《Python程序设计基础》

第一单元 打开编程的大门

1.1 编程世界初探

- **程序**
  “程序”常被认为是执行一系列特定任务或解决问题的核心**指令集合。**这些指令的集合描述了计算机解决某一问题的工作步骤。
- **程序设计语言**
  是一种用于编写指令的集合的**形式化语言**，可以用于**完成特定的任务或解决问题。**
  目的：提供一种方式，使人类能够以高效，清晰，结构化的方式表达计算机逻辑和数据操作
  - **低级语言**（机器语言，汇编语言）

    机器语言：是由计算机处理器执行的最基础的语言，它完全由二进制代码（1,0）组成，并且与硬件架构密切相关

    汇编语言：又称符号语言，是一种符号化的机器语言，使用计算机助记符（如`MOV`，`ADD`）来代替二进制代码，需要通过**汇编器**转换成机器语言。常用于需要精确控制硬件操作的程序，如嵌入式系统，驱动程序开发等
  - **高级语言**

    采用更接近自然语言的语法和结构，简化了编程过程，易于维护和阅读，且能够通过编译器或解释器转换成机器可执行的代码

编程语言	主要特点	常见的应用领域
`C语言`	**面向过程**，抽象化，高效性和控制能力强	系统编程，嵌入开发
`C++`	**面向对象**，性能强	游戏开发，高性能应用
`Python`	**跨平台**，**面向对象**，易读性强，库支持丰富	`Web`开发，数据科学，自动化，人工智能
`Java`	**跨平台**，**面向对象**，强大的生态系统	企业级应用，安卓系统开发
`JavaScript`	`Web`开发核心，支持前后端	前端开发，部分后端应用
`C#`	**面向对象**，与`.NET`框架紧密集成	`Windows`应用，游戏开发

1.2 初识Python语言

1. **Python发展历史**

年份	事件描述
1989	荷兰的`吉多·范罗苏姆`开始创建`Python`编程语言
1991	`Python 0.9.0`版本**首次发布**，包括模块、异常处理和函数等特性
1994	`Python 1.0`版本发布，引入循环、异常处理、函数和模块等基本特性
2000	`Python 2.0`版本发布，引入列表推导、**垃圾回收**等特性
2008	`Python 3.0`版本发布，进行了重大改进，包括`Unicode`**支持 新的**`I/O`**库等**
2010	`Python 3.1`版本发布，标志着`Python 3.x`系列的**稳定。**`Python 3.9`开始停止`Windows7`**支持**
2020	`Python 2.7.18`版本为`Python 2.x`的**最后一个版本**`Python`
2010-至今	`Python 3.x`系列持续改进和增强，引入了一系列新特性，包括异步编程、类型注解等。

2. **Python特点**

   - **语法特点**

     Python的语法设计**简洁、清晰**，强调代码的**可读性**，使开发者能够编写易于理解和维护的代码。

   - **免费开源**

     Python是一个**开源且免费**的编程语言，这意味着任何人都可以自由地使用、研究、修改和分发

   - **跨平台性**

     Python的**跨平台性**让它能够在`Windows`、`Linux`和`macOS`等多种操作系统上运行，**不需要修改代码**就能在这些平台之间迁移，极大地提高了开发的灵活性和应用的可移植性。

   - **强大的库**

`Python`的库是其**最大的特点之一**。

第三方库	用途
`NumPy`，`Pandas`	**数据分析**
`Matplotlib`，`Seaborn`	**数据可视化**
`Flask`，`Django`	**`Web`开发**
`TensorFlow`，`PyTorch`	**深度学习**

- **解释性语言**

  解释型语言指的是程序在执行时由**解释器逐行将源代码转换为机器码**，然后立即执行，**无须事先编译。**

3. **Python应用领域**
   - **Web开发**：`Python`的`Django`和`Flask`框架为开发高效、安全的网站和应用程序提供了强大的支持。
   - **人工智能**：`Python` 配合 `TensorFlow`、`PyTorch`等深度学习框架，极大地简化了机器学习模型的开发。
   - **网络爬虫**：利用Python的`BeautifulSoup`和`Scrapy`等库，开发者可以高效地抓取和处理网页数据。
   - **游戏开发**：Python在大型游戏开发中`不是主流选择`，但通过`Pygame`等库，它支持制作`2D`游戏。
4. **面向对象编程**

