**1.1 程序设计基本方法**

**1. 计算机：根据指令操作数据的设备**

* 功能性：对数据的操作，表现为数据计算、输入输出处理和结果储存等
* 可编程性：根据一系列指令自动地、可预测地、准确地完成操作者等意图
* 计算机等发展参照摩尔定律，表现为指数方式

**2. 程序设计/编程：是计算机可编程性的体现，是深度应用计算机的主要手段**

**3. 程序设计语言/编程语言：一种用于交互（交流）的人造语言，是程序设计的具体实现方式**

* C语言诞生于1972年，是第一个被广泛使用的编程语言
* Python语言诞生于1990年，是当前最流行最好用的编程语言

**4. 编程语言的执行方式（编译与解释）**

* 源代码：采用某种编程语言编写的计算机程序，人类可读（如：result = 2 + 3）
* 目标代码：计算机可直接执行，非专家一般不可读（如：11010010 00111011）

|  |  |
| --- | --- |
|  |  |

（1）编译：将源代码一次性转换成目标代码的过程（一次性翻译，之后不再需要源代码，类似英文翻译）

编译器(compiler)：执行变异过程的程序

（2）解释：将源代码逐条转换成目标代码同时逐条运行的过程（每次程序运行时随随翻译随执行，类似于同声传译）

解释器(interpreter)：执行解释过程的程序

**5. 静态语言和脚本语言**

根据执行方式不同，编程语言分为两类

（1）静态语言：使用编译执行的编程语言（如C/C++语言、Java语言）

编译器一次性生成目标代码，优化更充分，程序运行速度更快

（2）脚本语言：使用解释执行的编程语言（如Python语言、JavaScript语言、PHP语言）

执行程序时需要源代码，维护更灵活且可跨多个操作平台运行

**6. 程序的基本编写方法（IPO）**

* -I：Input输入，程序的输入
* -P：Process处理，程序的主要逻辑
* -O：Output输出，程序的输出

（1）输入：一个程序的开始（文件输入、网络输入、控制台输入、交户界面输入、内部参数输入等）

（2）处理：程序对输入数据进行计算产生输出结果的过程（处理方法统称算法，它是程序最重要的部分）

（3）输出：程序展示运算结果的方式（控制台输出、图形输出、文件输出、网络输出、操作系统内部变量输出等）

**7. 问题的计算部分**

一个待解决问题中，可以用程序辅助完成的部分

* 计算机只能解决计算问题，即问题的计算部分
* 一个问题可能有多种角度理解，产生不同的计算部分
* 问题的计算部分一般都有输入、处理和输出过程

**8. 编程解决问题的步骤**

1. 分析问题：分析问题的计算部分，想清楚
2. 划分边界：划分问题的功能边界，规划IPO
3. 设计算法：设计问题的求解算法，关注算法
4. 编写程序：编写问题的计算程序，编程序
5. 调试测试：调试程序使正确运行，运行调试
6. 升级维护：适应问题的升级维护，更新完善

**9. 求解问题的精简步骤**

1. 确定IPO：明确计算部分及功能边界
2. 编写程序：将计算求解的设计变成现实
3. 调试程序：确保程序按照正确逻辑正确运行

**1.2 Python开发环境配置**

**1. Python的两种编程方式（交互式和文件式）**

* 交互式：对每个输入语句即使运行结果，适合语法练习
* 文件式：批量执行一组语句并运行结果，编程的主要方式

**2. 实例**

（1）根据半径r计算园面积

|  |  |
| --- | --- |
| 交互式 | 文件式 |
| >>> r = 25  >>> area = 3.1415 \* r \* r  >>> print(area)  1963.4375000000002  >>> print("{:.2f}".format(area))  1963.44 | r = 25  area = 3.1415 \* r \* r  print(area)  print("{:.2f}".format(area)) |

（2）绘制多个同切圆

|  |  |
| --- | --- |
| 交互式 | 文件式 |
| >>> import turtle  >>> turtle.pensize(1)  >>> turtle.circle(10)  >>> turtle.circle(40)  >>> turtle.circle(80)  >>> turtle.circle(160) | import turtle  turtle.pensize(1)  turtle.circle(10)  turtle.circle(40)  turtle.circle(80)  turtle.circle(160) |

（3）绘制一个五角星

|  |  |
| --- | --- |
| 交互式 | 文件式 |
| >>> from turtle import \*  >>> color('red', 'red')  >>> begin\_fill()  >>> for i in range(5):  fd(200)  rt(144)  >>> end\_fill() | from turtle import \*  color('red', 'red')  begin\_fill()  for i in range(5):  fd(200)  rt(144)  end\_fill()  done() |

**1.3 实例1:温度转换**

**0. 温度体系**

* 摄氏度：以1标准大气压下水的结冰电为0度，沸点为100度，将温度进行等分刻画
* 华氏度：以1标准大气压下水的结冰电为32度，沸点为212度，将温度进行等分刻画

**1. 该问题中计算部分的理解和确定**

* 理解1：直接将温度值进行转换
* 理解2：将温度信息发布的声音或图像形式进行理解和转换
* 理解3：监控温度信息发布渠道，实时获取并转换温度值

**2. 分析问题**

（1）采用理解1：直接将温度值进行转换

* 温度数值需要标明温度体系，即摄氏度或华氏度
* 转换后也需要给出温度体系

（2）划分边界

* -输入：带华氏或摄氏标志的温度值
* -处理：根据温度标志选择适当的温度转换算法
* -输出：带摄氏或华氏标志的温度值

（3）输入输出格式设计

标识放在温度最后，F表示华氏度，C表示摄氏度（82F表示华氏82度，28C表示摄氏28度）

（4）设计算法

C = (F - 32) / 1.8

F = C \* 1.8 + 32

**3. 编程**

|  |
| --- |
| # TempConvert.py  TempStr = input('请输入带有符号的温度值：')  if TempStr[-1] in ['F', 'f']:  C = (eval(TempStr[0:-1]) - 32) / 1.8  print("转换后的温度是{:.2f}C".format(C))  elif TempStr[-1] in ['C', 'c']:  F = 1.8 \* eval(TempStr[0:-1]) + 32  print("转换后的温度是{:.2f}F".format(F))  else:  print("输入格式错误") |

**1.4 Python语法元素分析**

|  |  |
| --- | --- |
| # TempConvert.py  TempStr = input("请输入带有符号的温度值：")  if TempStr[-1] in ['F', 'f']:  C = (eval(TempStr[0:-1]) - 32) / 1.8  print("转换后的温度是{:.2f}C".format(C))  elif TempStr[-1] in ['C', 'c']:  F = 1.8 \* eval(TempStr[0:-1]) + 32  print("转换后的温度是{:.2f}F".format(F))  else:  print("输入格式错误") |  |

**1. 程序的格式框架**

（1）代码高亮：编程的色彩辅助体系，不是语法要求

（2）缩进：一行代码开始前的空白区域，表达程序的格式框架（单层缩进；双层缩进）

* 严格明确：缩进是语法的一部分，缩进不正确程序运行错误
* 所属关系：表达代码间包含和层次关系的唯一手段
* 长度一致：程序内一致即可，一般用4个空格或1个TAB

（3）注释：用于提高代码可读性的辅助性文字，不被程序执行

|  |  |  |
| --- | --- | --- |
| 单行注释 | 以#开头，其后内容为注释 | # 这里是单行注释 |
| 多行注释 | 以'''开头和结尾 | ''' 这是多行注释第一行  这是多行注释第二行''' |

**2. 命名与保留字**

（1）变量：程序中用于保存和表示数据的占位符号

* 变量采用标识符（名字）来表示，关联标识符的过程叫命名（TempStr是变量名字）
* 可以使等号（=）向变量赋值或修改值，=被称为赋值符号（TempStr = '82F' # 向变量TempStr赋值'82F'）

（2）关联标识符的过程

* 命名规则：大小写字母、数字、下划线和汉字等字符的组合（如：TempStr，Python\_Great，这是门Python好课）
* 注意事项：大小写敏感、首字符不能是数字、与保留字不同（Python和python是不同的变量，123Python不合法）

（3）保留字：被编程语言内部定义并保留使用的标识符

* Python语言有33个保留字/关键字（如：if，elif，else，in）
* 保留字是编程语言的基本单词，大小写敏感（if是保留字，If可当作变量）

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

**3.** **数据类型（字符串、整数、浮点数、列表···）**

（1）10,011,101该如何解释？

* 这是一个二进制数字或者十进制数字（作为二进制数字，10,011,101的值是十进制的157）
* 这是一段文本或者用逗号,分隔的3个数字（作为一段文本，都好是文本的一部分，一共包含10个字符）

（2）供计算机程序理解的数据形式

* 程序设计语言不允许存在语法歧义，需要定义数据的形式（如：需给10,011,101关联一种计算机可以理解的形式）
* 程序设计语言通过一定方式向计算机表达表达数据的形式（如："123"表示文本字符串123，123则表示数字123）

（3）10,011,101

* 整数类型：10011101
* 字符串类型："10,011,101"
* 列表类型：[10, 011, 101]

Ⅰ. 字符串：由0个或多个字符组成的有序字符序列

* 字符串由一对单引号或一对双引号表示（"请输入带有符号的温度值："或者"C"）
* 字符串是字符的有序序列，可对其中的字符进行索引（"请"是"请输入带有符号的温度值："的第0个字符）

①字符串的序号：正向递增序号和反向递减序号



②字符串的使用：使用[ ]获取字符串中一个或多个字符

* 索引：返回字符串中单个字符 <字符串>[M]（如："请输入带有符号的温度值："[0]或者TempStr[-1]）
* 切片：返回字符中一段字符子串 <字符串>[M:N]（如："请输入带有符号的温度值："[1:3]或者TempStr[0:-1]）

Ⅱ. 数字类型：整数和浮点数都是数字类型

* 整数：数学中的整数，包含整数和负数（如：32或者-89）
* 浮点数：数学中的实数，带有小数部分，包含整数和负数

Ⅲ. 列表类型：由0个或多个数据组成的有序序列

* 列表使用[ ]表示，采用逗号(,)分隔各元素（['F', 'f']表示两个元素'F'和'f'
* 使用保留字in判断一个元素是否在列表中（TempStr[-1] in ['C, 'c]判断前者是否与列表中某个元素相同）

**4.** **语句与函数**

（1）赋值语句：由赋值符号构成的一行代码

* 赋值语句用来给变量赋予新的数据值（C = (eval(TempStr[0:-1]-32)/1.8) # 右侧运算结果赋给变量C）
* 赋值语句右侧的数据类型同时作用于变量（TempStr = input(" ") # input()返回一个字符串，TempStr也是字符串）

（2）分支语句：由判断条件决定程序运行方向的语句

* 使用保留字if elif else构成条件判断的分支结构（if TempStr[-1] in ['F', 'f']: # 如果条件为True则执行冒号后语句）
* 每个保留字所在行最后存在一个冒号(:)，语法的一部分（冒号及后续缩进用来表示后续语句与条件的所属关系）

（3）函数：根据输入参数产生不同输出的功能过程

类似数学中的函数，y=f(x)（print("输入格式错误") # 打印输出“输入格式错误”）

函数采用<函数名>(<参数>)方式使用（eval(TempStr[0:-1]) # TempStr[0:-1]是参数）

**5. Python程序的输入输出**

（1）输入函数input()：从控制台获得用户输入的函数

* input()函数的使用格式：<变量> = input(<提示信息字符串>)
* 用户输入的信息以字符串类型保存在<变量>中（TempStr = input("请输入") # TempStr保存用户输入的信息）

（2）输出函数print()：以字符形式向控制台输出结果的函数

* Print()函数的基本使用格式：print(<拟输出字符串或字符串变量>)
* 字符串类型的一对引号仅在程序内部，输出无引号（print("输入格式错误") # 向控制台输出 输入格式错误）

（3）Print()函数的格式化：print("转换后的温度是{:.2f}C".format(C))

* { }表示槽，后续变量填充到槽中；{:.2f}表示将变量C填充到这个位置时取小数点后2位
* print("转换后的温度是{:.2f}C".format(C)),如果C的值是123.456789，则输出结果为：转化后的温度是123.45C

（4）评估函数eval()：去掉参数最外侧引号并执行余下语句的函数

* eval()函数的基本使用格式：eval(<字符串或字符串变量>)

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

**6. 温度转换代码分析**

略。

**n. 总结**

* 缩进、注释、命名、变量、保留字
* 数据类型、字符串、整数、浮点数、列表
* 赋值语句、分支语句、函数
* input()、print()、eval()、print()格式化

**1.n 第一章作业及学习资料**

**一、单选题**

1. Guido van Rossum正式对外发布Python版本的年份是：

|  |  |  |  |
| --- | --- | --- | --- |
| A. 1998 | B. 2002 | C. 2008 | D. 1991 |
| Python成功了，所以早年的开发历史也受到关注，以下是Guido自述的启动阶段时间表：  December, 1989 Implementation started  1990 Internal releases at CWI  February 20, 1991 0.9.0 (released to alt.sources)  February, 1991 0.9.1  鉴于Internal release不算对外发布，普遍认为Python语言诞生于1991年。 | | | |

2. 以下关于Python语言中“缩进”说法正确的是：

|  |
| --- |
| A. 缩进统一为4个空格 |
| B. 缩进是非强制的，仅为了提高代码可读性 |
| C. 缩进在程序中长度统一且强制使用 |
| D. 缩进可以用在任何语句之后，表示语句间的包含关系 |
| Python语言的缩进只要统计即可，不一定是4个空格（尽管这是惯例）。 |

3.以下不属于IPO模型的是：

|  |  |  |  |
| --- | --- | --- | --- |
| A. Program | B. Output | C. Process | D. Input |
| IPO：Input Process Output | | | |

4.字符串是一个字符序列，给字符串s，以下表示s从右侧向左第三个字符的是：

|  |  |  |  |
| --- | --- | --- | --- |
| A. s[-3] | B. s[0:-3] | C. s[3] | D. s[:-3] |
| 字符串有正向递增和反向递减两套序号体系 | | | |

