

Python复习和回顾

考试题型

- 选择题- (1.5*20道) 30
- 填空题- (1.5*8个空) 12
- 判断题- (1*9道) 9
- 程序推断 (看代码, 推断写出代码运行后的结果或者对给出的代码进行修改-5*5道) 25
- 程序书写 (根据题目要求, 写出相应的程序) 8*3道-24

考试章节和内容

- Python的发展历史（python2和python3的主要不同，python没有考虑向下兼容，即python3不能不加修改地去运行一个用python2语法编写的程序）
- Python的主要优点（语法简洁 跨平台 胶水语言 开源 强制可读 支持中文 支持面向对象和面向过程 类的库丰富）
- Python执行代码的两种方式：文件式和交互式
- Python的书写规范（变量 标识符 保留字 内置函数 缩进）
- Python的库（标准库和第三方库）
- Python的数据类型
- 循环结构

Python2和python3

- 如何成功打印一个字符串语句（print的使用方法） - 选择填空题

`print(" ")`

numpy和科学计算（选择 填空）

- numpy是第三方库，需要安装（有别于random time datetime turtle等标准库）
- 相应的方法：统计元素个数-`np.size()`，元素包含的字节长度-`np.itemsize()`，创建全为0的矩阵`zeros`，全为1的矩阵`ones`，单位矩阵-`identity`，更改维度-`reshape`，可以实现转置的操作（`array.T` `np.transpose(array)` `array.swapaxes(0,1)`），可以实现平铺的操作（`array.flatten()` 和 `array.ravel()`，它们的区别是前者修改之后不会改变原有矩阵的数据）

按轴进行运算

```
1 c = np.arange(1,13).reshape([3,4])
2 # [[ 1  2  3  4]
3 #   [ 5  6  7  8]
4 #   [ 9 10 11 12]]
5 print(np.sum(c, axis=0))
6 # [15 18 21 24]
7 print(np.sum(c, axis=1))
8 # [10 26 42]
```



```
c = np.arange(1,25).reshape([2,3,4])
print(c)
# [[[ 1  2  3  4]
#    [ 5  6  7  8]
#    [ 9 10 11 12]]
#
#    [[13 14 15 16]
#     [17 18 19 20]
#     [21 22 23 24]]]
print(np.sum(c, axis=0))
# [[14 16 18 20]
#   [22 24 26 28]
#   [30 32 34 36]]
print(np.sum(c, axis=1))
# [[15 18 21 24]
#   [51 54 57 60]]
print(np.sum(c, axis=2))
# [[10 26 42]
#   [58 74 90]]
```

numpy的广播功能

```
>>> a = np.array([1,2,3])
>>> b =
np.array([[4,5,6],[7,8,9]])
>>> a + b
array([[5, 7, 9],
       [8, 10, 12]])
```



文件的读写

- 文件的分类（二进制文件和文本文件）;打开文件的两个参数（mode 文件名），打开文件相关的保留字：（with as），打开文件的读取方法包括：readall();read();readline();readlines(); 假设文件对象为fp，则fp.closed表示文件的打开状态，fp.name表示被打开的文件的名称。readlines会得到一个包含换行符的一个所有行的列表，writelines会将一个列表写入文件中。Seek后边可以接offset，其中0表示文件开头，1表示当前位置，2表示文件结尾。
- Python中使用文件对象的tell()方法返回文件的当前读写位置。
- 假如要读取所有的文本，可以使用read()，而括号中不写数字；readline可以读取一行，readlines则读取当前位置到文件末尾的所有行，并将这些行以列表的形式返回。

典型实例

实例1

```
1 fname = input("请输入要写入的文件: ")
2 fo = open(fname, "w+")
3 ls = ["唐诗", "宋词", "元曲"]
4 fo.writelines(ls)
5 for line in fo:
6     print(line)
7 fo.close()
```

程序执行结果如下:

```
>>>请输入要写入的文件: e:\liushuo\唐诗宋词.txt
>>>
```

实例2

```
1 fname = input("请输入要写入的文件: ")
2 fo = open(fname, "w+")
3 ls = ["唐诗", "宋词", "元曲"]
4 fo.writelines(ls)
5 fo.seek(0)
6 for line in fo:
7     print(line)
fo.close()
```

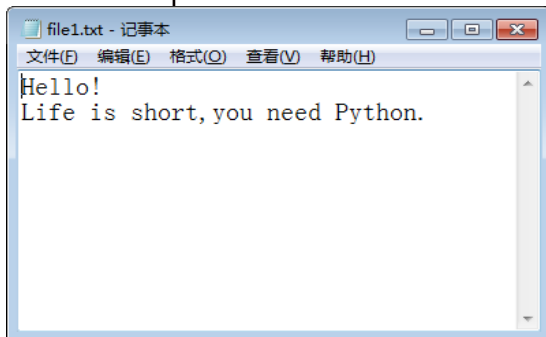
```
-----
请输入要写入的文件: e:\liushuo\唐诗宋词.txt
唐诗宋词元曲
```

典型实例2文件的基本操作

例 请输出文本文件file1.txt中的纯文本，并统计文本文件中大写
字母出现的次数。

分析：读取文件的所有行，以列表返回，然后遍历列表，统计大写字母的个数。

```
1 fp=open("file1.txt","r")
2 ls=fp.readlines()      #读取文件的所有行，以列表返回
3 c=0
4 for x in ls:           #遍历列表ls
5     print(x.strip())    #输出文件的每一个行（不包括'\n'）
6     for s in x:         #遍历每个列表元素
7         if s.isupper():
8             c=c+1
9 print("文件中大写字母有{}个.".format(c))
10 fp.close()
```



运行程序，结果如下：
Hello!
Life is short,you need Python.
文件中大写字母有3个。

例 输入一段字符串，转换成列表，并统计输入的一段字符串中大写字母出现的次数。

```
1 str1=input("请输入一段字符串:")
2 ls=list(str1)          #读取文件的所有行，以列表返回
3 c=0
4 for x in ls:           #遍历列表ls
5     if x.isupper():
6         c=c+1
7 print("字符串中大写字母有{}个.".format(c))
```

