python基础

python与c的区别 （解释型动态强类型和编译型静态强类型）；

变量 （赋值）；

数字类型 （整型和浮点型）；

字符串 （普通字符）；

列表 （普通容器）；

元组 （不可修改）

字典 （键值对）；

集合 （数据去重）；

格式化输出%和format （占位符）；

if...elif...else （条件判断）；

while循环 （普通循环）；

for循环 （迭代循环）；

可变 | 不可变数据类型 （赋值时引用对象 | 值）；

is和\=\=的区别 （is比较对象而\=\=比较值）；

字符编码 （写和读的编码一致）；

open文件 （需手动关闭文件）；

with...open文件 （自动关闭文件）；

定义函数 （函数的定义）；

调用函数 （函数的调用）；

位置参数 （普通参数）；

默认参数 （默认形参）；

关键字参数 （关键字传参）；

可变长参数 （\\*args和\\*\\*kwargs）；

名称空间 （装变量名的内存空间）；

作用域 （内置-局部-全局）；

闭包函数 （返回外部函数变量）；

无参装饰器 （普通装饰器）；

有参装饰器 （可以传参的装饰器）；

迭代器 （含有\_\_iter\_\_）；

生成器 （含有\_\_iter\_\_和\_\_next\_\_）；

面向函数编程 （先干什么再干什么）；

三元表达式 （结果...if...else）；

列表推导式 （[结果...for...in...]）；

生成器表达式 （ （结果...for...in...））；

递归 （栈）；

匿名函数 （没有名字的函数）；

内置函数 （python自带函数）；

工厂函数 （创建类实例的函数）；

内建函数 （具有某个功能的函数）；

模块与包 （python文件和含有\_\_init\_\_的文件夹）；

import... （导入模块）；

from...import... （导入模块）；

time和datetime模块 （时间）；

random模块 （随机）；

os模块 （系统环境）；

sys模块 （python环境）；

shutil模块 （复制粘贴）；

json和pickle模块 （序列化与反序列化）；

hashlib模块 （哈希算法）；

suprocess模块 （子进程）；

logging模块 （日志）；

re模块 （正则匹配）；

PIL模块 （画图）；

ByteIO （二进制传输数据）；

面向对象编程 （扩展性强）；

类 （class定义的一个类）；

实例对象 （类的实例）；

继承 （继承父类的属性和方法）；

继承的重用性 （可以使用父类的属性或方法）；

多重继承 （继承多个父类）；

菱形 （钻石）继承问题 （经典类深度有限而新式类广度优先）；

派生 （增加或重写类没有的属性和方法）；

抽象类 （子类必须实现绑定@abstractmethod的方法）；

super （）.\_\_init （继承父类属性）；

多态 （一类事物的多种形态）；

多态性 （不同的实例对象有相同的方法）；

封装 （\_类名\_\_属性名）；

property （方法变属性调用方式）；

classmethod （类绑定方法， 自动传第一个值）；

staticmethod （把方法变成普通函数）；

type和isinstance和issubclass （type不考虑继承关系 | isinstance考虑继承关系 | issubclass判断是否是子类）；

反射/自省 （字符串操作对象）；

hasattr | getattr | setattr | delattr （字符串判断是否存在 | 获取 | 修改 | 删除对象的属性或方法）；

\_\_getattribute\_\_ （不论属性是否存在都调用）；

\_\_getattr | setattr | delattr\_\_ （调用属性 | 赋值属性 | del删除属性触发）；

\_\_slots\_\_ （限制初始化属性）；

\_\_geitemt | setitem | delitem\_\_ （中括号取值 | 赋值 | del删除值）；

\_\_str | repr\_\_ （打印触发）；

\_\_format\_\_ （自定义格式化）；

\_\_next | iter\_\_ （实现迭代器）；

\_\_doc\_\_ （描述信息）；

\_\_module | class\_\_ （在哪个模块 | 类）；

\_\_del\_\_ （回收资源）；

\_\_call\_\_ （类实例化时调用）；

元类 （控制类的生成包括类名和基类和类的名称空间）；

\_\_new\_\_ （在类实例化之前调用）；

单例模式 （使用\_\_new\_\_）；

try...except （捕捉特定异常）；

try...except.Exception （捕捉所有异常）；

try...except...finally （一定会执行finally下代码）；

自定义异常 （继承BaseException）；

交叉赋值；

