扩展数据类型：怎样使用更复杂的数据类型?

目录

命名元组

- namedtuple() 是命名元组的工厂函数
- 命名元组使用前需要导入 collections 库
- 定义一个“点”，包含了 x 和 y 两个坐标：

```
Point = namedtuple('Point', ['x', 'y'])
```

双端队列

- deque 对象是实现双向队列的对象
- 双向队列能够支持从左右两端实现元素的添加和移除
- deque 比传统的列表多了 appendleft()、popleft() 方法

计数器

- Counter 对象 —— 计数器工具
- 计数器工具可以用来统计字典中元素的数量，也可以用来统计元素的出现次数

字典和列表子类化

- UserDict 类用于字典对象的二次开发
- UserList 类用于列表对象的二次开发
- 当你需要使用字典、列表，而他们又不能完全满足你的需要时，

  可以通过继承 UserDict 和 UserList，实现增强功能的字典和列表

总结

1. collections 模块，提供了一系列的扩展数据类型

2. 当传统的列表、字典无法满足你的需要时，可以利用计数器、双端队列、命名元组来丰富你的数据类型

课后作业

请你编写程序，统计一篇文章中出现频率在前五的单词，并将单词和出现次数一起输出到终端。

魔术方法：怎样通过类构造自己需要的数据类型?

目录

魔术方法的作用

- 魔术方法是 Python 预先定义好有特定功能的一类方法
- 魔术方法可以理解为数据类型的接口
- 改变数据类型，通常采用重写魔术方法来实现

基本数据类型中的魔术方法

- 字典中的魔术方法：

  dict1 = {'a':1}

  dict1[a]　# 结果为 1

  dict1.__getitem__('a') #结果为 1

- 如果你需要重写字典取值过程，可以通过 __getitem__() 魔术方法来实现

基本数据类型中的魔术方法

- 使用 dir() 方法，可以得到各数据类型中的魔术方法：

  dir(字典)

类的魔术方法

类的魔术方法有：

__init__()　# 初始化

__new__()　# 创建对象

__call__()　# 可调用，类似函数

__del__()　# 对象在内存中释放

总结

1. 魔术方法是一类以"__"开始和结束的方法
2. 魔术方法在对象中有特定用途
3. 当你需要修改特定对象的行为时，可以通过魔术方法实现修改

课后作业

请编写一个自定义类，要求该类能实现字典所有功能，当用户为字典赋值，遇到重复的 key 时，向用户报错并禁止覆盖该 key，例如：

```
mydict = MyDict()
mydict['a'] = 1
mydict['a'] = 1  #对相同的 key 赋值报错。
```

提示：赋值错误，key "a" 已存在，放弃赋值。

怎样将 Python 和 C++ 结合起来混合编程?

目录

为什么需要混合编程

- Python 是解释型语言，在进行数学运算场景下，性能是瓶颈
- C++ 性能卓越，但是学习门槛高且开发效率比 Python 低
- C++ 可以用于密集型计算并用 Python 进行调用

实现混合编程的方式

1. 使用ctypes库加载C++编写的动态链接库

   参考链接：https://docs.python.org/zh-cn/3.10/library/ctypes.html

2. 使用pybind将C++编译为Python库

   参考链接：https://github.com/pybind/python_example

3. 使用Pythran库将Python直接转换为C++代码

   参考链接：https://pypi.org/project/pythran

使用Pythran转换代码步骤

1. 编写需要转换为 C++ 函数
2. 使用Pythran命令行转换为C++代码
3. 编写C++测试代码

Pythran命令行参数

- 获得命令行帮助：

```
$> pythran --help
    usage: pythran [-h] [-o OUTPUT_FILE] [-P] [-E] [-e] [-v] [-w] [-V] [-p pass]
                   [-I include_dir] [-L ldflags] [-D macro_definition]
                   [-U macro_definition] [--config config] [-ftime-report]
                   input_file
```

Pythran命令行参数

```
pythran –e 要转换的python源码.py

           –p pythran.optimizations.ConstantFolding

           –o output.hpp
```