递归和循环

实例1- 求阶乘(假设参数为5)

循环

```
def fact(n) :
```

```
    s=1
```

```
    for i in range(1, n+1):
```

```
        s*= i
```

```
    return s
```

```
fact(5)
```

递归

```
1 def fact(n):
2     if n == 0:
3         return 1
4     else:
5         return n * fact(n-1)
6 num = eval(input("请输入一个整数: "))
7 print(fact(abs(int(num))))
```

递归和循环

实例2-**循环**-最大公约数计算。从键盘输入两个整数，编写程序求这两个整数的最大公约数和最小公倍数。（求最大公约数使用辗转相除法，求最小公倍数用两数的积除以最大公约数）

- `a=int(input("请输入数字: "))`
- `b=int(input("请输入数字: "))`
- `# 先给两数排序，保证大数除以小数`
- `m=max(a,b)`
- `n=min(a,b)`
- `t=m%n`
- `while t!=0:`
 - `m,n=n,t # 每个除式的m、n都是上一个式子的n和余数`
 - `t=m%n # 更新余数`
- `print("{}和{}的最大公约数为{},最小公倍数为{}".format(a,b,n,a*b/n))`

递归和循环

实例2-递归-最大公约数计算。

- `def Gongyueshu(a,b):`
- `# 比大小，保证大数除以小数`
- `if a<b:`
- `a,b=b,a`
- `# 判断是否能整除，若能整除，直接返回被除数`
- `if a%b==0:`
- `return b`
- `# 若不能整除，则返回函数gongyueshu，参数做相应变化`
- `else:`
- `return gongyueshu (b,a%b)`
- `a=int(input("请输入第一个数字： "))`
- `b=int(input("请输入第二个： "))`
- `gcd=Gongyueshu (a,b)`
- `print(f"{a}和{b}的最大公约数为{gcd}")`

递归和循环

实例3-循环-斐波那契数列（设天数为10）

```
def fib(n):  
    a, b = 0, 1  
    while a < n:  
        print(a, end=' ')  
        a, b = b, a+b  
    print()  
  
fib(10)
```

递归

```
def f(n):  
    if n == 1 or n == 2 :  
        return 1  
    else :  
        return f(n-1) + f(n-2)  
  
print(f(10))
```

python库的安装（选择填空判断）

使用pip工具 pip install 库， pip可以实现安装扩展包 可以卸载扩展包 可以更新扩展包。

turtle标准库

库的几种导入方法（选择填空） **from <库名> import <函数名> ;from <库名> import *;import turtle;import numpy as np**

–turtle相应的方法：**goto(),penup(), pendown(),pensize(), pencolor(), turtle.fd(), turtle.circle(), turtle.setheading(angle)-turtle.seth(angle)(angle: 行进方向的绝对角度), turtle.left(angle) turtle.right(angle)(同样是绝对角度)**

**代码相关部分（使用turtle相应的方法来绘制常见的图形，如多边形，熟悉如何常规绘制和用for循环来完成）
使用turtle库中的turtle.fd()函数和turtle.seth()函数绘制一个等边三角形，边长为100：**

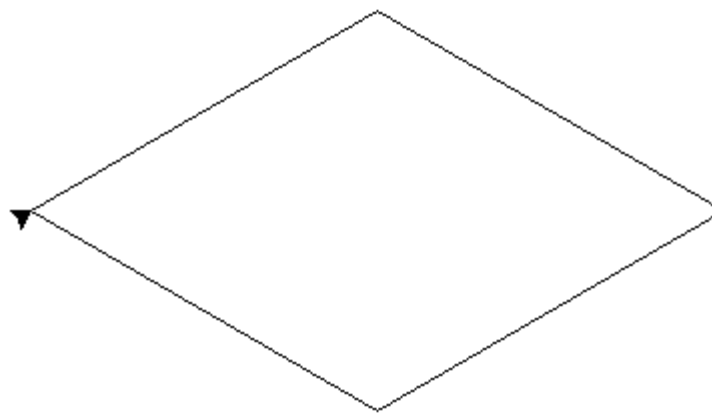
```
from turtle import *  
fd(100)  
left(120)  
fd(100)  
left(120)  
fd(100)
```

用for循环来完成绘制
import turtle as t
for i in range(3):
 t.seth(i* 120)
 t.fd(100)

如何去绘制正多边形？（使用for循环）

如何去绘制一个长度为200的菱形？（使用for循环）

```
import turtle as t
t.right(-30)      #改变出发角度
for i in range(2): #在此循环中,i取值为0和1
    t.fd(200)
    t.right(60*(i+1))
for i in range(2):
    t.fd(200)
    t.right(60*(i+1))
```



IPO描述（选择填空判断）

IPO是指 input process output
接收华氏温度和摄氏温度并相互转换的IPO描述是什么？
Input-输入： 带华氏或摄氏标志的温度值
Process-处理： 根据温度标志选择适当的温度转换算法
Output-输出： 带有摄氏或华氏标志的温度值

Python的缩进和注释（选择填空判断）
严格缩进 可以通过什么样的方法来引入注释

Python的命名规则（区分大小写 数字不能出现在首字符 中间不能出现空格）

1. 在Python中，以下标识符合法的是

A. _	B. 3C	C. it's	B. str
------	-------	---------	--------

Python的保留字
典型的保留字的名称和用法。

False ↵	class ↵	from ↵	or ↵
None ↵	continue ↵	global ↵	pass ↵
True ↵	def ↵	if ↵	raise ↵
and ↵	del ↵	import ↵	return ↵
as ↵	elif ↵	in ↵	try ↵
assert ↵	else ↵	is ↵	while ↵
async ↵	except ↵	lambda ↵	with ↵
await ↵	finally ↵	nonlocal ↵	yield ↵
break ↵	for ↵	not ↵	↵

Input eval 格式化方法（选择填空判断 代码题目）

input获得的是字符串形式，
eval可以执行语句，把返回结果输出。
format格式化，

```
>>>eval("1")    1
>>>eval("1+2")   3
>>>eval('print("Hello")')
Hello
```

"{ } : 计算机{ } 的CPU占用率为{ } %".format("2018-10-10", "C", 10)