   面向对象编程是一种**以对象概念为核心**的编程范式，同时也是一种指导程序开发的抽象策略。提高了开发效率和代码的可重用性。
   - **类（`Class`）**：是**对象的模板**，定义了一组对象共有的属性和方法，描述了相同种类对象共享的行为和状态。
   - **对象（`Object`）**：是类的实例。基于类的定义创建的对象拥有类中定义的属性（数据）和方法（行为）。
   - **属性（`Attributes`）**：用来描述对象**特征或状态的信息**。
   - **行为（`Methods`）**：对象可以执行的动作或方法。

     面向对象编程还包括**继承、封装、多态**等其他特性

1.3 迈出Python编程的第一步

1. **Python版本的选择**：选择与操作系统兼容的`Python`版本是软件顺利运行的基本保障
2. **Python安装及环境变量配置**：将`Python` 添加到`PATH`环境变量是为了方便用户在命令提示符中直接调用`Python`解释器，无须指定完整路径。
3. **检查Python版本**：用户可以在命令提示符中运行“`python --version`”命令来确认Python版本
4. **文件保存**：保存文件时，应确保文件以`.py`扩展名结尾，这样`Python`解释器可以识别
5. **`Pycharm`专业版与社区版的主要区别**

特性/功能	`Pycharm`专业版	`Pycharm`社区版
价格	付费软件，需购买订阅	**免费开源**
Web开发	支持Web开发	支持Web开发
数据库支持	提供数据库和`SQL`编辑支持	提供有限的数据库支持
框架支持	`Django`、`Flask`、`PyQT`、`Angular`等	`PyQT`
部署	`Docker`，`Docker Compose`，`Kubernetes`	`Docker`、`Docker Compose`

第二单元 与Python语言熟悉起来

2.1 流程图

1. **算法**

   运用信息技术解决问题的**一系列计算步骤**称为算法。

2. **程序流程图**

作用	图形
起止框	圆角矩形
输入输出框	平行四边形
处理框	矩形/长方形
判断框	菱形
流程线	带方向箭头的流程符号
连接点	小圆圈

3. **程序的三种思维逻辑**
   - **顺序结构**

     按先后顺序**自上而下**逐条执行，直到执行完所有步骤。
   - **选择结构**

     通过**选择判断**决定程序走向，从给定的两种或多种操作中选择一种执行，不同的执行流程就是程序的一个分支。
   - **循环结构**

     重复执行某语句，直到条件不符合为止

2.2 `turtle`

- **海龟(`turtle`)概念及画图原理**

  中文翻译：海龟

  定位：`turtle`是一个`Python`中用于绘图的**内置标准模块。**

  起始坐标：**(0，0)**

默认方向：**向右**

- `turtle` **模块的常用语句**
  - `import turtle`，表示导入turtle模块
  - turtle模块常见动作（`turtle`.后所跟的＜语句＞）

语句	描述
`forward（参数n）`	表示前进`npx`，简写为`fd(n)`。
`backward（参数n）`	表示后退`npx`，简写为`bk(n)`。
`left（参数n）`	表示方向左转角`n°`，可简写为`lt(n)`
`right（参数n）`	表示方向右转角`n°`，可简写为`rt(n)`
`goto（参数 x，y）`	表示光标移到`（x，y）`位置，可取正负值
`color（参数 m，参数 n）`	表示设置颜色，**m表示画笔颜色（若省略与n相同），n表示填充颜色**
`begin_fill（）`与`end_fill（）`	表示在**封闭图形**中开始填充颜色与结束填充颜色，**需成对出现**
`penup（）`与`pendown（）`	表示**提笔**与**落笔**，**需成对出现。**
`circle（参数n，参数m）`	表示画**半径**为`npx`的圆或圆弧，`m>0` **逆时针；**`m<0` **顺时针（若省略**`m`**则默认**`360°`**）**
`pensize（参数 n）`	表示设置画笔宽度（粗细）为`npx`
`done（）`	表示结束海龟画图，**不自动关闭画布**

