Python 基础面经

1.GIL 全局解释器锁

(1) 因为 python 不是线程安全的，GIL 确保任意时刻只有一个线程可以执行 python 代码。

(2) 工作原理：每个线程在执行 python 字节码之前先获取 GIL，当一个线程获取 GIL 之后，其他线程必须等待该线程释放 GIL 才能继续执行。

(3) GIL 存在的主要目的是简化内存管理，但在多核 CPU 上限制了多线程的并行执行。

(4) 对于 I/O 密集型任务（文件读取，网络请求），GIL 影响较小，因为即使一个线程被阻塞，其他线程仍然可以运行。

(5) 如何避免 GIL：将 python 代码转为 C 语言代码；使用 multiprocessing 多进程模块；使用 C 扩展模块，在 C 扩展模块中释放 GIL，允许其他线程执行 python 代码；使用 numpy 矢量化计算。

2.面向对象编程

面向对象编程的核心是类还有对象。类定义了对象的属性和方法，对象是类的实例和具体实现。

四大特性：封装、继承、多态、抽象。

封装：将数据和具体操作代码放在某个对象里面，外部无法直接访问，必须调用类的方法。

继承：在现有类的基础上，创建新类。子类继承父类的属性，可以重用父类的代码，并可以在子类中对父类进行修改或拓展。

多态：当子类和父类有相同方法时，子类的方法会覆盖父类的方法。

抽象：隐藏复杂的实现细节，暴露简单的接口供用户使用。

3.*args & **kwargs

*args：接收任意数量的非关键字参数并打包成元组。

**kwargs：接收任意数量的关键字参数并打包成字典。

函数接收实参，按顺序分配给形参，但遇到带 * 的形参时，就把未分配出去的一起以元组 or 字典形式打包分配给带星号的形参。

4.数据结构（列表，元组，集合，字典）

类型	是否有序	是否可变	是否允许重复	典型用途
list 列表	✅ 有序	✅ 可变	✅ 允许重复	存储有序数据序列
tuple 元组	✅ 有序	❌ 不可变	✅ 允许重复	存储不可变数据，如函数返回多值
set 集合	❌ 无序	✅ 可变	❌ 不允许重复	去重、集合运算
dict 字典	✅ 有序 (3.7+)	✅ 可变	key 不重复	键值映射、快速查找

操作	list 列表	tuple 元组
新增	`append(x)` 末尾加 `insert(i,x)` 指定位置 `extend([...])` 扩展	❌ 不可变
删除	`remove(x)` 按值删 `pop(i)` 按索引删 `clear()` 全删	❌ 不可变
修改	`lst[i]=x` 直接改	❌ 不可变
查找	`lst[i]` 索引 `in` 判断是否存在 `index(x)` 找位置	`t[i]` 索引 `in` 判断存在
遍历	`for x in lst`	`for x in t`
去重	需转 set	需转 set

set 集合	dict 字典
`add(x)` 加元素 `update({...})` 批量加	`d[key]=value` 新增键值
`remove(x)/discard(x)` 按值删 `pop()` 随机删	`pop(key)` 按键删 `del d[key]` `clear()` 全删
只能先删后加	`d[key]=new_value`
`in` 判断存在	`d[key]` 或 `d.get(key)`
`for x in s`	`for k,v in d.items()`
✅ 自动去重	key 自动唯一

例如提问：列表和元组有什么区别？

回答：**列表是可变的，元组是不可变的，列表元素可以修改、添加、删除，但是元组一旦创建就不可以修改。**

提问：字典如何按照 key 排序？

回答：用 sorted 函数实现

Python

```
d = {'b': 2, 'a': 3, 'c': 1}
sorted_items = sorted(d.items())  # 按 key 升序
print(sorted_items)
# 输出:  [('a', 3), ('b', 2), ('c', 1)]
```

5.深拷贝、浅拷贝（常考）

浅拷贝是指复制对象的最外层结构，内部引用共享。copy.copy( )

深拷贝是递归复制内部的所有元素及其可变对象。copy.deepcopy( )