0 1 2 0 1 2

字符串中槽{}的默认顺序 format()中参数的顺序

"{1} : 计算机{0} 的CPU占用率为{2} %".format("2018-10-10", "C", 10)

接收用户输入的一个小于 20的正整数，在屏幕上逐行递增显示从 001 到该正整数，数字显示的宽度为 3，不足位置补 0，后面追加一个空格，然后显示‘*’号，‘*’号的个数等于行首数字。

```
x=int(input("请输入一个小于20的正整数: "))
for i in range(1,x+1):
    print("{:0>3}".format(i),"*"*(i))
```

```
请输入一个小于20的正整数: 12
001 *
002 **
003 ***
004 ****
005 *****
006 ****
007 *****
008 *****
009 *****
010 *****
011 *****
012 *****
```

```
>>> "{0:=^20}".format("PYTHON")
'=====PYTHON====='

>>> "{0:*>20}".format("BIT")
'*****BIT'

>>> "{:10}".format("BIT")
'BIT      '
```

:	填充	对齐	<宽度>
引导 符号	用于填 充的单 个字符	< 左对齐	槽的设定输 出宽度
		> 右对齐	
		^ 居中对齐	

python输出菱形图案

给定一个菱形的行数，打印出相应的效果，菱形中间行星星的个数是Python控制台输入的数字，且没有空格

```
x=int(input('请输入菱形宽度: '))
for i in range(0,x+1): #外层循环控制上半个三角形共有几行，也就是菱形的宽
    for j in range(0,x-i): #内层循环控制每一行中的空格数目
        print(' ',end=") #end="即语句没结束不换行的意思
    print(i * '*' ) #根据i的值显示出对应行的星号的数目
for i in range(1,x): #下半个三角形比上半个三角形少一行，所以i从1开始
    for j in range(0,i): #与上半个三角形的语句相似，即空格的数目
        print(' ',end=") #没结束不换行
    print((x-i)* '*' )
```

请输入菱形宽度：7

```
      *
     **
    ***
   ****
  *****
 *****
  *****
   ****
    ***
     **
      *
```

基本的数据类型

Python语言中，一切皆为对象，而每个对象都属于某个数据类型
Python的数据类型包括内置的数据类型、模块中定义的数据类型和用户自定义的类型
数值数据类型：int、bool、float、complex （int float complex这几种数据类型转换的规则）
序列数据类型：不可变（str、tuple）和可变（list）
集合数据类型：set
字典数据类型：dict。例如：{1: "one", 2: "two"}

pow()-求幂次的函数

pow(x, y)函数：计算 x^y

```
>>> a=123
>>> type(a)
<class 'int'>
```

#输出：<class 'int'>

- bin(): 转换为2进制;
- oct(): 转换为8进制;
- hex(): 转换为16进制;
- int(): 转换为10进制;

数值运算操作符

x + y	+ x	x += y	x -= y	x *= y	x /= y	abs(x)
x - y	- y					
x * y	x % y	x //= y	x %= y	x **= y		divmod(x,y)
x / y						
x // y	x ** y					pow(x, y[, z])

基本的数据类型

数据运算之间的优先级问题（是前结合，还是后结合）-选择填空判断

如：以下运算的输出结果为：

bool数据类型和相关运算符-选择填空判断

>>> not 0	#输出: True
>>> 1 or 2	#输出: 1
>>> 0 or 2	#输出: 2
>>> False or True	#输出: True
>>> 1 and 2	#输出: 2
>>> 0 and 2	#输出: 0

循环与分支结构

```
+ - - - - + - - - - +
*           *           *
*           *           *
*           *           *
*           *           *
*           *           *
+ - - - - + - - - - +
*           *           *
*           *           *
*           *           *
*           *           *
+ - - - - + - - - - +
```






```
for i in range(11):
    if i in [0,5,10]:
        print("+ - - - - + - - - - +")
    else:
        print("*           *           *")
```

编写程序，计算 $S_n=1-3+5-7+9-11+\dots$

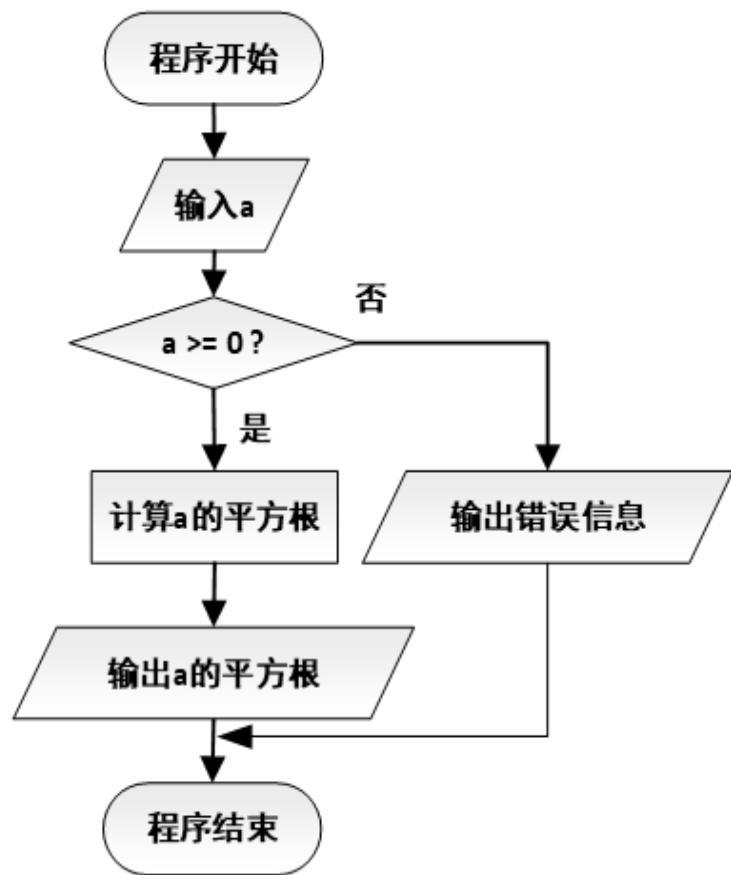
```
n=int(input("请输入整数n: "))
s=0
for i in range(1,n+1):
    if i%2==0:
        s-=(2*i-1)
    else:
        s+=(2*i-1)
print(s)
```

