1 print与转义符

2021年7月6日　　19:20

1. print函数
    a. 输出结果
        i. 字符串
        ii. 数字
        iii. 含有数字和运算符表达式的结果

```
print("Hello World")
print('Hello world')
print(50)
print(10+2)
```

    b. 输出结果
        i. 输出在控制台
        ii. 输出在文件

操作同c语言

```
fp=open("read.txt","a+")
print("Hello World",file=fp)
fp.close()
```

    c. 单行输入，用逗号隔开多个输入

```
print("Hello","World")
```

2. 转义字符 \
    a. \n 换行符
    b. \t 制表符，相当于四个字符，不足四个字符自动填充空格为四个空格
    c. \r 回车，退位回行首
    d. \b 退一位字符
    e. 对特殊字符进行转义，如\\代表反斜杠，\“代表输入字符
    f. 原字符：不希望字符串中转义字符起作用

```
print(r"Hello\nWorld")
```

    g. 注意字符串结尾不能是单个转义符

2 变量与保留字

2021年7月8日 18:28

1. 保留字

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

2. 标识符：变量、函数、类、模块和其他模块的名字，规则类似于c语言
   a. 字母数字下划线
   b. 不以数字开头
   c. 不能为保留字
   d. 区分大小写
3. 变量的定义与使用
   a. 变量的含义：可以赋给值的标签
   b. 变量三部分构成
      i. 标识：对象所存储的内存地址，使用内置函数id(boj)获取
      ii. 类型：表示的是对象的数据类型。使用内置函数type(obj)获取
      iii. 值：变量所存储的内容

```
name="Hello World"
print("值",name)
print("类型",type(name))
print("地址",id(name))
输出结果为
值 Hello World
类型 <class 'str'>
地址 2843130223344
```

   c. 变量多次赋值：多次赋值后，变量名会指向新的内存空间

```
name="Hello World"
print("值",name)
print("类型",type(name))
print("地址",id(name))
name="Hello World2"
print("值",name)
print("类型",type(name))
print("地址",id(name))
输出结果为
值 Hello World
类型 <class 'str'>
地址 1765221524144
值 Hello World2
类型 <class 'str'>
地址 1765221489392
```

4. 数据类型
    a. 常见的类型
        i. 整数类型：int->2,3,4
        ii. 浮点数类型：float->2.33,50.72415
        iii. 布尔类型：bool->True,Flase
        iv. 字符串类型：str->Hello world
    b. 整数类型，int
        i. 不同的表示方法
            1) 十进制：默认的表示方式
            2) 二进制：0b开头
            3) 八进制：0o开头
            4) 十六进制：0x开头
    c. 浮点数类型，float
        i. 浮点数存储不精确

```
print(3.3)
print(1.1+2.2)
输出结果为
3.3
3.3000000000000003
```

        ii. 解决方案：导入decimal模块

```
from decimal import Decimal
print(3.3)
print(1.1+2.2)
print(Decimal("1.1")+Decimal("2.2"))
输出结果为
3.3
3.3000000000000003
3.3
```

    d. bool类型
        i. True/False可以转为整型的0或1
    e. 字符串类型
        i. 称为不可变的字符序列
        ii. 可以使用单引号，双引号，三引号定义
        iii. 单双引号只能在一行内输入字符串，三引号可以分布在多行

```
str1='Hello World'
str2="Hello World"
str3='''Hello
World'''
print(str1)
print(str2)
print(str3)
输出结果为
Hello World
Hello World
Hello
World
```

        iv. 格式化字符串
            1) upper()函数使得字符串全部大写

2) lower()函数使得字符串全部小写
3) title()函数使字符串首字母大写

v. f字符串

1) 作用：在字符串中插入变量

```
first_name="ada"
last_name="lovelace"
full_name=f"{first_name.title()} {last_name}"
print(full_name)
输出结果为
Ada lovelace
```

vi. 删除空白

1) rstrip()可以去除字符串结尾的空白
2) lstrip()可以去除字符串开头的空白
3) strip()可以去除字符串开头或者结尾的空白

5. 数据类型转换

a. 目的：将不同数据类型的数据拼接在一起
b. 通常为str，int，float之间转换
c. 转换函数有str()、int()、float()
d. str()：将其他类型转化为字符串类型

```
n1=22
n2=22.7
n3=True
print(type(n1),type(n2),type(n3))
print(str(n1),str(n2),str(n3))
print(type(str(n1)),type(str(n2)),type(str(n3)))
输出结果为
<class 'int'> <class 'float'> <class 'bool'>
22 22.7 True
<class 'str'> <class 'str'> <class 'str'>
```

e. Int():仅适用于float类型，整数字符串，bool类型

```
n1="123"
n2=123.3
n3="123.3"
n4=True
n5="Hello"
print(type(n1),type(n2),type(n3),type(n4),type(n5))
print(int(n1),type(int(n1)))
print(int(n2),type(int(n2)))
#print(int(n3),type(int(n3)))
print(int(n4),type(int(n4)))
#print(int(n5),type(int(n5)))
输出结果为
<class 'str'> <class 'float'> <class 'str'> <class 'bool'> <class 'str'>
123 <class 'int'>
123 <class 'int'>
1 <class 'int'>
```

f. float():仅适用于int类型，数字字符串，bool类型

```
n1="123.3"
n2=123
n3="123"
n4=True
n5="Hello"
```

```
print(type(n1),type(n2),type(n3),type(n4),type(n5))
print(float(n1),type(float(n1)))
print(float(n2),type(float(n2)))
print(float(n3),type(float(n3)))
print(float(n4),type(float(n4)))
# print(float(n5),type(float(n5)))
```

输出结果为

```
123.3 <class 'float'>
123.0 <class 'float'>
123.0 <class 'float'>
1.0 <class 'float'>
```

6. 注释
   a. 单行注释：以#表示
   b. 多行注释：三引号之间的代码成为多行注释
7. 进制转换

	2进制	8进制	10进制	16进制
2进制		bin(int(n,8))	bin(int(n,10))	bin(int(n,16))
8进制	oct(int(n,2))		oct(int(n,10))	oct(int(n,16))
10进制	int(n,2)	int(n,8)		int(n,16)
16进制	hex(int(n,2))	hex(int(n,8))	hex(int(n,10))	

转换过程可以分为两步，第一步是将输入的字符串转换成十进制类型，int()函数，第一个参数是待转换的字符串，第二个参数是当前数字的进制。第二步时将转化出的十进制数字通过类型转化函数转化成对应的进制，返回值是字符串。

3 运算符

2021年7月9日　　13:25

1. Input()
    a. 基本用法

```
name=input('请输入')
```

    赋值符号前为变量名

    赋值符号后为输入函数，括号内为输入提示

    输入结果保留为字符串类型

    b. 进阶用法

    对input()的输出结果进行类型转化，进而得到自己希望的结果

```
num1=int(input('请输入第一个数字'))
num2=int(input("请输入第二个数字"))
print(num1+num2)
输出结果为
请输入第一个数字10
请输入第二个数字20
30
```

2. 运算符
    a. 算数运算符

+	-	*	/	//	%	**
加	减	乘	除	整除	取余	幂运算

    ○ 一正一负整除，向下取整

```
print(9//-2)
输出结果为
-5
```

    ○ 一正一负取余，公式为：余数=被除数-除数*商

```
print(9%-4)
print(-9%4)
输出结果为
-3
3
```

    b. 赋值运算符
        i. 执行顺序为从右向左
        ii. 可以链式赋值，每个变量指向的地址相同

```
a=b=c=20
print(a,id(a))
print(b,id(b))
print(c,id(c))
输出结果为
20 1541380205456
20 1541380205456
```

```
20 1541380205456
```

iii. 支持参数赋值

+=	-=	*=	/=	//=

iv. 支持系列解包赋值，两边的变量和值应相同

```
a,b,c=10,20,30
print(a,b,c)
输出结果为
10 20 30
```

交换示例

```
a,b,c=10,20,30
a,b,c=c,a,b
print(a,b,c)
输出结果为
30 10 20
```

c. 比较运算符：返回值为bool类型

>	<	>=	<=	!=	==

- 比较两个变量的标识即id用 is

```
a=10
b=10
print(a==b)
print(a is b)
输出结果为
True
True
```

d. bool运算符:返回值为bool类型

and	or	not	in	not in

- In/not in可用于判断字符在字符串中是否存在

e. 位运算符：把数据转成二进制进行运算

i. 按位与：&

两个比较值二进制位上都为1时，对应的二进制返回值为1，否则为1

ii. 按位或：|

两个比较值二进制位上都为0时，对于的二进制返回值为0.否则为1

iii. 按位取反：~

将二进制每一位取反

iv. 按位异或：^

两个比较值二进制位上不同则对应的二进制返回值为1，否则为0

v. 左移位：<<

将二进制左移一位，高位溢出，空位补0

常见用法为倍乘

vi. 右移位：>>

将二进制右移一位，低位溢出，空位补0

常见用法为倍除

```
print(4<<1)
print(4<<2)
```

```
print(4>>1)
print(4>>2)
输出结果为
8
16
2
1
```

