(mylist)
print (studentsSet)
```

In [ ]:
```
studentsSet.add('xu')
print (studentsSet)
```

In [ ]:
```
studentsSet.remove('xu')
print (studentsSet)
```

In [ ]:
```
a = set("abcnmaaaaggsng")
print ('a=', a)
```

In [ ]:
```
b = set("cdfm")
print ('b=', b)
```

In [ ]:
```
#交集
x = a & b
print( 'x=', x)
```

In [ ]:
```
#并集
y = a | b
print ('y=', y)
#差集
z = a - b
```

```
print( 'z=', z)
#去除重复元素
new = set(a)
print( z)
```

2.4字典(dict)

Python中的字典dict也叫做关联数组，用大括号{}括起来，在其他语言中也称为map，使用键-值（key-value）存储，具有极快的查找速度，其中key不能重复。

```
k = {"name":"weiwei", "home":"guilin"}
print (k["home"])
```

```
print( k.keys())
print( k.values())
```

添加、修改字典里面的项目

```
k["like"] = "music"
k['name'] = 'guangzhou'
print (k)
```

```
k.get('edu', -1) # 通过dict提供的get方法，如果key不存在，可以返回None，或者自己指定的value
```

删除key-value元素

```
k.pop('like')
print (k)
```

2.5 列表、元组、集合、字典的互相转换

```
type(mylist)
```

```
tuple(mylist)
```

```
list(k)
```

```
zl = zip(('A', 'B', 'C'), [1, 2, 3, 4]) # zip可以将列表、元组、集合、字典‘缝合’起来
print (zl)
print (dict(zl))
```

3 Python控制流

在Python中通常的情况下程序的执行是从上往下执行的，而某些时候我们为了改变程序的执行顺序，使用控制流语句控制程序执行方式。Python中有三种控制流类型：顺序结构、分支结构、循环结构。

另外，Python可以使用分号";"分隔语句，但一般是使用换行来分隔；语句块不用大括号"{}"，而使用缩进（可以使用四个空格）来表示

3.1 顺序结构

In [ ]:
```
s = '7'
num = int(s) # 一般不使用这种分隔方式
num -= 1 # num = num - 1
num *= 6 # num = num * 6
print (num)
```

3.2 分支结构：Py[illegible]

if <True or Flase表达式>: 执行语句块 elif <True or Flase表达式>： 执行语句块 else： # 都不满足 执行语句块 # elif子句可以有多条，elif和else部分可省略

In [ ]:
```
salary = 1000
if salary > 10000:
    print ("Wow!!!!!!!")
elif salary > 5000:
    print ("That's OK.")
elif salary > 3000:
    print ("5555555555")
else:
    print (".........")
```

3.3 循环结构

while 循环

while <True or Flase表达式>: 循环执行语句块 else： # 不满足条件 执行语句块 #else部分可以省略

In [ ]:
```
a = 1
while a < 10:
    if a <= 5:
        print (a)
    else:
        print ("Hello")
    a = a + 1
else:
    print ("Done")
```

- for 循环

for (条件变量) in (集合)： 执行语句块 # “集合”并不单指set，而是“形似”集合的列表、元组、字典、数组都可以进行循环 # 条件变量可以有多个

In [ ]:
```
heights = {'Yao':226, 'Sharq':216, 'AI':183}
for i in heights:
    print (i, heights[i])
```

In [ ]:
```
for key, value in heights.items():
    print(key, value)
```

In [ ]:
```
total = 0
for i in range(1, 101):
    total += i#total=total+i
print (total)
```

3.4 break continue和pass

break:跳出循环 continue:跳出当前循环,继续下一次循环 pass:占位符，什么也不做

In [ ]:
```
for i in range(1, 5):
    if i == 3:
        break
    print (i)
```

In [ ]:
```
for i in range(1, 5):
    if i == 3:
        continue
    print (i)
```

In [ ]:
```
for i in range(1, 5):
    if i == 3:
        pass
    print (i)
```

3.5 列表生成式

三种形式

- [<表达式> for (条件变量) in (集合)]
- [<表达式> for (条件变量) in (集合) if <'True or False'表达式>]
- [<表达式> if <'True or False'表达式> else <表达式> for (条件变量) in (集合)]

In [ ]:
```
fruits = ['"Apple', 'Watermelon', '"Banana"']
[x.strip('"') for x in fruits]
```

In [ ]:
```
# 另一种写法
test_list=[]
for x in fruits:
    x=x.strip('"')
    test_list.append(x)
test_list
```

In [ ]:
```
[x ** 2 for x in range(21) if x%2]
```

In [ ]:
```
# 另一种写法
test_list=[]
for x in range(21):
    if x%2:
        x=x**2
        test_list.append(x)
test_list
```

In [ ]:
```
[m + n for m in 'ABC' for n in 'XYZ']
```

In [ ]:
```
# 另一种写法
test_list=[]
for m in 'ABC':
    for n in 'XYZ':
        x=m+n
        test_list.append(x)
test_list
```

In [ ]:
```
d = {'x': 'A', 'y': 'B', 'z': 'C' }
[k + '=' + v for k, v in d.items()]
```

In [ ]:
```
# 另一种写法
test_list=[]
for k, v in d.items():
    x=k + '=' + v
    test_list.append(x)
test_list
```

4 Python函数

函数是用来封装特定功能的实体，可对不同类型和结构的数据进行操作，达到预定目标

4.1 调用函数

- Python内置了很多有用的函数，我们可以直接调用，进行数据分析时多数情况下是通过调用定义好的函数来操作数据的

In [ ]:
```
str1 = "as"
int1 = -9
```

