第三版：Python 55 道

编写一个函数，找出数组中没有重复的值的和

```
def func(lis):
lis1=[]
del_lis=[]
for i in lis:
if i not in lis1:
if i not in del_lis:
    lis1.append(i)
else:
del_lis.append(i)
lis1.remove(i)
return sum(lis1)

def func2(lis):
return sum([i for i in set(lis) if lis.count(i)==1])

print(func2([3,4,1,2,5,6,6,5,4,3,3]))
```

下面代码的执行结果是

```
a=1
def bar():
a+=3

bar()
```

```
print(a)
```

答案：运行出错

写一个函数，计算出以下字母所代表的数字，每个字母值不一样

```
for A in range(1,10):
for B in range(10):
if A==B:
continue
for C in range(1,10):
if C in [A,B]:
    continue
for D in range(10):
    if D in [A,B,C]:
        continue
    for E in range(1,10):
        if E in [A,B,C,D]:
            continue
        for F in range(10):
            if F in [A,B,C,D,E]:
                continue
            for G in range(1,10):
                if G in [A,B,C,D,E,F]:
                    continue
                for H in range(10):
                    if H in [A,B,C,D,E,F,G]:
                        continue
                    for P in range(1,10):
                        if P in [A,B,C,D,E,F,G,H]:
```

```
                continue
            if (A*10+B)-(C*10+D)==(E*10+F) and
(E*10+F)+(G*10+H)==(P*100+P*10+P):
            print(A,B,C,D,E,F,G,H,P)
```

写出如下代码的输出结果

```
def decorator_a(func):
print('Get in decorator_a')
def inner_a(*args, **kwargs):
print('Get in inner_a')
return func(*args, **kwargs)
return inner_a

def decorator_b(func):
print('Get in decorator_b')
def inner_b(*args, **kwargs):
print('Get in inner_b')
return func(*args, **kwargs)
return inner_b

@decorator_b #f=decorator_b(f)
@decorator_a #f=decorator_a(f)
def f(x):
print('Get in f')
return x 2
f(1)
```

答案

Get in decorator_a

Get in decorator_b

Get in inner_b

Get in inner_a

Get in f

解释

当我们对 f 传入参数 1 进行调用时，inner_b 被调用了，他会先打印 Get in inner_b，然后在 inner_b 内部调用了 inner_a，所以会再打印 Get in inner_a，然后再 inner_a 内部调用原来的 f，并且将结果作为最终的返回总结：装饰器函数在被装饰函数定义好后立即执行从下往上执行函数调用时从上到下执行

写出以下代码的输出结果：

```
def test():
try:
raise ValueError('something wrong')
except ValueError as e:
print('error occured')
return
finally:
print('ok')
test()
```

结果(finally 无论怎样都会执行)

error occured

ok

求出以下代码的输出结果

```
mydict={'a':1,'b':2}
def func(d):
d['a']=0
return d

func(mydict)
mydict['c']=2
print(mydict)
```

结果

```
{'a': 0, 'b': 2, 'c': 2}
```

写个函数接收一个文件夹名称作为参数，显示文件夹中文件的路径，以及其中包含的文件夹中文件的如今

```
# 方法一
import os
def Test1(rootDir):
list_dirs = os.walk(rootDir)
for root, dirs, files in list_dirs:
for d in dirs:
print(os.path.join(root, d))
for f in files:
print(os.path.join(root, f))
Test1(r'C:\Users\felix\Desktop\aaa')
```

```
print('#############')
# 方法二
import os
def Test2(rootDir):
paths=os.listdir(rootDir)
for lis in paths:
path=os.path.join(rootDir,lis)
print(path)
if os.path.isdir(path):
 Test2(path)
Test2(r'C:\Users\felix\Desktop\aaa')
```

re 的 match 和 search 的区别

match()函数是在 string 的开始位置匹配，如果不匹配，则返回 None

search()会扫描整个 string 查找匹配；也就是说 match()只有在 0 位置匹配成功的话才有返回，

什么是正则的贪婪匹配？贪婪模式和非贪婪模式的区别？

参考文档

贪婪匹配:正则表达式一般趋向于最大长度匹配，也就是所谓的贪婪匹配。

非贪婪匹配：就是匹配到结果就好，就少的匹配字符。

区别：默认是贪婪模式；在量词后面直接加上一个问号？就是非贪婪模式。

如何使用 python 删除一个文件或者文件夹？

```
import os
import shutil
os.remove(path) # 删除文件
os.removedirs(path) # 删除空文件夹
shutil.rmtree(path) # 删除文件夹，可以为空也可以不为空
```