5. 以下不是Python语言合法命名的是：

|  |  |  |  |
| --- | --- | --- | --- |
| A. 5MyGod | B. MyGod | C. MyGod5 | D. \_MyGod\_ |
| 合法命名的首字符不能是数字。 | | | |

6. 在Python中，用于获取用户输入的函数是：

|  |  |  |  |
| --- | --- | --- | --- |
| A. print() | B. input() | C. get() | D. eval() |
| get()不是Python内置函数，获得用户输入只有一种方式：input()。 | | | |

7. 下面不属于Python保留字的是：

|  |  |  |  |
| --- | --- | --- | --- |
| A. elif | B. def | C. type | D. import |
| type不是Python保留字，而是内置函数 type()。 | | | |

8. 以下不是Python数据类型的是：

|  |  |  |  |
| --- | --- | --- | --- |
| A. 字符串 | B. 列表 | C. 整数 | D. 实数 |
| 实数是数学中的概念，在Python中对应浮点数。 | | | |

9.哪个选项给出的保留字不直接用于表示分支结构？

|  |  |  |  |
| --- | --- | --- | --- |
| A. elif | B. else | C. in | D. if |
| if-elif-else是分支表达，in用来进行成员判断。 | | | |

10.

|  |  |  |  |
| --- | --- | --- | --- |
| A. {:.2} | B. {.2f} | C. {.2} | D. {:.2f} |
| :.2f 哪一个都不能少 | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 题号： | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |
| 答案： | D | C | A | A | A | B | C | D | C | D |

**二、程序题**

1. Hello World的条件输出

*获得用户输入的一个整数，参考该整数值，打印输出"Hello World"，要求：*

*如果输入值是0，直接输出"Hello World"*

*如果输入值大于0，以两个字符一行方式输出"Hello World"（空格也是字符）*

*如果输入值小于0，以垂直方式输出"Hello World"*

|  |  |
| --- | --- |
| 我的解答 | 参考答案 |
| num = eval(input())  Str = "Hello World"  Len = len("Hello World")  if num == int(num):  if num == 0:  print(Str)  elif num < 0:  for s in Str:  print(s)  else:  i = 0  while i < Len:  if i % 2 == 0:  print(Str[i], end = '')  else:  print(Str[i])  i += 1 |  |

2. 数值运算

*获得用户输入的一个字符串，格式如下：*

*M OP N*

*其中，M和N是任何数字，OP代表一种操作，表示为如下四种：+, -, \*, /（加减乘除）*

*根据OP，输出M OP N的运算结果，统一保存小数点后2位。*

*注意：M和OP、OP和N之间可以存在多个空格，不考虑输入错误情况。*

|  |  |
| --- | --- |
| 我的解答 | 参考答案 |
| temp = input()  temp2 = eval(temp)  print( "{:.2f}".format(temp2)) |  |

**2.1 深入理解Python语言**

**1. 计算机技术的演进**

|  |  |  |  |
| --- | --- | --- | --- |
| 1946-1981年（35年） | 计算机系统结构时代 | 计算能力问题 | 2008年，安卓操作系统诞生  ——“PC时代”向移动时代转换 |
| 1981-2008年（27年） | 网络和视窗时代 | 交互问题 |
| 2008-2016年（08年） | 复杂信息系统时代 | 数据问题 |
| 2017年—— | 人工智能时代 | 人类的问题 |

**2. 编程语言的多样初心**

（1）程序语言有哪些

|  |
| --- |
| Basic C, C++, C#, CSS, Fortran, Go, HTML, Java, JavaScript, Lisp, Lua, Matlab, Object C, Pascal,  PHP, PostScript, Python, Ruby, Scala, SQL, Swift, VBA, VB.NET, Verilog, VHDL, Visual Basic |

（2）不同编程语言的初心和适用对象

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编程语言 | 学习内容 | 语言本质 | 解决问题 | 适用对象 |
| **C** | 指针、内存、数据类型 | 理解计算机系统结构 | 性能 | 计算机类专业 |
| **Java** | 对象、跨平台、运行时 | 理解主客体关系 | 跨平台 | 软件类专业 |
| **C++** | 对象、多态、继承 | 理解主客体关系 | 大规模程序 | 计算机类专业 |
| **VB** | 对象、按钮、文本框 | 理解交互逻辑 | 桌面应用 | 不确定 |
| **Python** | 编程逻辑、第三方库 | 理解问题求解 | 各类问题 | 所有专业 |

**3. Python语言的特点**

|  |  |
| --- | --- |
| 语法简洁→10x效率 | 10x效率←生态高产 |
| C代码量的10% | >13万第三方库 |
| 强制可读性 | 快速增长的计算生态 |
| 较少的底层语法元素 | 避免重复造轮子 |
| 多种编程方式 | 开放共享 |
| 支持中文字符 | 跨操作系统平台 |

**4. 如何看待Python语言**

（1）人生苦短，我学Python

* C/C++：Python归Python，C归C
* Java：针对特定开发和岗位需求
* HTML/CSS/JS：不可替代的前端技术，全栈能力
* 其他语言：R/Go/Matlab等，特定领域

（2）Python是最高产的程序设计语言及……

* 掌握抽象并求解计算问题综合能力的语言
* 了解产业界解决复杂计算问题方法的语言
* 享受利用编程将创新变为实现乐趣的原因

（3）工具决定思维：关注工具变革的力量！

**5. “超级语言”的诞生**

（1）编程语言的分类

①机器语言

* 一种二进制语言，直接使用二进制代码表达指令
* 计算机硬件(CPU)可以直接执行，与具体CPU型号有关
* 完成2+3功能的机器语言（11010010 00111011）

②汇编语言

* 一种将二进制代码直接对应助记符的编程语言
* 汇编语言与CPU型号有关，程序不通用，需要汇编器转换
* 完成2+3功能的汇编语言（add 2, 3,result）

③高级语言

* 更接近自然语言，同时更容易描述计算问题
* 高级语言代码与具体CPU型号无关，编译后运行
* 完成2+3功能的高级语言（result = 2 + 3）

④超级语言

* 具有庞大计算生态，可以很容易利用已有代码功能
* 编程思维不再是刀耕火种，而是集成开发
* 完成2+3功能的高级语言（result = sum(2,3)）

*机器语言：代码直接执行，与CPU型号有关*

*汇编语言：有助记符，汇编器，与CPU型号有关*

*高级语言：接近自然语言，编译器，与CPU型号无关*

*超级语言：粘性整合已有程序，具备庞大计算生态*

**2.2 Python蟒蛇绘制**

**1. 问题分析：用程序绘制一条蟒蛇**

问题一：计算机绘图是什么原理？（一段程序为何能够产生窗体？为何能在窗体上绘制图形？）

问题二：Python蟒蛇绘制从哪里开始呢？（如何绘制一条线？如何绘制一个弧形？如何绘制一个蟒蛇？）

**2. 实例编写**

|  |  |
| --- | --- |
| #PythonDraw.py  import turtle  turtle.setup(650, 350, 200, 200)  turtle.penup()  turtle.fd(-250)  turtle.pendown()  turtle.pensize(25)  turtle.pencolor("purple")  turtle.seth(-40)  for i in range(4):  turtle.circle(40, 80)  turtle.circle(-40, 80)  turtle.circle(40, 80/2)  turtle.fd(40)  turtle.circle(16, 180)  turtle.fd(40 \* 2/3)  turtle.done() |  |

**3. 举一反三**

（1）Python语法元素理解

* Python蟒蛇绘制共17行代码，但很多行类似
* 清楚理解这17这行代码能够掌握Python基本绘图方法
* 参考框架结构、逐行分析、逐词理解

（2）程序参数的改变

* Python蟒蛇的颜色：黑色、白色、七彩色…
* Python蟒蛇的长度：1节、3节、10节…
* Python蟒蛇的方向：向左走、斜着走…

（3）计算问题的扩展

* Python蟒蛇绘制问题是各类图像绘制问题的代表
* 图形绘制、五角星绘制、国旗绘制、机器猫绘制…

**2.3 模块1：****turtle库的使用**

**1. turtle库基本介绍**

（1）turtle(海龟)库是turtle绘图体系的Python实现

* turtle绘图体系：1969年诞生，主要用于程序设计入门
* Python语言的标准库之一
* 入门级的图形绘制函数库

（2）标准库

Python计算生态 = 标准库 + 第三方库

* 标准库：随解释器直接安装到操作系统中的功能模块
* 第三方库：需要经过安装才能使用的功能模块
* 库Library、包Package、模块Module，统称模块

（3）turtle库的原理

turtle(海龟)是一种真是的存在

* 有一只海龟，其实在窗体正中心，在画布上游走
* 走过的轨迹形成了绘制的图形
* 海龟由程序控制，可以改变颜色、宽度等

（4）turtle的魅力

**2. turtle绘图窗体布局**

|  |  |
| --- | --- |
| * turtle的一个画布空间 * 最小单位是像素   turtle.setup(width, height, startx, starty)   * setup()设置窗体大小及位置 * 4个参数中后两个可选（不选时，默认在正中心） * setup()不是必须的 |  |

**3. turtle空间坐标体系**

（1）绝对坐标

|  |  |
| --- | --- |
| turtle.goto(x, y) |  |
| import turtle  turtle.goto(100, 100)  turtle.goto(100, -100)  turtle.goto(-100, -100)  turtle.goto(-100, 100)  turtle.goto(0, 0) |

（2）海龟坐标

turtle.bk(d)；turtle.fd(d)；turtle.circle(r,angle)

**4. turtle角度坐标体系**

（1）绝对角度

|  |  |
| --- | --- |
| turtle.seth(angle)   * seth()改变海龟行进方向 * seth()只改变方向但不行进 * angle为绝对度数 |  |
| import turtle  turtle.left(45)  turtle.fd(150)  turtle.right(135)  turtle.fd(300)  turtle.left(135)  turtle.fd(150) |

**5. RGB色彩体系**

（1）RGB色彩模式

* RGB指红蓝绿三个通道的颜色组合
* 覆盖视力所能感知的所有颜色
* RGB每色取值范围0-255整数或0-1小数

（2）常用RGB色彩

|  |  |  |  |
| --- | --- | --- | --- |
| **中文名称** | **英文名称** | **RGB整数值** | **RGB小数值** |
| 白色 | white | 255, 255, 255 | 1, 1, 1 |
| 黄色 | yellow | 255, 255, 0 | 1, 1, 0 |
| 洋红 | magenta | 255, 0, 255 | 1, 0, 1 |
| 青色 | cyan | 0, 255, 255 | 0, 1, 1 |
| 蓝色 | blue | 0, 0, 255 | 0, 0, 1 |
| 黑色 | black | 0, 0, 0 | 0, 0, 0 |
| 海贝色 | seashell | 255, 245, 238 | 1, 0.96, 0.93 |
| 金色 | gold | 255, 215, 0 | 1, 0.84, 0 |
| 粉红色 | pink | 255, 192, 203 | 1, 0.75, 0.80 |
| 棕色 | brown | 165, 42, 42 | 0.65, 0.16, 0.16 |
| 紫色 | purple | 160, 32, 240 | 0.63, 0.13, 0.94 |
| 番茄色 | tomato | 255, 99, 71 | 1, 0.39, 0.28 |

（3）turtle的RGB色彩模式

|  |  |
| --- | --- |
| 默认采用小数值，可切换为整数值：  turtle.colormode(mode) | 1.0：RGB小数值模式  255：RGB整数值模式 |

**2.4 turtle程序语法元素分析**

**1. 库引用与import**

|  |  |
| --- | --- |
| #PythonDraw.py  import turtle  turtle.setup(650, 350, 200, 200)  turtle.penup()  turtle.fd(-250)  turtle.pendown()  turtle.pensize(25)  turtle.pencolor("purple")  turtle.seth(-40)  for i in range(4):  turtle.circle(40, 80)  turtle.circle(-40, 80)  turtle.circle(40, 80/2)  turtle.fd(40)  turtle.circle(16, 180)  turtle.fd(40 \* 2/3)  turtle.done() |  |

（1）库引用：扩充Python程序功能的方式

* 使用import保留字完成，采用<a>.<b>()编码风格

1. import<库名>
2. <库名>.<函数名>(<函数参数>)

（2）import更多用法：使用from和import保留字共同完成

1. from <库名> import <函数名>
2. from <库名> import \*
3. <函数名>(<函数参数>)

（3）import更多用法：两种方法比较

* 第一种方法不会出现函数重名问题
* 第二种方法会出现

（4）import更多用法：使用import和as保留字共同完成

* 给调用的外部库关联一个更短、更适合自己的名字

1. import <库名> as <库别名>
2. <库别名>.<函数名>(<函数参数>)

|  |  |
| --- | --- |
| import turtle  turtle.setup(650, 350, 200, 200)  turtle.penup()  turtle.fd(-250)  turtle.pendown()  turtle.pensize(25)  turtle.pencolor("purple")  turtle.seth(-40)  for i in range(4):  turtle.circle(40, 80)  turtle.circle(-40, 80)  turtle.circle(40, 80/2)  turtle.fd(40)  turtle.circle(16, 180)  turtle.fd(40 \* 2/3)  turtle.done() | import turtle as t  t.setup(650, 350, 200, 200)  t.penup()  t.fd(-250)  t.pendown()  t.pensize(25)  t.pencolor("purple")  t.seth(-40)  for i in range(4):  t.circle(40, 80)  t.circle(-40, 80)  t.circle(40, 80/2)  t.fd(40)  t.circle(16, 180)  t.fd(40 \* 2/3)  t.done() |

**2. turtle画笔控制函数**

|  |
| --- |
| penup()  pendown()  pensize(width)  pencolor(colorstring)  pencolor(r,g,b)  pencolor((r,g,b)) |

（1）画笔操作后一直有效，一般成对出现

* turtle.penup()，别名turtle.pu()：抬起画笔，海龟在飞行
* turtle.pendown()，别名turtle.pd()：落下画笔，海龟在爬行

（2）画笔设置后一直有效，直至下次重新设置

* turtle.pensize(width)，别名turtle.width(width)：画笔宽度，海龟的腰围
* turtle.pencolor(color)，color为颜色字符串或r,g,b值：画笔颜色，海龟在涂装

