1. Python

# -\*- coding: UTF-8 -\*-

1. lambd表达式：
2. sum = lambda a,b:a\*b  
   print(sum(4,5))
3. 装饰器

这并不是我们想要的！Ouput输出应该是"a\_function\_requiring\_decoration"。这里的函数被warpTheFunction替代了。它重写了我们函数的名字和注释文档(docstring)。幸运的是Python提供给我们一个简单的函数来解决这个问题，那就是functools.wraps。我们修改上一个例子来使用functools.wraps：

from functools import wraps

def a\_new\_decorator(a\_func):

@wraps(a\_func)

def wrapTheFunction():

print("I am doing some boring work before executing a\_func()")

a\_func()

print("I am doing some boring work after executing a\_func()")

return wrapTheFunction

@a\_new\_decorator

def a\_function\_requiring\_decoration():

"""Hey yo! Decorate me!"""

print("I am the function which needs some decoration to "

"remove my foul smell")

print(a\_function\_requiring\_decoration.\_\_name\_\_)

# Output: a\_function\_requiring\_decoration

**@wraps**接受一个函数来进行装饰，并加入了复制函数名称、注释文档、参数列表等等的功能。这可以让我们在装饰器里面访问在装饰之前的函数的属性。

现在好多了。我们接下来学习装饰器的一些常用场景。

蓝本规范:

from functools import wraps

def decorator\_name(f):

@wraps(f)

def decorated(\*args, \*\*kwargs):

if not can\_run:

return "Function will not run"

return f(\*args, \*\*kwargs)

return decorated

@decorator\_name

def func():

return("Function is running")

can\_run = True

print(func())

# Output: Function is running

can\_run = False

print(func())

# Output: Function will not run

1. 迭代器：

迭代器：本文主要讲解python中的迭代器Iterator。

**什么是迭代器**

迭代器是访问可迭代对象的工具

迭代器是指用iter(obj)函数返回的对象(实例)

迭代器是指用next(it)函数获取可迭代对象的数据

**迭代器函数(iter和next)**

iter(iterable)从可迭代对象中返回一个迭代器,iterable必须是能提供一个迭代器的对象

next(iterator) 从迭代器iterator中获取下一了记录,如果无法获取下一条记录,则触发stoptrerator异常

说明:

1.迭代器只能往前取值,不会后退

2.用iter函数可以返回一个可迭代对象的迭代器

示例

l = [1,3,5,7]

   it = iter(1) #让l提供一个能访问自己的迭代器

   next(it) #1  从迭代器中取值,让迭代器去获取l中的一个元素

   next(it) #3

   next(it) #5

   next(it) #7

   next(it) # StopIterable 异常

迭代器的用途:

用迭代器可以依次访问可迭代对象的数据

示例:

#以下用迭代器来访问l列表中的元素

  l = [2,3,5,7]

  it = iter(l)

  while True:

      try:

          x = next(it) #获取一个数据并绑定到x

          print(x)  #2,3,5,7

      except StopIteration:

      pass

练习

有一个集合:

   s = {'唐僧','悟空','悟能','悟净'}

用for语句来遍历所有的元素如下;

    for x in s:

            print(x)

  else:

            print('遍历结束')

将上面的for语句改写为while语句和迭代器实现

s = {'唐僧','悟空','悟能','悟净'}

it = iter(s)

try:

while True:

x = next(it)

print(x)

except StopIteration:

print('遍历结束')

1. 文件读写：

基础：文件I/O;

打印到屏幕print()

读取键盘输入input()

打开文件 fo = open(“myfile.txt”,”w”); 文件名：fo.name();关闭：fo.close()

写入到文件中 fo.write(“zhanghui\nzhanghui01”)

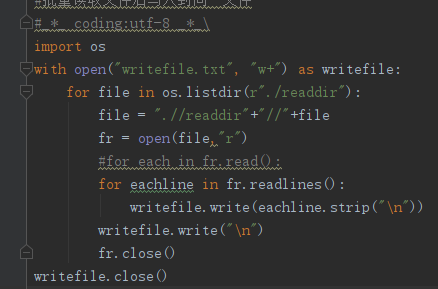
读取文件内容 str=fo.read(10) #参数是读取的字节数，没有传参尽可能多的读