程序流程图-选择填空判断

- 又称为程序框图，是描述程序运行具体步骤的图形表示
- 流程图的基本元素主要包括以下几种

1. 开始框和结束框()。表示程序的开始和结束
2. 输入/输出框()。表示输入和输出数据。
3. 处理框()。表示要执行的流程或处理。
4. 判断框()。表示条件判断，根据判断的结果执行不同的分支。
5. 箭头线()。表示程序或算法的走向。

【例】 使用程序流程图（如图所示）描述计算所输入数据a的平方根的程序



循环与分支结构

单分支结构示例：输入两个数a和b，比较两者大小，使得a大于b

```
a = int(input("请输入第1个整数: "))
b = int(input("请输入第2个整数: "))
print("输入值: {0}, {1} ".format(a, b))
if (a < b):      #a和b交换
    t = a
    a = b
    b = t
print("降序值: {0}, {1} ".format(a, b))
```

- if语句单分支结构的语法形式：
 - If (条件表达式) : ...语句/语句块
- 当条件表达式的值为真 (True) 时，执行if后的语句 (块) ， 否则不做任何操作，控制将转到if语句的结束点

双分支结构

if语句双分支结构的语法形式

当条件表达式的值为真 (True) 时，执行if后的语句 (块) 1， 否则执行else后的语句 (块) 2

- If (条件表达式) :
 - ...语句/语句块1
- else:
 - ...语句/语句块2

条件表达式

- 条件为真时的值 • if • (条件表达式) • else • 条件为假时的值。

双分支结构示例：输入一个数a，如果a大于0，则在屏幕上显示“正数”， 否则在屏幕上显示“不是正数”

```
a=input("请输入一个数: ")
a=int(a)
if a>0:
    print(a,"is positive. ")
else:
    print(a,"is not positive. ")
```

多分支结构

if语句多分支结构的语法形式

- if (条件表达式1) :
 ...语句/语句块1
- elif (条件表达式2) :
 ...语句/语句块2
 ...
- elif (条件表达式n) :
 ...语句/语句块n
- [else:
 ...语句/语句块n+1

```
mark = int(input("请输入分数: "))  
if (mark >= 90): grade = "优"  
elif (mark >= 80): grade = "良"  
elif (mark >= 70): grade = "中"  
elif (mark >= 60): grade = "及格"  
else: grade = "不及格"
```

for 循环 while 循环 break continue (选择填空判断代码)

使用continue语句跳过循环。要求输入若干学生成绩（按Q或q结束），如果成绩<0，则重新输入。统计学生人数和平均成绩

```
num = 0; scores = 0; #初始化学生人数和成绩和  
while True:  
    s = input('请输入学生成绩（按Q或q结束）：')  
    if s.upper() == 'Q':  
        break  
    if float(s) < 0: #成绩必须>=0  
        continue  
    num += 1    #统计学生人数  
    scores += float(s) #计算成绩之和
```

for 循环 while循环 break continue (选择填空判断代码)

请重点关注range对象相关的点

#计算 1到100的整数和

```
result = 0 #保存累加结果的变量
```

```
for i in range(101):
```

```
    result =result+i
```

```
    #上述语句逐个获取从1到100这些值, 并做累加操作
```

```
print("1到100的整数和为:", result)
```

运行程序, 结果如下:

1到100的整数和为: 5050

```
i = 1          #设置循环变量i初值
sum=0          #设置整数和sum初值
# 设置循环的初始化条件
while i <= 100: # 当i小于100时, 执行循环体
    print( i)
    sum=sum+i
    i += 1      #改变循环变量i的值
print("1~100 的所有整数和: ",sum)
```

运行程序, 结果如下:

1

2

.....

99

100

1~100 的所有整数和: 5050

使用try...except...else...finally语句捕获处理异常-一 般为选择填空判断，也有少量的可能出代码题目

```
try: ↵
    ... 可能产生异常的语句 ↵
except Exception1: ... #捕获异常 Exception1 ↵
    ... 发生异常时执行的语句 ↵
except (Exception2, Exception3): ... #捕获异常 Exception2、Exception3
    ... 发生异常时执行的语句 ↵
except Exception4 as e: ... #捕获异常 Exception4，其实例为 e
    ... 发生异常时执行的语句 ↵
except: ... #捕获其它所有异常 ↵
    ... 发生异常时执行的语句 ↵
else: ... #无异常 ↵
    ... 无异常时执行的语句 ↵
finally: ... #不管发生异常与否，保证执行 ↵
    ... 不管发生异常与否，保证执行的语句 ↵
```

Python内置的序列数据类型

- **元组**也称之为定值表，用于存储值固定不变的值表。例如：

```
>>> s1=(1,2,3)
>>> s1                #输出: (1, 2, 3)
>>> s1[2]             #输出: 3
```

- **列表**也称之为表，用于存储其值可变的表。例如：

```
>>> s2=[1,2,3]
>>> s2[2]=4
>>> s2                #输出: [1, 2, 4]
```

- **字符串**是包括若干字符的序列数据，支持序列数据的基本操作。例如：

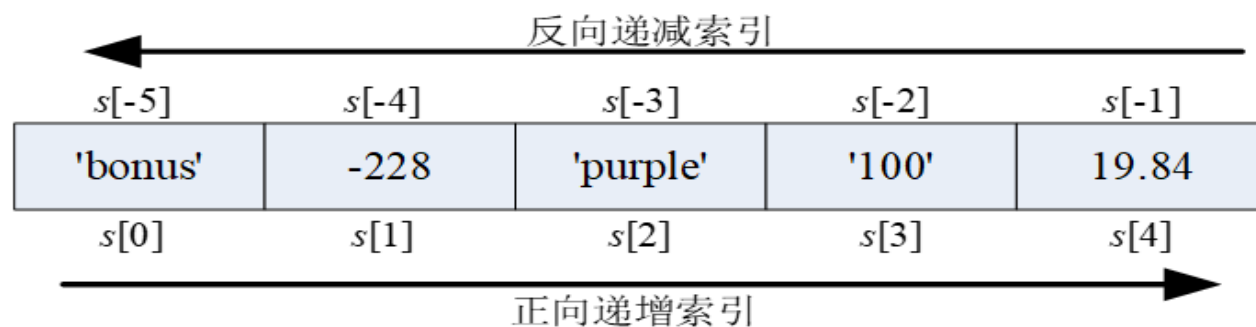
```
>>> s3="Hello, world!"
>>> s3[:5] #字符串前5个字符。输出: 'Hello'
```