2.3 常见数据标识与语句

1. **常见标识符**
   - **常量**

     常量指在程序运行过程中值**保持不变的量。**
   - **变量**

     变量是指在程序运行过程中值**会发生变化的量。**命名规则如下：
     1. **只能包含**字母、数字和下划线；
     2. 不能以**数字开头；**
     3. 英文字母**区分**大小写
     4. 建议以**见名知意**的原则来命名
   - **标识符的名称**
     - **命名规范**
       1. **大驼峰命名法**：每一个单词**首字母大写**
       2. **小驼峰命名法**：第一个单词**首字母小写**，其余单词**首字母大写**

3. **下划线命名法**：单词之间以**下划线相连**

- **其他注意事项**

1. 不能使用**系统关键字**作为变量名

```
import keyword        #查看所有系统关键字
print(keyword.kwlist)
```

2. **不建议**使用系统内置函数名、类型名、模块名来定义变量

```
print(dir(__builtins__))     #查看所有内置函数名、类型名、模块
```

2. **赋值语句**

将等号右边的值输送到等号左边的变量中的语句即为赋值语句。

3. **输入语句**

Python中，使用`input`函数接收用户的键盘输入

```
<变量名>=input("提示文字")       #无论输入数字还是其他字符都将作为字符串读取
```

4. **输出语句**

Python中，使用`print`函数可以将结果输出到标准控制台上

```
print(输出内容)
```

2.4 命名与注释

1. **注释**

注释是程序的一个重要组成部分，其内容会被`Python`解释器忽略，不会出现在程序的运行结果中。

- **单行注释**

单行注释以“#”开头，到换行为止。“#”后面所有的内容都作为注释的内容。

```
#注释内容
```

- **多行注释**

多行注释以三个单引号`''' '''`或三个双引号`""" """`作为开始和结束符号。

```
'''
注释内容1
'''
或者
"""
注释内容2
"""
```

2. **代码缩进**

Python采用代码**缩进和冒号**来区分代码块之间的层次。缩进的空格数**是可变的**，一般是**4个空格、8个空格或1个制表符。**

但是同一个代码块的语句**必须包含相同**的缩进空格数

3. **空行**

   但是空行的作用在于**分隔两段不同功能或含义的代码**，便于日后代码的维护或重构。

4. **多行语句**

   Python 通常是一条语句在一行中，但如果语句很长，可以使用以下方法来实现多行续行语句

   - 使用反斜杠“\”续行

   ```
   total=5+4+3+\
             2+1
   ```

   - 在`[]、{}或()`中的多行语句，不需要使用反斜杠“\”

   ```
   total=[5+4+3+
             2+1]
   ```

5. **同一行显示多条语句**

   Python可以在同一行中使用多条语句，语句之间使用**英文**分号`“;”`分隔

   ```
   print("I love Python");print("very much!")
   ```

第三单元 程序世界中的数据奥秘

3.1 数据类型的概念

- **Python中常用的数据类型**

  1. **数值**（`Numeric Types`）

     数值型数据主要用于表示数学运算中的**数字**

  2. **字符串**（`String`）

     字符串主要用于表示**文本或字符序列。**

  3. **列表**（`List`）

     列表是一种**有序的集合**，主要用于保存一组按照特定顺序排列的元素。

  4. **元组**（`Tuple`）

     元组是另一种**有序**的数据结构，主要用于存储一组相关且不变的值。

  5. **集合**（`Set`）

     集合是一个**无序且不重复元素序列。**主要用于成员检测（即检查某个元素是否存在于集合中）和**消除重复元素**

  6. **字典**（`Dictionary`）

     字典是一种**可变**的数据结构，主要用于存储键值（`key-value`）对。字典中的每个元素都由一个键和一个值组成，**键必须是唯一的**，而值可以是任何数据类型。

- **数据类型的分类方式**

  1. **有序**：字符串，列表，元组
  2. **无序**：字典，集合
  3. **可变**：列表，字典，集合

4. **不可变**：数值，字符串，元组

3.2 数值类型

1. **数值类型的分类**
   - **整数**（`int`）

     整数是没有小数部分的数字，可以是正数、负数或零。在Python中，整数的范围**只受限于可用内存大小**
   - **浮点数**（`float`）

     浮点数是有小数部分的数字。由整数部分、小数点和小数部分组成：0.9、-0.12;

