练习：

**1:强类型语言和弱类型语言的区别,python是什么类型的语言(强类型,弱类型,解释型,编译型)**

|  |
| --- |
| 强类型语言，要求所有的变量必须先定义后使用，并且指定类型的变量只能接受类型与之匹配的值;弱类型则不需要  编译型语言，在程序运行之前需要先编译成机器可读的二进制形式(C,java,C++)  解释性语言只需要在运行时有个解释器就行需要经过编译(python,PHP,JS) |

**2:is 和 == 的区别**

|  |
| --- |
| is判断id ;==判断值  a = 9999  b = 9999  a == b 结果为True  a is b 结果为False |

**3:进程、线程、协程的区别**

|  |
| --- |
| 进程 进程是一个程序在一个数据集中的一次动态执行过程，可以简单理解为“正在执行的程序”，它是CPU资源分配和调度的独立单位。 进程一般由程序、数据集、进程控制块三部分组成。我们编写的程序用来描述进程要完成哪些功能以及如何完成；数据集则是程序在执行过程中所需要使用的资源；进程控制块用来记录进程的外部特征，描述进程的执行变化过程，系统可以利用它来控制和管理进程，它是系统感知进程存在的唯一标志。 进程的局限是创建、撤销和切换的开销比较大。  线程 线程是在进程之后发展出来的概念。 线程也叫轻量级进程，它是一个基本的CPU执行单元，也是程序执行过程中的最小单元，由线程ID、程序计数器、寄存器集合和堆栈共同组成。一个进程可以包含多个线程。 线程的优点是减小了程序并发执行时的开销，提高了操作系统的并发性能，缺点是线程没有自己的系统资源，只拥有在运行时必不可少的资源，但同一进程的各线程可以共享进程所拥有的系统资源，如果把进程比作一个车间，那么线程就好比是车间里面的工人。不过对于某些独占性资源存在锁机制，处理不当可能会产生“死锁”。  协程 协程是一种用户态的轻量级线程，又称微线程，英文名Coroutine，协程的调度完全由用户控制。人们通常将协程和子程序（函数）比较着理解。 子程序调用总是一个入口，一次返回，一旦退出即完成了子程序的执行。 协程的起始处是第一个入口点，在协程里，返回点之后是接下来的入口点。在python中，协程可以通过yield来调用其它协程。通过yield方式转移执行权的协程之间不是调用者与被调用者的关系，而是彼此对称、平等的，通过相互协作共同完成任务。  协程的特点在于是一个线程执行，与多线程相比，其优势体现在： 协程的执行效率非常高。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。 协程不需要多线程的锁机制。在协程中控制共享资源不加锁，只需要判断状态就好了。 |

**4:GIL锁是什么**

|  |
| --- |
| GIL 锁 全局解释器锁（只在 cpython 里才有在工作中使用的多为cpython,还有其余版本jpython,pypy）  作用：限制多线程同时执行，保证同一时间只有一个线程执行，所以 cpython 里的多线程其实是伪  多线程! |

**5:MySQL常见数据库引擎及比较**

|  |
| --- |
| 主要 MyISAM 与 InnoDB 两个引擎，其主要区别如下：  Mysql5.1之前MyISAM 是默认引擎，5.1之后默认引擎是Innodb  InnoDB 支持事务，MyISAM 不支持，这一点是非常之重要。事务是一种高级的处理方式，如在一  些列增删改中只要哪个出错还可以回滚还原，而 MyISAM 就不可以了；  MyISAM 适合查询以及插入为主的应用，InnoDB 适合频繁修改以及涉及到安全性较高的应用；  InnoDB 支持外键，MyISAM 不支持；  InnoDB 不支持 FULLTEXT 类型的索引；  InnoDB 中不保存表的行数，如 select count() from table 时，InnoDB；需要扫描一遍整个表来  计算有多少行，但是 MyISAM 只要简单的读出保存好的行数即可。注意的是，当 count()语句包含  where 条件时 MyISAM 也需要扫描整个表；  对于自增长的字段，InnoDB 中必须包含只有该字段的索引，但是在 MyISAM 表中可以和其他字  段一起建立联合索引；清空整个表时，InnoDB 是一行一行的删除，效率非常慢。MyISAM 则会重建  表；  InnoDB 支持行锁（某些情况下还是锁整表，如 update table set a=1 where user like '%lee%' |

