python的语法基础

python的语法基础

一、计算机语言的发展与分类

测试开发：核心是"测试"能力，'开发'的目的是能利用技术手段更好的服务好测试，与其说测试开发是一种"新岗位"，不如说测试开发是对当今测试行业从业者提出新能力的要求

这节课主要是理论，听和了解就可以。

1.1 程序语言的介绍

编程语言（programming language）可以简单的理解为一种计算机和人都能识别的语言。一种计算机语言让程序员能够准确地定义计算机所需要使用的数据，并精确地定义在不同情况下所应当采取的行动。

- 计算机编程语言：Java、C、C++、PHP、Python、C#（C sharp）、HTML、SQL、Ruby、Go、易语言、汇编等
- https://www.tiobe.com/tiobe-index/

1.2 计算机编程语言的发展

1. 机器语言：计算机只能识别二进制（计算机内部的元件通过高低电压来表示信息，高电压是1，低电压是0），早期编程用二进制实现，比如：10100010；
2. 汇编语言：通过指令集来表示具体的操作，不同硬件的指令集不同，程序可移植性差；
3. 高级语言：跟人的自然语言接近，易于理解，比如C/C++/Java/Python等；

1.3 语言的分类

1.3.1 按照程序的执行方式

- 编译型
  - 编译型语言以C/C++为代表；
  - 编译型语言的程序在运行前需要先编译成机器语言，机器语言能够被计算机识别，因此不需要解释就直接运行；C语言的编译器有GCC，C++的编译器有G ++；
- 解释型
  - 解释型语言以Python/Ruby为代表
  - 解释型语言的程序不需要编译，程序在运行时才翻译成机器语言，每执 行一次都要翻译一次

    注：Java语言较为特殊，先把.java文件编译成.class字节码文件（十六进制），然后再由JVM解释运行，所以Java既是编译型也是解释型；
- 优劣势对比：编译型语言的优势在于运行效率更高，对系统的资源要求更低，一般来说编译型语言用于实现后台的复杂逻辑，比如MySQL数据库、Nginx服务器、CPython等都是用C语言编写的；解释型语言的优势在于跨平台性更好（相对而言编译型的可移植性差，C语言程序进行移植后，要重新编译），经常用于脚本的开发。

1.3.2 按照程序的设计思想来分

- 面向过程
  - 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
  - 举例：蛋炒饭
  - 代表语言：C语言
  - 应用场景：
    - 适合编写系统软件：编译器，JVM，驱动，操作系统内核
    - 嵌入式设备的编程
- 面向对象
  - 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。(简单说面向对象是完成一件事情，只需要找到某个或某些对象，一一组装完成即可)
  - 举例：盖浇饭
  - 代表语言：Java ,C++,C#,PHP,JavaScript,Python
  - Java
    - 面向对象的语言
    - 跨平台
    - web开发，主流SSH
    - Android应用开发
    - 服务端应用接口开发
    - 微信公众号开发
  - C++
    - 主要用于底层代码的开发（比如一些大型的游戏lol）
    - 一个不完全面向对象的编程语言，也可以面向过程
    - 之前用作MFC界面的开发
    - 现在更多用于数据计算
    - 数据仓库的开发
  - C#

        - 读音：Csharp
        - 微软公司的面向对象的语法
        - 运行在.NET framework
        - 主要开发Windows桌面应用，Windows store 应用
        - 配合asp.net，开发基于Windows Server服务器的web应用
    - PHP
        - 脚本语言，慢慢走向标准化面向对象语言
        - 主要开发动态网页
        - web开发
    - JavaScript
        - 和Java没有关系
        - 脚本语言
        - 主要用作网页的交互以及动效
    - Python
        - 脚本语言
        - 语法很简单，优美
        - 可以做web开发
        - 数据计算开发
        - 非常适合做自动化测试
- 面向过程语言和面向对象语言优缺点对比
    - 面向过程：
        - 优点：性能比面向对象高，因为面向对象的类调用时需要实例化；比如单片机，嵌入式开发、linux/unix等一般采用面向过程开发，性能是最重要的因素。
        - 缺点：没有面向对象易维护、易复用、易扩展。
    - 面向对象：
        - 优点：易维护、易复用、易扩展，由于面向对象有封装、继承、多态的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护
        - 缺点：性能比面向过程低。

1.4 Python语言

- 什么是Python?

- 怎么来的?
    - Python的创始人为荷兰人吉多·范罗苏姆[3] （Guido van Rossum）。1989年圣诞节期间，在阿姆斯特丹，Guido为了打发圣诞节的无趣，决心开发一个新的脚本解释程序，作为ABC 语言的一种继承。
- 为什么取名Python?

- 之所以选中Python（大蟒蛇的意思）作为该编程语言的名字，是因为他是一个叫Monty Python的喜剧团体的爱好者。
- 

1.5 Python的特点

- 简单易学
- 开源免费
- 高级语言
- 解释型语言
- 可移植性好
- 面向对象
- 强大的功能
- 可扩展性

1.5.1 简单易学

- Python 是一种代表简单注意思想的语言，阅读一个良好的 Python 程序，即使是在 Python 语法要求非常严格的大环境下，给人的感觉也像是在读英语段落一样。
- 换句话说，Python 编程语言最大的优点之一，是其具有伪代码的特质，它可以让我们在开发 Python 程序时，专注于解决问题，而不是搞明白语言本身。

1.5.2 开源免费

- Python 是 FLOSS（自由/开源源码软件）之一，简单地理解就是，用户使用 Python 进行开发和发布自己编写的程序，不需要支付任何费用，也不用担心版权问题，即使作为商业用途，Python 也是免费的。
- 开源正在成为软件行业的一种发展趋势，现在有很多商业软件公司都开始将自己的产品变成开源的（例如 Java）。也许，Python 的开源正是它如此优秀的原因之一，因为会有这么一群人，他们希望看到一个更加优秀的 Python，从而为了这个目标，不断地对 Python 进行创造，不断地改进。
- 很多优秀的软件都是开源免费的，比如CentOS、MySQL、Nginx、Tomcat、JMeter等，开源的软件企业喜欢用，开源的更安全，开源的软件用户越多生态越好也越来越优秀。

1.5.3 高级语言

- Python 是高级语言，因此当使用 Python 语言编写程序时，我们无需再考虑一些底层细节方面的问题。例如，如何管理程序使用的内存等等。

1.5.4 解释型语言

- 一个用编译型语言（如 C 或 C++）写的程序，可以从源文件转换到一个计算机使用的语言。这个过程主要通过编译器完成。当运行程序的时候，我们可以把程序从硬盘复制到内存中并且运行。
- 而 Python 语言写的程序，则不需要编译成二进制代码，可以直接从源代码运行程序。在计算机内部，由 Python 解释器把源代码转换成字节码的中间形式，然后再把它翻译成计算机使用的机器语言并运行。
- 事实上，由于不再担心如何编译程序，使得使用 Python 变得更加简单，我们只需要将 Python 程序复制到另外一台计算机上，它就可以工作了。因此，Python 程序更加易于移植。

1.5.5 可移植性

- ”胶水语言“，由于 Python 是开源的，它已经被移植到许多平台上。如果能够避免使用依赖系统的特性，那就意味着，所有 Python 程序都无需修改就可以在好多平台上运行，包括 Linux 、Windows、FreeBSD、Solaris 等等，甚至还有 PocketPC、Symbian 以及 Google 基于 Linux 开发的 Android 平台。
- 解释型语言几乎天生就是跨平台的。Python 作为一门解释型的语言，它天生具有跨平台的特征，只要为平台提供了相应的 Python 解释器，Python 就可以在该平台上运行。

1.5.6 面向对象

- Python 既支持面向过程编程，也支持面向对象编程。在“面向过程”的语言中（如 C 语言），程序仅仅是由可重用代码的函数构建起来的；而在“面向对象”的语言（如 C++）中，程序是由数据和功能组合而成的对象构建起来的。
- 与其他编程语言（如 C++ 和 Java）相比，Python 是以一种非常强大，而又简单的方式实现的面向对象编程。

1.5.7 强大的功能

- Python 强大的功能也许才是很多用户支持 Python 的最重要的原因，从字符串处理到复杂的 3D 图形编程，Python 借助扩展模块都可以轻松完成。
- 实际上，Python 的核心模块已经提供了足够强大的功能，使用 Python 精心设计的内置对象可以完成许多功能强大的操作。
  此外，Python 的社区也很发达，即使一些小众的应用场景，Python 往往也有对应的开源模块来提供解决方案。
- 在Python中，几乎你要做的每件事情都有对应的库和模块来实现，其实我们学Python就是学习基础语法和各种模块的使用，比如我们学Web自动化就是学Selenium库，学接口自动化就是学requests库等等。你如果要掌握所有的Python库，可能你一辈子都学不完。

1.5.8 可扩展性

- Python 的可扩展性体现为它的模块，Python 具有脚本语言中最丰富和强大的类和库，这些类和库覆盖了文件 I/O、GUI、网络编程、数据库访问、文本操作等绝大部分应用场景。
- Python 可扩展性的一个最好的体现是，当我们需要一段关键代码运行的更快时，可以将其用 C 或 C++ 语言编写，然后在 Python 程序中使用它们即可。

1.5.9 Python的弱点

- **速度慢**：Python 程序比 Java、C、C++ 等程序的运行效率都要慢。
- **源代码加密困难**：不像编译型语言的源程序会被编译成目标程序，Python 直接运行源程序，因此对源代码加密比较困难。

  其实，这两个缺点并不是什么大问题，首先，由于目前计算机的硬件速度越来越快，软件工程往往更关注开发过程的效率和可靠性，而不是软件的运行效率；至于第二个问题就更不是问题了，现在软件行业的大势本就是开源，就像 Java 程序同样很容易反编译，但丝毫不会影响它的流行。

1.6 Python能干什么?

- web应用开发（代表）
- 自动化运维
- 人工智能领域
- 网络爬虫
- 科学计算
- 游戏开发
- 除此之外，python可以直接调用Open GL实现3D绘制，这是高性能游戏引擎的技术基础。事实上，有很多python语言实现的游戏引擎，例如：Pygame，Pyglet以及Cocos 2d等

1.7 测试人员选择Python的理由

- 测试人员的编程能力相对较弱，而Python作为一种脚本语言，不仅功能强大，而且语法优美，支持多种自动化工具，而且学习上手比较容易；
- 高级语言，不用考虑底层逻辑，不用研究语言的本身，专注于解决问题；
- Python的社区发展比较好，有着非常多的文档和支持库；
- 对于一定编程基础的人员，使用Python作为自动化测试的语言可以非常流畅，几乎没有学习成本。

当前时代，作为测试，在面试中掌握Python和Java之一就足够了。

二、Python的环境和简单入门（提前安装）

2.1 Python的版本

Python有两个分支：Python2.x和Python3.x，这两个分支在很长一段时间是分别维护的，两个分支在语法上有差异，在工作中可能会遇到有Python2.x写的一些代码。

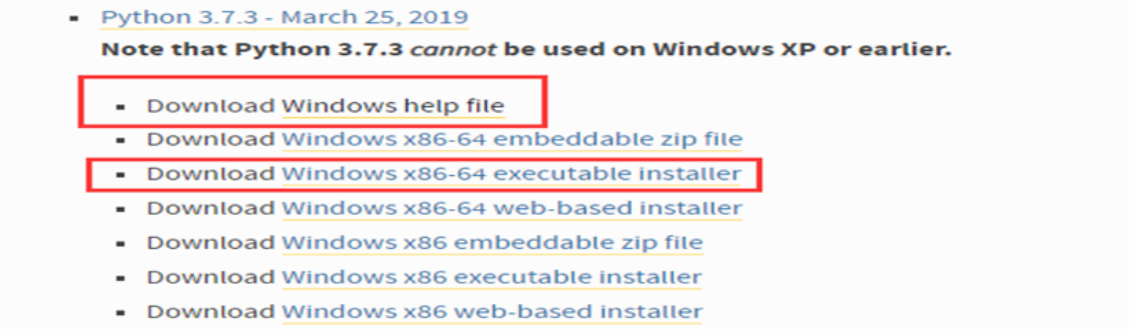
- Python2.x：最终版本为2.7，在2020年1月1日停止维护，用户会逐步切换到Python3.x；
- Python3.x：最新版本是Python3.10.4

2.2 Python解释器的种类

简单地说，Python是一门编程语言，任何一种编程语言都需要用另一种语言来实现它，比如C语言就是用机器语言来实现的。所以，Python根据实现方式不同分为了CPyhton、PyPy、Jython等。

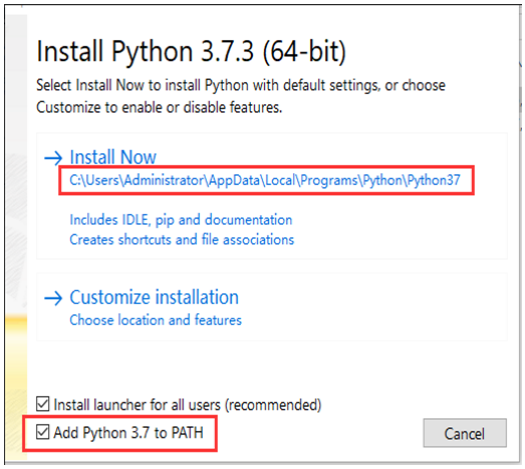
- Cpython：这是官方版本的解释器，这个解释器是用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。CPython是使用最广的Python解释器。python的解释器就是python.exe,其实就是应用程序——解释代码（是用来解释运行你编写的Python代码的）
- Jython：Jython是用Java实现运行在JVM上的解释器。
- IPython：IPython是用Microsoft.NET CLR实现的解释器。
- PyPy：PyPy是用Python实现的Python解释器。

2.3 Python的安装

- 在Linux上，自带了Python2.7的环境，原因是Linux的某些服务是用Python编写的，这个自带的Python环境千万不能轻易删除，比如我们常用的yum安装。——我们现在大部分情况下都是用的python3.X以上,主要就是你们以后买资料或者在网上查询资料的时候注意不要是python2.X版本编写的
- 我们平时编写和调试代码是在Windows电脑上写，所以需要在Windows上安装Python环境。
- 认识python官网：https://www.python.org/
- 

- 安装的路径，建议用默认路径（默认是C:\Users\Administrator\AppData\Local\Programs\Python\Python37）

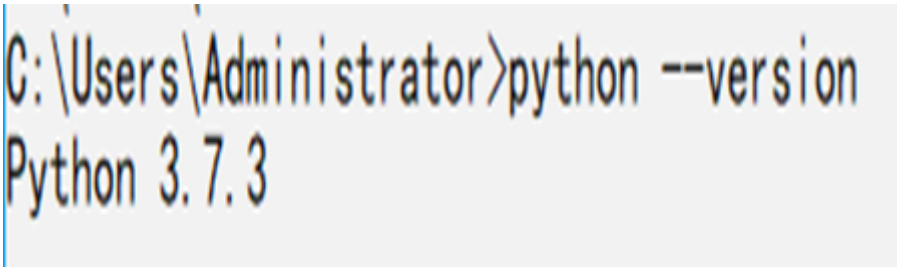
  注意：不要安装在中文路径中（如果路径中有中文的就自定义安装）
- 如图勾选默认



如果没有勾选添加环境变量，则需要手动把python的安装目录及安装目录下的Scripts目录添加到PATH变量中。也就是：C:\Users\Administrator\AppData\Local\Programs\Python\Python37\和C:\Users\Administrator\AppData\Local\Programs\Python\Python37\Scripts\

检查是否安装完成

在dos窗口输入python -V回车，如果返回有python的相关信息，则安装成功



思考：2.X和3.X版本是否能同时使用?

2.4 Python集成开发环境（IDE)的选择

- IDE：Integrated Development Environment，集成开发环境是用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面等工具。
- IDE的想法是把各种命令行的开发工具结合起来，提供一个抽象化的工具，来减少学习编程语言的时间，增加开发人员的生产力，同时也将各种开发工作做更密切的整合，来提高生产力。
- 其它的一些常用编译器：
    - Eclipse：常用的Java IDE
    - IntelliJ IDEA：常用的Java IDE
    - IDLE：Python自带的IDE
    - Vscode：Shell脚本的IDE
    - SQLyog/Navicate：数据库IDE

2.5 集成开发环境（PyCharm的安装与配置——PyCharm2022.1社区版）

- 下载地址：https://www.jetbrains.com
- 选择理由：
    - 图形化的智能代码提示和补全功能
    - 可以轻松查看方法和类
    - 语法高亮
    - 提供调试的工具
    - 源代码版本管理工具的支持

2.6 Python标识符

命名规则：所有需要自己命名的地方都要遵守以下规则

1. 可以包含数字、字母、_，但是不能以数字开头；
2. 标识符不能是Python中的关键字（保留字），也不建议使用python中的函数名作为标识符，但可以把关键字（保留字）作为标识符的一部分；

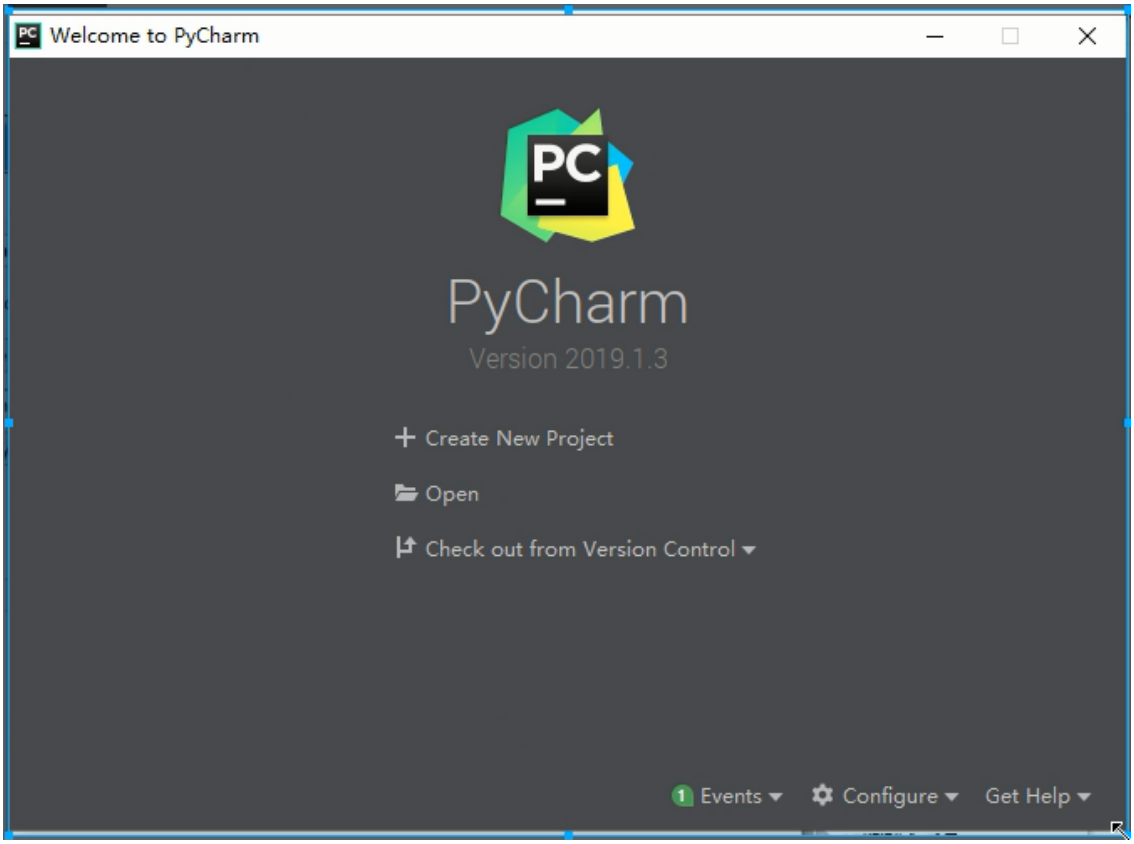
举例：如下哪些标识符是合法的?