解压容器类型；

range （）和xrange （）；

三种不赋值交换a和b的方法；

print底层实现原理；

with...open的底层实现；

read和readline和readlines的区别；

赋值和浅拷贝和深拷贝；

sort和sorted的区别；

\_\_init\_\_和\_\_new\_\_的区别；

os模块和sys模块的区别；

单元测试编写；

进制转换和数据类型转换；

字典和json的区别；

python为什么没有自增；

保证列表顺序的去重；

python垃圾回收机制和分代回收；

函数传值的机制；

map函数和reduce函数的区别；

回调函数如何通信；

正则中match和search的区别(match位置0匹配，search在文本内匹配)；

网络编程

C/S架构设计 （客户端/服务端架构）；

osi七层协议 （）；

soket层 （应用层与TCP/IO协议族中间的抽象层）；

基于TCP的套接字 （数据传输后保存）；

基于UDP的套接字 （数据传输后不保存）；

粘包问题解决 （加上请求头）；

socketserver实现并发 （server类解决链接和request类解决通信）

并发编程

多道技术 （单核实现并发）；

串行 （完成一个再运行下一个）；

并发 （看起来同时运行）；

并行 （同时运行）；

同步 （没得到计算结果不返回）；

异步 （没得到结果去运行另一个任务）；

阻塞 （无调用结果遇到IO当前线程会被挂起）；

非阻塞 （无调用结果也会立刻返回）；

进程的三种状态 （运行和就绪和阻塞）；

python实现进程 （multiprocessing模块）；

多进程 （多核CPU同时运行）；

僵尸进程 （有害子进程退出父进程没有调用waitpid）；

孤儿进程 （无害父进程退出子进程还在运行）；

守护进程 （等主进程运行完毕后回收）；

IPC通信机制 （队列 | 管道 （了解）解决数据共享出错问题）；

进程信号量 （锁的个数）；

事件 （主线程控制其他线程）；

进程池 （控制最大并发数）；

python实现线程 （threading模块）；

多线程 （单核CPU同时运行）；

守护线程 （等主线程运行完毕后回收）；

GIL锁 （解释器级别锁控制线程间数据共享造成伪线程）；

CPU密集型 | 计算密集型 （多进程）；

IO密集型 （多线程）；

互斥锁 （拿到锁才能运行）；

死锁 （两个进程或线程互相等待）；

递归锁 （线程拿到资源计数该线程计数不为0其他线程不能拿到资源）；

线程信号量 （锁的计数器）；

线程定时器 （指定n秒后执行其他操作）；

线程队列 （线程运行先进先出）；

线程安全 （locals定义一个全局对象该全局对象不会被一起调用）；

python实现协程 （gevent模块greenlet）；

协程 （单线程并发即函数回调修改共享数据不加锁）；

poll （不断地去询问任务有没有就绪）；

epoll （任务就绪通过回调函数通知任务）；

三次握手四次挥手 （不进行数据交互所以只要三次握手）

MySQL

mysql之show；

select；

distinct；

limit；

order by；

where；

like；

regexp；

concat （计算字段）；

函数；

group by （组合）；

having；

子查询；

insert；

inner join内联结；

left （right） outer join外联结；

union组合查询；

全文本搜索 （MyISAM引擎）；

update更新数据；

delete删除数据；

视图 （虚拟表）；

存储过程 （保存MySQL语句的集合）；

游标 （存储过程结果集）；

触发器 （满足条件触发）；

事务处理 （批量语句操作）；

改善数据库性能 （不只有一种编写SQL的select语句， 可以多尝试联结、并或子查询等找出最佳的方法 | 应该总是使用正确的数据类型， 尽量少用select \*， 即不要查询比需求多的数据 | 使用union比使用多个or会好很多 | 索引虽然增加数据检索的性能， 但损害数据插入、删除和更新的性能 | 最好使用fulltext而不是like | 理想的优化和配置会随着数据库的更新而更新 | 最重要的是每条规则在某些条件下都会被打破， 具体情况具体分析）

```

import MySQLdb

db = MySQLdb.connect （'localhost'， 3306， 'root'， '123'， 'TESTDB'， charset='utf8'）

cursor = db.cursor （） # 相当于游标

cursor.excute （'执行语句'）

data = cursor.fetchone （）

db.close （）

```

**JavaScript**

var声明的是局部变量,不加var是全局变量,可以在代码第一行写上’use strict’必须得写var(为避免变量相互影响,尽量使用);