**6:char和varchar的区别**

|  |
| --- |
| char是不可变长度,varchar是可变长度  定义个char[10]和varchar[10]  分别存python进去  char占用的依旧是10个长度的字符串,varchar占用为6个  尽管char占用的比较多但是char也有其优势,在查找时char查找速度比varchar要快 |

**7:深拷贝和浅拷贝的区别**

|  |
| --- |
| 深拷贝 浅拷贝分别是copy.copy copy,deepcopy  在一维列表中是没有区别的  在二维甚至多维列表中区别就出来了  list\_1 = [1,2,3,[4,5,6]]  list\_2 = copy.copy(list\_1)  list\_3 = copy.deepcopy(list\_1)  当我们改变list\_1的元素时  例如: list\_1[0] = 0  这时候查看list\_2,和list\_3时是没有区别的  我们再改变list\_1[3][1] = 9  这时再查看list\_2和list\_3就可以发现不同了  浅拷贝只拷贝第一层的内容,第二层的依然是引用 |

**8:什么是lambda是函数,在工作中有什么作用**

|  |
| --- |
| lambda是一个函数，但是又不想费神去命名一个函数的场合下使用，也就是指匿名函数。  在工作中往往会碰到需要局部使用的小函数,但是命名又特别麻烦,也有可能导致函数冲突,在简单的函数中使用lambda函数就足够了,能省去命名的麻烦也能达到需要的效果.  例如:list\_1 = [1,2,3,4,7,9,14,51,17]取出其中为3的倍数的数据时  print(list(filter(lambda x:x%3==0, list\_1))) |

**9:如何分析SQL语句的执行计划？怎样查看正在执行的SQL？**

|  |
| --- |
| explain sql  show profile  详情参考:https://www.shixinke.com/mysql/mysql-sql-optimization-with-using-explain-and-show-profile |

**10:git代码切换到上一次提交的状态**

|  |
| --- |
| git reset --hard HEAD?：这里也可以是版本号  git revert HEAD |

**11:什么是SQL注入，如何避免SQL注入**

|  |
| --- |
| 把SQL命令插入到Web表单的输入域或页面请求的查询字符串，欺骗服务器执行恶意的SQL命令。  在使用ORM不会出现sql注入的情况,sql注入多发生于提交原生sql,在Django中使用pymysql或者mysqldb提交原生SQL时会将特殊的sql语句转义,在不使用这些第三方包的情况下提交sql时采用占位符去填充sql语句也可以转义特殊的语句 |

**12:备份远程服务器(112.113.114.115)的数据库(端口为默认端口,帐号root,密码 mysql)到本地/home/python/Desktop下的命令是什么**

|  |
| --- |
| Mysqldump –h{ip:port} –u{username} –p{password} {database} {table} > /home/python/Desktop/back\_up.sql |

**13:如何将已经排序好的列表变为无序列表**

|  |
| --- |
| import random  random.shuffle(list) |

**14:python2和python3的区别,举例几个**

|  |
| --- |
| print在python2中不需要括号,python3需要括号  xrange和range  items和iteritems |

**15:在项目中导包的顺序**

|  |
| --- |
| LEGB:  locals -> enclosing function -> globals -> \_\_builtins\_\_  locals 是函数内的名字空间，包括局部变量和形参  enclosing 外部嵌套函数的名字空间（闭包中常见）  globals 全局变量，函数定义所在模块的名字空间  builtins 内置模块的名字空间 |

**16:POST和GET的区别**

|  |
| --- |
| 数据携带方式  数据携带的大小  安全性  从这几个方面进行分析 |

**17:COOKIE和session的区别**

|  |
| --- |
| 1、cookie 数据存放在客户的浏览器上，session 数据放在服务器上。  2、cookie 不是很安全，别人可以分析存放在本地的 cookie 并进行 cookie 欺骗考虑到安全应当使用 session。  3、session 会在一定时间内保存在服务器上。当访问增多，会比较占用服务器的性能考虑到减轻服务器性能方面，应当使用 cookie。  4、单个 cookie 保存的数据不能超过 4K，很多浏览器都限制一个站点最多保存 20 个 cookie。  5、建议： 将登陆信息等重要信息存放为 SESSION 其他信息如果需要保留，可以放在 cookie 中 |