1. 小明
2. @#￥%&.....*（
3. 1003hello
4. hello world
5. python、def、class、abc，

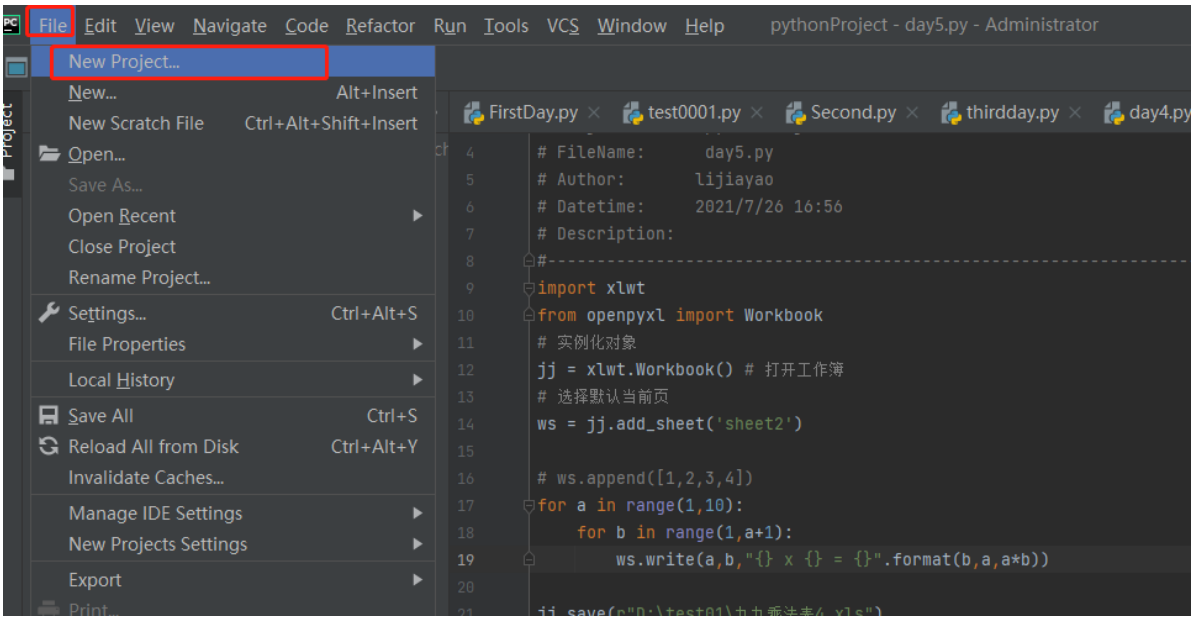
建议标识符命名规范：模块名全小写+下划线，类名大驼峰，方法和变量小写+下划线连接，常量大写，变量用名词，方法用动词，一般以英文单词命名，不能随意地写abc等名字，在一个项目组里，一定要有统一的规范，这样代码的可阅读性才会更好。

2.7 PyCharm的使用入门

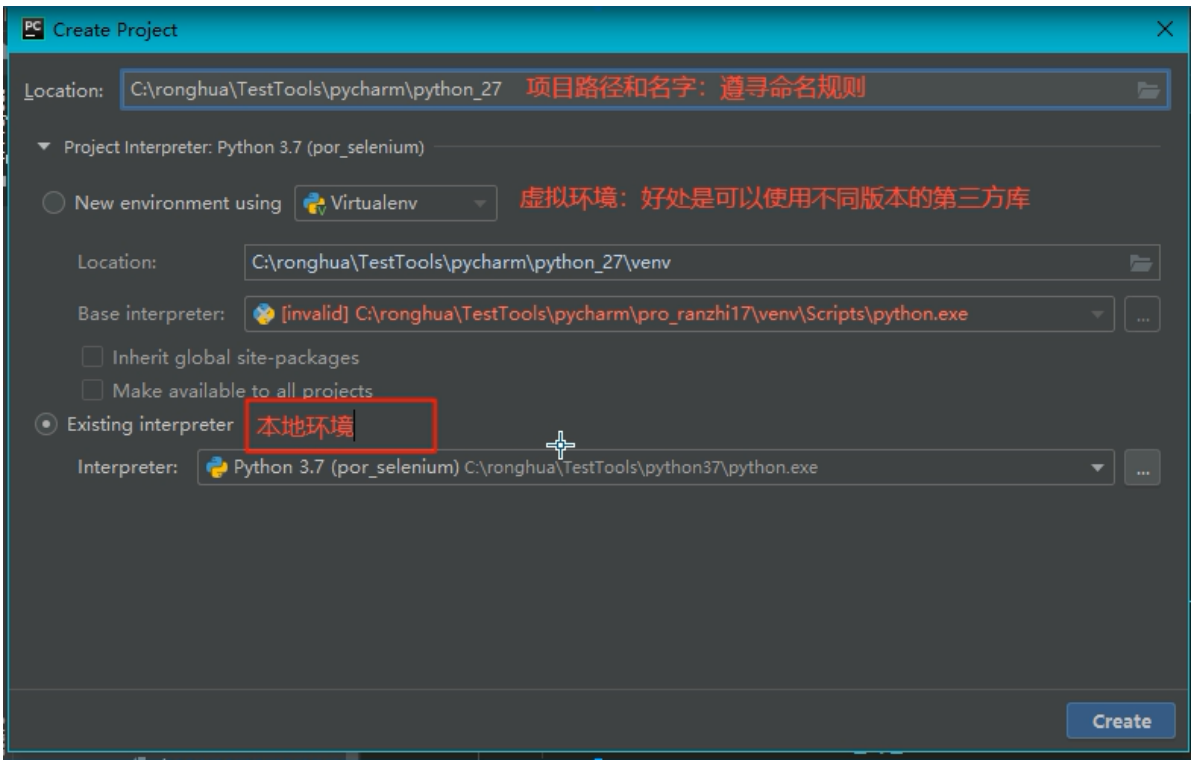
- 入口1：


Welcome to PyCharm
PyCharm
Version 2019.1.3
Create New Project
Open
Check out from Version Control
Events
Configure
Get Help

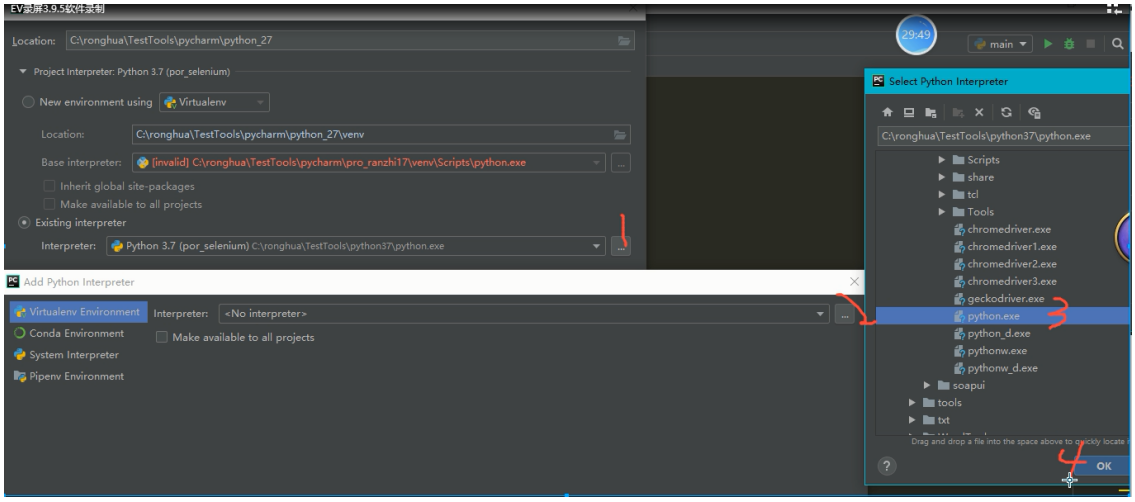
- 入口2：


File Edit View Navigate Code Refactor Run Tools VCS Window Help
pythonProject - day5.py - Administrator
New Project...
New... Alt+Insert
New Scratch File Ctrl+Alt+Shift+Insert
Open...
Save As...
Open Recent
Close Project
Rename Project...
Settings... Ctrl+Alt+S
File Properties
Local History
Save All Ctrl+S
Reload All from Disk Ctrl+Alt+Y
Invalidate Caches...
Manage IDE Settings
New Projects Settings
Export
Print...
FirstDay.py
test0001.py
Second.py
thirdday.py
day4.py
# FileName: day5.py
# Author: lijiayao
# Datetime: 2021/7/26 16:56
# Description:
import xlwt
from openpyxl import Workbook
# 实例化对象
jj = xlwt.Workbook() # 打开工作簿
# 选择默认当前页
ws = jj.add_sheet('sheet2')
# ws.append([1,2,3,4])
for a in range(1,10):
for b in range(1,a+1):
ws.write(a,b,"{} x {} = {}".format(b,a,a*b))

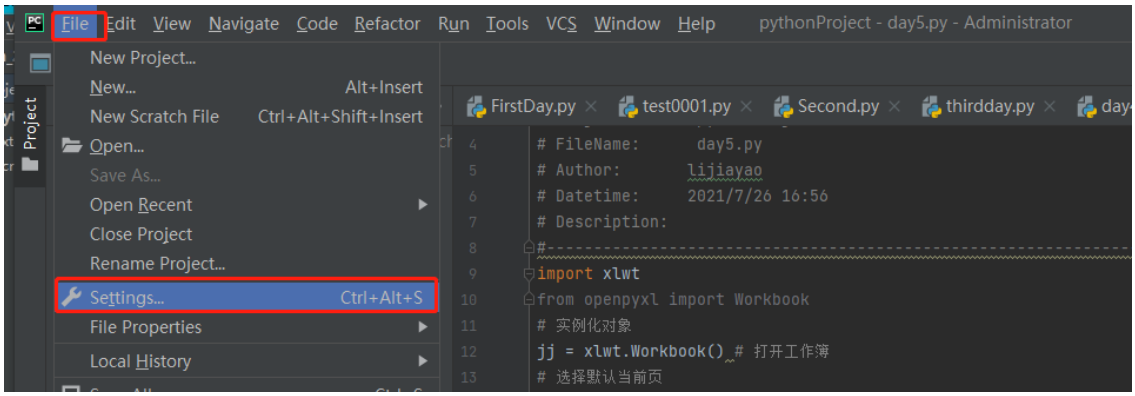
- 创建项目：



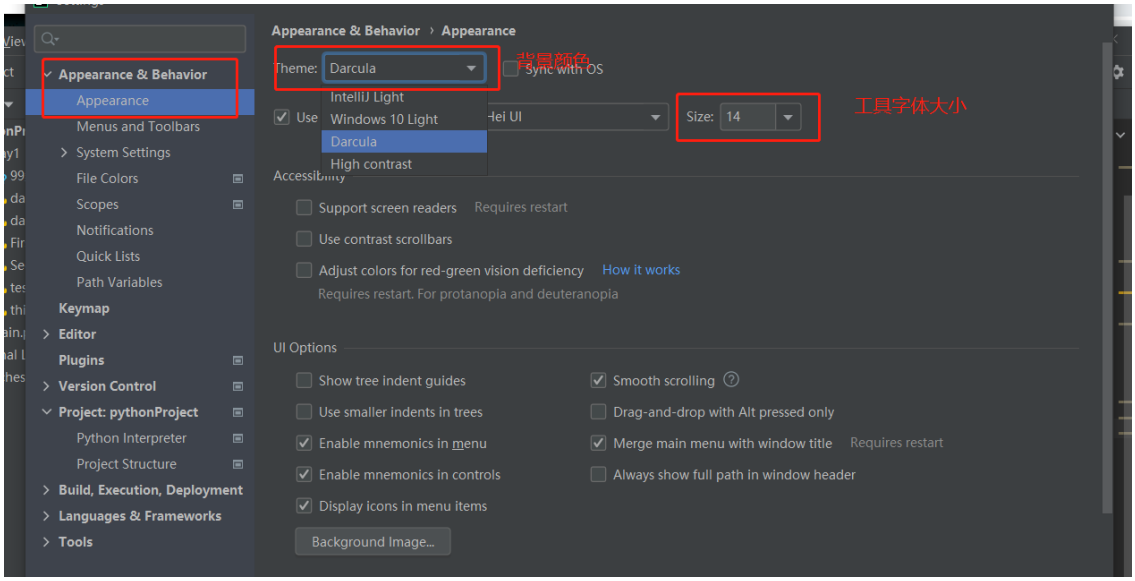
- 解释器选择



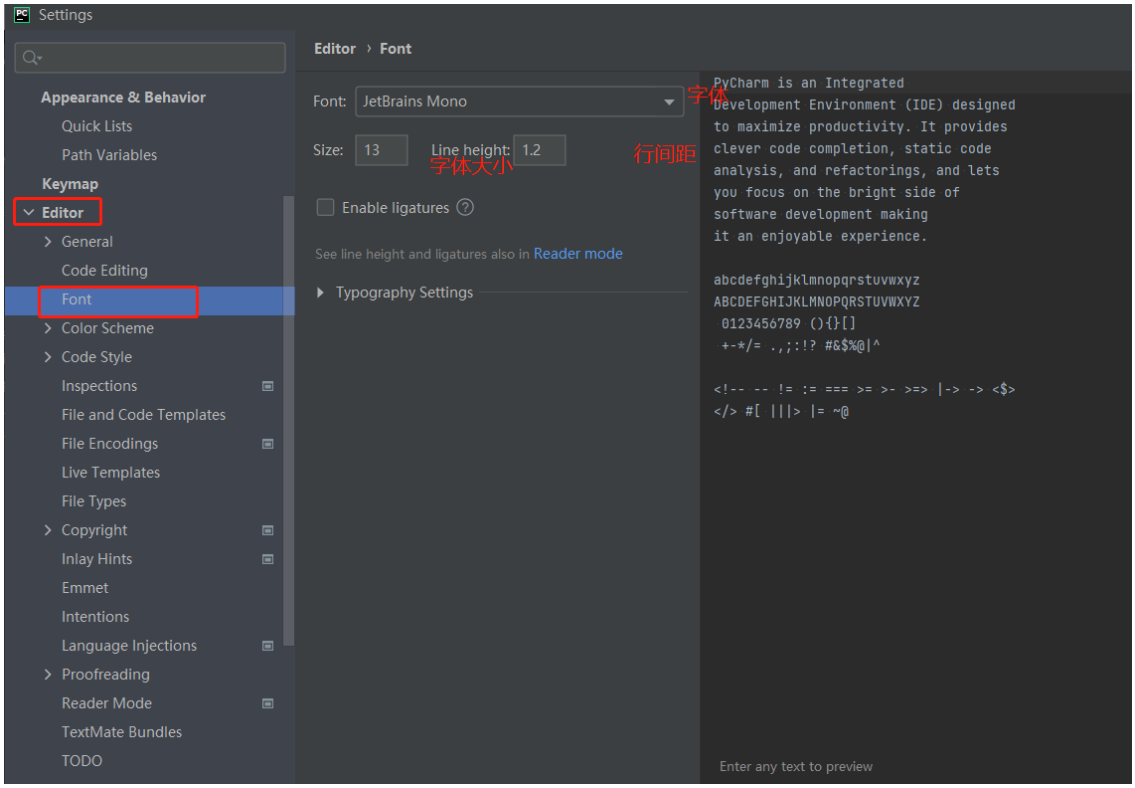
- 背景颜色选择



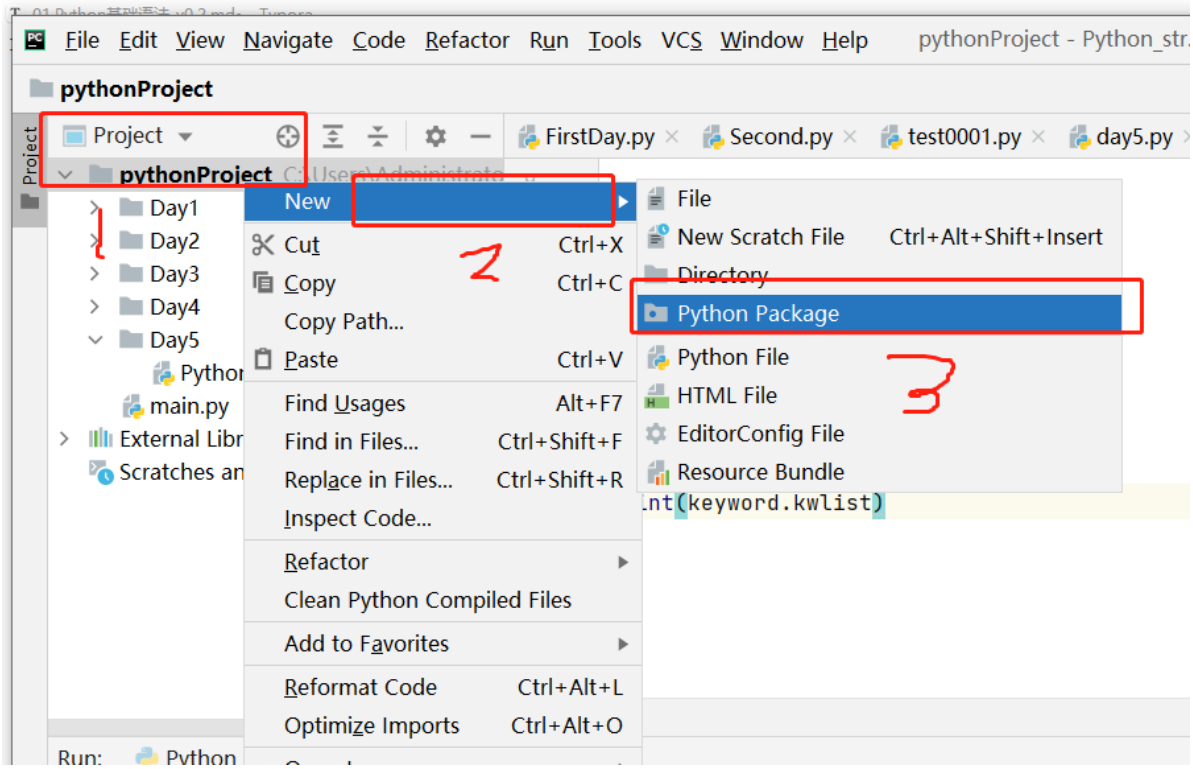
  - 背景颜色和工具的字体大小



- 代码内容的字体设置

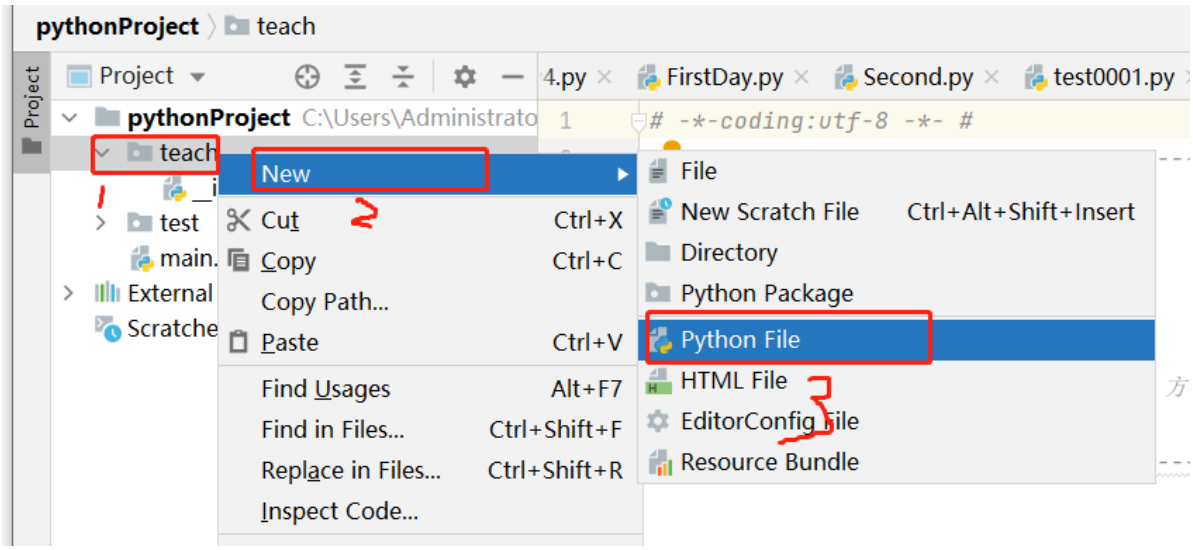


- 创建Python Package：

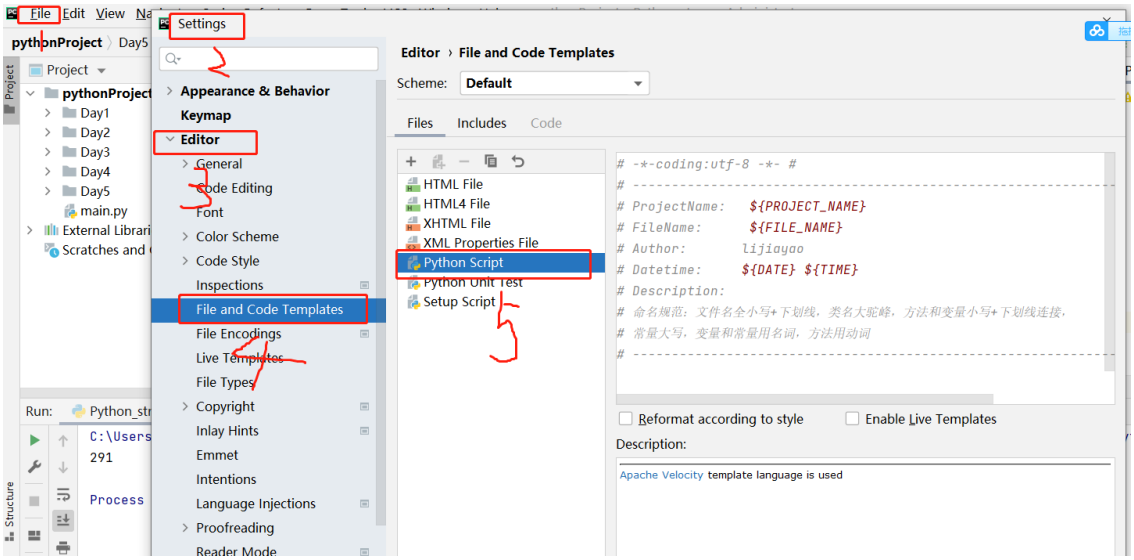


目录和Python包的区别？（标识文件、用途一个是放静态资源一个是放Python代码）

- 创建Python文件



- 文件里的抬头设置



```
# -*-coding:utf-8 -*- #
# -------------------------------------------------------------------------------
-
# ProjectName:   ${PROJECT_NAME}
# FileName:      ${FILE_NAME}
# Author:       xxxxxxx
# Datetime:     ${DATE} ${TIME}
# Description:
# 命名规范：文件名全小写+下划线，类名大驼峰，方法和变量小写+下划线连接，
# 常量大写，变量和常量用名词，方法用动词
# -------------------------------------------------------------------------------
-
```