logging 模块的作用以及应用场景

logging 模块定义的函数和类为应用程序和库的开发实现了一个灵活的事件日志系统。
记录日志

json 序列化时可以处理的数据类型有哪些？如何定制支持 datetime 类型？序列化时，遇到中文转成 unicode，如何保持中文形式？

1、 可以处理的数据类型是 string、int、list、tuple、dict、bool、null

2、 通过自定义时间序列化转换器

```
import json
from json import JSONEncoder
from datetime import datetime
class ComplexEncoder(JSONEncoder):
def default(self, obj):
if isinstance(obj, datetime):
return obj.strftime(  '%Y-%m-%d %H:%M:%S  ')
```

```
else:
return super(ComplexEncoder,self).default(obj)
d = {  'name ': 'alex ', 'data ':datetime.now()}
print(json.dumps(d,cls=ComplexEncoder))
# {"name": "alex", "data": "2018-05-18 19:52:05"}
```

3、 使用 ensure_ascii=False 参数

写出邮箱的正则表达式

```
import re
pp=re.compile('[a-zA-Z0-9_-]+@[0-9A-Za-z]+(\.[0-9a-zA-Z]+)+')
if pp.match('1403179190@qq.com'):
print('ok')
```

写 python 爬虫分别用到了哪些模块？分别有什么用?

模块

request，发起请求

pyquery，解析 html 数据

beautifulsoup，解析 html 数据

aiohttp，异步发送请求

框架

pyspider，web 界面的爬虫框架

scrapy，爬虫框架

selenium，模拟浏览器的爬虫框架

sys.path.append('xxx')的作用

添加搜索路径

输入某年某月某日，判断这是这一年的第几天?

```
date=input('请输入某年某月某日，格式：xxxx.xx.xx')
def get_day(date):
days1=[31,28,31,30,31,30,31,31,30,31,30,31]
days2=[31,29,31,30,31,30,31,31,30,31,30,31]
year,month,day = [int(i) for i in date.split('.')]
if year % 400 ==0 or (year % 4==0 and year % 100!=0):
days=days2
else:
days=days1
return sum(days[:month-1])+day
print(get_day(date))
```

简述面向对象的三大特性?

继承，封装和多态

继承：

继承就是继承的类直接拥有被继承类的属性而不需要在自己的类体中重新再写一遍，其中被继承的类叫做父类、基类，继承的类叫做派生类、子类。

封装：

封装就是把类中的属性和方法定义为私有的，方法就是在属性名或方法名前加双下划线，而一旦这样定义了属性或方法名后，python 会自动将其转换为***类名属性名（方法名）的格式，在类的内部调用还是用双下划线加属性名或方法名，在类的外部调用就要用类名***属性名（方法名）。父类的私有属性和方法，子类无法对其进行修改。

多态：

多态就是不同的对象可以调用相同的方法然后得到不同的结果，有点类似接口类的感觉，在 python 中处处体现着多态，比如不管你是列表还是字符串还是数字都可以使用+和*。

什么是鸭子模型?

鸭子类型（英语：duck typing）是动态类型的一种风格。在这种风格中，一个对象有效的语义，不是由继承自特定的类或实现特定的接口，而是由当前方法和属性的集合决定。

super 的作用

当子类中的方法与父类中的方法重名时，子类中的方法会覆盖父类中的方法，那么，如果我们想实现同时调用父类和子类中的同名方法，就需要使用到 super()这个函数，用法为 super().函数名()

mro 是什么?

对于支持继承的编程语言来说，其方法（属性）可能定义在当前类，也可能来自于基类，所以在方法调用时就需要对当前类和基类进行搜索以确定方法所在的位

置。而搜索的顺序就是所谓的「方法解析顺序」（Method Resolution Order，或 MRO）。

什么是 c3 算法?

c3 算法是 python 新式类中用来产生 mro 顺序的一套算法。即多继承的查找规则。

列举面向对象中带双下划线的特殊方法

__new__：可以调用其它类的构造方法或者直接返回别的对象来作为本类的实例。
__init__： 负责类的实例化
__call__：对象后边加括号，触发执行
__str__：print 打印一个对象时。
__doc__：类的注释，该属性是无法继承的。
__getattr__：在使用调用属性（方式、属性）不存在的时候触发
__setattr__：添加/修改属性会触发它的执行
__dellattr__：删除属性的时候会触发
__delete__：采用 del 删除属性时，触发

