目 录

[Linux部分 3](#_Toc517680867)

[Day01 3](#_Toc517680868)

[Day02 7](#_Toc517680869)

[Python部分 14](#_Toc517680870)

[Day03 14](#_Toc517680871)

[Day04 20](#_Toc517680872)

[Day05 25](#_Toc517680873)

[Day06 30](#_Toc517680874)

[Day07 33](#_Toc517680875)

[Day08 37](#_Toc517680876)

[Day09 41](#_Toc517680877)

[Day10 44](#_Toc517680878)

[Day11 50](#_Toc517680879)

[Day12 56](#_Toc517680880)

[Day13 59](#_Toc517680881)

[Day14 67](#_Toc517680882)

[Day15 70](#_Toc517680883)

[Day16 74](#_Toc517680884)

[Day17 78](#_Toc517680885)

[Day18 83](#_Toc517680886)

[Day19 87](#_Toc517680887)

[Day20 92](#_Toc517680888)

[Day21 96](#_Toc517680889)

[Day22 99](#_Toc517680890)

[MySQL部分 104](#_Toc517680891)

[Day23 104](#_Toc517680892)

[Day24 111](#_Toc517680893)

[Day25 116](#_Toc517680894)

[Day26 122](#_Toc517680895)

[网络编程部分 127](#_Toc517680896)

[Day27 127](#_Toc517680897)

[Day28 132](#_Toc517680898)

[Day29 136](#_Toc517680899)

[Day30 138](#_Toc517680900)

[Day31 143](#_Toc517680901)

[Day32 144](#_Toc517680902)

[Day33 148](#_Toc517680903)

[Day34 152](#_Toc517680904)

[Day35 154](#_Toc517680905)

[MongoDB部分 155](#_Toc517680906)

[Day36 155](#_Toc517680907)

[Day37 162](#_Toc517680908)

[Day38 166](#_Toc517680909)

[Day39 174](#_Toc517680910)

[Day40 176](#_Toc517680911)

[Day41 183](#_Toc517680912)

[Day42 185](#_Toc517680913)

[Day43 186](#_Toc517680914)

[Web部分 188](#_Toc517680915)

[Day44 188](#_Toc517680916)

[Day45 194](#_Toc517680917)

[Day46 198](#_Toc517680918)

[Day47 203](#_Toc517680919)

[Day48 208](#_Toc517680920)

[Day49 212](#_Toc517680921)

[Day50 215](#_Toc517680922)

[Day51 219](#_Toc517680923)

[Day52 223](#_Toc517680924)

[Day53 225](#_Toc517680925)

[Day54 227](#_Toc517680926)

[Day55 231](#_Toc517680927)

[Day56 235](#_Toc517680928)

[Day57 238](#_Toc517680929)

# Linux部分

## Day01

魏明择

Email :weimz@tedu.cn

QQ:88126169

项目经理加入方法：

AID1803 +姓名（所在地）

我是AID1803魏明择（北京）

０１年　哈电集团（６个月国企）

500kv

０２年３月　黑龙江电力集团PLC（可编程逻辑控制器）

０４年文曲星（电子辞典）《英雄传说》

０５年诺亚舟（网络学习机）

０８年医疗电子（电子血压，心电监护仪）

１２年　ＩＴ教育

精通语言：

c c++ Python

熟悉的语言：

Java objective-c fortran77-90 basic pascal

三大操作系统：

UNIX

AIX(IBM)

Solaris(SUN/Oracle)

MAC OS X(Apple)

IOS(Apple)

Linux(当前教学)

Ubuntu:

用户名：tarena 密码：tarena

Redhat

CentOS

Android(安卓)

windows

win10 win7 winxp win98 win95 win3.2

计算机的组成：

硬件

处理器（ｃｐｕ）

运行内存（ｒａｍ）

主板（总线设备）

外部存储设备（硬盘，Ｕ盘）

输入输出设备

软件

操作系统（内核模式）

标准库软件

应用软件（用户模式）

vmware

开启ｂｉｏｓ里的虚拟机支持功能

打开终端和退出终端：

１．点击图标

２．搜索命令：gnome-terminal或终端

退出终端：

exit

Linux命令的格式：

命令名　[选项]　[参数]

例如：

pwd

ls -l

cd /

ls /usr/bin -l

说明：

有些命令没有选项和参数

目录树和根

根：root

linux: /

windows: c:或d: ...

例如：

/usr/bin/python3(Linux下的文件)

c:\Windows\System32\calc.exe(window下的文件)

路径：

路径是用来记录一个文件或文件夹的字符串

分为两种：

绝对路径　　和　　相对路径

绝对路径：

以'/'字符开头的路径为绝对路径

绝对路径通常能标识一个唯一的文件或文件夹

相对路径：

不以'/'开头的路径为相对路径

开始符号：

以文件名或文件夹名开头

. 开头当前文件夹

.. 开头（上一级文件夹）

~ 用户主目录

示例：

cd /usr/bin #进入/usr/bin目录

ls -l ./python3 查看当前目录下python3文件信息

cd .. 返回上一级目录

pwd 命令：

作用：

显示当前操作的路径位置

cd 命令：

作用：

改变当前的工作路径（进入某个目录）

格式：

cd [目录名]

示例：

cd /home/tarena

cd #默认进入用户主目录

ls 命令：

作用：

用来显示文件或文件夹的信息

格式：

ls [选项]　[文件夹名或文件名]

常用选项：

-l 列表显示文件的详细信息

-a 显示全部文件/文件夹信息

实例：

ls -l /usr/bin/python3

ls ~

mkdir 命令：

作用：

创建一个或多个文件夹

格式：

mkdir 文件夹名１　文件夹名２．。。

示例：

mkdir a b c d

常用选项：

-p 如果中间文件夹不存在，则逐级创建所有文件夹

示例：

mkdir -p a/b/c/d

rmdir 命令：

作用：

删除一个或多个文件夹（文件夹内必须没文件）

格式：

同　mkdir

选项：

-p 如果中间的文件也为空，则逐级删除中间的空文件夹

示例：

rmdir a b c d

rmdir -p a/b/c/d

tree 命令：

作用：

显示目录树

安装方法：

sudo apt install tree

touch 命令：

作用：

１．如果文件不存在，则创建一个空文件

２．如果文件或目录存在，则用系统时间更新它的修改时间

示例：

touch a.txt[ b.txt...]