（3）pencolor(color)的color参数可以有三种形式

* 颜色字符串：turtle.pencolor("purple")
* RGB的小数值：turtle.pencolor(0.63, 0.13, 0.94)
* RGB的元组值：turtle.pencolor((0.63, 0.13, 0.94))

**3. turtle运动控制函数**

（1）控制海龟行进：走直线&走曲线

①turtle.forward(d)，别名turtle.fd(d)：向前行进，海龟走直线

d：形近距离，可以为负数

②turtle.circle(r, extent=None)：根据半径r绘制extent角度的弧形

r：默认圆心在海龟左侧距离r距离的位置

extent：绘制角度，默认是360度圆角

**4. turtle方向控制函数**

（1）控制海龟面对方向：绝对角度&海龟角度

①turtle.setheading(angle)，别名turtle.seth(angle)：改变行进方向，海龟走角度

angle：改变行进方向，海龟走角度

②turtle.left(angle)：海龟向左转

turtle.right(angle)：海龟向右转

angle：在海龟当前前行方向上旋转角度

**5.** **循环语句与range()函数**

for和in保留字range()

（1）循环语句：按照一定次数循环执行一组语句

|  |
| --- |
| for <变量> in range(<参数>)  <被执行的语句> |

<变量>表示每次循环的次数，0到<次数>-1

|  |  |
| --- | --- |
| >>> for i in range(5):  print(i)  0  1  2  3  4 | >>> for i in range(5):  print("Hello:",i)  Hello: 0  Hello: 1  Hello: 2  Hello: 3  Hello: 4 |

* print()中用逗号隔开，两边字符打印出来会用空格隔开

（2）range()函数：产生循环计数序列

* range(N)：产生0到N-1个整数序列，共N个
* range(M, N)：产生M到N-1个整数整数序列，共N-M个

**6. “Python蟒蛇绘制”代码分析**

|  |
| --- |
| import turtle  turtle.setup(650, 350, 200, 200)  turtle.penup()  turtle.fd(-250)  turtle.pendown()  turtle.pensize(25)  turtle.pencolor("purple")  turtle.seth(-40)  for i in range(4):  turtle.circle(40, 80)  turtle.circle(-40, 80)  turtle.circle(40, 80/2)  turtle.fd(40)  turtle.circle(16, 180)  turtle.fd(40 \* 2/3)  turtle.done() |

**2.n 第二章作业及学习资料**

**一、单选题**

1. 哪个选项不能正确引用turtle库进而使用setup()函数？

|  |  |  |  |
| --- | --- | --- | --- |
| A. from turtle import\* | B. import turtle | C. import turtle as t | D. import setup from turtle |
| import只有三种使用方法，以turtle库为例：  import turtle  from turtle import setup 或 from turtle import \*  import turtle as t（其中t是别名，可以更换其他名称） | | | |

2. 关于turtle库，哪个选项的描述是错误的？

|  |
| --- |
| A. turtle库是一个直观有趣的图形绘制函数库 |
| B. turtle库最早成功应用于LOGO编程语言 |
| C. turtle坐标系的原点默认在屏幕左上角 |
| D. turtle绘图体系以水平右侧为绝对方位的0度 |
| turtle坐标系的原点默认在屏幕正中间 |

3. 哪个选项是turtle绘图中角度坐标系的绝对0度方向？

|  |  |  |  |
| --- | --- | --- | --- |
| A. 画布正右方 | B. 画布正左方 | C. 画布正下方 | D. 画布正上方 |
| import只有三种使用方法，以turtle库为例：  import turtle  from turtle import setup 或 from turtle import \*  import turtle as t（其中t是别名，可以更换其他名称） | | | |

4. 那个选项是下面代码的执行结果：

|  |
| --- |
| turtle.circle(-90,90) |
| A. 绘制一个半径为90像素的弧形，圆心在小海龟当前行进的右侧 |
| B. 绘制一个半径为90像素的整圆形 |
| C. 绘制一个半径为90像素的弧形，圆心在小海龟当前行进的左侧 |
| D. 绘制一个半径为90像素的弧形，圆心在画布正中心 |
| circle(x, y)表示以x长度为半径，y为角度，当前方向左侧x出为圆心，画圆。其中x和y都可以是负数，相应取反。 |

5. 关于turtle库绘图函数，哪个选项的描述是错误的？

|  |
| --- |
| A. turtle.pensize(size)函数的作用是改变画笔的宽度为size像素 |
| B. turtle.fd(distance)函数的作用是向小海龟当前行进方向前进distance距离 |
| C. turtle.circle(radius, extent=None)函数的作用是绘制一个椭圆形，extent参数可选 |
| D. turtle.seth(to\_angle)函数的作用是设置小海龟当前行进方向为to\_angle，to\_angle是角度的整数值 |
| circle()函数不能绘制椭圆形。 |

6. 关于turtle库的画笔控制函数，哪个选项的描述是错误的？

|  |
| --- |
| A. turtle.colormode()的作用是设置画笔RGB颜色的表示模式 |
| B. turtle.penup()的别名有turtle.pu(),turtle.up() |
| C. turtle.pendown()作用是落下画笔，并移动画笔绘制一个点 |
| D. turtle.width()和turtle.pensize()都可以用来设置画笔尺寸 |
| turtle.pendown()只是放下画笔，并不绘制任何内容。 |

7. 哪个选项不能改变turtle画笔的运行方向？

|  |  |  |  |
| --- | --- | --- | --- |
| A. right() | B. left() | C. bk() | D. seth() |
| bk()只能后退，但不改变方向，"后退"不是"转向"。 | | | |

8.

|  |  |  |  |
| --- | --- | --- | --- |
| A. if和else | B. while和def | C. range() | D. for和in |
| 循环相关保留字是：for..in和while，但def用于定义函数，不相关。 | | | |

9 哪个选项能够使用turtle库绘制一个半圆形？

|  |  |  |  |
| --- | --- | --- | --- |
| A. turtle.circle(100, 90) | B. turtle.circle(100, -180) | C. turtle.fd(100) | D. turtle.circle(100) |
| circle(x,y)函数的用法，绘制半圆，第二个参数y是180的奇数倍。 | | | |

10. 哪个选项对turtle.done()的描述是正确的？

|  |
| --- |
| A. turtle.done()用来停止画笔绘制，但绘图窗体不关闭 |
| B. turtle.done()用来暂停画笔绘制，用户响应后还可以继续绘制 |
| C. turtle.done()用来隐藏turtle绘制画笔，一般放在代码最后 |
| D. turtle.done()放在代码最后，是turtle绘图的必要要求，表示绘制完成 |
| 建议在每个turtle绘图最后增加turtle.done()。 |

**二、程序题**

1. turtle八边形绘制

|  |  |
| --- | --- |
|  | #请在横线中填写Python表达式或语句，实现所需要的功能  #注意：补充代码将以匹配方式评阅，代码中不要出现空格  import turtle as t  t.pensize(2)  for i in range(8):  t.fd(100)  t.left(45) |

2. turtle八边图形绘制

|  |  |
| --- | --- |
|  | #请在横线中填写Python表达式或语句，实现所需要的功能  #注意：补充代码将以匹配方式评阅，代码中不要出现空格  import turtle as t  t.pensize(2)  for i in range(8):  t.fd(100)  t.left(135) |

**3.1 数字类型及操作**

**1.** **整数类型**

（1）与数学中整数的概念一致

* 可正可负，没有取值范围限制
* pow(x,y)函数：计算xy，想算多大算多大

|  |  |
| --- | --- |
| >>> pow(2,100)  1267650600228229401496703205376 | >>> pow(2,pow(2,15))  1415461031044954789001553······ |

（2）4种进制表示形式

* 十进制：（如：1010，99，-217）
* 二进制：以0b或0B开头（如：0b010，-0B101）
* 八进制：以0o或0O开头（如：0o123，-0O456）
* 十六进制：以0x或0X开头（如：0x9a，-0X89）

**2. 浮点数类型**

（1）与数学中实数的概念一致

* 带有小数点及小数的数字
* 浮点数值取值范围和小数精度都存在限制，但常规计算可忽略
* 取值范围数量级约-10308至10308，精度数量级10-16

（2）浮点数间运算存在不确定维数，不是bug

|  |  |
| --- | --- |
| >>> 0.1 + 0.3  0.4  >>> 0.1 + 0.2  0.30000000000000004 | >>> 0.1 + 0.2 == 0.3  False  >>> round(0.1 + 0.2, 1) == 0.3  True |
| 0.1（53位二进制表示小数部分，约10-16）  0.00011001100110011001100110011001100110011001100110011010（二进制表示）  0.1000000000000000055511151231257827021181583404541015625 （十进制表示）  0.1+0.2结果无限接近0.3，但可能存在尾数（二进制表示小数，可以无限接近，但不完全相同） | |

* round(x, d)：对x四舍五入，d是小数截取位数
* 浮点数间运算运算及比较用round()函数辅助
* 不确定尾数一般发生在10-16左右，round()十分有效

（3）浮点数可以采用科学计数法表示

* 使用字母e或E作为幂的符号，以10为基数，格式为：<a>e<b> 表示a\*10b
* 例如：4.3e-3值为0.0043；9.6E5值为960000.0

（4）关于Python浮点数，需要知道多些

* 取值范围和精度基本无限制
* 运算存在不确定尾数 round()
* 科学计数法表示

**3. 复数类型**

（1）与数学中复数的概念一致

* 如果x2=-1，那么x的值是什么？
* 定义，以此为基础，构件数学体系
* a+bj被称为复数，其中，a是实部，b是虚部

（2）复数实例

* Z = 1.23e-4 + 5.6e +89j
* 实部是什么？——z.real获得实部
* 虚部是什么？——z.imag获得虚部

**4.** **数值运算操作符**

（1）操作符是完成运算的一种符号体系

|  |  |
| --- | --- |
| 操作符及使用 | 描述 |
| x + y | 加，x与y之和 |
| x – y | 减，x与y之差 |
| x \* y | 乘，x与y之积 |
| x / y | 除，x与y之商（10/3结果是3.3333333333333335）（两数相除为浮点数结果） |
| x // y | 整数除，x与y之整数商（10//3结果是3） |
| + x | x本身 |
| - y | y的负值 |
| x % y | 余数，模运算（10%3结果是1） |
| x \*\* y | 幂运算，x的y次幂，xy（当y是小数，开方运算10\*\*0.5结果是） |

（2）二元操作符有对应的增强赋值操作符

|  |  |
| --- | --- |
| 增强操作符及使用 | 描述 |
| x op= y | 即x = x op y，其中，op为二元操作符 |
| x += y；x -= y；x \*= y；x /= y；x //= y；x %= y；x \*\*= y |
| >>> x = 3.1415  >>> x \*\*= 3 # 与x = x \*\* 3等价  >>> x  31.003533398375005 |

（3）类型间可进行混合运算，生成结果为“最宽”类型

* 三种类型存在一种逐渐“扩展”或“变宽”的关系：整数→浮点数→复数
* 例如：123 + 4.0 = 127.0（整数 + 浮点数 = 浮点数）

**5.** **数值运算函数**

（1）一些以函数形式提供的数值运算功能

|  |  |  |
| --- | --- | --- |
| 函数及使用 | 描述 | |
| abs(x) | 绝对值，x的绝对值 | abs(-10.01)结果为10.01 |
| divmod(x, y) | 商余，(x//y, x%y)，同时输出商和余数 | divmod(10, 3)结果为(3, 1) |
| pow(x, y[, z]) | 幂余，(x\*\*y)%z，[…]表示参数z可省略 | pow(3, pow(3, 99), 10000)结果为4587 |
| round(x[, d]) | 四舍五入，d是保留小数位数，默认值是0 | round(-10.123, 2)结果为-10.12 |
| max(x1, x2, … , xn) | 最大值，返回x1, x2, … , xn中的最大值，n不限 | max(1, 9, 5, 4, 3)结果为9 |
| min(x1, x2, … , xn) | 最小值，返回x1, x2, … , xn中的最小值，n不限 | min(1, 9, 5, 4, 3)结果为1 |
| int(x) | 将x变成整数，舍弃小数部分 | int(123.45)结果为123；int("123")结果为123 |
| float(x) | 将x变成浮点数，增加小数部分 | float(12)结果为12.0；float("1.23")结果为1.23 |
| complex(x) | 将x变成复数，增加虚数部分 | complex(4)结果为4+0j |

**3.2 实例3：天天向上的力量**

**1. “天天向上的力量”问题分析**

（1）基本问题：持续的价值

* 一年365天，每天进步1%，累计进步多少呢？——1.01365
* 一年365天，每天退步1%，累计剩下多少呢？——0.99365
* 数学公式可以求解，似乎没必要用程序
* 如果是“三天打鱼两天晒网”呢？
* 如果是“双休日又不退步呢？”

**2. “天天向上的力量”第一问**

（1）问题1：1‰的力量

* 一年365天，每天进步1‰，累计进步多少呢？——1.001365
* 一年365天，每天退步1‰，累计剩下多少呢？——0.999365

|  |
| --- |
| # DayDayUpQ1.py  dayup = pow(1.001, 365)  daydown = pow(0.999, 365)  print("向上：{:.2f}，向下：{:.2f}".format.(dayup, daydown)) |

* （运行结果）向上：1.44，向下：0.69（1.001365=1.44；0.999365=0.69）

**3. “天天向上的力量”第二问**

（1）问题2：5‰和1%的力量

* 一年365天，每天进步5‰或1%，累计进步多少呢？——1.005365；1.01365
* 一年365天，每天退步5‰或1%，累计剩下多少呢？——0.995365；0.99365

|  |  |
| --- | --- |
| # DayDayUpQ2.py  dayfactor = 0.005  dayup = pow(1 + dayfactor, 365)  daydown = pow(1 - dayfactor, 365) | # DayDayUpQ2.py  dayfactor = 0.01  dayup = pow(1 + dayfactor, 365)  daydown = pow(1 - dayfactor, 365) |
| print("向上：{:.2f}，向下：{:.2f}".format.(dayup, daydown)) | |