     也可以用科学计数法表示,如`1.2e3`、`-2E5`。
   - **复数**（`complex`）

     复数表示数学中的复数，**包含实部和虚部**，通常用于解决一些在实数范围内无法解决的数问题，用`J`或者`j`表示虚数部分。
2. **数值的运算**
   - **算术运算**

     算术运算是最基础的数学操作,用于对数字进行基本计算

运算类型	运算符	描述
加法	`+`	两个数相加
减法	`-`	两个数相减
乘法	`*`	两个数相乘
除法	`/`	两个数相除，结果为浮点数
整数除法	`//`	两个数相除,结果为整数(向下取整)
取余	`%`	返回第一个数除以第二个数的余数
幂运算	`**`	返回第一个数的第二个数次幂
负号	`-`	返回相反数

     算术运算的优先级：括号 > 幂运算 > 正负号 > 乘 = 除 > 取模 = 整除 > 加 = 减。
   - **比较运算**

     比较运算用于判断两个值之间的关系,包括它们是否相等、一个值是否大于或小于另一个值等。

     这些比较运算的结果不是`True`就是`False`。

运算类型	运算符	描述
等于	`==`	检查两个数是否相等
不等于	`!=`	检查两个数是否不相等
大于	`>`	检查第一个数是否大于第二个数

运算类型	运算符	描述
小于	`<`	检查第一个数是否小于第二个数
大于等于	`>=`	检查第一个数是否大于等于第二个数
小于等于	`<=`	检查第一个数是否小于等于第二个数

Python中的布尔类型用`bool`表示,布尔类型只有两个值:`True`(真)和`False`(假)

它是整数型的子类,其中`True`相当于整数值 1 ,`False`相当于整数值 0 。这两个值都是Python的**内置常量**,它们的**首字母大写。**

- **逻辑运算**

  逻辑运算是计算机中用于处理布尔值(真或假)的数学运算,也被称为**布尔运算。**

运算符	描述	举例
`and`	与	`x and y`。如果 x 为`True`,则返回 y 值;如果 x 为`False`,则返回 x 值
`or`	或	`x or y`。如果 x 为`True`,则返回 x 值;否则返回 y 值
`not`	非	`not x`。如果 x 为`True`,则返回`False`;如果 x 为`False`,则返回`True`

  运算结果**不一定是布尔值**:逻辑运算的结果**不一定局限于布尔值**`True`或`False`。

  逻辑运算符的优先级:`not > and > or`。可以通过使用括号来改变运算顺序。

- **三者之间的优先级**

  算术运算 > 比较运算 > 逻辑运算

3. **数值的函数**

函数名	描述	举例	运行结果
`abs(x)`	返回数字 x 的**绝对值**	`abs(-1.5)`	1.5
`max()`	返回给定参数的**最大值**,参数可以为序列	`max(1,9,8,4,5)`	9
`min()`	返回给定参数的**最小值**,参数可以为序列	`min(23,45,56,-3)`	-3
`sqrt(x)`	返回数字 x 的平方根,需导入`math`模块(结果为浮点数)	`math.sqrt(100)`	10.0
`pow(x, y)`	返回 x 的 y 次方的值	`pow(2, 3)`	8

4. **数值类型的转换**

- **转换成整数型**

  `int()`函数将只包含数字的字符串转换为整数。还可以将布尔型`True`转换为整数 1 ,将布尔型`False`转换为整数 0 。

- **转换成浮点型**

  `float()`函数将包含数字和小数点的字符串转换为浮点数。

- **转换为数值型**

  `eval()`函数可以用来执行一个字符串表达式，并返回表达式的值，结果将自动转换成为相应的数值类型。

3.3 字符串类型

1. **字符串的索引及切片**

   要访问字符串中的一段字符,可以用：字符串名`[start: stop: step]`的方法来表示

   `start`是切片的开始索引(可选，默认为 0)，`stop`是切片的结束索引(不包含该索引，可选)，`step`是步长(可选,默认为 1)。

   - Python中的字符串索引**是从0开始的**
   - 切片产生的字符串**不包含**结束索引位置的字符
   - 冒号左右两边的索引是可以**灵活省略的**;省略起始索引表示从字符串的首位开始,省 略结束索引表示取值到字符串的末位;两者均省略,则表示取整个字符串。
   - 负数索引从字符串的末尾开始计数
   - `[::-1]`也是切片操作符,表示从字符串的末尾到开头的**逆向截取**(即步长为`-1`)。