- Pycharm中，代码提示的含义

2.8 代码入门

2.8.1 第一行代码

```
#输出hello world
print("hello world")
```

2.8.2 输出函数print

作用：实现输出一个结果到控制台。

举例：

```
#输出数字
print(12312)
#输出字符串
print("hello world")
#输出汉字
print("中国")
```

2.9 代码的注释

- 单行注释：用#实现，选中多行按Ctrl+/可以快速添加单行注释；
- 多行注释：用三单引号（'''.....''' ）或三双引号（"""....."""）实现

2.10 常用快捷键

- 快速注释：Ctrl+/
- 快速换行：Shift+Enter：不管光标在哪个位置，都能够在当前行的下面新起一行
- 快速复制：Ctrl+d
- 快速回退：Ctrl+z

三、数据类型

3.1 Python的六个标准的数据类型

- Number(数字)
- String(字符串)
- List(列表)
- Tuple(元组)
- Set(集合)
- Dictionary(字典)

3.2 Python变量的声明与使用（id()内置函数，它可以返回指定变量在内存中存放的地址）

- 变量：作用是存储程序运行中的值，变量的值在程序运行中是可以变化的，变量必须先声明再使用。
- Python变量的声明：变量名=值，比如a=1
- 常量：

  注意：

  - 在python中变量是没有数据类型的，变量的数据类型取决于赋值的类型，这与其他语言不一样。
  - 比如在Java中声明变量的时候需要指定变量的数据类型，赋的值的类型必须跟变量的类型保持一致，否则会报错，比如声明一个变量a赋值为1要写成int a=1，意思是先声明一个int类型的变量a，再对a赋值为1，变量的类型必须与赋值的类型保持一致，在Java中int a=3.5的写法是错误的。
  - Java是先定义类型再根据类型赋值，而python是根据值来确定类型
- 举例：

```
#声明一个变量num1，赋值为整数8
num1 = 8
#查看num1的类型
print(type(num1))
#使用变量num1
print(num1)

#声明一个变量str1，赋值为字符串hello world
str1 = "hello world"
#查看str1的类型
print(type(str1))
#使用变量str1
print(str1)
```

3.3 数值型

Python中的数值型包括：int（整型）、float（浮点数）、complex（复数），在Python2中还有long（长整型）

3.3.1 int

int（整型）：表示一个整数，包括正整数、负整数

```
a=9
print(type(a))
```

3.3.2 float

float（浮点数）：表示一个小数，类似于数学中的小数，浮点数必须包含一个小数点，否则会被当做int类型处理——（每个浮点对象都精确地表示一个数字。浮点对象不代表近似值。考虑浮点数的正确方法是——浮点数是精确的数字，但浮点运算（二进制对很多浮点数无法准确表示只能用一个近似值代替，而当使用这些以近似值代替的浮点数进行运算时本质上是这些进似值参与了运算，出来的结果也就是进似值运算后的结果）后的结果大多数都是近似值）

```
a=3.11
b=1.5
print(a-b)  #输出1.6099999999999999而并不是1.61
```

如果要表示精确值，可以使用Decimal对象来实现

```
#通过实例化Decimal对象来表示精确小数
from decimal import Decimal
print(Decimal("3.11")-Decimal("1.5"))
```

3.3.3 其它进制的表示与进制直接的相互转换（计算机基础课已讲）

常用的进制：二进制、八进制、十进制、十六进制等

- 二进制（Binary）
  - 由0, 1组成，逢二进一
  - 二进制的声明：在数字前加0b表示二进制

```
#声明一个变量a，赋值为二进制1001
a=0b1001
#输出a的值，输出的是十进制9
print(a)
```

- 八进制（Octal）
  - 由0, 1, ... ,7组成，逢八进一
  - 八进制的声明：在数字前加0o表示八进制

```
#声明一个变量a，赋值为八进制0011
a=0o0011
#输出a的值，输出的是十进制9
print(a)
```

- 十进制（Decimal）
  - 由0, 1, ... ,9组成，逢十进一
  - 十进制的声明：Python中默认的数值就是十进制的，因为人习惯的是十进制
- 十六进制（Hexadecimal）
  - 由0, 1, ... ,9, A, B, C, D, E, F，逢十六进一
  - 十六进制的声明：在数字前加0x表示十六进制

```
#声明一个变量a，赋值为十六进制000F
a=0X000F
#输出a的值，输出的是十进制15
print(a)
```

进制的转换（不需要）

- 二进制与十进制之间的相互转换

```
#转换为十进制：用从左往右的第一个数乘以2的第一个数所在位置的次方加上第二个数乘以2的第二个数所在位置的次方
例：
0b100101
  543210(二进制数的位置编号，从右往左从0开始编号)

十进制数等于：1x2^5+0x2^4+0x2^3+1x2^2+0x2^1+1x2^0 = 37

#十进制转换二进制
十进制如何转二进制：将该数字不断除以2直到商为零，然后将余数由下至上依次写出，即可得到该数字的二进制表示，
如：
11 除以 2 商5 余1
5  除以 2 商2 余1
2  除以 2 商1 余0
1  除以 2 商0 余1
# 11转换为二进制为：1011
```

- 八进制与十进制之间的相互转换

```
#八进制转换为十进制
方法：八进制数从低位到高位（即从右往左）计算，第0位的权值是8的0次方，第1位的权值是8的1次方，第2位的权值是8的2次方，依次递增下去，把最后的结果相加的值就是十进制的值了。
如：
八进制的45
转换为10进制：5x8^0+4x8^1 = 37
#十进制转换为八进制和二进制差不多
方法：采用除8取余法（和二进制方法类似）
```

- 十六进制与十进制之间的相互转换

```
#十六进制转换为10进制，和二进制，八进制方法相同
#十六进制的数：0x123
#转换为十进制：1x16^2+2x16^1+3x16^0 = 291
#十进制转换为十六进制
方法：采用除16取余法（和二进制类似）
```

进制转换函数

- bin()：把十进制转成二进制
- oct()：把十进制转成八进制
- hex()：把十进制转成十六进制
- int(x,base=y)：表示把y进制的字符串x转换成十进制的整数
- 举例：

```
print(bin(10)) # 十进制转换为二进制
print(oct(10)) # 十进制转换为八进制
print(hex(17)) # 十进制转换为十六进制
print(int("1000",base=2))#表示把2进制的字符串“1000”转换成十进制的整数
print(int("1002",base=2))#思考：输出是什么？
```

3.4 字符串

3.4.1 Python中的引号

python中的引号：单引号''、双引号""、三引号''' '''或者三双引号""" """，一般情况下三种引号可以通用，但必须成对出现；

三引号还可以用来做多行的注释。

3.4.2 字符串的声明

字符串用引号来声明（单引双引三引都可以），字符串是有序的一串字符。

```
#声明一个空字符串
str0=""
print(type(str0),len(str0))

#声明一个非空字符串
str1='hello world'
```

```
print(type(str1))
print(str1)

#声明一个非空字符串
str1="hello world"
print(type(str1))
print(str1)

#声明一个非空字符串
str1="""hello world"""
print(type(str1))
print(str1)

#字符串的加法和乘法运算
str1="hello"*3
str2 = "hello"+"world"
```

3.4.3 字符串的切片

字符串是有序的，字符串中的每个字符是有索引下标的，字符串的索引有两种：

- 正序索引：从左往右开始下标依次为0,1,2...
- 倒序索引：从右往左开始下标依次为-1,-2,...

字符串切片语法：str[start : end : step]，注意它的规则是左闭右开规则，省略start表示从开头取，省略end表示取到末尾，步长默认为1，字符串切片得到的是一个字符串。

```
# 切片 str[起始值：终止值：步长] 起始值从0开始
str3 = "123456789"
print(str3[4])    # 取当前位置的值
print(str3[1:4])  # 步长不输默认为1
print(str3[:4])
print(str3[4:])   # 终止值不输默认取到最后
print(str3[:])
print(str3[4:7:1]) # 区间左闭右开
# 1.取13579
# 2.取2468
# 3.取987654321   print(str3[::-1])
# 4.取97531
# 5.取753
# 步长为正：从左往右取；步长为负：从右往左取
print(str3[-1:-5])
print(str3[-1:-5:-1])
print(str3[-5:-1])
print(str3[-5:-1:-1])

# 取区间不在范围内的不会报错，返回空字符串，
#去找这个范围内的值是否存在。如果是单个值不在范围内，则会报错。
print(str3[20])
print(str3[20:30])
```

3.4.4 字符串的常用函数和方法

- **len()函数**

  作用：计算字符串的长度

  举例：

```
str1="hello world"
print(len(str1))
```

- **find()方法**：(rfind找最大索引值)

  作用：在字符串中找第一个出现的子串的下标，如果找不到返回-1

  举例：

```
str2 = 'nihao,feichanghao'
# 查询第一个a元素的下标索引
print(str2.find('a'))
# 查询下标索引位置从10到17的区间中的第一个a的下标索引
print(str2.find('a',10,17))
```

- **index()方法：**

  作用：在字符串中找第一个出现的子串的下标，如果找不到抛ValueError错误

  举例：

```
print(str2.index('e'))

print(str2.index('4'))
```

  find()和index()方法的区别：find()如果在指定字符中没有找到则返回-1；而index则会抛出ValueError异常

- **format()方法**

  作用：实现字符串的格式化输出

  举例：

```
#场景一：format方法中的值按位置替换字符串中的{}
print("我叫{}，我来自{}，今年{}岁了。".format('jack','成都','10'))

#场景二：以索引的方式来表示替换参数，{0}位置用传入的第一个变量替换，{1}位置用传入的第二个变量替换，{2}位置用传入的第三发变量替换。可以允许传入的参数比需要的多，但不能比需要的少
print("我叫{0}，我来自{1}，今年{2}岁了。".format('jack','成都','10'))
print("我叫{1}，我来自{1}，今年{2}岁了。".format('jack','成都','10'))
print("我叫{0}，我来自{1}，今年{2}岁了。".format('jack','成都','10','北京'))
print("100+200={0}".format(100+200))

#场景三：在大括号中除了用下标之外，还可以用变量名，这种情况下传入的参数顺序就没有关系了。
print("我叫{name}，我来自{city}，今年{age}岁了。".format(name='jack',city='成都',age='10'))
print("我叫{name}，我来自{city}，今年{age}岁了。".format(city='成都',age='10',name='jack'))
print(str2)
```

在Python中除了用format方法实现格式化输出之外，还可以使用格式符来实现格式化输出，常用的格式符有如下：

%s 字符串

%d 十进制整数

%f 浮点数

参数必须位置、数量一一对应，格式化输出举例：

```
name = "小王"
age = 9
#如果不适用格式化输出，会多出一个空格
print("老王的儿子叫",name,"今年",age,"岁了")

#使用格式化输出
print("老王的儿子叫%s今年%d岁了"%(name,age))

# %.2f表示四舍五入保留2位小数输出
print("%.2f"%1.33333)

# 也可以使用round函数实现四色五入，保留n位小数
print(round(1.34356789,4))
```

- **count()方法**

  作用：统计子串出现的次数

  举例：

```
str2 = "elgjlajlnvayzdfad"
print(str2.count('e'))

print(str2.count('ay'))

print(str2.count('z'))
```

- **join()方法：**

  作用：传入一个可迭代对象，把可迭代对象中的每个元素通过调用它的字符串来拼接，返回拼接后的字符串。

  举例：

```
# 在字符串中拼接“_”,把每个字符分开
str2="todayissundaya"
print("_".join(str2))

#错误的，join()方法传入的参数必须是迭代对象（包括字符串、列表、元组等等）
print("_".join(100))

#join方法中传入列表，但列表的元素必须是字符串
print("".join(["1","3","5"]))

#join方法中传入元组，但元组的元素必须是字符串
print("".join(("1","3","5")))
```

- **replace(old,new[,count])方法**：

作用：替换旧的字符串成新字符串，count参数可选

举例：

```
print(str2.replace('a','A'))  #把str2中的a替换成A

print(str2.replace('a','A',1)) #只替换一次，把a替换成A

print(str2.replace('a','A',2).replace('A','a',1)) #把str2中的第2个a替换成A
```

- **split(sep)方法：**

  是将一个字符串分裂成多个字符串返回列表的方法，详细即通过指定分隔符对字符串进行切片，如果参数num 有指定值，则仅分隔 num 个子字符串

  作用：在字符串中遇到sep就分割，返回一个列表路径拼接的时候会用到，还有定位方式封装的时候也会用到分割

  举例：

```
str2 = "你好_xiao_再来"
print(str2.split('_')) #得到['你好','xiao','再来']
```

- **rstrip()/lstrip()/strip() 方法**

  作用：

  rstrip()方法是去除字符串右边的空格

  lstrip() 方法是去除字符串左边的空格

  strip() 方法,去除字符串两边的空格

  举例：

```
str1 = " jlkj agjl "
# 去除开头的空格
print(str1.lstrip())
#去除结尾的空格
print(str1.rstrip())
#去除两边的空格
print(str1.strip())
```

- capitalize()方法

  作用：将字符串的首字母大写，其余字母全部小写

  举例：

```
print(str1.capitalize())

print("today is wednesday".capitalize())
```

- upper()

  作用：把字母全部转换为大写

  举例：

```
print(str1.upper())
```

- lower()

  作用：把字母全部转换为小写

  举例：

  ```
  print(str1.lower())
  ```

- title()

  作用：字母的首字母都大写，标题化

  举例：

  ```
  str3 = "hello new world"
  print(str3.title())
  ```

- endswith()方法：

  作用：判断字符串以xx结尾

  举例：

  ```
  print(str2.endswith("x"))
  ```

- startswith()方法：

  作用：判断字符串以xx开头

  举例：

  ```
  print(str2.startswith("t"))
  ```

所有is开头的方法返回结果都是布尔值（True、False）

- isalnum()：字符串中所有字符都是字母或数字且非空则返回True
- isalpha()：字符串中所有字符都是字母且非空则返回True
- isdigit()：字符串中所有字符都是数字且非空则返回True
- isupper():字母都为大写且非空返回True
- islower():字母都为小写且非空返回True
- istitle():单词的首字母都为大写且非空返回True

```
print("we123".isalnum())

print("".isalnum())

print("%^*".isalnum())

print("we123".isalpha())
```

举例：从键盘输入密码，判断密码是否是由数字或字母组成，如果是返回输入正确，否则返回输入错误

```
str1 = input("请输入密码：")
if str1.isalnum():
   print("输入正确")
else:
   print("输入错误")
```

- ord()函数

  作用：是返回字符的ascii码

  举例：

```
print(ord('a'))
print(chr(97))

print(ord('A'))
```

- chr()函数

  作用：跟ord()相反，它根据ascii码返回对应的字符

  举例：

```
print(chr(99))
```

3.4.5 字符串的驻留机制

在编程语言中，=表示赋值，==表示等于，is表示判断两个对象是否是同一个对象(内存地址)，==与is的区别是：==比较的是值，is比较的是内存地址。

字符串的驻留机制：在存放两个相同的字符串时，是否申请新的内存空间来分别存放取决于字符串的复杂程度（有多种类型的字符，如果只有一种即便很长那也不算复杂）

```
str1="helloworldnihao"
str2="helloworldnihao"
str3="monday%^ sunday"
str4="monday%^ sunday"

print(str1==str2)
print(str1 is str2)
print(id(str1))
print(id(str2))
print(str3==str4)
print(str3 is str4)
print(id(str3))
print(id(str4))
```

在PyCharm上运行和在dos下运行是有区别的

3.4.6 转义字符

在编程语言中，定义了一些特殊字符来表示特定的含义，这被称为转义字符，之所以叫转义是因为字符原本的含义发生了改变。比如：（转义符是“\”）

\'　表示单引号，在单引中输出单引时需要用这个转义符

\"　　表示双引号，在双引中输出双引时需要用这个转义符

\""" 表示三引号，在三引中输出三引时需要用这个转义符

\\　　表示反斜线，在输出反斜线的时候需要用到

\t　　表示水平制表符，作用类似于tab健，在输出需要有间隔时使用

\n 换行符，在输出时，需要换行时使用

' 单引号:双引号里面只能用单引号

```
print("i'm a boy")
```

" 双引号

\ 反斜杠\:如果刚好遇到"\test"这种，系统会把\t识别成转义字符，这时要在前面再加一个\，如果路径很长，可以在最前面加个r

```
print("\\test")
print(r"c:\test\rh.txt")
```

\x 表示后面的字符是十六进制数

3.4.7 课后作业

1. 声明一个字符串“hello world,w1234”，从第2个w开始切片直到最后；
2. 声明一个字符串“hello world,1234”，将字符串的最后4个字符反转输出
3. 打印“那车水马龙的人世间,那样地来 那样地去,太匆忙"中最后一次出现"那"的位置
4. 判断一个的字符串是否是 .py 结束
5. 去掉字符串“ hello world ”两边的空格
6. 有如下格式的字符串“name-age-sex-address”,去掉中间的“-”。
7. 统计字符串”hello world“中”l“的个数

3.5 列表list

3.5.1 列表的声明

- 列表是由一系列有序的元素组成，所有的元素被都在中括号中，它里面的元素可以是数字、字符串、甚至可以是列表，元素跟元素之间用逗号分隔。
- 列表的声明：通过[]或list()或列表推导式来声明（一维列表、多维列表）
- 声明一个列表：

```
#声明一个空列表
list1=[]
print(type(list1),len(list1))
#声明一个非空列表
list2=['a',1,5,'hello',[2,4,6,'world']]
print(type(list2),len(list2))
#通过创建list对象来声明一个列表
list1 = list("hello")
print(list1)
num = 231314
print(list(num)) # 错误的，不能转换
#通过列表推导式来声明一个列表
print([i for i in range(1,101) if i%2==0])

#二维列表,只能是如下的列表(最常见的是二维列表和三维列表)
```

```
list1 = [[1,2,a],[],[],[]]
```

- 修改列表中的元素

```
list2=['a',1,5,'hello',[2,4,6,'world']]
print(list2) # 修改前
# 修改列表中的5为99
list2[2] = 99
print(list2) # 修改后
```