* （运行结果）向上：6.17，向下：0.16（1.005365=6.17；0.995365=0.16）
* （运行结果）向上：37.78，向下：0.03（1.01365=37.78；0.99365=0.03）

**4. “天天向上的力量”第三问**

（1）问题3：工作日的力量

* 一年365天，一周5个工作日，每天进步1‰
* 一年365天，一周2个休息日，每天退步1‰
* 这种工作日的力量，如何呢？
* 1.01365（数学思维）→for…in…（计算思维）

|  |
| --- |
| # DayDayUpQ3.py  dayup = 1.0  dayfactor = 0.01  for i in range(365):  if i % 7 in [6, 0]:  dayup = dayup \* (1 - dayfactor)  else:  dayup = dayup \* (1 + dayfactor)  print("工作日的力量：{:.2f}".format(dayup)) |

* （运行结果）工作日的力量：4.63

**5. “天天向上的力量”第四问**

（1）问题4：工作日的努力

* 工作日模式要努力到什么水平，才能达到每天努力1%一样？
* A君：一年365天，每天进步1%，不停歇
* B君：一年365天，每周工作5天休息2天，休息日下降1%，要多努力呢？
* for…in…（计算思维）→def…while…（“笨办法”试错）

|  |
| --- |
| # DayDayUpQ4.py  def dayUP(df):  dayup = 1  for i in range(365):  if i % 7 in [6, 0]:  dayup = dayup \* (1 - 0.01)  else:  dayup = dayup \* (1 + df)  return dayup  dayfactor = 0.01  while dayUP(dayfactor) < 37.78:  dayfactor += 0.001  print("工作日的努力参数是：{:.3f}".format(dayfactor)) |

* （运行结果）工作日的努力参数是：0.02

**6. “天天向上的力量”举一反三**

（略）

**3.3 字符串类型及操作**

**1. 字符串类型的表示**

（1）字符串：由0个或多个字符组成的有序字符序列

* 字符串由一对单引号或一对双引号表示（如："请输入带有符号的温度值："或者"C"）
* 字符串是字符的有序序列，可以对其中的字符进行索引（"请"是"请输入带有符号的温度值："的第0个字符）

（2）字符串有2类共4种表示方式

1. 由一对单引号或双引号表示，仅表示单行字符串
2. 由一对三单引号或三双引号表示，可表示多行字符串
3. 如果希望在字符串中包含双引号或单引号呢？
4. 如果希望在字符串中既包括单引号又包括双引号呢？

|  |
| --- |
| "请输入带有符号的温度值："或者"C" |
| '''Python  语言''' |
| '这里有个双引号(")' 或者 "这里有个单引号(')" |
| '''这里既有单引号(')又有双引号(")'''' |

（3）字符串切片高级用法：使用[M: N: K]根据步长对字符串切片

* <字符串>[M: N]，M缺失表示至开头，N缺失表示至结尾
* "〇一二三四五六七八九十"[:3]结果是"〇一二"
* <字符串>[M: N: K]，根据步长K对字符串切片
* "〇一二三四五六七八九十"[1:8:2]结果是"一三五七"
* "〇一二三四五六七八九十"[::-1]结果是"十九八七六五四三二一〇"

（4）字符串的特殊字符：转义符

* 转移符表达特定字符的本意
* "这里有个双引号(\")"结果为 这里有个双引号(")
* 转义符形成一些组合，表达一些不可打印的含义
* "\b"回退 "\n"换行（光标移动到下行首）
* "\r"回车（光标移动到本行首）

**2.** **字符串操作符**

（1）由0个或多个字符组成的有序字符序列

|  |  |
| --- | --- |
| 操作符及使用 | 描述 |
| x + y | 连接两个字符串x和y |
| n \* x 或 x \* n | 复制n次字符串x |
| x in s | 如果x是s的字串，返回True，否则返回False |

（3）获取星期字符串

* 输入：1-7的整数，表示星期几
* 输出：输入整数对应的星期字符串
* 例如：输入3，输出星期三

|  |
| --- |
| #WeekNamePrintV1.py  weekStr = "星期一星期二星期三星期四星期五星期六星期日"  weekId = eval(input("请输入星期数字(1-7)："))  pos = (weekId - 1) \* 3  print(weekStr[pos: pos+3]) |
| #WeekNamePrintV2.py  weekStr = "一二三四五六七"  weekId = eval(input("请输入星期数字(1-7)："))  print("星期" + weekStr[weekId-1]) |

**3.** **字符串处理函数**

（1）一些以函数形式提供的字符串处理功能

|  |  |  |
| --- | --- | --- |
| 函数及使用 | 描述 | |
| len(x) | 长度，返回字符串x的长度 | len("一二三456")结果为6 |
| str(x) | 任意整数类型x所对应的字符串形式 | str(1.23)结果为"1.23" str([1,2])结果为"[1,2]" |
| hex(x)或oct(x) | 整数x的十六进制或八进制小写形式字符串 | hex(425)结果为"0x1a9" oct(425)结果为"0o651" |
| chr(u) | u为Unicode编码，返回其对应的字符 |  |
| ord(x) | x为字符，返回其对应的Unicode编码 |  |

（2）Unicode编码：Python字符串的编码方式

* 统一字符编码，即覆盖几乎所有字符的编码方式
* 从0到1114111（0x10FFFF）空间，每个编码对应一个字符
* Python字符串中每个字符都是Unicode编码字符

|  |
| --- |
| >>> "1 + 1 = 2" + chr(10004)  '1 + 1 = 2✓'  >>> "这个字符♉的Unicode值是：" + Str(ord("♉"))  Traceback (most recent call last):  File "<pyshell#1>", line 1, in <module>  "这个字符♉的Unicode值是：" + Str(ord("♉"))  NameError: name 'Str' is not defined  >>> "这个字符♉的Unicode值是：" + str(ord("♉"))  '这个字符♉的Unicode值是：9801'  >>> for i in range(12):  print(chr(9800 + i), end ="")  ♈♉♊♋♌♍♎♏♐♑♒♓ |

**4.** **字符串处理方法**

（1）“方法”在编程中是一个专有名词

* “方法”特指<a>.<b>()风格中的函数<b>()
* 方法本身也是函数，但与<a>有关，<a>.<b>()风格使用
* 字符串及变量也是<a>，存在一些方法

（2）一些以方法形式提供的字符串处理功能

|  |  |  |
| --- | --- | --- |
| 方法及使用 | 描述 | |
| str.lower()或str.upper() | 返回字符串的副本，全部字符小写/大写 | "AbCdEfGh".lower()结果为"abcdefgh" |
| str.split(sep=None) | 返回一个列表，由str根据sep被分隔的部分组成 | "A,B,C".split(",")结果为['A', 'B', 'C'] |
| str.count(sub) | 返回子串sub在str中出现的次数 | "an apple a day".count(a)结果为4 |
| str.replace(old, new) | 返回字符串str副本，所有old子串被替换为new | "python".replace("n","n123.io")结果为"python123.io" |
| str.center(width[,fillchar]) | 字符串str根据宽度width居中，fillchar可选 | "python".center(20, "=")结果为'=======python=======' |
| str.strip(chars) | 从str中去掉在其左侧和右侧chars中列出的字符 | "= python=".strip(" =np")结果为"ytho |
| str.join(iter) | 在iter变量除最后元素外每个元素后增加一个str | ",".join("123456"结果为"1,2,3,4,5" # 主要用于字符串分隔等 |

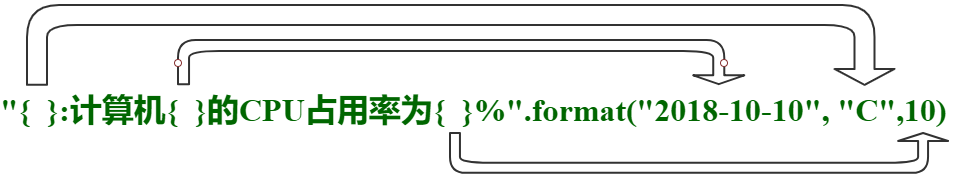
**5.** **字符串类型的格式化**

（1）格式化是对字符串进行格式化表达的方式

* 字符串格式化使用.format()方法，用法如：<模板字符串>.format(<逗号分隔的参数>)

（2）槽

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| "{ }:计算机{  }的CPU占用率为{ }%".format("2018-10-10", "C", 10) | | | | | |
| 0 | 1 | 2 | 0 | 1 | 2 |
| 字符串中槽{}的默认顺序 format()中参数的顺序 | | | | | |



（3）format()方法的格式控制

* 槽内部对格式化的配置方式：{<参数序号> : <格式控制标记>}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| : | <填充> | <对齐> | <宽度> | <, > | <. 精度> | <类型> |
| 引导符号 | 用于填充的单个字符 | < 左对齐  > 右对齐  ^ 居中对齐 | 槽设定的输出宽度 | 数字的千位分隔符 | 浮点数小数精度或字符串最大输出长度 | 整数类型  b, c, d, o, x, X  浮点数类型  e, E, f, % |
| >>> "{0:=^20}".format("PYTHON")  '=======PYTHON======='  >>> "{0:\*>20}".format("BIT")  '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*BIT'  >>> "{:10}".format("BIT")  'BIT ' | | | | | | |
| >>> "{0:,.2f}".format(12345.6789)  '12,345.68'  >>> "{0:b},{0:c},{0:d},{0:o},{0:x},{0:X}".format(425)  '110101001,Ʃ,425,651,1a9,1A9'  >>> "{0:e},{0:E},{0:f},{0:%}".format(3.14)  '3.140000e+00,3.140000E+00,3.140000,314.000000%' | | | | | | |

**3.4 模块2：****time库的使用**

**1. time库基本介绍**

（1）time库是Python中处理时间的标准库

* 计算机时间的表达
* 提供获取系统时间并格式化输出功能
* 提供系统级精确计时功能，用于程序性能分析

|  |
| --- |
| import time  time.<b>() |

（2）time库包括三类函数

* 时间获取：time()，ctime()，gmtime()
* 时间格式化：strftime()，strptime()
* 程序计时：sleep()，perf\_counter()

**2. 时间获取**

|  |  |  |
| --- | --- | --- |
| 函数 | 描述 | >>> import time |
| time() | 获取当前时间戳，即计算机内部时间值，浮点数 | >>> time.time()  1563180824.2236238 |
| ctime() | 获取当前时间并以易读方式表示，返回字符串 | >>> time.ctime()  'Mon Jul 15 16:53:53 2019' |
| gmtime() | 获取当前时间，表示为计算机可处理的时间格式 | >>> time.gmtime()  time.struct\_time(tm\_year=2019, tm\_mon=7, tm\_mday=15, tm\_hour=8, tm\_min=56, tm\_sec=13, tm\_wday=0, tm\_yday=196, tm\_isdst=0) |

**3. 时间格式化：将时间以合理的方式展示出来**

* 格式化：类似字符串格式化，需要有展示模板
* 展示模板由特定的格式化控制符组成

|  |  |  |
| --- | --- | --- |
| 函数 | 描述 | >>> import time |
| strftime(tpl, ts) | tpl是格式化模板字符串，定义输出效果  ts是计算机内部时间类型变量 | >>> t = time.gmtime()  >>> time.strftime("%Y-%m-%d %H:%M:%S",t)  '2019-07-15 09:03:36' |
| strptime(str, tpl) | str是字符串形式的时间值  tpl是格式化模板字符串，定义输出效果 | >>> timeStr = '2019-07-15 09:03:36'  >>> time.strptime(timeStr, "%Y-%m-%d %H:%M:%S")  time.struct\_time(tm\_year=2019, tm\_mon=7, tm\_mday=15, tm\_hour=9, tm\_min=3, tm\_sec=36, tm\_wday=0, tm\_yday=196, tm\_isdst=-1) |

* 时间格式化控制符

|  |  |  |
| --- | --- | --- |
| 格式化字符串 | 日期/时间说明 | 值范围和实例 |
| %Y | 年份 | 0000~9999，例如：1900 |
| %m | 月份 | 01~12，例如：10 |
| %B | 月分名称 | January~December，例如：April |
| %b | 月份名称缩写 | Jan~Dec，例如：Apr |
| %d | 日期 | 01~31，例如：25 |
| %A | 星期 | Monday~Sunday，例如：Wednesday |
| %a | 星期缩写 | Mon~Sun，例如：Wed |
| %H | 小时（24h制） | 00~23，例如：12 |
| %I | 小时（12h制） | 01~12,liru :7 |
| %p | 上/下午 | AM/PM，例如：PM |
| %M | 分钟 | 00~59，例如：26 |
| %S | 秒 | 00~59，例如：26 |

**4. 程序计时应用**

* 程序计时指测量起止动作所经历时间的过程
* 测量时间：perf\_counter()
* 产生时间：sleep()

|  |  |  |
| --- | --- | --- |
| 函数 | 描述 | >>> import time |
| perf\_counter() | 返回一个CPU级别的精确时间计数值，单位为秒  由于这个计数值起点不确定，连续调用差值才有意义 | >>> start = time.perf\_counter()  >>> start  102.144764115  >>> end = time.perf\_counter()  >>> end  119.327769536  >>> end - start  17.183005421000004 |
| sleep(s) | s拟休眠的时间，单位是秒，可以是浮点数 | >>> def wait():  time.sleep(3.3)  >>> wait() #程序将等待3.3秒后再退出 |

**3.5 实例4：文本进度条**

**1. “文本进度条”问题分析**

（1）文本进度条

* 采用字符串方式打印可以动态变化的文本进度条
* 进度条需要能在一行中逐渐变化

（2）如何获得文本进度条的变化时间？

* 采用sleep()模拟一个持续的进度

**2. “文本进度条”简单的开始**

|  |  |
| --- | --- |
| #TextProBarV1.py  import time  scale = 10  print("------执行开始------")  for i in range(scale + 1):  a = '\*' \* i  b = '.' \* (scale - i)  c = (i/scale)\*100  print("{:^3.0f}%[{}->{}]".format(c,a,b))  time.sleep(0.1)  print("------执行结束------") | ------执行开始------  0 %[->..........]  10 %[\*->.........]  20 %[\*\*->........]  30 %[\*\*\*->.......]  40 %[\*\*\*\*->......]  50 %[\*\*\*\*\*->.....]  60 %[\*\*\*\*\*\*->....]  70 %[\*\*\*\*\*\*\*->...]  80 %[\*\*\*\*\*\*\*\*->..]  90 %[\*\*\*\*\*\*\*\*\*->.]  100%[\*\*\*\*\*\*\*\*\*\*->]  ------执行结束------ |