rm 命令：

作用：

删除文件或文件夹

格式：

rm [选项] 文件或文件夹

常用选项：

-r 递归删除文件内部的全部文件和文件夹

-i 删除钱给出提示（y代表yes,n代表no）

-f 强制删除，不给出任何提示

练习：

１．进入到/usr/bin目录下你查看是否有python3和ls 这两个文件

２．在进入到/etc目录下，查看passwd这个文件的大小是多少节

３．回到用户主目录下

４．查看当前目录下都有哪些隐藏文件

５．创建一个文件夹‘aid1803’

６．在aid1803下创建一个文件夹Linux

７．在aid1803/Linux下创建三个文件夹：day01 day02 day03

８．在day01下创建day01.txt,在day02下创建day02.txt,在day0　　　　3下创建day03.txt

９．删除aid1803/Linux/day03文件夹

## Day02

help 帮助选项

选项：

--help

作用：

查看一个命令有哪些选项，这些选项都是做什么的

示例：

ls --help

man 命令帮助

格式：

man linux/UNIX命令名

作用：

查看命令所对应的帮助文档（手册）

示例：

man ls

man mkdir

说明：

退出键：q

Tab键

作用：

自动补全命令或文件路径

history 命令

作用：

把以前键入的历史命令都列出来

注：

在终端$提示符下，用上下键可以翻出历史命令

通配符

\* 代表０个，１个或多个任意字符

? 代表１个任意字符

示例：

有如下文件：

a ab ac abc aabb bc cd

a\* 代表所有以a开头的文件

a\*b 代表以a开头，以b结尾的文件

a? 代表　ab ac

?c 代表ac bc

...

练习１：

１．创建文件夹myfiles

２．在myfiles文件夹内创建文件：

a.mp3 ab.txt ac.mp3 abc.txt aabb.mp3 cd.mp3

３．列出所有以.mp3结尾的文件

４．删除.之前只有一个字符的文件

５．列出含有c这个字符的文件

６．删除myfiles文件内所有的mp3文件

答案见code内的lianxi1.sh

文本编辑器

sublime text 3

记事本

vi

notepad++

gedit

sublime text 文本编辑器

启动命令:

subl<回车>

subl 文件名或文件夹名

快捷操作：

１．打开和隐藏侧边栏

view->Side Bar->show/hide Side bar

快捷键：Ctrl + k + b

２．字体的放大缩小

Ctrl + 鼠标滚轮

快捷键：Ctrl + '+='/'-'

３．创建和取消多个光标

Ctrl + 鼠标左键点击　　创建多个光标

ESC　取消多个光标

４．批量多选相同的选择块

Ctrl + d

５．注释：

Ctrl + /

６．交替换行

Ctrl + Shift　+　上下键

７．撤销undo/重做redo

Ctrl + z

Ctrl + y

cat 命令

作用：

将文本文件的内容作为标准输出显示

格式：

cat file1 file2...

cp　命令

作用：

复制文件或文件夹

格式：

cp [选项] 源文件或文件夹　目标文件或文件夹

常用选项：

-a 复制文件夹及其内部的内容

mv　命令

作用：

文件搬移或者更名

格式：

mv 源文件或文件夹　目的文件或文件夹

find 命令

作用：

根据文件名称的信息查找指定的文件或文件夹

常用格式：

find 路径 [选项１]　-name "文件名或文件夹" [选项２]

常用选项１：

-type f 普通文件

-type d 目录

常用选项２：

-delete 删除

-perm 777　　权限

-user tom　　用户名

grep 命令

作用：

查找文件中相应的内容的文本信息

格式：

grep "内容"　[选项] 文件名或文件夹

常用选项：

-n 显示行号

-r 递归搜索文件夹内的文件

练习２：

１．查找gzip 和 tar这两个文件在什么位置

find / -name "gzip"

find / -name "tar"

２．查找/etc文件夹下哪些文件里含有"tarena"这个内容

grep "tarena" -nr /etc

３．将/etc/group 文件复制到当前文件夹

cp /etc/group .

４．将/etc/passwd 文件复制到当前文件夹并改名为“系统账户管理配置文件.txt”

cp /etc/passwd ./系统账户管理配置文件.txt

答案见code内的lianxi2.sh

gizp 命令

作用：

用zip无损数据压缩算法对文件进行压缩，生成压缩后的文件

注：

只能对单个文件进行压缩，不能对文件夹进行压缩

格式：

gzip 文件名

gunzip 命令

作用：

解压缩　.gz 文件

格式：

gunzip 文件名（.gz文件）

tar 命令

作用：

对文件或文件夹进行打包和解包的操作

格式：

tar [选项] 生成的文件名　源文件或文件夹

常用选项：

-c　　　　　创建包

-x　　　　　解包

-f　　　　　操作的文件名

-v　　　　　显示操作的文件细节

-z　　　　　用gzip/gunzip对包进行压缩和解压缩

示例：

#将day02打包为day02.tar,但不压缩

tar -cvf day02.tar day02

#解包day02.tar

tar -xvf day02.tar

#以下将打包和压缩的工作一起进行

tar -czvf day02.tar.gz day02

#解压缩一个tar.gz的包

tar -xzvf day02.tar.gz

文件的权限管理及命令

查看文件权限

ls -l 文件名

文件的权限类型：

r 读权限

具有读取文件内容的权限

w 写权限

具有新增，修改，删除文件内容的权限

x 执行权限

具有执行文件的权限

- 无权限

最高权限：rwxrwxrwx 最低权限：---------

权限上的分组：

用户权限（user）

组权限（group）

其他（other）

chmod 命令

作用：

用来修改文件权限

格式：

chmod 权限　文件名或文件夹名

权限：

u 用户

g　　同组用户

o　　其他用户

a　　所有用户

+　　加权限

-　　去除权限

输出重定向

输出分两种：

１．标准输出

２．标准错误输出

标准输出重定向

> 将一个命令的标准输出重定向到指定文件

>> 将一个命令的标准输出追加到指定文件末尾

示例：

find /etc -name "group" > output.txt

标准错误输出重定向

2> 将一个命令的错误信息重定向到指定文件

2>> 将一个命令的错误信息追加到指定文件末尾

示例：

find /etc -name "group" 2> error.txt

重定向所有输出

&>

&>>

作用：

将所有输出重定向到一个文件

示例：

find /etc -name "group" &> all.txt

/dev/null 文件

写入到此文件的内容全部被丢掉

示例：

find / -name "gzip" 2> /dev/null

clear 命令

作用：

清屏

快捷键：　ctrl + l

《python基础》

python的诞生

创建人：Guido van Rossum(荷兰人)