2. **字符串的格式化输出**

格式化输出方法	展示
`format`	`print("你好 {}, 你今年 {} 岁。".format(name, age))`
`f-string`	`print(f"你好 {name}, 你今年 {age} 岁。")`
`%`操作符	`print("你好 %s, 你今年 %d 岁。" % (name, age))`

`format`中可以使用“`{:.2f}`”来保留小数位数

输出的填充和对齐可以使用`^`、`<`或`>`**来分别表示居中对齐、左对齐或右对齐。**`{0:width}`表示字段宽度,如果需要填充,可以在冒号后面添加填充字符,如`{0:fill<width}`。例如,`"{:*>5}".format(123)`的输出结果为“`**123`”,即把123补齐为5位,右对齐,前面以*填充。例如,`"{:*^7}".format(123)`的输出结果为“`**123**`”,即以中间对齐的形式输出123,两端以“*”*补足7位。

3. **字符串的运算**

   字符串的运算逻辑基本与Python的运算逻辑相似

运算符	描述
`+`	表示字符串的连接
`*`	表示字符串的重复
`>`	检查一个字符串按字典顺序是否大于另一个字符串。
`in , not in`	判断字符串中是否包含指定的字符串或字符。

注意事项：

- 字符串的比较**遵循字典顺序**,按照ASCII的大小比较
- 中文字符的比较,**不能**直接使用“==”、“<”、“>”等比较运算符。

4. **字符串的函数**

函数名	**描述**	**举例** `str = "HelloPython"`	**运行结果**
`len()`	返回字符串`str`的**长度**	`len(str)`	11
`max()`	返回字符串中**最大**的字母	`max(str)`	y
`min()`	返回字符串中**最小**的字母	`min(str)`	H
`str.replace(old, new)`	将字符串中的old(旧字符串)**替换**成 new (新字符串)	`str.replace('Python','Java')`	`HelloJava`
`str.find(x)`	检测字符串中是否包含子字符串x,若存在,**则返回索引值,不存在则返回-1**	`str.find('Java')` `str.find('Python')`	-1 5
`','.join(str)`	将序列中的元素以指定的字符(,)连接,生成**一个**新的字符串	`','.join(str)`	H,e,1,1,0, P, y,t,h,o, n
`str.upper()`	将字符串中的小写字母**转为大写字母**	`str.upper()`	`HELLOPYTHON`
`str.lower()`	将字符串中的大写字母**转为小写字母**	`str.lower()`	`hellopython`
`str.count(x)`	统计字符串里某个字符(x)出现的**次数**	`str.count('o')`	2

函数名	描述	举例 `str = "HelloPython"`	运行结果
`str.split(x)`	以指定分隔符(x)对字符串进行**切片**	`str.split('o')`	`['Hell', 'Pyth','n']`
`str.index(x)`	检测字符串中是否包含子字符串x,若存在,**则返回索引值,不存在则报错**		
`str.strip()`	删除字符串**两端**的空格	`str.strip('  abc  ')`	`abc`
`str.lstrip()`	删除字符串**左端**的空格	`str.lstrip('  abc  ')`	`abc`
`str.rstrip()`	删除字符串**右端**的空格	`str.rstrip('  abc  ')`	`abc`