**2. “文本进度条”单行动态刷新**

* 刷新的本质是：用后打印的字符覆盖之前的字符
* 不能换行：print()需要被控制
* 要能回退：打印后光标退回到之前的位置\r

|  |  |
| --- | --- |
| #TextProBarV2.py  import time  for i in range(101):  print("\r{:3}%".format(i), end="")  time.sleep(0.1) |  |

**3. “文本进度条”实例完整效果**

|  |
| --- |
| #TextProBarV3.py  import time  scale = 50  print("执行开始".center(scale//2, "-"))  start = time.perf\_counter()  for i in range(scale+1):  a = '\*' \* i  b = '.' \* (scale - i)  c = (i/scale)\*100  dur = time.perf\_counter() - start  print("\r{:^3.0f}%[{}->{}]{:.2f}s".format(c,a,b,dur),end='')  time.sleep(0.1)  print("\n"+"执行结束".center(scale//2,'-')) |

**4. “文本进度条”举一反三**

（1）计算问题扩展

* 文本进度条程序使用了perf\_counter(0计时
* 计时方法适合各类需要统计时间的计算问题
* 例如：比较不同算法时间、统计部分程序运行时间

（2）进度条扩展

* 在任何运行时间需要较长的程序中增加进度条
* 在任何希望提高用户体验的应用中增加进度条
* 进度条是人机交互的纽带之一

（3）文本进度条的不同设计函数

|  |  |  |
| --- | --- | --- |
| 设计名称 | 趋势 | 设计函数 |
| Linear | Constant | f(x) = x |
| Early Pause | Speeds up | f(x) = x + (1-sinx(x\*π\*2+π/2)/-8 |
| Late Pulse | Slows down | f(x) = x + (1-sinx(x\*π\*2+π/2)/8 |
| Slow Wavy | Constant | f(x) = x + sinx(x\*π\*5)/20 |
| Fast Wavy | Constant | f(x) = x + sinx(x\*π\*20)/80 |
| Power | Speeds up | f(x) = (x + (1 - x)\*0.03)2 |
| Inverse Power | Speeds down | f(x) = 1 + (1 - x)1.5\*-1 |
| Fast Power | Speeds up | f(x) = (x + (1 – x)/2)8 |
| Inverse Fast Power | Slows down | f(x) = 1 + (1-x)3\*-1 |

**4.1 程序的分支结构**

**1. 单分支结构：根据判断条件结果而选择不同向前**

|  |  |
| --- | --- |
| if <条件>:  <语句块> | |
| guess = eval(imput())  if guess == 99  print("猜对了") | if Ture:  print("条件正确") |

**2.** **二分支结构：根据判断条件结果而选择不同向前路劲的运行方式**

|  |  |
| --- | --- |
| if <条件>:  <语句块1>  else:  <语句块2> | |
| guess = eval(imput())  if guess == 99  print("猜对了")  else:  print("猜错了") | if Ture:  print("语句块1")  else:  print("语句块2") |

* 紧凑形式：适用于简单表达式的二分支结构

<表达式1> if <条件> else <表达式2>

|  |
| --- |
| guess = eval(input())  print("猜{}了".format("对" if guess==99 else "错")) |

**3.** **多分支结构：根据多个条件来选择不同语句块运行的分支结构**

|  |
| --- |
| if <条件>:  <语句块1>  elif:  <语句块2>  ……  else:  <语句块n> |
| score = eval(input())  if score >= 90:  grade = "A"  elif score >= 80:  grade = "B"  elif score >= 70:  grade = "C"  elif score >= 60:  grade = "D"  prit("输出成绩属于级别{}".format(grade)) |

**4.** **条件判断及组合**

（1）操作符

|  |  |  |
| --- | --- | --- |
| 操作符 | 数学符号 | 描述 |
| < | < | 小于 |
| <= | ≤ | 小于等于 |
| >= | ≥ | 大于等于 |
| > | > | 大于 |
| == | = | 等于 |
| != | ≠ | 不等于 |

（2）用于条件组合的三个保留字

|  |  |
| --- | --- |
| 操作符 | 描述 |
| x and y | 两个条件x和y的逻辑与 |
| x or y | 两个条件x和y的逻辑或 |
| not x | 条件x的逻辑非 |
| guess = eval(input())  if guess > 99 or guess < 99:  print("猜错了")  else:  print("猜对了") | if not True:  print("语句块2")  else:  print("语句块1") |

**5. 程序的异常处理**

|  |  |
| --- | --- |
| num = eval(input("请输入一个整数："))  print(num \*\* 2)  >>>  请输入一个整数：abc | Traceback (most recent call last):  File "t.py", line 1, in <module>  num = eval(input("请输入一个整数："))  File "<string>", line 1, in <module>  NameError: name 'abc' is not defined |

（1）异常处理的基本使用

|  |  |
| --- | --- |
| try:  <语句块1>  except:  <语句块2> | try:  <语句块1>  except <异常类型>:  <语句块2> |
| try:  num = eval(input("请输入一个整数："))  print(num\*\*2)  except:  print("输入不是整数") | try:  num = eval(input("请输入一个整数："))  print(num\*\*2)  except NameError:  print("输入不是整数") |

（2）异常处理的高级使用

|  |  |
| --- | --- |
| try:  <语句块1>  except:  <语句块2>  else:  <语句块3>  finally:  <语句块4> | -else对应语句块4一定执行  -finally对应语句块4一定执行 |

**4.2 实例5：身体质量指数BMI**

**1. “身体质量指数BMI”问题分析**

（1）BMI(Body Mass Index)：国际上常用的衡量人体肥胖和健康程度的重要标准，主要用于统计分析

（2）定义：BMI = 体重(kg) / 身高2(m2)

（3）国际（世界卫生组织） 国内（国家卫生健康委员会）

|  |  |  |
| --- | --- | --- |
| 分类 | 国际BMI值(kg/m2) | 国内BMI值(kg/m2) |
| 偏瘦 | <18.5 | <18.5 |
| 正常 | 18.5~25 | 18.5~24 |
| 偏胖 | 25~30 | 24~28 |
| 肥胖 | ≥30 | ≥28 |

（4）问题需求

* 输入：给定体重和身高值
* 输出：BMI指标分类信息（国际和国内）

**2. “身体质量指数BMI”实例讲解**

（1）思路方法：难点在于同时输出国际和国内对应的分类

* 思路1：分别计算并给出国际和国内BMI分类
* 思路2：混合计算并给出国际和国内BMI分类

|  |  |
| --- | --- |
| #CalBMIv1.py  height, weight = eval(input("请输入身高(米)和体重\(公斤)[逗号隔开]："))  bmi = weight / pow(height, 2)  print("BMI 数值为：{:.2f}".format(bmi))  who = ""  if bmi < 18.5:  who = "偏瘦"  elif 18.5 <= bmi < 25:  who = "正常"  elif 25 <= bmi < 30:  who = "偏胖"  else:  who = "肥胖"  print("BMI 指标为：国际'{0}'".format(who)) | #CalBMIv2.py  height, weight = eval(input("请输入身高(米)和体重\(公斤)[逗号隔开]："))  bmi = weight / pow(height, 2)  print("BMI 数值为：{:.2f}".format(bmi))  nat = ""  if bmi < 18.5:  nat = "偏瘦"  elif 18.5 <= bmi < 24:  nat = "正常"  elif 24 <= bmi < 28:  nat = "偏胖"  else:  nat = "肥胖"  print("BMI 指标为：国际'{0}'".format(nat)) |
| #CalBMIv3.py  height, weight = eval(input("请输入身高(米)和体重\(公斤)[逗号隔开]："))  bmi = weight / pow(height, 2)  print("BMI 数值为：{:.2f}".format(bmi))  who, nat = "", ""  if bmi < 18.5:  who, nat = "偏瘦", "偏瘦"  elif 18.5 <= bmi < 24:  who, nat = "正常", "正常"  elif 24 <= bmi < 25:  who, nat = "正常", "偏胖"  elif 25 <= bmi < 28:  who, nat = "偏胖", "偏胖"  elif 28 <= bmi < 30:  who, nat = "偏胖", "肥胖"  else:  who, nat = "肥胖", "肥胖"  print("BMI指标为：国际'{0}'，国内'{1}'".format(who,nat)) | |

**3. “身体质量指数BMI”举一反三**

（1）关注多分支条件的组合

* 多分支条件之间的覆盖是重要问题
* 程序可运行，单不正确，要多注意分支
* 分支结构是程序的重要框架，读程序先读分支

**4.3 程序的循环结构**

**1.** **遍历循环：遍历某个结构形成的循环运行方式**

|  |
| --- |
| for <循环变量> in <遍历结构>:  <语句块> |

* 由保留字for和in组成，完整遍历所有元素后结束
* 每次循环，所获得元素放入循环变量，并执行一次语句块

（1）计数循环（N次）

* 遍历由range()函数产生的数字序列，产生循环

|  |  |
| --- | --- |
| for i in range(N):  <语句块> | |
| >>> for i in range(5):  print(i)  0  1  2  3  4 | >>> for i in range(5):  print("Hello：",i)  Hello： 0  Hello： 1  Hello： 2  Hello： 3  Hello： 4 |

（2）计数循环（特定次）

|  |  |
| --- | --- |
| for i in range(M, N, K):  <语句块> | |
| >>> for i in range(1, 6):  print(i)  1  2  3  4  5 | >>> for i in range(1, 6, 2):  print("Hello：",i)  Hello： 1  Hello： 3  Hello： 5 |

（3）字符串遍历循环

|  |
| --- |
| for c in s:  <语句块> |
| >>> for c in "Python123":  print(c, end=",")  P,y,t,h,o,n,1,2,3, |

（4）列表遍历循环

|  |
| --- |
| for item in ls:  <语句块> |
| >>> for item in [123, "PY", 456]:  print(item, end=",")  123,PY,456, |

（5）文件遍历结构

|  |  |
| --- | --- |
| for line in fi:  <语句块> | |
| 【文本文件】  “优美胜于丑陋  明了胜于隐晦  简介胜于复杂” | >>> for line in fi:  print(line)  优美胜于丑陋  明了胜于隐晦  简介胜于复杂 |

**2.** **无限循环：由条件控制的循环运行方式**

（1）反复执行语句块，直到条件不满足时结束

|  |
| --- |
| while <条件>:  <语句块> |

（2）无限循环的条件

|  |
| --- |
| >>> a = 3  >>> while a > 0:  a = a - 1  print(a)  2  1  0 |

**3****. 循环控制保留字：break和continue**

* break：跳出并结束当前整个循环，执行循环后的语句
* continue：结束当次循环，继续执行后续次数循环
* break和continue可以与for和while循环搭配使用

|  |  |
| --- | --- |
| >>> for c in "PYTHON":  if c == "T":  continue  print(c, end="")  PYHON | >>> for c in "PYTHON":  if c == "T":  break  print(c, end="")  PY |

* Break仅跳出当前最内层循环

|  |  |
| --- | --- |
| >>> s = "PYTHON"  >>> while s != "":  for c in s:  print(c, end="")  s = s[:-1]  PYTHONPYTHOPYTHPYTPYP | >>> s = "PYTHON"  >>> while s != "":  for c in s:  if c == "T":  break  print(c, end="")  s = s[:-1]  PYTHONPYTHOPYTHPYTPYP |

**4. 循环的高级用法：****循环与else**

|  |  |
| --- | --- |
| for <循环变量> in <遍历结构>:  <语句块1>  else:  <语句块2> | for <条件>:  <语句块1>  else:  <语句块2> |

* 当循环没有被break语句退出时，执行else语句块
* else语句块作为“正常”完成循环的奖励

|  |  |
| --- | --- |
| >>> for c in "PYTHON":  if c == "T":  continue  print(c, end="")  else:  print("正常退出")  PYHON正常退出 | >>> for c in "PYTHON":  if c == "T":  break  print(c, end="")  else:  print("正常退出")  PY |

**4.4 模板3：****random库的使用**

**1. random库基本介绍**

（1）random库是使用随机数的Python标准库

* 伪随机数：采用梅森旋转算法生成的（伪）随机序列中元素
* random()库主要用于生成随机数
* 使用random()库：import random

（2）random()库包括两类函数，常用共8个

* 基本随机数函数：seed()，random()
* 扩展随机数函数：randint()，getrandbits()，uniform()，randrange()，choice()，shuffle()

**2. 基本随机数函数**

（1）随机数种子

* 随机数种子→梅森旋转算法→随机序列（→随机数）

|  |  |  |  |
| --- | --- | --- | --- |
| 函数 | 描述 | | >>> import random |
| seed(a = None) | 初始化给定的随机数种子，默认为当前系统时间 | | >>> random.seed(10) #产生种子10对应的序列 |
| random() | 生成一个[0.0, 1.0)之间的随机小数 | | >>> random.random()  0.5714025946899135  >>> random.random()  0.4288890546751146  `````` |
| >>> import random  >>> random.seed(10)  >>> random.random()  0.5714025946899135  >>> random.random()  0.4288890546751146  `````` | | >>> import random  >>> random.seed(10)  >>> random.random()  0.5714025946899135  >>> random.seed(10)  >>> random.random()  0.5714025946899135 | |

