python核心

数据类型 函数 list tuple set 面向对象 模块 异常处理

项目

Java

数据类型 方法 面向对象 String

1.Python简介

编程语言

1.1 计算机基础结构

- 硬件：键盘/显示器/鼠标 CPU/内存/硬盘
- 软件
  - 系统软件：操作系统、计算机网络...
  - 应用软件：计算机应用程序

软件：程序+文档

程序：一组 计算机能识别和执行 的指令集合

文档：为了便于了解程序所需的说明性资料

1.2 基础知识

1.2.1 背景

python创始人： 荷兰人 吉多*范罗苏姆 1989 圣诞节 决定开发新的脚本解释程序

Python（蟒蛇）：名字来源于英国20世纪70年代首播的电视喜剧《蒙提*派森的飞行马戏团》（Monty Python）

1991年发布了第一个版本

1.2.2 Python定义

简洁优雅、免费开源、跨平台、面向对象的**解释型** 计算机高级编程语言

```
public class HelloWorld{
    public static void main(String[] args){
        System.out.println("Hello World");
    }

}
```

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

1.2.3 Python执行方式

1. 交互式

在命令行输入指令，回车得到结果

优点：简单，用于测试基础语法

缺点：执行的语句会随着交互环境的关闭而消失

2.文件式

将指令编写到 .py 结尾的文件中，可以重复运行程序

总结：

执行方式	特点	使用场景
交互式	进入交互环境就可以执行简单的语句，**退出即销毁**	测试单条或语句不多时使用
文件式	创建文件，编写语句，执行文件，**不删除永存**	编写项目或逻辑复杂使用

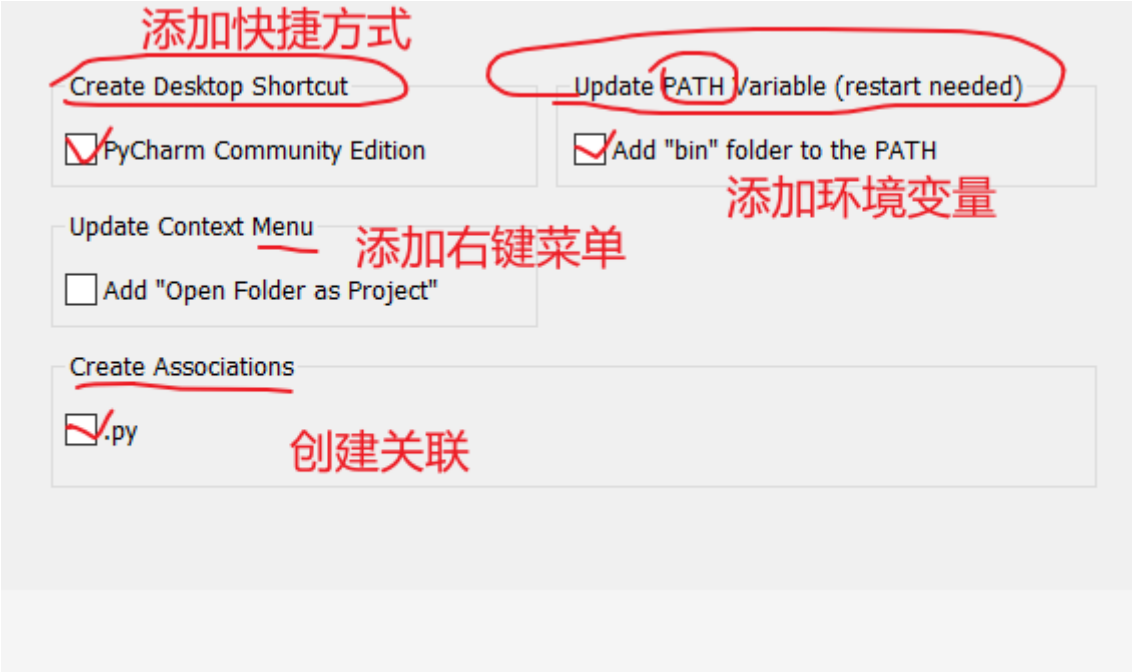
1.2.4 执行过程

计算机只能识别机器码（1 0）,不能识别源代码（python）

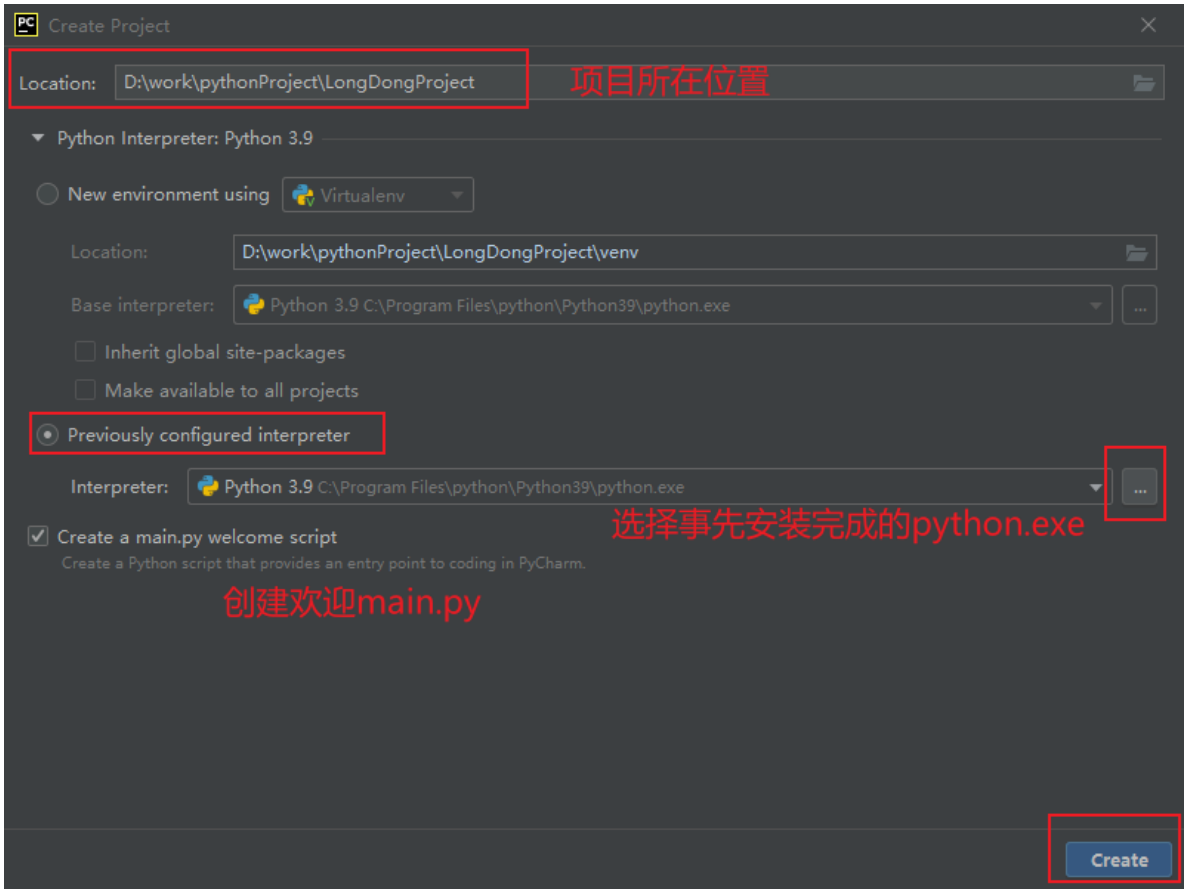
由源代码转变成机器码的过程：编译和解释

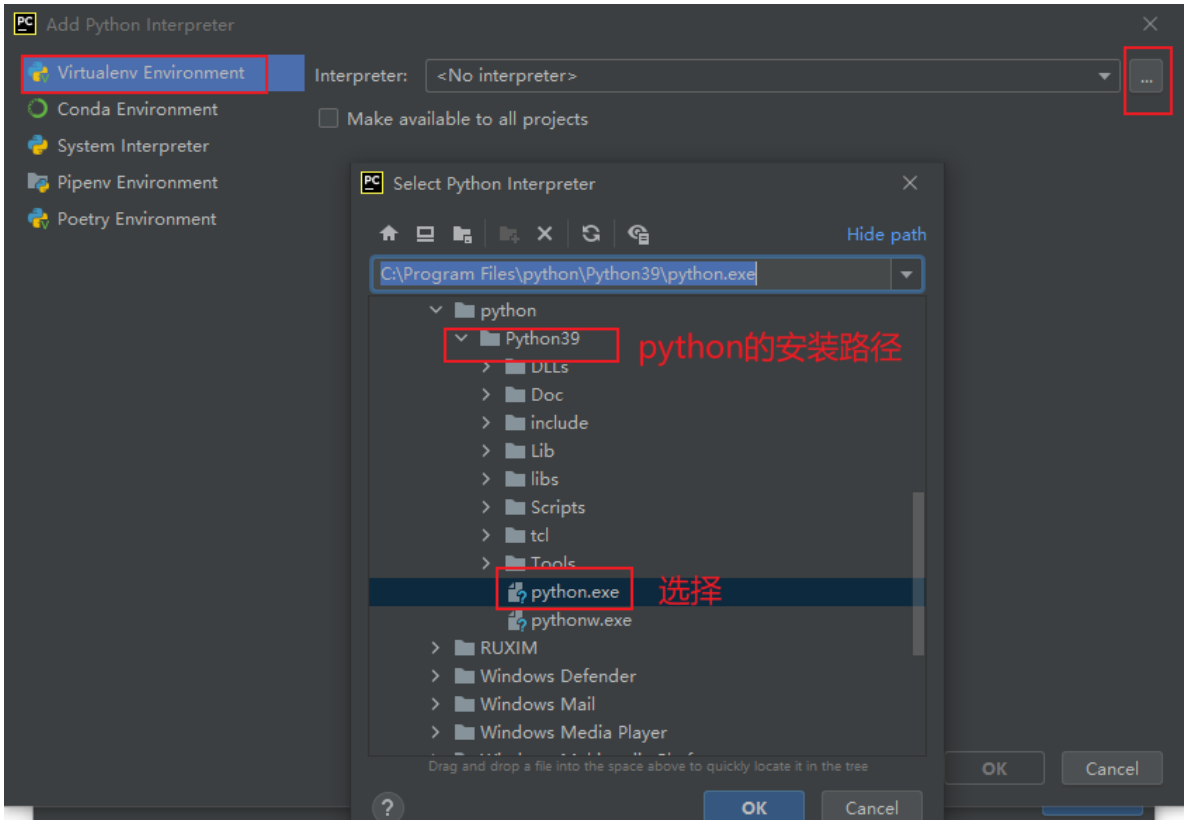
- 编译：在程序运行之前，通过编译器将源代码变成机器码，例如：C、Java
  - 特点：运行速度快 开发效率低
- 解释：在程序运行之时，通过解释权对程序 **逐行翻译执行**，例如：Python 、JavaScript
  - 特点：运行速度慢 开发效率高

1.2.5 PyCharm软件


添加快捷方式
Create Desktop Shortcut
PyCharm Community Edition
Update PATH Variable (restart needed)
Add "bin" folder to the PATH
添加环境变量
Update Context Menu
添加右键菜单
Add "Open Folder as Project"
Create Associations
.py
创建关联

新建项目


Create Project
Location:
D:\work\pythonProject\LongDongProject
项目所在位置
Python Interpreter: Python 3.9
New environment using
Virtualenv
Location:
D:\work\pythonProject\LongDongProject\venv
Base interpreter:
Python 3.9 C:\Program Files\python\Python39\python.exe
Inherit global site-packages
Make available to all projects
Previously configured interpreter
Interpreter:
Python 3.9 C:\Program Files\python\Python39\python.exe
选择事先安装完成的python.exe
Create a main.py welcome script
Create a Python script that provides an entry point to coding in PyCharm.
创建欢迎main.py
Create