双下划线和单下划线的区别

1、 "单下划线" 开始的成员变量叫做保护变量，意思是只有类对象和子类对象自己能访问到这些变量；

2、 "双下划线" 开始的是私有成员，意思是只有类对象自己能访问，连子类对象也不能访问到这个数据。

实例变量和类变量的区别

1、 实例变量是对于每个实例都独有的数据

2、 类变量是该类所有实例共享的属性和方法

实例方法、静态方法和类方法的区别

1、 实例方法：第一个参数必须是实例对象，通常为 self。实例方法只能由实例对象调用。

2、 类方法：使用装饰器@classmethod。第一个参数为当前类的对象，通常为 cls。实例对象和类对象都可以调用类方法。

3、 静态方法：使用装饰器@staticmethod。没有 self 和 cls 参数。方法体中不能使用类或者实例的任何属性和方法。实例对象和类对象都可以调用。

isinstance 和 type 的作用

1、 两者都用来判断对象的类型

2、 对于一个类的之类对象的类型判断，type 就不行了，而 isinstance 可以。

```
class A(object):
pass
class B(A):
pass
```

```
ba=B()
ab=A()
print(type(ba)==A) # False
print(type(ab)==A) # True
print(isinstance(ab,A)) # True
print(isinstance(ba,A)) # True
```

使用 with 语句的好处是什么

1、 使用 with 后不管 with 中的代码出现什么错误，都会进行对当前对象进行清理工作。例如 file 的 file.close()方法，无论 with 中出现任何错误，都会执行 file.close （） 方法

2、 只有支持上下文管理器的对象才能使用 with，即在对象内实现了两个方法：**enter**()和 **exit**()

写一个的支持 with 语句的类

参考链接

```
class W(object):
def __init__(self):
pass
def __enter__(self):
print('进入 with 语句')
return self

def __exit__(self,*args,**kwargs):
print('退出 with 语句')
```

```
with W() as w:
print('之前')
print(w)
print('之后')
```

实现一个单例模式。（尽可能多的方法）

```
# 方法一：使用__new__()
import threading
class Singleton(object):
_instance_lock = threading.Lock()
def __init__(self):
pass

def __new__(cls, *args, **kwargs):
if not hasattr(Singleton, "_instance"):
with Singleton._instance_lock:
    if not hasattr(Singleton, "_instance"):
        Singleton._instance = object.__new__(cls)
return Singleton._instance

obj1 = Singleton()
obj2 = Singleton()
print(obj1 is obj2)

# 方法二：使用元类来创建
import threading

class SingletonType(type):
```

```
_instance_lock = threading.Lock()
def __call__(cls, *args, **kwargs):
if not hasattr(cls, "_instance"):
with SingletonType._instance_lock:
    if not hasattr(cls, "_instance"):
        cls._instance = super().__call__(*args, **kwargs)
return cls._instance

class Singleton(metaclass=SingletonType):
def __init__(self):
pass

obj1 = Singleton()
obj2 = Singleton()
print(obj1 is obj2)
```

如何判断一个对象是否可调用？哪些对象是可调用对象？如何定义一个类，使其对象本身就是可调用对象？

1、 使用 callable 函数判断。

2、 可调用对象有 7 类：

3、 用户自定义函数

4、 内置函数

5、 内置方法

6、 方法(定义在类中的函数)

7、类

8、 类实例(如果类中定义了 **call** 方法，那么这个类的实例就是可调用对象)

9、 生成器函数

在类中定义 **call** 方法，实例对象加()是即调用 **call** 的方法

手写一个栈

```
#给一个点，我们能够根据这个点知道一些内容
class Node(object):
def __init__(self,val): #定位的点的值和一个指向
self.val=val #指向元素的值,原队列第二元素
self.next=None #指向的指针

class stack(object):
def __init__(self):
self.top=None #初始化最开始的位置

def push(self,n):#添加到栈中
n=Node(n) #实例化节点
n.next=self.top #顶端元素传值给一个指针
self.top=n
return n.val

def pop(self): #退出栈
if self.top == None:
return None
```

```
else:
tmp=self.top.val
self.top=self.top.next #下移一位，进行
return tmp

if __name__=="__main__":
s=stack()
print(s.pop())
s.push(1)
print(s.pop())
s.push(2)
s.push(3)
print(s.pop())
s.push(3)
s.push(3)
s.push(3)
print(s.pop())
print(s.pop())
print(s.pop())
print(s.pop())
```