**3. 扩展随机数函数**

|  |  |  |
| --- | --- | --- |
| 函数 | 描述 | >>> import random |
| randint(a, b) | 生成一个[a, b]之间的整数 | >>> random.randint(10,100)  83 |
| randrange(m, n[, k]) | 生成一个[m, n]之间以k为步长的随机整数 | >>> random.randrange(10,100,10)  10 |
| getrandbits(k) | 生成一个k比特长的随机整数 | >>> random.getrandbits(16)  13506 |
| uniform(a, b) | 生成一个[a, b]之间的随机小数 | >>> random.uniform(10,100)  51.632245728365426 |
| choice(seq) | 从序列seq中随机选择一个元素 | >>> random.choice([1,2,3,4,5,6,7,8,9])  8 |
| shuffle(seq) | 将序列seq中元素随机排列  返回打乱后的序列 | >>> s = [1, 2, 3, 4, 5, 6, 7, 8, 9]  >>> random.shuffle(s)  >>> print(s)  [2, 6, 7, 8, 4, 9, 1, 3, 5] |

* 需要掌握的能力
* 能够利用随机数种子产生“确定”伪随机数
* 能够产生随机整数
* 能够对序列类型进行随即操作

**4.5 实例6：圆周率的计算**

**1. “圆周率的计算”问题分析**

（1）圆周率的近似计算公式

（2）蒙特卡罗方法

**2. “圆周率的计算”实例讲解**

（1）圆周率的近似计算公式

|  |
| --- |
| #CalPiV1.py  pi = 0  N = 100  for k in range(N):  pi += 1/pow(16,k)\*( \  4/(8\*k+1) - 2/(8\*k+4) - \  1/(8\*k+5) - 1/(8\*k+6))  print("圆周率值是：{}".format(pi)) |

* （运行结果）圆周率值是：3.141592653589793

（2）蒙特卡罗方法

|  |  |
| --- | --- |
| #CalPiV2.py  from random import random  from time import perf\_counter  DARTS = 1000\*1000  hits = 0.0  start = perf\_counter()  for i in range(1, DARTS+1):  x, y = random(), random()  dist = pow(x \*\* 2 + y \*\* 2, 0.5)  if dist <= 1.0:  hits = hits + 1  pi = 4 \* (hits/DARTS)  print("圆周率值是：{}".format(pi))  print("运行时间是：{:.5f}s".format(perf\_counter()-start)) |  |

* （运行结果）圆周率值是：3.139964；运行时间是：0.71169s

**3. “圆周率的计算”举一反三**

（1）理解方法思维

* 数学思维：找到公式，利用公式求解
* 计算思维：抽象一种过程，用计算机自动化求解
* 谁更准确？

（2）程序运行时间分析

* 使用time库的计时方法获得程序运行时间
* 改变撒点数量，理解程序运行时间的分布
* 初步掌握简单的程序性能分析方法

（3）计算问题扩展

* 不求解圆周率，而是某个特定图形的面积
* 在工程计算中寻找蒙特卡罗方法的应用场景

**5.1 函数的定义与使用**

**1. 函数的理解和定义**

（1）函数的定义：函数是一段代码的表示

* 函数是一段具有特定功能的、可重用的语句组
* 函数是一种功能的抽象，一般函数表达特定功能
* 两个作用：降低编程编程难度和代码复用

|  |
| --- |
| def <函数名>(<参数(0个或多个)>)  <函数体>  return <返回值> |

（2）案例：计算n！

|  |
| --- |
| def fact(n):  s = 1  for i in range(1, n+1):  s \*= i  return s |

（3）y = f(x)

* 函数定义时，所指定的参数是一种占位符
* 函数定义后，如果不经过调用，不会被执行
* 函数定义时，参数是输入、函数体式处理、结果是输出(IPO)

**2. 函数的使用和调用过程**

（1）调用是运行函数代码的方式

* 调用时要给出实际参数
* 实际参数替换定义中的参数
* 函数调用后得到返回值

|  |  |
| --- | --- |
| def fact(n):  s = 1  for i in range(1, n+1):  s \*= i  return s | a = fact(10)  print(a)  3628800 |

**3. 函数的参数传递**

（1）参数个数：函数可以有参数，也可以没有，但必须保留括号

|  |
| --- |
| def <函数名>():  <函数体>  return <返回值>  def fact():  print("我也是函数") |

（2）可选参数传递：函数定义时可以为某些参数指定默认值，构成可选参数

|  |
| --- |
| def <函数名>(<非可选参数>, <可选参数>):  <函数体>  return <返回值> |

（3）可选参数传递：计算n!//m

|  |  |
| --- | --- |
| def fact(n, m=1):  s = 1  for i in range(1, n+1):  s \*= i  return s//m | >>> fact(10)  3628800  >>> fact(10, 5)  725760 |

（4）可变参数传递：函数定义时可以设计可变数量参数，即不确定参数总数量

|  |
| --- |
| def <函数名>(<参数>, \*b):  <函数体>  return <返回值> |

（5）可变参数传递：计算n!乘数

|  |  |
| --- | --- |
| def fact(n, \*b):  s = 1  for i in range(1, n+1):  s \*= 1  for item in b:  s \*= item  return s | >>> fact(10, 3)  10886400  >>> fact(10, 3, 5, 8)  435456000 |

（6）参数传递的两种方式：函数调用时，参数可以按照位置或名称方式传递

|  |  |
| --- | --- |
| def fact(n, m=1):  s = 1  for I in range(1, n+1):  s \*= i  return s//m | >>> fact(10, 5)  725760 |

**4. 函数的返回值**

（1）函数可以返回0个或多个结果

* return保留字用来传递返回值
* 函数可以有返回值，也可以没有，可以有return，也可以没有
* return可以传递0个返回值，也可以传递任意多个返回值

|  |  |
| --- | --- |
| def fact(n, m=1):  s = 1  for I in range(1, n+1):  s \*= i  return s//m, n, m | >>> fact(10, 5)  (725760, 10, 5)  >>> a,b,c = fact(10,5)  >>> print(a,b,c)  725760 10 5 |

**5. 局部变量和全局变量**

|  |  |
| --- | --- |
| n, s = 10, 100  def fact(n):  s = 1  for i in range(1, n+1):  s \*= i  return s  print(fact(n), s) | >>>  3628800 100 |

（1）规则1：局部变量和全局变量是不同变量

* 局部变量是函数内部的占位符，与全局变量可能重名但不同
* 函数运算结束后，局部变量被释放
* 可以使用global保留字在函数内部使用全局变量

|  |  |
| --- | --- |
| n, s = 10, 100  def fact(n):  global s  for i in range(1, n+1):  s \*= i  return s  print(fact(n), s) | >>>  362880000 362880000 |

（2）规则2：局部变量为组合数据类型且未创建，等同于全局变量

|  |  |
| --- | --- |
| ls = ["F", "f"]  def func(a):  ls.append(a)  return  func("C")  print(ls)  >>>  ['F', 'f', 'C'] | ls = ["F", "f"]  def func(a):  ls = []  ls.append(a)  return  func("C")  print(ls)  >>>  ['F', 'f'] |

（3）使用规则

* 基本数据类型，无论是否重名，局部变量与全局变量不同
* 可以通过global保留字在函数内部声明全局变量
* 组合数据类型，如果局部变量未真实创建，则为全局变量

**6. lambda函数：lambda函数返回函数名作为结果**

* lambda函数是一种匿名函数，即没有名字的函数
* 使用lambda保留字定义，函数名是返回结果
* lambda函数用于定义简单的、能够在一行内表示的函数

|  |  |
| --- | --- |
| <函数名> = lambda <参数>:<表达式> | def <函数名>(<参数>):  <函数体>  return <返回值> |
| >>> f = lambda x, y: x + y  >>> f(10, 15)  25 | >>> f = lambda: "lambda函数"  >>> print(f())  lambda函数 |

**5.2 实例7：七段数码管绘制**

**1. “七段数码管绘制”问题分析**

* 用程序绘制七段数码管→turtle绘图体系→七段数码管绘制

**2. “七段数码管绘制”实例讲解（上）**

（0）基本思路

* 步骤1：绘制单个数字对应的数码管
* 步骤2：获得一串数字，绘制对应的数码管
* 步骤3：获得当前系统时间，绘制对应的数码管

（1）步骤1：绘制单个数字对应的数码管

* 七段数码管由7个基本线条组成
* 七段数码管可以有固定顺序
* 不同数字显示不同的线条

|  |
| --- |
| import turtle  def drawLine(draw): #绘制单段数码管  turtle.pendown() if draw else turtle.penup()  turtle.fd(40)  turtle.right(90)  def drawDigit(digit): #根据数字绘制七段数码管  drawLine(True) if digit in [2,3,4,5,6,8,9] else drawLine(False)  drawLine(True) if digit in [0,1,3,4,5,6,7,8,9] else drawLine(False)  drawLine(True) if digit in [0,2,3,5,6,8,9] else drawLine(False)  drawLine(True) if digit in [0,2,6,8] else drawLine(False)  turtle.left(90)  drawLine(True) if digit in [0,4,5,6,8,9] else drawLine(False)  drawLine(True) if digit in [0,2,3,5,6,7,8,9] else drawLine(False)  drawLine(True) if digit in [0,1,2,3,4,7,8,9] else drawLine(False)  turtle.left(180)  turtle.penup() #为绘制后续数字确定位置  turtle.fd(20) #为绘制后续数字确定位置  …… |

（2）步骤2：获取一段数字，绘制多个数码管

|  |
| --- |
| ……  def drawDate(date): #获得要输入的数字  for i in date:  drawDigit(eval(i)) #通过eval()函数将数字变为整数  def main():  turtle.setip(800, 350, 200, 200)  turtle,penup()  turtle.fd(-300)  turtle.pensize(5)  drawDate('20181010')  turtle.hideturtle()  turtle.done()  main() |

**3. “七段数码管绘制”实例讲解（下）**

（1）获得当前系统时间，绘制对应的数码管

|  |
| --- |
| import turtle  def drawGap(): #绘制数码管间隔  turtle.pemup()  turtle.fd(5)  def drawLine(draw): #绘制单段数码管  drawGap()  turtle.pendown() if draw else turtle.penup()  turtle.fd(40)  drawGap()  turtle.right(90)  def drawDigit(digit): #根据数字绘制七段数码管 |

* 使用time库获得系统当前时间
* 增加年月日标记
* 年月日颜色不同

|  |
| --- |
| import turtle, time    def drawDate(date): #data为日期，格式为'%Y-%m=%d+'  turtle.pencolor("red")  for i in date:  if i == '-':  turtle.write('年',font=("Arial",18,"normal"))  turtle.pencolor("green")  turtle.fd(40)  elif i == '=':  turtle.write('月',font=("Arial",18,"normal"))  turtle.pencolor("blue")  turtle.fd(40)  elif i == '+':  turtle.write('年',font=("Arial",18,"normal"))  else:  drawDigit(eval(i))  def main():  turtle.setup(800, 350, 200, 200)  turtle.penup()  turtle.fd(-300)  turtle.pensize(5)  drawDate(time.strftime('%Y-%m=%d+',time.gmtime()))  turtle.hideturtle()  turtle.done()  main() |

（2）最终结果

|  |
| --- |
| import turtle, time  def drawGap(): #绘制数码管间隔  turtle.penup()  turtle.fd(5)  def drawLine(draw): #绘制单段数码管  drawGap()  turtle.pendown() if draw else turtle.penup()  turtle.fd(40)  drawGap()  turtle.right(90)  def drawDigit(digit): #根据数字绘制七段数码管  drawLine(True) if digit in [2,3,4,5,6,8,9] else drawLine(False)  drawLine(True) if digit in [0,1,3,4,5,6,7,8,9] else drawLine(False)  drawLine(True) if digit in [0,2,3,5,6,8,9] else drawLine(False)  drawLine(True) if digit in [0,2,6,8] else drawLine(False)  turtle.left(90)  drawLine(True) if digit in [0,4,5,6,8,9] else drawLine(False)  drawLine(True) if digit in [0,2,3,5,6,7,8,9] else drawLine(False)  drawLine(True) if digit in [0,1,2,3,4,7,8,9] else drawLine(False)  turtle.left(180)  turtle.penup() #为绘制后续数字确定位置  turtle.fd(20) #为绘制后续数字确定位置  def drawDate(date): #data为日期，格式为'%Y-%m=%d+'  turtle.pencolor("red")  for i in date:  if i == '-':  turtle.write('年',font=("Arial",18,"normal"))  turtle.pencolor("green")  turtle.fd(40)  elif i == '=':  turtle.write('月',font=("Arial",18,"normal"))  turtle.pencolor("blue")  turtle.fd(40)  elif i == '+':  turtle.write('日',font=("Arial",18,"normal"))  else:  drawDigit(eval(i))  def main():  turtle.setup(800, 350, 200, 200)  turtle.penup()  turtle.fd(-300)  turtle.pensize(5)  drawDate(time.strftime('%Y-%m=%d+',time.gmtime()))  turtle.hideturtle()  turtle.done()  main() |

**4. “七段数码管绘制”举一反三**

（1）理解方法思维

* 模块化思维：确定模块化接口，封装功能
* 规则化思维：抽象过程为规则，计算机自动执行
* 化繁为简：将大功能变为小功能组合，分而治之

（2）应用问题的扩展

* 绘制带小数点的七段数码管
* 带刷新的时间倒计时效果
* 绘制高级的数码管

**5.3 代码复用与函数递归**

**1. 代码复用与模块化方法**

（1）把代码当成资源进行抽象

* 代码资源化：程序代码是一种用来表达计算的“资源”
* 代码抽象化：使用函数等方法对代码赋予更高级别的定义
* 代码复用：同一份代码在需要时可以被重复使用

（2）函数和对象是代码复用的两种主要形式

* 函数：将代码命名在代码层面建立了初步抽象
* 对象：属性和方法 <a>.<b>和<a>.<b>()在函数之上再次进行抽象

（3）分而治之

* 通过函数或对象封装将程序划分为模块及模块间的表达
* 具体包括：主程序、子程序和子程序间的关系
* 分而治之：一种分而治之、分层抽象、体系化的设计思想

（4）紧耦合 松耦合

* 紧耦合：两个部分之间交流很多，无法独立存在
* 松耦合：两个部分之间交流较少，可以独立存在
* 模块内部紧耦合、模块之间松耦合

**2. 函数递归的理解**

（1）函数递归的定义：函数定义中调用自身的方式

（2）函数递归的两个关键特征：

* 链条：计算过程存在递归链条
* 基例：存在一个或多个不需要再次递归的基例

（3）函数递归的实现：函数+分支结构

（4）类似数学归纳法

* 证明当n取第一个值n0时命题成立
* 假设当nk时命题成立，证明当n=nk+1时命题成立
* 递归是数学归纳法思维的编程体现

**3. 函数递归的调用过程**

|  |  |
| --- | --- |
|  | def fact(n):  if n == 0:  return 1  else:  return n\*fact(n-1) |

