Python程序设计复习

考试信息

- 考试时间地点：
  - 2023年6月11日 8:30-10:00，H3108，闭卷（纸质，没有计算机调试）考试
- 考试题型：

1.阅读程序写输出3道（30%）。3 * 10 =30%

2.程序填空题2道（20%）。2 * 2 * 5 = 20%

3.程序改错题1道（10%）。2 * 5 = 10%　指出错误行的位置并且改正

4.编程题3道（40%）。 10 + 10 + 20 = 40%

解释和编译：了解两者区别，python属于哪种

- 如何将使用高级语言编写的程序（源代码source code）转换为目标代码（机器码）？
  - 解释器（Interpreter）：**将语句翻译成机器码，并且执行**，Python、Javascript、Perl、PhP等
    - 修改程序方便，修改代码重新运行就可以了
    - 每次运行，都要进行翻译，运行速度会有影响
    - 必须有解释器才可以运行，跨平台
  - 编译器（Compiler）：**将语句翻译成机器码，形成目标代码文件**，C、C++等
    - 编译时相比解释可以作更多的优化
    - 修改程序后需要进行编译
    - 编译一次，然后在执行过程中不再需要翻译语句
    - 编译后的目标代码可以直接在相应的操作系统中运行，不再需要编译器
  - 有些语言将解释和编译结合在一起
    - Java语言：源代码首先通过编译器（javac）转换为中间的Java字节码（Byte Code），然后在目标机器上通过解释器（java虚拟机）来运行
    - Python也支持伪编译。对于被import的模块源程序，首先将对应.py程序转换为.pyc字节码来优化程序和提高执行速度

Python语言

- Python以是一种**解释型高级**动态编程语言，广泛应用在系统管理、科学计算、大数据、Web应用、图形用户界面开发、游戏等
- Python语言的特点*：
  - **简单、易学**：Python是一种代表简单主义思想的语言。
  - **开源、免费**： Python是FLOSS（自由/开放源码软件）之一。使用者可以自由地发布这个软件的拷贝、阅读它的源代码、对它做改动、把它的一部分用于新的自由软件中。
  - **跨平台**：所编写程序在解释器支持下可无需修改在Windows、 Linux、 Mac等操作系统上使用
  - **灵活性**：Python支持多种编程范式，包括过程式编程、面向对象编程、函数式编程，
  - **可扩展和可嵌入性**：支持采用C、C++等语言编写扩充模块，也可为C、C++程序提供脚本功能
  - **丰富的扩展库支持**：拥有大量的几乎支持所有领域应用开发的成熟扩展库。

了解python语言的主要特征

- 掌握对象与名字的概念
- 了解对象的三个基本属性
- 掌握isinstance(obj, class or tuple)

1.4 Python基础知识

- Python对象：任何数据都是对象
  - ID：该对象的唯一标识，内部表示方式，用户一般不用关心其具体取值
  - Type:对象的类型，决定了其取值范围以及支持的运算
  - Value:对象的取值，根据值是否可以修改分为不可变和可变对象
- （变量）名字：表示对于某个对象的引用（reference）
  - name与object之间的绑定一般通过赋值语句来完成
  - name可以属于不同的命名空间namespace（模块、函数）
  - 所谓name的类型实际上指的是name所指向的对象的类型：无需预先声明name的类型
  - **名字出现在RHS表示引用该名字指向对象的值**
  - **名字出现在LHS表示将RHS计算后得到的对象与名字绑定起来**
  - 解释器会自动回收那些不再有名字绑定的对象
  - 可以通过del 语句解除名字与对象的绑定

什么是一个合法的变量名?

什么是一个合法的变量名?

- 必须以**字母或下划线开头，其后的字符可以是字母、下划线或数字**
- 以下划线开头的变量在Python中有特殊含义
- 变量名中不能有空格以及标点符号（括号、引号、逗号、斜线、反斜线、冒号、句号、问号等）
- 英文字母的**大小写敏感**，例如student和Student是不同的变量
- 一些特殊的标识符保留为Python关键字，不能用作变量名

第1章 基础知识：数字类型与运算

- 数字：整数(int)、浮点数(float)、复数(complex，不考)、布尔（bool）
  - 整数字面量：十进制整数、十六进制、八进制和二进制整数（0x/0o/0b…）
  - 浮点数字面量：
    - 小数表示：3.14　10.　.001
    - **科学计数法**：15e-2 ($=15*10^{-2}=0.15$)　3.14e-10　1.0E100
  - **/：浮点除法或真除法，结果为浮点数**
  - **//：整除法，如果都为整数，则结果为整数，如果其中有浮点数则结果为浮点数**
  - **求模运算：可用于判断是否整除，判断奇偶性，可与//结合在一起获得整数的各位数字**
  - 幂运算：**
- 算术运算：　+ - * / // % 一元+或-　**
  - 优先级顺序，　**最高，接下来为一元改符号，然后是乘除，最后为加减
  - 结合：之后**为右结合，其余为左结合
  - 除法运算：/ 为真除法，最后为浮点数，　// 为整除，求整商，全部为整数时结果为整数，其中有浮点数则结果为浮点数；
  - 除了/运算外，两个不同数字类型之间的混合运算通过**类型提升**提升为更高的类型后计算，　int→float→complex
- 比较运算符：　== != > < >= <=　　in　not in　is　is not
  - 比较运算符支持连续比较，但是没有传递性　x <y < z，x<y > z
- 内置数学函数: divmod、max、min、sum等
- math模块：sqrt、log等
- **dir和help函数查找有哪些方法以及方法的帮助**

- 掌握真除法和求整商运算符的区别
- 掌握求整商和求余(求模）运算符，知道如何实际运用

运算符优先级

- 优先级越低，越最后计算
- 除了**为右结合外，其他运算符都是左结合
- lambda < if-else < 逻辑运算符(or,and,not) < 比较运算符 < 两元位运算 < 算术运算 < 一元运算 < 求幂运算 < 下标、切片、函数调用、字面量定义等
- 赋值不是运算符，其采用右结合方式