Java 包：解决命名冲突

cn.tedu.longdong.wanggong.zhansan

cn.tedu.longdong.wanggong.lisi

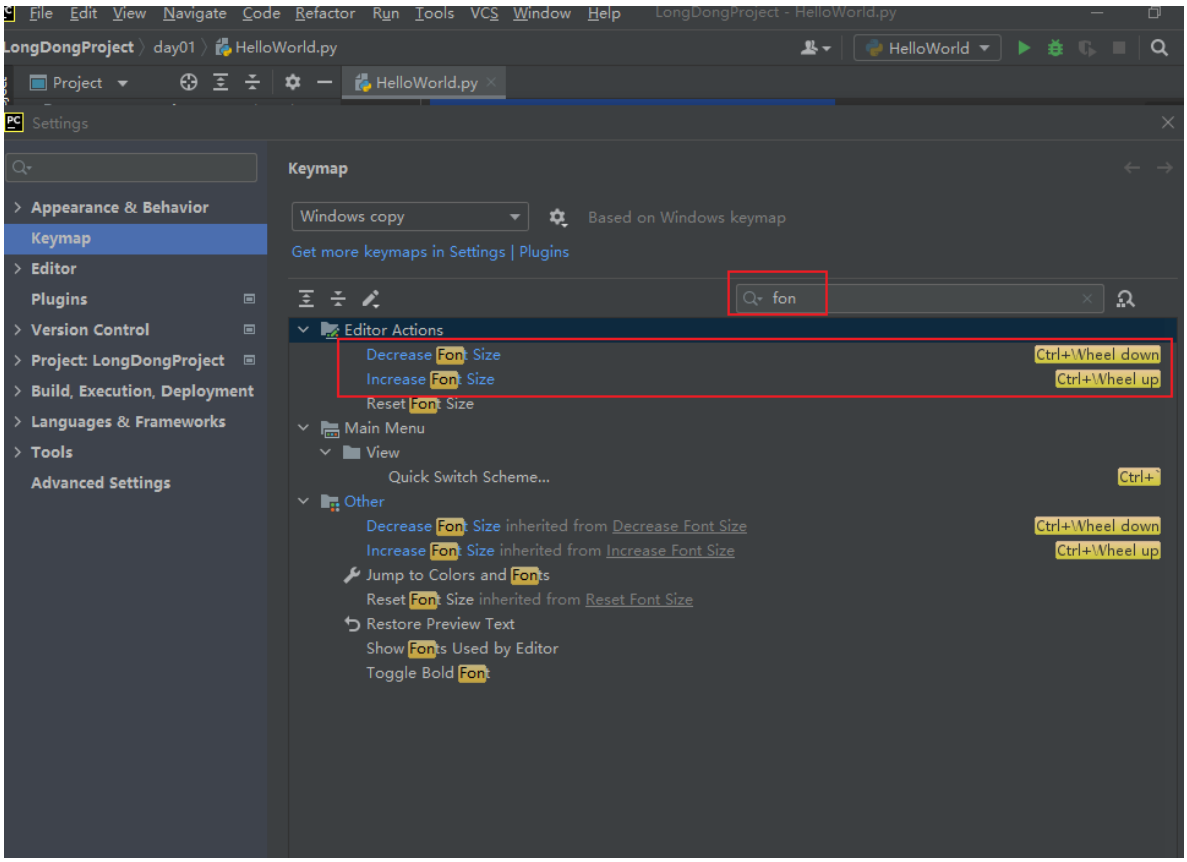
1.2.6 HelloWorld

创建Project --> 创建Day01目录 --> 创建HelloWorld.py文件

```
print("Python基础 第一天 ！！！")
```

【补充】：

设置界面字体缩放：



2.数据基本运算

2.1 基础知识

2.1.1 注释 Note

作用：给开发者看的，通常是对代码的描述信息

- 单行注释：以 # 开头
- 多行注释：三引号开头，三引号结尾

```
# ctrl + / 快捷键:将选中的内容进行注释
"""
输出函数:
    print()
"""
```

2.1.2 函数 Function

作用：表示一个特定的功能

关注：功能、参数、返回值

2.1.2.1 print 函数

- 语法：print(value,...,sep=' ',end='\n')
- 作用：将括号的内容显示在控制台中

- 参数：
  - value：表示要打印的数据，可以有任意个
  - sep：分隔符，两个或以上值的分隔方式，默认是空格（字符串类型）
  - end：结束符，数据输出完毕后末尾打印的字符串，默认为\n
- 返回值：无
- 说明：
  - 1.seq表示多个值之间的分隔方式
  - 2.要实现不换行，可以设置end的\n设置成其他字符即可
  - 3.end 和sep 两个参数的设置顺序无关

```
# 数据为单引号
# 单引号，引用的内容原样输出;单引号中使用双引号，可以将双引号打印出来
print('单引号')
print('"单引号2"')
# print(''单引号3'') 报错

# 数据为双引号 引用的内容原样输出;双引号中使用单引号，可以将单引号打印出来
print("双引号")
print("'双引号2'")
# print(" " 双引号3" ") 报错

# 数据为三引号 引用多行文本
print("---------------------------------")
print("""现在时间：18:05，
距离下课还有35分钟！！！""")
print("---------------------------------")


# 空格
# 1.直接在引号打空格，空格数不限，都会体现
print("AA BB  CC")
# 2.多个数据之间用,隔开，以空格作为分隔符显示
print("AA","BB","CC")
# 补充：如果想要更改分隔符，定义sep="分隔符"即可
print("AA","BB","CC",sep="*")

print("AA","BB",)
print("CC","DD")
# 拼接
print("AA""BB")
print("AA"+"BB")

# 换行
# 可以使用end参数设置结束符
print("坚持\n就是胜利",end=" ")
print("加油",end=" ")
print("AAA",end=" ")
```

2.1.2.2 input 函数

- 语法：**变量名=input(prompt=None)**

- 作用：将用户输入的内容赋值给变量
- 参数：
  - prompt：表示提示信息（输入显示的内容），默认是空
- 返回值：输入的字符串数据

```
# input() 输入函数
# hello = input("Hello，马上下课")
# food = input("一会儿吃啥？")
#
# print(hello)
# print(food)

a = input("输入内容1：")
b = input("输入内容2：")
print('内容1:',a,'内容2：',b,'结束')
```

课堂案例

1.三种方式在控制台输出古诗《静夜思》

提示：

1.分行输出

2.sep设置成\n

3.三引号

```
# 要求:打印古诗
# 方法1
# print("      静夜思")
# print("       李白")
# print("    床前明月光，")
# print("    疑是地上霜。")

# 方法2
# print("    静夜思","    李白",sep="\n")

# 方法3
print("""              静夜思
             李白
            床前明月光，
            疑是地上霜。
            ...
""")
```

2.红包：在控制台输入发的红包的金额，红包的个数，打印每个人平均每个人的金额

输入：

请输入红包的金额： 20

请输入红包的个数： 5

输出：

每人平均可以收到：4 元

提示：

int(变量名)/int(变量名)

```
# 要求：红包：在控制台输入发的红包的金额，红包的个数，打印每个人平均每个人的金额

# 1.获取数据
money = input("请输入红包的总金额：")
number = input("请输入红包的个数：")

# 2.逻辑处理
money2 = int(money)/int(number)

# 3.输出结果
print("每人平均收到的红包为：",money2,"元")
```

2.2 数据

2.2.1 变量

定义：关联一个数据对象的标识符

命名规则：

- 字母 数字 下划线组成
- 不能以数字开头
- 不能使用Python关键字
- 严格区分大小写
- 不能使用Python内置函数名

命名方法：

- 驼峰命名法

  ```
  # 大驼峰命名法：每个单词首字母大写
  SaveToExcel HelloWorld

  # 小驼峰命名法：首个单词首字母小写，其他单词首字母大写
  saveToExcel helloWorld
  ```

- 匈牙利命名法：每个单词之间使用下划线的方式连接

  ```
  # 匈牙利命名法
  save_to_excel hello_world
  ```

- 应用场景：变量名、函数名、类名、文件名、项目名
- 核心要求：见名知意

2.2.2 赋值

作用：创建或改变一个变量所关联的数据对象

语法：

- 变量名 = 数据对象
- 变量名1 = 变量名2 = 数据对象
- 变量名1,变量名2 = 数据对象1，数据对象2 作用：数据交换

说明：

- 变量在使用前必须赋值
- 一个变量只能绑定在一个数据对象上
- 一个对象可以被多个变量关联

案例：

```
# 赋值
# 1.变量在使用之前必须先赋值
# print(first_name)

# 2.1 创建
first_name = "张三"
print(first_name)

# 2.2 修改 ：一个变量只能绑定在一个数据对象上
first_name = "李四"
print(first_name)

# 3.一个数据对象可以关联多个变量
second_name = "王五"
# 变量名1 = 变量名2 = 数据对象
first_name = second_name = "赵六"
print(first_name)
print(second_name)

# 变量名1,变量名2 = 数据对象1，数据对象2  作用：数据交换
first_name,second_name = "喜羊羊","灰太狼"
print(first_name)
print(second_name)
```

2.2.3 删除语句

语法：

- del 变量名
- del 变量名1,变量名2...

作用：

- 用于删除变量，同时解除和对象的关联

自动化内存管理的引用计数：

- 每个对象记录被变量绑定/引用的次数，当引用次数为0时被销毁

  数据对象的内存构成：

  - id：存储空间地址，唯一，由操作系统分配
  - type：数据类型，记录数据的类型状态
  - value：变量值，存储的值本身
  - references：引用次数，数据对象被引用的次数

2.2.4 核心类型

在python中变量没有类型，但是关联的对象有类型

2.2.4.1 整型 int

包含正数 、 负数 、0

字面值：

十进制：每位用十种状态（0-9）计数，逢十进一，写法：0-9

二进制：每位用两种状态计数，逢二进一，写法：0b开头，后面跟0、1

八进制：每位用八种状态计数，逢八进一，写法：0o开头，后跟0~7。

十六进制：每位用十六种状态计数，逢十六进一，写法：0x开头，后跟0-9, A-F, a-f

```
# 十进制 默认
num01 = 10

# 二进制 0b开头
num2 = 0b10
print(num2)

# 八进制 0o开头
num3 = 0o10
print(num3)

# 十六进制 0-9 a-f 0x开头
num4 = 0x10
print(num4)
```

2.2.4.2 浮点型 float

包含正数、负数、0.0

字面值：

小数：1.0 2.4 -3.14 0.0

科学计数法：

e/E（正负号）指数

例如： 1.23e-2 等同于0.0123

```
# 浮点型
# 小数
num1 = 1.2
# 科学计数法
# num2 = 1.23e-2
num2 = 1.23456e5
print(num2)
```

2.2.4.3 字符串 str

用来记录文本信息

字面值：单引号 、 双引号

```
# 字符串
name01='张三'
print(name01)
name02="张三"
print(name02)
```

2.2.4.4 布尔类型 bool

用来表示真和假

只有两个值：True（真，本质上是1） 和 False（假，本质上是0）

【注意】：True、False关键字首字母大写

```
# 布尔类型
result = input("时间：") == "10"
print(result)
```

2.2.4.5 空值对象 None

概念：表示不存在的特殊对象

```
# None
method=print("HelloWorld!")
# print(method,"---------")
print(None==method)
```

None在Python解释器启动时自动创建，解释器退出时销毁，在一个解释器进程中**只有一个None**

所以：None is None and None，不是0，空字符串

```
print(None == 0)
print(None == '')
print(None == "")
print(None == False)
print(None == None)
```

扩展：type

- 格式：**type(object)**
- 作用：查看对象的数据类型
- 参数：object:查看的数据对象
- 返回值：对象的数据类型（ < class 'xxxxx' >）

```
# type()
# num01 = 1.23 # <class 'float'>
# num01 = 1 # <class 'int'>
# str1 = "张三" #<class 'str'>
# result = type(True) # <class 'bool'>
result = type(None) # <class 'NoneType'>
print(result)
```