（1）递归的实现：函数+分支语句

* 递归本身是一个函数，需要函数定义方式描述
* 函数内部，采用分支语句对输入参数进行判断
* 基例和链条，分别编写对应代码

**4. 函数递归实例解析**

（1）字符串反转：将字符串s反转后输出

|  |
| --- |
| >>> s[::-1] |
| def rvs(s):  if s == '':  return s  else:  return rvs(s[1:]+s[0]) |

（2）斐波那契数列：一个经典数列

|  |  |
| --- | --- |
|  | def f(n):  if n ==1 or n ==2:  return 1  else:  return f(n-1) + f(n-2) |

（3）汉诺塔

|  |
| --- |
| count = 0  def hanoi(n, src, dst, mid):  global count  if n == 1:  print("{}:{}->{}".format(1, src, dst))  count += 1  else:  hanoi(n-1, src, dst, mid)  print("{}:{}->{}".format(n, src, dst))  count += 1  hanoi(n-1, mid, dst, src) |

**5.4 模块4：PyInstaller库的使用**

**1. PyInstaller库基本介绍**

（1）将.py源代码转换成无需源代码的可执行文件

（2）PyInstaller库是第三方库

* 官方网站：<http://www.pyinstaller.org>
* 第三方库：使用前需要额外安装
* 安装第三方库需要使用pip工具

（3）(cmd命令行)pip install pyinstaller

**2. PyInstaller库使用说明**

（1）(cmd命令)pyinstaller -F SevenDigitsDraw.py

（2）PyInstaller库常用参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| -h | 查看帮助 |
| --clean | 清理打包过程中的临时文件 |
| -D, --onedir | 默认值，生成dist文件夹 |
| -F, --onefile | 在dist文件夹中只生成独立的打包文件 |
| -i <图标文件名.ico> | 指定打包程序使用的图标(icon)文件 |

（3）使用举例

(cmd命令)pyinstaller -i curve.ico -F SevenDigitsDraw.py

**5.5 实例8：科赫雪花小包裹**

**1. “科赫雪花小包裹”问题分析**

（1）高大上的分形几何：分形几何是一种迭代的几何图形，广泛存在于自然界中

（2）科赫曲线，也叫雪花曲线

（3）用Python绘制科赫曲线

**2. “科赫雪花小包裹”实例讲解（上）**

（1）科赫曲线的绘制：递归+turtle库

|  |  |
| --- | --- |
| #KochDrawV1.py  import turtle  def koch(size, n):  if n == 0:  turtle.fd(size)  else:  for angle in [0, 60, -120, 60]:  turtle.left(angle)  koch(size/3, n-1)  def main():  turtle.setup(800, 400)  turtle.penup()  turtle.goto(-300, -50)  turtle.pendown()  turtle.pensize(2)  koch(600, 3) #3阶科赫曲线，阶数  turtle.hideturtle()  main() | #KochDrawV2.py  import turtle  def koch(size, n):  if n == 0:  turtle.fd(size)  else:  for angle in [0, 60, -120, 60]:  turtle.left(angle)  koch(size/3, n-1)  def main():  turtle.setup(600, 600)  turtle.penup()  turtle.goto(-200, 100)  turtle.pendown()  turtle.pensize(2)  level = 3 #3阶科赫曲线，阶数  koch(400, level)  turtle.right(120)  koch(400, level)  turtle.right(120)  koch(400, level)  turtle.hideturtle()  main() |

**3. “科赫雪花小包裹”实例讲解（下）**

（1）(cmd命令)pyinstaller -i curve.ico -F KochDrawV2.py

**4. “科赫雪花小包裹”举一反三**

（1）绘制条件的扩展

* 修改分形几何绘制阶数
* 修改科赫曲线的基本定义及旋转角度
* 修改绘制科赫雪花的基础框架图形

（2）分形几何千千万

* 康托尔集、谢尔宾斯基三角形、门格海绵…
* 龙形曲线、空间填充曲线、科赫曲线

**6.1 集合类型及操作**

**1. 集合类型定义**

（1）集合是多个元素的无序组合

* 集合类型与数学中的集合概念一致
* 集合元素之间无序，每个元素唯一，不存在相同元素
* 集合元素不可更改，不能是可变数据类型——为什么？
* 集合用大括号{}表示，元素间用逗号分隔
* 建立集合类型用{}或set()
* 建立空集合类型，必须使用set()

|  |
| --- |
| >>> A = {"Python", 123, ("Python", 123)} #使用{}建立集合  >>> A  {'Python', 123, ('Python', 123)}  >>> B = set("pypy123") #使用set()建立集合  >>> B  {'3', 'p', '2', '1', 'y'}  >>> C = {"python", 123, "python", 123}  >>> C  {123, 'python'} |

* 集合中每个元素 唯一，不存在相同元素
* 集合元素之间无序

**2. 集合操作符：S|T\_并；S-T\_差；S&T\_交；s^T\_补**

（1）6个操作符

|  |  |
| --- | --- |
| 操作符及应用 | 描述 |
| S|T | 返回一个新集合，包括在集合S和T中的所有元素 |
| S-T | 返回一个新集合，包括在集合S但不在T中的元素 |
| S&T | 返回一个新集合，包括同时在集合S和T中的元素 |
| S^T | 返回一个新集合，包括集合S和T中的非相同元素 |
| S<=T或S<T | 返回True/False，判断S和T的子集关系 |
| S>=T或S>T | 返回True/False，判断S和T的包含关系 |

（2）4个增强操作符

|  |  |
| --- | --- |
| 操作符及应用 | 描述 |
| S|=T | 更新集合S，包括在集合S和T中的所有元素 |
| S-=T | 更新集合S，包括在集合S但不在T中的元素 |
| S&=T | 更新集合S，包括同时在集合S和T中的元素 |
| S^=T | 更新集合S，包括集合S和T中的非相同元素 |

（3）举例

|  |
| --- |
| >>> A = {"p", "y", 123}  >>> B = set("pypy123")  >>> A - B  {123}  >>> B -A  {'2', '3', '1'}  >>> A&B  {'p', 'y'}  >>> A|B  {'2', '3', 'p', 123, '1', 'y'}  >>> A^B  {123, '2', '3', '1'} |

**3. 集合处理方法**

|  |  |
| --- | --- |
| 操作符或方法 | 描述 |
| S.add(x) | 如果x不在集合S中，将x增加到S |
| S.discard(x) | 移除S中元素x，如果x不在集合S中，不报错 |
| S.remove(x) | 移除S中元素x，如果x不在集合S中，产生KeyError异常 |
| S.clear() | 移除S中所有元素 |
| S.pop() | 随机返回S的一个元素，更新S，若S为空，产生KeyError异常 |
| S.copy() | 返回集合S的一个副本 |
| len(S) | 返回集合S的元素个数 |
| x in S | 判断S中元素x，x在集合S中，返回True，否则返回False |
| x not in S | 判断S中元素x，x不在集合S中，返回True，否则返回False |
| set(x) | 将其他类型变量x转变为集合类型 |

举例：

|  |
| --- |
| >>> A = {"p", "y", 123}  >>> for item in A:  print(item, end="")  123py  >>> A  {123, 'p', 'y'} |
| >>> try:  while True:  print(A.pop(), end="")  except:  pass  123py  >>> A  set() |

**4. 集合类型应用场景**

（1）包含关系比较

|  |
| --- |
| >>> "p" in {"p", "y", 123}  True  >>> {"p", "y"} >= {"p", "y", 123}  False |

（2）数据去重：集合类型所有元素无重复

|  |
| --- |
| >>> ls = ["p", "p", "y", "y", 123]  >>> s = set(ls) #将列表转换为集合  >>> s #利用了集合无重复元素的特点  {123, 'p', 'y'}  >>> lt = list(s) #将集合转换为列表  >>> lt  [123, 'p', 'y'] |

**6.2 序列类型及操作**

**1. 序列类型定义**

（1）序列是具有先后关系的一组元素

* 序列是一维向量元素，元素类型可以不同
* 类似数学元素序列：*s0, s1, …, sn-1*
* 元素间由序号引导，通过下标访问序列的特定元素

（2）序列是一个基类类型

* 序列类型：①字符串类型；②元组类型；③列表类型

（3）序号的定义

**2. 序列处理函数及方法**

（1）6个操作符

|  |  |
| --- | --- |
| 操作符及应用 | 描述 |
| x in s | 如果x是序列s的元素，返回True，否则返回False |
| x not in s | 如果x是序列s的元素，返回False，否则返回False |
| s + t | 连接两个序列s和t |
| s\*n或n\*s | 将序列s复制n次 |
| s[i] | 索引，返回s中的第i个元素，i是序列的序号 |
| s[i:j]或s[i:j:k] | 切片，返回序列s中第i到j以k为步长的元素子序列 |

（2）序列类型操作实例

|  |
| --- |
| >>> ls = ["python", 123, ".io"]  >>> ls[::-1]  ['.io', 123, 'python']  >>> s = "python123.io"  >>> ls[::-1]  ['.io', 123, 'python'] |

（3）5个函数和方法

|  |  |
| --- | --- |
| 操作符及应用 | 描述 |
| len(s) | 返回序列s的长度 |
| min(s) | 返回序列s的最小元素，s中元素需要可比较 |
| max(s) | 返回序列s的最大元素，s中元素需要可比较 |
| s.index(x)或s.inddex(x, i , j) | 返回序列s从i开始到j位置中第一次出现元素x的位置 |
| s.count(x) | 返回序列s中出现x的总次数 |

举例：

|  |
| --- |
| >>> ls = ["python", 123, ".io"]  >>> len(ls)  3  >>> s = "python123.io"  >>> max(s)  'y' |

**3. 元组类型及操作**

（1）元组是序列类型的一种扩展

* 元组是一种序列类型，一旦创建就不能被修改
* 使用小括号()或tuple()创建，元素间用逗号,分隔
* 可以使用或不使用小括号

|  |
| --- |
| def func():  return 1, 2 |

举例：

|  |
| --- |
| >>> creature = "cat", "dog", "tiger", "human"  >>> creature  ('cat', 'dog', 'tiger', 'human')  >>> color = (0x001100, "blue", creature)  >>> color  (4352, 'blue', ('cat', 'dog', 'tiger', 'human')) |

（2）元组继承序列类型的全部通用操作

* 元组继承了序列类型的全部通用操作
* 元组因为创建后不能修改，因此没有特殊操作
* 使用或不使用小括号

|  |
| --- |
| >>> creature = "cat", "dog", "tiger", "human"  >>> creature[::-1]  ('human', 'tiger', 'dog', 'cat')  >>> color = (0x001100, "blue", creature)  >>> color[-1][2]  'tiger' |

**4. 列表类型及操作**

（1）列表是序列类型的一种扩展，十分常用

* 列表是一种序列类型，创建后可以随意被修改
* 使用方括号[]或list()创建，元素间用逗号,分隔
* 列表中各元素类型可以不同，无长度限制

|  |
| --- |
| >>> ls = ["cat", "dog", "tiger", 1024]  >>> ls  ['cat', 'dog', 'tiger', 1024]  >>> lt = ls #方括号[]真正创建一个列表，赋值仅传递引用  >>> lt  ['cat', 'dog', 'tiger', 1024] |

（2）列表类型操作函数和方法

|  |  |
| --- | --- |
| 函数或方法 | 描述 |
| ls[i] = x | 替换列表ls第i元素为x |
| ls[i:j:k] = lt | 用列表lt替换ls切片后所对应元素子列表 |
| del ls[i] | 删除列表ls中第i元素 |
| del ls[i:j:k] | 删除列表ls中第i到第j以k为步长的元素 |
| ls += lt | 更新列表ls，将列表lt元素增加到列表ls中 |
| ls \*= n | 更新列表ls，其元素重复n次 |

举例：

|  |
| --- |
| >>> ls = ["cat", "dog", "tiger", 1024]  >>> ls[1:2] = [1, 2, 3, 4]  >>> ls  ['cat', 1, 2, 3, 4, 'tiger', 1024]  >>> del ls[::3]  >>> ls  [1, 2, 4, 'tiger']  >>> ls \* 2  [1, 2, 4, 'tiger', 1, 2, 4, 'tiger'] |

（3）列表类型操作函数和方法：

|  |  |
| --- | --- |
| 函数或方法 | 描述 |
| ls.append(x) | 在列表ls最后增加一个元素x |
| ls.clear() | 删除列表ls中所有元素 |
| ls.copy() | 生成一个新列表，赋值ls中所有元素 |
| ls.insert(I,x) | 在列表ls的第i位置增加增加元素x |
| ls.pop(i) | 在列表ls的第i位置取出并删除该元素 |
| ls.remove(x) | 将列表ls中出现的第一个元素x删除 |

举例：

|  |
| --- |
| >>> ls = ["cat", "dog", "tiger", 1024]  >>> ls.append(1234)  >>> ls  ['cat', 'dog', 'tiger', 1024, 1234]  >>> ls.insert(3, "human")  >>> ls  ['cat', 'dog', 'tiger', 'human', 1024, 1234]  >>> ls.reverse()  >>> ls  [1234, 1024, 'human', 'tiger', 'dog', 'cat'] |

（4）练习

|  |  |  |
| --- | --- | --- |
| * 定义空列表lt * 向lt新增5个元素 * 修改lt中第2个元素 * 向lt中第2个位置增加一个元素 * 向lt中第1个位置删除一个元素 * 删除lt中第1-3位置元素 * 判断lt中是否包含数字0 * 向lt新增数字0 * 返回数字0所在lt中的索引 * lt的长度 * lt中最大元素 * 清空lt | >>> lt = []  >>> lt += [1, 2, 3, 4, 5]  >>> lt[2] = 6  >>> lt.insert(2, 7)  >>> del lt[1]  >>> del lt[1:4]  >>> 0 in lt  >>> lt.append(0)  >>> lt.index(0)  >>> len(lt)  >>> max(lt)  >>> lt.clear() | False  2  3  5 |