**18:L = [i for i in range(100)] 获取L中元素的个数的方法,向L中添加元素的方法,计算L个出现的某个参数出现的次数**

|  |
| --- |
| len(L)  L.append(10)  L.count(10) |

**19:判断字典中是否存在这个key的方法**

|  |
| --- |
| key是否存在字典中  “key” in dic  例如 dic\_person = {“name”:”小明”,”age”:20,”gender”:”男”}  判断 dic\_person是否存在name  name in dic\_person |

**20:装饰器的作用以及项目中的应用**

|  |
| --- |
| 装饰器本质上是一个 Python 函数，它可以让其他函数在不需要做任何代码变动的前提下增加额外功能  在flask中用来做路由功能和校验用户是否登录 |

**21:python获取系统当前时间**

|  |
| --- |
| import time  time.localtime() |

**22:获取当前目录下所有文件夹**

|  |
| --- |
| os.listdir() |

**23:写一段代码实现输入和输出的字母大小写翻转;例如输入”AAbbccddEEFF”输出”aaBBCCDDeeff”**

|  |
| --- |
| def fanzhuanstr(str):  temp\_list = []  for i in str:  if i.islower():  i = i.upper()  elif i.isupper():  i = i.lower()  temp\_list.append(i)  new\_str = ''.join(temp\_list)  return new\_str |

**24:写一段代码统计用户输入的字符串中大写字母个数,小写字母个数,数字个数**

|  |
| --- |
| str = str(input(“请输入一串字符串:”))  upper\_num = 0  lower\_num = 0  dig\_num = 0  for i in list(str):  if i.isisdigit():  dig\_num += 1  if i.isupper():  upper\_num += 1  if i.islower():  lower\_num += 1  print(“大写字母个数{}\n\t 小写字母个数\n\t{} 数字个数\n\t{}”.format(upper\_num,lower\_num,dig\_num)) |

**25:写一段代码输出1000以内为7的倍数或者包含7的数字例如’14,17,21’**

|  |
| --- |
| for i in range(1001):  if i%7 ==0 or 7 in str(i):  print(i) |

**26:生成器是什么,迭代器是什么,他们之间有什么区别**

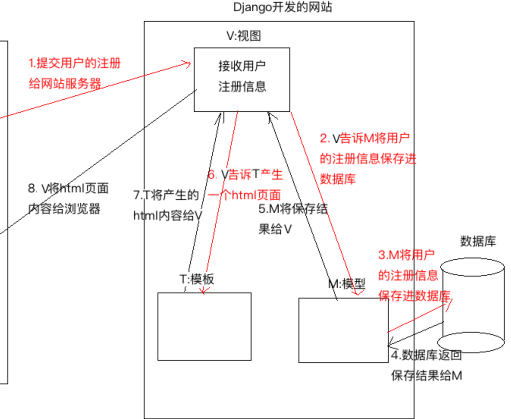
|  |
| --- |
| 迭代器是一个更抽象的概念，任何对象，如果它的类有 next 方法和 iter 方法返回自己本身，对于 string、list、dict、tuple 等这类容器对象，使用 for 循环遍历是很方便的。在后台 for 语句对容器对象调用 iter()函数，iter()是 python 的内置函数。iter()会返回一个定义了 next()方法的迭代器对象，它在容器中逐个访问容器内元素，next()也是 python 的内置函数。在没有后续元素时，next()会抛出一个 StopIteration 异常。  生成器（Generator）是创建迭代器的简单而强大的工具。它们写起来就像是正规的函数，只是在需要返回数据的时候使用 yield 语句。每次 next()被调用时，生成器会返回它脱离的位置（它记忆语句最后一次执行的位置和所有的数据值）  区别：生成器能做到迭代器能做的所有事,而且因为自动创建了 iter()和 next()方法,生成器显得特别简洁,而且生成器也是高效的，使用生成器表达式取代列表解析可以同时节省内存。除了创建和保存程序状态的自动方法,当发生器终结时,还会自动抛出 StopIteration 异常。 |