2.3 运算

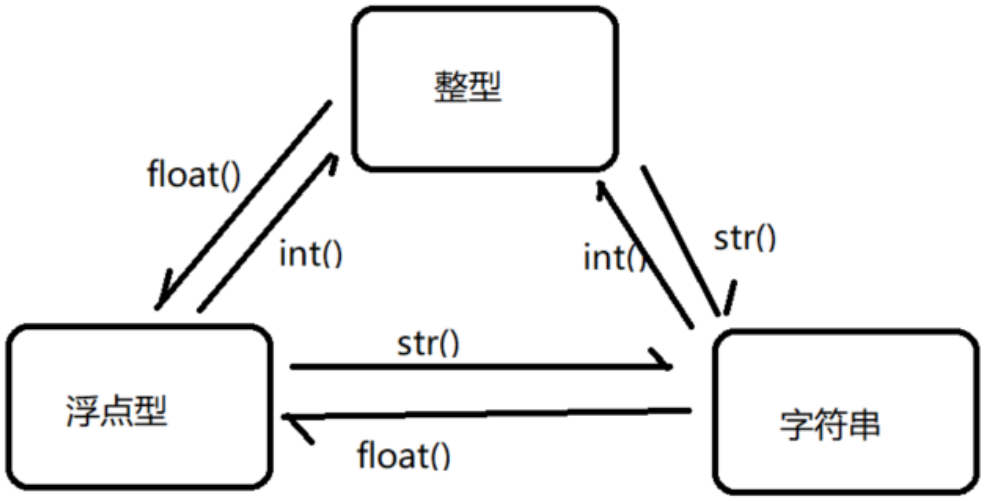
2.3.1 类型转换

- 转化为整型： int(数据)
- 转换为浮点型：float(数据)
- 转换为字符串：str(数据)
- 转换成布尔：bool(数据)

结果为假： 0 0.0 '' "" Flase None

【注意】：字符串转为其他类型时，需要”长的像“目标类型

总结：




```
# 运算
# 1.类型转换
# str-> int
```

```
# data01=int("3")
# print(data01)
# print(type(data01))

# int -> str
# data02 = str(3)
# print(data02)
# print(type(data02))

# str -> float
# data03 = float("1.1")
# print(data03)
# print(type(data03))
# # float->str
# data04 = str(1.2)
# print(data04)
# print(type(data04))

# int->float
# data05 = float(25)
# print(data05)
# # float->int
# data06 = int(3.7)
# # 3 向下取整
# print(data06)

# 字符串转为其他类型时，需要”长的像“目标类型
# 报错 ValueError
# print(int("11.2"))
# print(float("a"))
```

进制转换

- bin(number)：把整型转为二进制
- oct(number)：把整型转为八进制
- hex(number)：把整型转为十六进制

```
# 2.进制转换
# 十转二
# print(bin(6))
# # 十转八
# print(oct(8))
# # 十转十六
# print(hex(15))

print(0b10)
print(int("0b10",2))

print(0o10)
print(int("0o10",8))

print(0x10)
print(int("0x10",16))
```

课堂案例

在控制台输入商品单价（浮点型）、商品购买数量（整型）和支付的金额（浮点型），计算找回的零钱

```
输入：
请输入商品单价： 5
请输入商品购买数量： 4
请输入支付金额： 30

输出：
应找回：10.0
```

乘法：*

2.3.2 运算符

2.3.2.1 算术运算符

符号	功能	使用
+	加法	1 + 2
-	减法	5 - 4
*	乘法	2 * 3
/	除法，获取的商是小数（结果为浮点型）	8 / 2
%	取余	8 % 3
//	地板除，向下取整，用于获取整数商	10//3
**	求幂，表示一个数的多少次方	2**3

【注意】：

1. 除法返回的结果是浮点型
2. 凡是浮点数参与的运算结果是浮点型

- 优先级

```
()
**
* / % //
+ -
```

- round(number,ndigits=None)
  - 功能：返回浮点数的四舍五入的值
  - 参数：number：要计算的数值

    ndigits（可选）：表示保留的小数点位数，如果没有就是整数
  - 返回值：浮点数的四舍五入的值

四舍五入规则：

1. 保留位数的后一位 <= 4，不进位
2. 保留位数的后一位 = 5
    - 该位数后面没有数了，不进位
    - 该位数后面还有数，进位
3. 保留位数的后一位 >=6 ，进位

```
print(round(2.215,2)) # 2.21
print(round(2.2151,2)) # 2.22
```

【补充】：

1.当小数部分为0.5，整数为奇数进1，为偶数不进（舍掉小数）

2.当小数部分不为0.5，只有第一个参数，四舍五入

保留小数：

1.小数部分保留位置后一位为5，保留位置 奇 不进 偶 进

```
print(round(2.215,2)) # 2.21
print(Decimal(2.215))
print(round(2.2151,2)) # 2.22
print(Decimal(2.2151))
```

练习

1. 反转三位数

   输入任意三位数，例如：

   输入：123 输出：321

   输入：120 输出：21

   输入：100 输出：1

   ```
   # 反转三位数
   number = int(input("请输入三位数："))

   hundred = number // 100
   ten = number % 100 // 10
   one = number % 10

   result = one * 100 + ten * 10 + hundred
   print("反转后的数字为：",result)
   ```

2. 将温度从华氏度转摄氏度，结果保留1位

   转化公式：C=5/9*(F-32)

```
# 华氏转摄氏
temp = float(input("请输入华氏温度："))
c = round(5/9*(temp-32),1)
print(temp,"对应的摄氏度为：",c,"C")
```

2.3.2.2 增强运算符

符号	等效写法	案例(y=10,x=2)
y += x	y = y+x	y+=2 y=12
y -= x	y = y-x	y-=2 y=8
y*=x	y = y *x	y *=2 y=20
y/=x	y = y / x	y /=2 y=5.0
y %= x	y = y % x	y %=2 y=0
y ** =x	y = y ** x	y **=2 y=100
y //=x	y = y //x	y //=2 y=5

- 作用：为算术运算符增加对自身赋值的功能

练习

1.输入一个四位整数，计算每位数字相加的和

例如：输入 1234 ，打印 1+2+3+4

```
# 四位数 每位相加
number = int(input("请输入四位数："))
# 1234
result = number // 1000 # 1
result += number // 100 % 10  # 1 +=  2
result += number // 10 % 10  # 3 += 3
result += number % 10

print("结果为:",result)
```

2.3.2.3 比较运算符

符号	描述	案例（x=10,y=10）
<	小于	x<y # False
<=	小于等于	x<=y # True
>	大于	x>y #Flase
> =	大于等于	x>=y # True
==	等于	x==y # True
!=	不等于	x!=y # False

- 返回值：布尔类型的值

练习

1. 输入的是正整数
2. 输入的是正确月份

```
# 正整数
# print(0 <= int(input("请输入正整数："))

# 月份
print(0 < int(input("请输入一个月份：")) < 13)

# result = input("月份:")
# print(type(result))
```

3. 输入的不是偶数

```
# 不是偶数
number = int(input("请输入一个整数："))
print(number % 2 != 0)
```

2.3.2.4 逻辑运算符

1．and - 与

- 作用：表示 且 的关系 **一假俱假**
- 格式： **表达式1 and 表达式2**
- 说明：只要有一个表达式为False，结果都是False

2. **or - 或**

- 作用：表示 或者 的关系 **一真俱真**
- 格式：**表达式1 or 表达式2**
- 说明：表达式中只要有一个为真结果都为True

3.not - 非

- 作用：表示 取反 ，真变假，假变真
- 格式： **not 表达式**
- 说明：如果表达式为真，则返回 False，反之为True

4.短路运算

- 说明：一旦结果确定，后面的语句不再执行
- 在逻辑运算时，复杂的（耗时长）判断 放到最后面
- and：如果表达式1为假，表达式2不执行
- or：如果表达式1为真，表达式2不执行

练习

1.年龄大于20，身高小于180

```
# 逻辑运算符
print(int(input("请输入年龄：")) > 20 and int(input("请输入身高：")) < 180)
```

2.3.2.5 身份运算符

1. 语法

   ```
   x is y
   x is not y
   ```

2. 作用

   is用于判断两个对象是否是同一个对象，是返回True，否则返回False；is not 的作用和is相反

3. 说明

   是否是同一个对象：对象的内存存储地址是否一致

   ```
   # 身份运算符
   a = 'input()'
   b = "input()"
   print(a is b)
   print(a == b)
   ```

4. id()
   - 格式：id(object)
   - 作用：查看对象的计算机内存地址
   - 参数：object：查看的对象
   - 返回值：对象在计算机的内存地址（十进制）

2.3.2.6 优先级

符号	描述	顺序
小括号	()	高
乘方	**	
符号运算符	+（正号） -（负号）	
乘除	* / // %	
加减	+ -	
比较运算符	== != > >= < <=	
身份运算符	is isnot	
逻辑非	not	
逻辑与	and	
逻辑或	or	低

3.语句

3.1 行

1. 物理行：开发者编写代码的行
2. 逻辑行：python解释器需要执行的指令

建议：

- 一个逻辑行在一个物理行上
- 如果一个物理行中使用多个逻辑行，用 ; 隔开

3.换行

如果逻辑行过长，可以使用换行

隐式换行：所有的括号的内容换行，包括（）[] {}

显式换行：通过换/折行符（\ 反斜杠）换行，必须放在一行的末尾，目的是告诉解释器，下一行也是本行的语句

```
# 4个物理行 4个逻辑行
# a = 1
# b = 2
# c = a + b
# print(c)

# 1个物理行 4个逻辑行
# a=1;b=2;c=a+b;print(c)

# 换行
# 换行符 \
# a = 1+\
```

```
#     2+\
#     3+4+\
#     5+6
# print(a)

# 括号
a = (1+2
     +3
     +4+5)
print(a)
print(type(a))
```

程序执行的三个结构：顺序结构、循环结构、选择结构

- 顺序结构：程序本来按照从上到下的顺序默认执行
- 选择结构：if
- 循环结构：while for

3.2 选择语句

3.2.1 if结构

- 格式

```
语句0
if 条件表达式:
    语句1
语句2
```

- 示例

```
gender = input("请输入性别：")
if gender == "男":
    print("你好")
    print("boy")
print("123")
```

3.2.2 if-else结构

- 格式

```
语句0
if 条件表达式:
    语句1
else:
    语句2
```

- 示例

```
gender = input("请输入性别：")
if gender == "男":
    print("你好")
    print("boy")
else:
    print("你好，girl")
print("123")
```

3.2.3 if-elif-else结构

- 格式

```
语句0
if 条件表达式1:
    语句1
elif 条件表达式2:
    语句2
elif ..

else:
    语句n
```

  - elif 子句可以有0个或多个
  - else可以有0个或1个，且放在if最后
- 示例

```
# if-elif-else
gender = input("请输入性别：")
if gender == "男":
    print("你好")
    print("boy")
elif gender == '女':
    print("你好，girl")
else:
    print("性别不详")
```

练习

1.输入整数，判断是正数、负数、零

```
# 输出正负
number = int(input("请输入一个整数："))
if number > 0:
    print("正数")
elif number < 0:
    print("负数")
else:
    print("0")
```

3.2.4 真值表达式

- 定义：条件表达式的布尔结果为真

```
if 100:
    print("真")

# 等同于
if bool(100):
    print("真")
```

练习

输入一个整数，如果是奇数，为变量state赋值"奇数"，反之“偶数”

```
# 真值表达式
number = int(input("请输入一个正整数："))
if number % 2:
    state = "奇数"
    print("state为", state)
else:
    state = "偶数"
    print("state为",state)
```

3.2.5 if语句嵌套

- 定义：在if语句中嵌套if语句

示例:

```
# if嵌套
number = int(input("请输入一个正整数："))

if number > 0 :
    state = "偶数"
    if number % 2 :
        state = "奇数"
    print("state为",state)
else:
    print("输入有误")
```

练习

1. 商品优惠活动：内容如下:

   如果是VIP，消费小于500，打85折，否则打8折

   如果非VIP，消费大于800，打9折，否则不打折

   根据用户输入的账户类型及消费的金额，打印折扣信息及折扣后的应付金额

   ```
   # 商品优惠活动：内容如下:
   # 如果是VIP，消费小于500，打85折，否则打8折
   # 如果非VIP，消费大于800，打9折，否则不打折
   ```

```
# 根据用户输入的账户类型及消费的金额，打印折扣信息及折扣后的应付金额

user_status = input("是否为VIP:")
money = int(input("请输入消费金额："))
discount = "不打折"
if user_status == "是":
    if money < 500:
        money *= 0.85
        discount = "85折"
    else:
        money *= 0.8
        discount = "8折"
else:
    if money > 800:
        money *= 0.9
        discount = "9折"
print("折扣：",discount,"应付金额：",money)
```

2. 模拟出租车计价器：

   起步价（3公里以内，包括3公里）：8元

   超过3公里，没有超过15公里，每公里 2.4 元

   超过15公里，每公里加收 2.4的50%

   要求：计算并打印应付的车费，小数部分保留2位数（可使用round函数）

```
km = float(input("请输入公里数："))
if km > 0:
    money = 8
    if 3 <= km <= 15:
        money += (km - 3) * 2.4
    elif km > 15:
        money += (km - 15)* (2.4 * 1.5) + 12 * 2.4
    print("行驶了：",km,"公里，应付：",round(money,2),"元")
else:
    print("输入有误")
```

3.2.6 if条件表达式

- 语法：变量 = 结果1 if 条件 else 结果2

```
value = 1 if input("请输入性别：")=="男" else 0
```

- 作用：根据条件（True、False）来决定返回结果1还是结果2
- 说明：逻辑判断复杂时会导致代码阅读性变差

练习

判断闰年

输入一个年份，如果是闰年，为变量day赋值为29，否则 28

闰年的条件：年份能被4整除但是不能被100整除，或者能被400整除

例如：

请输入年份：2020

输出：2020年2月有29天

要求：1.if-else语句 2.if条件表达式

```
# 闰年2月天数
year = int(input("请输入年份："))

if year > 0:
    # 方法1 if-else
    # if year % 4 == 0 and year % 100 != 0 or year % 400 == 0:
    #     day = 29
    # else:
    #     day = 28

    # 方法2 条件表达式
    day = 29 if year % 4 == 0 and year % 100 != 0 or year % 400 == 0 else 28

    print(year,"年的2月有：",day,"天")
else:
    print("输入有误")
```

3.3 循环语句

3.3.1 while 语句

- 作用：根据满足条件重复执行

3.3.1.1 while 结构

格式：

```
while 表达式：
    满足条件执行的语句
else:
    不满足条件执行的语句
```

说明：else子句可以省略

```
# while
count = 0
while count < 3:
    print(count)
    count += 1
else:
    print("结束")
```

3.3.1.2 while-if结构

格式：

```
while 表达式:
    if 条件表达式:
        满足条件执行的语句
    ...
else:
    不满足条件执行的语句
```

```
# while if
count = 3
i = 0
while i < count:
    if i == 0:
        print(count * "*")
    else:
        print("A",i)
    i += 1
else:
    print("结束",i)
```

练习

1. 10个亿，一天花一半，能花多少天（花完标准即钱小于1）

```
money = 1e9
days = 0
while money > 1:
    money /= 2
    days += 1
else:
    print("10个亿",days,"天花完")
```

2.打印1-20的整数，每行打印5个，打印4行：

1 2 3 4 5

6 7 8 9 10

....

```
x = 1
while x < 21:
    print(x,end='\t')
    if x % 5 == 0 :
        print()
    x += 1
```

3.3.2 for语句

- 作用：用来**遍历可迭代对象**的数据元素

可迭代对象：指能一次获取数据元素的对象，例如：容器类型 str等

遍历：指将可迭代对象中数据依次赋值给遍历

数据元素：可迭代对象中的每个数据

- 语法

```
for 变量列表 in 可迭代对象:
    循环体
else:
    语句
```

- 说明：
  - else子句可以省略，放到最后
  - 在循环体中用break终止循环时，else子句不执行

示例：输入整数，计算累加和

例如："1234"，输出10

```
# 字符串累加
str_number = input("请输入数字：")
sum_number = 0

for n in str_number:
    sum_number += int(n)
print(sum_number)
```

3.3.2.1 range函数

- 语法：**range(start,stop,step)**
- 作用：用来创建一个生成一系列整数的可迭代对象（也叫整数序列生成器）
- 参数：
  - start：起始值，可选，默认是0
  - stop：终止值，**无法获取**
  - step：步长，可选，默认为1，可正可负

    正值：正向取值（数据从小到大）

    负值：反向取值（数据从大到小）

    【注意】：
    - 1. 取值方向与步长方向不一致，则为空
      2. 起始值和终止值相同，结果为空

```
# range
# 1. range(开始，结束，步长)
# for item in range(1,3,1):
#     print(item)

# 2. range(开始，结束)
# for item in range(1,6):
#     print(item)

# 3. range(结束)
for item in range(3):
    print(item)
```

示例：

1. 打印10以内偶数（正反序）
2. 打印 0 -1 -2 -3 -4 -5

```
# 打印 偶数
# for i in range(0,11,2):
#     print(i,end=" ")

# for i in range(10,-1,-2):
#     print(i,end=" ")

# 打印 0 -1 -2 -3 -4 -5
for i in range(0,-6,-1):
    print(i,end=" ")
```

3.3.3 跳转语句

3.3.3.1 break

作用：用来终止当前循环语句的执行

说明：**while/for循环中执行了break语句，后面的代码不执行，else语句也不执行**

```
# 跳转语句
number = 3
for item in range(1,10):
    if number == item:
        break
    print("当前item：",item)
else:
    print("哈哈哈哈")
```

3.3.3.2 continue

作用：跳出本次循环，继续下次循环

```python
# 累加1-10能被3整除的数字
sum = 0
for item in range(1,11):
    if item % 3 != 0:
        continue
    sum += item
print(sum)
```

练习

1.输入数，判断该数是否为素数（2）

```python
# 素数
number = int(input("请输入一个数："))

if number >= 2:
    for i in range(2,number):
        if number % i == 0:
            print(number,"不是素数")
            break
    else:
        print(number,"是素数")
else:
    print("输入有误")
```

2.随机产生一个1到100的随机数，猜数字，猜对为止，每次提示：大了或者小了，打印猜的次数

```python
# 猜数字  随机产生一个1到100的随机数，猜数字，猜对为止，
# 每次提示：大了或者小了，打印猜的次数
import random

number = random.randint(1,100)
print(number)
times = 0

while True:
    user_number = int(input("请输入猜的数："))
    times += 1
    if user_number > number:
        print("大了")
    elif user_number < number:
        print("小了")
    else:
        print("猜对了")
        print("猜了",times,"次")
        break
```

3.模拟用户登录，规则如下：

设定初始用户名和密码为：张三 123456，限定 3 次用户名及密码验证机会，在三次机会内输入正确打印登录成功

在三次机会内：如果用户名和密码都输入错误，需要重新录入用户名密码；

如果用户正确，仅需验证密码，同时在3次机会内提示剩余多少次机会；

三次机会都失败，则打印账号被锁定，程序终止

```
# 模拟登录
init_user = "张三"
init_password = "123456"
times = 3

while times > 0 :
    user = input("请输入用户名:")
    password = input("请输入密码：")
    times -= 1

    if user == init_user and password == init_password:
        print("登录成功")
        break
    elif user == init_user:
        print("密码错误")
        print("还剩",times,"次机会")
        while times > 0:
            password = input("请输入密码：")
            times -= 1
            if password == init_password:
                print("登录成功")
                break
            else:
                print("密码错误")
            if times > 0:
                print("还剩", times, "次机会")
            else:
                print("账户被锁定")
        break
    else:
        print("用户名或密码错误")
    if times > 0:
        print("还剩", times, "次机会")
else:
    print("账户被锁定")
```

3.3.4 循环嵌套

1. while循环中嵌套while

```
语句0
while 表达式1:
    语句1
    while 表达式2:
        语句2
```

2. for循环中嵌套for

```
for 变量1 in 可迭代对象1:
    for 变量2 in 可迭代变量2:
        循环体
```

练习

1.生成99乘法表（正乘法表和倒乘法表）

```
# 乘法表
# 正乘法表
# for i in range(1,10):
#     for j in range(1,i+1):
#         print(j,"*",i,"=",i*j,end="\t")
#     print()


# 倒乘法表
for i in range(9,0,-1):
    for j in range(1,i+1):
        print(j,"*",i,"=",i*j,end="\t")
    print()
```

4. 容器

4.1 通用操作

4.1.2 数学运算符

- +：用来拼接两个容器
- +=：用原容器和右侧的容器拼接，并重新绑定变量
- *：重复生成容器元素
- *=：用原容器生成重复元素，并重新绑定变量
- < <= > >= == !=：依次比较两个容器中元素，不同则返回比较结果

适用于： 字符串 列表 元组

```
# 1.拼接
# name = "喜羊羊"
# name += "灰太狼"
# print(name)

# 2.重复元素
# name = "喜羊羊"
# name *= 2
# print(name)

# 3.比较
print("abcd">"abc")
```

练习

1.输入一个整数，作为边长，打印矩形

输入：5

输出：

```
$$$$$
$   $
$   $
$   $
$   $
$$$$$
```

```
# 打印矩形
side = int(input("请输入边长："))

if side > 0:
    for i in range(side):
        if i == 0 or i == side - 1:
            print("$"*side)
        else:
            print("$"+(side-2)*" "+"$")
```

2.输入整数，整数作为边长，打印菱形

输入：5

输出：

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

4:
 **
****
 **
```

abs(num)：绝对值函数

```
# 打印菱形
side = int(input("请输入边长："))

if side > 0:
    for x in range(-side//2+1,side//2+1):
        x = abs(x)
        blank = x * " "
        star = (side - 2 * x) * "*"
        print(blank+star)
```

4.1.3 成员运算符

- 语法：

  数据 in 序列

  数据 not in 序列

- 作用：如果在指定的序列中找到值，返回bool类型
- 使用：字符串、列表、元组、字典、集合容器

```
msg = "法外狂徒张三"
# print("张三" in msg)
# print("徒张" in msg)
# print("张" in msg)

print("狂三" in msg)
```

4.1.4 索引

- 作用：定位容器单个元素
- 语法：**容器[index]**
- 说明：
  - 正向索引：从0开始，第二个索引是1
  - 反向索引：从-1开始，-1表示最后一个，-2表示倒数第二个，以此类推

str	法	外	狂	徒	张	三
正向	0	1	2	3	4	5
反向	-6	-5	-4	-3	-2	-1

- 函数： len(str)

  作用：返回字符串长度

```
msg = "法外狂徒张三"
# len(str)
# print(msg[2])
# print(msg[-2])
# print(len(msg))
print(msg[6])
```

4.1.5 切片 slice

- 语法：sequence[start:stop:step]
- 作用：定位多个容器元素
- 参数：

- start：起始索引值，**可选**，默认是字符串最开头位置
- stop：结束索引值，**可选**，不含结束索引值，默认表示最末尾位置（表示切割到结尾）
- step：步长，**可选**，元素移动的方向和偏移量，默认是1
  - 正数，正向切片
  - 负数，反向切片

```
msg = "法外狂徒张三"
# print(msg[:])
# 特殊，翻转
# print(msg[::-1])
# print(msg[:-1]) # 法外狂徒张
# print(msg[2:5:1]) # 狂徒张
# print(msg[2:5:2]) # 狂张
# print(msg[2:-1:2]) # 狂张
# print(msg[3:]) # 徒张三
# print(msg[1:1]) # 空
print(msg[-2:2:-1]) # 张徒
```

示例：

1.检验回文

上海自来水来自海上

ABCCBA