时间：１９８９年圣诞期间

计算机语言发展：

１．二进制机器指令

２．汇编语言：

ADD R0 R1

ST R2

３．编译型语言

C / C++ / PASCAL / GO

４．解释执行语言：

python / java / basic / c# / swift

python的应用领域

系统自动化运维

网络编程（搜索引擎，爬虫，服务器编程）

科学计算

人工智能，机器人

云计算

大数据及数据库编程

教育

游戏

...

python优点：

面向对象（java c++ python C# swift）

免费

可移植

可混合编程（c c++ java .net）

简单易学易用

开发效率高

应用邻域广泛

开源

python缺点：

与c/c++相比，执行速度不够快

不能封闭源代码

不能写操作系统内核和驱动程序

python官网：

www.python.org

python的版本：

v2.7(2020年结束维护)

v3.5(当前教学)

v3.8(最新)

python的解释执行器：

CPython(C语言开发)　　/usr/bin/python3

Jyphon(Java语言开发)

IronPython(.net开发)

练习３：

写一个test.py文件，此文本写入python三条打印语句，打印如下：

这是我的python第一条语句

我现在开始学python了

这是最后一条语句

要求：

１．用两种方式执行上述程序打印这三条语句

２．将这三句语句输出重定向到myprint.txt中

# Python部分

## Day03

常用的程序文件的后缀：

.py 　　python文件

.c　　　　c语言源文件

.cpp　　　c++语言文件

.java java语言文件

python解释执行器

python

python3

python解释执行器的交互模式

$ python3<回车>　　＃进入交互模式

退出交互模式

>>>　exit() <回车>

>>> quit() <回车>

快捷键：Crtl + D

python 注释

python注释是以#开头，直至行尾

作用：

让注释内容不参加解释执行

python程序的组成

程序由模块组成

模块由语句，函数，类的等组成

语句包含表达式

表达式建立并处理数据对象且能返回数据对象的引用关系

python的核心数据类型

数字（整数，浮点型，复数，布尔型（bool））

字符串　str

列表　list

...

整型数　　int

整型数是不带有小数部分的数字。包括自然数，０及负数自然数

如：-2,100,0

整数字面值的表示方式：

十进制方式表示：

-10,0,999999

八进制表示方式（0o开头，后跟0~7）

0o177 (127)

0o11 (9)

十六进制表示方式（0x开头，后跟0~9,A~F,a~f）

0x11 (17)

0xff (255)

二进制表示方式（0b开头，后跟0~1）

0b1001

二进制转十六进制，从右往左取４位

例如：0b 10 0110 = 0x26

二进制转八进制，从右往左取３位

例如：0b 100 110 = 0o46

浮点型数　float

浮点数是带有小数部分的数字（小数部分也可以为０）

浮点数的两种表示形式

小数表示：

3.14 / 3. / .14 /　3.0

科学计数法：

格式：

小数 e/E 指数(可正可负)

如：6.18E-1 (6.18\*(10\*\*-1))

2.99e8 (2.99\*(10\*\*8))

复数　complex

分为两部分：

实部（real）

虚部（image）

注：虚部必须是以j或J结尾的数

如：1j,i+2j ,1-3j,-100+10J

布尔类型　bool

用来表示真假两种状态的类型：

True 表示真（条件满足或成立）

False 表示假（条件不满足或不成立）

说明：

True 值为１

False 值为０

混合类型自动升级

１＋１．３　　返回类型为浮点型

空值 None

None 是一个表示不存在的特殊对象

作用：

１．用来语法占位

２．变量解除绑定

表达式　和　运算符

表达式　expression

由数字或数字和运算符组成

作用：

通常让计算机做一些事情并返回结果

示例：

１＋２

运算符

算术运算符

+　　　　加法

-　　　　减法

\*　　　　乘法

/　　　　除法（返回的是浮点类型）

//　　　求整（返回类型由除数和被除数的类型决定）

% 求余

\*\* 幂运算

优先级

\*\* (高)

\* / // % (中)

+ - (低)

括号分组子表达式：

用（）可以将表达式分组，（）内部的表达式优先计算

基本输出函数　print

格式：

print(要打印的东西)

示例：

print('hello')

print(1+2+2)

print(1+2,3+4,100)

练习１：

１．写程序在终端输出图形：

########

# #

# #

########

２．已知矩形的长边为６ｃｍ，短边为４ｃｍ，求矩形的周长和面积，并用print函　　　数打印在终端上

答案见code内lianxi1

变量

变量是关联一个对象的标识符（符号）

作用：

用来绑定一个数据对象，以便于以后重复使用

变量名的命名方法：

１．变量名必须为字母或下划线开头，后跟字母、下划线或数字

２．不能使用python的关键字（保留字）

python3关键字：

True,False,None,is,del,if,elif,else,pass,not,in....(共计３３个)

注：

python变量名区分大小写

赋值语句

符号：=

语法：

变量名 = 表达式

或

变量名１　= 变量名２ = 表达式

或

变量名１，变量名２，... = 序列

作用：

用于将一个变量绑定（或关联）在一个对象上

说明：

１．当变量不存在时，创建该变量，并绑定这个对象

２．当变量存在时，改变这个变量的绑定关系

３．一个变量只能绑定一个对象

４．多个变量可以绑定同一个对象

python中自动化内存管理和引用计数

每个对象都会记录有几个变量引用自身，当引用的数量为０时，则此对象将被销毁

练习２：

变量交换练习，已知有两个变量：a = 10,b = a

问如何在不创建新的对象的情况下，让a和b绑定的对象

1.a,b = b,a

2.t = a

a = b

b = t

is / is not 运算符

is 的作用：

判断两个对象是否是同一个对象，如果是同一个对象，

返回True,否则返回False

is not 的作用额is 相反

语法：

x is y

x is not y

小整数对象池：

CPython中，整数-5至256永远存在内存中不会释放,并可以重复使用

示例：

a = 100

b = 100

a is b #返回True

id　函数

作用：

返回一个对象在内存中的地址

用法：

id(x)

del 语句

作用：

用于删除变量，同时解除与对象的绑定关系，如果可能，释放对象

语法：

del 变量名

在交互模式下，查看当前作用域内变量的方法：

>>> help('\_\_main\_\_')