- 掌握运算符优先级以及结合方式
- 先算后比，比完之后进行逻辑运算
- 一元优先级比两元优先级更高
- and比or优先级更高

运算符	描述
lambda	Lambda表达式
if-else	3元条件表达式
or	布尔“或”
and	布尔“与”
not x	布尔“非”
in，not in	成员测试
is，is not	同一性测试
<，<=，>，>=，!=，==	比较
|	按位或
^	按位异或
&	按位与
<<，>>	移位
+，-	加法与减法
*，/，%	乘法、除法与取余
+x，-x	正负号，一元运算或单目运算
~x	按位求反
**	求幂
x.attribute	属性
x[index] x[index:index]	下标和切片
f(arguments...)	函数调用
(experession,...)	元组字面量
[expression,...]	列表字面量
{key:datum,...}	字典字面量
'expression,...'	字符串字面量

input(prompt=None):

- 如果传递prompt，则首先显示提示符。等待用户输入，直到用户键入回车时input返回，将**回车之前的内容(字符串)**返回给调用者
- 注意**返回的是str**，可能需要**通过int/float等转换为数字类型的对象**，如果输入多个数据时通过split进行分割后再转换

```
text = input('....')
if not text:
  # 输入为空字符串，结束
```

- 掌握input()函数
- 掌握 输入一个合法的整数或者浮点数，和循环、异常处理一起结合处理
- 掌握输入多个整数：输入一行文本，将文本分割

```
while True:
    try:
        text = input('...')
        t = int(text)
        ...
        break
    except Exception as e:
        print(type(e), e)  # 输入有误
```

如果要输入的是多个参数(整数)，以某个分隔符隔开，比如/

```
text = text.strip()
args = text.split('/')
try:
    # args = [int(item.strip()) for item in args]
    args = [int(item) for item in args]
    ....
except Exception as e:
    print(e)  # 输入有误
```

- **eval(source):** 分析字符串source中的表达式，返回一个该表达式对应的对象
  - eval('123')返回整数123
  - eval('1+2')返回3
  - eval('1,2')返回元组(1,2)
- **print(value1,value2,…,sep=' ',end='\n',**file=sys.stdout**)**：
  - 注意sep和end的含义
  - 输出每个参数，参数之间以sep隔开，最后输出end
  - 返回None

- 掌握eval(source)函数，分析并计算字符串source中的表达式，返回该表达式运算后的值
- 掌握print()函数的用法，掌握sep/end的含义， 返回None

第1章 基础知识：模块

- 模块导入：
  - **import module [as alias]**
    - 如果该module首次导入，寻找模块源文件，加载模块，创建模块对象
    - module或者alias与该模块对应的模块对象绑定
  - **from module import obj [as alias]:** 从模块中导入特定的对象，即obj与对象绑定
  - **from module import ***
- 模块的__name__属性：
  - 作为script运行时，__name__等于'__main__'
  - 否则被import时，__name__为模块的名字

- 掌握import语句：从哪个模块中导入，把什么导入到当前名字空间？
- 了解名字__name__

第2章 Python序列：序列概述

- 掌握序列对象的区别，是否有序、是否可变等
- 有序对象(包括range对象)可以通过下标和切片访问

- 无序和有序：
  - set、dict为无序序列
  - list、tuple、str、range为有序序列：
    - 可以通过下标、切片等来访问序列中的元素
- str的元素为单个字符，而list、tuple、set、dict的元素可是其它python对象
  - set要求其元素为不可变对象
  - dict要求其key为不可变对象
- 可变和不可变：
  - list、set、dict为可变序列
  - tuple、str为不可变序列，其中**tuple为元素不可添加、删除、修改，但是其元素所指向的对象本身的值可以修改**

第2章 Python序列：序列创建

- 掌握各个序列对象的字面量定义
- 掌握各个序列对象的构造函数方法，注意字典对象作为iterable对象时，等价于调用其keys()方法

list、tuple、dict、set各个序列的定义方式

- 调用构造函数： list([iterable])、tuple、dict、set
  - 其中dict传递的参数为iterable时，要求每个元素必须包括两个子元素
  - dict(**kwargs)也可以采用关键字参数，参数名作为key，值作为value
  - dict.fromkeys(seq[,value]): 序列中的元素作为key，值为value，缺省为None
- 字面量(Literal)定义：
  - 分别为[]、( )、{ }、{ }
  - 元素中间用,分割
  - 考虑到()也可用于表达式，如果tuple只有一个元素时，添加逗号，即 (1,)
  - dict和set都采用{}，因此空set不能用literal定义，而是set()
  - dict的每个元素用 key:value的形式表示，key必须是immutable对象，但是immutable对象不一定可以作为key，value可是任意python对象

第2章 Python序列：range

range(stop):

range(start,stop[,step])

- 返回一个range对象，可产生一系列的整数，从start（缺省为0）开始，直到stop为止（不包括），step为步长，缺省为1

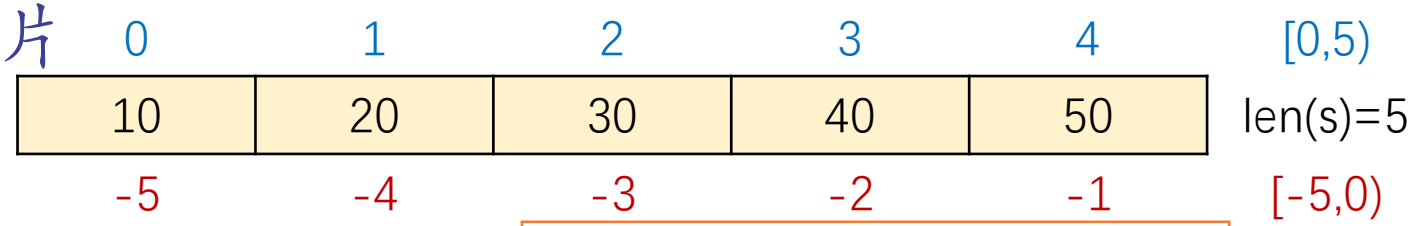
range(10)

range(1, 100, 2)

range(10, 0, -1)