```
# 检验回文
string = input("请输入文字：")

if string == string[::-1]:
    print("是回文")
else:
    print("不是回文")
```

4.1.6 内建函数

函数	描述
len(x)	返回序列的长度
max(x)	返回序列的最大值元素
min(x)	返回序列的最小值元素
sum(x)	返回序列中的所有元素的和（元素是数值类型）

```
# 内建函数
ms = [1,2,3,4,5.5]
print(len(ms))
print(max(ms))
print(min(ms))
print(sum(ms))
```

4.2 字符串

4.2.1 定义

- 有一系列的字符组成的 **不可变 有序** 的序列容器，存储的是字符的编码值
- 特点： 占用连续内存空间，按需存储（存多少字符开辟多大的存储空间）
- 应用：存储数字、字母、符号等

4.2.2 编码

相关函数：

ord(字符串)：返回该字符串的Unicode码

chr(整数)：返回该整数对应的字符串

```
# 1.编码
number = ord("a")
print(number)

char = chr(25105)
print(char)
```

示例：生成字符串： AaBbCc...Zz

```
str_letters = ""
for i in range(ord('A'),ord('Z')+1):
    str_letters += chr(i) + chr(i+32)
print(str_letters)
```

转义字符：

取消转义：

```
a = r"C:\python\test.py"
print(a)
```

4.2.3 格式化字符串

- 作用：生成一定格式的字符串

4.2.3.1 占位符

- 格式：
  - ’占位符‘%变量
  - ’占位符1,占位符2...‘%(变量1，变量2,..)

- 占位符
  - %s 接收字符串
  - %d 接收整数
  - %f 接收浮点数（默认保留到小数点后6位）
    - 保留n位： %.nf
- 修饰符
  - -:左对齐，字符串默认右对齐
  - %m:占m个字符宽度，若数据长度大于m，m值无效

```
# 2. 占位符
name = "Zhangsan"
age = 20
patio = 25.23131121

# print("我叫:%s,今年:%d岁，成绩排名:%f" % (name,age,patio))
# print("我叫:%s,今年:%d岁，成绩排名:%.2f%%" % (name,age,patio))
print("我叫:%10s,今年:%4d岁，成绩排名:%10.2f%%" % (name,age,patio))
print("我叫:%-10s,今年:%-4d岁，成绩排名:%-10.2f%%" % (name,age,patio))
```

4.2.3.2 format函数

- 格式：

  '{}{}...'.format(value1,value2,...)

  '{0}{1}...'.format(value1,value2,...)
- 3.6版本以上缩写
  - f'{变量名1}{变量名2}...'
  - f'{数据1}{数据2}...'

```
name = "Zhangsan"
age = 20
patio = 25.23131121
# print("我叫:{},今年:{}岁，成绩排名:{}%".format(name,age,patio))
print("我叫:{0},今年:{1}岁，成绩排名:{1}%".format(name,age,patio))

# 3.6以上版本缩写
print(f"我叫:{name},今年:{age}岁，成绩排名:{patio}%")
print(f"我叫:{'李四'},今年:{10}岁，成绩排名:{25.5}%")
```

练习

示例：进度条

```
import time

for i in range(1,101):
    # \r 返回光标到首行（覆盖）
    print('\r{} {}%'.format('>' * i,i),end="")
    time.sleep(0.1)
```

4.2.4 常用函数

- str.isdigit()
  - 功能：判断字符串中字符是否全是数字
  - 返回值：全是数字返回True，否则False
  - 场景：字符串是否为数字的判断
- str.upper()/lower()
  - 功能：将字符串中的英文字母字符转成大写/小写
  - 返回值：返回字符全是大写/小写的字符串结果
  - 场景：大小写转换
- str.strip(char=None)
  - 功能：去除字符串左右测的空白字符或指定字符
  - 参数：char：去除的字符
  - 返回值：去除后的字符串结果
  - 场景：误操作、数据清洗
- str.replace(old,new,count=-1)
  - 功能：将字符串中指定个数的旧字符替换为新字符
  - 参数：

    old：被替换的字符串

    new：用于替换old的字符串

    count：替换的个数，默认为-1表示全替换
  - 返回值：替换结果的字符串
  - 场景：替换字符
- str.count(sub,start,end)
  - 功能：返回字符串指定的索引范围内字符串出现的次数
  - 参数：

    sub：查找的子字符串

    start：开始查找的位置，可选，默认第一个字符，索引为0

    end：结束查找的位置，可选
  - 返回值：字符出现的次数

```
# 4.常用函数
string = "abcdeaabaa"
# string = "123"
# print(string.isdigit())
# print(string.upper())
# print(string.lower())
# print(string.strip())
# print(string.strip('a'))
# print(string.replace('ab','E',1))
print(string.count("ab",1,7))
```

4.3 列表 list

4.3.1 定义

- 由一系列变量组成的 **可变 有序**的序列容器（可存储任意类型数据）
- 特点：占用连续内存空间

4.3.2 基础操作

表示方式： [元素1，元素2,...]

1.创建列表：

- 列表名=[]
- 列表名=list(可迭代对象)

2.查看列表：

- 长度：len(列表)
- 存在： 元素 in/not in 列表
- 索引：列表名[index]
- 切片：列表名[[start]:[stop] [:step]]

3.添加元素

- 列表名.append(元素)
- 列表名.insert(索引，元素)
- 列表名.extend(可迭代对象)

4.修改元素

- 列表名[索引] = 容器
- 列表名[切片] = 容器
- 变量 = 列表名[切片] # 赋值给变量是切片创建的新列表

5.遍历

正向：从头到尾读取元素

```
for 变量名 in 列表名:
    # 变量名就是元素
```

正向：非从头到尾读取元素

```
for 索引名 in range(len(列表名))
    # 元素是 列表名[索引名]
```

反向：倒序读取

```
for 索引名 in range(len(列表名)-1,-1,-1):
    # 元素是 列表名[索引名]
```

6.删除元素

- 列表.remove(元素)
- del 列表[索引或切片]
- 列表.clear() 清空列表

7.列表排序

- 列表名.sort() -->升序

```
# list
# 1.创建
list_names = ["张三","李四","王五"]
list_ages = [29,22,30]

list_names2 = list("张三")
# print(list_names)
# print(list_ages)
# print(list_names2)

# 2.添加
# --追加 append
# list_names.append("六六")
# print(list_names)
# # -- 插入 insert
# list_names.insert(2,"七七")

# 3.定位
# -- 索引
# -- 读取
element = list_names[1]
# print(element)
# print(list_names)
# list_names[1] = "张四"
# print(list_names)

# -- 切片
# -- 读取
print(list_names)
# names=list_names[:2]
# print(names)
# -- 修改  通过切片修改，遍历右侧的数据，依次存入左侧
# list_names[:1] = ["AA","BB","CC"]
# list_names[:1] = 100  # 报错，因为100不能被遍历

# 4.遍历
# -- 方式1 从头到尾依次
# for name in list_names:
#     print(name)

# -- 方式2 非从头到尾
# for i in range(len(list_names)):
#     print(list_names[i])
# -- 方式3 倒序
# for i in range(len(list_names)-1,-1,-1):
#     print(list_names[i])
print(list_names)


# 5.删除
```

```
# -- 方式一
# list_names.remove("张三")
# print(list_names)

# -- 方式二
# del list_names[0]
del list_names[-1:]
print(list_names)
```

4.3.3 列表和字符串转换

1. 列表转字符串：
    - 变量名 = "连接符".join(列表)
2. 字符串转列表
    - 列表名 = “字符串”.split("分隔符")

```
# 6. 列表转字符串
# list01 = ["a","b","c"]
# result = "-".join(list01)
# print(result)

#7. 字符串转列表
str_data = "A,B,C"
list_result = str_data.split(",")
print(list_result)
```

4.3.4 列表推导式

- 定义：使用简易方法将可迭代对象转为列表
- 格式：变量名 = [表达式 for... if...]

```
list01 = [9,15,60,5,68,79]
# 需求：从list01挑出能被3整除的数字存到list02
# list02 = []
# for item in list01:
#     if item % 3 == 0 :
#         list02.append(item)

# list02 = [item for item in list01 if item % 3 == 0]
# print(list02)

# 需求：在list01将所有数字的个位存list03
# list03 = []
# for item in list01:
#     list03.append(item % 10)
list03=[item % 10 for item in list01]
```

```
print(list03)
```

4.4 元组 tuple

4.4.1 定义

- 由一系列变量组成的 **不可变 有序** 序列容器（可存储任意类型的数据）
- 特点：占用连续内存空间，按需分配空间
- 说明：不可变-一旦创建，不可以再添加/删除/修改元素

4.4.2 基础操作

表示方式：(元素1,元素2...)

1.创建空元组

- 元组名=()
- 元组名= tuple()

2.创建非空元组

- 元组名=(数据1,)
- 元组名=(数据1,数据2...)
- 元组名=数据1,数据2..
- 元组名=tuple(可迭代对象)

3.查看元组

- 长度：len(元组名)
- 存在：元素 in/not in 元组
- 索引：元组[index]
- 切片：元组[start,stop,step]

4.获取元素

- 变量名=元组名[索引]
- 变量名=元组名[切片]

5.遍历

```
正向
for 变量名 in 元组名:
    # 变量名就是元素

反向：
for 索引名 in range(len(元组名)-1,-1, -1)
```

```
# 元组
```

```
# 1.创建
tuple01 = (10,20,30)
list01 = ["a","b","c"]
# tuple01 = tuple(list01)
# tuple01[0] = 10
# print(tuple01)

# 2. 定位
# print(tuple01[0])
# print(tuple01[:2])

# 3.遍历
# for item in tuple01:
#     print(item)
# for i in range(len(tuple01)-1,-1,-1):
#     print(tuple01[i])

# 4.特殊
# tuple02 = 10,20,30
# print(tuple02)

# tuple02 = (10,)
# print(type(tuple02))
# print(tuple02)

# 拆包：多个变量 = 容器
# a,b,c = tuple01
# a,b = tuple01  # 报错
# a,b,c = ["A","B","C"]
# a,b,c = "喜羊羊"
a,*b,c="喜羊羊和灰太狼"
print(a)
print(b)
print(c)
```

4.5 字典 dict

4.5.1 定义

- 有一系列**键值对**组成的**可变 散列**的容器
- 特点：占用离散空间
- 散列：对键进行哈希运算，确定在内存的存储位置，每条数据存储无先后顺序
- 键必须唯一且不可变（字符串，元组，数字），值无限制

4.5.2 基础操作

表示方式：{键1:值1,键2:值2,...}

1.创建字典

- 字典名={键1:值1,键2:值2,...}
- 字典名=dict(可迭代对象)

2.查看字典

- 长度：len(字典)
- 存在： 键 in/not in 字典

3.添加/修改元素

语法：

字典名[键] = 数据

说明：

- 键存在，修改数据
- 键不存在，添加数据

4.获取元素

- 变量名=字典名[键]
- 变量名=字典.get(键)
- 变量名=字典.keys() 获取字典所有的键
- 变量名=字典.values() 获取字典所有的值
- 变量名=字典.items() 获取字典所有的键值对

5.遍历

```
for 键名 in 字典名:
    字典名[键名]

for 键名，值名 in 字典名.items():
    语句
```

6.删除

- del 字典名[键]
- 字典.pop(键) # 删除并返回键对应的值

```
# 1.创建
dict_xyy = {"name":"喜羊羊","age": 6,"gender":"男"}
dict_myy = {"name":"美羊羊","age": 5,"gender":"女"}
dict_lyy = {"name":"懒羊羊","age": 5,"gender":"男"}
# 列表转字典的格式要求：列表元素能够一分为二
# --dict([(,)])
list01 = [("name1","沸羊羊"),("name2","慢羊羊"),("names","暖羊羊")]
# print(dict(list01))
# print(dict_xyy)

# 2.添加
dict_xyy["IQ"] = 100
dict_xyy["age"] = 100
```

```
print(dict_xyy)