退出键：q

复合赋值算术运算符：

y += x 等同于　y = y + x

y -= x 等同于　y = y - x

y \*= x 等同于　y = y \* x

y /= x 等同于　y = y / x

y //= x 等同于　y = y // x

y %= x 等同于　y = y % x

y \*\*= x 等同于　y = y \*\* x

练习３：

１．在终端打印出图形：

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

２．中国古代的秤是１６两一斤，请问古代的２１６两是古代的几斤几两？

３．从凌晨0:0:0计时，到现在已经过了６３３３２０秒，请问现在是几时，几分，几秒。

见code内的lianxi3

## Day04

比较运算符

< 小于

<= 小于等于

> 大于

>= 大于等于

== 等于

!= 不等于

语法：

左表达式　< 右表达式

说明：

比较运算符返回布尔类型的值

数值对象的构造（创建）函数

float(obj)　　　　　　　　　　　＃用字符串或数字转换为浮点数

int(x=0,base=10) 　　　　＃用字符串或数字转换为浮点数＃用数字或字符串转换为

　　整数，如果不给出参数，则返回0，base=10表示十进

　　制,base=2表示二进制，base=16表示十六进制

complex(r=0.0,i=0.0) #用数字创建一个复数

bool(x) 　　　　　　　　　　　　　#用x创建一个布尔值

bool(x)返回的情况：

None, 0, 0.0, 0j(所有的数字零)，　以及所有空容器（'',[],{},()...）返回False,其他都为True

函数调用:

函数名（传参列表）

说明：

函数调用是表达式，通常用于返回一个对象的引用关系

练习１：

将数字3.14用变量pi绑定

将pi变量转换为整数用变量i绑定

将pi变量与i变量相减，结果用变量f绑定

１．判断f是否为0,14

２．删除所有变量

预置（内建）的数值型函数

abs(x) 取x的绝对值

round(number[,ndigits]) 对数值进行四舍五入，ndigits是小数点向右取整数的位数，负数表示向左取整

示例：

round(123.5) #123

round(123.500001) #124

round(123.5678 ,2) #123.57

round(123.5678 ,-2) #100.0

round(123456.789,-2) #123500.0

pow(x,y,z=None) 相当于x\*\*y或x\*\*y%z

help() 查看函数的帮助：

>>> help(函数名)

例：

help(abs)

语句　statements

语句是python执行的最小单位，语句可以独立执行

示例：

a = 100 　　　　#赋值语句

del a　　　　　 #del语句

print('hello')　 #表达式语句

1 + 2 + 3　　　　#表达式语句

注：多条语句写在一行内需要用分号（;）分开，

　　　通常一条语句写在一行内（复合语句除外）

折行符　\

告诉解释执行器，下一行的内容也是此语句的内容

隐式换行：

所有的括号的内容换行时，解释执行器会自动去下一行找到相对应的括号，直到找到为止，称作隐式换行

基本输入输出函数：

输入函数　input

作用：

从标准输入设备上读取一个字符串（注：末尾的换行字符会被删除）

格式：

input('提示字符串')

注：返回一个字符串，提示字符串可以为空

输出函数　print

作用：

将一系列的值以字符的形式输出到标准设备上，默认为终端

格式：

print(value,...,sep=' ',end='\n')

关键字参数说明：

sep:两个值之间的分隔符，默认为空格（' '）

end:输出完毕后在末尾自动追加一个字符串，默认为换行符('\n')

练习２：

１．假设１年只有３６５天，输入年数，打印这些年有多少周

２，分三次输入当前的小时，分钟，秒数，在终端打印此时间距离０：０：０过了多　少秒？

见code内的lianxi2

if 语句

作用：

让程序根据条件选择性的执行某条语句或某些语句

语法：

if 真值表达式１：

语句块１...

elif 真值表达式２：

语句块２...

...

else 真值表达式３：

语句块３...

说明：

１．真值判断会自上而下进行，如果有一条为True，则执行其中的语句，然后结束if语句的执行，如果所有真值表达式都为false,则执行else子句里的语句

２．elif子句可以有０个，１个或多个

３．else子句可以有０个或１个

练习：

输入一个正整数，判断这个数是奇数还是偶数，打印结果：

是奇数

是偶数

练习３：

１．输入一个季度１～４，输出这个季度有那几个月，如果不是输入１～４整数，则提示用户您输错了

２．输入一年中的月份（１～１２），输出这个月在哪个季度，如果输入的是其他的数，则提示您输入有错

见code 内lianxi3.py

if 语句的嵌套

if 语句本身是由多条子句组成的一条复合语句

if 语句可以作为语句嵌套到另一个语句的内部

示例见：

code内if\_embed.py

条件表达式

语法：

表达式1 if 真值表达式 else 表达式2

作用：

如果真值表达式的布尔值为True时，则执行表达式1，并返回结果对象的引用，否则执行表达式2，并返回对象的引用

示例见：

code 内 if\_else\_express.py

练习4：

1.写入一个程序，输入一个数，用if 语句计算并打印这个数的绝对值

2.写入一个程序，输入一个数，用if 语句计算并打印这个数的绝对值

见code 内 lianxi4

pass 语句

作用：

通常用来填充语法空白

布尔运算

运算符：

not and or(优先级低于比较运算符)

布尔非操作 not

语法：

not x

作用：

对 x 进行布尔取非，如bool(x)为True则返回False，否则返回True

布尔与操作 and

语法：

x and y

注：x,y代表表达式

作用：

优先返回假值对象，当x的布尔值为False时，返回x,否则返回y

True and True #返回True

True and False #返回False

False and True #返回False

False and False #返回False

布尔或操作 or

语法：

x or y

作用：

优先返回真值对象，当x为True，返回x，否则返回y

True or True #返回True

True or False #返回True

False or True #返回True

False or False #返回False

示例：

s = input('输入数字：') or '0'#如果输入为空，则返回'0'

score = int(s)

print(score)