len()、max()、min()，获取序列的长度、序列中元素最大值、序列中元素最小值
sum()获取列表或元组中各元素之和

```
>>> t1=(1,2,3,4)
>>> sum(t1) #输出: 10
10
>>> t2=(1,'a',2)
>>> sum(t2) #TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> s='1234'
>>> sum(s) #TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> sum([[1,2],[3,4]],[])
[1, 2, 3, 4]
```

```
>>> s='abcdefg'
>>> len(s)
7
>>> max(s)
'g'
>>> min(s)
'a'
>>> s2=""
>>> len(s2)
0
```

字符串 集合 列表 元组 字典



```
>>> s='abcdef'
```

```
>>> s[0]
```

```
'a'
```

```
>>> s[2]
```

```
'c'
```

```
>>> s[-1]
```

```
'f'
```

```
>>> s[-3]
```

```
'd'
```

```
>>> t=('a','e','i','o','u')
```

```
>>> t[0]
```

```
'a'
```

```
>>> t[1]
```

```
'e'
```

```
>>> t[-1]
```

```
'u'
```

```
>>> t[-5]
```

```
'a'
```

```
>>> lst=[1,2,3,4,5]
```

```
>>> lst[0]
```

```
1
```

```
>>> lst
```

```
[1, 2, 3, 4, 5]
```

```
>>> lst[2]='a'
```

```
>>> lst[-2]='b'
```

```
>>> lst
```

```
[1, 2, 'a', 'b', 5]
```

序列的切片操作

`s[i:j]` 或者 `s[i:j:k]`

- 【例】序列的切片操作示例

```
>>> s='abcdef'
```

```
>>> s[1:3]
```

```
'bc'
```

```
>>> s[3:10]
```

```
'def'
```

```
>>> s[8:2]
```

```
''
```

```
>>> s[:]
```

```
'abcdef'
```

```
>>> s[:2]
```

```
'ab'
```

```
>>> s[::-2]
```

```
'ace'
```

```
>>> s[::-1]
```

```
'fedcba'
```

```
>>> t=('a','e','i','o','u')
```

```
>>> t[-2:-1]
```

```
('o',)
```

```
>>> t[-2:]
```

```
('o','u')
```

```
>>> t[-99:-5]
```

```
()
```

```
>>> t[-99:-3]
```

```
('a','e')
```

```
>>> t[::]
```

```
('a','e','i','o','u')
```

```
>>> t[1:-1]
```

```
('e','i','o')
```

```
>>> t[1::2]
```

```
('e','o')
```

```
>>> lst=[1,2,3,4,5]
```

```
>>> lst[:2]
```

```
[1, 2]
```

```
>>> lst[:1]=[]
```

```
>>> lst
```

```
[2, 3, 4, 5]
```

```
>>> lst[:2]
```

```
[2, 3]
```

```
>>> lst[:2]='a'
```

```
>>> lst[1:]='b'
```

```
>>> lst
```

```
['a', 'b']
```

```
>>> del lst[:1]
```

```
>>> lst
```

```
['b']
```


序列的比较运算操作

- **【例】序列的比较运算示例**

```
>>> s1='abc'  
>>> s2='abc'  
>>> s3='abcd'  
>>> s4='cba'  
>>> s1 > s4
```

False

```
>>> s2 <= s3
```

True

```
>>> s1 == s2
```

True

```
>>> s1 != s3
```

True

```
>>> 'a' > 'A'
```

True

```
>>> 'a' >= ''
```

True

```
>>> t1=(1,2)  
>>> t2=(1,2)  
>>> t3=(1,2,3)  
>>> t4=(2,1)  
>>> t1<t4
```

True

```
>>> t1 <= t2
```

True

```
>>> t1 == t3
```

False

```
>>> t1 != t2
```

False

```
>>> t1 >= t3
```

False

```
>>> t4 > t3
```

True

```
>>> s1=['a','b']  
>>> s2=['a','b']  
>>> s3=['a','b','c']  
>>> s4=['c','b','a']  
>>> s1<s2
```

False

```
>>> s1<=s2
```

True

```
>>> s1==s2
```

True

```
>>> s1!=s3
```

True

```
>>> s1>=s3
```

False

```
>>> s4>s3
```

True

序列的排序操作

`sorted(iterable, key=None, reverse=False) ...` #返回系列的排序列表

- **【例】** 序列的排序操作示例

```
>>> s1='axd'
```

```
>>> sorted(s1)
```

```
['a', 'd', 'x']
```

```
>>> s2=(1,4,2)
```

```
>>> sorted(s2)
```

```
[1, 2, 4]
```

```
>>> sorted(s2,reverse=True)
```

```
[4, 2, 1]
```

```
>>> s3='abAC'
```

```
>>> sorted(s3, key=str.lower)
```

```
['a', 'A', 'b', 'C']
```

列表解析表达式

- `[expr for i1 in 序列 1... for iN in 序列 N] ...` #迭代序列里所有内容，并计算生成列表
- `[expr for i1 in 序列 1... for iN in 序列 N if cond expr]` #按条件迭代，并计算生成列表

【例】列表解析表达式示例。

```
>>> [i**2 for i in range(10)]    #平方值
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
>>> [(i,i**2) for i in range(10)] #序号, 平方值
```

```
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81)]
```

```
>>> [i for i in range(10) if i%2==0] #取偶数
```

```
[0, 2, 4, 6, 8]
```

```
>>> [(x, y, x*y) for x in range(1, 4) for y in range(1, 4) if x>=y] #二重循环
```

```
[(1, 1, 1), (2, 1, 2), (2, 2, 4), (3, 1, 3), (3, 2, 6), (3, 3, 9)]
```