3.5.2 列表的切片

- 列表的切片：规则与字符串的切片类似，但它返回的是一个列表。
- 举例：

```
list1 = [1,3,4,5,6,7,8,9,0,123,4566,888,[345,567,999],"hello,中国"]
print(list1[1]) # 取下标索引为1的元素
print(list1[:6]) # 取下标索引默认从0开始到6结束步长默认为1的元素
print(list1[::]) # 全部默认，取整个列表
print(list1[::-1]) # 全部默认，取整个列表的倒序
print(list1[12][1]) # 取列表中的列表里的元素
print(list1[13][1]) # 取列表中的字符串里的元素
```

3.5.3 列表的常用函数

- len(list)
  - 作用：计算列表的长度，即计算列表中的元素个数
  - 举例：

```
list1 = [1,2,3,4,"niaoogho","你好", "a","z","A"]
print(len(list1))
```

- max(list)
  - 作用：取列表中的最大值
  - 举例：

```
# 字母排序是根据ASCII码排序
正向例子：
list2 = ["a","z","A"]
print("最大值是：",max(list1))
#反向例子
list1 = [1,2,3,4,"niaoogho","你好", "a","z","A"]
print("最大值是：",max(list1))
```

- min(list)
  - 作用：取列表中的最小值
  - 举例：

```
# 字母排序是根据ASCII码排序
list1 = [1,2,3,4,"niaoogho","你好", "a","z","A"]
print("最小值是：",min(list1))
```

3.5.4 列表的常用方法

- append()
  - 作用：在列表的末尾增加一个元素
  - 举例：

```
list1 = [1,2,4,5,65,"真的"]
print("添加前：",list1)
list1.append(300)
print(list1)
list1.append([400,500]) # 添加一个列表，作为一个元素添加在末尾
print(list1)
```

- insert()
  - 作用：在索引的前面添加单个元素
  - 举例：

```
list1 = [1,2,4,5,65,"真的"]
# 在元素下标为3的位置插入999
print("添加前：",list1)
list1.insert(3,999)
print("添加后：",list1)
# 在超出索引位置添加？-会添加在列表的最后
list1.insert(8,88)
print("超出索引添加：",list1)
```

- extend()
  - 作用：列表的拼接,添加多个元素
  - 举例：

```
list1 = ["你好"]
list2=[111,222,333]
# list1.extend([10001,10002,10003])
list1.extend(list2)
print(list1)
# +号拼接，结果和extend一样
print("两个列表相加：",list1+list2)
# ，号拼接只是把两个放一起，没有变成一个
print("两个列表逗号拼接：",list1,list2)
```

- index(x):
  - 作用：从列表中找出某个值，第一个匹配项的索引位置
  - 举例：

```
list1 = [1,2,3,5,7,"zheng","很好"]
print(list1.index(2)) # 返回2的下标索引位置
# 如果查询的不在列表内？-报错：ValueError: 8 is not in list
print(list1.index(8))
```

- count():
  - 作用：计算元素出现的次数
  - 举例：

```
list1 = [1,2,3,1,7,"zheng","很好"]
print(list1.count(1))
```

- remove()方法
  - 作用：删除列表中的某个元素：括号中传入要删除的元素值，不返回删除元素，返回None
  - 举例：

```
list1 =["a","b","d","c"]
print("删除前",list1)
print(list1.remove('a')) # 打印删除时是否返回
print("删除后",list1)
```

- pop()方法
  - 作用：传入要删除元素的下标，pop方法会返回删除的元素,如果不传默认删除最后一个
  - 举例：

```
list1 =["a","b","d","c"]
print("删除的元素是：",list1.pop(3))
print(list1)
```

- del list1[index]
  - 作用：删除列表或列表中的数据
  - 举例：

```
list1=[[2, 4, 6, 'world'], 'hello', 5, 1]
# 根据下标索引删除，删单个元素
del list1[1]
print(list1)
# 根据下标索引删除，删区间
del list1[1:3]
print(list1)
# 删除整个列表
del list1
print(list1)
```

- clear()方法
  - 作用：清空列表，得到一个空列表
  - 举例：

```
list1.clear()
print(list1) # 得到一个空列表[]
```

- copy()
  - 作用：复制列表
  - 举例：

```
list1 =[1,2,3,4]
print("复制前：", list1)
b = list1.copy()
print("复制后：",b)
```

- sort()方法
  - 作用：对列表的元素排序。
  - 通过reverse参数来控制排序是升序还是降序，reverse只能传布尔值True或者False，默认是False（升序），所以如果不传入任何参数，表示默认升序排列，数字从小到大，字符根据ASCII码来排；
  - 数值跟列表以及字符串之间不能排序；
  - 通过key参数来控制排序的权重，即根据第几个字符来排序，如果有元素排序的条件相同，则最终排序结果跟存储的顺序保持一致。

    举例：

```
list1 = [1,2,3,4,5]
list2 = ["a","b","A","B","a"]
list3 = ["中国","你好","加油"]
# 默认升序排列
list1.sort()
print(list1)
list2.sort()
print(list2)
list3.sort()
print(list3)
# 降序排列
list1.sort(reverse=True)
print(list1)
```

- sorted()函数
  - 作用：sorted 是内置函数，可对所有可迭代的对象进行排序操作，返回新的list。sorted函数与sort方法的区别是sorted函数会返回排序后的新列表，而sort方法不会返回。
  - 举例：

```
list1 = [7,8,4,6,33,1,3]
list2 = sorted(list1)
print(list2)
```

- reverse()方法

* 作用：反向列表中的元素，reverse()方法和列表[::-1]的区别是前者会改变列表中的元素顺序，而后者不会改变顺序，只会把列表的元素反着取出来。

* 举例：

```
list1 = ["1","a","9","你好","heog"]
list1.reverse()
print(list1)
# 切片方式反向
print(list1[::-1])
print(list1)
```

- 列表与元素之间的转换
  - 通过字符串的join方法可以实现列表中的元素转字符串，通过创建list类的对象可以实现字符串的元素转列表
  - 举例：

```
list1=['w', 'e', 'a' ,'r', 'e', 'g', 'o', 'o', 'd', 's', 't', 'u', 'd',
'e', 'n', 't', 's']
print(type("".join(list1))) #列表转字符串，打印类型
print("".join(list1))

print(list("135"))
```

3.5.5 课后作业

一. 一个列表list1=[1,1,1,1,1,1,10,11];

（a）统计该列表中出现1元素几次；

（b）然后要求反转不改变原列表内存存储?

（c）要求在倒数第二个1前面添加一个元素列表list2=[2,3,4,2,1]，要求返回的是一个一维列表

二.已知字符串"ronghuanettest@vip.com"，要求对字符串进行切片操作，切片得到的每个字符串以列表返回，最终结果是：["rong","hua","net","test","vip","com"]

三.已知一个列表
lst = [1,2,3,4,5]

1. 求列表的长度
2. 判断6 是否在列表中
3. lst + [6, 7, 8] 的结果是什么?
4. lst*2 的结果是什么
5. 列表里元素的最大值是多少
6. 列表里元素的最小值是多少
7. 列表里所有元素的和是多少
8. 在索引1的位置新增一个的元素10
9. 在列表的末尾新增一个元素20

3.6 元组tuple

3.6.1 元组的声明

- 元组通过()或tuple()来声明。
- 元组是一组有序的数，用小括号()来表示，元组的元素跟元素之间用英文逗号隔开，元组跟列表的最大区别是：元组中的单个元素不能增删改。元组通常用于保护数据而存在
- 声明元组

```
#空元组
tupl0=()
print(type(tupl0),len(tupl0))
#非空元组
tup1=(1,2,5)
print(type(tup1),len(tupl1))
#用tuple实例化对象来声明元组
num=123
str1="你好吗"
list1=[1,2,4,5]
print(tuple(num)) # 报错
print(tuple(str1))
print(tuple(list1))
```

- 声明一个只有一个元素的元组：需要加一个逗号来表明这是一个单元素的元组，否则解释器会理解成在外面加了一个普通的括号，打印类型为单个元素的类型。

```
#一个元素的元组
tupl2 =(2,)
tupl3 =("nihao",)
```

3.6.2 元组的作用

1. 可以实现对数据进行保护；
2. 在操作不定长参数函数时，其参数的类型就是一个元组，所以可以直接将元组进行传入；
3. 在函数或者方法中返回值可以返回多个值，默认是把多个值以元组的形式返回。

3.6.3 元组和列表的区别

列表中的元素是可变的，可以对元素进行增删改操作；元组中的元素是不可变的，不能对元素进行增删改操作。

3.6.4 元组的切片

- 元组的切片（有序的，可以使用下标索引）,切片出来的数据还是保存在元组里的规则跟字符串、列表一致

```
tup1=(1,2,(1,2,3),5,999,'fgjgjfs',[1,2,3])
# 不在范围内
print(tup1[1111:])
# 取下标为2的元素
print(tup1[2])
# 取下标为1开始，终止值默认到最后，步长默认为1
print(tup1[1:])
# 倒序？
# 取元组中的元组里的数据
```

3.6.5 元组的常用操作

- 元组的修改（不是修改，实际是形成了一个新的元组）

  a. 转成列表间接使用列表的方法来操作，最后再转成元组

```
print(tuple([1,2,3]))
print(tuple("youseeyou"))
list1=list(tup1)
list1.insert(list1.index(5)+1,100)
tup1=tuple(list1)
print(tup1)
```

  b. 也可以通过切片再拼接的方式实现。注意对字符串、列表、元组都可以用+来实现拼接操作。

```
print(tup1[:4]+(100,)+tup1[4:])
```

- 元组的删除

  元组只能用del 来删除整个元组

```
tup1=(1,2,(1,2,3),5,999,'fgjgjfs',[1,2,3])
del tup1
print(tup1)
```

3.6.6 元组的常用函数

- len()
- max()
- min()

其中max()和min()两个函数要保证比较的元素类型一致

3.7 range对象序列

- range是不可变的序列，元素为int类型，通常和for循环结合起来使用
- range(起始值=0，终止值，步长=1)：表示以指定的步长从起始值取到终止值-1，取值为左闭右开
  - 如果只传一个值，则表示终止值：range(3)
  - 如果传两个值，则表示，起始和终止值：range(1，3)
  - 如果传三个值，则表示起始，终止值和步长：range(1,10,2)

```
for i in range(5):
    print(i)
for i in range(1,5):
    print(i)
for i in range(1,5,2):
    print(i)
```

3.8 字典dict（映射类型，无序）

3.8.1 字典的声明

- 声明方式：通过{}或dict()或字典推导式来声明
- 字典的特点：
    - 字典是无序的，由键key和对应的值value组成，比如{key:value}；
    - 字典的键必须是唯一的，值可以不唯一；（键重复）
    - 每个键与值之间用英文的冒号:隔开,两个键值对之间用英文的逗号,隔开。

```
#创建空字典
dict1 ={}
print(type(dict1),len(dict1))
#创建非空字典
dict2={"名称":"乔峰","年龄":18}
#键重复的例子
dict2 = {"名称":"乔峰","年龄":18,"年龄": 20}
#通过dict来实例化对象（转字典）
dict1 = dict(name="xiaoming",age=8,like1="篮球")
print(dict1)
#通过字典推导式声明{结果 for 变量 in 迭代对象} 或者 {结果 for 变量 in 迭代对象 if 布尔表达式};
people = [('小红', 18), ('小明', 45), ('小王', 22)]
dict3 = {k:v for k,v in people}
print(dict3)
```

3.8.2 根据key取value

方法1：需要根据键名去取，格式：字典名[键名]，注意这里一定是传入键名根据键名去取键值。

方法2：通过get方法去取键值

举例：

```
#通过key值取
dict2={"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]}
print(dict2["名称"])
print(dict2["喜欢"][0])
#通过get方法去取键值
print(dict2.get("年龄"))
```

两种方法区别：方法1，如果key不存在，报-KeyError；方法2则返回None

3.8.3 修改字典

修改已存在的值

```
dict2={"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]}
#找到已在的key，然后重新赋值，修改的是value
dict2["名称"] = "段誉"
print(dict2)
```

新增键值对-单个

在字典中增加一个元素（键值对）,格式：字典名[键名]=键值，如果键名在字典中不存在就会增加，如果已存在就会修改它的键值

```
dict2={"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]}
dict2["虚竹"] = "憨子"
print(dict2)
```

新增多个键值对

- update()方法

  作用：在字典中增加多个键值对，相当于字典的拼接，如果键名重复就覆盖原来的键值

  举例：

  ```
  dict1={"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]}
  dict2={"学历":"高中","毕业时间":2010,"city":"绵阳"}
  dict1.update(dict2)
  print(dict1)
  dict1.update({"性别":"Female","公司":"华为"})
  print(dict1)
  # 字典相加不能用+
  print(dict1+dict2)#错误的
  ```

3.8.4 字典的删除

- del

  作用：删除整个字典或者根据健删除指定的键值对

  举例：

  ```
  dict1={"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]}
  #删除某个键值对-用del 字典名[键名]的方式来删除指定的键值对
  del dict1["名称"]
  print(dict1)
  #删除整个字典
  del dict1
  print(dict1)
  ```

- pop()方法

  作用：pop方法的作用是根据键名删除键值对，并返回键值，如果找不到键名就返回传入的default参数或者KeyError

  举例：

```
dict1={"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]}
print(dict1.pop("年龄"))
print("删除后：",dict1)
```

- popitem()方法

  作用：popitem()方法的作用是删除并返回字典的最后一个键值对,以元组返回。

  举例：

```
dict1={"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]}
print(dict1.popitem())
print("删除后：",dict1)
```

- clear()
  - 作用：清空字典
  - 举例：

```
dict1={"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]}
dict1.clear()
print(dict1)
```

- copy()
  - 作用：复制字典
  - 举例：

```
dict1={"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]}
dict2=dict1.copy()
print(dict1)
print(dict2)
```

  获取k值和value值、获取keys、values

  for k,v in dict2.items()

  获取key值和value值后交换

3.8.5 字典的嵌套

字典中的value可以是元组、列表或者另一个字典（可以是任何数据类型），key可以是数值型，字符串，元组（不可变数据类型），不能是list和dict

```
dict1={"信息":{"名称":"乔峰","年龄":18,"喜欢":["阿朱","阿紫"]},"级别":[1,2,4],"排名":
(1,2,3)}
#1.求年龄
print(dict1["信息"]["年龄"])
#2.求"阿朱"
print(dict1["信息"]["喜欢"][0])
#3.修改年龄为20
dict1["信息"]["年龄"] =20
print(dict1)
```

3.8.6 在格式化输出中传入字典

```
#字符串
print("小明喜欢%s，今年%d岁了"%("小花",9))
#1.字典（类型要对应）
print("小明喜欢%(name)s，今年%(age)d岁了"%{"name":"小花","age":9})
##format -这里传参时，*表示是元组，**表示字典，这里传的是字典，是一个整体，需要用**来解包，把字典解包为单个键值对，才能取到
print("小明喜欢{name}，今年{age}岁了".format(**{"name":"小花","age":9}))
dict1={"name":"小花","age":9}
print("小明喜欢{name}，今年{age}岁了".format(**dict1))
```

3.8.7 课后作业

1、字典prices = { 'A':123, 'B':450.1, 'C':12, 'E':444}

a）分别获取字典中最大的键和值?

b）获取字典中最大的键及对应的值?

c）获取字典中最大的值及对应的键?

3.9 集合set（无序的）

3.9.1 集合的声明

- 集合的声明：集合通过{}或set()或集合推导式来声明。
- 集合的特点：集合是一个无序的不重复元素序列，集合中的元素可以是任意不可变的数据类型

```
#定义空集合
set1 = set()
set2 = {}会表示字典类型
print(type(set1),len(set1))
#定义非空集合
set1={9,1,5,6,5,1,2,2}
set2 = {"k","k","k","khjj"}
print(type(set1),len(set1))
```

- 利用集合中元素不可重复的特点对列表、元组、字符串去重

```
list1=[1,1,1,2,3,2]
print(type(set(list1)))      #得到的是一个集合
#列表转换成集合
print(set(list1))
#集合转换成列表
print(list(set(list1)))
tup1=(1,1,1,3,3,3)
#元组转换成集合
print(set(tup1))
#集合转换成元组
print(tuple(set(tup1)))
str1 = "aaaaffffcccc"
print("".join(set(str1)))
```

```
print((set(str1)))
```

3.9.2 集合的运算

集合的运算：交集、并集、差集、对称差集

- 交集：取两个集合的公共部分，也就是同时存在于两个集合中的元素

```
set1={9,1,5,6,5,1,2,2}
set2={1,6,2,20,50}
# 运算符
print(set1&set2)
# 方法
print(set1.intersection(set2))
```

- 并集：两个集合中的所有元素，重复的元素只算一次

```
#运算符
print(set1|set2)
#方法
print(set1.union(set2))
```

- 差集：set1-set2的结果存在于set1中但不存在于set2中的元素（set1减去set1和set2的交集）

```
#运算符
print(set1-set2)
#方法
print(set1.difference(set2))
```

- 对称差集：取两个集合的交集，然后根据交集取两个集合的补集，最后将补集取并集（并集减去交集），即去除两个集合的交集，各自剩下的元素组成一个新的集合

```
#运算符
print(set1^set2)
#方法
print(set1.symmetric_difference(set2))
```

举例：利用集合来判断两个字典的差异。

```
expect={"name":"张三","age":20,"sex":"男"}
actual={"name":"李四","heigh":170,"sex":"男"}
#实现断言：如果预期和实际不一致，要打印出它们的差异，输出：预期是{"name":"张三","age":20}实际是{"name":"李四","heigh":170}
set_expect=set(expect.items())
set_actual=set(actual.items())
print(set_expect)
print(set_actual)
print("预期是{}实际是{}".format(dict(set_expect-set_actual),dict(set_actual-set_expect)))
```

3.9.3 集合的常用方法（集合的方法晚上布置作业自己去练习）

- add(x):
  - 作用：将元素x添加到集合中，如果元素已存在，则不进行任何操作
  - 举例：

```
set1 = {1,2,3,4,5}
set1.add("nihao") #不能添加列表和字典
print(set1)
```

- update(x):
  - 作用：可以添加元素，且参数可以是列表，元组，字典等 除了数值型
  - 举例：

```
set1 = {1,2,3,4,5}
set1.update([1,6,7])#要添加可迭代的对象，添加字典时，只添加key值
print(set1)
```

- remove(x)方法:
  - 作用：将元素x从集合中移除，如果元素不存在，则会发生错误
  - 举例：

```
#remove
set1 = {1,2,3,4,5}
set1.remove(3) #不会返回删除的值
print(set1)
```

- discard(x)方法:
  - 作用：移除集合中的元素，且如果元素不存在，不会发生报错
  - 举例：

```
#discard
set1.discard(3) #也不会返回删除的值
print(set1)
```

- pop():
  - 作用：随机删除集合中的一个元素
  - 举例：

```
#pop
set1.pop()#不用传参数，且随机删除一个元素(每次删除的都是集合排序里的第一个元素)，并返回删除的元素
print("删除前",set1)
print(set1.pop())
print("删除后",set1)
```

- del:
  - 作用：删除整个集合
  - 举例：

```
#del,删除整个集合
del set1
print(set1) # 删除后打印会报错
```

- len()函数:
  - 作用：计算集合s元素的个数
  - 举例：

```
set1 = {1,2,(2,3),"hello"}
print(len(set1))
```

- copy():
  - 作用：复制集合
  - 举例：

```
set1 = {1,2,(2,3),"hello"}
set2 = set1.copy()
print(set1,set2)
```

- clear():
  - 作用：清空集合s
  - 举例：

```
set1 ={1,2,4}
set1.clear()
print(set1)  # 清空过后为空集合set()
```

3.10 布尔类型bool

- 布尔型bool：布尔型只有两个结果即True和False，在python中，布尔型跟数值型参与运算时，True相当于1，False相当于0
- 可以用实例化bool类的对象把一个对象转成布尔值，在python中，空字符串、空列表、空元组、空字典、空集合都被当做是False；非空即True

```
a=True
print(type(a))
print(a+5)
print(False*100)
print(bool(0))
print(bool(111))                #True
print(bool("fff"))              #True
print(bool([1,2,4]))            #True
print(bool(3,))                 #True
print(bool({"A":1}))            #True
print(bool({3,'a'}))            #True
print(bool(""))                 #False
print(bool([]))                 #False
print(bool(()))                 #False
print(bool({}))                #False
```

```
print(bool(set()))              #False
print(type({}))
print(type(set()))
```