正负号运算符：

+(正号) -(负号)

语法：

+ 表达式

- 表达式

注：正负运算符是一元运算符（只有一个元素参加运算）

示例：

a = 5

b = -a

c = +a

练习5：

1.别进出租车计价器：

收费标准：

3公里以内收费13元，超出3公里，每公里基本单价2.3元/公里，空驶费：超出15公里后，每公里加收1.15元空驶费

要求：

输入公里数，打印出费用金额（以元为单位四舍五入）

2.输入一个学生的三科成绩：

1）打印出最高分是多少分？

2）打印出最低分是多少分？

3.给出一个年份，判断是否为闰年并打印

4.BMI指数（Body Mass Index）又称身体质量指数

BMI值计算公式：

BMI=体重（公斤）/身高的平方（米）

标准表

BMI < 18.5 体重过轻

18.5 <= BMI <= 24 正常范围

BMI > 24 体重过重

输入身高和体重，打印BMI值并打印体重状态

## Day05

字符串 str

作用：

用来记录文本信息

字面值表示方法：

用引号括起来的部分都是字符串

' 单引号

" 双引号

''' 三单引号

""" 三双引号

字符串的字面值表示方式：

空字符串表示：

''

""

''''''

""""""

非空字符串表示：

'hello'

"hello"

'''hello'''

"""hello"""

单引号和双引号的区别：

单引号内的双引号不算结算符

双引号内的单引号不算结算符

三引号字符串的作用：

三引号内可以包含单引号个双引号

三引号字符串中的换行会自动转换为换行符'\n'

示例：

>>>print('''尊敬的：

...您好！

... dhfksjhfk

...''')

>>>

隐式字符串字面值拼接

s = "I'm a teacher," 'My name is "xxx"。'

用转义序列代表特殊的字符

字符串字面值中，用字符反斜杠\,后跟一些字符代表特殊的一个字符

例如：

\' 代表一个双引号

\" 代表一个双引号

\n 换行

字符串中反斜杠转义字符表

\'

\"

\\

\n

\r 返回光标至行首

\f 换页

\t 水平制表符

\v 垂直制表符

\b 退格

\0 字符串，字符值为零

\xXX XX为两位十六进制表示的字符

示例：

>>>print('\x41')

>>>A

\uXXXX Unicode16的十六进制表示的字符

示例：

>>>print('\u0041')

>>>A

\UXXXXXXXX Unicode32de 三十二进制表示的字符

ASCII 编码

码表见：

$ man ascii

常见的ASCII编码：

字符 十进制 十六进制

'0' 48 0x30

'A' 65 0x41

'a' 97 0x61

'\0' 0 0x00

'\n' 10 0x0A

UNICODE编码：

全世界字符统一码

分类：

UNICODE16(两个字节)

UNICODE32(四个字节)

raw 字符串（原始字符串）

格式：

r'字符串内容'

r"字符串内容"

r'''字符串内容'''

r"""字符串内容"""

作用：

让转义符号 \ 无效

字符串的运算

+ 加好运算符用于拼接字符串

+= 运算符用原字符串与右侧字符串拼接生成新的字符串

示例：

>>>'abc' + 'efg'

'abcdefg'

\* 生成重复的字符串

\*= 生成重复的字符串并让原变量绑定生成后的字符串

示例：

>>>'abc' \* 3

'abcabcabc'

练习1：

写一个程序，打印一个高度为4行的矩形方框

要求：输入一个整数数字，此整数代表矩形的宽度，输出此矩形

如：

请输入矩形宽度：10

##########

# #

# #

##########

见code 内的 lianxi1.py

字符串的比较运算：

>

>=

<

<=

==

!=

格式：

x > y

比较规则：

1.字符串x的第一个字母与字符串y的第一个字母比较，如果不相同，则直接的得到比较结果，如果相同，则再取第二个字母进行比较，以此类推

2.比较的依据是字符的UNICODE编码值

示例：

'ad' > 'abc' #True

'abc' != 'cba' #True

'ab' <= 'abc' #True

in / not in 运算符

作用：

用于序列，字典，集合等容器中，用于判断某个值是否存在于容器中，如果存在返回True,否则返回False

not in 与 in 运算符的返回结果相反

格式：

对象 in 序列

示例：

'a' in 'abc' #True

>>>x = 'welcom to tarena'

>>>'to' in x

True

>>>'abc' in x

False

字符串的索引 index

python的字符串是不可以改变的字符序列

语法：

字符串[整数表达式]

说明：

1.python序列都可以用索引来访问序列中的对象

2.python序列的正向索引是从0开始的，第二个索引为1,以此类推，最后一个索引是len(s)-1

3.python序列的反向索引是从-1开始的，-1代表最后一个，-2代表倒数第二个，以此类推，第一个是-len(s)

示例：

>>>s = 'abcd'

>>>print(s[0])

a

>>>print(s[1])

b

>>>print(s[-1])

d

练习2

输入一个字符串

1.打印这个字符串的第一个字符

2.打印这个字符串的最后一个字符

3.如果此字符串的长度为偶数，打印一个’|‘，如果长度为奇数，打印中间的字符

见 code 内的 lianxi2.py

切片 slice

从字符串序列中取出相应的元素，重新组成一个字符串序列

语法：

字符串[（开始索引）：（结束索引）[:步长]]

语法说明：

1.开始索引是切片切下的位置，0代表第一个元素，1代表第二个元素，-1代表最后一个元素

2.结束索引是切片终止索引（但不包含终止点）

3.步长是切片每次获取完当前元素后移动的方向和偏移量

1）没有步长，相当于取值完毕后向右移动一个索引的位置（默认为1）

2）当步长为正整数时，取正向切片

3）当步长为负整数时，取反向切片，反向切片时，默认的起始位置最后一个元素，终止位置是第一个元素的前一个位置

练习3

输入任意一个字符串，判断这个字符串是否是回文

回文示例：

上海自来水来自海上

见 code 内的 lianxi3.py

字符串编码转换函数

ord(c) 返回一个字符的Unicode值

示例：

>>>ord('a')

97

chr(i) 返回i这个值所对应的字符

示例：

>>>chr(65)

'A'

整数转换为字符串函数：

hex(i) 将整数转换为十六进制字符串

oct(i) 将整数转换为八进制字符串

bin(i) 将整数转换为二进制字符串

python3中常用的序列函数

len(seq) 返回序列长度

max(seq) 返回序列中最大值的元素

min(seq) 返回序列中最小值的元素

练习4

1.写一个程序，输入一段字符串，如果字符串不为空，则把第一个字符的Unicode编码值打印出来

2.写一个程序，输入一个整数值（0~65535）,打印这个编码所对应的字符

见code内的lianxi4.py

练习5

输入三行文字，让这三行位子在一个方框内居中显示（不要输入中文）

如:

请输入第一行：hello tarena!