5. **字符串的转换**

   可以直接使用**内置**的 `str()` 函数可以将整数、浮点数转换为字符串。

3.4 列表类型

1. **列表元素的访问**
   - **索引访问**

     通过指定元素的索引位置(从0开始),可以直接访问列表中的特定元素。
   - **切片访问**

     利用切片操作(使用冒号分隔起始索引和结束索引),可以访问列表中的一个 子序列,即连续多个元素的集合。
2. **列表的运算**

运算符	描述
加法运算（`+`）	两个列表相加会返回一个**新列表**, 其中包含两个列表中所有的元素。
乘法运算（`*`）	列表与整数相乘会返回一个**新列表**， 其中包含重复指定次数的原列表元素。
比较运算(`==、!=、>、>=、<=、<`)	逐个元素地比较两个列表, 直到找到一对不满足的元素或者一个列表被**完全遍历**。
成员运算(`in, not in`)	判断某个元素是否**存在于列表中**。

3. **列表的函数**

函数名	描述	示例
`append()`	向**列表末尾**添加一个元素	`lst.append(4)`
`extend()`	将另一个列表的元素添加到当前列表的**末尾**	`lst.extend([5, 6])`
`insert()`	在指定位置**插入**一个元素	`lst.insert(1, 'b')`
`remove()`	移除列表中**第一个出现**的指定元素	`lst.remove('a')`
`pop()`	移除列表中的指定元素(**默认是最后一个**), 并返回它	`lst.pop(0)`
`index()`	返回指定元素在列表中的索引(**没找到报错**)	`lst.index('a')`
`count()`	返回列表中指定元素出现的**次数**	`lst.count('a')`
`sort()`	对列表中的元素进行排序(**直接改变原列表**)	`lst.sort()`
`reverse()`	反转列表中的元素(**直接改变原列表**)	`lst.reverse()`

4. **列表的转换**

   可以通过内置的 `list()` 函数将其他类型(元组、集合、字符串)的数据转换成列表。

第四单元 控制程序的“指挥棒”

4.1 顺序结构与选择结构

- **顺序结构**

  顺序结构是程序中最基本的程序结构,没有特定的语法结构,按照代码语句的先后顺序, 从左到右,从上到下依次进行,一直执行到最后一条代码语句。

- **选择结构**

  选择结构可控制程序的流程,根据表达式从指定的两种或多种操作中选择一种执行,不同的执行流程就是程序的分支。

  1. **单分支**：如果条件成立,执行代码段1;条件不成立,不执行代码段1。

     ```
     if 条件:
         代码段1
     ```

  2. **双分支**：当条件满足时,执行代码段1;当条件不满足时,执行代码段2。

     ```
     if 条件:
         代码段1
     else:
         代码段2
     ```

  3. **多分支**

```
if 条件1:
    代码段1
elif 条件2:
    代码段2
else:
    代码段3
```

4.2 `while`循环和`for`循环

1. **循环结构**

   循环结构是指在程序中**反复执行某个功能**代码。一般由循**环关键字、循环条件、循环体**组成。

2. `for`**循环**

   - **格式一**

     ```
     for 迭代变量 in 字符串|列表|元组|字典|集合 :
         循环体语句
     ```

     在Python 中 for 循环常用于遍历**字符串、列表等序列类型**,逐个获取序列中的各个元素,**迭代变量**(循环变量)用于存放从序列中逐个取出的元素。

   - **格式二**

     ```
     for 迭代变量 in range(start , stop, step):
         循环语句
     ```

     `range` 函数产生一个**整数**序列,程序先读取整数系列中的第一个整数,放到循环变量中,然后执行循环语句体完成一次循环操作,再重复操作(即取下一个整数,执行循环语句体),直到所有的**整数全部取完**,结束for循环。

     1. `start`：计数开始值。默认为 0 ,**可省略。**
     2. `stop`：计数结束值。结束值**不能省略。**
     3. `step`：步长,默认为1,**可省略。**

3. `while`**循环**

   - **格式一**

     ```
     while 表达式:
         循环体语句体
     ```

     while循环的执行流程为：**先判断条件**，若为真则执行循环体，执行完毕后再次进行条件判断，直到**条件为假时循环结束。**

   - **格式二**

     ```
     while True:
         循环语句体
         break
     ```

     无限循环的条件始终为真，会一直执行循环体，通常需要在循环体内使用 break 语句来终止循环。

4. `break` **语句和** `continue` **语句**

   用在循环体内,控制程序的流程走向。

   - `break` 语句的作用是**终止循环。**
   - `continue` 语句会跳过本次循环中其后的代码，立即开始下一次循环。

4.3 流程嵌套

1. **流程嵌套**

   流程嵌套是指在一个**控制结构**内部包含另一个控制结构。例如**循环内嵌套循环**，或**条件语句内嵌套条件语句。**这种结构使程序能够处理**更复杂的多层次逻辑问题。**