文件定位 返回文件当前位置fo.tell()； 改变文件的当前位置fo.seek(offset[, from]) from=0(开头),1(当前),2(末尾)

重命名：os.rename(“oldname.txt”,” newname.txt”); 删除文件：os.remove(filename)

创建目录：os.madir(“filename.txt”);改变目录：os.chdir(“/home/newdir”);显示当前工作目录：os.getcwd（）

批量操作-os模块

批量读取文件： 

with方法帮我们实现了finally中f.close

1. 闭包：在一个外函数中定义了一个内函数，内函数中用到了外函数的临时变量，且外函数的返回值是内函数的引用。

def outer(a):

b=10

def inner():

print(a+b)

return inner

一般情况下，在我们认知当中，如果一个函数结束，函数的内部所有东西都会释放掉，还给内存，局部变量都会消失。但是闭包是一种特殊情况，如果外函数在结束的时候发现有自己的临时变量将来会在内部函数中用到，就把这个临时变量绑定给了内部函数，然后自己再结束。

闭包内函数修改外函数的局部变量方法：1.声明为nonlocal变量；2.使用可变类型数据变量

1. map（）函数：

第一个参数是fun，第二个参数是一般是list，第三个参数可以写list，也可以不写，根据需求；

1. 不可变数据类型：数值型、字符串型string和元组tuple

不允许变量的值发生变化，如果改变了变量的值，相当于是新建了一个对象，而对于相同的值的对象，在内存中则只有一个对象（一个地址），如下图用id()方法可以打印对象的id

可变数据类型：列表list和字典dict；

允许变量的值发生变化，即如果对变量进行append、+=等这种操作后，只是改变了变量的值，而不会新建一个对象，变量引用的对象的地址也不会变化，不过对于相同的值的不同对象，在内存中则会存在不同的对象，即每个对象都有自己的地址，相当于内存中对于同值的对象保存了多份，

1. sort 与 sorted 区别：

sort 是应用在 list 上的方法，sorted 可以对所有可迭代的对象进行排序操作。

list 的 sort 方法返回的是对已经存在的列表进行操作，无返回值，而内建函数 sorted 方法返回的是一个新的 list，而不是在原来的基础上进行的操作。

sorted 语法：sorted(iterable, cmp=None, key=None, reverse=False)

students = [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]

sorted(students, key=lambda s: s[2], reverse=True) # 按降序

[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]

1. 列表推导式：list=[ I for I in [1,2,3,4] if I > 2]
2. 随机整数：random.randint(a,b),生成区间内的整数

随机小数：习惯用numpy库，利用np.random.randn(5)生成5个随机小数

0-1随机小数：random.random(),括号中不能传参

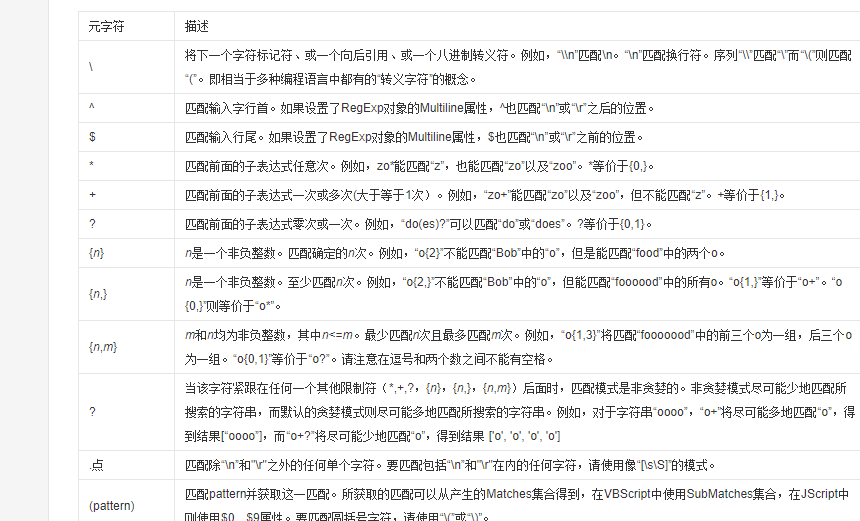
1. re库，正则匹配<https://www.cnblogs.com/lisa2016/p/11246191.html> 匹配6~10位由数字和26个英文字母混合而成的密码：  
   ^(?![0-9]+$)(?![a-zA-Z]+$)[0-9A-Za-z]{6,10}$