# 3.定位 字典名[键]
# print(dict_xyy["name"])

# 4.删除
# del dict_xyy["IQ"]
# print(dict_xyy)

# 5.遍历
# for key in dict_xyy:
#     print(key)

# for value in dict_xyy.values():
#     print(value)

for key,value in dict_xyy.items():
    print(key)
    print(value)
```

4.5.3 字典推导式

语法：

{键:值 for 变量 in 迭代对象 if 条件}

4.6 集合 set

4.6.1 定义

- 由一系列 **不重复** 的 **不可变类型变量**（元组/数字/字符串）组成的 **可变 散列**容器，相当于是只有键没有值的字典
- 特点：占用离散空间

4.6.2 基础操作

表示方式：{元素1,元素2,...}

1.创建

集合名=set()

集合名=set(可迭代对象)

集合名={元素1,元素2,...}

2.查看

- 长度：len(集合)
- 存在： 元素 in/not in 集合

3.添加

- 集合.add(元素)

4.删除

- 集合.remove(元素)
- 集合.discard(元素)

```
# 1.创建
set01 = {"喜羊羊","美羊羊","灰太狼"}
# list01 = ["喜羊羊","美羊羊","灰太狼"]
# set01 = set(list01)
print(set01)

# 2.添加
# set01.add("懒洋洋")
# set01.add("灰太狼")
# print(set01)

# 3.遍历
# for item in set01:
#     print(item)

# 4. 删除
if "灰太狼" in set01:
    set01.discard("灰太狼")

print(set01)
```

4.6.3 运算

1. 交集 &： 返回共同元素

2. 并集 | ： 返回不重复元素

3. 补集 - ： 返回只属于其中集合之一的元素

4. 补集 ^ : 返回两个集合不同的元素

```
# 5. 运算
s1 = {1,2,3}
s2 = {2,3,4}
# print(s1 & s2)
# print(s1 | s2)
# print(s1 - s2)
# print(s2 - s1)
print(s2 ^ s1)
print(s1 ^ s2)
```

5. 子集 < ： 判断一个集合的所有元素是否完全在另一个集合中

6. 超集 >：判断一个集合是否具有另一个集合的所有元素

```
s1 = {1,2,3}
s2 = {2,3}
print(s2 < s1)
print(s1 > s2)
```

7.相同或不同 == != :判断集合中的所有元素是否和另一个集合相同

```
s1 = {1,2,3}
s2 = {3,2,1}
print(s1 == s2)
print(s1 != s2)
```

4.7 固定集合 frozenset

- 一系列不重复的不可变类型变量组成的不可变 散列的无序容器

5.函数 function

5.1 定义

可以重复执行的具有某个特定功能的语句块

5.2 作用

提高代码的复用性 和 可维护性

5.3 基础语法

5.3.1 定义函数

```
def 函数名([形参]):
    '''文档字符串'''
    函数体
```

- def 关键字：全称的define 定义
- 函数名：命名规则和变量名相同
- 形参：可选
- 文档字符串：可选，描述函数的**功能、参数、返回值**
- 函数体：完成功能的语句

函数内的第一行语句建议使用文档字符串描述函数的功能与参数

5.3.2 调用函数

语法: 函数名([实际参数])

示例:

```
# 打印矩形
def print_star(number):
    for row in range(number):
        if row == 0 or row == number - 1:
            print("*"*number)
        else:
            print("*%s*"%(" "*(number - 2)))

number = int(input("请输入整数:"))
print_star(number)
```

5.3.3 返回值

定义:

- 函数定义者告诉调用者结果

语法:

```
def 函数名([形参]):
    '''文档字符串'''
    函数体
    return [数据]
```

说明:

- 在函数中没有return语句或者return语句后没有数据，相同于返回 None
- 函数的返回值可以有多个，即：元组

作用:

- 返回执行结果
- 中断函数执行

```
# 求和
def add(num1,num2):
    '''
     计算并返回2个数的和
     :param num1: int 计算的数1
     :param num2: int 计算的数2
     :return: int 计算结果
     '''
    result = num1 + num2
    return result

# 1.获取数据
num1 = int(input("输入第一个数:"))
num2 = int(input("输入第二个数:"))
result = add(num1,num2)
print(result)
```

5.3.4 跨函数调用

概念：在一个函数调用另一个函数

```
# 跨函数调用
def m1():
    print("喜羊羊")
    return "KO"

def m2(count):
    list01=[]
    for i in range(count):
        data = m1()
        list01.append(data)
    return list01

print(m2(3))
```

5.3.5 函数的内存分配

1.将函数的代码存储到代码区，函数体中的代码不执行

2.调用函数时，在内存中开辟空间（栈帧），存储函数内部定义的变量

3.函数调用完，栈帧立即被释放

不可变类型参数：

- 数值型 （整数、浮点数）
- 布尔值
- None
- 字符串
- 元组

可变类型参数：

- 列表
- 字典
- 集合

传参：

- 不可变类型的数据传参，函数内部不会改变原数据的值
- 可变类型的数据传参时，函数内部可以改变原数据

```
def m1(p1,p2):
    p1 = "张三"
    p2["李四"] += 50

a = "三"
b = {"李四":80}
m1(a,b)
print(a)
print(b)
```

```
def m1(p1,p2):
    # p1 = [100,200]
    p1[:] = [100,200]
    print(id(p1))

a = [10,20]
b = [30,40]
m1(a,b)
print(a)
print(id(a))
print(b)
```

5.3.6 作用域 scope

5.3.6.1 定义

变量起作用的范围

5.3.6.2 分类

LEGB

- Local：局部作用域，函数内部
- Enclosing：外部嵌套作用域，函数嵌套
- Global：全局作用域，模块（.py文件）内部
- Builtin：内置模块作用域：builtins.py文件

变量名查找原则：

从内向外：L -> E-> G -> B：在访问变量时，先查找本地变量，然后是包裹次函数外部的函数内部的变量，之后是全局变量，最后是内置变量

5.3.6.3 局部变量

定义：定义在函数内部的变量（形参也是局部变量）

作用范围：只能在函数内部使用

说明：调用函数时才被创建，函数结束自动销毁

5.3.6.4 全局变量

定义：定义在函数外部，模块内部的变量

作用范围：在整个模块（py文件）范围内访问

5.3.6.5 global语句

作用：

- 在函数内部修改为全局变量
- 在函数内部定义全局变量

语法：

- global 变量1,变量2...

说明：

- 在函数内，使用global语句声明，并为变量赋值，视为创建新的全局变量
- 不能先为要声明的变量赋值，再用global声明
- global语句声明的变量名不能出现在形参列表中
- 可变数据类型可以在函数内直接被修改，可以不用global语句声明

作用：在函数内部修改全局变量

```
# 全局变量
data02 = 20
data03 = [30]

def f1():
    # 局部作用域：函数内部
    # 局部变量：一个函数内
    data01 = 10
    print(data01)
    print(data02)
def f2():
    # 不能访问其他函数局部变量
    # print(data01)
    print(data02)
def f3():
    # 在局部作用域中不能修改全局变量
    # data02 = 200
    # global语句
    global  data02
    data02 = 200
def f4():
    data03[0] = 300

# f1()
# f2()
f3()
f4()
print(data02)
print(data03)
```

5.4 函数参数

5.4.1 实参传递方式 argument

作用：与形参如何对应

5.4.1.1 位置传参

定义：实参与形参的位置依次对应传递数据。

注意：实参的个数与形参的个数相同

```
def fun01(p1, p2, p3):
    print(p1)
    print(p2)
    print(p3)

# 位置实参:根据顺序与形参进行对应
fun01(1, 2, 3)
```

5.4.1.2 序列传参

定义：实参用 * 将序列拆解后与形参的位置依次对应传递数据。

```
def func01(p1, p2, p3):
    print(p1, p2, p3)

# 序列实参:拆,按照顺序与形参对应
list01 = [1, 2, 3]
name = "孙悟空"
tuple01 = (4, 5, 6)
# func01(list01)
func01(*list01)  # 拆  1, 2, 3
func01(*name)  # 拆  孙 悟 空
func01(*tuple01)  # 拆  4, 5, 6
```

5.4.1.3 关键字传参

定义：实参根据形参的名字进行对应传递数据。

```
def fun01(p1, p2, p3):
    print(p1)
    print(p2)
    print(p3)

# 关键字实参:根据名字与形参进行对应
fun01(p1=1, p2=2, p3=3)
fun01(p2=2, p1=1, p3=3)
```

5.4.1.4 字典关键字传参

(1) 定义：实参用 ** 将字典拆解后与形参的名字进行对应传递数据。

(2) 作用：配合形参的缺省参数，可以使调用者随意传参。

```
def func01(p1, p2, p3):
    print(p1, p2, p3)

# 字典实参:拆,按照名称与形参对应
dict01 = {"p2":"B","p1":"A","p3":"C"}
func01(**dict01)
```

5.4.1.5 实参传递优先级

优先级从高到低：

1. 位置传参、序列传参
2. 关键字传参、字典关键字传参

- **总结**

参数顺序	参数分类	实参传递方式
有序	按顺序传递	位置传参、序列传参
无序	按名字传递	关键字传参、字典关键字传参

参数数量	实参传递方式
少数	位置传参、关键字传参
多数	序列传参、字典关键字传参

5.4.2 形参定义方式 parameter

作用：限制（约束）实参

5.4.2.1 缺省形参

(1) 语法：

```
def 函数名(形参名1=默认实参1, 形参名2=默认实参2, ...):
        函数体
```

(2) 说明：

- 缺省参数可以有0、1或多个。
- 缺省参数必须自右至左依次存在，如果一个参数有缺省参数，则其右侧的所有参数都必须有缺省参数。

```
def func01(p1 =0, p2="", p3=100):
    print(p1)
    print(p2)
    print(p3)

func01(p2=2)
func01(p2=2,p3=3)
# 支持同时使用位置实参与关键字实参
func01(1,p3=3)
# 注意1:先位置实参,后关键字实参
# func01(p1 =1,2,3) # 错误
```

5.4.2.2 位置形参

定义：形参按照位置依次接收实参的数据。

语法：

```
def 函数名(形参名1, 形参名2, ...):
      函数体
```

5.4.2.3 命名关键字形参

1) 语法：

```
def 函数名(*args, 命名关键字形参1, 命名关键字形参2, ...):
       函数体

def 函数名(*, 命名关键字形参1, 命名关键字形参2, ...):
      函数体
```

(2) 作用：

- 强制实参使用关键字传参

```
# 命名关键字形参:
# 星号元组形参后面的位置形参
# 限制实参必须是关键字实参
def func01(*args, p1, p2):
    print(args)
    print(p1)
    print(p2)

func01(p1=1, p2=2)
func01(1, 2, 3, p1=1, p2=2)


def func02(p1, *, p2=0):
    print(p1)
    print(p2)

# 通常星号后面的命名关键字形参属于辅助参数,可选.
func02(1)
func02(1, p2=2)
```

5.4.2.4 星号元组形参

(1) 语法:

```
def 函数名(*元组形参名):
      函数体
```

(2) 作用:

- 可以将多个位置实参合并为一个元组【接收多余的位置实参】

(3) 说明:

- 一般命名为'args'，且形参列表中最多只能有一个

```
# 位置实参数量可以无限
def func01(*args):
    print(args)

func01()  # 空元组
func01(1, 2, 34)  # (1, 2, 34)
# 不支持关键字实参
# func01(args = 1,a=1)
```

5.4.2.5 双星号字典形参

(1) 语法:

```
def 函数名(**字典形参名):
        函数体
```

(2) 作用:

- 可以将多个关键字实参合并为一个字典【接收多余的关键字实参】

(3) 说明:

- 一般命名为'kwargs', 形参列表中最多只能有一个

