目录

[1.函数式编程的核心特性包括： 1](#_Toc239831422)

[2.命令式编程的核心： 1](#_Toc711933631)

[3.Python类型 2](#_Toc2011570840)

[4.三种 import 方式： 2](#_Toc1249031646)

[5.常量与变量： 2](#_Toc411251871)

[6.标识符命名规则： 3](#_Toc710704787)

[7.逻辑运算符： 3](#_Toc1439008973)

[8.列表： 3](#_Toc976570953)

[9.私有成员与共有成员： 4](#_Toc749074780)

[10.文件内容操作： 4](#_Toc1582315157)

[11.异常处理结构： 5](#_Toc2077564718)

[编程题 6](#_Toc843401325)

[例1: 提示用户输入一个三位及以上的整数，并输出其百位以上的数字 6](#_Toc1892693328)

[例2: 编写程序，生成包含1000个0至100之间的随机整数，并统计每个元素的出现次数 6](#_Toc899578154)

[例3: 编写程序，生成包含20个随机数的列表，然后将前10个元素升序排列，后10个元素降序排列，并输出结果 7](#_Toc1359223522)

[例4: 编写程序，用户输入4位整数作为年份，判断其是否为闰年 7](#_Toc187238105)

[例5: 编写程序，用户输入一段英文，然后输出这段英文中所有长度为3个字母的单词 8](#_Toc952169015)

[例6: 编写函数，判断一个整数是否为素数，并编写主程序调用该函数 8](#_Toc652402779)

[def is\_prime(number): 8](#_Toc924709799)

[例7: 编写函数，接收一个字符串，统计大写字母、小写字母、数字、其他字符的个数，并以元组的形式返回结果 9](#_Toc2014411761)

[例8: 假设有一个英文文本文件，编写程序读取其内容并将其中的大写字母变为小写字母 9](#_Toc699034730)

## 1.函数式编程的核心特性包括：

* **数据不可变性**：数据一旦创建，其值就不能更改，不可被修改。这使得函数式编程更加简洁、安全。
* **能够接受函数作为参数**：函数可以作为其他函数的参数传递，增强了灵活性和可组合性。
* **产生相同输出的函数**：相同的输入会产生相同的输出，符合确定性（没有副作用）。

## 2.命令式编程的核心：

* **处理性能**：通常命令式编程的性能比函数式编程高，因为它直接操作数据并改变其状态。

## 3.Python类型

Python 2.x系列已于2020年4月20日全面停止维护和支持。

Python 3.x系列：每次发布新版本都会增加一些新特性、更新标准库、增强某些内建函数或模块，同时也会对一些内置对象的实现进行优化，以提高性能。

**注意**：Python语言编写的程序在不同版本之间可能会出现兼容性问题。高版本的Python解释器能够兼容低版本代码，但在版本升级时可能会有不兼容的改动：删除了一些低版本中的对象、修改了某些对象的特性。解决方案是查阅扩展库的升级历史，并检查代码中的不兼容之处。

## 4.三种 import 方式：

（这里可以根据实际情况填写具体内容）

## 5.常量与变量：

在Python中，常量是指在程序的整个生命周期内保持不变的值。Python没有内置的常量类型，但可以通过将变量名全部大写表示常量。这样的常量在程序运行过程中其值不能改变。

Python中的常量可以是数字、字符串、布尔值和空值等：

* 数字包括整数、浮点数和复数。
* 字符串是由字符组成的序列。
* 布尔值只有 True 和 False 两种。
* 空值 None 用于表示缺少值。

#### 关于变量的特点：

Python中的变量并不直接存储值，而是存储了值的内存地址或引用。变量本身是一个指向数据对象的引用。

## 6.标识符命名规则：

在定义变量名、函数名、类名、方法名等标识符时，需要注意以下几点：

* 必须以字母（汉字或英文字母）或下划线开头。
* 不能包含空格和标点符号（如括号、引号、逗号、斜线、反斜线、冒号等）。
* 不能使用Python的关键字（可以通过 import keyword 查看所有Python关键字）。
* 标识符区分大小写，例如 student 和 Student 是不同的变量。
* 不建议使用系统内置的模块名、类型名或函数名，这样会改变其类型和含义（可以通过 dir(builtins) 查看所有内置模块、类型和函数）。

## 7.逻辑运算符：

逻辑运算符 and、or、not 常用来连接多个条件表达式，构成更加复杂的条件表达式。and 和 or 具有惰性求值的特点。

* 例如：
  + 表达式 exp1 and exp2 等价于 exp1 if not exp1 else exp2
  + 表达式 exp1 or exp2 等价于 exp1 if exp1 else exp2

在编写复杂条件表达式时，合理安排不同条件的先后顺序，可以利用这个特点提高代码执行效率。

## 8.列表：

列表包含若干元素的有序连续内存空间。当列表增加或删除元素时，列表对象会自动进行内存的扩展或收缩，从而保证相邻元素之间没有缝隙。

* Python列表的内存自动管理功能可以大幅度减少程序员的负担，但在非尾部位置插入和删除元素时，会涉及到列表中大量元素的移动，严重影响效率。
* 列表推导式是一个非常简洁的方式来快速生成满足特定需求的列表。可以对原有数据对象中的元素进行计算或过滤，代码简洁且易于阅读。

**列表推导式语法**：

[expression for item in iterable if condition]