字符串的用法(toUpperCase,toLowerCase,indexOf(字符串出现的位置),substring(指定索引区间)),

修改数组的length会改变数组的长度,多出来的都是undefined;

数组索引赋值超过数组长度,会赋值成功并且中间会多出undefined;

数组的用法(indexOf,slice,push和pop(末尾添加),unshift和shift(头部),sort,reverse,splice(删除后添加、删除不添加、添加不删除),concat(联立两个数组返回新数组),join,多维数组);

对象的变量名可以不加’’,.号访问,如果包含特殊字符,该变量名需使用’’括起来,访问也得使用[]访问;

do...while循环至少循环一次,while循环可能一次都不循环；

forEach对所有iterable元素有用，返回当前元素的索引和值；

函数传参undefined计算结果为NaN；

函数内部自带一个arguments参数，他会接受所有参数，并且相当于一个array；

变量提升（先扫描函数体内所有声明的变量，但只是让一个后定义的变量提前使用不报错，他还是undefined）；

名字空间（为了变量名冲突，可以把变量名绑定在一个定义为专门放变量名的字典里）；

方法（对象中绑定的函数）；

this（在方法内使用this指向的是该方法的对象，即可引用该对象的其他属性或方法，但是如果只是在对象中引用该方法this的指向还是window，并且如果把对象中的方法赋值给对象外的变量，那么this指向也不是对象；如果在对象方法中再定义一个函数，那么该函数的this指向也不是对象，可以在方法中先把this赋值后传给方法中的函数）；

apply（在调用函数的时候.apply(this要指向的对象,[接收参数的数组])，这样可以保证对象外调用函数this指向是对象而不是window）；

call（类似于apply，但是他不是用数组接收参数，而是按顺序传入参数）；

函数重新指向（JavaScript是动态的，因此它的函数名可以指向其他的函数，类似于python）；

map | reduce | filter | sort（全都类似于python）；

闭包（类似于python）；

typeof（获取对象类型，但无法区分null，array，{}，因为他们都是object）；

.toString（除了undefined和null都可以转成字符串）；

使用if…undefined判断某个变量是否存在；

Date | RegExp（可以new 出一个日期 | 正则对象，使用他们特有的方法）；

Json（JSON.stringify（）序列化和JSON.parse()反序列化）；

xiaoming.\_\_proto\_\_ = Student（xiaoming继承Student对象，类似于python的继承）；

创建对象（自定义一个函数，可以通过new这个函数类实例化出一个对象）；

prototype（暂且理解为类的独有对象，实例化对象无法调用，有兴趣自己了解去吧）；

原型继承（什么垃圾玩意，自己看去，傻XJavaScript）；

class继承**（ES6）**（和python的class差不多，有了这个就不用去看那上面两个知识点了）；

浏览器对象window | navigator | screen | location | document | history（全局作用域和浏览器窗口 | 浏览器的信息 | 屏幕的信息 | 当前页面的url信息 | 浏览器对象 | 浏览器历史记录）；

操作DOM（getElementById通过id获取标签 | getElementsByTagName通过标签名获取标签 | getElementsByClassName通过类名获取标签 | cookie获取cookie信息 | querySelector通过选择器获取标签 | querySelectorAll）；

更新DOM （innerHTML修改标签结构 | innerText修改标签内文本 | .style获取标签的style属性）；

插入DOM （appendChild插入一个子节点到父节点的最后一个子节点 | insertBefore插入到一个节点前）

删除DOM（removeChild先拿到父节点，然后再拿到要删除的子节点，通过父节点删除子节点）

DOM操作表单（value获取表单值 | checked判断选择框是否被选中 | form.submit提交表单也可以通过加个事件返回True或False）；

DOM操作文件（new FileReader().onload文件加载成功后执行）；

Ajax（request.onreadystatechange = function () {}）；

Promise（异步执行？没兴趣了解）；

Canvas（绘制文本的标签，没兴趣了解）；

**ES6新增**

`...`(多行字符串);

`${name}`(占位符);

Map（一个键只能对应一个值，查找速度非常快，键值对结构）；

set（没有值的对象）；

for…of循环元组内原有的元素，新增的不会循环出来；

function foo(a, b, ...rest)，如果传多了会被rest接受，如果传少了会接受undefineds；

let（声明一个块级作用域）；

const(定义常量)；

解构赋值（相当于python的解压缩）；

箭头函数（x => x\*x，类似于python匿名函数）；

generator（类似于python的yield生成器）；

class继承（ES6）（和python的class差不多）；

JQuer中Ajax模板

```

<script>

$ （""）.click （function （）{

$.ajax （{

url:""；

type:""；

data:；

success:；

error: ；

complete: ；

statusCode: ；

}）

}）

</script>