列表的序列操作

- 索引访问、切片操作、连接操作、重复操作、成员关系操作、比较运算操作，以及求列表长度、最大值、最小值等
- 【例】列表的序列操作示例

表示删除位置为2的这个元素

```
>>> s=[1,2,3,4,5,6]
>>> s[1]='a'
>>> s
[1, 'a', 3, 4, 5, 6]
>>> s[2]=[]
>>> s
```

```
[1, 'a', [], 4, 5, 6]
>>> del s[3]
>>> s
[1, 'a', [], 5, 6]
>>> s[:2]
[1, 'a']
```

```
>>> s[2:3]=[]
>>> s
[1, 'a', 5, 6]
>>> s[:1]=[]
>>> s
['a', 5, 6]
```

```
>>> s[:2]='b'
>>> s
['b', 6]
>>> del s[:1]
>>> s
[6]
```

list对象的方法-选择填空判断

假设表中的示例基于s=[1,3,2]

方法	说明	示例
s.append(x)	把对象x追加到列表s尾部	s.append('a') #s= [1, 3, 2, 'a'] s.append([1,2]) #s= [1, 3, 2, 'a', [1, 2]]
s.clear()	删除所有元素。相当于del s[:]	s.clear() #s= []
s.copy()	拷贝列表	s1=s.copy() #s1= s=[1,3,2] id(s),id(s1) #(3143376592008, 3143376591496)
s.extend(t)	把序列t附加到s尾部	s.extend([4]) #s= [1, 3, 2, 4] s.extend('ab') #s= [1, 3, 2, 4, 'a', 'b']
s.insert(i, x)	在下标i位置插入对象x	s.insert(1,4) #s= [1, 4, 3, 2] s.insert(8,5) #s= [1, 4, 3, 2, 5]
s.pop([i])	返回并移除下标i位置对象，省略i时为最后对象。若超出下标，将导致IndexError	s.pop() #输出2。s= [1, 3] s.pop(0) #输出1。s=[3]
s.remove(x)	移除列表中第一个出现的x。若对象不存在，将导致ValueError	s.remove(1) #s= [3, 2] s.remove('a') #ValueError: list.remove(x): x not in list
s.reverse()	列表反转	s.reverse() #s=[2, 3, 1]
s.sort()	列表排序	s.sort() #s=[1, 2, 3]

使用tuple对象创建元组实例对象

- **【例】** 使用tuple对象创建元组实例对象示例

```
>>> t1=tuple()
>>> t2=tuple("abc")
>>> t3=tuple([1,2,3])
>>> t4=tuple(range(3))
>>> print(t1,t2,t3,t4) #输出: () ('a', 'b', 'c') (1, 2, 3) (0, 1, 2)
```

字典的定义

{键 1:值 1,[键 2:值 2,..., 键 n:值 n]}

- `dict()` #创建一个空字典 ↵
- `dict(**kwargs)` #使用关键字参数，创建一个新的字典。此方法最紧凑
- `dict(mapping)` #从一个字典对象创建一个新的字典 ↵
- `dict(iterable)` #使用序列，创建一个新的字典 ↵

• 【例】创建字典对象示例

```
>>> {}
```

```
{}
```

```
>>> {'a':'apple','b':'boy'}
```

```
{'a': 'apple', 'b': 'boy'}
```

```
>>> dict()
```

```
{}
```

```
>>> dict({1:'food', 2:'drink'})
```

```
{1: 'food', 2: 'drink'}
```

```
>>> dict([('id','1001'),('name','Jenny')])
```

```
{'id': '1001', 'name': 'Jenny'}
```

```
>>> dict(baidu='baidu.com', google='google.com')
```

```
{'baidu': 'baidu.com', 'google': 'google.com'}
```

字典的访问操作

- `d[key]` ······ #返回键为 `key` 的 `value`; 如果 `key` 不存在, 则导致 `KeyError`
- `d[key]=value` ······ #设置 `d[key]` 的值为 `value`; 如果 `key` 不存在, 则添加键/值对
- `del d[key]` ······ #删除字典元素; 如果 `key` 不存在, 则导致 `KeyError`

- 【例】字典的访问示例

```
>>> d={1:'food', 2:'drink'}
```

```
>>> d
```

```
{1: 'food', 2: 'drink'}
```

```
>>> d[1]
```

```
'food'
```

```
>>> d[3]='fruit'
```

```
>>> d
```

```
{1: 'food', 2: 'drink', 3: 'fruit'}
```

```
>>> del d[2]
```

```
>>> d
```

```
{1: 'food', 3: 'fruit'}
```

```
>>> d[2]
```

```
Traceback (most recent call last):
```

```
File "<pyshell#100>", line 1, in <module>
```

```
    d[2]
```

```
KeyError: 2
```


字典的视图对象

- `d.keys()` ······ #返回字典 `d` 的键 `key` 的列表 ↵
- `d.values()` ······ #返回字典 `d` 的值 `value` 的列表 ↵
- `d.items()` ······ #返回字典 `d` 的 `(key, value)` 对的列表

- 【例5.34】字典的视图对象示例

```
>>> d={1:'food', 2:'drink', 3:'fruit'}
>>> d.keys()
dict_keys([1, 2, 3])
>>> for k in d.keys():
    print(k, end=" ")
1 2 3
```

```
>>> d.values()
dict_values(['food', 'drink', 'fruit'])
>>> for v in d.values():
    print(v, end=" ")
food drink fruit
```

```
>>> d.items()
dict_items([(1, 'food'), (2, 'drink'), (3, 'fruit')])
>>> for item in d.items():
    print(item, end=' ')
(1, 'food') (2, 'drink') (3, 'fruit')
```