使用两个队列实现一个栈

```
class Stack(object):
def __init__(self):
self.queueA=[]
self.queueB=[]
def push(self,node):
self.queueA.append(node)
def pop(self):
```

```
if len(self.queueA)==0:
return None
while len(self.queueA)!=1:
self.queueB.append(self.queueA.pop(0))
self.queueA,self.queueB=self.queueB,self.queueA
return self.queueB.pop()

st=Stack()
print(st.pop())
st.push(1)
print(st.pop())
st.push(1)
st.push(1)
st.push(1)
print(st.pop())
print(st.pop())
print(st.pop())
```

注意上面两个栈的实现方法，第一种的效率高，队列的这种方法效率低

有如下链表类，请实现单链表逆置。

```
class ListNode:
def __init__(self,val):
self.val=val
self.next=None

class Solution:
def reverseList(self,pHead):
if not pHead or not pHead.next:
```

```
return pHead
last=None
while pHead:
tmp=pHead.next
pHead.next=last
last=pHead
pHead=tmp
return last
```

类的加载和实例化过程

1、 在堆内存中生成 class 对象，把静态变量和静态方法加载到方法区，这个堆内存中的 class 对象是方法区数据的入口

2、 静态变量默认初始化

3、 静态变量显式初始化

4、 执行静态代码块

5、 成员变量默认初始化，显示初始化

6、 执行构造函数

手写一个队列

```
class Queue(object):
def __init__(self,size):
self.queue=[]
```

```
self.size=size
def is_empty(self):
return not bool(len(self.queue))
def is_full(self):
return len(self.queue)==self.size
def enqueue(self,val):
if not self.is_full():
self.queue.insert(0,val)
return True
return False
def dequeue(self):
if not self.is_empty():
return self.queue.pop()
return None
s=Queue(2)
print(s.is_empty)
s.enqueue(1)
s.enqueue(2)
print(s.is_full())
print(s.dequeue())
print(s.dequeue())
print(s.is_empty())
```

python 的底层网络交互模块有哪些

socket，urllib，requests，pycurl

简述 OSI 七层协议

为了实现计算机系统的互连，OSI 参考模型把整个网络的通信功能划分为 7 个层次，同时也定义了层次之间的相互关系以及各层所包括的服务及每层的功能。OSI 的七层由低到高依次为：物理层、数据链路层、网络层、传输层、会话层、表示层、应用层，下三层（物理层、数据链路层、网络层）面向数据通信，而上三层（会话层、表示层、应用层）则面向资源子网，而传输层则是七层中最为重要的一层。它位于上层和下层中间，起承上启下的作用。

什么是 C/S 和 B/S 架构

1、 C/S 架构是一种典型的两层架构，其全称是 Client/Server，即客户端服务器端架构，其客户端包含一个或多个在用户的电脑上运行的程序，而服务器端有两种，一种是数据库服务器端，客户端通过数据库连接访问服务器端的数据；另一种是 Socket 服务器端，服务器端的程序通过 Socket 与客户端的程序通信。

2、 B/S 架构的全称为 Browser/Server，即浏览器/服务器结构。Browser 指的是 Web 浏览器，极少数事务逻辑在前端实现，但主要事务逻辑在服务器端实现，Browser 客户端，WebApp 服务器端和 DB 端构成所谓的三层架构。B/S 架构的系统无须特别安装，只有 Web 浏览器即可。

简述 TCP 三次握手，四次挥手的流程。

三次握手

1、 第一次握手：客户端的应用进程主动打开，并向客户端发出请求报文段。其首部中：SYN=1,seq=x。

2、 第二次握手：服务器应用进程被动打开。若同意客户端的请求，则发回确认报文，其首部中：SYN=1,ACK=1,ack=x+1,seq=y。

3、 第三次握手：客户端收到确认报文之后，通知上层应用进程连接已建立，并向服务器发出确认报文，其首部：ACK=1,ack=y+1。当服务器收到客户端的确认报文之后，也通知其上层应用进程连接已建立。