```

Vue

导入；

el挂载；

data数据传递；

methods方法函数；

computer计算属性；

多个属性修改一个属性间接改；

watch监听数据；

一个属性修改多个属性间接修改；

delimiters属性；

修改数据的引用方式；

生命周期钩子 （一般用created和mounted是vue挂载）；

钩子 （提前准备未来在满足某个条件会被触发的事件多为系统调用）；

vue的视图相关操作；

{{ }}数据的引入；

v-model数据双向绑定；

v-text文本绑定；

v-bind绑定属性 （缩写' :属性名'）；

v-once只导入一次属性；

v-cloak消除页面刷新闪烁或将script导入vue放入head内部；

v-on绑定事件名 （缩写' @事件名'；）；

v-html可以写html文本；

v-bind绑定系统预定义属性 （绑定字符串， 以数组绑定多个变量， 以对象绑定多个变量， 绑定多个属性）；

v-on绑定系统预定义事件 （@click="fn2" | @click="fn3 （msg， 88888）" | @click="fn4 （$event， msg）"）；

v-model双向绑定多针对于表单/可输入元素 （存储的值为输入框的value值 | 存储的是单选框的默认值 | 存储的是单一复选框的true | false值 | 存储的是多选框的默认值）；

v-show取值为true | false但是display=none；

v-if | v-else-if | v-else:取值为true | false不会渲染到DOM上；

v-for （迭代数组 | 迭代对象 | 迭代复杂数据 （v-for时key值只要是惟一的值即可建立缓存进行优化））；

根组件中使用components注册局部组件；

组件只能解析一个根标签；

组件内部用data.function （{}）返回所需要的数据；

即一个组件的数据修改另一个组件的数据并不会受影响；

父向子组件通信属性绑定 （数据是父组件的， 子组件props通过绑定数据的属性来接收属性对应的数据）；

子向父组件通信事件绑定 （子组件用$emit通过事件把数据发送给父组件父组件通过事件接收数据）；

组件todolist （增加父向子通信而删除子向父通行）；

Rabbitmq

import pika；

为啥要使用MQ （将一些不需要立即生效的操作拆分出来异步执行，消息持久化）；

队列 | 生产者 | 消费者 （队列是RabbitMQ的内部对象 | 生产者生产消息投递到队列中 | 多个消费者可以从队列中获取消息消费）；

为什么不直接使用queue而使用rabbitmq （生产者可以通过交换器把生产的多条消息放入不同的队列中）；

避免单条消息丢失 （在生产者端加上一个属性使消息永久存在并在消费者端需要加上确认消息代码）；

消息队列宕机 （队列在第一次声明的时候就需要持久化途中设置回报错）；

消息的能者多劳 （在消费者端配置perfetch=1意味着我还没有处理完消息就不用给我发新消息了）；

exchange发布消息的四种类型 （headers交换机匹配消息的头部字节已经被舍弃 | direct单播交换机消息中的路由键和队列名完全匹配则只把该消息发送到该路由中 | fanout广播交换机消息会发送到绑定在交换机上的所有队列上 | topic通过模式匹配类似正则匹配的方式分配消息的路由属性）