3.11 特殊类型None

- None：是python中的一个特殊的数据类型，在函数没有返回值的时候，输出函数的执行结果得到的就是None。

```
print(type(None))
list1=["nihao",2,3]
print(list1.remove(2)) #删除函数，这个只是删除数据没有返回数据，所以打印的是None
```

- None与0，null，空字符串，空列表不同
  - None与其他任何数据类型比较都返回False

```
print(len(None)) #空，没有长度
print(len(""))#0,长度为0
if 0 and None:
    print(1)
else:
    print(2)
print("" == None)
```

  - null为java里的空，和None一样，都是没有元素

3.12 深拷贝和浅拷贝

- 拷贝就是一个变量的值传给另外一个变量。在python 中 id()方法可以查看存放变量的内存地址，
- 浅拷贝是指把存放变量的地址值传给被赋值，最后两个变量引用了同一份地址；
- 深拷贝是指被赋值的变量开辟了另一块地址用来存放要赋值的变量的值（内容）。
- 区别：
  - 浅拷贝可以使用列表自带的copy()函数（如list.copy()），或者使用copy模块的copy()函数。深拷贝只能使用copy模块的deepcopy(),所以使用前要导入：from copy import deepcopy
  - 如果拷贝的对象里的元素只有值，没有引用，那浅拷贝和深拷贝没有差别，都会将原有对象复制一份，产生一个新对象，对新对象里的值进行修改不会影响原有对象，新对象和原对象完全分离开。
  - 如果拷贝的对象里的元素包含引用（像一个列表里储存着另一个列表，存的就是另一个列表的引用），那浅拷贝和深拷贝是不同的，浅拷贝虽然将原有对象复制一份，但是依然保存的是引用，所以对新对象里的引用里的值进行修改，依然会改变原对象里的列表的值，新对象和原对象完全分离开并没有完全分离开。而深拷贝则不同，它会将原对象里的引用也新创建一个，即新建一个列表，然后放的是新列表的引用，这样就可以将新对象和原对象完全分离开。
- 浅拷贝：.copy()方法
- 深拷贝：.deepcopy()方法

```
import copy
a = [1, 2, 3, 4, ['a', 'b']]  #原始对象
b = a  #赋值，传对象的引用
c = copy.copy(a)  #对象拷贝，浅拷贝
d = copy.deepcopy(a)  #对象拷贝，深拷贝
```

```
print ('a = ', a,"id(a)=",id(a),"id(a[4])=",id(a[4]),id(a[1]))
print ('b = ', b,"id(b)=",id(b),"id(b[4])=",id(b[4]),id(a[1]))
print ('c = ', c,"id(c)=",id(c),"id(c[4])=",id(c[4]),id(a[1]))
print ('d = ', d,"id(d)=",id(d),"id(d[4])=",id(d[4]),id(a[1]))
a.append(5)  #修改对象a
a[4].append('c')  #修改对象a中的['a', 'b']数组对象
print ('a = ', a,"id(a)=",id(a),"id(a[4])=",id(a[4]),id(a[1]))
print ('b = ', b,"id(b)=",id(b),"id(b[4])=",id(b[4]),id(a[1]))
print ('c = ', c,"id(c)=",id(c),"id(c[4])=",id(c[4]),id(a[1]))
print ('d = ', d,"id(d)=",id(d),"id(d[4])=",id(d[4]),id(a[1]))
```

3.13 数据类型总结

3.13.1 可变和不可变类型（对象）

所谓可变对象，是指对象的内容是可变的，比如修改对象的内存时对象的内存地址不改变，例如 list。而不可变的对象则相反，当改变它的内容时对象的内存地址也会改变。

- 可变数据类型：List（列表）、Dictionary（字典）、Set（集合）
- 不可变数据类型：Number（数字）、String（字符串）、Tuple（元组）

举例：

```
num1=100
print(id(num1))
num1=101
print(id(num1))
list1=[1,2]
print(id(list1))
list1.append(3)
print(list1)
print(id(list1))
```

3.13.2 可迭代和不可迭代的类型（对象）

- 可迭代即可以重复的取出数据，存在__iter__()方法的，就是可迭代的对象
  - 可迭代：List（列表）、Dictionary（字典）、Set（集合）、String（字符串）、Tuple（元组）
  - 不可迭代：Number（数字）、bool（布尔类型）
- 举例：

```
#数据类型，后面能有__iter__()方法，就是可迭代的
[].__iter__()
"".__iter__()
{2,1}.__iter__()
{"name":"小明","age":18}.__iter__()
```

3.13.3 有序的和无序的类型（对象）

- 所谓序列，指的是一块可存放多个值的连续内存空间，这些值按一定顺序排列，可通过每个值所在位置的编号(称为索引)访问它们。
    - 有序：List（列表）、String（字符串）、Tuple（元组）
    - 无序：Dictionary（字典）、Set（集合）

四、Python运算符

- 算数运算符
- 比较运算符
- 逻辑运算符
- 三目运算符
- 身份运算符
- 成员运算符
- 赋值运算符

4.1 算数运算符

Python中的算术运算符：

- +
- -
- *
- /
- //（整除，不要余数部分只要整数部分的商）
- %(取余、取模)
- **（幂运算）

举例：

```
print(5*2) # 5乘以2
print(5/2) # 5除以2
print(5//2) # 5除以2取商
print(5%2) # 5除以2取余
print(5**3) # 5的3次方
#开方呢？125开3次方
print(125**(1/3))
#字符串、列表、元组的拼接
print("hello"+"world")
print("hello"*3)
print([1,6,8]+[1,6,8])
print([1,6,8]*3)
print((1,6,8)+(1,6,8))
print((1,6,8)*3)
```

4.2 比较运算符

Python中的比较运算符：

- >
- <
- >=

- <=
- ==
- !=（不等于）
- <>（在python2.x中这个也表示不等于，在python3.x中不支持）

比较运算的结果是布尔值。

举例：

```
print(5>7)
print(5>=7)
print(5<=7)
print(5!=7)
print(5==5)
```

4.3 逻辑运算符

Python中的逻辑运算符：

- and：逻辑与，表示两个条件同时满足
- or：逻辑或，表示两个条件满足其中一个
- not：逻辑非，表示取反

```
#True and True   --- True
#True and False   --- False
#False and True   --- False
#False and False  --- False
#True or True   --- True
#True or False   --- True
#False or True   --- True
#False or False  --- False
#not True     --- False
#not False     --- True
# 优先级：括号>not>and>or
if 100>50 and (100<90 or 100!=200):
    print("OKkkkk")
else:
    print("Nok")
```

4.4 身份运算符

- is：判断两个对象是同一个对象
- is not：判断两个对象不是同一个对象

is比较的是两个变量的内存地址，is not和is相反。运算结果是布尔值。

```
a="hello world"
b="hello world"
print(id(a),id(b))
print(a is b)
```

is与==的区别： is比较的是内存地址，是判断两个对象是否是同一个对象，==比较的是值，是判断两个对象的值是否相等。

4.5 成员运算符

Python中的成员运算符：

- in
- not in

格式：a in b，表示判断a是否是b的成员

```
if 'a' in 'hashdaskak':
    print("在的")
if "b" not in 'hashdaskak':
    print("不在")
#in经常用在for循环中，遍历元素
for i in [1,2,3,4,5,6,7,8]:
    print(i)
#判断某个值是否在字典的value中
dict1={"name":"张三","age":20}
if "张三" in dict1.values():
    print("在里面")
```

4.6 赋值运算符

- =
- +=　扩展后的赋值运算符
- -=
- *=
- /=
- //=
- %=
- **=

```
# 赋值数字
a,b,c=2,3,4
# 赋值字符串(“”，引号里面的都是字符串)
name ="nihao,小明"
# 变量交换
a = 6
b = 9
#一行代码实现交换两个变量的值
a,b = b,a
print(a,b)
# 赋值运算
a +=1  #相当于a=a+1
b -=1
print(a,b)
a *=2
b /=3
print(a,b)
a //=4
b %= 6
print(a,b)
# 赋值逻辑运算and or not （返回为true和false）
qw = 4
er = 6
rt = 10
```

```
print(qw > er or er < rt)
print(rt < er and qw < er)
print(not er < qw)
# 赋值关系运算
a = 5 <4
b =4 !=4
print(a,b)
```

4.7 三(目）元运算符

语法：变量名= 结果值1 if bool表达式 else 结果值2

举例：

```
#用if...else...语句实现
year1 = int(input("请输入年份："))
if year1%2 ==0:
    print("偶数")
else:
    print("奇数")

#三元运算符格式
year1 = int(input("请输入年份："))
print("偶数" if year%2==0 else "奇数")
```

五、流程控制

- 1.条件判断
- 2.循环

5.1 条件判断

条件判断用if语句实现，if语句的几种格式：

- 1）if...表示满足条件，则执行相应的操作，否则什么都不执行；
- 2）if...else...表示如果满足条件则执行if后面的操作，否则执行else后面的操作；
- 3）if...elif...elif...else...表示多条件判断，如果满足第一个条件就执行第一个操作，如果满足第二个条件就执行第二个操作，...，如果都不满足就执行else后面的操作。

注意：if可以独立存在，但else必须跟if配对，else是跟它前面最近的同级的if配对。

python中的代码缩进：在python中相同缩进的代码属于同级，一般上下级之间缩进4个空格（按1次Tab）

5.1.1 if.....

作用：如果if后面的布尔表达式值为True就执行if下的代码块，如果为False就不执行

格式：

if 布尔表达式:

代码块

举例：

```
#从键盘输入一个数字，如果大于10就输出“大于10”
num1=int(input("请输入一个数字："))
if num1>10:
    print("大于10")

#结合布尔值（为True则执行，为False则不打印
if 8：
    print("你好")
if "不为空"：
    print("真的")

#结合成员运算符（in、not）-满足条件执行print，不满足则不执行
str1 = "adgcgag"
if "ad" in str1:
    print("在里面")
```

5.1.2 if.....else...

作用：如果if后面的布尔表达式值为True就执行if下的代码块，如果为False就执行else下的代码块。

格式：

if 布尔表达式:

代码块

else:

代码块

举例：

```
#从键盘输入一个数字，如果大于10就输出“大于10”，如果不大于10就输出“不大于10”
num1=int(input("请输入一个数字："))
if num1>10:
    print("大于10")
else:
    print("不大于10")

#从键盘输入一个整数，如果整数是奇数，就打印奇数，如果不是奇数，就打印偶数
third=int(input("请输入一个整数："))
if third%2==1:
    print("奇数")
else:
    print("偶数")
#注意if和else是同级的，应该对齐。
```

5.1.3 if....elif....elif....else...

作用：从第一个条件开始逐一判断，遇到第一个满足的条件就执行它下面的语句，并结束判断不再往下判断。

格式：

if 布尔表达式:

代码块

elif 布尔表达式:

代码块

...

else:

代码块

举例：

```
#从键盘输入一个数字，如果大于10就输出“大于10”，如果小于10就输出“小于10”，否则输出等于10
num1=int(input("请输入一个数字："))
if num1>10:
    print("大于10")
elif num1<10:
    print("小于10")
else:
    print("等于10")
#举例：当遇到某个条件满足时就不会再往下判断
forth=int(input("请输入一个整数："))
if forth<10:
    print("小于10")
elif forth>=10:
    print("大于10")
elif forth>=20:
    print("大于20")
else:
    print("其它")
```

5.1.4 练习

1. 掷骰子表演节目，如果是1就唱歌，2就跳舞，3就讲相声，4就俯卧撑，5就劈叉，6就胸口碎大石。点数从键盘输入。
2. 从键盘输入一个年份，判断它是闰年还是平年？闰年：普通年能被四整除且不能被100整除的为闰年，世纪年能被400整除的是闰年。（判断普通年、世纪年和平年）

5.2 循环

- 1）while循环
- 2）for循环

5.2.1 while循环

while循环：当满足循环条件时就执行循环体里的代码，直到循环条件不满足为止。else是可选的，如果有else，那么else后面的代码会在循环正常结束后执行。使用while循环一定要有修改循环条件的语句，否则会造成死循环（在某条件下，一直执行语句，直到条件为False）。

格式：

```
while 布尔表达式:
    循环体
[else:
    语句块]
```

举例：

```
# 不带else语句
i=1  #给个初始值
while i<=10:    #1.条件限制
    print(i)
    i+=1    #2.条件，递增
#两个条件要结合，如果不限制，会导致死循环
#带else语句
i=1
while i<=10:#1.条件限制
    print(i)
    i+=1 #2.条件，递增
else:
    print("打印完毕")

#实现1+2+3+...+100，并输出计算结果
num1=0
i=1
while i<=100:
    num1+=i
    i+=1
else:
    print(num1)

#思考：1*2*3*...*100 ？
```

1.当布尔表达的值为True的时候，就执行循环体的代码，直到布尔表达的值为False的时候或者被break;

2.else语句是可选的，如果有else语句，当循环正常结束（不是被break语句结束的情况）后会执行else后面的语句，如果循环是被break掉，就不会执行else后面的操作。

5.2.2 for循环

迭代(Iteration)：指重复执行某个操作，迭代最常用的表现就是遍历，经常用for循环来遍历可迭代对象（Iterable），常见的可迭代对象包括字符串、列表、元组、集合和字典等。

结构：

```
for 变量名 in 可迭代对象:
    循环体
[else:
   语句块]
```

举例：

```
#1.将字符串"helloworld"遍历出来添加到列表list1中
list1=[]
for i in "helloworld":
    list1.append(i)
print(list1)

#2.遍历元组中的元素
tuple1 = (1,3,5,6,7)
for i in tuple1:
    print(i)
```

```
else:
    print("遍历完成")

#3.用for循环遍历字典
dict1={"姓名":"张三","性别":"男","年龄":20}
#遍历字典，默认取到的是字典的key
for i in dict1:
    print(i)
#遍历字典的key
for i in dict1.keys():
    print(i)
#遍历字典的value
for i in dict1.values():
    print(i)
#遍历字典的键值对
for i,j in dict1.items():
    if i=="年龄":
        print(i,j)

#4. 用for循环实现1+2+3+...+100，并输出计算结果
```

while循环和for循环的区别：while循环执行前，循环次数是不确定的；for循环在循环前执行次数是已经确定的。

5.2.3 循环的嵌套

- while中套while循环

  举例：

  ```
  # 在10中找两个数i和j，满足条件j <= (i / j)时，如果i能除尽j则打印“能除尽”并打印i和j的值，如果不满足则打印“不满足条件”并打印不满足条件的i和j的值
  i = 2
  while (i < 10):
      j = 2
      while (j <= (i / j)):
          if not (i % j):
              print("能除尽",i,j)
          j = j + 1
      else:
          print("不满足条件",i,j)
      i = i + 1
  print("算完")
  ```

- for循环中套for循环

  举例：

  ```
  # 九九乘法表
  for i in range(1,10): # 依次取数1到9
      for j in range(1,i+1): # 根据i的取值来取值，i取1时,j的范围也是1
          print("{} x {} = {}\t".format(j,i,i*j),end="") # 根据上面两个循环取出的书打印格式化数据。
      print()# 把一次循环的数据打印完成后换行
  ```

- 除此之外，还可以在while循环中嵌套for循环，以及在for循环中嵌套while循环。

5.2.4 循环中的continue/break/pass语句

- continue
- break
- pass

5.2.4.1 continue语句

- 作用：continue的作用是结束本次循环（在某一次循环中遇到continue语句时，直接结束本次循环，不执行continue后面的代码，而开始执行下一次循环）。

  举例：

```
# 在0到10的数字中打印所有奇数，不是奇数的就不打印出来
a = 0
while a < 10:
    a += 1
    if a%2 ==0:
        continue    # 跳过本次循环，进入下一次循环
    print("奇数",a)
else:
    print("没有奇数了")

#3、输出0到100之间被3整除的整数，如果这个数能被5整除则不输出
for i in range(101):
    if i%3==0:
        if i%5==0:
            continue
        print(i)
```

5.2.4.2 break语句

- 作用：是结束本层循环，可以跳出for和while的循环体(所有循环)，任何对应的循环else块将不执行。

  举例：

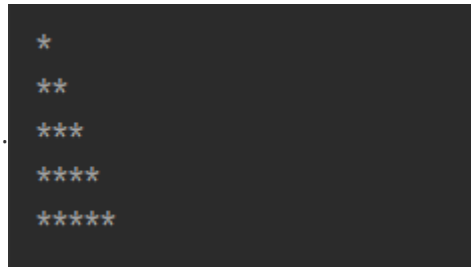
```
# 整数0到10中，取能被2整除的数，取到第一个数就结束循环，不执行后面的代码
a = 0
while a < 10:
    a += 1
    if a%2 ==0:
        break # 满足上面条件了，直接结束本层所有循环，不执行后面的语句
    print("奇数",a)
else:
    print("没有奇数了")
```

5.2.4.3 pass语句

- 作用：是空语句，是为了保持程序结构的完整性。不做任何事情，一般用做占位语句

  举例：

```python
# 输出 Python 的每个字母
for str1 in 'Python':
    if str1 == 'h':
        pass # 占位，不做任何操作
    print(str1)
```

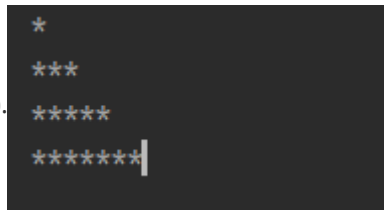
- 思考：
  - 求100以内的偶数
  - 100以内的奇数
  - 打印图形
    1. 

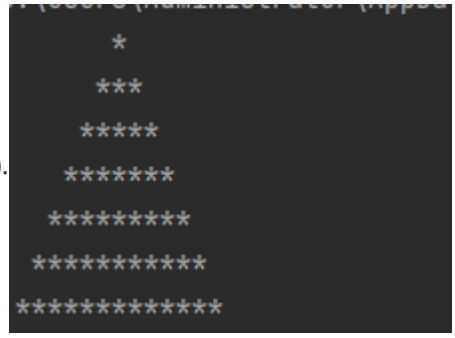
- 作业：

  1.打印九九乘法表？？

  2.1到100的和?

  3.打印图形?

  b). 

  c). 

  d). 