**27:flask中请求上下文和应用上下文是什么**

|  |
| --- |
| 请求上下文：保存了客户端和服务器交互的数据。  应用上下文：flask 应用程序运行过程中，保存的一些配置信息，比如程序名、数据库连接、应用信息等。 |

**28:MVT的理解**

|  |
| --- |
| MVT是django中的一个设计模式  M: Model，模型，和数据库进行交互  V : View，视图，业务逻辑的处理  T : Template,模板,负责Html页面的产生 |



**29:celery分为哪几块分别执行什么任务**

|  |
| --- |
| 1、任务发布者 发布新任务(短信验证码发送,激活邮件发送)  2、消息队列(broker) 负责任务的调度实现方式有redis,rabbitMQ,rocketMQ,activeMQ等  3、执行者(worker) 以轮询的方式去broker中查询是否有新的任务产生.发现新任务就执行  4、任务执行结果(task result store) 存储执行结果地方 |

**30:你在项目中是否使用过select \* from table.**

|  |
| --- |
| 在开发中最好不要使用select \*,每次使用select \*时都会检索整张表效率十分低下,在开发过程中使用select \* 是毫无开发经验的写法!. |

**31:mysql如何给账户设置权限**

|  |
| --- |
| 先创建用户  # 先连接数据  mysql –uuser -ppassword  # 切换到mysql库中  use mysql;  # 创建用户为python密码为python666的用户  create user python identified by ‘python666’;  # 给项目创建数据库  create databases python default charset=utf8;  # 给账户配置数据库权限  Grant all on python.\* to ‘python’@’%’;  # 刷新权限  flush privileges; |

**32:Git上传代码发生了冲突怎么处理**

|  |
| --- |
| 方案1:根据提示找到冲突的地方,解决冲突之后将冲突的文件或者代码重新add,之后再commit.  方案2:找到产生冲突的文件和同事商量,覆盖他的代码 然后让同事重新commit. |

**33:L = [I for i in range(101)], 取出L中前10个元素,取出倒数10个元素,取出第10到40个元素的倒序并且为双数的.**

|  |
| --- |
| 取前十个 L[0:10]  取后十个 L[-10:]或者L[-10::]  取出第10到40个元素的倒序并且为双数的[i for i in L[40:10:-1] if i%2==0] |

**34:写一段代码实现数学中的阶乘,例如输入5,输出为5\*4\*3\*2\*1**

|  |
| --- |
| def jiecheng(nums):  result = 1  if nums==0:  return(result)  else:  for I in range(1, nums+1):  result = result\*i  return(result) |

**35:请写一个函数函数的输入是一个string类型的英文句子函数的输出也是一个string类型的应为句子：是把输入的句子里面的单词按逆序输出，但是每个单词按内部的字符按顺序输出示例：**

**输入： “I like you, but just I like you.”**

**输出： “.you like I just but, you like I”**

**注意事项：**

**标点符号当作一个单词处理**

|  |
| --- |
| def fanzhuanstr(str):  temp\_list = []  for i in str:  if i.islower():  i = i.upper()  elif i.isupper():  i = i.lower()  temp\_list.append(i)  new\_str = ''.join(temp\_list)  return new\_str |

**36: 给定一个包含 n 个整数的数组nums，判断nums中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？找出所有满足条件且不重复的三元组。**

**Nums = [-1,0,1,2,-1,4]**

|  |
| --- |
| l = []  for a in range(len(nums)):  for b in range(len(nums)):  for c in range(len(nums)):  if (a != b) and (a != c) and (b != c) and (nums[a] + nums[b] + nums[c] == 0):  if sorted([nums[a],nums[b],nums[c]]) not in l:  l.append(sorted([nums[a],nums[b],nums[c]]))  print(L) |

**37:二维列表去重[[1,2,3],[3,2,1],[1,2,4],[4,5,6],[2,1,4]]去重后得到[[1,2,3],[1,2,4],[4,5,6]]**

|  |
| --- |
| sorted(list(list(t) for t in(set([tuple(sorted(t)) for t in l])))) |