git

git init （添加文件）；

git status （查看文件状态）；

git diff （查看修改内容）；

git rm （删除文件）；

git add （把文件保存在暂存区）；

git commit -m （提交到版本库，必须先add）；

git checkout -- file （丢掉工作区的修改）；

git reset --hard commit\_id （回退暂存区文件）；

git log （查看提交历史）；

git reflog （查看命令历史）；

git pull (抓取远程库的新提交，即更新本地库)；

git push -u origin master （推送或更新分支所有内容，最好使用git pull先抓取远程的新提交）；

git remote add origin git@server-name:path/repo-name.git （关联远程仓库）；

git clone （克隆一个仓库）；

git remote -v （查看远程库的信息）；

git branch （查看分支）；

git branch <name> （创建分支）；

git checkout <name> （切换分支）；

git checkout -b <name> （创建+切换分支）；

git merge <name> （合并某分支到当前分支）；

git branch -d <name> （删除分支）；

git log --graph （查看分支合并图）；

git branch -D <name> （前行删除分支）；

合并分支的模式 （合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。）；

git stash （暂停工作任务）；

git stash pop （回退工作任务）；

git tag <tagname> （创建标签）；

git tag -a <tagname> -m "blablabla..." （指定标签信息）；

git tag （查看所有标签）；

git push origin <tagname> （推送一个本地标签）；

git push origin --tags （推送全部未推送过的本地标签）；

git tag -d <tagname> （删除一个本地标签）；

git push origin :refs/tags/<tagname> （删除一个远程标签）；

Redis

Redis和memcached的区别 （redis单线程单进程而memcached只存储字符串）；

Redis的常见操作 （操作时不关心数据类型的一些）；

字符串操作；

链表操作；

集合set操作；

有序集合orderset操作；

Hash数据结构操作 （相当于python字典）；

Python操作Redis；

pipeline操作 （事务操作）；

统计页面点击数/控制用户登录发送验证码 （INCR自增）；

如何保存一个对象数据 （哈希键值对）；

社交圈子数据 （集合并集）；

反垃圾系统 （集合去重）；

Pub/Sub构建实时消息系统 （链表blpop （没有消息会等待一段时间）操作实现长轮询Ajax）；

```

import redis

r=redis.Redis （host='localhost'， port=6379， password=''， db=0）

# 连接池的redis启动

pool = redis.ConnectionPool （host='127.0.0.1'， port=6379， max\_connections=100）

r = redis.Redis （connection\_pool=pool）

p.execute （）

```

Memcached

减轻主库压力，保存如cookie之类的信息；

get；

set；

delete；

爬虫

request.get （）；

request.post （data=data）；

proxies （设置代理池）；

request.post （files=files （打开的文件））；

request.session （操作带有cookie的网页）；

selenium （模拟浏览器的操作）；

PhantomJS （无界面浏览器）；

selenium和PhantomJS破除浏览器懒加载；

浏览器图片懒加载(图片未加载标签是src2，加载成功后变成src)；’

scrapy框架 （每个spider有自己的名字）；

解析函数获取url回调；

spider爬去的东西通过items解析；

通过爬虫过程自定义保存爬虫文件；

scrapy可以对url设置规则；

Django基础

django简介；

mvc和mvt模式；

django的请求生命周期；

路由控制；

有名分组 （^?P<id>/d+/$）；

路由分发 （include）；

反向解析 （name）；

名称空间 （namespace）；

django2.0的path和repath （类似于flask）；

视图层；

httprequest （封装成了request）；

httpresponse；

render；

redirect；

jsonresponse；

CBV和FBV；

模板层；

模板语法变量；

模板过滤器；

模板标签；

自定义标签和过滤器；

模板导入和继承相关 （extends， include， block）；

静态文件相关模板 （{%load statice %}；

单表操作；

python脚本中调用django环境 （可以用来生成数据）；

django终端打印sql语句；

多表操作；

基于对象的跨表查询 （注意\_set是表查字段）；

基于双下划线的跨表查询；

聚合查询 （aggregate与SUM等函数连用）；

分组查询 （annotate）；

F查询 （进行加减乘除计算）；

Q查询 （逻辑真假判断）；

所有字段和参数；

自定义字段类型；

model进阶；

中介模型 （多对多可以通过through自定义第三张表）；

查询优化 （一对多select\_related而多对多用prefetch\_related类似于depth深度查询）；

extra （加入额外SQL语句）；

整体插入 （bulk\_create一次性插入多条数据）；

ajax上传文件 （表单enctype等于multipart/form-data）；

分页器组件 （paginator=Paginator （book\_list， 10））；

forms组件；

渲染标签的三种方式；

渲染错误信息；

组件的参数配置；

局部钩子；

全局钩子；

cookie与session （cookie在客户端 | session在服务端）；

django中操作cookie；

django中操作session；

django中配置session存储位置；

cbv加装饰器；

中间件；

中间件的作用 （控制httprequest和httpresponse）；

自定义中间件 （先process\_request再路由再process\_view再视图再process\_exception最后process\_response）；