- 掌握range对象的使用方法
- range(n) 产生n个数值，可用来重复n次
- range(len(s)): 产生s中每个元素对应的下标
- range(len(s)-1, -1, -1) 或range(-1, -len(s)-1, -1) 逆序的下标

第2章 Python序列：下标和切片



- 有序对象的下标：
  - 下标从0开始，到len(s)-1为止
  - 负数下标从-1开始到 –len(s)为止，与正数下标的关系为加上len(s)

s[:] 有序对象的copy
s[::-1]: 逆序
s[1:-1]: 去掉第一个和最后一个
s[0::2] : 偶数位置
s[1::2]: 奇数位置

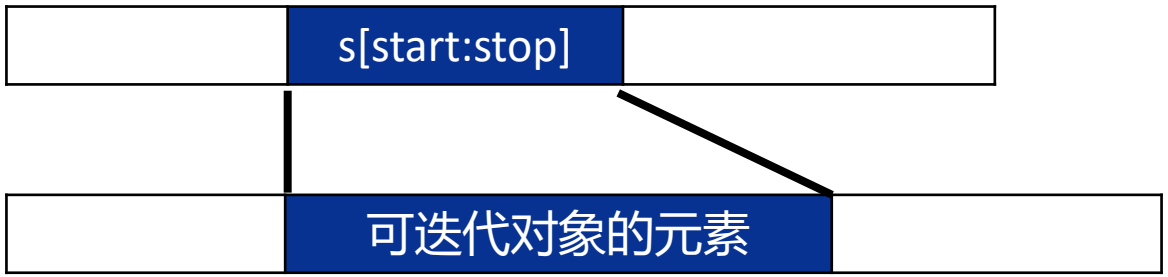
- 切片访问： [start:stop] or [start:stop:step]
  - step省略，表示缺省为1
  - 如果**step大于0**，从下标为start的元素开始按照顺序**往后**直到下标为stop的元素为止（不包括下标为stop的元素）
    - start可省略，表示下标为0
    - stop可省略，表示下标为列表结束后的下一个位置，即len(s)
  - 如果**step小于0**，则从下标为start的元素开始，按照相反的顺序**往前**直到下标为stop的元素为止（不包括下标为stop的元素）
    - start可省略，表示下标为-1或者len(s)-1
    - stop可省略，表示下标为-len(s)-1
  - start/stop可以不在合法下标范围，**会自动截取到合适的位置**
  - 切片出现在RHS时表示返回的是原列表的相应元素组成的新的序列对象，比如 new_lst = lst[:]
- 可以使用切片原地修改列表内容： 插入`lst[i:i] = iterable` 更新`list[i:j] = iterable`

切片：原地修改列表

- **切片出现在表达式处，表示根据指定位置的元素构造一个新的列表**
- **切片出现在赋值语句左边，表示原地修改(插入、替换或删除）列表中的元素**

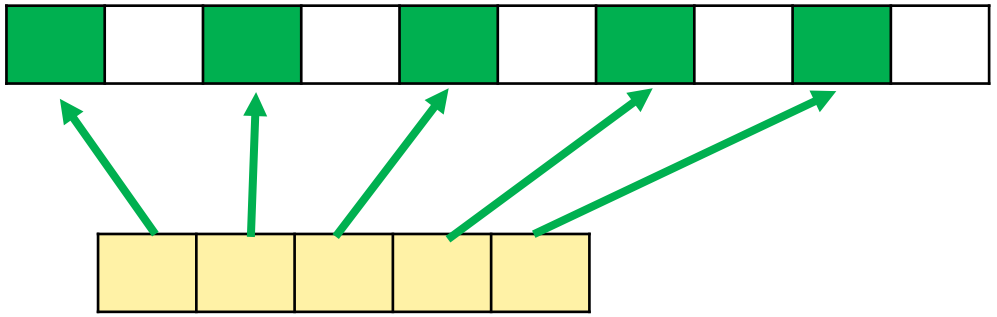
- 掌握切片的两种格式，两种格式中哪些可以省略，表示什么含义：s[1:3], s[1:5:2]
- 全切片指的是s[:]
- 掌握切片可以适用于有序对象
- 掌握使用切片原地修改列表的方法
- 掌握del语句删除列表中的元素或者多个元素的方法

在下标i处插入多个元素： `list_var[i:i] = iterable`



替换或者删除元素

```
s[start:stop] = iterable
```



```
s[::2] = iterable
```

步长不为1时要求左右两边的元素个数一样
→ 无法通过赋值来删除不连续的多个元素

删除列表中的多个元素

```
del s[start:stop]
del s[start:stop:step]
```

第2章 Python序列：有序序列的操作

- 掌握哪些对象可以比较：complex与dict无法比较
- 掌握序列对象的比较运算

- 有序序列支持：下标和切片
- 成员关系判断：in not in （注意与is is not的区别）
- index和count，返回元素在序列中的下标和出现次数，不存在时会抛出ValueError
- 运算符：
  - + 创建新的序列，两个序列元素合并
  - += 对于列表，表示原地合并列表
  - * 其中一个参数为整数，表示重复序列的元素
- 比较运算：
  - 同种类型的对象的比较，但注意数字类型可以进行类型提升： 1 < 1.45
  - 不是所有的对象都支持比较，比如complex、dict不支持
  - 序列的比较为对应元素的比较，直到确定关系为止： (2,3) < (2,4,5) (2,4) < (2,4,5)
  - set的比较运算为子集，真子集的运算