3. 运算符优先级

乘方	**					
按位取反	~					
符号运算符	+	-				
乘除	*	/	//	%		
加减	+	-				
位移	<<	>>				
按位与	&					
按位异或	^					
按位或	|					
比较运算符	>	<	<=	>=	==	!=
is运算符	is	Is not				
in运算符	in	not in				
逻辑非	not					
逻辑与	and					
逻辑或	or					
逗号运算符	,					

4 分支结构

2021年7月9日　23:50

1. 顺序结构：程序从上到下顺序的执行
2. 分支结构
    a. 对象的bool值
        i. python一切皆对象，所有对象都有一个bool值
        ii. 用bool()获取对象的bool值
        iii. 下列对象的bool值为False

False	数值0	空字符串	空列表	空元组	空字典	空集合

    b. 选择结构：根据条件的bool值选择性地执行部分代码
        i. 单分支结构

If 条件表达式:
　　条件执行体

```
money=1000
a=int(input("请输入取款金额:"))
if money>=a:
    money-=a
    print("取款成功，余额为",money)
输出结果为
请输入取款金额:100
取款成功，余额为 900
```

        ii. 双分支结构

If 条件表达式1:
　　条件执行体1
else:
　　条件执行体2

```
a=int(input("请输入一个整数:"))
if a%2==0:
    print(a,"为偶数")
else:
    print(a,"为奇数")
输出结果为
请输入一个整数:5
5 为奇数
```

        iii. 多分支结构

If 条件表达式1:
　　条件执行体1
elif:
　　条件执行体2
elif:
　　条件执行体3
else:
　　条件执行体4(可选)

```
a=int(input("请输入分数:"))
if a>=90:
    print("等第为A")
elif a>=80:
```

```
    print("等第为B")
elif a>=70:
    print("等第为C")
elif a>=60:
    print("等第为D")
else:
    print("等第为E")
输出结果为
请输入分数:69
等第为D
```

iv. 嵌套if

注意缩进

If 条件表达式1:

    If 条件表达式2:

        条件执行体1

    else:

        条件执行体2

else:

    条件执行体3

v. 条件表达式

- 是双分支结构的缩写，格式为x if 判断条件 else y（有些类似c语言的三元表达式）

```
num_a=int(input("请输入第一个整数"))
num_b=int(input("请输入第二个整数"))
print(num_a if num_a>num_b else num_b,"更大")
输出结果为
请输入第一个整数10
请输入第二个整数20
20 更大
```

vi. pass语句

1) 语句什么都不做，只是一个占位符，用在语法上需要语句的地方
2) 什么时候使用：先搭建语法结构，还没有想好代码怎么写
3) 和什么语句一起使用：
    a) if语句的条件执行体
    b) for-in语句的循环体
    c) 定义函数时的函数体

```
a=int(input("请输入分数:"))
if a>=90:
    pass
elif a>=80:
    pass
elif a>=70:
    pass
elif a>=60:
    pass
else:
    pass
```

这种情况下pass取代了条件表达式中的执行语句，执行代码并不会报错，也不会有输出结果，可以用于构架整体框架。

5 循环结构

2021年7月11日　　22:20

1. range()函数
    a. 创建range对象的三种方式
        i. range(stop):创建一个(0,stop)之间的整数序列，步长为1
        ii. range(start,stop):创建一个(start,stop)之间的整数序列，步长为1
        iii. range(start,stop,step):创建一个(start,stop)之间的整数序列，步长为start
    b. 返回值为一个迭代器

```
a=range(10)
print(list(a))
b=range(1,10)
print(list(b))
c=range(0,10,2)
print(list(c))
输出结果为
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
```

    c. 使用in或者not in可以判断某种整数是否在整数数列中
    d. range对象的优点：range类型的占用空间是相同的，只需要存储start，stop，step三个元素
2. 循环结构
    a. while循环
        i. 语法结构为

            while 条件表达式：
                循环语句

        ii. 初始化变量->条件判断->条件执行体->改变变量

```
#计算累加和
num=int(input("请输入一个正整数"))
sum=0
a=0
while a<=num:
    sum+=a
    a+=1
print("累加和为:",sum)
输出结果为
请输入一个正整数10
累加和为: 55
```

```
#计算一到100间的偶数和
a=0
sum=0
while a<=100:
    if a%2==0:
        sum+=a
    a+=1
print("1到100间偶数和为: ",sum)
```

```
输出结果为
1到100间偶数和为： 2550
```

b. for-in循环

i. in表达从(字符串、序列等)中依次取值，成为遍历

ii. for-in遍历的对象必须是可迭代对象

iii. 语法结构为

for 自定义的变量 in 可迭代的对象：

循环体

iv. 如果循环体变量在循环过程中不需要用到，可以用下划线_代替

```
#计算1到100间偶数和
sum=0
for i in range(1,101):
    if i%2==0:
        sum+=i
print("1到100间偶数和为:",sum)
输出结果为
1到100间偶数和为: 2550
```

```
#生成100到999间的水仙花数
num=0
for i in range(100,1000):
    if (i%10)**3+(i//10%10)**3+(i//100)**3==i:
        print(i)
        num+=1
print("100到999间水仙花数个数有:",num)
输出结果为
153
370
371
407
100到999间水仙花数个数有: 4
```

c. 流程控制语句break

用于结束循环结构，通常与分支结构if一起使用

d. 流程控制语句continue

用于结束当前这一次循环结构，进入下一次循环结构，通常与分支结构if使用

```
for item in range(1,51):
    if item%5!=0:
        continue
    print(item)
输出结果为
5
10
15
20
25
30
35
40
45
```

```
50
```

e. else 语句

i. 还可以与while和for-in循环使用

ii. 当循环不是由break结束时执行else中的语句

```
for _ in range(3):
    keyword=input("请输入密码")
    if(keyword=="8888"):
        print("密码正确")
        break
    else:
        print("密码错误")
else:
    print("三次密码输入均错误")
输出结果
请输入密码777
密码错误
请输入密码7777
密码错误
请输入密码6767
密码错误
三次密码输入均错误
```

f. 嵌套循环

i. 循环结构中又嵌套了另外的完整的循环，其中内层循环作为外层循环的循环体语句执行

```
#输出一个三行四列的矩阵
for i in range(1,4):
    for j in range(1,5):
        print(4*(i-1)+j,end="\t")
    print("")
输出结果为
1    2    3    4
5    6    7    8
9    10    11    12
```

```
#输出一个三角形
for i in range(1,11):
    for j in range(i):
        print("*",end="")
    print("")
输出结果为
*
**
***
****
*****
******
*******
********
*********
**********
```

end语句用于输出不换行，可将其视为把默认的\n替换成=后的内容

ii. 二重循环中的break和continue只用于控制本层循环

6 列表

2021年7月12日 14:41

1. 相当于其他语言中的数组
2. 列表的创建
    a. 使用中括号
    b. 使用内置函数list()
3. 列表的特点
    a. 列表元素按顺序有序排序
    b. 索引映射唯一一个数据

索引	0	1	2	3	4	5	6
数据	"Hell"	99	62	12	23.3	16	"world"
索引	-7	-6	-5	-4	-3	-2	-1

    c. 可以存储重复数据
    d. 任意数据类型混存
    e. 根据需要动态分配和回收内存
4. 列表的查询
    a. 获取列表中指定元素的索引：index()函数
        i. 如果列表中有重复元素，只返回第一个元素的索引
        ii. 如果列表中不存在查找元素，会报错
        iii. 可以指定区域范围内查找元素

```
lst=["hello",97,"world","hello"]
print(lst.index("hello"))
print(lst.index("hello",1,4))
输出结果为
0
3
```

    b. 获取列表中的单个元素
        i. 正向索引：0到n-1
        ii. 逆向索引：-n到-1
        iii. 指定索引不存在，会报错
    c. 获取列表中的多个元素：切片
        i. 语法格式：列表名[start:stop:step]
        ii. 切片的结果：原列表的片段的拷贝
        iii. 范围左闭右开
        iv. 不写step，step默认为1；不写start，start默认为0；不写stop，stop默认为n
        v. 若step为-1，逆序切片，start>stop

```
lst=["hello",97,"world","hello",12,32,14]
lst2=lst[1:4:1]
lst3=lst[1:4]
```

```
lst4=lst[1:4:]
lst5=lst[:6:2]
lst6=lst[1::2]
lst7=lst[::-1]
lst8=lst[6:0:-1]
print(lst)
print(lst2)
print(lst3)
print(lst4)
print(lst5)
print(lst6)
print(lst7)
print(lst8)
```