请输入第二行：my name is weimingze.

请输入第三行：today is sunday!

显示如下：

+---------------------------------- +

| hello tarena! |

| my name is weimingze. |

| today is sunday! |

+---------------------------------- +

注：max()函数可以输入多个参数，返回其中最大值

## Day06

字符串格式化表达式

作用：

生成一定格式的字符串

运算符：

%

格式：

格式化字符串 % 参数值

或

格式化字符串 % （参数值1，参数值2，...）

示例：

>>>fmt = '姓名：%s, 年龄：%d'

>>>fmt % ('张飞'，20)

姓名：张飞，年龄：20

占位符 % 和类型码之间的格式语法：

% [格式语法] 类型码(d,s,f等)

格式语法：

- 左对齐

+ 显示正负号

0 补零

宽度

宽度.精度

示例：

'%10d' %123 #' 123'

'%-10d' %123 #'123 '

'%10s' %'abc' #' abc'

'%-5s' %'abc' #'abc '

'%+10d' %123 #' +123'

'%010d' %123 #'0000000123'

'%.2f' %3.14159 #'3.14'

'%7.2f' %3.14159 #' 3.14'

练习1

1.输入三行文字，让这三行文字依次以20个字符的宽度右对齐输出

如：

请输入第一行：hello world

请输入第二行：abcd

请输入第三行：a

2.能否以最长字符串的长度进行右对齐显示，左侧填充空格

见code内的lianxi1

循环语句（有两种）

while 语句

for 语句

while 语句

作用：

根据一定条件，重复的执行一条语句或多条语句

语法：

while 真值表达式：

语句块1...

else:

语句块2

说明：

1.先执行真值表达式，测试布尔值为True 或 False

2.如果真值表达式为True，则执行语句1，

然后再返回第一步重复进行测试

3.如果真值表达式为False，则执行else子句中的语句块2，

然后结束次while语句的执行。如果没有else子句，

则直接结束while语句的执行

4.else子句部分可以省略

示例：

见while.py

while 语句的注意事项：

1.要控制循环的真值表达式的值来防止死循环

2.通常用真值表达式中的循环变量来控制循环条件

3.通常在循环语句块内需要改变循环变量来控制循环的次数和

变量的走向

while 语句嵌套

while语句本身是语句，和其它语句一样可以嵌套在

其它复合语句中(如:if语句，while语句，for语句...的内部)

练习2

1.写程序输入一个整数n

打印n以内的全部大于0的偶数（不含n）

2.打印从零开始的浮点数，

每个数增加0.5，打印出10以内所有这样的数

3.打印1-20的整数，打印在一行显示，空格分割

练习3

1.打印三角形

\*

\*\*

\*\*\*

\*\*\*\*

再打印：

\*

\*\*

\*\*\*

\*\*\*\*

再打印

2.写一个程序

输入一个开始的整数值用变量begin绑定

输入一个结束的整数值用变量end绑定

打印从begin到end（不包含end）的每个整数（打印在同一行）

如：

请输入8

请输入30

打印结果：

8 9 10 11 12 13 ...29 30

附加思考：

如何一行打印5个数字

3.用while循环生成如下字符串

1.生成‘ABCD...XYZ’

2.生成‘AaBbCcDd...XxYyZz’

提示用chr和ord函数

## Day07

可迭代对象（iterable）：

可以依次获取数据元素的对象

for 语句（循环语句）

作用：

用来遍历可迭代对象的数据元素

语法：

for 变量列表 in 可迭代对象：

语句块1...

else:

语句块2...

语法说明：

1.可迭代对象每次提供一个元素依次赋值给变量列表中的变量，赋值完毕后执行语句块

1，重复执行次步骤

2.当可迭代对象不能够提供数据时，执行else子句部分的语句块2，然后退出循环

3.else子句部分可以省略

4.当在语句内部用break终止循环时，else子句部分语句块2不会执行

示例见：

code内的for.py, for\_break.py

练习1

写程序，任意输入一行字符串，打印这个字符串内有多少个空格（要求用for语句）

range函数

作用：

用来创建一个生成一系列整数的可迭代对象（也叫整数序列生成器）

调用格式：

range(stop)

从零开始，每次生成一个整数后加1操作，直到stop为止（不包含stop）

range(start,stop[,step])

从start开始，每次生成一个整数后移动step,直到stop为止（不含stop，

且step可以为负整数）

示例：

range(4) #生成0 1 2 3

range(3,6) #生成3 4 5

range(1,10,2) #生成1 3 5 7 9

range(5,0，-2) #生成5 3 1

range(4,1) #空

见range.py

for 语句嵌套

示意：

for x in 'abc':

for y in 'xyz':

print(x+y)

练习2

用for循环嵌套打印如下矩形：

输入一个数n(10以内)，代表矩形的宽度和高度

如：

输入：5

打印如下：

1 2 3 4 5

2 3 4 5 6

3 4 5 6 7

4 5 6 7 8

5 6 7 8 9

见code内的lianxi2.py

continue 语句

作用：

用于循环语句（while,for语句）中，不再执行本次循环内continue之后的语句，

重新开始一次新的循环

说明：

1.在while语句中执行continue语句，将直接跳转到while语句的真值表达式处重新

判断循环条件

2.在for语句中执行continue语句，将会从可迭代对象中取下一个元素，绑定变量

后再次进行循环

示例见：

continue.py

练习3：

写一个程序，求1-100之间所有不能被5，7，11整除的数的和

见code内的lianxi3.py

python3中的序列：

字符串 str

列表 list

元组 tuple

字节串 bytes

字节数组 bytearray

列表 list

定义：

列表是由一系列元素组成的，元素与元素之间没有任何的关联关系，但他们之间有

先后顺序关系

列表是一种容器

列表是一种序列

列表是可以被改变的序列

创建空列表的字面值：

l = []