第2章 Python序列：有序序列的常用内置函数

- 掌握这些常用内置函数的用法，特别是sorted/enumerate/zip函数

- reversed(iterable):
  - 返回一个迭代器对象，为可迭代对象的各个元素逆序
  - obj[::-1] 有序对象可通过切片获得其逆序
- sorted(iterable, reverse=False, key=None): 排序后返回新的list
  - 如何定义函数或采用lambda表达式进行排序
- len(iterable)、max(iterable)、min(iterable)： 长度，最大值，最小值
  - max和min也可以如同sorted一样，传递key参数，作为求最大最小值的基准，如 max(text.splitlines(), key=len)
- sum(iterable): 数值元素的和
- enumerate(iterable[,start]): 返回枚举对象，每次返回包含下标和值的元组
  - start缺省为0
  - 用于遍历iterable对象，在遍历的同时也知道是第几个元素
  - 除了采用enumerate外，还可以采用 for i in range(len(s))，或者引入变量idx + for/while来实现上述功能
- zip(iter1,iter2...): 返回zip对象，每次返回各个可迭代对象中对应的相同位置的各个元素组成的元组，相当于zip shortest。
  - 如果希望zip longest呢？可使用functools的zip_longest方法，或者自己填充短的，使得一样长： list(iterable1) + [None] * (len(iterable2) - len(iterable1))

第2章 Python序列：列表的常用操作

除了index, count, pop等，其他方法返回值都是None

- list.append(x)和list.extend(L)：注意两者的区别
- list.reverse()和list.sort()：原地逆序和排序操作，与内置函数reversed和sorted的区别

列表元素增加方法	别名	效果
list1 + list2	拼接	新建列表
append(obj)	附加	原地修改
extend(iterable)	扩展	原地修改
insert(index, obj)	插入	原地修改
list1 * n	复制	新建列表
list1 += iterable	复合赋值	原地修改
list1 *= n	复合赋值	原地重复

方法	作用	返回	备注
del list_obj[index]	删除指定位置元素	无返回值	IndexError,TypeError
pop([index])	删除指定位置元素并返回该元素	返回元素	IndexError,TypeError
remove(value)	删除首次出现的指定元素, 返回None	返回None	没有相应元素时ValueError
clear()	删除所有元素	返回None	空列表

第2章 Python序列：列表解析式和生成器表达式

[表达式 for 变量 in 列表] 或者 [表达式 for 变量 in 列表 if 条件]

- 对列表中的每个元素，在满足条件的情况下，计算表达式的值作为新列表的元素
- 表达式可以包括函数调用等

- 掌握列表解析式(包括字典解析式等）的写法，要求掌握一个for子句的情形
- 了解生成器表达式与列表解析式的区别

```
[x**2 for x in range(100) if x%3 == 0 ]

[weapon.strip() for weapon in freshfruit]

[(x,y,z) for z in range(100) for y in range(1,z) for x in range(1,y) if x*x + y*y == z*z ]

{k:len(v) for k, v in d.items() if k > 0}
```

- **生成器表达式**的写法与列表推导式基本类似，只是用圆括号，但是：
  - 列表推导式计算的结果是**新的列表**，一次生成所有的元素
  - 生成器表达式的结果是一个生成器对象，**符合迭代器协议（打开的魔盒）**，调用next时才会实时返回下一个元素

```
gen = (i*2 for i in range(10))
for item in gen: print(item)
for item in gen: print(item)    # 不会有任何输出
```

第2章 Python序列：序列解包

- 掌握赋值语句中的基本序列解包以及嵌套序列解包
- 掌握函数调用时的序列解包
- 掌握序列解包可用于交换元素
- 了解扩展序列解包

LHS = RHS

- LHS为对象引用，变量或者下标切片描述的多个list元素
- LHS允许出现*seq，但只允许出现一次
  - 该引用前后的变量一一对应赋值后，剩余的变量转变为list然后赋予该引用
- RHS可以是任何可迭代对象，包括tuple、list、dict、range、str等，逐个取该序列的元素赋予左边对应位置的对象引用
- 除*seq以外的LHS的变量引用与RHS的元素必须一一对应
- 嵌套序列解包
- 交换a和b:　a, b = b, a
- 与eval结合在一起进行输入：　a,b,c = eval(input('Please input a,b,c'))

```
a, b,c = [1,2,3]
a,*b,c = range(1,7)
for index,value in enumerate(dList):
      print('%d:%s' % (index,value))
```

```
for index,(key, value) in enumerate(dictobj.items()):
      print('%d. %s:%s' % (index, key, value))
```

序列解包的各种情形

使用场景	描述
x, y, z = range(3)	赋值语句的基本序列解包，支持嵌套序列解包
x, y, z, *seq = range(10)	**赋值语句的扩展序列解包**，支持嵌套序列解包。首先匹配其他位置的对象引用，剩余的元素作为seq所指向的新列表中的元素
func(x, y, *seq)	函数调用时的序列解包，将可迭代对象seq的各个元素拆分后作为位置实参
func(x, y, *seq, **map)	函数调用时的序列解包，表示将map对象(如字典)的各个元素拆分后作为关键实参 (var=value)
`[*range(4), 4]`	元组、列表、集合字面量定义时的序列解包，将可迭代对象的各个元素拆分后作为新的对象中的元素
`{'x': 1, **{'y': 2}}`	字典字面量定义时的序列解包，将map对象的各个元素(key:value)拆分后作为新的字典对象中的元素

使用场景	描述

def func(*args, **kwargs):
  print(args)
  print(kwargs)
	函数定义时可变长度位置参数以及可变长度关键字参数，收集调用时传递的尚未匹配的位置参数以及关键字参数，以元组和字典形式保存

第2章 Python序列：字典

- 掌握字典如何初始化，如何添加或修改元素
- 掌握字典的value为不可变对象时基本的更新方法，特别是get()方法
- 掌握字典的value为可变对象时基本的更新方法

- 字典元素的读取：
  - d[key] 以键作为下标可以访问字典元素，若键不存在则抛出异常KeyError
  - d.get(key[,default])：key不存在时返回0
- 字典元素的添加与修改： d[key]=value
  - d.update(another)
  - d.setdefault(key, default)： update + get结合
- 字典元素的删除：
  - del d[key]
  - d.pop(key)/d.pop(key, value)
    - reverse = d.pop('reverse', True)
  - d.popitem()
  - d.clear()