输出结果为

```
['hello', 97, 'world', 'hello', 12, 32, 14]
[97, 'world', 'hello']
[97, 'world', 'hello']
[97, 'world', 'hello']
['hello', 'world', 12]
[97, 'hello', 32]
[14, 32, 12, 'hello', 'world', 97, 'hello']
[14, 32, 12, 'hello', 'world', 97]
```

d. 判断指定元素在列表中是否存在

元素 in 列表名

元素 not in 列表名

5. 列表元素的增加
   a. append()在列表结尾添加一个元素，没有产生新的列表，如果使用append添加一个列表，列表会作为原列表的一个元素添加，不产生新的列表
   b. extend()在列表的末尾至少添加一个元素，如果使用extend添加一个列表，会将列表中的每个元素添加到原列表中，不产生新的列表
   c. Insert()在任意位置上添加元素，不产生新的列表
   d. 切片在任何位置添加至少一个元素，不产生新的列表

```
lst=["hello",97,"world","hello",12,32,14]
print("添加元素前:",lst,id(lst))
lst.append(100)
print("添加元素后:",lst,id(lst))
lst2=[10,100]
```

输出结果为

```
添加元素前: ['hello', 97, 'world', 'hello', 12, 32, 14] 1805395385408
添加元素后: ['hello', 97, 'world', 'hello', 12, 32, 14, 100] 1805395385408
```

```
print("----------------------------
extend-------------------------------------")
lst.append(lst2)
print(lst)
lst.extend(lst2)
print(lst)
```

输出结果为

```
--------------------------extend----------------------------------
['hello', 97, 'world', 'hello', 12, 32, 14, 100, [10, 100]]
['hello', 97, 'world', 'hello', 12, 32, 14, 100, [10, 100], 10, 100]
```

```
print("--------------------------------
```

```
insert-------------------------------------")
print(lst)
lst.insert(1,29)
print(lst)
输出结果为
----------------------------insert---------------------------------
['hello', 97, 'world', 'hello', 12, 32, 14, 100, [10, 100], 10, 100]
['hello', 29, 97, 'world', 'hello', 12, 32, 14, 100, [10, 100], 10, 100]
```

```
print("-----------------------------------切
片---------------------------------")
lst3=[24,47,46]
print(lst)
lst[1:]=lst3
print(lst)
输出结果为
-------------------------------切片-------------------------------
['hello', 29, 97, 'world', 'hello', 12, 32, 14, 100, [10, 100], 10, 100]
['hello', 24, 47, 46]
```

6. 列表元素的删除
    a. remove()函数
        i. 删除指定元素，如果有重复元素，只会移除第一个
        ii. 如果不存在指定元素则会报错
    b. pop()函数
        i. 删除指定索引的元素
        ii. 如果指定索引不存在则会报错
        iii. 这个函数的返回值是弹出的元素，这让我们可以输出弹出值，
        iv. 如果不指定参数，则会删除列表最后一个元素
    c. 切片
        i. 一次至少删除一个元素
        ii. 将生成一个新的列表对象
        iii. 可以将切片的列表部分赋值成空列表，一次删除多个列表而不生成新列表
    d. clear()
        i. 清空列表
    e. del
        i. 删除列表，列表定义被清除，如果再次使用列表则会报错
        ii. 删除列表中指定索引的元素

```
print("-------------------------------
remove---------------------------------")
lst=[10,20,30,40,50,60,70,80,90,10]
print(lst)
lst.remove(10)
print(lst)
print("-------------------------------
clear----------------------------------------")
lst=[10,20,30,40,50,60,70,80,90,10]
print(lst)
lst.clear()
print(lst)
输出结果为
```

```
----------remove----------
[10, 20, 30, 40, 50, 60, 70, 80, 90, 10]
[20, 30, 40, 50, 60, 70, 80, 90, 10]
```

```
print("----------
pop----------")
lst=[10,20,30,40,50,60,70,80,90,10]
print(lst)
a=lst.pop(2)
print(lst)
print(a)
```

输出结果为

```
----------pop----------
[10, 20, 30, 40, 50, 60, 70, 80, 90, 10]
[10, 20, 40, 50, 60, 70, 80, 90, 10]
30
```

```
print("----------切
片----------")
lst=[10,20,30,40,50,60,70,80,90,10]
lst2=lst[1:3]
print(lst)
print(lst2)
lst[1:3]=[]
print(lst)
```

输出结果为

```
----------切片----------
[10, 20, 30, 40, 50, 60, 70, 80, 90, 10]
[20, 30]
[10, 40, 50, 60, 70, 80, 90, 10]
```

```
print("----------
clear----------")
lst=[10,20,30,40,50,60,70,80,90,10]
print(lst)
lst.clear()
print(lst)
```

输出结果为

```
----------clear----------
[10, 20, 30, 40, 50, 60, 70, 80, 90, 10]
[]
```

```
print("----------
del----------")
lst=[10,20,30,40,50,60,70,80,90,10]
print(lst)
del lst[1]
print(lst)
del lst
```

输出结果为

```
----------del----------
[10, 20, 30, 40, 50, 60, 70, 80, 90, 10]
[10, 30, 40, 50, 60, 70, 80, 90, 10
```

7. 列表元素的修改
   a. 指定索引元素的重新赋值
   b. 切片赋值

```
lst=[10,20,30,40,50,60]
```

```
print(lst)
lst[2]=25
print(lst)
lst[1:3]=[100,200,300,400,500,600]
print(lst)
```

输出结果为

```
[10, 20, 30, 40, 50, 60]
[10, 20, 25, 40, 50, 60]
[10, 100, 200, 300, 400, 500, 600, 40, 50, 60]
```

8. 列表元素的排序
   a. 调用sort()方法，对列表的元素做升序排序，可以指定，reverse=True，进行降序排序，不产生新的列表
   b. 调用内置函数sorted()码可以指定reserve=True，进行降序排序，原列表不发生改变，产生一个新的列表对象。
   c. reverse()函数可以反转函数，如car.reverse()可以在升序排序后让列表反转从而形成倒序

```
lst=[10,2,29,21,43,56,13,35,26,45,25,16]
print(lst,id(lst))
lst.sort()
print(lst,id(lst))
lst.sort(reverse=True)
print(lst,id(lst))
print("------------------------------------------------------------------
-------------")
lst=[10,2,29,21,43,56,13,35,26,45,25,16]
lst2=sorted(lst)
print(lst,id(lst))
print(lst2,id(lst2))
lst3=sorted(lst,reverse=True)
print(lst3,id(lst3))
```

输出结果为

```
[10, 2, 29, 21, 43, 56, 13, 35, 26, 45, 25, 16] 2035247700992
[2, 10, 13, 16, 21, 25, 26, 29, 35, 43, 45, 56] 2035247700992
[56, 45, 43, 35, 29, 26, 25, 21, 16, 13, 10, 2] 2035247700992
------------------------------------------------------------------------
[10, 2, 29, 21, 43, 56, 13, 35, 26, 45, 25, 16] 2035247739392
[2, 10, 13, 16, 21, 25, 26, 29, 35, 43, 45, 56] 2035247700992
[56, 45, 43, 35, 29, 26, 25, 21, 16, 13, 10, 2] 2035247739008
```

9. 列表生成式

   [与i有关的表达式 for i in range(1,10)]

```
lst=[i*2 for i in range(1,6)]
print(lst)
```

输出结果为

```
[2, 4, 6, 8, 10]
```

10. 数字列表相关函数
    a. sum() 和函数
    b. max()/min() 最大最小值函数
11. 复制列表

```
list1=[10,20,30,40,50,60]
list2=list1[:]
list3=list1
```

```
list1.append(70)
list2.append(80)
list3.append(90)
print(list1)
print(list2)
print(list3)
```