比如创建一个列表嵌套列表，分别使用两种拷贝方法，然后在嵌套列表里面增加一个元素，print 结果，会发现，浅拷贝里面的嵌套列表也增加了这个元素，但是深拷贝没有。

浅拷贝：

- 对象内部元素是不可变对象，或不关心内部共享。
- 如：复制配置模板、树结构的节点引用。

深拷贝：

- 当对象内部含有可变对象，并且需要完全独立修改，避免改动相互影响。
- 如：复制嵌套列表、复杂 JSON 配置。

6.is 和 == 的区别

== 是比较两个元素的值是否相等

is 是比较两个元素内存地址是否相同

7.函数的默认参数是可变对象，会有什么影响？

所有调用共享一个对象，导致后续调用受到前一次修改的影响。

原因：因为默认参数在函数定义时只初始化一次，后续调用都用同一个对象。

解决方法：默认参数用 None，然后在函数内部初始化。

```python
def add_item(x, lst=[]):  # 默认参数是一个可变 list
    lst.append(x)
    return lst

print(add_item(1))  # [1]
print(add_item(2))  # [1, 2]
print(add_item(3))  # [1, 2, 3]
```

8.python 导入库的顺序

标准库：os，sys；第三方库：numpy，pytorch，pandas； 本地库：自己写的代码模块和包。

9.实例方法、静态方法、类方法?

实例方法：类方法中默认都是实例方法，特点就是包含一个 self 参数，调用时需要先实例化。

静态方法：需要使用@staticmethod 修饰，不包含 self 参数，可以直接用类名或者类对象调用。

类方法：需要使用@classmethod 修饰，至少包含一个 cls 参数，使用类名直接调用。

10.装饰器、生成器、迭代器、列表推导式？上下文管理器

装饰器：用来修改或扩展函数方法的高阶函数，接受一个函数作为参数并且返回另一个函数，而不需要修改源代码。例如代码重用、日志记录、性能计时。@cache

生成器：通过 yield 语句生成一个值。它不会一次返回所有的值，而是每次请求一个值时才生成一个。这种惰性求值可以节省内存。

迭代器：实现了 iter 和 next 方法，它允许我们遍历一个集合而不需要暴露集合的内部结构。（例如生成器本身它就是一个迭代器）。

列表推导式：一般是由一个表达式 + 可选条件 +for 语句构成，用一种简洁的方式来创建基于某种逻辑的列表。squares = [x**2 for x in range(5)]

上下文管理器：通过 with 语句来实现，主要用来自动管理资源，保证用完后能正确释放，比如文件读写和数据库连接等。

11.python 如何处理异常?

try,except,else,finally 语句

try 尝试可能引发异常的操作，except 捕获并处理特定异常类型，else 在 try 模块没有引发异常时执行，finally 是无论是否发生异常都会执行，一般用于清理操作、释放资源。

12.lambda 函数?

匿名函数，结构就是 lambda 参数：表达式，通过简洁的表达式来定义一个函数功能。

13.python 中的断言 assert?

用于调试的工具，检查代码中某个条件是否为真。assert 条件表达式，"错误信息（可选）"。成立正常运行，不成立就抛出异常。

14.__slots__是什么？如何使用?

它是一个类变量，用于限制类的实例可以拥有的属性，使用 slots 可以节省内存，因为它阻止了实例字典的创建。

Python

```python
class Person:
    __slots__ = ('name', 'age')  # 限定只能有 name 和 age 两个属性

    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Alice", 30)
# print(p.__dict__)  # 会报错：AttributeError: 'Person' object has no attribute '__dict__'
```

15.map、filter、reduce 函数?

map：对每个元素应用函数，生成新序列。

filter：保留所有返回 True 的元素。

reduce：对元素进行累积处理，返回一个单一值。

Python

```python
def square(x):
    return x * x

nums = [1, 2, 3, 4]
squares = list(map(square, nums))  # 使用 square 函数
print(squares)  # [1, 4, 9, 16]

def is_even(x):
    return x % 2 == 0

nums = [1, 2, 3, 4, 5, 6]
evens = list(filter(is_even, nums))  # 使用 is_even 函数
print(evens)  # [2, 4, 6]

from functools import reduce

def multiply(x, y):
    return x * y

nums = [1, 2, 3, 4]
product = reduce(multiply, nums)  # 使用 multiply 函数
print(product)  # 24
```

16.常见的数据结构（非常重要）！！

(1) 数组和链表的区别?

数组和链表是两种常见的数据结构，它们的主要区别在于内存存储方式、访问效率和插入删除的性能。

- 数组使用**连续内存**，支持通过索引进行**快速随机访问**，时间复杂度是 O(1)。
- 链表使用**非连续内存**，每个节点通过指针连接，只能**顺序访问** O(n)。

在插入和删除方面：

- 数组如果要在中间插入或删除元素，通常需要移动后续元素，效率是 O(n)；
- 链表插入和删除更高效，只需修改指针即可 O(1)，前提是已知前驱节点。

所以如果应用场景是频繁访问元素，适合用数组；如果是频繁插入删除，链表会更合适。

(2) 栈是一种什么数据结构？插入或者删除栈顶元素，指针是否变化?