控制视图函数返回值 （process\_template\_response）；

中间件的应用 （ip访问频率和url访问过滤）；

csrf跨站请求伪造解决方法 （ajax | 注释中间件 | 加上csrf\_exempt局部禁用 | protect局部使用装饰器）；

auth模块 （实现用户注册 | 登录 | 认证 | 注销 | 修改密码等操作）；

auth模块常用方法；

扩展默认的auth\_user表；

contenttype组件 （优惠券表连接多个表但是不会生成额外的列）

RESTful

restful （软件架构风格）；

restful规范 （[api设计的10种风格](https://www.cnblogs.com/liuqingzheng/articles/9762022.html)）；

apiview；

序列化组件；

序列化之serializers （serializers.CharField）；

序列化之ModelSerializers；

序列化之请求数据校验和保存；

局部钩子和全局钩子；

视图组件(自定义所有的方法)；

mixin和generice类简写视图CBV(把不需要传参和需要传参的函数分开，传参必须得传id)；

使用ModelViewSet快速生成视图CBV(通过自动化路由把所有函数分装在一起)；

解析器 （根据content-type选择对应的解析器parsers）；

全局使用解析器；

局部使用解析器；

认证组件 （用户满足某种条件才允许干其他的事authentication）；

全局使用认证组件；

局部使用认证组件；

自定义认证组件；

权限组件 （对用户分类不同的用户做不同的事permissions）；

全局使用权限组件；

局部使用权限组件；

频率组件 （控制用户对某个url请求的频率throttling）；

自定义频率类；

全局使用频率组件；

局部使用频率组件；

分页器 （pagination）；

简单分页 （PageNumberPagination）；

偏移分页 （LimitOffsetPagination）；

加密分页 （CursorPagination保存访问的页码只能访问上下一页）；

响应器 （渲染器如.json和renderers）；

全局使用响应器；

局部使用响应器；

url控制器；

全自动路由 （自动生成路由include （router.urls））；

半自动路由 （视图类继承ModelViewSet）；

自定义路由 （原始方式）；

版本控制 （versioning）；

局部使用版本控制；

全局使用版本控制；

Linux

man [command] （查看command的详细用法）；

pwd （查看当前目录）；

cd ~ | . | .. | - （家 | 本 | 上一级 | 上一次目录）；

mkdir （创建文件夹）；

mkdir b{1...10} （创建多个文件夹）；

mkdir /opt/test/oldboy/a -p （创建多层目录不加则创建2层）；

rmdir file （删除文件）；

rm -rf （删除含有子文件的文件）；

ls -al | -l | -a | lt | -h （显示隐藏文件并列显示 | 列显示文件 | 显示隐藏文件 | 排序显示文件 | 显示文件大小）；

ll （相当于ls -l）；

文件第一个字符代表的文字 （·普通文件 | d目录 | l软链接）；

touch filename （创建文件）；

touch filename{1...10} （创建多个文件）；

rm -f | -rf （删除所有文件不删除目录 | 删除所有文件和目录）；

vi | vim filename （打开可编辑文件）；

a | o | u | dd （文件快捷键追加 | 换行写 | 撤销 | 删除一行）；

:q | :wq | :q! （保存 | 保存并退出 | 强制退出文件）；

:set nu （文件内显示行号）；

[num]+g （切换到num行）；

shift+G （切换到最后一行）；

/[string] （匹配要查找的内容）；

/[string] （匹配要查找的所有内容）；

cat [file] （查看文件file）；

cat -n [file] （查看带有行号的文件file）；

echo 'aaa' | >[file] | >>[file] （把'aaa'输出到屏幕 | 生成file文件并写入'aaa' | 追加写入file文件）；

aaa 2> | >> b （aaa的错误会重定向到b | 错误追加重定向到b）；

cp [file] [director] （复制文件file到director目录下）；

cp [file] [newfile] （同目录下复制文件）；

cp -arp [director] [otherdirector] （复制目录）；

mv [filename] [newfilename] （移动文件）；

rename c文本 e文本 c （把c文件改成e文本）；

alias （别名当前登录有效）；

unalias [command] （临时取消command的别名）；

find . -type | f | d | l | name | atime | +7 | size | delete （查找所有文件 | 隐藏文件也会查询 | 目录 | 软链接 | 匹配 | 时间 | 7天前的 | 大小 | 删除）；

rm -rf\`find . -type d -name '1' -atime -7' （通过返回找到文件给rm删除）；

less | more | head 5 | tail 5 （查看文件少于多少 | 多余多少 | 前5行 | 后5行）；

grep 'matchcontent' file1 file2 -e -v -n （从file1和file2过滤出带有matchcontent的行 | 使用正则 | 取反 | 显示行）；