输出结果为

```
[10, 20, 30, 40, 50, 60, 70, 90]
[10, 20, 30, 40, 50, 60, 80]
[10, 20, 30, 40, 50, 60, 70, 90]
```

list3和list1指向的内容相同，改变一个，另一个也会改变

而list2则是复制了list1的内容而独立于list1

7 字典

2021年7月13日 13:41

1. 定义及特点:
   a. 数据结构之一，是一个可变序列
   b. 以键值对的方式存储数据，是一个无序的序列，列表是一个有序的序列
   c. 放在字典中的键是一个不可变序列(不可以进行增删改，例如字符串)
   d. key不允许重复(会出现值覆盖的现象)，value可以重复
   e. 字典可以根据需要动态的伸缩
   f. 字典会浪费比较大的内存，是一种使用空间换时间的数据结构
2. 实现原理:
   - 字典根据key查找value所在的位置
3. 字典的创建
   a. 大括号
   b. dict()函数

```
dict1={"color":"green","num":100,}
print(dict1)
print(type(dict1))
dict2=dict(color="red",num=90)
print(dict2)
print(type(dict2))
dict3={}
print(dict3)
print(type(dict3))
```

输出结果为

```
{'color': 'green', 'num': 100}
<class 'dict'>
{'color': 'red', 'num': 90}
<class 'dict'>
{}
<class 'dict'>
```

4. 字典元素的获取
   a. []
   b. get()方法

```
dict1={"color":"green","num":100,"name":"ada","age":13}
print(dict1["color"])
print(dict1.get("color"))
num2=dict1.get("num1")
print(num2)
print(type(num2))
```

输出结果为

```
green
green
None
<class 'NoneType'>
```

使用[]取元素，如果对应的键不存在，程序会报错

如果使用get()方法，对应的键不存在，会返回None,返回值的类型为NoneType

5. key的判断
    - 用in或者not in
6. 键值对的删除
    a. del
    b. clear()

```
dict1={"color":"green","num":100,"name":"ada","age":13}
print(dict1)
del dict1["color"]
print(dict1)
dict1.clear()
print(dict1)
输出结果为
{'color': 'green', 'num': 100, 'name': 'ada', 'age': 13}
{'num': 100, 'name': 'ada', 'age': 13}
{}
```

7. 键值对的增加和修改

```
dict1={"color":"green","num":100,"name":"ada","age":13}
print(dict1)
dict1["lastname"]="lover"
print(dict1)
dict1["color"]="yellow"
print(dict1)
输出结果
{'color': 'green', 'num': 100, 'name': 'ada', 'age': 13}
{'color': 'green', 'num': 100, 'name': 'ada', 'age': 13, 'lastname': 'lover'}
{'color': 'yellow', 'num': 100, 'name': 'ada', 'age': 13, 'lastname': 'lover'}
```

8. 获取字典视图的三个方法

keys()	获取字典中所有key
values()	获取字典中所有value
items()	获取字典中所有key，value对

```
dict1={"color":"green","num":100,"name":"ada","age":13}
keys=dict1.keys()
values=dict1.values()
items=dict1.items()
print(keys,type(keys))
print(values,type(values))
print(items,type(items))
输出结果为
dict_keys(['color', 'num', 'name', 'age']) <class 'dict_keys'>
dict_values(['green', 100, 'ada', 13]) <class 'dict_values'>
dict_items([('color', 'green'), ('num', 100), ('name', 'ada'), ('age', 13)]) <class 'dict_items'>
```

可以将其转换成由元组组成的列表

9. 字典元素的遍历

遍历所有key，通过调用key对	dict1={"color":"green","num":100,"name":"ada","age":13} for item in dict1: print(item,dict1[item],dict1.get(item)) 输出结果为

应的value来遍历整个字典	color green green num 100 100 name ada ada age 13 13
使用item()函数，使用两个循环变量遍历所有键值对 可以通过另外的keys()和values()函数分别遍历键和值	dict1={"color":"green","num":100,"name":"ada","age":13} for key,value in dict1.items(): print(key,value) 输出结果为 color green num 100 name ada age 13
按特定顺序遍历字典，使用sorted()函数	dict1={"color":"green","num":100,"name":"ada","age":13} for key in sorted(dict1.keys()): print(key) 输出结果为 age 13 color green name ada num 100

10. 字典的嵌套
    a. 字典列表：将一系列字典储存在一个列表中
    b. 字典中存储列表：将列表作为字典的值
    c. 字典中存储字典：将字典作为字典的值
11. 字典生成式
    a. 内置函数zip()：用于将可迭代的对象作为参数，将对象中对应的元素打包成一个元组，然后返回由这些元组组成的列表，如果长度不同则会按照较短的生成

```
keys=["name","age","weight","height"]
values=["mike",20,65,180]
dict1={key:value for key,value in zip(keys,values)}
print(dict1)
keys=["name","age","weight","height"]
values=["mike",20,65,175,180]
dict2={key:value for key,value in zip(keys,values)}
print(dict2)
输出结果为
{'name': 'mike', 'age': 20, 'weight': 65, 'height': 180}
{'name': 'mike', 'age': 20, 'weight': 65, 'height': 175}
```

8 元组与集合

2021年7月14日 14:02

1. 元组
   a. 特点：

      python内置的数据结构之一，是一个不可变序列(没有增删改操作)

   b. 创建
      i. 小括号(小括号可以省略)
      ii. 内置函数tuple()
      iii. 如果元组的元素只包含一个元素，则需要使用逗号和小括号

```
t=(10,20,30,"cc98","zju")
print(t)
print(type(t))
t1=10,22,30,"cc98","zju"
print(t1)
print(type(t1))
t2=tuple((10,23,30,"cc98","zju"))
print(t2)
print(type(t2))
t3=(10,)
print(t3)
print(type(t3))
t4=()
print(t4)
print(type(t4))
```

输出结果为

```
(10, 20, 30, 'cc98', 'zju')
<class 'tuple'>
(10, 22, 30, 'cc98', 'zju')
<class 'tuple'>
(10, 23, 30, 'cc98', 'zju')
<class 'tuple'>
(10,)
<class 'tuple'>
()
<class 'tuple'>
```

   c. 为什么元组是不可变序列
      i. 多任务环境下，同时操作对象不需要加锁
      ii. 注意事项
         1) 如果元组中对象本身是不可变对象，则不能再引用其他对象
         2) 如果元组中的对象是可变对象，则可变对象的引用不允许改变，但数据可以改变
         3) 通俗来说，元组中元素的地址不能改变，而不可变对象的值改变的同时地址也会改变，所有不能更改不可变对象。而类似列表的可变对象，即使对列表进行增删改，也不会改变列表的地址，因此可以对其进行改变。

   d. 遍历
      i. 通过索引
      ii. 通过 for in循环进行遍历

2. 集合
   a. 特点
      i. 内置的数据结构，是可变类型的序列

ii. 没有value的字典

b. 创建

i. {}

ii. 内置函数set()将括号内的序列、列表、元组、字符串、集合转换成集合

```
s={10,20,30,"zju","cc98"}
print(s,type(s))

s1=set(range(6))
print(s1,type(s1))

s2=set([10,20,24,"cc98","zju"])
print(s2,type(s2))

s3=set((10,20,"ZJU","CC98"))
print(s3,type(s3))

s4=set("Python")
print(s4,type(s4))

s5=set()
print(s5,type(s5))

s6={}
print(s6,type(s6))
```

数据结构

```
{'cc98', 'zju', 20, 10, 30} <class 'set'>
{0, 1, 2, 3, 4, 5} <class 'set'>
{10, 'cc98', 'zju', 20, 24} <class 'set'>
{10, 'CC98', 20, 'ZJU'} <class 'set'>
{'y', 'h', 't', 'n', 'P', 'o'} <class 'set'>
set() <class 'set'>
{} <class 'dict'>
```

c. 判断

in或者not in

d. 增加

i. add()方法，一次添加一个元素

ii. update()方法，一次至少添加一个元素,类似于set()函数，会把其中的字符串拆成一个个字符（如果需要添加字符串，可以使用add()方法或者将字符串放在列表或者元组中添加。

```
s={10,20,30,"zju","cc98"}
print(s)
s.add("cs")
print(s)
s.update(["cse","ai","ml"])
print(s)
s.update("ads")
print(s)
```

