数据科学之路：02-1Python快速入门（代码）

Python语言是一种解释型、面向对象、动态数据类型的高级程序设计语言,如今已成为绝大部分数据分析师的首选数据分析语言。其设计的核心理念是代码的易读性，以及允许编程者通过若干行代码轻松表达想法创意。举个例子，下面是用Python实现的经典的quicksort算法例子：

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)

print(quicksort([9,3,8,19,5,21,13]))
# 输出 "[3, 5, 8, 9, 13, 19, 21]"
```

Python版本

本文所以代码采用Python3.6进行编写，使用**python --version**命令可查看版本。

基本数据类型

- 数字：整型(int)和浮点型(float)
- 布尔：True和Flase
- 字符串：str

变量：储存数据类型的工具，通过=进行赋值。

- **命名规则**: 大小写字母、数字、下划线和汉字等字符及组合
- **注意事项**：大小写敏感、首字符不能是数字、不与保留字相同

```
# int
abc = 1

# float
cba = 9.99

# booleans
true_boolean = True
false_boolean = False

# string
my_name = "Leandro Tk"
```

复合数据类型

Python有以下四种复合数据类型：列表（lists）、元组（tuples)、字典（dictionaries）和 集合（sets）。

列表（list）

列表是Python中的数组，但是列表长度可变，且列表中的元素可变，能包含不同类型元素。

```
# 创建列表
a = [1,2,3]
# 通过list()方法
b = list((12,3,4))
```

列表常用方法

- append()：末尾增加元素
- remove():删除元素
- sort():排序(默认从小到大)
- reverse()：反转
- clear()：清空
- count()：统计元素出现的次数
- index()：求下标

- pop()：弹出最后一个数据
- insert()：插入新数据
- insert(index下标，obj)：索引位置插入

索引:Python语言中所有的索引都是从0开始计数的，如果列表中有n个元素，那么最后一个元素的索引是$n-1$。

```
x = [1,2,3,4,5,6]
print(x[0]) #输出‘1’
print(x[-1]) #输出‘6’
```

切片:切片能够一次性获取列表中多个元素。

```
y = [6,5,4,3,2,1]
print(y[1:3]) #输出[5, 4]
print(y[:])  #输出[6, 5, 4, 3, 2, 1]
print(y[:-3])  #输出[6, 5, 4]
```

列表推导:将一种数据类型转换为另一种

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)  # 输出[0, 1, 4, 9, 16]
```

元组（tuples)

元组是一个值的有序列表（不可改变）。

```
# 创建元组
a = (1,2,3,4)
# 通过tuple()方法
b = tuple([4,3,2,1])
```

从很多方面来说，元组和列表都很相似。和列表最重要的不同在于，元组可以在字典中用作键，还可以作为集合的元素，而列表不行。

```
d = {(x, x + 1): x for x in range(10)}  # 用元组键创建字典
t = (5, 6)
print(type(t))    # 输出"<class 'tuple'>"
print(d[t])       # 输出"5"
print(d[(1, 2)])  # 输出"1"
```

元组的索引与切片和列表类似，这里就不一一介绍了。

字典（dictionaries）

字典是一种通过名字或者关键字引用的数据类型，这种结构类型也称之为映射。通过这种数据结构，我们可以使用数值型、字符型或其它类型的索引。字典的每个键值 (key=>value) 对用冒号 (:) 分割，每个对之间用逗号 (,) 分割，整个字典包括在花括号 ({})中。

```
## 创建字典
a = {'one':1,'two':2,'three':3}   #输出"{'one': 1, 'three': 3, 'two': 2}"

## 通过dict()方法
b = dict([('one',1),('two',2),('three',3),('four',4)])

c=dict(one=1,two=2,three=3)
```

常用方法

- keys():以列表返回一个字典所有的键
- values():以列表返回字典中的所有值
- items():以列表返回可遍历的(键, 值) 元组数组

```
print(a.keys())   #输出“dict_keys(['one', 'two', 'three'])”
print(b.values()) #输出“dict_values([1, 2, 3, 4])”
print(c.items())  #输出“dict_items([('one', 1), ('two', 2), ('three', 3)])#
```

字典推导:和列表推导类似，允许你方便地构建字典。

```
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)  #输出"{0: 0, 2: 4, 4: 16}"
```

集合（sets）

集合是独立不同个体的无序集合。

```
# 创建方法
a = {'1','2','3'}   #输出"{'1','2','3'}"

#  通过set()方法
b = set([1,2,3])   #输出”{1, 2, 3}“
c = set((1,2,3))   #输出”{1, 2, 3}“
d = set('123')     #输出”{'1','2','3'}“
e = set({'one':1,'two':2,'three':3})   #输出”{'one', 'three', 'two'}“
```

常用方法

- add():增加元素
- update():拆分元素传入集合
- remove():删除元素

集合操作

- 与：set1 & set2
- 或：set1 | set2
- 与非：set1 ^ set2
- 减：set1 - set2

面向对象

函数

Python函数使用**def**来定义函数。函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。灵活利用函数能提高应用的模块性，和代码的重复利用率。

```
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))
# 输出"negative", "zero", "positive"
```

参数

- 必需参数:必需参数须以正确的顺序传入函数,调用时的数量必须和声明时的一样。
- 关键字参数:关键字参数和函数调用关系紧密，函数调用使用关键字参数来确定传入的参数值。
- 默认参数:调用函数时，如果没有传递参数，则会使用默认参数。
- 不定长参数:处理比当初声明时更多的参数。

匿名函数

python 使**lambd**来创建匿名函数。所谓匿名，意即不再使用**def**语句这样标准的形式定义一个函数。

- lambda 只是一个表达式，函数体比 def 简单很多。
- lambda的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去。
- lambda 函数拥有自己的命名空间，且不能访问自己参数列表之外或全局命名空间里的参数。
- 虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

```
sum = lambda arg1, arg2: arg1 + arg2    #语法”lambda [arg1 [,arg2,.....argn]]:expression“
print ("相加后的值为 : ", sum( 10, 20 ))  #输出”相加后的值为 :  30“
```

类（classes）

Python从设计之初就已经是一门面向对象的语言，正因为如此，在Python中创建一个类和对象是很容易的。

面向对象技术简介

- **类(Class)**: 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。
- **类变量**：类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。

- **数据成员**：类变量或者实例变量, 用于处理类及其实例对象的相关的数据。
- **方法重写**：如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。
- **实例变量**：定义在方法中的变量，只作用于当前实例的类。
- **继承**：即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个Dog类型的对象派生自Animal类，这是模拟"是一个（is-a）"关系（例图，Dog是一个Animal）。
- **实例化**：创建一个类的实例，类的具体对象。
- **方法**：类中定义的函数。
- **对象**：通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。

```
class Greeter(object):

    # 构造函数
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # 实例方法
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)

g = Greeter('Fred')  # 构建Greeter类的一个实例
g.greet()            # 调用实例方法；输出"Hello, Fred"
g.greet(loud=True)   # 调用实例方法；输出"HELLO, FRED!"
```

源代码