四次挥手

1、 第一次挥手：数据传输结束以后，客户端的应用进程发出连接释放报文段，并停止发送数据，其首部：FIN=1,seq=u。

2、 第二次挥手：服务器端收到连接释放报文段之后，发出确认报文，其首部：ack=u+1,seq=v。此时本次连接就进入了半关闭状态，客户端不再向服务器发送数据。而服务器端仍会继续发送。

3、 第三次挥手：若服务器已经没有要向客户端发送的数据，其应用进程就通知服务器释放 TCP 连接。这个阶段服务器所发出的最后一个报文的首部应为：FIN=1,ACK=1,seq=w,ack=u+1。

4、 第四次挥手：客户端收到连接释放报文段之后，必须发出确认：ACK=1,seq=u+1,ack=w+1。再经过 2MSL(最长报文端寿命)后，本次 TCP 连接真正结束，通信双方完成了他们的告别。

如果已经建立了 TCP 连接，但是客户端突然出现故障了怎么办

TCP 还设有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为 2 小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔 75 秒发送一次。若一连发送 10 个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

什么是 arp 协议

1、 ARP 全称“Address Resolution Protocol”，地址解析协议。

2、 实现局域网内通过 IP 地址获取主机的 MAC 地址。

3、 MAC 地址 48 位主机的物理地址，局域网内唯一。

4、 ARP 协议类似 DNS 服务，但不需要配置服务。

5、 ARP 协议是三层协议。

TCP 和 UDP 的区别

1、 TCP 面向连接（如打电话要先拨号建立连接）;UDP 是无连接的，即发送数据之前不需要建立连接

2、 TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP 尽最大努力交付，即不保证可靠交付

3、 UDP 具有较好的实时性，工作效率比 TCP 高，适用于对高速传输和实时性有较高的通信或广播通信。

4、 每一条 TCP 连接只能是点到点的;UDP 支持一对一，一对多，多对一和多对多的交互通信

5、 TCP 对系统资源要求较多，UDP 对系统资源要求较少。

为什么基于 tcp 协议的通信比基于 udp 协议的通信更可靠

TCP 是面向连接的传输协议，每次都需要建立一个可以相互信任的连接，中间有个三次握手过程。而 UDP 是面向无连接的传输协议，不需要建立安全的连接，

什么是局域网和广域网

1、 局域网（Local Area Network），简称 LAN，是指在某一区域内由多台计算机互联成的计算机组。“某一区域”指的是同一办公室、同一建筑物、同一公司和同一学校等，一般是方圆几千米以内。局域网可以实现文件管理、应用软件共享、打印机共享、扫描仪共享、工作组内的日程安排、电子邮件和传真通信服务等功能。局域网是封闭型的，可以由办公室内的两台计算机组成，也可以由一个公司内的上千台计算机组成。

2、 广域网（Wide Area Network），简称 WAN，是一种跨越大的、地域性的计算机网络的集合。通常跨越省、市，甚至一个国家。广域网包括大大小小不同的子网，子网可以是局域网，也可以是小型的广域网。

3、 两者区别：

4、 范围不同，广域网比局域网广

5、 接口类型不同

6、 速率不同

7、 协议不同

什么是 socket？简述基于 tcp 协议的 socket 通信流程？

socket 通常也称作"套接字"，用于描述 IP 地址和端口，是一个通信链的句柄。

通信流程：

1、 服务端创建一个 ServerSocket 对象,指定端口号,ServerSocket 对象等待客户端的连接请求。

2、 客户端创建一个 Socket 对象,指定主机地址和端口号,向服务端发出连接请求。

3、 服务端接收到客户端的连接请求,建立一条 TCP 连接,再创建一个 Socket 对象与客户端的 Socket 对象进行通信。

4、 服务端和客户端分别创建字节输入流和字节输出流,通过字节输入流获得对方发来的数据,通过字节输出流向对方发送数据。

5、 当一方决定结束通信时,向对方发送结束信息;另一方接收到结束信息后,双方分别关闭各自的 TCP 连接。

6、 ServerSocket 对象停止等待客户端的连接请求。

什么是粘包？出现粘包的原因？

1、 粘包：多个数据包被连续存储于连续的缓存中，在对数据包进行读取时由于无法确定发生方的发送边界，而采用某一估测值大小来进行数据读出，若双方的 size 不一致时就会使指发送方发送的若干包数据到接收方接收时粘成一包，从接收缓冲区看，后一包数据的头紧接着前一包数据的尾。