输出结果为

```
{'zju', 20, 10, 'cc98', 30}
{'zju', 20, 'cs', 10, 'cc98', 30}
{10, 'cc98', 20, 'cs', 'cse', 30, 'ml', 'ai', 'zju'}
{10, 'cc98', 'd', 's', 20, 'a', 'cs', 'cse', 30, 'ml', 'ai', 'zju'}
```

e. 删除

i. remove()方法，一次移除一个元素，如果元素不存在则报错

ii. discard()方法，一次移除一个元素，如果元素不存在不会报错

iii. pop()方法，随机移除一个元素

iv. clear()方法，清空

```
s={10,20,30,"zju","cc98"}
print(s)

s.remove(10)
print(s)

s.discard(20)
print(s)

s.discard(40)
print(s)

s.pop()
print(s)

s.clear()
print(s)
输出结果为
{'zju', 20, 10, 'cc98', 30}
{'zju', 20, 'cc98', 30}
{'zju', 'cc98', 30}
{'zju', 'cc98', 30}
{'cc98', 30}
set()
```

f. 集合间的关系

i. 可以通过==或者!=判断两个集合是否相等

ii. 可以判断一个集合是否是另外一个集合的子集，使用issubset()方法

iii. 可以判断一个集合是否是另外一个集合的超集，使用issuperset()方法

iv. 两个集合是否有交集，使用isdisjoint()方法(没有交际为True)

```
s={10,20,30,"zju","cc98"}
s1={"zju","cc98"}
s2={10,15}

print(s1.issubset(s))
print(s2.issubset(s))

print(s.issuperset(s1))
print(s.issuperset(s2))

print(s.isdisjoint(s1))
print(s.isdisjoint(s2))
输出结果为
True
False
True
False
False
False
```

g. 集合的数学操作

i. 交集 intersection()方法或者&

ii. 并集 union()方法或者|

iii. 差集 difference()方法或者-

iv. 对称差集 symmetric_difference()方法或者^

```
s1={10,20,30,40}
s2={30,40,50,60,70}
```

```
print(s1.intersection(s2))
print(s1 & s2)

print(s1.union(s2))
print(s1 | s2)

print(s1.difference(s2))
print(s1 - s2)

print(s1.symmetric_difference(s2))
print(s1 ^ s2)
```

输出结果为

```
{40, 30}
{40, 30}
{70, 40, 10, 50, 20, 60, 30}
{70, 40, 10, 50, 20, 60, 30}
{10, 20}
{10, 20}
{50, 20, 70, 10, 60}
{50, 20, 70, 10, 60}
```

h. 集合生成式

把列表生成式中的[]改成{}即可

3. 总结

数据结构	是否可变	是否重复	是否有序	定义符号
列表list	可变	可重复	有序	[]
元组tuple	不可变	可重复	有序	()
字典dict	可变	key不可重复，value可以重复	无序	{}
集合set	可变	不可重复	无序	{}

9 字符串

2021年7月15日 21:49

1. 特点
   a. 基本数据类型，不可变的字符序列
   b. 可使用单引号，双引号，三引号定义
2. 驻留机制
   a. 仅保存一份相同且不可变字符串的方法，不同的值被存放在字符串的驻留池中，python的驻留机制对相同的字符串只保留一份拷贝，后续创建相同字符串时，不会开辟新空间，而是把该字符串的抵制赋给新创建的变量

```
a='python'
b="python"
c='''python'''
print(a,id(a))
print(b,id(b))
print(c,id(c))
输出结果为
python 2362352248816
python 2362352248816
python 2362352248816
```

   b. 几种情况（交互模式），部分编译器会对驻留机制进行优化
      i. 字符串的长度是0或者1
      ii. 符合标识符的字符串（字母数字下划线）
      iii. 字符串只在编译时进行驻留，而非运行时
      iv. -5到255的整数
   c. 多个字符串拼接时最好使用join方法，避免创建多个字符串变量，效率更高
3. 查询

index()	查找子字符串第一次出现的位置，若不存在，则报错
rindex()	查找子字符串最后一次出现的位置，若不存在，则报错
find()	查找子字符串第一次出现的位置，若不存在，则返回-1
rfind()	查找子字符串最后一次出现的位置，若不存在，则返回-1

```
a='python,python'
print(a.index("py"))
print(a.rindex("py"))
print(a.find("py"))
print(a.rfind("py"))
print(a.find("java"))
print(a.rfind("java"))
输出结果为
0
7
```

```
0
7
-1
-1
```

4. 大小写转换（转换之后产生了一个新的字符串

upper()	所有字符全部大写
lower()	所有字符全部小写
swapcase()	大写字母变成小写字母，小写字母转成大写字母
capitalize()	第一个字符转换成大写，其余字符转换成小写
title()	把每个单词的第一个字符转换成大写，剩余字符变成小写

```
a='pyThon,pyThon'
print(a)
print(a.upper())
print(a.lower())
print(a.title())
print(a.swapcase())
print(a.capitalize())
输出结果为
pyThon,pyThon
PYTHON,PYTHON
python,python
Python,Python
PYtHON,PYtHON
Python,python
```

5. 内容对齐

center()	居中对齐，第一个参数指定宽度，第二个参数（可选，默认为空格）指定填充符，若宽度小于实际宽度则返回原字符串
ljust()	居中对齐，第一个参数指定宽度，第二个参数（可选，默认为空格）指定填充符，若宽度小于实际宽度则返回原字符串
rjust()	居中对齐，第一个参数指定宽度，第二个参数（可选，默认为空格）指定填充符，若宽度小于实际宽度则返回原字符串
zfill()	右对齐，左边用0填充，只接收一个参数，指定字符串的宽度，如果指定的宽度小于等于字符串的长度，返回原字符串

```
a='pyThon,pyThon'
print(a.center(20,"*"))
print(a.ljust(20,"*"))
print(a.rjust(20,"*"))
print(a.zfill(20))
输出结果为
***pyThon,pyThon****
pyThon,pyThon*******
*******pyThon,pyThon
0000000pyThon,pyThon
```

6. 劈分

split()	1.从字符串的左边开始劈分，默认的劈分字符是空格字符

	串，返回值是一个列表 2. 通过参数sep指定劈分字符串的劈分符 3. 通过参数maxsplit指定最大劈分次数，经历最大劈分之后，剩余的子串会单独作为一部分(注意字符数量是劈分次数加一)
rsplit()	1. 从字符串的右边开始劈分，默认的劈分字符是空格字符串，返回值是一个列表 2. 通过参数sep指定劈分字符串的劈分符 3. 通过参数maxsplit指定最大劈分次数，经历最大劈分之后，剩余的子串会单独作为一部分(注意字符数量是劈分次数加一)

```
a='python c java php html css'
b='python|c|java|php|html|css'
print(a.split())
print(b.split(sep='|'))
print(a.split(maxsplit=2))
print(b.split(sep='|',maxsplit=2))
print(a.rsplit())
print(b.rsplit(sep='|'))
print(a.rsplit(maxsplit=2))
print(b.rsplit(sep='|',maxsplit=2))
```

输出结果为

```
['python', 'c', 'java', 'php', 'html', 'css']
['python', 'c', 'java', 'php', 'html', 'css']
['python', 'c', 'java php html css']
['python', 'c', 'java|php|html|css']
['python', 'c', 'java', 'php', 'html', 'css']
['python', 'c', 'java', 'php', 'html', 'css']
['python c java php', 'html', 'css']
['python|c|java|php', 'html', 'css']
```

7. 判断字符串的组成

isidentifier()	判断指定的字符串是不是合法的标识符(字母，数字，下划线)
isspace()	判断字符串是否全部由空白字符组成(回车、换行、水平制表符等)
isalpha()	判断字符串是否全部由字符组成
isdeciaml()	判断字符串是否全部由十进制的数字组成
isnumeric()	判断指定的字符串是否全部由数字(包括中文数字和罗马数字)组成
isalnum()	判断字符串是否全部由字母和数字组成

```
print(1,"Hello".isidentifier())
print(2,"Hello_".isidentifier())
print(3,"Hello_123".isidentifier())

print(1," ".isspace())
```

```
print(2," \t".isspace())
print(3," \t\n".isspace())

print(1,"abc".isalpha())
print(2,"吴朝晖".isalpha())

print(1,"123".isdecimal())
print(2,"123.5".isdecimal())
print(3,"一二三".isdecimal())

print(1,"123".isnumeric())
print(2,"一二三".isnumeric())
print(3,"ⅠⅡⅢ".isnumeric())
```