**5. 序列类型应用场景**

（1）序列类型应用场景

* 元组用于元素不改变的应用场景，更多用于固定搭配场景
* 列表更加灵活，它是最常用的序列类型
* 最主要作用：表示一组有序数据，进而操作它们

（2）元素遍历

|  |
| --- |
| for item in ls:  <语句块>  for item in tp:  <语句块> |

（3）数据保护

* 如果不希望数据被程序改变，转换成元组类型

|  |
| --- |
| >>> ls = ["cat", "dog", "tiger", 1024]  >>> lt = tuple(ls)  >>> lt  ('cat', 'dog', 'tiger', 1024) |

**6.3 实例9：基本统计值计算**

**1. “基本统计值计算”问题分析**

（1）基本统计值

* 需求：给出一组数，对它们有个概要理解
* 该怎么做？
* 总个数：len()
* 求和：for…in…
* 平均值：求和/总个数
* 方差：各数据与平均数差的平方的和的平均数
* 中位数：排序，然后…奇数找中间1个，偶数找中间2个取平均

**2. “基本统计值计算”实例讲解**

（1）最终代码

|  |
| --- |
| #CalStatisticsV1.py  def getNum():  nums = []  iNumStr = input("请输入数字(回车退出):")  while iNumStr != "":  nums.append(eval(iNumStr))  iNumStr = input("请输入数字(回车退出)：")  return nums  def mean(numbers): #计算平均值  s = 0.0  for num in numbers:  s = s + num  return s / len(numbers)  def dev(numbers, mean): #计算方差  sdev = 0.0  for num in numbers:  sdev = sdev + (num - mean)\*\*2  return pow(sdev / (len(numbers)-1), 0.5)  def median(numbers): #计算中位数  sorted(numbers)  size = len(numbers)  if size % 2 == 0:  med = (numbers[size//2-1] + numbers[size//2])/2  else:  med = numbers[size//2]  return med  n = getNum()  m = mean(n)  print("平均值：{},方差：{:.2},中位数：{}.".format(m, dev(n,m), median(n))) |

**3. “基本统计值计算”举一反三**

（1）技术能力扩展

获取多个数据：从控制台获取多个不确定数据的方法

分隔多个函数：模块化设计方法

充分利用函数：充分利用Python提供的内容函数

**6.4 字典类型及操作**

**1. 字典类型定义**

（1）理解“映射”

映射是一种键（索引）和值（数据）的对应

（2）字典类型是“映射”的体现

* 键值对：键是数据索引的扩展
* 字典是键值对的集合，键值对之间无序
* 采用大括号{}和dict()创建，键值对用冒号:表示

{<键1>:<值1>, <键2>:<值2>, …, <键n>:<值n>}

（3）在字典变量中，通过键获得值

<字典变量> {<键1>:<值1>, <键2>:<值2>, …, <键n>:<值n>}

<值> = <字典变量>[<键>]

<字典变量>[<键>] = <值>

[ ]用来向字典变量中索引或增加元素

|  |
| --- |
| >>> d = {"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}  >>> d  {'中国': '北京', '美国': '华盛顿', '法国': '巴黎'}  >>> d["中国"]  '北京'  >>> de = {} ; type(de) #type()返回变量x的类型  <class 'dict'> |

**2. 字典处理函数及方法**

（1）字典类型操作函数和方法

|  |  |
| --- | --- |
| 函数或方法 | 描述 |
| del d[k] | 删除字典d中键k对应的数据值 |
| k in d | 判断键k是否在字典d中，如果在返回True，否则返回False |
| d.keys(0 | 返回字典d中所有的键信息 |
| d.values() | 返回字典d中所有的值信息 |
| d.items() | 返回字典d中所有的键值对信息 |
| d.get(k, <default>) | 键k存在，则返回相应值，不在则返回<default>值 |
| d.pop(k, <default>) | 键k存在，则取出相应值，不在则返回<default>值 |
| d.popitem() | 随机从字典d中取出一个键值对，以元组形式返回 |
| d.clear() | 删除所有的键值对 |
| len(d) | 返回字典d中元素的个数 |

（2）举例

|  |
| --- |
| >>> d = {"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}  >>> "中国" in d  True  >>> d.keys()  dict\_keys(['中国', '美国', '法国'])  >>> d.values()  dict\_values(['北京', '华盛顿', '巴黎'])  >>> d.get("中国", "伊斯兰堡")  '北京'  >>> d.get("巴基斯坦", "伊斯兰堡")  '伊斯兰堡'  >>> d.popitem()  ('法国', '巴黎') |

（3）练习

|  |  |  |
| --- | --- | --- |
| * 定义空字典d * 向d新增2个键值对元素 * 修改第2个元素 * 判断字符"c"是否是d的键 * 计算d的长度 * 清空d | >>> d = {}  >>> d["a"] = 1; d["b"] = 2  >>> d["b"] = 3  >>> "c" in d  >>> len(d)  >>> d.clear() | False  2 |

**3. 字典类型应用场景**

（1）映射的表达

* 映射无处不在，键值对无处不在
* 例如：统计数据出现的次数，数据是键，次数是值
* 最主要的作用：表达键值对数据，进而操作它们

（2）元素遍历

|  |
| --- |
| for k in d:  <语句块> |

**6.5 模块5：jieba库的使用**

**1. jieba库基本介绍**

（1）jieba是优秀的中文分词第三方库

* 中文文本需要通过分词获得单个的词语
* jieba是优秀的中文分词第三方库，需要额外安装
* jiaba库提供三种分词模式，最简单只需掌握一个函数

（2）jieba库的安装

(cmd命令行) pip install jieba

（3）jieba分词的原理：依靠中文词库

* 利用一个中文词库，确定汉字之间的关联概率
* 汉字间概率大的组成词组，形成分词结果
* 除了分词，用户还可以添加自定义的词组

**2. jieba库使用说明**

（1）jieba分词的三种模式

* 精确模式：把文本精确地切分开，不存在冗余单词
* 全模式：把文本中所有可能的词语都扫描出来，有冗余
* 搜索引擎模式：在精确模式基础上，对分词再次切分

（2）jiaba库常用函数

|  |  |  |
| --- | --- | --- |
| 函数 | 描述 | |
| jieba,lcut(s) | 精确模式  返回一个列表类型的分词结果 | >>> jieba.lcut("中国是一个伟大的国家")  ['中国', '是', '一个', '伟大', '的', '国家'] |
| jieba.lcut(s, cut\_all=true) | 全模式  返回一个列表类型的分词结果  （存在冗余） | >>> jieba.lcut("中国是一个伟大的国家",cut\_all=True)  ['中国', '国是', '一个', '伟大', '的', '国家'] |
| jieba.lcut\_for\_search(s) | 搜索引擎模式  返回一个列表类型的分词结果  （存在冗余） | >>> jieba.lcut\_for\_search("中华人民共和国是伟大的")  ['中华', '华人', '人民', '共和', '共和国', '中华人民共和国', '是', '伟大', '的',] |
| jieba.add\_word(w) | 向分词词典增加新词w | >>> jieba.add\_word("蟒蛇语言") |

**6.6 实例10：文本词频统计**

**1. “文本词频统计”问题分析**

（1）文本词频统计

* 需求：一篇文章，出现了哪些词？哪些词出现得最多？
* 该怎么做？

（2）中文/英文

* 英文文本：Hamlet——分析词频

<https://python123.io/resources/pye/hamlet.txt>

* 中文文本：《三国演义》——分析人物

<https://python123.io/resources/pye/threekingdoms.txt>

**2. “hamlet英文词频统计”实例讲解**

（1）哈姆雷特(HAMLET)

|  |
| --- |
| #CalHamletV1.py  def getText():  txt = open("hamlet.txt", "r").read()  txt = txt.lower()  for ch in '!"#$%&()\*+,-./:;<=>?@[\\]^\_‘{|}~':  txt = txt.replace(ch, " ") #将文本中特殊字符替换为空格  return txt  hamletTxt = getText()  words = hamletTxt.split()  counts = {}  for word in words:  counts[word] = counts.get(word,0) + 1  items = list(counts.items())  items.sort(key=lambda x:x[1], reverse=True)  for i in range(10):  word, count = items[i]  print ("{0:<10}{1:>5}".format(word, count)) |

**3. “《三国演义》人物出场统计”实例讲解（上）**

（1）《三国演义》人物出场统计

|  |  |
| --- | --- |
| #CalThreeKingdomsV1.py  import jieba  txt = open("threekingdoms.txt", "r", encoding='utf-8').read()  words = jieba.lcut(txt)  counts = {}  for word in words:  if len(word) == 1:  continue  else:  counts[word] = counts.get(word,0) + 1  items = list(counts.items())  items.sort(key=lambda x:x[1], reverse=True)  for i in range(15):  word, count = items[i]  print ("{0:<10}{1:>5}".format(word, count)) | 曹操 953  孔明 836  将军 772  却说 656  玄德 585  关公 510  丞相 491  二人 469  不可 440  荆州 425  玄德曰 390  孔明曰 390  不能 384  如此 378  张飞 358 |

**4. “《三国演义》人物出场统计”实例讲解（下）**

（1）《三国演义》人物出场统计-优化

|  |  |
| --- | --- |
| #CalThreeKingdomsV2.py  import jieba  txt = open("threekingdoms.txt", "r", encoding='utf-8').read()  excludes = {"将军","却说","荆州","二人","不可","不能","如此",}  words = jieba.lcut(txt)  counts = {}  for word in words:  if len(word) == 1:  continue  elif word == "诸葛亮" or word == "孔明曰":  rword = "孔明"  elif word == "关公" or word == "云长":  rword = "关羽"  elif word == "玄德" or word == "玄德曰":  rword = "刘备"  elif word == "孟德" or word == "丞相":  rword = "曹操"  else:  rword = word  counts[rword] = counts.get(rword,0) + 1  for word in excludes:  del counts[word]  items = list(counts.items())  items.sort(key=lambda x:x[1], reverse=True)  for i in range(10):  word, count = items[i]  print ("{0:<10}{1:>5}".format(word, count)) | 曹操 1451  孔明 1383  刘备 1252  关羽 784  张飞 358  商议 344  如何 338  主公 331  军士 317  吕布 300 |

（2）《三国演义》人物出场顺序前20

|  |
| --- |
| 曹操、孔明、刘备、关羽、张飞、吕布、赵云、孙权、司马懿、周瑜、袁绍、马超、魏延、黄忠、姜维、马岱、庞德、孟获、刘表、夏侯惇 |

**5. “文本词频统计”举一反三**

（1）应用问题的扩展

* 《红楼梦》、《西游记》、《水浒传》···
* 政府工作报告、科研论文、新闻报道···
* 进一步呢？未来还有词云···

**7.1 文件的使用**

**1. 文件的类型**

（1）文件是数据的抽象和集合

* 文件是存储在辅助存储器上的数据序列
* 文件是数据存储的一种形式
* 文件展现形态：文本文件和二进制文件

（2）文本文件 vs. 二进制文件

* 文件文件和二进制文件只是文件的展示方式
* 本质上，所有文件都是二进制形式储存
* 形式上，所有文件采用两种方式展示

（3）文本文件：文件是数据的抽象和集合

* 由单一特定编码组成的文件，如UTF-8编码
* 由于存在编码，也被看成是存储着的长字符串
* 适用于例如：.txt文件、.py文件

（4）二进制文件：文件是数据的抽象和集合

* 直接由比特0和1组成，没有统一字符编码
* 一般存在二进制0和1的组织结构，即文件格式
* 适用于例如：.png文件、.avi文件

|  |  |
| --- | --- |
| #文本形式形式打开文件  tf = open("f.txt", "rt")  print(tf.readline())  tf.close()  >>>  中国是个伟大的国家！ | #二进制形式形式打开文件  bf = open("f.txt", "rb")  print(bf.readline())  bf.close()  >>>  b'\xd6\xd0\xb9\xfa\xca\xc7\xb8\xf6\xce\xb0\xb4\xf3\xb5\xc4\xb9\xfa\xbc\xd2\xa3\xa1' |

**2. 文件的打开和关闭**

（1）文件处理的步骤：打开-操作-关闭

|  |
| --- |
| a = opne( , ) |
| a.read(size)  a.readline(size)  a.readlines(hint) |
| a.write(s)  a.writelines(lines)  a.seek(offset) |
| a.close() |

（2）文件的打开：<变量名> = open(<文件名>, <打开模式>)

* <变量名>：文件句柄（如："D:\PYE\f.txt"）
* <文件名>：文件路径和名称，源文件同目录可省路径（如："D:/PYE/f.txt"；"D:\\PYE\\f.txt"；"./PYE/f.txt"；"f.txt"）
* <打开模式>：文本or二进制，读or写

|  |  |
| --- | --- |
| 'r' | 只读模式，默认值，如果文件不存在，返回FileNotFoundError |
| 'w' | 覆盖写模式，文件不存在则创建，存在则完全覆盖 |
| 'x' | 创建写模式，文件不存在则创建，存在则返回FileExistError |
| 'a' | 追加写模式，文件不存在则创建，存在则在文件最后追加内容 |
| 'b' | 二进制文件模式 |
| 't' | 文本文件模式，默认值 |
| '+' | 与r/w/x/a一同使用在原功能基础上增加同时读写功能 |
| f = open("f.txt') | 文本形式、只读模式、默认值 |
| f = open("f.txt', "rt") | 文本形式、只读模式、同默认值 |
| f = open("f.txt', "w") | 文本形式、覆盖写模式 |
| f = open("f.txt', "a+") | 文本形式、追加写模式+读文件 |
| f = open("f.txt', "x") | 文本形式、创建写模式 |
| f = open("f.txt', "b") | 二进制形式、只读模式 |
| f = open("f.txt', "wb") | 二进制形式、覆盖写模式 |