```
print (len(str1))
print (abs(int1))
```

In [ ]:
```
fruits = ['Apple', 'Banana', 'Melon']
fruits.append('Grape')
print (fruits)
```

4.2 定义函数

当系统自带函数不足以完成指定的功能时，需要用户自定义函数来完成。

def 函数名(): 函数内容 函数内容 <return 返回值>

In [ ]:
```
def my_abs(x):
    if x >= 0:
        return x
    else:
        return -x

my_abs(-9)
```

可以没有return

In [ ]:
```
def filter_fruit(someList, d):
    for i in someList:
        if i == d:
            someList.remove(i)
        else:
            pass

print (filter_fruit(fruits, 'Melon'))
print (fruits)
```

多个返回值的情况

In [ ]:
```
def test(i, j):
    k = i * j
    return i, j, k

a , b , c = test(4, 5)
print (a, b , c)
type(test(4, 5))
```

4.3 高阶函数

- 把另一个函数作为参数传入一个函数，这样的函数称为高阶函数

函数本身也可以赋值给变量，函数与其它对象具有同等地位

In [ ]:
```
myFunction = abs
myFunction(-9)
```

- 参数传入函数

In [ ]:
```
def add(x, y, f):
    return f(x) + f(y)

add(7, -5, myFunction)
```

- 常用高阶函数

map/reduce: map将传入的函数依次作用到序列的每个元素，并把结果作为新的list返回；reduce把一个函数作用在一个序列[x1, x2, x3...]上，这个函数必须接收两个参数，reduce把结果继续和序列的下一个元素做累积计算

In [ ]:
```
myList = [-1, 2, -3, 4, -5, 6, 7]
map(abs, myList)
```

In [ ]:
```
from functools import reduce
def powerAdd(a, b):
    return pow(a, 2) + pow(b, 2)

reduce(powerAdd, myList) # 是否是计算平方和？
```

filter： filter()把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素

In [ ]:
```
def is_odd(x):
    return x % 3  # 0被判断为False，其它被判断为True

filter(is_odd, myList)
```

sorted: 实现对序列排序，默认情况下对于两个元素x和y，如果认为x < y，则返回-1，如果认为x == y，则返回0，如果认为x > y，则返回1

默认排序：数字大小或字母序（针对字符串）

In [ ]:
```
sorted(myList)
```

- 返回函数: 高阶函数除了可以接受函数作为参数外，还可以把函数作为结果值返回

In [ ]:
```
def powAdd(x, y):
    def power(n):
        return pow(x, n) + pow(y, n)
    return power

myF = powAdd(3, 4)
myF
```

In [ ]:
```
myF(2)
```

- 匿名函数：高阶函数传入函数时，不需要显式地定义函数，直接传入匿名函数更方便

In [ ]:
```
f = lambda x: x * x
f(4)
```

等同于：

In [ ]:
```
def f(x):
    return x * x
```

In [ ]:
```
map(lambda x: x * x, myList)
```

匿名函数可以传入多个参数

In [ ]:
```
reduce(lambda x, y: x + y, map(lambda x: x * x, myList))
```

返回函数可以是匿名函数

In [ ]:
```
def powAdd1(x, y):
    return lambda n: pow(x, n) + pow(y, n)

lamb = powAdd1(3, 4)
lamb(2)
```

其它

- 标识符第一个字符只能是字母或下划线，第一个字符不能出现数字或其他字符；标识符除第一个字符外，其他部分可以是字母或者下划线或者数字，标识符大小写敏感，比如name跟Name是不同的标识符。
- Python规范：

- 类标识符每个字符第一个字母大写；
- 对象\变量标识符的第一个字母小写，其余首字母大写，或使用下划线'_' 连接；
- 函数命名同普通对象。

- 关键字

关键字是指系统中自带的具备特定含义的标识符

In [ ]:
```
# 查看一下关键字有哪些，避免关键字做自定义标识符
import keyword
print (keyword.kwlist)
```

- 注释

Python中的注释一般用#进行注释

- 帮助

可以用？或者help()