如果字典的value保存的为不可变对象

```
d = dict()  #  d = { }

d[key] = d.get(key,0) + 1

if  key in d:
   d[key] += 1
else:
   d[key] = 1

try:
   d[key] += 1
except KeyError:
    d[key] = 1
```

如果字典的value保存的为可变对象，比如为列表

```
current_or_init_value = d.get(key, [ ])
current_or_init_value.append(value)
d[key] = current_or_init_value

d[key] = d.get(key, [ ])
d[key].append(value)

if key not in d:
   d[key] = [value]
else:
   d[key].append(value)

try:
   d[key].append(value)
except KeyError:
   d[key] = [value]

d.setdefault(key, [ ]).append(value)

# 最不推荐的方法
d[key] = d.get(key, [ ]) + [value]
```

第2章 Python序列：字典的迭代

- keys(): 返回可迭代对象，元素为字典中的所有key
- values(): 元素为所有value
- items()：元素为(key,value)元组
- iter(d) or for key in d: 等价于iter(d.keys())：元素为所有key

- 掌握遍历字典中的元素的方法，知道key可以得到其对应的value
- 掌握字典与sorted、切片结合在一起，实现类似于topn的功能

字典的知识点：

- 看到采用字典，怎么样更新或者添加字典元素？ dictobj[key] = ...
- 更新字典元素：
  - dictobj[key] = dictobj.get(key, value) + xxxx
  - d.setdefault(key, [ ]).append(value)
- 要遍历字典的元素： for key in d / for key in d.keys() d.values() d.items()

第2章 Python序列：set

- 集合的特性： 不重复且无序，元素必须为不可变对象
- == != in not in len ….

t = [1, 2, 3, 1, 4, 1]

- add(x), remove(x), discard(x)

- 掌握集合的特性
- 掌握如何利用集合的不重复特性去除重复对象和判断是否有重复

怎么去除重复对象?

s = list(set(t))

s = list(dict.fromkeys(t))

怎么判断字符串中字符是否有重复?

len(set(t)) == len(t)

将集合a中的那些属于集合b中的元素移走：{'reverse', 'key', 'option'} - {'reverse', 'key', 'verbose'}

第3章 选择与循环：条件表达式

- 掌握对象的真值判断方法
- 掌握 and/or/not/ifelse逻辑运算符，什么情况下真值判断为真?
- 掌握短路逻辑的概念
- 掌握德摩根定理和分配律
- 了解and/or运算的最终结果

- 任何表达式都是条件表达式，可用于选择和循环结构中
  - 表达式不能包含赋值类语句
- 条件表达式的真值判断：
  - False: 值为0、None、空序列对象（长度为0）
  - True: 非0、非空对象
- 逻辑运算符：and or not，注意短路逻辑
  - and : 返回第一个假(None、空或者数值0)的表达式或者最后一个表达式
  - or: 返回第一个真（非None、非空或者非0）的表达式或者最后一个表达式
  - not: not True = False, not False = True
  - 主要考察逻辑运算用于选择或循环结构中作为条件判断的情况
- if else三元表达式： value = x ** 2 if x % 2 == 0 else x ** 3
- **德摩根定理(De Morgan's Law)**

```
not (expr1 and expr2) = not expr1 or  not expr2
not (expr1 or  expr2) = not expr1 and not expr2
```

分配律：expr1 and (expr2 or expr3) = (expr1 and expr2) or (expr1 and expr2)

expr1 or (expr2 and expr3) = (expr1 or expr2) and (expr1 or expr3)

短路逻辑

- 逻辑运算 and/or为短路逻辑，多个条件，如果其中有条件无需运算就已经知道最终结果，这就是短路逻辑

5< 4 and 'a' in 'abc' 短路逻辑， 5<4 为False，最终结果为False，无需判断 'a' in 'abc'

5 >4 and 'a' in 'abc' or 6+4 > 7

首先决定运算符优先级： 5>4 'a in 'abc' 6+ 4 10 >7

((5>4) and 'a' in 'abc') or (10 > 7)

or前面条件为真，短路逻辑，无需判断 6+4>7

- 单分支、双分支和多分支
- 选择结构的嵌套，注意缩进

- 掌握选择结构，包括单分支和双分支
- 掌握选择结构的多分支

```
if expr:
    statement1
elif expr2:      # 前面条件不满足时
     statement2
elif expr3:
    statement3
else:
   statement_else
```

- break和continue语句的含义
- else在什么情况下执行？循环不是通过break跳出时才执行
- 理解循环嵌套

- 掌握break和continue语句
- 掌握else子句

```
while 条件表达式:
    循环体
    if 条件表达式1: break  #可选
    # Exit loop now, skip else if present
    if 条件表达式2: continue  # 可选
    # Go to top of loop now
else:  #可选
    # Run if we didn't hit a 'break'
    else子句
```

```
for target in iterable: # Assign iterable items to
target
    循环体
    if 条件表达式1: break #可选
     # Exit loop now, skip else if present
    if 条件表达式2: continue #可选
   #go to top of loop now
else:  #可选
    else子句 # If we didn't hit a 'break'
```

- 看到 for i in range(xx): 边界条件是否正确?
- 看到 for item in iter_obj:且循环体中 iter_obj.remove(...)，应该采用 iter_obj[:]或其他方法
- 看到while循环，循环变量是否有更新

问题求解:

- 首先确定步骤
- 将每个步骤转变为python代码
- 对于多分支情况：确定条件表达式以及相应分支
- 对于要执行多次情况：采用循环结构
  - 循环次数是否已知： for 或者 while expr
  - 确定循环结束的条件:
    - if expr: break
    - while not expr:
  - 确定循环继续条件：continue
  - 不要忘记更新循环变量

- 取整数的各位上的数字

loop: 从最低位开始取，重复以下步骤，直到当前数为0:

使用%运算，取最低位数字

使用//运算，取剩下的数字

- 如果已知整数字符串，如何得到对应的整数呢？不使用int

- 栅栏循环：用户连续输入整数，每次输入后进行计算，直到用户输入空字符串结束

```
while True:
    text = input('....')
    if not text:
        break
    try:
        arg = int(text.strip())
        print(arg)
    except Exception as e:
        print(e)
```

- 用循环解决级数求和问题
  - 项数之间的关系: 对于第n项，等于前一项/n --> item /= n
  - 循环变量n，初始值为1，和的初始值为1，结束值未知，结束条件为item <= error_factor
  - 循环变量n每次加1

```
approx = item = 1
n = 1
while True:
    item /= n
    approx += item
    if item <= error_factor:
        break
    n += 1
```

- 掌握使用循环结构解决实际问题的方法
- 掌握用户输入合法整数的方法
- 掌握遍历容器对象，对于每个元素（还要知道是第几个元素）进行处理的方法
- 掌握级数求和问题的解决方法
- 掌握图案问题的求解方法

图案问题

- 关键是观察各个行的变化情况，找到反映这个变化的改变量，从而确定外层循环变量
- 接下来在外层循环变量已经确定的情况下看当前行是否能够从整体上考虑还是需要进一步考虑各个字段的变化情况(99乘法表的例子)
- 整体上考虑的话，需要找到对应的特征，同时还可以利用填充和对齐。比如打印三角形

多重循环

- 确定到底需要几层循环，
- 每层循环的初始值和结束值是什么，注意内层循环可以使用外层循环已经确定好的具体取值

比如水仙花数为三位数，其等于各位数字的立方和:

- 三层循环，每层循环为各个数字的可能取值，最内层已知这些数，组合成一个三位数，再检查其条件是否满足即可
- 会有多个解，首先初始化一个空列表，然后每找到一个解，附加到该列表即可

```
narcissi_fews = [ ]
for hundreds in range(1, 10):
    for tens in range(10):
        for ones in range(10):
            n = 100 * hundreds + 10 * tens + ones
            if  ones ** 3 + tens ** 3 + hundreds **3 == n:
                narcissi_fews.append(n)
```

第4章 字符串与正则表达式

- 字符串字面量
  - 单引号、双引号、三单双引号（可以跨越多行）
  - 字符转义：\n \t \\ \'
  - 原始字符串: 原来的字面量定义前添加r或R，比如r'c:\user\xxx\sample.txt'
- 字符串运算： + * %
- 常用字符串相关内置函数：
  - **chr、ord**
  - **int/float/str**

a	b	c	...	x	y	z
A	B	C		X	Y	Z
0	1	2	...	23	24	25

```
chr(ord(ch) - ord('A') + ord('a'))
alphabet='abcdefghijklmnopqrstuvwxyz'
lower = alphabet[ord(ch) - ord('A')]
```

- 掌握字符串字面量定义的方法，包括原始字符串
- 掌握基本的字符转义方法
- 掌握字符串运算 + %，以及常用内置函数(chr/ord/int/float/str)
- 掌握字符串的格式化方法：%和format方法
- 掌握字符串的常用方法：
  - 去除首尾空格(字符): strip
  - 判断字符串类型：isspace/isdigit/isalpha等
  - 判断字符串的前后模式：startswith/endswith
  - 字符串的大小写转换：lower/upper
  - 查找： count/index/rindex/find/rfind
  - 分割和合并：split和join
  - 替代：replace/maketrans + translate

格式化： % 和format方法

format_string % (value1, value2, …)

- % [flags] [width] [.precision] type

format_string.format(value1, value2, …)

- {field:[fill][align][width][.precision]type}
- 对齐： < 左对齐， ^居中对齐，>右对齐

```
"First %s, Second %d, Third %5.2f" %
('idle',90,93.4)
"First {0:s}, Second {1:d}, Third
{2:5.2f}".format('idle',90,93.4)
```

动态构造：

```
>>> '%*.*f' % (10, 2, 3.1415)
'    3.14'
'{0:{1}}'.format(12345, 10)    # '{0:10}'.format(12345)
'     12345'
```

% 格式	format格式	说明
%s	{:s}	输出字符串
%10s	{:10s}	格式化为字符串，最小宽度为10
%5d	{:5d}	将整数格式化为总宽度至少为5的十进制形式，不够填充空格
%10.2f	{:10.2f}	格式化浮点数，总宽度为至少10，四舍五入到小数点后第2位
%-d	{:<d}	缺省右对齐，-表示左对齐

第4章 字符串与正则表达式：方法

- 通用序列方法：下标、切片、in、len、sorted、reversed、enumerate、zip
- 去首尾(空白）字符：strip([chars]) lstrip([chars]) rstrip([chars])
  - 去除字符串前后的某些字符。
  - 缺省是去除空白类字符(空格、制表、换行等)

```
text = input('...')
text = text.strip()
args = text.split(',')
args = [item.strip() for item in args]
```

  - 分割单词时，如果要去除前后的标点符号

```
text = input('.....')
text = text.strip()
words = text.split()
words = [word.strip('\'"?.!') for word in words]
```

第4章 字符串与正则表达式：方法

- 字符串类型判断： isdigit isalpha isspace islower isupper
  - 判断字符串的**每个字符**是否是小写,大写,空格,数字,字母等
- 大小写转换：lower upper
- 测试和查找：
  - startswith(prefix) endswith(prefix)
  - count(sub) index(sub) rindex(sub)
  - find(sub) rfind(sub)：注意find和index的区别，是否抛出异常

```
if not line or line.isspace():
   print('empty line')
```

```
text = input('continue..? y or n')
if lower(text) == 'y' :
   print('bye')
```

```
pos = strobj.find(substring)
if pos != -1:
    print(strobj[:pos], substring,
strobj[pos + len(substring):])
```

```
try:
    pos = strobj.index(substring)
    print(strobj[:pos], substring, strobj[pos
+ len(substring):])
except Exception as e:
    print('not found')
```