Pythran命令行参数

```
$> `pythran-config --compiler --cflags` -std=c++11 cli_foo.cpp -o cli_foo
```

pythran-config --compiler --cflags: 编译需要的库

-std=c++11 使用C++而非C语言编译、链接

-o 输出文件名

总结

1. Python和C++混合编程有多种实现形式
2. 将Python完全转换成C++能减少代码不一致问题
3. Pythran还不能支持完整的Python标准库

课后作业

请你基于Pythran编写一个计算圆周率的Python函数，并使用C++调用该函数，运行后对纯Python代码和C++代码进行运行时间的比较。

怎样将已有算法改造成符合项目的特定算法?

目录

为什么要掌握算法

- 掌握成熟软件的设计思想
- 正确评估自己的代码质量

评估算法

- 利用大 O 复杂度表示法进行时间复杂度分析
- 常见的几种时间复杂度：

  O(1)

  O(logn)

  O(n)

  O(n*n)

改造算法

- 使用 Python 编写计算斐波那契数列程序，并使用 time() 函数计算开销时间

改造算法

- LRU 算法 —— LRU（Least Recently Used）最近最少使用
- functools.cache() —— 轻量级的函数缓存功能装饰器
- 计算缓存后的时间开销

总结

1. 算法是前人对一类问题的通用解决方案
2. 可以通过时间复杂度的方法来评估算法的优劣
3. 可以基于业务特点来改造现有算法

课后作业

编写一个能够计算 100 以内阶乘的函数，已知 60% 的人经常计算 10 和 100 的阶乘，请在此前提下优化阶乘函数。

设计模式：怎样合理组合多个函数和类？

目录

创建型模式

- 创建型模式的典型设计模式：单例模式
- 单实例模式可以由import来实现
- 可以借助类属性实现多实例共享(monostate)一个属性

结构型模式

- 最实用的是适配模式，通常用来解决兼容问题

行为模式

- 责任链模式实现了发送者和接收者解耦，让多个对象接收发送者请求，并沿着链式结构一直传递，直到有目标对象处理该请求为止

行为模式

- 责任链模式经典场景：异常的分层处理
- 当用户请求网站失败时，从DNS、HTTP、Socket逐层返回

总结

1. 设计模式非常丰富，主要根据行为分为：创建型、结构型和行为模式
2. 设计模式可以让你的程序应对更复杂的需求场景
3. 设计模式有较为成熟的代码和解决方案，可以利用现有代码缩短开发周期

课后作业

为了满足使用自动饮水机泡茶、吃药、冲咖啡等需求，需要为饮水机设计四个按钮，通过按钮，设定饮水机自动出水温度分别为：40 度、60 度、90 度、100 度。

请根据此需求设计合理的设计模式。

Redis 数据库：怎样使用 NoSQL 数据库?

目录

Redis 数据库的用途

- Redis 是一款高性能的键值数据库
- Redis 既有高性能的数据处理能力，又有丰富的编程接口

使用 Python 访问 Redis 数据库

- Python 需要使用 redis-py 库连接 Redis 数据库

  pip3 install redis

- 连接数据库的基本用法为：

  import redis

  conn = redis.Redis(host, port, password, db)

使用 Python 对 Redis 数据库进行读写操作

- Redis 和 Python 一样，都有自己的一组数据类型，它包含 6 个数据类型：

  字符串、双向链表、压缩列表、哈希表、跳表、整数数组

使用 Python 对 Redis 数据库进行读写操作

使用 Redis 的字符串：

```
conn.set('key', 'value') #存储字符串

conn.get('key') #读取字符串
```

总结

1. Redis 是企业开发中最常用的非关系型数据库
2. Python 连接 Redis 需要为其安装第三方库
3. Redis 支持丰富的数据类型，可以将 Python 中的数据类型进行直接存储，而不需要进行复杂的类型转换

课后作业

请将 Redis 作为 Python 的消息队列，实现消息的顺序存储和顺序读取。

Python 进程结束再重新启动，保证消息队列中的数据不会丢失。

关系型数据库：如何使用关系型数据库?