正则re.complie作用-re.compile是将正则表达式编译成一个对象

1. re.match( )方法匹配的是以xxx开头的字符串，若不是开头的，尽管属于str内，则无法匹配。
2. ?,\*,+,\d,\w 都是等价字符  
   　　?等价于匹配长度{0,1}  
   　　\*等价于匹配长度{0,}  
   　　+等价于匹配长度{1,}  
   　　\d等价于[0-9]

. 匹配任意字符

1. \D等价于[^0-9]  
   　　\w等价于[A-Za-z\_0-9]
2. \W等价于[^A-Za-z\_0-9]。
3. **常用运算符与表达式：**^ 开始  
   　　（） 域段  
   　　[] 包含,默认是一个字符长度  
   　　[^] 不包含,默认是一个字符长度  
   　　{*n*,*m*} 匹配长度  
   　　. 任何单个字符(\. 字符点)  
   　　| 或  
   　　\ 转义  
   　　$ 结尾  
   　　[A-Z] 26个大写字母  
   　　[a-z] 26个小写字母  
   　　[0-9] 0至9数字
4. [A-Za-z0-9] 26个大写字母、26个小写字母和0至9数字  
   　　， 分割  
   　　.  
   　　  
   　　**分割语法：**  
   　　[A,H,T,W] 包含A或H或T或W字母  
   　　[a,h,t,w] 包含a或h或t或w字母  
   　　[0,3,6,8] 包含0或3或6或8数字
5. **语法与释义：**基础语法 "^([]{})([]{})([]{})$"  
   　　正则字符串 = "开始（[包含内容]{长度}）（[包含内容]{长度}）（[包含内容]{长度}）结束"  
   　　  
   　　?,\*,+,\d,\w 这些都是简写的,完全可以用[]和{}代替，在(?:)(?=)(?!)(?<=)(?<!)(?i)(\*?)(+?)这种特殊组合情况下除外。  
   　　初学者可以忽略?,\*,+,\d,\w一些简写标示符，学会了基础使用再按表自己去等价替换





1. 功能测试：
   1. 互联网产品作为服务大众的多媒体工具，更关注的是用户体验。在设计测试用例的时候，除了功能测试相关的用例，更要关注异常情景以及用户体验相关的测试用例。

经典设计case：水杯

电梯

窗户

自动售货机

滴滴打车叫单

微信、微博聊天list分组

1. 计算机网络：这类问题在应聘服务端测试的时候居多。例如TCP/HTTP原理，HTTPS与HTTP的区别，IP地址的分类，TCP与UDP的区别以及常见状态码等；

一个http请求全过程：<https://www.cnblogs.com/guobm/p/9739704.html>

1. 数据库：数据库范式，简单的命令，主键与外健，内联结等；

select distinct name from student

1. 编程题以及情景设计题：

基础类：

常见排序算法

算法复杂度

树的遍历，递归非递归，深度优先和广度优先

链表题考到的最多：链表反转，环相关问题，删除中间元素等

等；

字符串处理：

字符串反转

文本的规范化（首字母大写，单词规范化）

关键字查找及过滤

等；

情景设计题

这类的题目会跟前面提到的很多问题相结合，考察知识的全面性，这里就直接列出来我遇到的问题吧，你们可以认真感受下每个问题背后潜藏的考察点：

代码实现，判断IP的合法性

输入到浏览器一个地址，到观察到页面这个过程中都发生了什么

这道题目很容易漏掉Cookie，也很容易漏掉该页面需要权限或者登陆的情况

如何测试一个IM系统

如何测试一个多线程安全的日志类库

对一个SDK网络协议接口进行升级，写出所有的测试用例

1. Linux:
2. 10个Linux常用命令

ls pwd cd touch rm mkdir tree cp mv cat more grep echo