**38：说明简单下MVT和MVC的区别，及Django的特点。**

|  |
| --- |
| MVC（Model、View、Controller）的简写，是一种广泛使用的软件架构模式。核心思想是分工、解耦，让不同的代码块之间降低耦合，增强代码的可扩展性和可移植性，实现向后兼容。  M，封装了对数据库层的访问，能对数据库中的数据进行增、删、改、查操作。  V, 用于封装结果，生成页面展示的html内容。  C，用于接收请求，处理业务逻辑，与Model和View交互，返回结果。    Django主要有两大特点：  \* 重量型   Django提供了众多用于方便开发的功能组件，如数据库ORM、模板、表单、Admin站点等  \* MTV结构模式   Django使用MVT结构模式构件整个工程项目，   M - Model层 用于数据库操作   V - View层 用于视图编写   T - Template层 用于页面展示   采用这种结构，可以让代码解耦分工，增强代码的可扩展可移动的灵活性。 |

**39：下面需求使用类视图完成。**

**1. 在Django中使用MySQL数据库，创建meiduo数据库，**

**2. 定义商品的模型类，商品字段包含：名字、价格、评论量、销售量、描述信息这几个字段。**

**3. 完成数据迁移，**

**4. 实现增加商品的功能。**

**5. 把增加的商品信息以json形式返回。**

代码：

settings.py中关于数据库的配置

|  |
| --- |
| DATABASES = {  'default': {  'ENGINE': 'django.db.backends.mysql',  'HOST': '127.0.0.1', # 数据库主机  'PORT': 3306, # 数据库端口  'USER': 'root', # 数据库用户名  'PASSWORD': 'mysql', # 数据库用户密码  'NAME': 'meiduo' # 数据库名字  }  } |

models.py

|  |
| --- |
| from django.db import models  class Goods(models.Model):  '''商品模型类'''  name = models.CharField(max\_length=60, verbose\_name='商品名字')  price = models.DecimalField(max\_digits=10, decimal\_places=2, verbose\_name='商品的价格')  comment = models.IntegerField(verbose\_name='评论量', default=0)  sales = models.IntegerField(verbose\_name='销售量', default=0)  descriptions = models.CharField(verbose\_name='描述信息', max\_length=120, null=True)  class Meta:  db\_table = 'tb\_goods'  verbose\_name = '商品信息'  verbose\_name\_plural = verbose\_name  def \_\_str\_\_(self):  return self.name |

views.py

|  |
| --- |
| from .models import Goods  from django.http import HttpResponse, JsonResponse  from django.views import View  class GoodsView(View):  def post(self, request):  '''增加商品'''  # 获取前端来的json数据  json\_str = request.body  json\_str = json\_str.decode()  req\_data = json.loads(json\_str)  # 添加数据库  good = Goods.objects.create(  name=req\_data.get('name'),  price=req\_data.get('price')  )  # 返回  return JsonResponse({  "id":good.id,  "name":good.name,  "price":good.price,  "sales":good.sales,  "descriptions":good.descriptions,  "comment":good.comment  }) |

urls.py

|  |
| --- |
| from django.conf.urls import url  from . import views  urlpatterns = [  # 添加商品  url(r'^goods/$', views.GoodsView.as\_view()),  ] |

**40：DRF 中的Request和Response与Django中的HttpRequest、HttpResponse有什么区别。**

|  |
| --- |
| REST framework 传入视图的request对象不再是Django默认的HttpRequest对象，而是REST framework提供的扩展了HttpRequest类的Request类的对象。  REST framework 提供了Parser解析器，在接收到请求后会自动根据Content-Type指明的请求数据类型（如JSON、表单等）将请求数据进行parse解析，解析为类字典对象保存到Request对象中。  Request对象的数据是自动根据前端发送数据的格式进行解析之后的结果  DRF中的Response用来构造响应对象，它可以根据请求头中的Accept（接收数据类型声明）来自动转换响应数据到对应格式， 如果前端请求中未进行Accept声明，则会采用默认方式处理响应数据。 |