  答案：

  1.

```python
# 九九乘法表
for a in range(1,10):
    for b in range(1,a+1):
        print("{} x {} = {}\t".format(b,a,a*b),end="")
    print()
```

```python
# 打印1到100相加的总和
# 方法1
'''
a = 1
b = 0
while a < 101:
    b = a + b
    a = a + 1
    print(b)
'''
# 方法2
'''
a = 0
list1 =[]
while a <=99:
    a +=1
    list1.append(a)
    print(sum(list1))
'''
# 方法三
'''
print(sum(range(1,101)))
```

```python
# 每层不同的*数量，依次递增(差一）
for a in range(6):
    print("*"*a)
# 每层不同，递增差二
#1.方法1
for b in range(1,8,2):
    print("*"*b)
#2.方法二
b = 5
for b in range(1,b+1):
    print("*"*(2*b-1))

# 金字塔
cengshu = 7
for b in range(1,cengshu+1):
    print(" "*(cengshu-b),"*"*(2*b-1))

# 金字塔，中间为空
for i in range(6):  #循环6次打印6行
    for j in range(5-i):   #打印空格每次循环递减
        print(' ',end='')
    for q in range(2*i+1): #打印星星
       if q==0 or  q==2*i: #判断打印星星位置，在开头，结尾和最后一行打印星星
           print('*',end='')
```

```
        else:
            print(' ',end='')
    print() #每行循环结束后换行
```

5.2.4.4 练习

1、猜数字游戏，系统随机产生一个数，输入一个数和系统的数进行比较，如果大了则打印你输入的数字大了，小了则提示输入小了，猜中则提示恭喜你答对了。

```
import random
num = random.randint(1,100)
while True:
    num1=int(input("输入数字："))
    if num1<num:
        print("小了")
    elif num1>num:
        print("大了")
    else:
        print("中奖了")
    break
```

5.2.4.5 作业

1. 猜数字游戏，限制只能猜5次
2. 输出0到100之间被3整除的整数，如果这个数能被5整除则不输出
3. 从键盘输入两个整数，实现这两个数之间的数求和，比如输出5和2，则计算2+3+4+5
4. 用循环实现一个一维整数列表的升序排序；比如：[5,2,1,7,6]---->[1,2,5,6,7]
5. 从键盘读入一串字符串，输出不重复的字符

六、推导式

- 列表推导式
- 字典推导式
- 集合推导式
- 生成器推导式

6.1 列表推导式

- 列表推导式的语法：[结果 for 变量 in 可迭代对象] 或者 [结果 for 变量 in 可迭代对象 if 布尔表达式]

  举例：

  ```
  # 普通形式
  ## 以列表的形式输出100以内的正偶数
  list1=[]
  for i in range(1,101):
      if i%2==0:
      list1.append(i)
  print(list1)

  # 列表推导式
  ## 用列表推导式完成
  ```

```
print([i for i in range(1,101) if i%2==0])
## 列表推导式不加if
print([i for i in range(2,101,2)])
#统计字符串中只出现一次的字符，以列表返回字符串
str1="helloworld"
print([i for i in str1 if str1.count(i)==1])
```

6.2 字典推导式

- 语法：{结果 for 变量 in 迭代对象} 或者 {结果 for 变量 in 迭代对象 if 布尔表达式}；注意字典推导式的结果是**键值对**，即key:value

  举例：

```
tupl1=(("姓名","张三"),("年龄",20),("体重",190),("身高",180))
print({i:j for i,j in tupl1})
#交换key和value的位置
print({j:i for i,j in tupl1})
#加判断条件
print({i:j for i,j in tupl1 if j !=190})
#统计字符串中每个字符出现的次数，以字典返回
str1="helloworld"
print({i:str1.count(i) for i in str1})
#统计字符串中只出现一次的字符，以字典返回字符及出现次数
str1="helloworld"
print({i:str1.count(i) for i in str1 if str1.count(i)==1})
```

6.3 集合推导式

- 语法：{结果 for 变量 in 迭代对象} 或者 {结果 for 变量 in 迭代对象 if 布尔表达式}；集合推导式跟字典推导式的区别是：字典推导式的结果是键对，集合推导式的结果是单个结果

  举例：

```
dict1={"姓名":"张三","年龄":20,"体重":180,"身高":180}
print({x for x in dict1.keys()})
print({x for x in dict1.values()})
# 加if判断,只取int类型的数据
print({x for x in dict1.values() if type(x)==int})
```

6.4 生成器推导式

- 语法：(结果 for 变量 in 可迭代对象)或者(结果 for 变量 in 可迭代对象 if 布尔表达式)

  举例：

```
d = (a**2 for a in range(1,10))
#用元组形式展示数据
print(tuple(d))
```

七、函数（最主要的目的就是封装一个功能）

- 定义：函数就是完成特定功能的一个语句组，这组语句可以作为一个单位使用，并且取一个名字，这个名字叫做函数名

- 函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段

7.1 函数的作用

- 可以通过函数名在程序不同的地方多次调用，这样就不用在不同的地方重复编写这些语句
- 函数是实现特定功能的一段代码，通过函数可以方便地重复使用代码，让程序的结构更清晰，方便代码的走读。
- 函数能提高应用的模块性，和代码的重复利用率

7.2 内置函数

函数分为两种：

- 1.系统内置函数

  比如len()/max()/min()/print()/ord()/chr()/...；

  - input函数举例：
    - python3.x从控制台接收输入内容：用input()函数实现输入
    - 2.x版本中使用的是raw_input
    - 语法：input("提示语")
    - 注意input函数默认把输入的内容当作字符串处理，如果要转换成整数，需要用强制数据类型转换比如int(a)

```
num = input("请输入密码：")
print(type(num)) # 打印输入信息的数据类型
```

7.3 自定义函数

7.3.1 函数的声明与调用

- 定义函数的语法：

  def 函数名(参数列表):

  　　语句体

  函数名（）

  说明：

  - 函数代码块以def关键字开头，后接函数名称和圆括号（），圆括号中间放传入的参数，也可以不传入参数，()后面加一个英文的冒号；
  - 函数的第一行语句可以选择性地添加注释，用来解释函数的作用、参数、返回值等；
  - return[表达式]结束函数，选择性地返回一个值给调用方。不带表达式return[]相当于返回None。

  举例：

```
# 定义函数
def speak():
    name = "小明"
    print("你好！{}".format(name))
# 调用函数
speak()
```

7.3.2 return语句

- return[表达式]语句用于**结束函数**，选择性的向调用方返回一个表达式或者一个值。不带参数值得return语句返回None
- 情况一：同级的多个return只返回第一个return的值，因为函数执行时遇到第一个return返回后就结束了
- 情况二：条件判断中，只返回第一个满足条件的第一个return的值，如果都不满足，则返回None
- 举例：

```
# 情况一：同级的多个return只返回第一个return的值
def add(a=1,b=2):
    print(a)
    print(b)
    return a+b
    return b-a
print(add())

# 条件判断中，只返回第一个满足条件的第一个return的值，如果都不满足，则返回None
def add(a,b):
    if a>b :
        return a-b
    if a<b :
        return a+b
print(add(3,4))
print(add(6,3))
print(add(4,4))
```

7.4 函数中的参数

7.4.1 形参和实参

- 形参：在定义声明时所传入的参数
- 实参：在调用函数时所传入的参数

```
#name，age为形参，定义时
def speak(name,age):
    print(name)
    print(age)

#中间的值为实参，即调用时传入的值为实参
speak("小马",18)
```

7.4.2 位置参数（也叫必须参数）

- 在调用函数时，必须以形参的顺序传入实参，调用时的数量必须和声明时的一样，不能传多也不能传少。这种传入参数的方式叫位置参数。

```
#声明一个函数
def speak(name,age,like):
    print("小明的女朋友{},今年{},喜欢{}".format(name,age,like))

#调用函数
speak("小茵",18,"逛街")
```

7.4.3 关键字参数

函数调用时指定形参名称来传入实参，传入的实参顺序可以与声明的形参顺序不一致，但不能传多也不能传少，这种方式叫关键字参数。（Python 解释器能够用参数名匹配参数值）

```
#声明一个函数
def speak(name,age,like):
    print("小明的女朋友{},今年{},喜欢{}".format(name,age,like))

#调用函数
speak(name="小茵",age=18,like="逛街")
speak(like="逛街",name="小茵",age=18)
```

7.4.3 默认参数（也叫缺省参数）

- 默认参数就是在声明函数的时候，给参数一个默认值，在调用时如果不对这个参数传值，就使用默认值，如果对这个参数传值，就使用实际传入的值；
- 如果在声明函数的时候，有默认参数，需要把默认参数放到位置参数之后。

```
def speak(name,age,like="看电影")
    print("小明的女朋友{},今年{},喜欢{}".format(name,age,like))

speak("小茵",18,"逛街") #可改默认参数的值
```

7.4.4 不定长参数

不定长参数包括不定长位置参数和不定长关键字参数两种形式，分别在形参前加*和**来表示。不定长参数在实际调用时，可以传入0个、1个或多个实参。（你可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，和上述 2 种参数不同，声明时不会命名。加了星号 * 的参数在调用时会以元组(tuple)的形式导入，存放所有未命名的变量参数）

不定长位置参数：形参前加*，可以接收任意多个实参，接收的实参以元组的形式参与函数内的运算。

声明：

def 函数名(*args):

函数体

举例：

```
#声明一个不定长参数函数
def f_1(*args):
    print(args)
```

```
    return sum(args)

#方法一：调用不定长参数的函数，可以传入0个、1个或多个参数，传入的参数是按元组参与函数运算
print(f_1())
print(f_1(1))
print(f_1(1,2))
print(f_1(1,2,3))

#方法二：调用不定长参数的函数，将多个参数放到列表、元组或集合中传入，但需要在列表、元组或集合前加*来对列表或元组解包，把列表中的元素作为多个参数一一传给函数
# print(f_1([1,2,3]))      #错误的
print(f_1(*[1,2,3]))
print(f_1(*(1,2,3)))
print(f_1(*{1,2,3}))
```

不定长关键字参数：形参前加，可以接收任意多个实参，接收的实参以字典的形式参与函数内的运算。**

声明：

def 函数名(**args):

函数体

举例：

```
#声明一个不定长关键字参数函数
def f_1(**kwargs):
    return kwargs

#方法一：调用不定长参数的函数，可以传入0个、1个或多个参数，必须以关键字参数的形式传入，传入的参数是按字典的形式参与函数运算
print(f_1())
print(f_1(name="张三"))
print(f_1(name="张三",age=20))
print(f_1(name="张三",age=20,sex="男"))

#方法二：调用不定长参数的函数，将多个参数放到字典中传入，但需要在字典前加**
# print(f_1({"name":"张三"}))      #错误的，原因是必须以关键字参数传入
print(f_1(**{"name":"张三"}))
print(f_1(**{"name":"张三","age":20}))
print(f_1(**{"name":"张三","age":20,"sex":"男"}))
```

注意：在函数内同时使用不定长位置参数（*）以及不定长关键字参数（**）时，不定长位置参数必须在不定长关键字参数之前。

举例：

```
def f(*a,**b):
    print(a)
    print(b)
f(1,2,34,5,d=4,e=6,g="你好")
```

7.4.5 位置参数、默认参数以及不定长参数同时使用

- 如果在声明和使用函数时，同时存在位置参数、默认参数以及不定长参数，必须遵守以下顺序：1.位置参数；2.不定长位置参数；3.默认参数；4.不定长关键字参数。

```
# 格式化字符串-混合传值
def person(a, *tuple1,like="篮球",**dict1):
    print(a)
    print(dict1)
    print("{name}今年{age}岁了，每{}天打{}次{}".format(a,*tuple1,like,**dict1))

#调用函数
person(2,*(3,),"足球",**{"name":"小王","age":30}) # 调用函数并传参

# 定义函数时传不同的参数类型，调用函数时传一组元素，会自动将元素放到对应的参数类型中
def f(a,*b,d1=9,**c):
    print(a) # 必须参数
    print(b) # 元组存数据
    print(c) # 字典存数据
    print(d1) # 默认参数9
#调用函数
f(1,2,34,5,8,d=4,e=6,g="你好")
```

7.5 匿名函数（lambda）

- 又称之为高效函数，一次性函数，丢弃函数，也叫lambda函数（没有函数名的函数）；因为在声明的时候可以直接调用（不需要先声明（定义）然后再调用）。而普通函数是先定义后再进行调用进行输出，比如是def 函数名（）：后面用的时候才进行调用。
- 匿名函数的定义语法：lambda 变量名.... : 语句表达式
- 匿名函数的调用：直接用括号把匿名函数括起来，在后面再用括号传参

```
# 求两个数的和
#普通的函数
def f_1(x,y):
    return x+y
print(f_1(3,7))

#匿名函数
print((lambda x,y:x+y)(3,7))
```

- 匿名函数举例

```
#例子1：传入两个整数参数，以列表返回两个整数之间所有整数
x=int(input("输入x："))
y=int(input("输入y："))
print((lambda x,y:list(range(x,y+1)) if x<y else list(range(y,x+1)))(x,y))

#例子2：传入两个整数参数，求该列表中的所有数的和
x=int(input("输入x："))
y=int(input("输入y："))
print(sum((lambda x,y:list(range(x,y+1)) if x<y else list(range(y,x+1)))
(x,y)))
```

7.6 函数的递归

- 函数内部可以调用其他函数，当然在函数内部也可以调用自己，在函数内部调用自己叫着函数的递归
- 函数内部的代码是相同的，只是针对参数不同，处理的结果不同，当参数满足一个条件时，函数不再执行。这个非常重要，通常被称为递归的出口，否则会出现死循环。

```
#举例1：通过递归实现阶乘，声明一个函数，传入参数n，实现1*2*…*n
def f_1(n):
    print(n)
    if n==1:
        return 1
    return n*f_1(n-1)
print(f_1(10))

#举例2：通过递归实现求和运算，实现声明一个函数，传入参数n，实现1+2+...+n
def f_2(n):
    if n==1:
        return 1
    return n+f_2(n-1)
print(f_2(10))
```

7.7 函数的全局变量和局部变量

- 全局变量：在函数的外面定义的变量就叫全局变量，所有函数可以调用；
- 局部变量：在函数的内部定义的变量叫局部变量，局部变量只能当前函数内调用。

```
a = "小明"  #全局变量
b = 1       #全局变量
def test():
    d = "篮球" #局部变量
    print("{}喜欢{}，每天可以打{}小时".format(a,d,b))
def test01():
    e = "老王"
    print("{}也喜欢打{},每天打{}小时".format(e,d,b))
test()
test01() #取不到d，会报错

##函数内部要修改全局变量，使用global来声明。注：只要是用global修改后，再调用的函数都传的是新值
a = "小明"  #全局变量
b = 1       #全局变量
def test():
    d = "篮球" #局部变量
    global b #声明要修改的全局变量
    b = 3
    print("{}喜欢{}，每天可以打{}小时".format(a,d,b))
def test01():
    e = "老王"
    print("{}也喜欢打球,每天打{}小时".format(e,d,b))
test()
test01()
```

7.8 作业

1. 定义一个函数，实现功能：给定两个参数min、max，把min到max之间的所有数输出；
2. 声明一个函数，实现功能：传入一个列表，把列表中的所有字符串拼接，把列表中的所有数字求和，并以一个元组返回拼接后的字符串和求和后的值。注意，列表中的元素可能是列表或元组等多层嵌套。
3. 函数求和：计算123....*10的结果

```
#计算1*2*3....*10的结果
def chengji(x, y):
    if x <= y:
        j = x
        k = 1
        while j <= y:
            k *= j
            j += 1
    else:
        j = y
        k = 1
        while j <= x:
            k *= j
            j += 1
    return k
print(chengji(1, 10))
```

4.
```
# 计算出1,2,3,4四个数的排列方式有多少种，并打印出来每种方式
list1 = []
for x in range(1,5):
    for y in range(1,5):
        for z in range(1,5):
            if x!=y and x!=z and y!=z:
                paixu = x,y,z
                list1.append(paixu)
print(len(list1),list1)
```

八、迭代、可迭代对象、迭代器、生成器

8.1 迭代

- 迭代Iteration：所谓迭代就是重复运行一段代码语句块的能力，就好比在一个容器中进行一层一层遍历数据，在应用过程中for循环最为突出。总结就是:迭代就是从某个容器对象中逐个地读取元素，直到容器中没有元素为止。迭代迭代，更新换代，在上一次基础上更新成新的东西

```
for i in "hishkfkhdsfdl":
    print(i)
```

8.2 可迭代对象

- 可迭代对象Iterable：可以被迭代的类型，怎么判断是否可迭代？所有的类型只要有__iter__()方法就是可迭代的。我们现在已知的可迭代对象有:(str，list，tuple，range，set，dict_keys，dict_values，dict_items)，怎么去判断某个对象是否有__iter__()方法?

  1）用dir()函数来查询是否包含__iter__()方法；

  2）对象.方法的方式去调用有没有；

  3）通过内置的实例对象函数isinstance判断，可迭代对象是Iterable类的实例，返回True就说明是可迭代对 象，False则表示不是可迭代对象。

```
#查看某个元素或序列是否有.__iter__()方法，有就是可迭代的对象
"hello world".__iter__()
[2].__iter__()
(1,).__iter__()
{"name":"jack"}.__iter__()
{"name":"jack"}.keys().__iter__()
range(11).__iter__()
list1=[]
a=9
#用dir()函数打印数据的全部方法，看看是否包含__iter__()方法
print(dir(list1))
#通过内置的实例对象函数isinstance判断
from collections.abc import Iterable,Iterator
print(isinstance(list1,Iterable)) ----结果是True
print(isinstance(a,Iterable)) ----结果是False
```

8.3 迭代器

- 迭代器Iterator：迭代器一定是可迭代对象，迭代器中有两个特殊的方法是__iter__()和__next__()方法，创建迭代器的方法：使用内置的iter()函数或者**iter**方法来创建迭代器；
- 创建迭代器

```
# 将列表转换为迭代器
list1=[1,3,5,7,9]
it1=iter(list1)或者Ist1 = list1.__iter__（）使用方法转
print(type(it1))
# 打印迭代器中的数据，每次只能打印一个元素，有多少元素就需要打印多少次
print(next(it1)) # 用next()函数取迭代器中的数据
print(it1.__next__()) # 用__next__()方法取迭代器中的数据
print(it1.__next__())
print(it1.__next__())
print(it1.__next__())
# print(it1.__next__())
# print(it1.__next__())
```

- 迭代器的特点是：
  - 1）迭代器一定是可迭代对象；
  - 2）迭代器通过__next__()方法取值，每迭代一次取一个值，只能往前取不能后退；
  - 3）当取完最后一个值的时候，再执行__next__()方法会报StopIteration异常，表示取完了。
- 判断一个对象是迭代器还是迭代对象：
  - 1)通过对象包含__iter__()和__next__()方法决定；

- 2)需要判定是迭代器还是迭代对象可以通过实例对象函数进行判定；迭代器和可迭代对象的对象类型分别是Iterator和Iterable（都是collections.abc模块下）

```
from collections.abc import Iterator,Iterable
# Iterator是判断是否是迭代器
# Iterable是判断是否是可迭代对象
print(isinstance(list1,Iterable))   #判断list1是否是可迭代对象返回True或False
print(isinstance(list1,Iterator))   #判断list1是否是迭代器返回True或False
print(isinstance(iter(list1),Iterator))         #True
print(isinstance(it1,Iterable))             #True
print(isinstance(it1,Iterator))             #True

#计算1+2+3+...+1000000000
#print(sum(list(range(1,1000000001))))
#print(sum(iter(range(1,1000000001))))
```

迭代器的优势是内存开销更小，它比一次性加载数据占用的内存更小，适用于大数据量的计算。

```
#计算1+2+3+...+1000000000

print(sum(list(range(1,100000001))))

print(sum(iter(range(1,100000001))))
```

for循环实现遍历的底层原理：（**for循环将其都封装好了，所以for循环in后面必须跟的是可迭代对象**）

1. for循环传入的一定是可迭代对象，在循环的时候先调用可迭代对象的__iter__()方法来生成迭代器；
2. 调用迭代器的__next__()方法来迭代每个元素；
3. 当迭代完最后一个元素时，再继续迭代会报StopIteration异常，for循环捕捉到这个异常就知道已经迭代完了，就结束循环。

8.4 生成器

生成器generator：在 python 中，生成器的本质就是一个迭代器，使用了yield 的函数被称为生成器（generator）。

8.4.1 return与yield的区别

- return的作用：
    1. 给调用者返回值；
    2. 执行遇到第一个return语句时就结束函数；
- yield的作用：
    1. 给调用者返回值；
    2. yield把函数变成了生成器；
    3. 生成器运行时遇到yield后先返回再挂起；

```
# 直接打印函数时，只会打印第一个值
def gen():
    yield 1
    yield 2
```

```
    yield 3
print(next(gen()))
print(next(gen()))
print(gen().__next__())

# 给函数重命名，可以打印所有,但是一次只会打印一个值
def generator():
    yield 1
    yield 2
    yield 3
gt=generator()
print(next(gt))
print(next(gt))
```

8.4.3 生成器怎么创建

- 1）通过关键字yield把函数变成生成器；
- 2）通过生成器推导式创建生成器(结果 for 变量 in 可迭代对象)

通过生成器推导式创建生成器

```
print(type((i for i in range(11))))
print((i for i in range(11)))#是一个生成器对象
```

8.4.4 生成器的运行

- 通过调用next()函数或者__next__()方法来运行生成器，next()函数实际也是调用__next__()方法

```
def generator():
    yield 1
    yield 2
    yield 3
gt=generator()
print(next(gt))

print(next(gt))

print(gt.__next__())

print(gt.__next__())
```

8.4.5 通过send()函数向生成器传入值，send()与next()函数的区别

- 1)next()和send()函数都用来执行生成器；
- 2)send()方法会首先把上一次挂起的yield语句的返回值通过参数设定，然后再执行生成器，从而实现与生成器方法的交互；
- 3)在执行生成器是，如果第一次执行使用send()函数，因为没有挂起的yield语句来接收传入的值，所以会报TypeError。

```
def generator():
    yield 1
    yield 2
    yield 3
gt=generator()
print(gt.send(100)) # 报错,需要先执行next()
```

```
print(gt.send(200))

def generator2():
  a=yield 1
  b=yield a
  yield b
  c=yield 2
  print(c)
  yield c

gt2=generator2()
print(next(gt2))
print(gt2.send(100))  #传入100给yield 1的执行结果，也就是a，然后再执行yield a,返回100，再挂起
print(gt2.send(2.5))  #传入2.5给yield a的执行结果，也就是b，然后再执行yield b,返回2.5，再挂起
print(gt2.send('abc')) #传入'abc'给yield b的执行结果，但并没有引用他，然后执行yield 2，返回2，再挂起
print(gt2.send('efh'))
```

8.4.6 生成器与迭代器的区别

- 生成器：
  生成器本质上就是一个函数，它记住了上一次返回时在函数体中的位置。
  对生成器函数的第二次（或第n次）调用，跳转到函数上一次挂起的位置。
  而且记录了程序执行的上下文。
  生成器不仅“记住”了它的数据状态，生成器还记住了程序执行的位置。
- 迭代器
  迭代器是一种支持next()操作的对象。它包含了一组元素，当执行next()操作时，返回其中一个元素。
  当所有元素都被返回后，再执行next()报异常—StopIteration
  生成器一定是可迭代的，也一定是迭代器对象
- 区别

1. 生成器是生成元素的，迭代器是访问集合元素的一中方式
2. 迭代输出生成器的内容
3. 迭代器是一种支持next()操作的对象
4. 迭代器（iterator）：其中iterator对象表示的是一个数据流，可以把它看做一个有序序列，但我们不能提前知道序列的长度，只有通过nex()函数实现需要计算的下一个数据。可以看做生成器的一个子集。
5. 调用一个生成器函数，返回的是一个迭代器对象。

九、面向对象

9.1 面向对象的基本概念

- 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。
- 面向对象：

  类class：类是抽象的，类具有事物的描述（属性，静态的特征）和操作（方法，动态的特征）；比如学生类，它具有的属性有姓名、年龄、身高、体重等，它具有方法有学习、吃饭、睡觉、打闹等等。

  对象object：对象是具体的事物，它具有特定事物的描述和操作，类相当于一个模板，对象就是根据这个模板创建出来的具体个体。

分析以下描述中的类和对象?