输出结果为

```
1 True
2 True
3 True
1 True
2 True
3 True
1 True
2 True
1 True
2 False
3 False
1 True
2 True
3 True
```

8. 其他
   a. replace()替换，注意替换后生成新的字符串

```
a="Hello,Python"
print(a)
a=a.replace("Python","c++")
print(a)
b="Hello,Python,Python,Python"
print(b)
b=b.replace("Python","java",2)
print(b)
```

输出结果为

```
Hello,Python
Hello,c++
Hello,Python,Python,Python
Hello,java,java,Python
```

   b. join()将列表或者元组中的字符串合并成一个字符串

```
a=["ZJU","CC98","JAVA","C++"]
print("|".join(a))
b=("ZJU","CC98","JAVA","C++")
print("".join(b))
c="Elvenpath"
print("*".join(c))
```

输出结果为

```
ZJU|CC98|JAVA|C++
ZJUCC98JAVAC++
E*l*v*e*n*p*a*t*h
```

9. 比较
    a. 运算符即比较运算符
    b. 比较法则为比较字符串中的字符，如果第一个相同就比较第二个，一次下去
    c. 比较算法即用ord()函数调用字符原始值，即asc码，用chr()可以调出
    d. is 比较的是两个变量的内存地址
10. 切片
    a. 会产生新的对象
    b. 切片法则与列表切片相似
11. 格式化字符串
    a. %作为占位符，%s为字符串，%d或者%i为整数，%f为浮点数
    b. 使用{}
    c. 基本用法

```
a="ZJU"
b="CC98"
print("我在%s，我热爱%s"%(a,b))
print("我在{}，我热爱{}".format(a,b))
print(f"我在{a}，我热爱{b}")
输出结果为
我在ZJU，我热爱CC98
我在ZJU，我热爱CC98
我在ZJU，我热爱CC98
```

    d. %num1.num2d,num1代表宽度，num2表示保留位数
    e. {:.num}表示保留num个数字，{:.numf}表示保留num位小数，{:num1.num2f}表示保留num2位小数，宽度num1

10 函数

2021年7月19日　16:22

1. 定义：完成特定功能的一段代码
2. 创建与调用:def 函数名(输入参数):

   函数体

   [return xxx]

```
def sum(a,b):
    c=a+b
    return c
result=sum(10,20)
print(result)
输出结果为
30
```

3. 参数传递
   a. 上述的函数定义句中的a和b就是形参，即函数完成工作时需要的信息。而调用过程中的10和20是实际参数，即函数调用过程中传递给函数的信息
   b. 传递实参的方式
      i. 位置实参：根据实参的顺序传递信息
      ii. 关键字实参：将名称与值相关联

```
def sum(a,b):
    c=a+b
    return c
result1=sum(10,20)
print(result1)
result2=sum(b=20,a=10)
print(result2)
输出结果为
30
30
```

```
def fun(arg1,arg2):
    print(arg1)
    print(arg2)
    arg1=100
    arg2.append(10)
n1=10
n2=[10,20,30]
fun(n1,n2)
print(n1)
print(n2)
输出结果为
10
[10, 20, 30]
10
[10, 20, 30, 10]
```

函数解析：在函数的调用过程中，agr1指向了n1指向的值即10，但是在函数内部，arg1又被赋值，指向了100，因此对原n1没有影响。arg2指向了n2指向的列表，使用了append()操作，不产生新的列表，在原列表的基础上增加了一个值，有些类似c语言的指针传递。

如果函数传递的变量是不可变对象，则函数体内的修改不会影响实际参数的值，如果变量是可变对象，则会影响实际参数的值

如果希望传递例如列表之类的可变对象，又不希望函数改变这个列表的实际值，则可以选择6中提到的列表拷贝的方式

   c. 默认参数值，在函数定义的时候可以将某个形参赋值，如果函数调用过程中没有对这个形参赋值，则会调用默认值，这样可以使得某些函数在缺少部分参数时也可以直接调用，而不是报错

```
def sum(a,b=100):
```

```
    c=a+b
    return c
result1=sum(10)
print(result1)
result2=sum(a=20)
print(result2)
```
输出结果为
```
110
120
```

d. 个数可变的位置参数：传入的实参自动保存为元组

```
def sum(*a):
    print(a)
sum(10)
sum(10,20,30)
```
输出结果为
```
(10,)
(10, 20, 30)
```

e. 个数可变的关键字实参：传入的实参自动保存为字典

```
def sum(**a):
    print(a)
sum(a=10)
sum(b=10,c=20,d=30)
```
输出结果为
```
{'a': 10}
{'b': 10, 'c': 20, 'd': 30}
```

f. 函数的定义过程中，只能有一个个数可变的位置参数，只能有一个个数可变的关键字参数。如果二者皆有，则位置参数在前，关键字参数在后。

g. 将列表或者字典中的元素传递给函数的形参（不是传递列表或字典，而是内部的元素

```
def p(a,b,c):
    print("a=",a)
    print("b=",b)
    print("c=",c)
lst=[10,20,30]
dct={"a":40,"b":50,"c":60}
p(*lst)
p(**dct)
```
输出结果为
```
a= 10
b= 20
c= 30
a= 40
b= 50
c= 60
```

h. 如果指定某些形参必须使用关键字函数传递，需要在形参之前加一个*

```
def p(a,b,*,c,d):
    print("a=",a)
    print("b=",b)
    print("c=",c)
    print("d=",d)
p(10,20,c=30,d=40)
```
输出结果为
```
a= 10
b= 20
```

```
c= 30
d= 40
```

4. 函数的返回值
    a. 函数没有返回值，可以省略return
    b. 返回值只有一个，返回对应类型的值
    c. 返回值有多个，返回元组
5. 变量的作用域
    a. 局部变量和全局变量
        i. 函数内部的变量称为局部变量，仅能在函数内部使用
        ii. 全局变量可以在函数内部和外部使用
        iii. 使用global声明变量，使得变量可以在全局调用
6. 递归函数：重复调用和终止条件
7. 将函数存储在模块中
    a. 模块是拓展名为.py的文件
    b. 导入模块用保留字import，调用函数的语句为：模块名.函数名
    c. 导入特定的函数用from 模块名 import 函数名，函数名，函数名，调用函数时不需要用到模块名
    d. 给函数定别名用from 模块名 import 函数名 as 别名，调用函数时也不需要用到模块名
    e. 给模块指定别名用import 模块名 as 别名 ，调用函数的语句为：别名.函数名
    f. 导入模块中所有函数 from 模块名 import * ，调用函数时不需要用模块名

11 BUG

2021年7月20日 11:54

1. 常见的BUG
    a. 类型不匹配
    b. 变量未定义
    c. 漏了末尾的冒号
    d. 缩进错误
    e. 中英文混搭
    f. 比较运算符和赋值运算符的混用
    g. 下表越界
    h. 函数或方法使用不当
2. 调试方法
    a. 使用print函数
    b. 使用注释
3. 用户操作失误导致的bug
    a. 使用try except异常处理机制

```
try:
    a=int(input("请输入第一个整数"))
    b=int(input("请输入第二个整数"))
    result=a/b
    print("结果为",result)
except ZeroDivisionError:
    print("除数不能为0")
except ValueError:
    print("只能输入数字")
print("程序结束")
输出结果1为
请输入第一个整数10
请输入第二个整数20
结果为 0.5
程序结束
输出结果2为
请输入第一个整数a
只能输入数字
程序结束
输出结果3为
请输入第一个整数10
请输入第二个整数0
除数不能为0
程序结束
```

b. try...except...else结构:如果try没有抛出异常，则执行else语句

```
try:
    a=int(input("请输入第一个整数"))
    b=int(input("请输入第二个整数"))
    result=a/b
except ZeroDivisionError:
    print("除数不能为0")
except ValueError:
    print("只能输入数字")
else:
    print("结果为",result)
print("程序结束")
输出结果1为
请输入第一个整数10
请输入第二个整数20
结果为 0.5
程序结束
输出结果2为
请输入第一个整数a
只能输入数字
程序结束
输出结果3为
请输入第一个整数10
请输入第二个整数0
除数不能为0
程序结束
```

c. try...except...else...finally结构：无论是否发生一场都会执行finally语句，常用于释放try块中申请的资源

4. 常见的Error

ZeroDivisionError	除或者取模0
IndexError	序列中没有此索引，下标越界
KeyError	映射中没有这个键
NameError	为声明/初始化对象
SyntaxError	语法错误
ValueError	传入无效的参数，比如函数的参数传递类型不符合