2. **枚举算法**

   枚举算法（穷举法） 是一种**流程嵌套。**其核心是**遍历所有可能方案或值**，并**检查每个方案是否满足特定条件。**在程序实现上，通常表现为**循环结构与选择结构的结合。**

3. **冒泡排序**

   排序的作用是将**无序列表变为有序列表。**

   冒泡排序通过**重复遍历列表**，在每次遍历中**比较相邻元素**的大小。若顺序不符合要求（如升序中前一个大于后一个），则**交换它们的位置。**经过多轮遍历，较大（或较小）的元素会逐渐移动到一端，如同**气泡上浮**，故此得名。

4. **程序调试**

   - **设置/取消断点**

     断点设置:单击代码所在行的**左侧行号。**

     断点设置取消:再次单击行号。

   - **调试运行**
   - **单步执行代码**

     单击"单步执行代码"按钮,系统将按照代码顺序逐步执行,在每一步执行后展示对应。

   - **监视内存变量值**

第五单元

5.1 内置函数与模块

1. **模块的导入**

   在使用相应模块内的函数时,首先需要先导入模块。

   - **导入模块的所有函数**

     ```
     from 模块名 import *
     #调用方法
     函数名
     ```

   - **导入模块的其中一个函数**

     ```
     from 模块名 import 函数名
     #调用方法
     函数名
     ```

- **导入模块**

```
import 模块名
#调用方法
模块名.函数名
```

- **导入模块同时别名**

```
import 模块名 as 代替名称
#调用方法
代替名称.函数名
```

2. `math` **模块的使用**

函数名	函数功能	举例	运行结果
`sum()`	返回参数的**和**	`sum([23, 12, 2, 5, 43])`	85
`max()`	返回参数的**最大值**,参数可以是字符串、列表等	`max(23, 12, 2, 5, 43)`	43
`min()`	返回参数的**最小值**,参数可以是字符串、列表等	`min(23, 12, 2, 5, 43)`	2
`sqrt(x)`	返回数字 x 的**平方根**	`sqrt(64)`	8.0
`abs(x)`	返回数字 x 的**绝对值**	`abs(-1.2)`	1.2
`pow(x,y)`	返回 (x 的 y 次方)的值	`pow(3, 3)`	27

在使用 `math` 模块时，需注意**参数的类型与单位**（如 `sqrt()` 的参数应为**非负数**），以及**返回值类型**（如 `sqrt()` 返回**浮点数**）。

需区分常用函数：`max()`、`min()` **是内置函数，无需导入**即可使用；而 `math.sqrt()` **等函数则必须导入** `math` **模块**后才能调用。

3. `random` **模块的部分函数使用**

函数名	函数功能	举例	运行结果
`randint(a, b)`	生成一个在 `a~b` (包括 a 和 b)的随机整数	`random.randint(1, 10)`	返回1~10的任意整数,包括1和10
`choice(x)`	从非空序列x中随机选择一个元素	`random.choice([1, 2, 3, 4, 5])`	返回列表中的任意元素
`uniform(a,b)`	生成一个在 `a-b` (包括 a 但不包括 b)的随机浮点数	`random.uniform(1.0, 10.0)`	返回1.0~10.0的任意浮点数、但不包括10.0

函数名	函数功能	举例	运行结果
`random()`	生成一个在 `0.0~1.0` (包括 0.0 但不包括 1.0)的随机浮点数	`random.random()`	返回值如 0.9238725911608153

4. `time` 函数的部分模块使用

函数名	函数功能	举例	运行结果
`time()`	返回当前时间的时间戳	`time.time()`	1705739518. 2664244
`sleep(seconds)`	暂停程序指定的秒数	`time.sleep(3.5)`	程序暂停3.58
`localtime([seconds])`	将一个时间戳转换为一个表示本地时间的“时间元组”	`time.localtime()`	`time.struct_time(...)`
`strftime(format[,t])`	将一个“时间元组”或“结构化时间”格式化为一个字符串	`time.strftime("%Y-%m-%d %H:%M:%S")`	`'2024-01-20 16:30:10'`