sed '3；

5p' file -n （取3-5行 | 安静模式）；

sed '3iaaa' file （在第三行前位置插入aaa）；

sed '3abbb' file （在第三行追加）；

sed '3d' file （删除）；

sed '3c' file （替换）；

sed -i '3caaa' file （不加-i只会输出文件不会替换）；

sed 's#root#oldboy#g' file （把root换成oldboy-可以正则）；

awk 'BEGIN{} pattern {command} END{print "end"}'；

awk -F ':' 'NR==2 | NR2，NR5{print $1 | $NF | （NF-1）}' file （以:分割文件取出第二行 | 2-6行的第一列 | 最后一列 | 倒数第二列）；

seq 10 | awk 'BEGIN {sum=0；

print"1+2+...+10="}{sum+=$1}END{print sum}' （计算1+...+10=55）；

awk 'BEGIN {h=0}{h+=1}END{print h}' file | awk 'END{print NR}' file （计算文件多少行）；

grep （文本搜索过滤和正则）；

sed （取行和文本替换）；

awk （取列及逻辑运算处理）；

top （任务管理器）；

free -mh （看内存）；

uptime （系统使用率）；

echo '$' （所见即所得）；

echo"$ （会解析变量）；

echo \`\`执行内部命令；

sz file （上传文件）；

lz file （下载文件）；

ps -aux （当前用户下的所有进程）；

kill pid （关掉一个进程）；

pkill servername （关掉一个服务）；

kill -9 pid （无论进程是否在运行都强行关掉）；

tar xf file.tar.gz （解压tar文件）；

unzip file （解压zip文件）；

ln filename newfilename （创建一个文件硬链接）；

ln -s filename newfilename （创建一个文件软链接）；

硬链接相当于同一个文件的别名inode相同；

软链接相当于windows的快捷方式inode不同

Flask

response三剑客 （HttpResponse | render\_template | redirect）；

静态文件存放 （static | templates）；

send\_file | jsonify(return send\_file() | jsonify({k:v}))；

request的方法form | values | data | json | files | save(FormData中数据 | 所有参数 | 无格式的数据 | json格式数据 | 文件数据 | 保存文件)

request的属性method | url | host | path；

模板语言；

路由(methods=[默认方法]， endpoint=’反向解析url地址’， default={‘nid’:默认值})；

路由解析(url\_for(‘endpoint’， arg))；

flask实例化配置(一般配置写两份， 两份DEBUG配置不一样)；

导入类的方式配置(app.config.from\_object(自定义配置类))；

SECRET\_KEY(生成session必备)；

蓝图(另一种形式的flask实例， 可以重新配置静态文件夹以及二级路由)；

flask特殊装饰器；

@app.template\_folder(类偏函数)；

@app.template\_global(全局函数)；

@app.errohandler(自定义错误页面)；

@app.before\_request(请求来之前的判断)；

@app.after\_reqeust(响应后的判断)；

CBV(可以不用写methods， 有什么方法写什么方法)；

MongoDB

使用不存在的对象创建对象；

collections （表）；

field （字段）；

db （查看当前数据库）；

show dbs （查看磁盘中数据库无数据的数据库不会显示）；

use SS2DAY04 （创建数据库）；

show tables （查表）；

db.users （创建users表）；

db.users.insert | insertOne （{name:'cyd'， age:18}） （插入数据， 不推荐使用）；

db.users.update （{'name':'cyd'}{$set:{'age':18}}） （修改数据， 不推荐使用）；

db.users.remove （{'name':'cyd'}） （删除数据， 不推荐使用）；

db.users.find （） （查找）；

数据类型 （）；

数学比较符 （）；

$修改器 （）；

db.users.insertOne （{}） （插入一条， 推荐使用）；

db.users.insertMany （[{}]） （插入多条数据）；

db.users.deleteOne | Many （相当于insert）；

db.users.deleteOne | Many （相当于insert）；

db.users.findOne （只能查出json格式）， db.users.updateOne （{'hobby':'egon'， name:'ywb'}， {$set:{'hobby.$':'dsb'}}） （$用于array并相当于egon）；

db.users.find （{$or:[{age:28}， {name:'ywb'}]}） （$or的使用）；

db.users.find （{age:{$in:[18， 73， 84]}}） （$in的使用）；

db.users.find （{hobby:{$all:[8， 2]}}） （$all内的元素必须是查找对象的子集）；

db.users.updateMany （{$set:{cursours:0}}） （$set修改 | $unset删除 | $inc | $push | $pull | $pop等修改器只能用于update中）；

db.users.find （）.limit （2） （显示2条）；

db.users.find （）.sort （{age:-1}） （按age倒排序）；

db.users.find （{}）.skip （0）.limit （2） （跳过第一页后一次拿两个）；

先sort （）再skip （）再limit （）；

```

import pymongo

mongo = pymongo.MongoClient （'127.0.0.1'， 27017）

MONGODB = mongo['SS2DAY04']

MONGODB.users.find （{}）