5. traceback模块打印异常模块

12 类

2021年7月20日 15:50

1. 编程思想
   a. **面向对象是把构成问题事务分解成各个对象**，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为。
   b. **面向过程就是分析出解决问题所需要的步骤**，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了。
   c. 比较经典的案例关于下五子棋：
      - **面向过程设计**就是先分解过程步骤：

        1、开始游戏
        2、黑子先走
        3、绘制画面
        4、判断输赢
        5、轮到白子
        6、绘制画面
        7、判断输赢
        8、返回步骤2
        9、输出最后结果

        然后把上面每个步骤用分别的函数来实现，问题就解决了。
      - **面向对象设计**又是另一套思路了。

        整个五子棋可以分为：

        1、**黑白双方**，这两方的行为是一模一样的
        2、**棋盘系统**，负责绘制画面
        3、**规则系统**，负责判定诸如犯规、输赢等。

        第一类对象（玩家对象）负责接受用户输入，并告知第二类对象（棋盘对象）棋子布局的变化，棋盘对象接收到了棋子的i变化就要负责在屏幕上面显示出这种变化，同时利用第三类对象（规则系统）来对棋局进行判定。
2. 类与对象
   a. 类：多个类似事务的群体的统称
   b. 对象：每一个类之下包含的相似的不同个例
3. 类的创建
   a. 类名由一个或者多个单词组成，每个单词的首字母大写，其余小写（规范）
   b. 类的组成
      i. 类属性：直接写在类里的变量
      ii. 实例方法：类里定义的函数
      iii. 静态方法：使用staticmethod进行修饰的函数
      iv. 类方法：使用classmethod进行修饰的函数
      v. 初始化方法：self.name称为实体属性，将局部变量name的值赋给了它，可供

类里所有方法使用。初始化只需要输入两个参数传递给name和age，不需要传递参数给self。可以定义属性默认值，如下图中gender属性就无需外部实参传递，即默认值

```
class Student:
    #初始化方法
    def __init__(self,name,age):
        self.name=name
        self.age=age
        self.gender="男"
    #类属性
    native_pace="CC98"
    #实例方法
    def eat(self):
        print("吃饭")
    #静态方法
    @staticmethod
    def method():
        print("静态方法")
    #类方法
    @classmethod
    def cm(cls):
        print("类方法")
```

4. 对象的创建：根据类对象创建出的实例属性

```
class Student:
    #初始化方法
    def __init__(self,name,age):
        self.name=name
        self.age=age
    #类属性
    native_pace="CC98"
    #实例方法
    def eat(self):
        print("吃饭")
    #静态方法
    @staticmethod
    def method():
        print("静态方法")
    #类方法
     @classmethod
    def cm(cls):
        print("类方法")
stu1=Student("吴朝晖",60)
print(stu1)
print(id(stu1))
print(type(stu1))
print(stu1.name)
print(stu1.age)
```

输出结果为

吴朝晖

5. 类组成的使用
   a. 类属性。类的类属性改变后，类的对象的类属性也随之改变。每一个类对象中由一个类指针指向类，可以调用类中的各种组成。
   b. 类方法，直接通过类名.方法名调用
   c. 静态方法,直接通过类名.方法名调用
   d. 实例方法，通过类名.方法名(对象名)或者对象名.方法名调用

```
class Student:
    #初始化方法
    def __init__(self,name,age):
        self.name=name
        self.age=age
    #类属性
    native_pace="CC98"
    #实例方法
    def eat(self):
        print(self.name,"吃饭")
    #静态方法
    @staticmethod
    def method():
        print("静态方法")
    #类方法
    @classmethod
    def cm(cls):
        print("类方法")
stu1=Student("张三",20)
stu2=Student("李四",20)
#类属性
print(stu1.native_pace)
print(stu2.native_pace)
Student.native_pace="浙江"
print(stu1.native_pace)
print(stu2.native_pace)
#类方法
Student.cm()
#静态方法
Student.method()
#实例方法
Student.eat(stu1)
stu2.eat()
```

输出结果为

```
CC98
CC98
浙江
浙江
类方法
静态方法
张三 吃饭
```

```
李四 吃饭
```

6. 动态绑定属性和方法

```
class Student:
    #初始化方法
    def __init__(self,name,age):
        self.name=name
        self.age=age
    #类属性
    native_pace="CC98"
    #实例方法
    def eat(self):
        print(self.name,"吃饭")
    #静态方法
    @staticmethod
    def method():
        print("静态方法")
    #类方法
    @classmethod
    def cm(cls):
        print("类方法")
stu1=Student("张三",20)
stu2=Student("李四",30)
#定义新的属性，只针对单个类对象
stu1.gender="男"
print(stu1.name,stu1.age,stu1.gender)
#绑定新的方法
def show():
    print("绑定新的方法")
stu1.show=show
stu1.show()
输出结果为
张三 20 男
绑定新的方法
```

13 类拓展

2021年7月21日 10:34

1. 封装：提高程序的安全性
   a. 将数据（属性）和行为（方法）包装到类对象中。在方法内部对属性进行操作，在类对象的外部调用方法。这样，无需关心方法内部的具体实现细节，从而隔离了复杂度。
   b. 在python中没有专门的修饰符用于属性的私有，如果该属性不希望再类对象外部被访问，前面使用两个“_”，但仍然可以通过下列方法调用，srds，尽量避免

```
class Student:
    def __init__(self,name,age) :
        self.name=name
        self.__age=age
    def show(self):
        print(self.name,self.__age)
stu1=Student("张三",20)
print(stu1.name)
stu1.show()
#print(dir(stu1))
print(stu1._Student__age)
输出结果为
张三
张三 20
20
```

2. 继承：如果编写的类是另一个现成类的特殊版本，可使用继承。一个类继承另一个类的所有属性和方法。原有的类称为父类，新类称为子类
   a. 如果一个类没有继承任何类，则默认继承object类
   b. 支持多继承
   c. 定义子类时，必须在其构造函数中调用父类的函数

```
class Person(object):
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def info(self):
        print(self.name,self.age)
class Student(Person):
    def __init__(self, name, age,gender):
        super().__init__(name, age)
        self.gender=gender
stu=Student("张三",20,"男")
stu.info()
输出结果为
张三 20
```

   d. 方法重写：子类对父类的某个属性或方法不满意，可以在子类中进行重写

```
class Person(object):
    def __init__(self,name,age):
```

```
        self.name=name
        self.age=age
    def info(self):
        print(self.name,self.age)
class Student(Person):
    def __init__(self, name, age,gender):
        super().__init__(name, age)
        self.gender=gender
    def info(self):
        super().info()
        print(self.gender)
stu=Student("张三",20,"男")
stu.info()
```

输出结果为

```
张三 20 男
```

3. object类
   a. dir()函数可以查看类的属性和方法
   b. _str_()用于返回一个对于"对象的描述，对应于内置函数str()经常用于print()方法，帮我们查看对象的信息，经常重写_str_()。

```
class Person(object):
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def info(self):
        print(self.name,self.age)
    def __str__(self) -> str:
        return "我叫{}，今年{}岁".format(self.name,self.age)
stu=Person("张三",20)
print(stu)
```

输出结果为

```
我叫张三，今年20岁
```

4. 多态：简单来说，多态就是具有多种形态，它指的是：即使不知道一个变量所引用的对象到底是什么类型，仍然可以通过这个变量调用方法，在运行过程中根据变量所引用对象的类型，动态决定调用哪个对象中的方法。
5. 特殊方法和特殊属性
   a. 特殊属性

__dict__	获得类对象或者实例对象所绑定的所有属性和方法的字典
__class__	实例对象所属于的类
__bases__	子类的父类，以元组格式输出
__mro__	类的层次结构
__subclasses__()	父类的子类

```
class Person(object):
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def info(self):
        print(self.name,self.age)
    def __str__(self) -> str:
```

```
        return "我叫{}，今年{}岁".format(self.name,self.age)
class Student(Person):
    pass
stu=Person("张三",20)
print(stu.__dict__)#实例对象的属性字典
print(Person.__dict__)#类的属性和方法字典
print(stu.__class__)#实例对象所属于的类
print(Person.__bases__)#子类的父类
print(Person.__mro__)#类的层次结构
print(Person.__subclasses__())#父类的子类
```