1.我有一部手机，品牌是苹果，型号是iPhone XS Max，颜色是土豪金，容量是256G，买它花了我15000块钱，这部手机可以打电话，刷抖音，还能吃鸡。

我：对象；手机：类；这一部手机：对象；品牌：属性；苹果：属性的参数（品牌=苹果）；型号：属性；iPhone XS Max属性的参数；颜色：属性；容量：属性；买：方法；打电话：方法；刷抖音：方法；吃鸡：方法

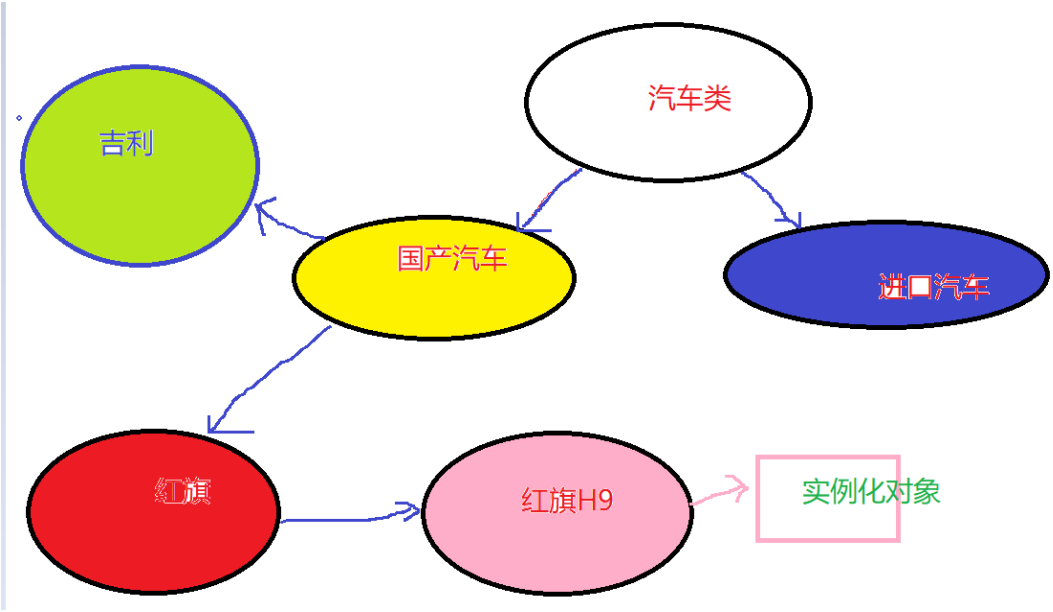
类：手机类 ---------->创建出来的对象是：这一部手机

属性：品牌、型号、颜色、容量---------->属性用变量来定义

方法：买、打电话、刷抖音、吃鸡---------->方法用函数来定义

根据类创建出来的对象，也具备类一样的属性和方法，可以根据类去创建多个对象。

- 如图举例：



9.2 类的声明与实例化

类是对客观世界中事物得抽象描述，而对象是类实例化后的实体

9.2.1 类的声明

- 使用class关键字定义一个类，类名一般用大驼峰表示
- 类的声明语法：

```
class 类名称:
    类的成员          #包括属性和方法
```

举例：

```
# 定义一个类
class People:
    #类的属性：用变量来表示
    type="高等生物"
    #类的方法：用函数来表示，但注意方法和函数两个概念有区别
    def eat(self):
        return "用筷子吃饭！"
```

9.2.2 属性和方法

- 属性和方法统称类的成员。
    1. 属性：类的静态特征叫属性，比如学生类的姓名、性别、年龄等特征就是静态特征，用变量来表示；
    2. 方法：类的动态特征叫方法，比如学生类的吃饭、睡觉、吃鸡等特征就是动态特征，用函数来表示；
- 举例：

```
# 定义一个类
class Students():
    #类的属性：用变量来表示
    type="学生"
    #类的方法：用函数来表示，但注意方法和函数两个概念有区别
    def study(self):
        return "学习软件测试技术！"
```

9.2.3 类的实例化（创建对象）

- 根据类创建对象的过程，就是类的实例化；一个类可以创建多个对象；创建的对象带有类的属性和方法。

```
# 定义一个类
class Students:
    #类的属性：用变量来表示
    type="学生"
    #类的方法：用函数来表示，但注意方法和函数两个概念有区别
    def study(self):
        return "学习软件测试技术！"
#实例化一个对象
Students()
print(id(Students()))

#实例化一个对象，并赋值给变量stu1，那么stu1就相当于对象的引用
stu1=Students()
print(id(stu1))
print(id(stu1))
#实例化另一个对象
stu2=Students()
print(id(stu2))
```

9.2.4 属性和方法的调用

- 因为类的对象有类的所有属性和方法，所以可以通过对象来调用类的属性和方法，调用的语法是：
    - 对象.属性
    - 对象.方法

```
# 定义一个类
class Students():
    #类的属性：用变量来表示
    type="学生"
    #类的方法：用函数来表示，但注意方法和函数两个概念有区别
    def study(self):
        return "学习软件测试技术！"
#实例化一个对象
stu1=Students()
#调用属性
print(stu1.type)
#调用方法
print(stu1.study())
```

9.2.5 对self的理解

- 方法中至少有1个参数self，并且self形参必须在第一个位置，但调用方法时又不需要给self参数传参。self表示对象本身。

```
# 定义一个类
class Students():
    #类的方法：用函数来表示，但注意方法和函数两个概念有区别
    def change_type(self,type):
        #如下表示调用对象的type属性进行操作，通过self.type实现调用
        self.type=type
        return self.type
#实例化一个对象
stu1=Students()
#调用方法
print(stu1.change_type("小学生"))
#实例化一个对象
stu2=Students()
print(stu2.change_type("中学生"))
```

9.2.6 构造方法和析构方法

1. 构造方法__init__()

- __init__是构造方法，在声明类的时候构造方法是可选的（也叫魔法方法）
- __init__在类中的作用是：创建对象，初始化对象。
- 当类中没有显式地声明构造方法的时候，系统会使用一个默认的不带参的构造方法来实现对象的创建，当我们需要对对象初始化的时候，就必须显式地声明一个带参的构造方法。

```
# 定义一个类
class Students():
    #类的属性：用变量来表示
    type="学生"
    def __init__(self,name,age):
        #实例的属性
        self.Name=name
        self.Age=age
```

```
    #类的方法：用函数来表示，但注意方法和函数两个概念有区别
    def study(self):
        return "学习软件测试技术！"

    def get_name(self):
        return self.Name

#实例化一个对象
stu1=Students("张三",20)
#调用属性
print(stu1.type)
print(stu1.Name)
print(stu1.Age)
#调用方法
print(stu1.study())
print(stu1.get_name())
```

9.2.7 函数与方法的区别

函数和方法的区别：

1. 函数定义在类外面，方法定义在类里面；
2. 函数通过函数名(参数)来调用，方法一般通过对象.方法名(参数)来执行；
3. 方法中必须有一个参数（通常命名为self）且这个参数一定在第一个位置，self表示对象本身，方法中的self参数在调用方法时不需要传实参；如果在函数声明时写一个叫self的形参，表示一个叫self的位置参数，在调用函数时必须对它传值。

9.3 类的封装

封装：类是封装属性和方法的载体。类封装的目的是实现访问控制（该暴露的就暴露，不该暴露的就隐藏）。在控制权限上，属性和方法分为public（公共的）和private（私有的）两种，默认是公共的。

9.3.1 属性的不同类型

9.3.1.1 类属性

类的属性可以通过类调用，也可以通过对象来调用，对于类的所有对象共同的属性，可以声明成类的属性。

```
# 定义一个类
class Students():
    #类的属性：用变量来表示
    type="学生"

    #类的方法：用函数来表示，但注意方法和函数两个概念有区别
    def study(self):
        return "学习软件测试技术！"

#实例化一个对象
stu1=Students()

#通过类调用类的属性
print(Students.type)
```

```
#通过对象调用类的属性
print(stu1.type)
```

9.3.1.2 实例属性（普通属性）

实例属性属于每个对象，每个对象的属性值可以不一样，实例属性通过对象来调用，实例属性是我们写得最多的属性。

```
# 定义一个类
class Students():
    #类的属性：用变量来表示
    type="学生"
    #构造方法
    def __init__(self,name,age):
        #实例的属性
        self.Name=name
        self.Age=age

#实例化一个对象
stu1=Students("张三",20)

#不能通过类调用实例的属性，因为这个属性属于对象而不属于类
print(Students.Name)
#通过对象调用实例的属性是Ok的
print(stu1.Name)
```

9.3.1.3 私有属性

属性和方法分为public（公共的）和private（私有的），默认是公共的。如果要声明私有的属性或方法，要在属性名或方法名前加__(两个下划线)，私有的方法不能通过类或者对象直接调用。

私有的属性或方法的调用方式：

1. 通过定义公有的方法来间接地调用私有的属性或者方法；
2. 通过_类名__属性/方法来调用，在python中没有绝对的私有，所谓的私有实际上是在名称前加了_类名而已。

```
# 定义一个类
class Students():
    # 类的属性：用变量来表示
    type = "学生"
    def __init__(self, name, age,weight):
        # 实例的属性
        self.Name = name
        self.Age = age
        #私有的属性
        self.__weight=weight
    #声明一个共有的方法来间接地调用私有方法
    def get_weight(self):
        return self.__weight
# 实例化一个对象
stu1 = Students("张三", 20, 80)
# 通过声明的共有方法来间接地实现私有属性的调用
print(stu1.get_weight())
# 通过在属性前加_类名来实现对私有方法的调用
```

```
print(stu1._Students__Weight)
```

在实际编程中，虽然Python无法实现绝对的私有，但我们可以通过声明私有属性或方法来告诉别人这个属性或方法是私有的，别人看到后就不会强制去调用，以免出现意料之外的结果。

9.3.1.4 实例属性与类属性重名

当实例属性和类属性重名时，通过类调用的默认是类属性，通过对象调用的默认是实例属性，相当于在对象中实例属性会覆盖类的属性。

```
# 定义一个类
class Students():
    # 类的属性：用变量来表示
    type = "学生"

    def __init__(self,name,age,weight,type):
        # 实例的属性
        self.Na me = name
        self.Age = age
        #私有的属性
        self.__weight=weight
        self.type=type

# 实例化一个对象
stu1 = Students("张三", 20, 80,"小学生")

# 重新传值覆盖类的属性的值
print(stu1.type)
print(Students.type)

#例如：
class Test(object):
  val = 100
  def __init__(self):
  self.val = 200

test = Test()
test.val #输出200
Test.val #输出100
```

9.3.2 方法的不同类型

9.3.2.1 类方法（classmethod装饰器）

使用classmethod装饰器来装饰一个方法，可以把这个方法变成类的方法。类的方法的特点是：

1. 类方法的第一个参数是类对象参数cls（cls代表类本身），在调用的时候不需要对cls传实参；
2. 如果一个方法被标示为类方法，则该方法可被类调用，也可以被类的对象调用。

```
# 定义一个类
class Students():
    # 类的属性：用变量来表示
    type = "学生"
    #声明一个类方法
    @classmethod
```

```
    def get_age(cls):
        return cls.type
# 实例化一个对象
stu1 = Students()

#通过类调用类的方法
print(Students.get_age())
#通过对象调用类的方法
print(stu1.get_age())
```

9.3.2.2 实例方法（普通方法）

实例的方法只能通过对象调用，这是我们写得最多的方法。

```
# 定义一个类
class Students():
    # 类的属性：用变量来表示
    type = "学生"

    def __init__(self,name,age,weight,type):
        # 实例的属性
        self.Name = name
        self.Age = age
        #私有的属性
        self.__weight=weight
        self.type=type
    # 方法：用函数来表示，但注意方法和函数两个概念有区别
    def study(self):
        return "学习软件测试技术！"

    def get_name(self):
        return self.Name

# 实例化一个对象
stu1 = Students("张三", 20, 80,"小学生")

#通过对象调用实例方法
print(stu1.get_name())
```

9.3.2.3 静态方法（staticmethod装饰器）

使用 staticmethod装饰器来装饰一个方法，可以把它变成一个静态方法，静态方法不需要传形参self和cls（就像一个在类里面的普通函数）。静态方法可以用类来调用，也可以用对象来调用。

```
# 定义一个类
class Students():
    # 类的属性：用变量来表示
    type = "学生"

    #声明一个静态方法
    @staticmethod
    def get_type():
        return Students.type

# 实例化一个对象
```

```
stu1 = Students()

#通过类调用静态方法
print(Students.get_type())
#通过对象调用静态方法
print(stu1.get_type())
```

9.3.2.4 私有方法（方法前面加两个下划线）

属性和方法分为public（公共的）和private（私有的），默认是公共的。如果要声明私有的属性或方法，要在属性名或方法名前加__(两个下划线)，私有的方法不能通过类或者对象直接调用。

私有的属性或方法的调用方式：

1. 通过定义公有的方法来间接地调用私有的属性或者方法；
2. 通过_类名__属性/方法来调用，在python中没有绝对的私有，所谓的私有实际上是在名称前加了_类名而已。

```
# 定义一个类
class Students():
    # 类的属性：用变量来表示
    type = "学生"

    #声明一个私有方法
    def __get_type_2(self):
        return self.type

    #声明一个公有方法来间接地调用私有方法__get_type_2
    def get_type_2(self):
        return self.__get_type_2()

# 实例化一个对象
stu1 = Students()
# 通过声明的公有方法来间接地实现私有方法的调用
print(stu1.get_type_2())
# 通过在属性前加_类名来实现对私有方法的调用
print(stu1._Students__get_type_2())
```

9.4 类的继承

9.4.1 继承的概念

- 继承是类与类之间的一种关系，子类继承父类。
    - 子类：需要继承的类
    - 父类：被继承的类
- 继承的特点：
    1. 父类拥有的属性和方法，则子类一定有（私有的属性和方法可以间接调用）；
    2. 父类拥有的属性和方法，子类可以修改；
    3. 父类没有的属性和方法，子类可以新增。
- 核心：当一个子类继承父类时，必然会先创建父类对象，然后再创建子类对象，这样子类对象中才拥有所有父类和子类的属性和方法。

9.4.2 继承的写法

在声明一个类的时候，如果这个类继承了父类，在类名后加括号来指定父类的名称。

```
# 定义一个父类
class People:
    type="高等生物"

    def get_type(self):
        return self.type
#定义一个子类，集成父类People
class Students(People):
    pass
```

9.4.3 子类继承父类的属性和方法

```
# 定义一个父类
class People:
    type="高等生物"

    def get_type(self):
        return self.type
#定义一个子类，集成父类People
class Students(People):
    pass

#创建Students类的对象
stu1=Students()
#子类对象可以调用父类的属性和方法，因为继承过来了
print(stu1.type)
print(stu1.get_type())
```

9.4.4 子类新增父类没有的属性和方法

```
# 定义一个父类
class People:
    type="高等生物"

    def get_type(self):
        return self.type
#定义一个子类，集成父类People
class Students(People):
    #子类的属性
    type2 ="小学生"
    #子类的方法
    def get_type2(self):
        return self.type2

#创建Students类的对象
stu1=Students()
#子类对象可以调用父类的属性和方法，因为继承过来了
print(stu1.type)
```

```
print(stu1.get_type())
```

9.4.5 子类重写父类的属性和方法

```
#把适用于自己的父类方法写成自己的方法
# 定义一个父类
class People:
    type="高等生物"

    def get_type(self):
        return self.type
#定义一个子类，集成父类People
class Students(People):
    type ="小学生"
    def get_type(self):
        return self.type

#创建Students类的对象
stu1=Students()
#子类对象可以调用父类的属性和方法，因为继承过来了
print(stu1.type)
print(stu1.get_type())
```

9.4.3 super对象的使用

super对象的作用是调用父类的方法，当在子类中调用父类的同名的方法或属性时，需要通过super对象来调用。不同名的方法可以直接通过self来调用。

在python3.x中通过super对象调用父类方法有两种写法：

1. super().父类的方法(参数)
2. super(子类名称,self).父类的方法(参数)

Python继承的3种典型场景：

1. 如果子类没有显式声明__init__()方法，不管父类有没有显式地声明__init__()方法，都不需要在子类中手动地调用父类的__init__()方法，系统会实现自动调用；
2. 如果子类显式声明了__init__()方法，但父类没有显式声明__init__()方法，同样不需要在子类中手动地调用父类的__init__()方法，系统会实现自动调用；
3. 如果子类和父类都显式声明__init__()方法，则必须在子类的__init__()方法中用super对象手动地调用父类的**init**()方法创建父类的对象（调用父类的__init__()方法时不需要传入self参数）。

举例：

```
#例子1：父类显式地声明了__init__方法，而子类没有显式地声明了__init__方法
#定义一个父类
class People:
    type="高等生物"
    #构造方法
    def __init__(self, name, age):
        # self表示对象本身
        # 定义属性
        self.name = name
```

```
        self.age = age
    def get_type(self):
        return self.type
#定义一个子类，集成父类People
class Students(People):
    pass

#创建Students类的对象，需要传入父类构造方法需要的参数，当子类没有显式地声明__init__方法的时候，在创建子类对象之前，系统会自动调用父类的__init__方法来完成父类对象的创建
stu1=Students("张三",30)
print(stu1.name)
print(stu1.age)

#例子2：父类没有显式地声明__init__方法，而子类显式地声明了__init__方法
# 定义一个父类
class People:
    type="高等生物"
    #构造方法
    # def __init__(self, name, age):
    #     # self表示对象本身
    #     # 定义属性
    #     self.name = name
    #     self.age = age
    def get_type(self):
        return self.type
#定义一个子类，集成父类People
class Students(People):
    def __init__(self, sex, school):
        self.sex=sex
        self.school=school

#创建Students类的对象，因为父类没有显式地声明__init__方法，当子类显式地声明__init__方法的时候，在创建子类对象之前，系统会自动调用父类的默认的不带参的__init__方法来完成父类对象的创建
stu1=Students("男","成都职业技术学院")
print(stu1.sex)
print(stu1.school)

#例子3：父类和子类都显式地声明了__init__方法
# 定义一个父类
class People:
    type="高等生物"
    #构造方法
    def __init__(self, name, age):
        # self表示对象本身
        # 定义属性
        self.name = name
        self.age = age
    def get_type(self):
        return self.type
#定义一个子类，集成父类People
class Students(People):
    def __init__(self, sex, school, name, age):
        #当父类和子类中都显式地声明了__init__方法时，需要在子类中用super对象来调用父类的__init__ 方法，以完成父类对象的创建，这样才能实现子类继承父类。此时子类的__init__方法的形参包括父类 __init__方法的形参
        super().__init__(name, age)
        self.sex=sex
        self.school=school
```

```
#例子4：
#子类重新了父类的方法，还需要调用父类的方法
#定义一个父类
class People:
    type="高等生物"

    def get_type(self,):
        return self.type
#定义一个子类，集成父类People
class Students(People):
    type1 ="小学生"
    def get_type(self):
        return self.type1
    def super_get_type(self): # 重新父类的方法后用作调用父类的方法
        return super().get_type()
#创建Students类的对象
stu1=Students()
#子类重新父类的方法后再调用父类的方法
print(stu1.super_get_type())
```