## 9.私有成员与共有成员：

类定义中以下划线开头或结束的变量名和方法名有特殊的含义，模块中以下划线开头的成员不会被 from ... import \* 导入。

* \_xxx：受保护成员（Protected Member），表示该成员不应被外部访问，但可以被子类继承和修改。
* \_\_xxx\_\_：系统定义的特殊成员（Special Methods），例如 \_\_init\_\_、\_\_str\_\_ 等，这些方法是 Python 语言内置的特殊方法。
* xxx：私有成员（Private Member），表示该成员仅供类对象自己访问，子类对象不能直接访问。然而，Python 并没有严格意义上的私有成员，虽然成员名以双下划线开头（如 \_\_xxx），但仍然可以通过一定的方式进行访问，例如 对象名.\_类名\_\_xxx。

**注意**：Python 中并没有严格意义上的私有成员。

## 10.文件内容操作：

文件可以分为文本文件和二进制文件：

**文本文件**：存储的是常规字符串，由若干文本行组成，通常每行以换行符 \n 结尾。  
**提示**：实际上，包括文本文件在内的任何类型文件在计算机内部都是以二进制形式存储的，只是文本文件的编码格式由软件自动识别。

**二进制文件**：二进制文件不能用记事本或普通字处理软件直接编辑，即使打开也无法直接阅读，需要使用专门的工具解码。图像文件、音视频文件、可执行文件等都属于二进制文件。

## 11.异常处理结构：

异常是指程序运行时引发的错误，错误的原因可以是除零、下标越界、文件不存在、网络异常等。如果这些错误没有得到正确处理，会导致程序终止运行。合理地使用异常处理结构，可以使得程序更加健壮，具有更强的容错性，不会因为用户输入错误或其他运行时原因造成程序崩溃。同时，可以提供友好的提示给用户。

## 编程题

### 例1: 提示用户输入一个三位及以上的整数，并输出其百位以上的数字

首先，我们提示用户输入一个三位及以上的整数。然后，通过整数除法来输出该数字的百位及以上的部分。

# 提示用户输入一个三位及以上的整数

number = int(input("请输入一个三位及以上的整数: "))

# 输出这个整数的百位及以上的数字

high\_digits = number // 100

print(high\_digits)

### 例2: 编写程序，生成包含1000个0至100之间的随机整数，并统计每个元素的出现次数

我们使用 random.randint(0, 100) 生成1000个0到100之间的随机整数，并利用 collections.Counter 来统计每个整数出现的次数。

import random

from collections import Counter

# 生成1000个0到100之间的随机整数

random\_integers = [random.randint(0, 100) for \_ in range(1000)]

# 统计每个整数出现的次数

counter = Counter(random\_integers)

# 打印出每个整数及其出现次数

for number, count in counter.items():

print(f"{number}: {count}")

### 例3: 编写程序，生成包含20个随机数的列表，然后将前10个元素升序排列，后10个元素降序排列，并输出结果

import random

# 生成包含20个随机数的列表

random\_numbers = [random.random() for \_ in range(20)]

# 将列表的前10个元素升序排列

sorted\_part1 = sorted(random\_numbers[:10])

# 将列表的后10个元素降序排列

sorted\_part2 = sorted(random\_numbers[10:], reverse=True)

# 将排序后的部分合并

sorted\_part1.extend(sorted\_part2)

print(sorted\_part1)

### 例4: 编写程序，用户输入4位整数作为年份，判断其是否为闰年

通过判断年份是否能被4整除但不能被100整除，或者能被400整除，来判断该年份是否为闰年。

def is\_leap\_year(year):

"""判断年份是否为闰年"""

if (year % 4 == 0 and year % 100 != 0) or year % 400 == 0:

return True

return False

# 用户输入年份

year = int(input("请输入一个4位整数年份: "))

# 判断并输出结果

if is\_leap\_year(year):

print(f"{year} 是闰年。")

else:

print(f"{year} 不是闰年。")

### 例5: 编写程序，用户输入一段英文，然后输出这段英文中所有长度为3个字母的单词

# 接收用户输入

input\_str = input("请输入一段英文文本: ")

# 将输入的文本转换为小写，并用空格分割成单词

words = input\_str.lower().split()

# 输出长度为3的单词

three\_letter\_words = [word for word in words if len(word) == 3]

print("\n以下是所有长度为3的单词:")

for word in three\_letter\_words:

print(word)

### 例6: 编写函数，判断一个整数是否为素数，并编写主程序调用该函数

### def is\_prime(number):

"""判断整数是否为素数"""

if number <= 1:

return False

for i in range(2, int(number \*\* 0.5) + 1):

if number % i == 0:

return False

return True

# 主程序调用

num = int(input("请输入一个整数: "))

if is\_prime(num):

print(f"{num} 是素数")

else:

print(f"{num} 不是素数")

### 例7: 编写函数，接收一个字符串，统计大写字母、小写字母、数字、其他字符的个数，并以元组的形式返回结果

def count\_chars(s):

uppercase = sum(1 for char in s if char.isupper())

lowercase = sum(1 for char in s if char.islower())

digits = sum(1 for char in s if char.isdigit())

others = len(s) - uppercase - lowercase - digits

return uppercase, lowercase, digits, others

### 例8: 假设有一个英文文本文件，编写程序读取其内容并将其中的大写字母变为小写字母

# 打开文件并读取内容

with open('example.txt', 'r') as file:

content = file.read()

# 将大写字母转换为小写

lowercase\_content = content.lower()

# 输出转换后的内容

print(lowercase\_content)