第4章 字符串与正则表达式：方法

- 拆分和组合： split( ) splitlines() rsplit( ) split(sep) join(iterable)
- 替代: replace()
- 填充和对齐： center ljust rjust(width[,fillchar])
- 翻译和转换: maketrans/translate

split(sep)以sep分割字符串，分割后可能出现空字符串，分割后的字符串可能前后有空格

```
text = input('...')
args = text.split(',')
args = [item.strip() for item in args]
args = [item for item in args if item]
```

```
text = input('....')
text = text.replace(',', ' ')
str_args = text.split()
```

可使用join方法将字符串列表(可迭代对象)合并成字符串

```
str_list = [str(i) for i in range(10)]
text = '+'.join(str_list)
```

```
tab2 = str.maketrans('123456789','*'*9)
s1.translate(tab2)
```

第4章 字符串与正则表达式:

正则表达式的元字符集： ^ $ * + ? | { } [ ] ( )

- 字符类： [xyz] [^xyz] [a-z] [^m-z]
- 预定义字符类: . \d \D \w \W \s \S
- 边界匹配符: ^ $ \b \B
- 重复限定符: ? * + {n} {n,} {n,m}
- 分组符() \1 (?P=name)
- 选择符|
- 元字符需要转义： * 或者[*]，如果需要动态构造正则表达式，可能要使用re.escape(pattern)
- 正则表达式模式：采用原始字符串 r'....'

- 传递正则表达式建议采用原始字符串
  - str字面量定义与re所采用的转义字符都是\
- 懒惰和贪婪匹配
  - r'<.+?>' 匹配 < 尽可能少匹配1个以上字符 >
  - r'<[^>]+>' 匹配< 尽可能多匹配1个以上非>字符 >

- 掌握基本的正则表达式的写法： 如何描述字符集(包括预定义字符类)，如何描述某个子模式或字符的重复次数，如何通过选择符匹配多个子模式中的任一个子模式
- 掌握正则表达式中转义字符的使用方法，特别是 \. \b * \(\)等， 掌握re模块的escape方法
- 掌握使用search/findall方法获得匹配内容的方法，掌握使用findall匹配时正则表达式出现分组符如何处理
- 掌握使用sub方法进行替代的方法
- 掌握使用split方法进行分割的方法

正则表达式使用方式

常用的flags:
- re.I re.IGNORECASE 大小写无关匹配
- re.S re.DOTALL .可匹配换行符
- re.M re.MULTILINE 多行匹配，即^ $可匹配行首和行尾
- re.U和re.A 预定义字符集所对应的字符集

- 直接调用re模块的方法
- 首先compile得到模式对象，然后调用该对象的方法

```
regexp = re.compile(pattern, flags)
```

- 传递正则表达式参数，由于正则表达式也使用\作为转义。建议采用原始字符串，原始字符串的内容就是要写的正则表达式
- 查找模式：search方法在text中查找第一个匹配的模式，返回match对象 。match方法判别text最开始是否为对应的模式，如果是，返回match对象
- regexp.finditer(text): 返回一个迭代器，其元素为所匹配的match对象
- re.search(pattern, text, flags) re.match(pattern, text, flags)

regexp.search(text, pos=...., endpos=....)

```
for match in regexp.finditer(text):
    print(match.group(1))
```

```
regexp = re.compile(r'\b((\d{4})/(\d{1,2})/(\d{1,2}))\b')
pos = 0
while True:
    match = regexp.search(text, pos)
    if not match:
        break
    print(match.group(1))
    pos = match.end( )
```

正则表达式使用方式

re.findall(pattern, text, flags)

regexp.findall(text)

- 返回一个列表，该列表中的每个元素描述了某次匹配的内容
  - 如果模式中有分组符，即包含子模式，则
    - 匹配的内容仅仅包含了子模式
    - 如果有多个子模式，则用元组方式组织
    - 注意不包括模式0，如果想要，则引入分组来描述整个模式，编号也相应改变
  - 如果模式中没有分组符，则包含整个匹配的模式

```
dates = re.findall('\b((\d{4})/(\d{1,2})/(\d{1,2}))\b', text)
dates = [date[0] for date in dates]
```

正则表达式使用方式

re.split(pattern, text, maxsplit=0, flags=0)

regexp.split(text, maxsplit=0)

- 查找模式作为分隔符来分割字符串

```
text = input('...')
words = re.split(r'[\s,;/]+', text)
# words = [word for word in words if word]
```

re.sub(pattern, new, text, count=0, flags=0)

- regexp.sub(pattern, new, count=0, flags=0)
- 查找模式将匹配的内容替换，返回替换后的字符串
- count为最大替换次数，缺省为全部替换
- 注意方法名不是replace

```
text = input('...')
text = re.sub(r'[\s,;/]+', ' ',  text)
words = text.split( )
```

- def func(...)：生成函数对象，赋值给func
- **实参与形参的定义和匹配**
  - 形参：位置形参、缺省值形参、可变长度形参、仅关键字传递的形参
  - 缺省值形参: 函数调用时不传递时使用缺省值
  - 实参：位置实参、关键字实参
  - 顺序：都是位置形式的参数在前
  - 首先匹配位置实参，然后是关键字实参，接下来剩下的位置实参收集到可变长度位置形参tuple，剩下的关键字实参收集到可变长度dict形参，最后是缺省值形参
- 函数调用采用pass by assignment的传递机制： 形参=实参
  - 形参指向实参所指的同一个对象
  - 缺省值形参：指向保存在函数对象的__defaults__属性中的同一对象
- return语句，没有时相当于 return None
- 函数体的第一行为字符串表达式时会被保存在函数对象的__doc__属性中

第5章 函数设计与使用：参数传递的序列解包

- func(*seq,**dict)：调用时实参进行序列解包：
- *seq：seq的元素展开为位置实参
- **dict：dict的元素变为关键字实参，key1 = value1, key2=value2

- **掌握函数定义和函数调用的方法**
- **了解实参与形参的定义和匹配方法**
- **掌握函数调用的序列解包**
- **了解函数作用域的LGB规则以及global语句**