创建非空列表的字面值

l = [1,2,3]

l = ['a','b','c']

l = [1,'a',2.3,[1,2,3],4]

列表的构造函数 list

list() #生成一个空列表

list(iterable) #用可迭代对象创建一个列表

示例：

>>>list()

[]

>>>list('hello')

['h','e','l','l','o']

列表的运算符：

运算符：

+ 用于拼接列表

x = [1,2,3]

y = [4,5,6]

z = x+y #z = [1,2,3,4,5,6]

+= 用原列表与右侧列表拼接，并用变量重新绑定列表

\* 生成重复的列表

x = [1,2]

y = x\*3 #y = [1,2,1,2,1,2]

\*= 生成重复的列表，并给原列表赋值

列表的比较运算

运算符：

> >= < <= == !=

同字符串的比较一致

练习4

输入一些行文字，将这些行文字保存在一个列表L中，当输入空行时结束输入，

并打印列表L

见code内的lianxi4.py

练习5

1.将昨天的星星题用for循环写一遍

2.输入一个unicode的开始值，用变量begin绑定

输入一个unicode的结束值，用变量end绑定

打印开始值至结束值所对应的文字，生成字符串并打印

3.输入一个整数（代表树干的高度）

打印如下一棵圣诞树

输入：2

\*

\*\*\*

\*

\*

输入：3

\*

\*\*\*

\*\*\*\*\*

\*

\*

\*

4.输入一个整数（代表树干的高度）

打印如下一棵圣诞树

输入：3

1

222

33333

\*

\*

\*

5.输入一个正整数，打印这个数是否是素数（只能被1和自身整除的整数，1除外）

见code内的lianxi5.py

## Day08

列表 in / not in 运算符

in 判断一个对象是否存在于容器内，如果存在则返回True，否则返回False

用法同字符串一致

语法：

数据对象 in 列表

列表的索引

语法:

列表[整数表达式]

用法：

取值时，等同于字符串的索引操作

索引分为正向索引和反向索引，规则和字符串一致

列表索引赋值：

列表是可变的序列。可以通过索引赋值改变列表中的元素

示例：

l = [1,2,3]

l[1] = 22 #将第二个元素改为22

列表的切片

语法：

列表[:]

列表[::]

列表的切片取值时，返回一个列表，规则等同于字符串切片规则

列表切片赋值

作用：

可以改变原列表的排序，可以插入和修改数据，可以用切片改变列表的对应

元素的值

语法：

列表[切片] = 可迭代对象

注：赋值运算符的右侧必须是一个可迭代对象

示例：

l = [2,3,4]

l[:1] = [1.1,2.2] #l = [1.1,2.2,3,4]

l[3:] = [5,6] #l = [2,3,4,5,6]

l[1:2] = 'abcd' #l=[2,'a','b','c','d',4]

注意事项：

对于步长不等于1的切片赋值，赋值运算符右侧的可迭代对象提供的元素个数

一定要等于切片切出的段数

例：

l = [1,2,3,4,5,6]

l[::2] = 'abc' #l = ['a',2,'b',4,'c',6]

l[::2] = 'abcde' #会报错

del 语句 用于删除列表中的元素

语法：

del 序列[整数表达式]

del 序列[切片]

示例：

l = [1,2,3,4,5,6]

del l[-1] #删除最后一个元素6

del l[::2] #删除元素1，3，5

python3中常用于序列的函数

len(seq) 返回容器的元素个数

max(x) 返回序列的最大值元素

min(x) 返回序列的最小值元素

sum(x) 返回序列中所有元素的和（元素为数值类型）

any(x) 真值测试，如果容器内其中一个值为True,则返回True,否则返回False

all(x) 真值测试，容器内所有元素的布尔值为True才返回True

python3中列表方法：

见"python3笔记附件"中的“list方法”

练习1

写一个程序，让用户输入很多个正整数，当输入小于零的数时结束输入

1）输出这些数的和

2）输出这些数的最大数和第二大的数

3）删除最小的一个数

4）按原来输入的顺序打印出剩余的这些数

见code内的lianxi1.py

浅拷贝 shallow copy 和 深拷贝 deep copy

浅拷贝：

是指复制过程中，只复制一层变量，不会复制深层变量绑定的对象的复制过程

示例：

l = [3.1,3.2]

l1 = [1,2,l] #l1 = [1,2,[3.1,3.2]]

l2 = l1.copy() #l2 = [1,2,[3.1,3.2]] ,浅拷贝

l[0] = 3.14

l1 #此时l1 = [1,2,[3.14,3.2]]

l2 #此时l2 = [1,2,[3.14,3.2]]为什么变成3.14？

#l1[2] 和 l2[2] 共同拥有l绑定的对象

深拷贝：

示例：

import copy 导入copy模块

l = [3.1,3.2]

l1 = [1,2,l] #l1 = [1,2,[3.1,3.2]]

l2 = copy.deepcopy(l1) #l2 = [1,2,[3.1,3.2]]

l[0] = 3.14

l1 #此时l1 = [1,2,[3.14,3.2]]

l2 #此时l2 = [1,2,[3.1,3.2]] #此列表不受影响

注：

深拷贝通常只对可变对象进行深层复制，不可变对象通常不会被复制

列表与字符串比较

1.列表和字符串都是序列，元素之间有先后关系

2.字符串是不可变的序列，而列表是可变的序列

3.字符串中每个元素只能存储字符，而列表可以存储任意类型的元素

4.列表和字符串都是可迭代对象

字符串文本解析方法 split 和 join

s.split(sep=None) 将字符串使用sep作为分隔符分割s字符串，返回分割后的

字符串列表，当不给定参数时，用空白字符作为分隔符分割

示例：

>>>s = 'hello world'

>>>s.split()

['hello','world']

>>>s.split('o')