十、Python中的常用模块

10.1 模块的导入

- 库、包、模块的关系讲解
- 1.模块：就是一个.py的文件

  2.包：就是包含了模块，还有一个名叫__init__.py的文件才叫包，否则就是文件夹，__init__.py可以是空文件也可以有代码。包里可以有包也可以有模块，可以多级结构混杂。

  3.库：具有相关功能的包和模块的合集。（web自动化用的是selenium第三方库，接口自动化用的是requests第三方库）
- 一个模块只会被导入一次，不管你执行了多少次import。
- 导入方式：

  1、import 模块　　import后面一般能跟模块，不可以跟类，在调用其函数/类/变量的时候，用模块名.函数/类/变量的形式 （比如，import random，random.函数、类、变量）

2. import...as...　　导入并重命名
3. from...import...　　from后面可以跟包名，也可以跟“包名.模块名”，import后面跟模块名或者类名，推荐使用该方式导入
4. from...import...as 在模块名或者包名过长时，为了后面的程序编写，可以给其取别名

10.2 随机模块-random

- random模块，用于生成伪随机数，之所以称之为伪随机数，是因为真正意义上的随机数（或者随机事件）在某次产生过程中是按照实验过程中表现的分布概率随机产生的，其结果是不可预测的，是不可见的。
- 举例：

```
# 打印随机数字，猜数字游戏
import random
num = random.randrange(1,100) #随机一个1到100的正整数
print(num)
shuru = int(input("输入一个数字："))
if shuru == num:
    print("猜对了！")
else:
    print("猜错了！")
```

10.3 字符串模块-string

- 这是一个内置模块，我们必须在使用任何常量和类之前将其导入
- 举例：

```
# 随机模块和字符串模块结合使用
import random
import string
# 随机取六位字母，判断输入和随机的是否相等
letter1 = random.sample(string.ascii_letters,6)
str1 = "".join(letter1)
print(str1)
str2 = input("请输入六位字母：")
if str2 == str1:
    print("输入正确")
else:
    print("输入错误")
```

10.4 os模块

10.4.1 os模块介绍

- os模块提供了多数操作系统的功能接口函数。当os模块被导入后，它会自适应于不同的操作系统平台，根据不同的平台进行相应的操作，在python编程时，经常和文件、目录打交道，这时就需要使用os模块。

10.4.2 os模块的常用函数

- os.name：操作系统的类型，nt表示windows，posix表示linux
- os.uname()：返回操作系统的版本信息，包括操作系统的名称、内核版本、硬件信息等，注意这个方法只在linux环境上支持，返回结果如：('Linux', 'instance-01k7rh09', '3.10.0-957.27.2.el7.x86_64', '#1 SMP Mon Jul 29 17:46:05 UTC 2019', 'x86_64')
- os.getcwd()：返回当前工作的目录，相当于linux上的pwd命令
- os.getenv(x)：返回环境变量x的值；

```
print(os.getenv("PATH"))
```

- os.path.isdir()：判断文件夹是否存在，返回布尔值

```
#判断C盘下是否存在一个名叫Python的文件夹？
os.path.isdir() #

os.path.isfile() #判断文件是否存在
os.path.exists() #既可以判断文件也可以判断文件夹
os.lisdir(路径) #获取盘下所有文件和目录名称
os.name #打印环境的操作系统
os.getpid() #打印运行id
os.getcwd() #求当前文件的路径
os.path.join() #路径拼接
```

- os.path.isfile()：判断文件是否存在，返回布尔
- os.path.exists()：即可判断文件夹也可以判断文件，返回布尔
- os.path.join()：实现两个路径的拼接
- os.mkdir()：创建目录，相当于linux的mkdir命令
- os.rmdir() 删除目录，只能删除空目录
- os.listdir() 查询路径中的所有目录，括号里传路径

- 作业：
    1. 判断C盘下有没有一个叫testdir的目录，如果没有就创建，
    2. C盘下的test_photo目录下文件下有多张图片，实现随机选择一张图片，返回图片的路径

10.5 time 模块

10.5.1 time模块说明

- time模块提供各种与时间相关的函数，time模块是Python自带的模块。
- Python中的四种格式的时间：
    1. 纪元时间epoch格式时间：以秒为单位进行换算所得到的的时间，指的是从计算机时间元年1970.1.1到现在所创建的时间之间的数转换成秒单位的浮点数，比如1630120707.7790058；
    2. 时间元组struct-time时间：比如time.struct_time(tm_year=2021, tm_mon=8, tm_mday=28, tm_hour=3, tm_min=18, tm_sec=58, tm_wday=5, tm_yday=240, tm_isdst=0)；

索引（Index）	属性（Attribute）	值（Values）
0	tm_year（年）	比如2011
1	tm_mon（月）	1 - 12
2	tm_mday（日）	1 - 31
3	tm_hour（时）	0 - 23
4	tm_min（分）	0 - 59
5	tm_sec（秒）	0 - 61
6	tm_wday（weekday）	0 - 6（0表示周日）
7	tm_yday（一年中的第几天）	1 - 366
8	tm_isdst（是否是夏令时）	默认为-1

3. python定义的英文格式显示时间：比如Wed Oct 16 09:49:02 2019；
4. 自定义格式时间：按自己需要的格式来表示，比如2021/8/20 11:20:20,。

10.5.2 time模块中的常用函数

- 获取系统的当前时间：
  - time.time()：返回当前时间，显示epoch格式时间
  - time.localtime()：返回当前时间，显示struct-time时间
  - time.asctime()：返回当前时间，显式英语格式
  - time.gmtime()：返回当前的格林威治时间
- 时间格式的转换：
  - time.gmtime()：传入一个epoch时间，转成时间元组格式，如果不传入参数则表示转换当前时间；
  - time.mktime()：作用和time.gmtime()相反,以元组的形式传值，必须为9个值；
  - time.strftime()：传入一个自定义的格式和struct-time格式的时间，把struct-time格式的时间转成自定义的格式；
    - %y 两位数的年份表示（00-99）
      %Y 四位数的年份表示（000-9999）
      %m 月份（01-12）
      %d 月内中的一天（0-31）
      %H 24小时制小时数（0-23）
      %I 12小时制小时数（0-12）
      %M 分钟数（0-59）
      %S 秒（00-59）
      %a 本地简化星期名称
      %A 本地完整星期名称
      %b 本地简化的月份名称
      %B 本地完整的月份名称
      %c 本地相应的日期表示和时间表示（e.g Thu Dec 10 09:54:27 2020）
      %j 年内的一天（001-366）
      %p 本地A.M.或P.M.的等价符
      %U 一年中的星期数（00-53）星期天为星期的开始
      %w 星期（0-6），星期天为星期的开始

%W 一年中的星期数（00-53）星期一为星期的开始
%x 本地相应的日期表示(e.g 12/10/20)
%X 本地相应的时间表示(e.g 09:58:15)
%Z 当前时区的名称(e.g 中国标准时间)
%% %号本身

- time.sleep(x)：表示程序执行到这一行就休眠x秒，经常用在自动化代码中实现等待的效果。

10.6 Excel的操作模块-openpyxl

10.6.1 openpyxl模块的介绍与安装

- openpyxl模块是专门用来读写excel文件的模块。
- 前面讲的模块都属于Python内置的模块，使用前不需要安装，直接导入即可；这里的openpyxl属于第三方模块，在使用前必须先安装。
- 安装方式有：

1. pip install openpyxl
2. 在pycharm中安装

在使用openpyxl模块之前，需要先搞清楚以下两个概念：

- workbook：工作簿，即整个excel文件
- worksheet：工作表，一个excel文件中可以有多个工作表

10.6.2 新建Excel文件进行读写操作

https://openpyxl.readthedocs.io/en/stable/

举例：

```
#导入workbook类
from openpyxl import workbook
#实例化workbook对象，相当于新建了一个excel文件
wb = workbook()
#获取活动的工作表
ws = wb.active
#指定单元格写入内容
ws["A6"] = "花儿乐队"
#列表中写入一行数据
list1 = [1,3,45,6]
ws.append(list1)
#列表中的嵌套列表中的数据导入
data = [[11,22,33],[44,55,66],[77,88]]
for i in data:
    ws.append(i)
#新建一个名叫“名单”的sheet页，并且把它放到第一个位置
ws_2 = wb.create_sheet("名单",0)
#操作名单的sheet页面
#保存文件
wb.save(r"c:\test01.xlsx")
```

10.6.3 打开已有的Excel文件进行读写操作

举例：

```
### 找指定单元格的数据
import openpyxl
# 打开文件
wb = openpyxl.load_workbook(r"C:\Users\lenovo\Desktop\33期\test01.xlsx")
# 选择sheet页
ws = wb["Sheet"] # 新方法，和下面老方法一样
# ws=wb.get_sheet_by_name("Sheet") # 老方法，逐步要淘汰了
#获取单元格对象
cell = ws["A3"]
# 打印单个单元格的列，行，值
print(cell.column,cell.row,cell.value)
# 打印单元格的坐标和值
print(cell.coordinate,cell.value)
#获取excel列的数量
print(ws.max_column)
#获取行列的数量
print(ws.max_row)
```

10.6.4 打开txt文件，读取内容写入excel（批量写入）

eval()函数使用实例：

字符串转换成其他数据类型："[]"转换出来列表，如果"()"转换出来为元组，如果为“{}”转换出来为字典

```
#打开文件
with open(r"d:\list1.txt") as fi:
    #读取文件内容
    data = fi.read()
    #将字符串转换为列表
    list01 = eval(data)
    for i in list01:
        get_sheet.append(i)
    wb.save(r"d:\test02.xlsx")
```

10.6.5 从文件excel中读取数据

```
### 找指定单元格的数据
import openpyxl
# 打开文件
dq = openpyxl.load_workbook(r"d:\test02.xlsx")
# 选择sheet页
get_sheet = dq["信息"]
li = get_sheet["a3"]
# 打印单个单元格的列，行，值
print(li.column,li.row,li.value)
# 打印单元格的坐标和值
print(li.coordinate,li.value)
```

```
### 批量读取excel中的数据
get_sheet = dq["信息"]
#第一个for循环读取的为一行的数据
for i in get_sheet:
    # 第二个for循环读取的为单个
    for li1 in i:
        print(li1.coordinate,li1.value)
```

```
# 读取excel中的文件并以列表格式展示
import openpyxl
# 打开文件
dq = openpyxl.load_workbook(r"d:\test02.xlsx")

get_sheet = dq["信息"]
# 取出来的数据用[[],[],[]]格式展示
all_list =[]
#第一个for循环读取的为一行的数据
for i in get_sheet:
    list1 = []
    # 第二个for循环读取的为单个
    for li1 in i:
        list1.append(li1.value)
    all_list.append(list1)
# 打印不要表头数据
print(all_list[1::])
```

10.7 MySQL数据库的操作-pymysql

- 在做测试过程中，我们可以把测试数据创建在数据库，也可以读取数据库中的数据进行断言操作
- python操作数据库之前需要安装数据库驱动
- 安装方式：

  1、pip install pymysql

  2、在pycharm中安装

10.7.1 连接并操作数据库

```
import pymysql
# 连接数据库
my_connect = pymysql.connect(host = '',user = '',password = '',database =
''.port = ,charset = 'utf8')
# 建立游标：目的是为了缓存数据方便操作(读取数据，遍历表中的所有数据，以便查询)
du_shuju = my_connect.cursor()
#执行sql语句
du_shuju.execute("select * from xxxxx;")
#获取数据
huoqu_data = du_shuju.fetchall()
print(huoqu_data)
#修改表中的数据
du_shuju.execute("update 表名 set 字段名="xxx" where 字段名="yyy";")
#提交数据,修改的数据要进行提交才能修改数据库中的数据，否则修改的只是游标缓存里的数据
```

```
my_connect.commit()
```

十一、io流处理

11.1 I/O流操作

- 文件I/O流指输入输出操作（input、output）
- 创建文件(必须先用Python内置的open()函数打开一个文件，创建一个file对象，才能调用相关的方法进行读写)
- ```
  # mode为“w”时，如果有这个文件，就直接打开，如果没有则新建一个再打开
  file_1 = open('d:xiao.txt',mode='w')
  # 操作完成后关闭文件
  file_1.close()
  ```
- 1、open函数
- 2、打开文件的模式（mode的值，默认为r）
  - r: 以只读方式打开文件
  - r+: 打开一个文件用于读写（会在已有的内容前且会覆盖原有的前面的内容添加数据）
  - w: 打开一个文件只用于写入（不存在则创建，存在则完全覆盖内容）
  - w+: 打开一个文件用于读写（覆盖已有的数据）
  - a：打开一个文件用于追加（不存在则创建，存在就在文件最后追加内容）
  - a+：打开一个文件用于读写（追加数据）
- 3、查询某个盘下面的所有文件夹

  ```
  # os.listdir()
  for i in os.listdir(r"d:"):
      print(i)
  ```
- 4、举例

  ```
  # 读取文件内容，写入另一个文件
  # 打开文件
  f1 = open(r"d:\test.txt")
  # 读取内容
  data = f1.read()
  #创建文件，mode=a或w，如果文件不存在就会先创建再写入a表示追加，w表示覆盖
  f2 = open(r"d:\test2.txt",mode="a")
  #写入内容
  f2.write(data)
  # 关闭文件
  f1.close
  f2.close
  ```
- 6.with open结构

  ```
  # 无论中间代码执行是否错误，最后都会关闭文件（with语句在打开文件后会自动调用finally并关闭文件）
  with open(r"d:\test.txt",mode='r') as f1:
      print(f1.read())
  ```
  ```

- 7.open和with open的区别

  1). open（）函数：这样直接打开文件，如果出现异常，如读取过程中文件不存在或异常，则直接出现错误，close方法无法执行，导致文件无法关闭。

  2). 用with语句的好处，就是到达语句末尾时，即便出现异常也会自动关闭文件

十二、装饰器

12.1 定义

装饰器：作用是装饰一个函数或者方法，它可以让其他函数在不需要做任何代码变动的前提下增加额外的功能。装饰器的本质其实是一个函数，但它的参数是函数对象。简单说：他们是修改其他函数（被装饰函数）的功能的函数

调用函数时，如果不加括号（传参），返回的是函数对象，如果加括号传参，才是真正地调用了函数。

装饰器的简单理解：

实际上就是为了给一个程序添加功能，但是该程序已经上线或者已被使用，那么就不能大批量的修改源码，这样不现实，因此就产生了装饰器。

注意点：

（1）不能修改被装饰的函数的源代码

（2）不能修改被装饰的函数的调用方式

12.2 不带参的装饰器和带参的装饰器：

- 1）不带参的装饰器：声明了一个两层嵌套的函数，函数中嵌套了函数，外层函数的返回值是里层函数的对象；
- 2）带参的装饰器:相当于三层嵌套函数，最外层的用来接收参数；最外层函数的返回值是中间函数的对象，中间函数的返回值是最里层函数的对象

12.3 装饰器的调用

调用装饰器有两种方式：

- 1）把被装饰函数的对象作为参数传入到装饰器；
- 2）在被装饰函数前面用语法糖符号@来调用装饰器

举例：

```
#不带参的装饰器：声明了一个两层的函数，函数中嵌套了函数，外层函数的返回值是里层函数的对象
def daka(f):
    def function1():
        print("早上9点打卡")
        f()
        print("晚上9点打卡")
    return function1
#声明两个被装饰的函数
def function3():
    print("小明高高兴兴地去上班")
def function4():
    print("小王去上班")

#调用装饰器有两种方式：1）把被装饰函数的对象作为参数传入到装饰器；2）在被装饰函数前面用语法糖符号@来调用装饰器
```

```
#调用daka函数，传入function3函数对象
daka(function3)()
daka(function4)()

#在被装饰函数前用@装饰器的名称来调用装饰器，实现被装饰函数的功能扩展
@daka
def function5():
    print("小李去上班")
#调用被装饰过的函数
function5()

#带参的装饰器:相当于3层函数，最外层的用来接收参数；最外层函数的返回值是中间函数的对象，中间函数的返回值是最里层函数的对象
def daka1(num1,num2):
    def daka(f):
        def function1():
            print("早上%d点打卡"%(num1))
            f(num1,num2)
            print("晚上9点打卡")
        return function1
    return daka

#在调用装饰器装饰时传入参数
@daka1(8,9)
def function6(num1,num2):
    print(num1)
    print("小小小来%d点来上班"%(num2))

#调用function6
function6()
```

十三、异常处理

- 异常即是一个事件，该事件会在程序执行过程中发生错误，影响了程序的正常执行
- 一般情况下，在python无法正常处理程序时就会发生一个异常，即报出错误
- 当python脚本发生异常时我们需要捕获并处理它，否则程序会终止执行
- 通过使用异常处理语句能让清晰的错误信息更能帮助你快速修复问题

13.1 异常处理语句 try...excpet...finally(常用于释放资源)

- try里面一般放你觉得可能会出错的代码
- 1、格式一
- raise 自动引发异常（常用），在web自动化时，可以用来获取日志，将报错信息写入日志中，方便查看问题。

```
input_str = input('请输入一个数字：')
try:
    print('输入内容是%d'%input_str)
except TypeError as e:#捕捉异常（Python提供不同异常类型接受不同的异常错误）
# 自动引发（抛出）异常--raise Exception('messages') 可以自定义报错信息
    raise TypeError('请输入数据类型的正确数字！')
```

- 2、格式二（常用）
- try：..... finally：..... :先运行try，然后再运行finally，不管try里面代码运行是否失败

```
try:
    print("进入try")
    file = open(r'd:\test.txt',mode='r')
    print("已经打开了文件")
finally:
    file.close
    print('结束')
```

- 3、格式三（不常用）:不管try执行是否正常，都会执行finally，只有异常才会执行except

```
try:
    print("进入try")
    file = open(r'd:\test.txt',mode='r')
    print("已经打开了文件")
except Exceptipon as e:
    print('进入except')
    print('报错信息：',e)
finally:
    print('结束')
```

- 4、格式四（不常用）： try:.....except:当try运行正常不执行except，当try运行不正常，运行except。

```
try:
    print("进入try")
    file = open(r'd:\test7.txt',mode='r')
    print(file.read())
    print("已经打开了文件")
except Exception as e:
    print('进入except')
    print('报错信息：',e)
```

- 5、格式五（不常用）
- try: ..... except: .....else :当try代码正常，则不执行except的代码，但是要执行else的代码；反之要执行except，而不执行else

```
try:
    print("进入try")
    file = open(r'd:\test08.txt',mode='r')
    print("已经打开了文件")
except Exception as e:
    print('进入except')
    print('报错信息：',e)
else:
    print("其它")
finally:
    print("最终的")

#捕获多个异常举例
try:
    1/0
```

```
except IndexError:
    print("异常1")
except ZeroDivisionError:
    print("除数不能为0")
else:
    print("没有异常")
finally:
    print("有没有异常都要执行")
```

异常处理语法小结：

1、except语句不是必须的，finally语句也不是必须的，但是二者必须要有一个，否则try就没有意义了。

2、except语句可以有多个，Python会按except语句的顺序依次匹配你指定的异常，如果异常已经处理就不会再进入后面的except语句。

3、Exception是常规错误的基类（足够我们使用了）；BaseException是所有异常的基类（Exception继承了它）。

4、如果有需要，切记要使用finally来释放资源，请不要忘记在处理异常后做清理工作或者回滚操作

```
try:
    # 可能发生异常的代码
except 异常类型1 as 变量名:#捕捉异常
    print(变量名) # 变量名存储的是具体的错误信息
except 异常类型2 as 变量名:
    print(变量名) # 变量名存储的是具体的错误信息
except Exception as 变量名:
    print(变量名) # 变量名存储的是具体的错误信息
except:
    #自动引发（抛出）异常--raise Exception('messages') 可以自定义报错信息
    raise Exception（msg）
else:
    print('如果以上代码没有发生异常以及异常处理工作就执行这里的代码')
    print('一般情况下else中的代码用来下结论')
finally:
    print('不管代码是否有异常都会执行')
    print('一般情况下用于这个函数中资源的回收，比如关闭浏览器、关闭数据库')
```