栈是一种后进先出的数据结构，**只能在栈顶进行插入（push）和删除（pop）操作。**

插入时，新元素放到栈顶，栈顶指针向上移动；删除时，栈顶元素被移除，指针向下移动。

(3) 栈和队列有什么区别？栈和堆有什么区别？

栈是后进先出结构，只能在栈顶插入和删除；队列是先进先出结构，从队尾插入、从队头删除。

堆通常指优先队列，是一种特殊的完全二叉树，用于高效地获取最大 / 最小值。

场景	选啥
随机访问快	数组
插删频繁	链表
后进先出	栈
先进先出	队列
快速查找映射关系	哈希表
需要排序 / 范围查询	树（BST/AVL/红黑树）
动态获取最大 / 最小值	堆

数据结构	逻辑结构	是否连续存储	访问方式	插入 / 删除	查找效率	典型应用
数组 (Array)	线性	✅ 连续内存	✅ 随机访问 O(1)	❌ 代价大 O(n)	按索引快 O(1)	固定大小、随机访问
链表 (Linked List)	线性	❌ 分散节点	顺序访问 O(n)	✅ 只需改指针 O(1)	慢 O(n)	频繁插入删除
栈 (Stack)	线性	可基于数组 / 链表	LIFO	只操作栈顶 O(1)	❌ 不支持随机查找	函数调用、括号匹配
队列 (Queue)	线性	可基于数组 / 链表	FIFO	只操作首尾 O(1)	❌ 不支持随机查找	消息队列、任务调度
哈希表 (Hash Table)	关联映射	✅ 数组 + 散列	通过 key O(1) 均摊	✅ O(1) 均摊	✅ O(1) 均摊	字典、缓存
树 (Tree)	层次结构	节点 + 指针	需遍历 O(log n)	插删需维护平衡	查找 O(log n)	组织层级数据
堆 (Heap)	完全二叉树	数组存储	只能访问堆顶	插删 O(log n)	❌ 不能随机查找	优先队列

17.python 小练习

(1) 如何从一个列表中删除重复元素？先把列表转化为集合，再转回列表，因为集合不允许重复元素。list（set（list））

上个问题也可以变成：如何判断列表有重复元素？

Python

```python
lst = [1, 2, 3, 4, 2]
has_duplicates = len(lst) != len(set(lst))
print(has_duplicates)  # 输出：True，说明有重复
```

（2）如何将多个列表组合成一个元组？使用 zip 函数，list（zip（list1，list2，list3））

```python
a = [1, 2, 3]
b = ['a', 'b', 'c']
c = [True, False, True]

z = zip(a, b, c)
print(list(z))
# [(1, 'a', True), (2, 'b', False), (3, 'c', True)]
```

（3）如何删除字符串中所有空白？"".join(s.split())

（4）如何删除文件？os.remove()

18.python 的参数传递是值传递，还是引用传递?

既不是纯粹的值传递，也不是纯粹的引用传递，而是对象引用传递。

对于不可变对象，函数内部对参数的修改不会影响到函数外部的变量；

对于可变对象，函数内部对参数的修改会影响到函数外部的变量。

19.python 中__init__和__new__有什么区别?

init 是实例初始化，用于初始化对象的属性。new 是实例创建方法，用于创建并返回一个新的实例。

20.如何在一亿个数据里面找到 top 100?

用**最小堆**来实现。维护一个大小为 100 的堆，每来一个新数，如果堆长度小于 100，就加入；如果比堆顶大，就替换堆顶。

Python

```python
import heapq
import random

def find_top_k(data, k=100):
    heap = []
    for x in data:
        if len(heap) < k:
            heapq.heappush(heap, x)
        else:
            if x > heap[0]:  # 比堆顶大就替换
                heapq.heappushpop(heap, x)
    return sorted(heap, reverse=True)  # 返回从大到小排序
```

21.面向过程和面向对象编程有什么区别?

面向过程 (Procedural Programming, POP)

- 核心：把问题分解为一个个 **函数 / 步骤**，一步一步解决。
- 关注点：**怎么做（流程）**。
- 程序结构：数据和操作分开。

面向对象 (Object-Oriented Programming, OOP)

- 核心：把问题抽象为 **对象**，对象封装了 **属性（数据）** 和 **方法（行为）**。
- 关注点：**谁来做（对象）**。
- 程序结构：数据和操作封装在对象内部。

22.python 为什么 async 和 await 可以一起用?

async 定义协程函数，返回协程对象；await 挂起当前协程等待另一个协程完成。两者结合才能实现事件循环中的异步调度。