```
# 关键字实参数量无限
def func01(**kwargs):
    print(kwargs) # {'a': 1, 'b': 2}

func01(a=1,b=2)
# func01(1,2,3) # 报错
```

5.4.2.6 形参定义优先级

- 位置形参 --> 星号元组形参 --> 命名关键字形参 --> 双星号字典形参-->缺省参数

- 总结

参数是否提供	形参定义方式
是	位置形参、命名关键字形参
否	缺省参数、星号元组形参、双星号字典形参

参数形式	形参定义方式
定长	位置形参、命名关键字形参
不定长	缺省参数、星号元组形参、双星号字典形参

6.面向对象

找到能够解决这个问题的对象

6.1 类和对象

类：用来描述具有相同属性或功能的对象集合

- 一个抽象的概念

对象：表示类的具体实例，属于某个类别的个体

类是创建对象的模板：

- 数据成员：表示事物的特征
- 方法成员：表示事物的功能

6.2 语法

6.2.1 定义类

```
class 类名:
    '''文档字符串'''
    类成员
    实例成员
```

说明：

- 类名：所有单词首字母大写

6.2.2 实例化对象

语法：

```
变量名 = 类名([参数])
```

说明：

- 变量名表示实例对象，类名()表示实例化
- 变量名存储的时实例化后的对象地址

6.3 实例成员

6.3.1 实例变量

语法:

- 定义： 对象.变量名
- 调用：对象.变量名
- 构造方法

```
class 类名:
    def __init__(self,数据):
        self.变量名 = 数据
```

- 说明1：
  - __ init __ 方法：构造方法或初始化方法，名字不可变
  - 在类实例化时自动调用
  - self：表示实例对象本身，用在构造方法或实例方法中被自动传递
  - 类名后面的参数安装构造方法形参传递
- 说明2：

  首次通过对象赋值是创建，再次赋值是修改

  通过定义在构造方法中

  每个对象存储一份，通过实例对象地址访问
- 查看：
  - 格式： 实例对象.__ dict __
  - 作用：查看实例对象的属性，用于存储自身实例变量的字典

6.3.2 实例方法

语法：

```
class 类名:
    def 方法名称(self,[参数]):
        方法体
```

调用：

```
对象.方法名（[参数]）
```

说明：

- 至少有一个形参，第一个参数绑定调用这个方法的实例对象，一般命名为self
- 无论创建多少对象，实例方法只有一份，并且被所有实例对象共享

本质：

- 定义在类内的“函数”

```
class Phone:
    # 构造方法
    def __init__(self,brand,price):
        self.brand = brand
        self.price = price

    # 实例方法
    def call(self):
        print(f"在用{self.brand}打电话")


hw = Phone('华为',5999)
hw.call()
# print(hw.brand)

mi = Phone('小米',29999)
mi.call()
```

练习

1.创建狗类，实例化两个对象并调用方法

- 数据：品种 kind、昵称nickname 、身长 bodyLength、体重 weight
- 行为：吃 eat（体重加1）

```
class Dog:
    def __init__(self,kind=None,nname=None,blen=None,weight=None):
        self.kind = kind
        self.nickname = nname
        self.bodyLength = blen
        self.weight = weight

    def eat(self):
        self.weight += 1
        print("{}的体重为：{}".format(self.nickname,self.weight))

jm = Dog("金毛","大毛",0.8,20)
print(jm.__dict__)
jm.eat()
jm.eat()
jm.eat()
print(jm.__dict__)
```

6.3.3 跨类调用

```
# 跨类调用
# 1.直接对象 资源浪费
class Person:
    def __init__(self,name=""):
        self.name = name

    def go_to(self,position):
        print("去",position)
        car = Car()
        car.run()

class Car:
    def run(self):
        print("小汽车滴滴滴~")

lz = Person("老张")
lz.go_to("东北")
```

```
# 2. 在构造中创建对象
class Person:
    def __init__(self,name=""):
        self.name = name
        self.car = Car()

    def go_to(self,position):
        print("去",position)
        self.car.run()

class Car:
    def run(self):
        print("小汽车滴滴滴~")

lz = Person("老张")
lz.go_to("东北")
# print(lz.__dict__)
```

```
# 3. 通过参数传递  灵活
class Person:
    def __init__(self,name=""):
        self.name = name

    def go_to(self,vehicle,position):
        print("去",position)
        vehicle.run()

class Car:
    def run(self):
        print("小汽车滴滴滴~")

lz = Person("老张")
bc = Car()
lz.go_to(bc,"东北")
```

练习

1.玩家攻击敌人，敌人掉血（根据玩家攻击力，减少敌人血量）

- 玩家 Player
  - 数据：攻击力 atk
  - 行为：攻击 attack(敌人)
- 敌人 Enemy
  - 数据：血条 hp
  - 行为：减血 hp < 0 程序终止 -hp，取决于ack

```
# 玩家攻击敌人，敌人掉血（根据玩家攻击力，减少敌人血量）
class Player:
    def __init__(self,atk):
        self.atk = atk

    def attack(self,enemy):
        print("开始攻击...")
        enemy.demage(self.atk)

class Enemy:
    def __init__(self,hp):
        self.hp = hp

    def demage(self,value):
        if self.hp <= 0:
            print("GG...")
            return
        self.hp -= value
        print("剩余血量:",self.hp)

p01 = Player(20)
e01 = Enemy(40)
p01.attack(e01)
p01.attack(e01)
p01.attack(e01)
```

2.模拟场景：

喜羊羊教灰太狼抓羊

灰太狼教喜羊羊放羊

```
class Animal:
    def __init__(self,name=None):
        self.name = name

    def tech(self,other,skill):
        print(f'{self.name} 教 {other.name} {skill}' )

xyy = Animal("喜羊羊")
htl = Animal("灰太狼")
xyy.tech(htl,"抓羊")
htl.tech(xyy,"放羊")
```

6.4 类成员

6.4.1 类变量

定义： 在类中，方法外

```
class 类名:
    变量名 = 数据
```

调用：

```
类名.变量名
```

特点：

- 随类的加载而创建
- 存在 **优先于** 实例对象
- 只有一份，被所有对象共享

作用：表达不同个体（实例对象）的共有数据

6.4.2 类方法

定义：

```
class 类名:
    @classmethod
    def 方法名(cls,[参数])
        方法体
```

调用：

```
类名.方法名([参数])
```

说明：

- 至少有一个形参，第一个形参用于绑定类，一般命名为cls
- 使用@classmethod修饰，目的是调用类方法时可以隐式传递类
- 类方法不能访问实例成员，实例方法中可以访问类成员

作用： 表达不同实例对象的相同行为

```
# 需求: 每开一家分行，总行的钱减少
class Bank:
    # 类变量
    total_money = 10000000

    # 构造方法
    def __init__(self,name,money):
        self.name = name
        self.money = money

    # 类方法
    @classmethod # 操作类变量
    def print_total_money(cls,value):
        # Bank.total_money -= self.money
        # Bank.total_money -= value
        print("类: ", cls)
        cls.total_money -= value
        print("总行的钱还剩: ",Bank.total_money,"元")

# 创建实例对象
b01 = Bank("庆阳支行",2000000)
# 调用类方法
Bank.print_total_money(b01.money)
# 访问类变量
print(Bank.total_money)
```

6.5 静态方法

定义:

```
class 类名:
    @staticmethod
    def 方法名([参数])
        方法体
```

调用:

```
类名.方法名([参数])
```

说明:

- 使用@staticmethod修饰，目的是该方法不需要隐式传递类参数
- 静态方法不能访问实例成员

```python
class Person:
    @staticmethod
    def info(name):
        print(name)

# p1 = Person("张三")
p1 = Person()
p1.info("李四")
# print(p1.__dict__)
```

7.三大特征

7.1 封装

7.1.1 数据角度

定义： 将一些基本数据类型复合成一个自定义类型

优点：

- 将数据与对数据的操作相关联
- 代码的阅读性更高

7.1.2 行为角度

定义： 向类外提供必要的功能，隐藏实现的细节

优点：简化编程，使用者不必了解具体的实现细节，只需要调用对外提供的功能

私有成员：

- 作用：无序向类外提供的成员，可以通过私有化进行屏蔽
- 语法：命名使用双下划线开头，例如：self. __ 属性名、def __ 方法名(self,参数)
- 名称会被修改为：

  ```
  _类名__成员名，可以通过 __dict__查看
  ```

属性 @property

- 作用：保护实例变量
- 说明：在写入方法中可实现对数据有效性验证
- 分类：

  ```python
  # 1.可读可写
  class 类名:
      def __init__(self,参数):
          self.属性名 = 参数

      @property
      def 属性名(self): # 读取方法
  ```

```
        return self.__属性名

    @属性名.setter  # 写入方法
    def 属性名(self,value):
        self.__属性名 = value

# 2.只读模式
class 类名:
    def __init__(self, 参数):
        self.属性名 = 参数

    @property
    def 属性名(self):  # 读取方法
        return self.__属性名

# 3.只写模式
class 类名:
    def __init__(self, 参数):
        self.属性名 = 参数

    @属性名.setter  # 写入方法
    def 属性名(self, value):
        self.__属性名 = value

    属性名 = property(fset=写入方法名)
```

- 调用：

```
对象.属性名 = 数据
变量 = 对象.属性名
```

练习

1. 创建敌人类，并保护数据在有效范围内

   数据：姓名、攻击力（0-100） 血量(0-500)

```
class Enemy:
    def __init__(self,name,atk,hp):
        self.name = name
        self.atk = atk
        self.hp = hp

    @property
    def atk(self):
        return self.__atk

    @atk.setter
    def atk(self,value):
        if 0 <= value <= 100:
            self.__atk = value
        else:
```

```
            raise Exception("攻击力不符合要求")

    @property
    def hp(self):
        return self.__hp

    @hp.setter
    def hp(self, value):
        if 0 <= value <= 500:
            self.__hp = value
        else:
            raise Exception("血量不符合要求")

e01 = Enemy("喜羊羊",10,2000)
print(e01.__dict__)
```

7.2 继承

定义：重用现有类的功能，并在此基础上进行扩展

7.2.1 继承方法

语法：

```
class 父类:
    def 父类方法(self):
        方法体

class 子类(父类):
    def 子类方法(self):
        方法体

子类对象名 = 子类()
子类对象名.子类方法()
子类对象名.父类方法()
```

说明：

- 子类直接拥有父类的方法

  子类 is a 父类

```
class Person:
    def say(self):
        print("说话")

class Student(Person):
    def study(self):
        self.say()
        print("好好学习，天天向上")

xm = Student()
xm.say()
xm.study()
```

- 内置函数

- isinstance(对象,类型)

  返回执行对象是否是某个类的对象

- type(对象)

  返回指定对象的类型

```
p1 = Person()

# 内置函数
# p1对象是 Person类的对象
print(isinstance(p1,Person))
print(isinstance(p1,Student))

print(isinstance(xm,Student))
# xm对象 是 Person的一种
print(isinstance(xm,Person))
```

7.2.2 继承数据

语法：

```
class 子类(父类):
    def __init__(self,父类参数,子类参数):
        super().__init__(参数) # 调用父类的构造方法
        self.实例变量 = 参数
```

说明：

- 子类无构造方法，将自动执行父类的构造方法（子类继承父类中定义的数据--实例变量）
- 子类有构造方法，则覆盖掉父类的构造方法，此时需要通过super()函数调用父类的构造方法，以确保子类实例变量被正常创建

```
class Car:
    def __init__(self,brand,speed):
        self.brand = brand
        self.speed = speed

class Electrocar(Car):
    def __init__(self,brand,speed,capacity):
        super().__init__(brand,speed)
        self.capacity = capacity

c1 = Electrocar("特斯拉",200,400)
print(c1.__dict__)
print(c1.brand,c1.speed,c1.capacity)
```

7.2.3 多继承

父类有多个 C++ 、Python

定义：一个子类继承两个或两个以上的基类，父类中的属性和方法同时被子类继承

同名方法顺序（MRO ）： 类自身 --> 父类继承列表（由左往右） --> 再上层父类

```
# 多继承
class A:
    def func01(self):
        print("A")
        # super().func01()

class B:
    def func01(self):
        print("B")


class C(A,B):
    def func01(self):
        print("C")
        # super().func01()


class D(A,B):
    def func01(self):
        print("D")
        # super().func01()


class E(C,D):
    def func01(self):
        print("E")
        super().func01()