5. `string` 模块的常用函数

函数名	函数功能	举例	运行结果
`ascii_letters`	获取大小写英文字母	`string.ascii_letters`	`abc... xyzABC...XYZ`
`digits`	获取 `0~9` 数字	`string.digits`	0123456789

函数名	函数功能	举例	运行结果
`ascii_uppercase`	获取大写英文字母	`string.ascii_uppercase`	`ABC... XYZ`
`punctuation`	获取所有的标点符号	`string.punctuation`	`!"#$%&'()*+,-./:;<=>? @ [\]^_^{}~`

6. `os` 模块的常用函数

函数名	函数功能	举例	运行结果
`system`	调用应用程序	`os.system('cmd')`	打开 `cmd` 窗口
`name`	返回当前使用平台的代表字符	运行 `os.name`	在 `windows` 系统下 `nt`
`getcwd`	返回当前工作目录	`os.getcwd()`	`c:\\test`
`listdir(path)`	返回 path 所指定目录下的文件和目录名	`os.listdir()`	`['dlls', 'doc', 'include', 'lib']`

5.2 自定义函数

1. **自定义函数的概念**

   Python 中可直接使用**内置函数**（如 `input()`、`max()`）或**模块函数**（如 `random.randint()`），也可以通过**自定义函数**实现特定功能。自定义函数是**由用户定义的可执行代码块**，用于完成内置函数无法实现的任务。

2. **自定义函数的语法**

```
def 函数名（参数):
    语句块
    return 返回值
```

   - **关键字**

     函数定义从 `def` 开始。

   - **函数名**

     `def` 其后是**函数名**，用于唯一标识函数；命名规则与变量名相同。函数名后必须紧跟**括号和冒号。**

   - **参数**

     参数是函数接受的**输入**，数量可以是 **0 个、1 个或多个**，多个参数之间用**逗号分隔**，并全部放置在**括号内。**

   - **函数体**

函数体从 `def` 下一行开始，必须**向右缩进**，是由一系列语句组成的**代码块**，用于执行**特定任务**。若函数体为空，可用 `pass` 语句占位。

- **返回值**

  函数通过 `return` **语句**返回结果，可以返回 **0个、1个或多个值**。多个返回值通常用**元组、列表等可迭代对象**表示；若未指定返回值，则默认返回 `None`。

3. **参数分类**

   函数参数按调用方式可分为：**位置参数**、**关键字参数**、**默认参数** 和 **不定长参数**。

4. **变量作用域**

   变量作用域函数内的变量称为**局部变量**，函数外的变量称为**全局变量**

变量	作用域	访问情况
局部变量	局部作用域	局部变量只能在被定义的函数内访问
全局变量	全局作用域	自定义赋值之后,可供后续的代码进行访问

5.3 异常处理

- **什么是程序异常**

  1. **语法错误**：如**缺少冒号、拼写错误或缩进错误**，导致代码无法运行，必须**修改代码**来解决。
  2. **运行时错误**：常与**用户输入或逻辑**有关（如**除数为零、索引越界**），可通过**检查与条件判断**来预防。
  3. **异常**：指**难以预料**的错误（如**输入类型错误、文件不存在**）。必须使用 `try-except` **语句**进行处理，否则程序可能意外终止。

- **什么是异常处理**

  异常处理指程序对运行时**非正常逻辑的意外情况**进行处理。在 Python 中，可使用 `try…except…else…finally` 机制：

```
try:
    # 可能存在异常的代码
except:
    # 发生异常时执行的代码
else:    # 可选
    # 未发生异常时执行的代码
finally: # 可选
    # 无论是否异常，都会执行的代码
```

  - 若 `try` 块中的代码出现错误，将立即跳转至对应的 `except` 块。
  - 若未发生异常，则执行 `else` 块（如果存在）。
  - `finally` **块**（可选）始终会执行，常用于清理资源。

出处：23数媒2班 陆云清