字典对象的方法

- 假设表中的示例基于d={1: 'food', 2: 'drink', 3: 'fruit'}

方法	说明	示例
d.clear()	删除所有元素	>>> d.clear();d #结果: {}
d.copy()	浅拷贝字典	>>> d1=d.copy(); id(d), id(d1) (2487537820800, 2487537277976)
d.get(k)	返回键k对应的值, 如果key不存在, 返回None	>>> d.get(1),d.get(5) ('food', None)
d.get(k, v)	返回键k对应的值, 如果key不存在, 返回v	>>> d.get(1,'无'),d.get(5,'无') ('food', '无')
d.pop(k)	如果键k存在, 返回其值, 并删除该项目; 否则导致KeyError	>>> d.pop(1), d ('food', {2: 'drink', 3: 'fruit'})
d.pop(k, v)	如果键k存在, 返回其值, 并删除该项目; 否则返回v	>>> d.pop(5,'无'), d ('无', {1: 'food', 2: 'drink', 3: 'fruit'})
d.update([other])	使用字典或键值对, 更新或添加项目到字典d	>>> d1={1: '食物', 4: '书籍'} >>> d.update(d1);d {1: '食物', 2: 'drink', 3: 'fruit', 4: '书籍'}

字典实例

编写程序实现如下功能：找出任意字符串中只出现一次的字符，如果有多个这样的字符，请全部找出

- `s=input("请输入一个英文字符串：")`
- `dict1={}`
- `for ch in s:`
 - `dict1[ch]=dict1.get(ch,0) +1`
- `list1=[k for k,v in dict1.items() if v==1]`
- `print(list1)`

字典和字符串实例

有字符串 “k:1|k1:2|k2:3|k3:4”，请处理成字典
{'k':1,'k1':2 , 'k2':3 , 'k3':4}

- `str1 = "k:1|k1:2|k2:3|k3:4"`
- `dic = {}`
- `lst = str1.split("|")`
- `for l in lst:`
 - `lst2 = l.split(":")`
 - `dic[lst2[0]] = eval(lst2[1])`
- `print(dic)`

集合

- 集合数据类型是没有顺序的简单对象的聚集，且集合中元素不重复
- 可变集合对象（set）的定义
- {}表示空的dict，因为dict也使用花括号定义。空集为set()

集合案例

存在三个数据集合s1、s2、s3, s1 = {11,34,25,67,33}; s2={22,33,15,66,25}; s3={87,63,22,25,76}, 现在需要求得以下结果:

- 1.获取s1、s2和s3内容相同的元素列表
- 2.获取s1中有, s2中没有的元素列表
- 3.获取s1和s2中内容都不同的元素

- s1 = {11,34,25,67,33}
- s2 = {22,33,15,66,25}
- s3 = {87,63,22,25,76}
- #1.获取s1、s2中和s3内容相同的元素列表
- sres1 = (s1 & s3)|(s2 &s3) # sres1=(s1 | s2) & s3
- print(sres1)
- #2.获取s1中有, s2中没有的元素列表
- sres2 = s1 - s2
- print(sres2)
- #3.获取s1和s2中内容都不同的元素
- sres3 = s1 ^ s2
- print(sres3)

集合实例

存在三个数据集合s1、s2、s3, $s1 = \{11, 34, 25, 67, 33\}$;

$s2 = \{22, 33, 15, 66, 25\}$; $s3 = \{87, 63, 22, 25, 76\}$, 现在需要求得以下结果:

4. 获取s1、s2、s3中都包含的元素

5. 获取s1、s2、s3中, 只出现一次的元素

- $s1 = \{11, 34, 25, 67, 33\}$

- $s2 = \{22, 33, 15, 66, 25\}$

- $s3 = \{87, 63, 22, 25, 76\}$

- #4. 获取s1、s2、s3中都包含的元素

- $sres4 = s1 \& s2 \& s3$

- `print(sres4)`

- #5. 获取s1、s2、s3中, 只出现一次的元素

- $sres5 = (s1 \wedge s2 \wedge s3) - (s1 \& s2 \& s3)$

- `print(sres5)`

str数据类型（字符串）选择填空判断

- Python中没有独立的字符数据类型，字符即长度为1的字符串
- Python内置数据类型str，用于字符串处理
 - str对象的值为字符系列
 - str对象（字符串）是不可变对象
- 使用单引号或双引号括起来的内容，是字符串的字面量

Python字符串字面量-选择填空判断

- (1) 单引号 (' '). 包含在单引号中的字符串，其中可以包含双引号。
- (2) 双引号 (" "). 包含在双引号中的字符串，其中可以包含单引号
- (3) 三单引号 (''' '''). 包含在三单引号中的字符串，可以跨行。
- (4) 三双引号 (""" """). 包含在三双引号中的字符串，可以跨行。

```
>>> 'abc'                #输出: 'abc'
'abc'
>>> "Hello"              #输出: 'Hello'
'Hello'
>>> type("python")      #输出: <class 'str'>
<class 'str'>
>>> 'Blue' 'Sky'         #输出: 'BlueSky'
'BlueSky'
```

字符串编码-选择填空判断

- Python 3字符默认为16位Unicode编码
- 使用内置函数`ord()`可以把字符转换为对应的Unicode码；
使用内置函数`chr()`可以把十进制数转换为对应的字符

```
>>> ord('A')          #输出: 65
65
>>> chr(65)           #输出: 'A'
'A'
>>> ord('张')          #输出: 24352
24352
>>> chr(24352)         #输出: '张'
'张'
```

str对象

- 创建str类型的对象实例

`str(object='')` ······ #创建 str 对象，默认为空字符串 ↵

- **【例】** 使用str()创建字符串对象的应用示例：将给定整数的各个位数上的数字累加

```
intN=123456    #给定一个整数
total=0        #累加和total赋初值
for s in str(intN): #将整数转换为字符串，利用for语句迭代字符串序列
    total += int(s) #将字符转换为整数，实现各个位数上的数字累加
print(total)    #输出给定整数的各个位数上的数字累加和
```

程序运行结果如下所示

21 ↵

str对象属性和方法

- str对象的方法有两种调用方式：字符串对象的方法和str类方法
- **【例】str对象方法示例**

```
>>> s='abc'
```

```
>>> s.upper()    #字符串对象s的方法。输出: 'ABC'
```

```
'ABC'
```

```
>>> str.upper(s) #str类方法，字符串s作为参数。输出: 'ABC'
```

```
'ABC'
```