输出结果为

```
{'name': '张三', 'age': 20}
{'__module__': '__main__', '__init__': <function Person.__init__ at
0x0000019ACA400D30>, 'info':
<function Person.info at 0x0000019ACA400DC0>, '__str__': <function Person.__str__ at
0x0000019ACA400E50>, '__dict__': <attribute '__dict__' of 'Person' objects>,
'__weakref__': <attribute '__weakref__' of 'Person' objects>, '__doc__': None}
<class '__main__.Person'>
(<class 'object'>,)
(<class '__main__.Person'>, <class 'object'>)
[<class '__main__.Student'>]
```

b. 特殊方法

特殊方法	__add__	通过重写__add__()方法，看时自定义对象由+的功能
特殊方法	__len__	通过重写__len__()方法，让内置函数len()的参数可以是自定义类型
特殊方法	__new__	用于创建对象
特殊方法	__init__	对创建的对象进行初始化

```
class Student:
    def __init__(self,name):
        self.name=name
    def __add__(self,other):
        return self.name+other.name
    def __len__(self):
        return len(self.name)
stu1=Student("张三")
stu2=Student("李四")
print(stu1.__add__(stu2))
print(stu1.__len__())
```

输出结果为

```
张三李四
2
```

```
class Person(object):
    def __init__(self,name,age):
        self.name=name
        self.age=age
        print("__inin__被调用了，self的id为{}".format(id(self)))
```

```
    def __new__(cls,*args,**kwargs):
        print("__new_被调用了，cls的id值为{}".format(id(cls)))
        obj=super().__new__(cls)
        print("创建的对象的id为:{}".format(id(obj)))
        return obj
print("object这个类对象的id为{}".format(id(object)))
print("Person这个类对象的id为{}".format(id(Person)))
p1=Person("张三",20)
print("p1的id为{}".format(id(p1)))
输出结果为
object这个类对象的id为140713771777536
Person这个类对象的id为2698126488864
__new_被调用了，cls的id值为2698126488864
创建的对象的id为:2698133139216
__inin__被调用了，self的id为2698133139216
p1的id为2698133139216
```

创建一个实例对象的过程为，先调用类对象中的__new__()方法，传入类对象的地址，创建了新的实例对象，传入给了实例对象的self，然后再赋值给了赋值对象。

6. 类的浅拷贝与深拷贝
   a. 变量的赋值操作：形成两个对象，实际还是指向同一个对象

```
class CPU:
    pass
class DISK:
    pass
class Computer:
    def __init__(self,cpu,disk):
        self.cpu=cpu
        self,disk=disk
cpu1=CPU()
cpu2=cpu1
print(cpu1,id(cpu1))
print(cpu2,id(cpu2))
输出结果为
<__main__.CPU object at 0x0000021A9B7C1FD0> 2313301008336
<__main__.CPU object at 0x0000021A9B7C1FD0> 2313301008336
```

   b. 浅拷贝：python内的拷贝一般都是浅拷贝，拷贝时，对象包含的子对象内容不拷贝，因此，原对象与拷贝对象会引用同一个子对象

```
class CPU:
    pass
class Disk:
    pass
class Computer:
    def __init__(self,cpu,disk):
        self.cpu=cpu
        self.disk=disk
cpu1=CPU()
disk1=Disk()
computer1=Computer(cpu1,disk1)
import copy
computer2=copy.copy(computer1)
```

```
print(computer1,computer1.cpu,computer1.disk)
print(computer2,computer2.cpu,computer2.disk)
```

输出结果为

```
<__main__.Computer object at 0x000002DF9FEF1F70> <__main__.CPU object at 0x000002DF9FEF1FD0> <__main__.Disk object at 0x000002DF9FEF1FA0>
<__main__.Computer object at 0x000002DF9FEF1B20> <__main__.CPU object at 0x000002DF9FEF1FD0> <__main__.Disk object at 0x000002DF9FEF1FA0>
```

c. 深拷贝：递归拷贝对象中包含的子对象，原对象和拷贝对象所有的子对象也不相同

```
class CPU:
    pass
class Disk:
    pass
class Computer:
    def __init__(self,cpu,disk):
        self.cpu=cpu
        self.disk=disk
cpu1=CPU()
disk1=Disk()
computer1=Computer(cpu1,disk1)
import copy
computer2=copy.deepcopy(computer1)
print(computer1,computer1.cpu,computer1.disk)
print(computer2,computer2.cpu,computer2.disk)
```

输出结果为

```
<__main__.Computer object at 0x000001FD55FD1F70> <__main__.CPU object at 0x000001FD55FD1FD0> <__main__.Disk object at 0x000001FD55FD1FA0>
<__main__.Computer object at 0x000001FD55FD1AC0> <__main__.CPU object at 0x000001FD55FD1490> <__main__.Disk object at 0x000001FD55FD14F0>
```

14 模块

2021年7月22日 10:08

1. 定义
    a. 一个模块中包含类，函数，语句
    b. 拓展名为.py的文件就是一个模块
    c. 好处
        i. 方便其他程序和脚本的导入和使用
        ii. 避免函数名和变量名冲突
        iii. 提高代码的可维护性
        iv. 提高代码的可重用性
2. 自定义模块
    a. 创建一个.py文件，名称尽量不要与python的自带模块相同
    b. 导入模块
        i. import 模块名称 （as 别名）
        ii. from 模块名称 import 函数/变量/类名 （as 别名）

```
import math
print(math.pi)
输出结果为
3.141592653589793
```

```
from math import pi
print(pi)
输出结果为
3.141592653589793
```

3. 以主程序形式运行

    if __name__ = "__main__":
        执行语句

    只有当这个程序是主程序时才会执行该语句
4. 包
    a. 分层次的目录结构，将一组功能相近的模块组织在同一个目录下
    b. 作用：
        i. 代码规范
        ii. 避免模块名称冲突
    c. 包与目录的区别：包中包含__init__.py文件
    d. 导入
        i. import 包名
        ii. import 模块名
        iii. from 包名 import 模块名
        iv. from 包名.模块名 import 函数/变量/类
5. 内置模块

sys	与python解释器及其环境操作相关的标准库
time	提供与事件相关的各种函数的标准库
os	提供访问操作系统服务功能的标准库
calendar	提供与日期相关的各种函数的标准库
urllib	用于读取来自网上（服务器）的数据标准库
json	用于使用JSON序列化和反序列化对象
re	用于在字符串执行中执行正则表达式匹配和替换
math	提供标准算数运算函数的标准库
decimal	用于进行精确控制运算精度、有效数位和四舍五入操作的十进制运算
logging	提供了灵活的记录时间、错位、警告和调试信息等日志信息的功能

6. 第三方模块
    a. 安装：pip install 模块名
    b. 使用：import 模块名

15 文件

2021年7月22日 16:18

1. 编码格式
   a. 常见的字符编码格式
      i. python的解释器使用的是Unicode(内存)
      ii. .py文件在磁盘上使用UTF-8存储(外存)
2. 文件的读写原理
   a. 文件的读写俗称IO操作
   b. 文件读写操作流程：python操作文件->打开或者新建文件->读写文件->关闭资源
   c. 操作原理
3. 文件的读写操作
   a. 创建文件对象
      file=open(filename[,mode,encoding])
   b. 文件打开模式

r	以只读模式打开文件，文件的指针将会放在文件的开头
w	以只写模式打开文件，如果文件不存在则创建，如果文件存在，则覆盖原有内容，文件指针在文件的开头
a	以追加模式打开文件，如果文件不存在则创建，文件指针在文件开头，如果文件存在，则在文件末尾追加内容，文件指针在原文件末尾
b	以二进制方式打开，不能单独使用，与其他模式一起使用，如ab，rb
+	以读写模式打开文件，不能单独使用，需要与其他模式一起使用，a+

   c. 常用方法

read([size])	从文件读取指定的字节数或字符数，如果未给定或为负则读取所有。
readline()	读取整行，包括 "\n" 字符。
readlines()	读取所有行并返回列表
write(str)	将字符串str写入文件
writelines(s_list)	将字符串列表s_list写入文本文件，不增加换行符
seek(offsef[,whence])	把文件指针移动到新的文职，offset表示相对于whence的位置，为正则向结束方向移动，为负则向开始方向移动。whence不同的值代表不同含义： 0，开头（默认值 1，当前位置 2：文件尾

tell()	返回文件指针的当前位置
flush()	把缓冲区的内容写入文件，但不关闭文件
close()	把缓冲区的内容写入文件，同时关闭文件

d. with语句
   i. with open(filename[,mode,encoding]) as file:
      操作语句
   ii. 不需要写close语句，自动管理上下文资源