```

websocket

轮询(客户端循环请求服务端)；

长轮询(一段时间不返回服务器断开， 然后客户端马上重连)；

websocket(类似于socket，服务端和客户端建立连接后一个只负责发， 一个只负责收)；

加密后解密消息才能真正的达到传输数据的目的；

有状态的；

数据结构

单链表 （头部插入方便，查找删除都需要遍历所有元素）；

[插入和删除为什么依然选择链表 （顺序表插入和删除需要移动后面或者前面的所有元素）；](https://www.cnblogs.com/chenyoude/p/9898779.html)

双链表 （删除时给的是值需要遍历后删除，但给的是节点，删除节点就行了）；

双端链表()；

[队列的三种实现方式和双端队列；](https://www.cnblogs.com/chenyoude/p/9898982.html)

优先级队列；

[栈；](https://www.cnblogs.com/chenyoude/p/9899015.html)

[哈希表；](https://www.cnblogs.com/chenyoude/p/9899577.html)

[哈希表-字典，哈希表-集合；](https://www.cnblogs.com/chenyoude/p/9899721.html)

[二叉查找树；](https://www.cnblogs.com/chenyoude/p/9900186.html)

[堆；](https://www.cnblogs.com/chenyoude/p/9900760.html)

[图；](https://www.cnblogs.com/chenyoude/p/9900893.html)

算法

[汉诺塔问题；](https://www.cnblogs.com/chenyoude/p/9899913.html)

线性查找(循环遍历)；

折半查找(折半再折半)；

斐波那契数列；

[排序算法总结；](https://www.cnblogs.com/chenyoude/p/9901074.html)

冒泡算法()；

选择排序；

插入排序；

快速排序；

归并排序；

希尔排序；

堆排序；

C程序设计

基本语法；

数据类型；

变量；

常量 （#define WIDTH 5）；

存储类 （auto （局部变量） | register （寄存器内变量） | static （本文件全局变量） | extern （其他文件也可访问变量））；

运算符；

判断 （swtich）；

循环 （goto...调到那个位置 | do...while）；

函数 （）；

数组 （多维数组）；

枚举 （enum DAY { MON=1， TUE， WED， THU， FRI， STA， SUN } day； ）；

指针 （变量的内存地址）；

函数指针；

回调函数；

字符串；

结构体 （struct Books { char title[50]； int book\_id； } book； ）；

共用体 （union Data { int i； char str[20]； } data； ）， 同一时间只能访问其中一个成员；

typedef （给类型或数值定义别名）；

输入&输出 （printf | scanf）；

预处理器；

强制类型转换运算符；

错误处理 （perror （））；

内存管理 （\*calloc （动态分配内存） | free （释放动态分配的内存） | \*malloc （分配指定大小内存） | \*realloc （重新分配内存））；

命令行参数 （argc （指定传入个数） | argv[]， 相当于python的argv）；

MUI

app web前端 布局框架；

UNI-APP vue.js(最新的前端布局框架)；

HTML5PLUS 封装移动操作系统的系统调用以及对硬件的操控；

补充

uwsgi = nginx + wsgi；

nginx （接收请求给wsgi） --> wsgi （序列化请求信息给app） --> app （处理请求信息）；

python中dir （） （看对象的所有属性）；

d={'a':1} # type:Dict 通过type可以把d定义为字典， 之后可以.出字典的方法；

f ’string{insert}’(等同于’string{insert}’.format())；

秒杀系统设计(加上乐观锁，就是加一个事务，该事务的作用是在用户秒杀成功之前判断之前查询的数据库信息和现在查询的数据库信息是否匹配，如果匹配成功返回秒杀成功，否则回退事务，即秒杀失败)；

偏函数（使用partial给一个函数传值，让这个函数变成一个被传值的新函数）；

u’’(unicode编码字符串);