第5章 函数设计与使用：变量作用域

- **命名空间（namespace）**是名字（变量）和对象的映射，python采用dict来实现
- 全局命名空间指的是在模块内部定义但是不在函数内部定义的名字空间
- 局部命名空间指的是在函数内部定义的名字空间
  - 允许嵌套函数，但不作要求
- 一个变量（名字）如何加入到某个名字空间?
  - 只有定义（赋值）、global和nonlocal（不作要求）才会建立或者改变变量的作用域(namespace)
    - 如果在函数内有赋值，则该名字属于本地名字空间。
      - 除非通过global var声明其为全局空间的名字
    - 如果在函数外赋值，则该名字属于全局名字空间

第5章 函数设计与使用：变量作用域

- 名字与对象的绑定采用late binding，即调用时才绑定
  - 非全局的自由变量在外层函数返回时绑定，不作要求
- 名字的命名空间（作用域）搜索： LEGB规则
  - 掌握LGB规则就可以，即首先搜索函数内部的本地命名空间，如果没有则搜索全局的命名空间，最后搜索内置名字空间
  - 在访问某个名字时，根据语句所在的位置决定查找的起始点，但是：
    - 如果显式说明为global，则仅仅在全局名字空间查找
    - 如果在函数内部，有赋值语句(不管在前面还是后面)，则只在local查找

第7章：文件操作：打开和关闭

- 掌握内置函数open，打开模式mode
- 掌握使用with语句，保证文件对象关闭
- 掌握使用read/readline/readlines以及文件对象本身作为迭代器时，对于文件读的方法
- 掌握使用write写文件，了解writelines方法
- 掌握二进制文件和文本文件的区别
- 了解json/pickle等模块的功能

- f = open(file, mode='r', encoding=None)
- file：如果有目录部分时可采用raw string
- **mode:** 不包括时缺省为打开文本文件读
  - rwxa: 取其中一种模式
  - tb+: 可选的
    - tb：文本和二进制模式
    - +: 读写模式，必须要指定rwxa中任意一种
- f.close(): 关闭文件释放资源
- 文件操作的三种模式：
  - open + process + close
  - open + try: process + finally： + close
  - with open as handler: + process，系统自动close释放资源

第7章：文件操作：读写

- **read、readline和readlines的区别：**
  - read指定长度或者全部读取到str
  - readline: 读取一行到str（换行符包括在内）
  - readlines：读取所有行(换行符包含在内），返回字符串列表
- write、writelines
  - write： 写字符串
  - writelines: 字符串列表中的每个字符串写入，注意中间不会添加换行符

```
# 一次读完，直接返回行列表
with open('text.txt', 'r') as f:
    lines = f.readlines()
```

```
# 一行一行地读
with open('text.txt', 'r') as f:
    while True:
        line = f.readline()
        if not line:  # 文件结尾
            break
        ....
```

```
# 文件对象为迭代器对象
with open('text.txt', 'r') as f:
    for line in f:  #文件对象为迭代器对象
        ....
```

```
# 一次全部读完，然后根据需要可以再分割为行进行处理
with open('text.txt', 'r') as f:
    text = f.read()
    lines = text.splitlines()
    # lines = text.split('\n') 稍有区别，但建议用splitlines()
```

第8章 异常处理结构与程序调试

- 异常：运行时错误，导致程序运行出错而跳出正常控制流
- 异常处理：捕获异常，进行相应的处理
- 常用的异常类：
  - StopIteration
  - ZeroDivisionError：除数为0
  - NameError
  - SyntaxError
  - IndexError
  - ValueError
  - KeyError
  - AssertionError
  - EOFError: 输入结束时触发

- 了解什么是异常
- 掌握异常处理中异常匹配的方法，Exception是一般情况下用户捕获的异常的大类
- 掌握使用raise子句抛出指定类型和原因的异常
- 掌握异常处理中没有异常，出现异常但是捕获、出现异常但没有捕获时的代码段执行轨迹
- 了解finally部分出现return/break等语句会取消异常的情况
- 掌握使用异常处理提高程序鲁棒性的方法，比如要求用户输入整数或浮点数
- 掌握with语句和文件打开结合的方法
- 了解assert语句
- 了解traceback模块

异常处理结构

- 了解出现异常时哪些代码块会执行
- finally: 不管有否异常都要执行
- else: 前面没有异常出现时执行
- except: 按照顺序匹配，如果匹配，则执行，后面的except跳过
  - except: 等价于except BaseException:
- 如果出现异常但是未捕获，则try语句后面的代码不会执行

```
try:
    <body>
finally:
    <finallyBody>
```

```
try:
    <body>
except [expression [as identifier]]:    # 至少一次，在多次出现时不带表达式except应该是最后一个。按序匹配找到对应的exception为止
    <exceptionBody>
else:    # 可选的，在没有异常出现时执行
    <elseBody>
finally:   # 可选的，不管异常有没有都要执行
    <finallyBody>
```

异常处理总结

- **可以通过raise子句来抛出异常**
- 异常处理不会引入新的namespace
- except子句用于捕获指定的异常，按照顺序匹配，一旦匹配后面的except子句跳过。except后面可以跟异常类型或者多个异常类型组成的元组。except Exception或except应该是最后一个except子句
- except子句可以通过 as instance来获得异常对象，但是注意except子句退出后instance不再有效
- 可选的else子句在没有异常时执行
- 可选的finally子句不管有没有异常都会执行
- 如果没有捕获异常，则该异常会往try外层抛出
  - 在finally字句中有return或者break时会取消要抛出的异常！

```
def f():
    try:
        t = 4
        print(4/0)
        return 4
    except Exception as e:
        print(e)
        return 1
    finally:
        print('done')
        return 0
print(f())
```

第8章 异常处理结构与程序调试：断言与上下文

- 断言：assert

```
assert expression[,reason]
```

- 上下文管理的使用：文件输入输出

```
if __debug__:
    if not expression:
        raise AssertionError
      # raise AssertionError(reason)
```

```
def read_file_with(file='sample.txt'):
    with open(file) as f:
        for line in f:
            print(line,end='')
```