目录

关系型数据库的用途

- 比普通文本更灵活，接口更完善
- 数据安全性比文本更强
- 查询速度更快

连接数据库

- 安装连接MySQL数据库的库：

```
pip3 install PyMySQL
```

- 连接数据库：

```
import pymysql
db = pymysql.connect(host, user, password, db)
```

获得游标：

```
with connection:
    with connection.cursor() as cursor:
```

- 提交数据：

```
db.commit()
```

数据查询

- 使用 cursor.execute() 可以执行 SQL 语句
- 查询的 SQL 语句为：

  SELECT 列 FROM 库.表 WHERE 查询条件

数据修改

- 插入数据：

  INSERT INTO 表(字段名) VALUES(字段值)

- 数据插入成功后，需要提交，插入失败则需回滚：

```
try:
    执行SQL语句
    db.commit()
except:
    db.rollback()
finally:
    db.close()
```

数据修改

- 更新数据：

  UPDATE 表 SET 字段=值 WHERE 更新条件

- 删除数据：

  DELETE FROM 表 WHERE 删除条件

总结

1. 关系型数据库是存储数据的常用做法
2. Python 可以对 MySQL 进行增删改查等操作
3. 数据修改后需要提交，修改的数据才能生效

课后作业

请尝试通过 Python 读取当前执行程序的电脑的 IP 地址，并将 IP 地址存储到 MySQL 数据库中。

计算资源充足时，如何通过并行设计提高效率?

目录

并行与并发的区别

- ✓ 并发：宏观上，多个任务同时执行
- ✓ 并行：同一时刻发生

- ✓ 并发：一个 CPU 核心交替运行多个程序
- ✓ 并行：多个 CPU 核心同时处理多个程序

利用 concurrent 实现并行计算

concurrent.futures 库中的 Executor 对象时并行任务的抽象类

它可以由线程和进程两种方式实现并行计算

Executor 可以通过 submit() 方式执行

线程池

- Executor 对象还支持 ThreadPoolExecutor 方式，使用线程池实现并发
- 它还支持 ProcessPoolExecuter 方式，以使用多核 CPU

总结

1. 并发编程能够更充分利用系统资源
2. 使用 concurrent.futures 库可以实现复杂的并行任务计划
3. concurrent.futures 库既可以使用多线程模型，也能使用多进程模型

课后作业

请使用并发任务模型同时访问 5 个网站，并将网页的数据存储到不同的文件中。

多进程间如何通信?

目录

进程和线程的差别

- 在 CPython 中，由于存在 GIL（全局解释器锁），同一时刻只有一个线程能执行
- I/O 密集型应用可使用多线程模型
- 计算密集型应用应当使用多进程模型

进程通信

- concurrent 并行任务的底层库时 multiprocessing 库
- 使用 multiprocessing 库可以实现多进程任务：

```
from multiprocessing import Pool

with Pool(10) as p:

    p.map(函数对象，参数)
```

进程通信

- 多个进程独立执行，无法进行数据交互
- 多进程间通信，可以支持以下几种方式：
  - 队列
  - 管道
  - 共享内存

进程通信

队列：

```
from multiprocessing import Queue

q = Queue()

q.put() # 写入数据

q.get() # 读取数据
```

进程通信

管道：

```
from multiprocessing import Pipe

parent, child = Pipe() # 管道默认情况下是双工的

# parent 和 child 都有 send() 、recv() 方法
```

进程通信

共享内存：

```
from multiprocessing import shared_memory

shm = shared_memory.SharedMemort(create=True, size=10)
```

总结

1. 多进程用于计算密集型程序

2. 多进程的变量无法直接通信，需要使用队列、管道和消息队列机制实现通信

课后作业

请编写一个多进程任务，一个进程用于产生 1～10000 的数字，另一个进程用于判断该数字是否能被 7 整除，如果可以被整除则输出到终端。

THANKS