# a = A()
# a.func01()
# b = B()
# b.func01()

# c = C()
# c.func01()

# d = D()
# d.func01()

e = E()
e.func01()
```

7.3 多态

- 字面含义：对于一种行为有不同的表现形态
- 定义：父类的同一种行为，在不同的子类上有不同的体现
- 作用：在继承的基础上，体现了子类的个性化

7.3.1 重写

概念：在子类中定义与父类中相同的方法

作用：改变父类的行为

```
class Shape:
    def draw(self):
        print("Shape的draw被调用")

class Point(Shape):
    def draw(self):
        print("正在画一个点")
class Circle(Point):
    def draw(self):
        print("正在画一个圆")

def my_draw(s):
    s.draw()

shapes1 = Circle()
shapes2 = Point()
my_draw(shapes1)
my_draw(shapes2)
```

7.3.2 内置方法

- 定义：Python中，以双下划线开头，双下划线结尾的是系统定义的成员
- __ str __:将对象转化成字符串

```
# 打印敌人对象 ： xx的攻击力是xx,血量是xx
class Enemy:
    def __init__(self,name='',atk=0,hp=0):
        self.name = name
        self.atk = atk
        self.hp = hp

    def __str__(self):
        return f'{self.name}的攻击力为:{self.atk}，血量为：{self.hp}'

e = Enemy("喜羊羊",100,200)
# print(e.__dict__)
print(e)
```

7.3.3 运算符重载

算术运算符重载

方法名	运算符和表达式	说明
`__add__(self, rhs)`	self + rhs	加法
`__sub__(self, rhs)`	self - rhs	减法
`__mul__(self, rhs)`	self * rhs	乘法
`__truediv__(self, rhs)`	self / rhs	除法
`__floordiv__(self, rhs)`	self // rhs	地板除
`__mod__(self, rhs)`	self % rhs	取模(求余)
`__pow__(self, rhs)`	self ** rhs	幂

```python
class Com:
    def __init__(self,value):
        self.value = value

    def __add__(self, other):
        lst = []
        for i in self.value:
            lst.append(i+other)
        return lst



c = Com([-1,3,5,3,6])
print(c+2)
```

增强运算符重载

方法名	运算符和复合赋值语句	说明
`__iadd__(self, rhs)`	self += rhs	加法
`__isub__(self, rhs)`	self -= rhs	减法
`__imul__(self, rhs)`	self *= rhs	乘法
`__itruediv__(self, rhs)`	self /= rhs	除法
`__ifloordiv__(self, rhs)`	self //= rhs	地板除
`__imod__(self, rhs)`	self %= rhs	取模(求余)
`__ipow__(self, rhs)`	self **= rhs	幂

比较运算符重载

方法名	运算符和复合赋值语句	说明
`__lt__(self, rhs)`	self < rhs	小于
`__le__(self, rhs)`	self <= rhs	小于等于
`__gt__(self, rhs)`	self > rhs	大于
`__ge__(self, rhs)`	self >= rhs	大于等于
`__eq__(self, rhs)`	self == rhs	等于
`__ne__(self, rhs)`	self != rhs	不等于

8.程序结构

8.1 模块 Module

8.1.1 定义

- 包含一系列的数据、函数、类的文件，通常以**.py**结尾

8.1.2 作用

让相关的数据、函数、类 有逻辑的组织在一起，有利于开发

8.1.3 导入

8.1.3.1 import

语法:

```
import 模块名
import 模块名 as 别名
```

作用： 将模块整体导入到当前模块

使用： 模块名.成员

原理：创建变量名记录文件地址，使用时通过变量名访问文件中的成员

8.1.3.2 from import

语法:

```
from 模块名 import 成员名
from 模块名 import 成员名 as 别名
from 模块名 import *
```

作用： 将模块内的成员导入到当前模块作用域中

使用： 直接使用成员名

原理：将模块的成员加入到当前模块的作用域中

mod_ex.py:

```
data = 200

def func01():
    print("func01开始执行...")

class MyClass:
    def func02(self):
        print("fun02开始执行")

    @classmethod
    def func03(cls):
        print("fun03开始执行")
```

model.py

```
# from mod_ex import data
# from mod_ex import func01
# from mod_ex import MyClass

import mod_ex

print(mod_ex.data)
# func01()
# MyClass.func03()
```

8.1.4 加载过程

- 在模块导入时，模块的所有语句会执行
- 如果一个模块已经导入，再次导入时不会重复执行模块内语句

注意：

- 模块之间不能循环调用
- 模块名不能以数字开头
- 模块名避免与内置或第三方模块同名

8.1.5 模块变量

- __ file __：查看模块对应的文件路径名
- __ name __：模块自身名，可以判断是否为主模块

- 此模块作为主模块（第一个运行的模块）运行时，_ *name* __绑定是* _main __'，不是主模块名，而是被其他模块到日时，存储模块名
- `if __name__ == '__main__'`，用来表明当前为程序的执行入口

8.1.6 模块分类

内置模块（builtins），在解析器内部直接使用

标准库模块，在python时已安装可以直接使用

第三方模块，需要自己安装

8.1.7 常用模块

time

关于时间的相关操作

```
# 导入
import  time

# 1.时间戳 1970.1.1到现在的秒数
print(time.time())

# 2. 时间元组
print(time.localtime())
print(time.localtime().tm_yday)
print(time.localtime()[-2])

# 3.时间字符串
print(time.strftime("%Y-%m-%d %H:%M:%S %A "))

# 4.时间字符串 --> 时间元组
print(time.strptime("2008/8/8 20:08:08","%Y/%m/%d %H:%M:%S"))

# 5.休眠 单位秒
time.sleep(3)
```

练习

定义函数，根据年月日 计算星期几

print(get_week(2023,3,22))

输出：星期三

```python
# 示例：根据年月日，计算星期
import time
def get_week(year,month,day):
    if year > 0 and 0 < month < 13 and 0 < day < 32:
        # 1. 将年月日转成时间元组
        time_tuple = time.strptime('%d%d%d'% (year,month,day),'%Y%m%d')
        # print(time_tuple)
        # 2. 根据时间元组中的参数对应星期
        week_tuple = ('星期一','星期二','星期三','星期四','星期五','星期六','星期日')
        return week_tuple[time_tuple[-3]]


print(get_week(2023,3,23))
```

random

关于随机数的操作

random .shuffle()：打乱顺序

```python
# random -生成随机数
import random

# 生成随机整数
# 范围[1,10]
# print(random.randint(1,10))
# 范围 range(1,10,2)
# print(random.randrange(1,10,2))

# 生成随机浮点数
# 范围[0,1)
# print(random.random())
# 范围[2,4]
# print(random.uniform(2,4))

# 从序列中随机返回元素
# list01 = [1,2,3,6,8,9]
# 随机返回一个
# print(random.choice(list01))
# 返回多个不重复的列表元素
# print(random.sample(list01,2))

# seed 随机数种子
random.seed(1)
print(random.random())
random.seed(1)
print(random.random())
random.seed(2)
print(random.random())
random.seed(2)
print(random.random())
```

练习

1.斗地主，规则：

- 生成 54 扑克牌存到列表 list_poker 中

  花色：

  - 黑桃 \u2660
  - 梅花 \u2663
  - 方块 \u2666
  - 红桃 \u2665

  大小王： \u2655-皇冠 大王 \u2654-王冠 小王

  组成： （A/J/Q/K + 2 ~ 10）* 花色 + 大小王

- 依次换行打印3个玩家手上的牌（17张） 和 3张底牌

要求：

1. 每次执行每位玩家手里的牌不同
2. 用类+random实现

```
# 斗地主发牌
import random

class Fight:
    def __init__(self):
        self.__color_poker = ('\u2660','\u2663','\u2666','\u2665')
        self.__poker = []

    def __generate_poker(self):
        for color in self.__color_poker:
            for i in range(2,11):
                self.__poker.append(color + str(i))
            for char in 'AJQK':
                self.__poker.append(color + char)
        self.__poker.extend(['\u2655','\u2654'])
        # 打乱牌
        random.shuffle(self.__poker)
        print(len(self.__poker))

    def __send_poker(self):
        print("开始发牌...")
        print("玩家1：",self.__poker[:51:3])
        print("玩家2：",self.__poker[1:51:3])
        print("玩家2：",self.__poker[2:51:3])
        print("底牌：",self.__poker[51:])

    def main(self):
        self.__generate_poker()
        self.__send_poker()

if __name__ == '__main__':
    ddz = Fight()
    ddz.main()
```

8.2 包 package

将模块以文件夹的形式进行分组管理

作用：将相关模块组织在一起，使逻辑结构更加清洗

8.2.1 导入

8.2.1.1 import

语法:

```
import 包名.模块名 [as 别名]
import 包名.子包名.模块名 [as 别名]
```

作用：优先将包中__ init __模块内整体导入到当前模块中

使用： 包名.模块名.成员

8.2.2.2 from import

语法:

```
from 包名 import 模块名
from 包名.模块名 import 成员名 as 别名
from 包名.子包.模块名 import 成员
from 包名.模块名 import *
from 包名.子包.模块名 import *
```

作用：优先将包中__ init __模块内整体导入到当前模块中

使用： 直接用模块名或成员名

8.2.2 __ init __模块

- 说明：在python包被导入时，会优先执行__ init __模块
- 作用：
    - 标识当前文件夹是Python 包
    - 用于限定被导入的模块(__ all __ = ["被导入的模块名1","被导入的模块名2",...])

```
# 结构
package01
    modu01.py
    modu02.py


## 定义在init模块

print("执行了 init 模块")

# 限定导入的模块
__all__ = ['modu01']
```

packageDemo.py

```
# 包

from package01 import *

print(modu01.name)
# print(modu02.name)
```

9.异常处理

定义：运行时检测到的错误

现象：当异常发生时，程序不会想往下执行，而转到函数的调用语句

常见异常：

异常类型	含义	描述
NameError	名称异常	变量未定义
TypeError	类型异常	不同类型数据进行运算
IndexError	索引异常	超出索引位置
AttributeErrror	属性异常	对象没有对应名称的属性
KeyError	键异常	没有对应名称的键
Exception	异常基类	包含程序中常见所有异常

9.1 处理

语法：

```
try:
    可能出现异常的语句
except 错误类型1 [as 变量1]:
    处理语句1
except 错误类型2 [as 变量2]:
    处理语句2
except Exception [as 变量3]
    不是以上异常类型的处理语句
else:
    未发生异常的语句
finally:
    无论是否发生异常的语句
```

作用：将程序由异常状态转为正常流程

说明：

- as 子句用于绑定错误对象的变量，可以省略
- except子句可以有一个或多个，用于捕获某种类型的错误
- else子句最多只有一个
- finally子句最多只有一个，如果没有except子句，需要存在
- 如果异常没有被捕获到，会向上层（调用处）继续传递，直到程序终止运行

示例：

输入int类型成绩，如果格式不正确，重新输入

效果： score = get_score()

print（"成绩是： %d"%score）

```
def get_score():
    while True:
        try:
            score = int(input('请输入成绩：'))
            if score > 0 :
                return score
        except Exception:
            pass
score = get_score()
print("成绩为：%d" % score)
```

9.2 raise语句

作用： 抛出一个错误，让程序进入异常状态

目的：在程序调用层次比较深时，向主调函数传递错误信息要层层return比较麻烦，所以认为抛出异常，直接传递错误信息

```
# def get_score():
#     while True:
#         try:
```

```
#             score = int(input('请输入成绩：'))
#             if score > 0 :
#                 return score
#         except Exception:
#             pass
# score = get_score()
# print("成绩为：%d" % score)


class Person:
    def __init__(self,age):
        self.age = age

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self,value):
        if 18 <= value <= 65:
            self.__age = value
        else:
            # 创建异常 -- 抛出错误信息
            raise Exception("不再年轻","if 18<=age<=65",1000)

while True:
    try:
        age = int(input("输入年龄："))
        p01 = Person(age)
        break
    except Exception as e:
        print(e.args)
```