字符串的类型判断-选择填空判断

- `str.isalnum()` #是否全为字母或者数字
- `str.isalpha()` #是否全为字母
- `str.islower()` #是否全为小写
- `str.isupper()` #是否全为大写
- `str.isnumeric()` #是否只包含数字字符

• 例】字符串类型判断示例

```
>>> s1='yellow ribbon'
>>> s2='Pascal Case'
>>> s3='123'
>>> s4='iPhone7'
```

```
>>> s1.islower()
True
>>> s2.isupper()
False
```

```
>>> s4.isalnum()
True
>>> s3.isnumeric()
True
```

```
>>> s1.isdigit()
False
>>> s2.istitle()
True
```

- `str.lower()` #转换为小写
- `str.upper()` #转换为大写

```
>>> s2='Pascal Case'
>>> s3='python3.7'
>>> s2.lower()
'pascal case'
>>> s3.upper()
'PYTHON3.7'
```

- `str.strip([chars])` #去两边空格, 也可指定要去除的字符列表
- `str.lstrip([chars])` #去左边空格, 也可指定要去除的字符列表
- `str.rstrip([chars])` #去右边空格, 也可指定要去除的字符列表

- `str.startswith(prefix[, start[, end]])` #是否以prefix开头
- `str.endswith(suffix[, start[, end]])` #是否以suffix结尾
- `str.count(sub[, start[, end]])` #返回指定字符串出现的次数
- `str.index(sub[, start[, end]])` #搜索指定字符串，返回下标，无则导致ValueError
- `str.split(sep=None, maxsplit=-1)` #按指定字符（默认为空格）分割字符串，返回列表maxsplit为最大分割次数，默认-1，无限制

```
>>> s1='123'
>>> s2=' 123 '
>>> len(s2)
6
```

```
>>> s2.strip()
'123'
>>> s2.lstrip()
'123 '
```

```
>>> s1="abABabCD"
>>> s1.startswith("AB")
False
>>> s1.startswith("AB",2)
True
```

```
>>> s1.endswith("CD")
True
>>> s1.count("ab")
2
>>> s1.index("AB")
2
```

编写程序，生成一个包含20个0到100之间的随机整数的列表并打印列表，然后对其中偶数下标的元素进行降序排列，奇数下标的元素不变

- `import random`
- `x=[random.randint(0, 100) for i in range(20)]`
- `print(x)`
- `y=x[::-2]`
- `y.sort(reverse=True)`
- `x[::-2]=y`
- `print(x)`

- 列表 元组 集合等的特点比较（是否可变 是否有序 是否允许有重复的元素）
- 对象的id和修改了相应的元素之后，其内存地址是否发生变化，元组和列表之间元素不可修改指的是不可原地修改，列表的“+”和列表的append之间的区别（append是原地修改）
- 此处可以出选择填空判断，也可以结合具体的实例出代码相关的题目

math库和random库

- **math模块包含两个常量math.pi和math.e，分别对应于圆周率 π (3.141592653589793) 和自然常数e (2.718281828459045)**
- **random模块包含各种伪随机数生成函数，以及各种根据概率分布生成随机数的函数，使用random模块函数seed()可以设置伪随机数生成器的种子**

- >>> import random
- >>> random.seed(1) #设置种子为1
- >>> for i in range(5): print(random.randint(1,5),end=',') #输出:
2,5,1,3,1,
- >>> for i in range(5): print(random.randint(1,5),end=',') #输出:
4,4,4,4,2,
- >>> random.seed(1) #重新设置种子为1, 结果重复
- >>> for i in range(5): print(random.randint(1,5),end=',') #输出:
2,5,1,3,1,
- >>> random.seed(10) #重新设置种子为10
- >>> for i in range(5): print(random.randint(1,5),end=',') #输出:
5,1,4,4,5,

名称	说明	示例	结果（随机）
randrange(stop)	返回随机整数N， N属于序列[0, stop)	for i in range(10): print(randrange(10),end=',')	8,6,7,4,0,9,1,5,4,9,
randrange(start, stop[, step])	返回随机整数N， N属于序列[start, stop, step)	for i in range(10): print(randrange(1,5),end=',')	3,4,1,2,3,4,4,4,1,1,
randint(a, b)	返回随机整数N， 使得$a \leq N \leq b$， 即 randrange(a, b+1)	for i in range(10): print(randint(1,5),end=',')	1,2,5,5,1,1,1,2,3,4,

random()	生成一个[0.0, 1.0)之间的随机小数
uniform(a, b)	生成一个[a, b]之间的随机小数
choice(seq)	从序列类型(例如： 列表)中随机返回一个元素
shuffle(seq)	将序列类型中元素随机排列， 返回打乱后的序列
sample(pop, k)	从pop类型中随机选取k个元素， 以列表类型返回

• random库实例

首先随机产生一个1~100以内的整数，请用户猜测具体是哪个数，
即：不断从标准输入读取用户的猜测值，并根据猜测值给出提示信息：
“太大”、“太小”或“正确！”

```
import random
secret = random.randrange(1, 101)
guess = 0
while guess != secret:
    guess = int(input("请猜测一个100之内的数: "))
    if (guess < secret): print('太小')
    elif (guess > secret): print('太大')
    else: print('正确! ')
```

其他可能会涉及到的考点

- Jieba库 wordcloud库 time库 datetime库
- 函数的全局变量和局部变量（global保留字）
- for else的搭配使用
- 素数 回文数...

Numpy的求和计算

```
>>> import numpy as np
>>> a=np.arange(24).reshape(3, 4, 2)
>>> a
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5],
        [ 6,  7]],
       [[ 8,  9],
        [10, 11],
        [12, 13],
        [14, 15]],
       [[16, 17],
        [18, 19],
        [20, 21],
        [22, 23]]])
>>> a.sum(axis=1)
array([[12, 16],
       [44, 48],
       [76, 80]])
>>> a.sum(axis=0)
array([[24, 27],
       [30, 33],
       [36, 39],
       [42, 45]])
>>> a.sum(axis=2)
array([[ 1,  5,  9, 13],
       [17, 21, 25, 29],
       [33, 37, 41, 45]])
```



谢谢