2、 出现粘包现象的原因是多方面的，它既可能由发送方造成，也可能由接收方造成。

3、 发送方引起的粘包是由 TCP 协议本身造成的，TCP 为提高传输效率，发送方往往要收集到足够多的数据后才发送一包数据。若连续几次发送的数据都很少，通常 TCP 会根据优化算法把这些数据合成一包后一次发送出去，这样接收方就收到了粘包数据。

4、 接收方引起的粘包是由于接收方用户进程不及时接收数据，从而导致粘包现象。这是因为接收方先把收到的数据放在系统接收缓冲区，用户进程从该缓冲区取数据，若下一包数据到达时前一包数据尚未被用户进程取走，则下一包数据放到系统接收缓冲区时就接到前一包数据之后，而用户进程根据预先设定的缓冲区大小从系统接收缓冲区取数据，这样就一次取到了多包数据。

发生粘包现象如何处理?

1、 对于发送方引起的粘包现象，用户可通过编程设置来避免，TCP 提供了强制数据立即传送的操作指令 push，TCP 软件收到该操作指令后，就立即将本段数据发送出去，而不必等待发送缓冲区满；

2、 对于接收方引起的粘包，则可通过优化程序设计、精简接收进程工作量、提高接收进程优先级等措施，使其及时接收数据，从而尽量避免出现粘包现象；

3、 由接收方控制，将一包数据按结构字段，人为控制分多次接收，然后合并，通过这种手段来避免粘包。

IO 多路复用的作用?

I/O 多路复用是用于提升效率，单个进程可以同时监听多个网络连接 IO。

与多进程和多线程技术相比，I/O 多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程，从而大大减小了系统的开销。

什么是防火墙？防火墙的作用是什么？

在互联网上防火墙是一种非常有效的网络安全模型，通过它可以隔离风险区域(即 Internet 或有一定风险的网络)与安全区域(局域网)的连接，同时不会妨碍人们对风险区域的访问。所以它一般连接在核心交换机与外网之间。

防火墙的作用：

1、 过滤进出网络的数据

2、 管理进出访问网络的行为

3、 封堵某些禁止业务

4、 记录通过防火墙信息内容和活动

5、 对网络攻击检测和告警

select、poll、epoll 模型的区别

1、 支持一个进程所能打开的最大连接数

select 的最大连接数大概 32 *32，或者 32*64

poll 本质和 select 没区别，但是它没有最大连接数限制

epoll 大概 10 万左右(1G 的机器)

2、 FD 剧增后带来的 IO 效率问题

select 和 poll 每次调用都会对连接进行线性遍历，所以会随着 FD 的增加会造成遍历速度慢的“线性下降性能问题”

epoll 没有前两个的线性下降的性能问题，但是当 socket 都很活跃的情况下，可能会有性能问题。

3、 消息传递方式

select 和 poll 内核需要将消息传递到用户空间，都需要内核拷贝动作。

epoll 通过内核和用户空间共享一块内存来实现

简述进程，线程，协程的区别以及应用场景?

区别：

1、 线程是资源分配的单位

2、 线程是操作系统调度的单位

3、 进程切换需要的资源很大，效率很低

4、 线程切换需要的资源一般，效率一般(在不考虑 GIL 的情况下

5、 协程切换任务资源很小，效率高

6、 多进程，多线程根据 cpu 核数不一样可能是并行的，但是协程是在一个线程中，所以是并发。）

应用场景

1、 协程：当程序中存在大量不需要 cpu 的操作时，适用协程

2、 计算密集型，用进程。IO 密集型，用线程。

什么时 GIL 锁

1、 即全局解释器锁，

2、 一个时间点只有一个线程处于执行状态。

python 中如何使用进程池和线程池

```
from concurrent.futures import
ThreadPoolExecutor,ProcessPoolExecutor
import os,time,random
from multiprocessing import Pool

def task(n):
print('%s is runing' %os.getpid())
time.sleep(random.randint(1,3))
return n**2

if __name__ == '__main__':
# 多进程方式一
pool2=Pool()
pool2.map(task,range(10))

# 多进程方式二，下面这种多进程和多线程的用法一模一样
```

```
executor=ThreadPoolExecutor(max_workers=3)
futures=[]
for i in range(11):
future=executor.submit(task,i)
futures.append(future)
executor.shutdown(True)
print('+++>')
for future in futures:
print(future.result())
```