['hell',' w','rld'] # w 前有一个空格

s.join(iterable) 用可迭代对象中的字符串，返回一个中间用s进行分隔的字符串

示例：

>>>'-'.join('hello')

'h-e-l-l-o'

>>>'+'.join(['1','2','3'])

'1+2+3'

列表推导式 list comprehension

列表推导式是用可迭代对象依次生成带有多个元素的列表的表达式

作用：

用简易方法生成列表

语法：

[表达式 for 变量 in 可迭代对象]

或

[表达式 for 变量 in 可迭代对象 if 真值表达式]

示例：

>>>[i\*\*2 for i in range(10)]

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>>[i\*i for i in range(10) if i%2==0]

[4,16,36,64]

>>>[i\*\*2 for i in range(1,10,2)]

[1,9,25,47,81]

练习2

输入一个数值作为开始的数，用begin绑定

再输入一个结束的整数用end绑定

将开始至结束的数中，平方加1，能被5整除的数放入列表中

[i for i in range(1,100) if (i\*\*2+1)%5 == 0]

列表推导式的嵌套

语法：

[表达式1 for 变量1 in 可迭代对象1 if 真值表达式

for 变量2 in 可迭代对象2 if 真值表达式]

示例：

>>>[x + y for x in 'abc' for y in '123']

['a1','a2','a3','b1','b2','b3','c1','c2','c3']

练习3

1.输入一个整数n,代表结束的数，将1-n之间所有素数计算出来并存入到列表L中

1）最后打印此列表中的全部素数

2）打印这些素数的和

2.求100以内有哪些整数与自身加1的乘积再对11求余的结果等于8？

x\*(x+1)%11 == 8

打印这些数，将这些数存于列表中（尝试使用列表推导式）

3.计算20个斐波那契数，存于列表中，最后打印这20个数

1，1，2，3，5，8，13...

## Day09

元组 tuple

元组是不可改变的序列，同list一样，元组可以存放任意类型的元素

表示方法：

用小括号（）括起来，单个元素括起来后加逗号（，）区分单个

对象还是元组

创建空元组的字面值

t = ()

创建非空元组字面值

t = (1,2,3)

t = 1,

t = (1,)

t = 1,2,3

元组的错误示例：

t = (20)

x,y,z = 100,200,300 #序列赋值

x,y,z = (100,200,300) #序列赋值

x,y,z = [100,200,300] #序列赋值

x,y,z = 'abc' #序列赋值

type(x) 函数用来返回 x 的类型

元组的构造（创建）函数tuple

tuple() 生成一个空元组，等同于（）

tuple(iterable) 用可迭代对象生成一个元组

元组的运算：

元组的元素同列表的运算方式一致

算术运算：

+ += \* \*=

比较运算：

> >= < <= == !=

in / not in 运算

元组的索引和切片等同于列表的索引和切片

元组的切片返回一个元组

元组不能索引赋值和切片赋值

元组的方法：

t.count(value) 返回元素个数

t.index(value[,start,stop]) 返回元素的索引

序列相关的函数：

len(x)

max(x)

min(x)

sum(x)

any(x)

all(x)

str(obj) 将对象序列化为字符串

list(iterable)

tuple(iterable)

reversed(seq) 返回反向顺序的迭代对象

sorted(iterable,reverse=False) 返回已排序的列表

字典 dict

字典是一种可变的容器，可以存储任意类型的数据

字典中的每个数据都是用‘键’进行索引的

字典中的数据没有先后顺序关系，字典的存储是无序的

字典中的数据以键（key）-值（value）对的形式进行映射存储

字典的键不能重复，且只能用“不可变类型”作为字典的键

字典的字面值的表示方式

字典的表示方式以{}括起来，以冒号（:）分割键-值对，各键-值

对之间用逗号分隔开

创建空字典：

d = {} #d绑定空字典

创建非空字典：

d = {'name':'tarena','age':16}

d = {'year':2018,'month':4,'day':12}

字典的构成函数 dict

dict() 生成一个空字典，等同于{}

dict(iterable) 用可迭代对象初始化一个字典

dict(\*\*kwargs) 用命名关键字传参形式生成一个字典

示例：

d = dict()

d = dict([('name','tarena'),('age',15)])

d = dict(name='tarena',age=15)

字典的键为不可变类型：

bool, int, float, complex, str, tuple, frozenset(固定集合)

bytes(字节串)， None

可变的类型（不能充当字典的键）

list, dict, set(集合), bytearray(字节数组)

字典的基本操作：

键索引：

语法：

字典[键]

示例：

>>>d = {'name':'tarena','age':16}

>>>d['name']

'tarena'

>>>d['age']

15

添加/修改键值对：

字典[键] = 值对象

示例：

>>>d = {}

>>>d['name'] = 'tarena'

>>>d

{'name':'tarena'}

删除字典元素 del语句

语法：

del 字典[键]

示例：

>>>d = {'name':'tarena','age':16}

>>>del d['age'] #'age'键被删除

>>>d

{'name':'tarena'}

练习：

1.算出100-999以内的水仙花数

水仙花数是指百位的3次方，加上十位的3次方，加上个位的3次方

等于原来的数字

例如：

153=1\*\*3+5\*\*3+3\*\*3

for baiwei in range(1,10):

for shiwei in range(10):

for gewei in range(10):

x = baiwei \*100 + shiwei \* 10 + gewei

if x = baiwei \*\* 3 + shiwei \*\* 3 + gewei \*\*3:

print(x)

2.任意输入一些大于零，存于列表L中，当输入-1时结束输入

L = [1,3,5,3,7,9,3,7,6,5]

1）打印出这些数

2）打印出这些数的和

3）去掉列表L中重复第二次或之后出现的数，再次存到另一个列表L2

L2 = [1,3,5,7,9,6]打印这些数

4）打印L2列表中的数据的和

5）将L列表中，出现两次的数村到另一个列表L3中

L3 = [5,7]

## Day10

字典的成员资格判断 in / not in 运算符

用 in 运算符可以判断一个键是否存在于字典中，如果存在则返回True,否则返回False

not in 返回值与 in 相反

示例：

d = {1:'一',2:'二'}

1 in d #True

2 in d #True

2 not in d #False

'二' in d #False

列表的 in 运算符和字典的 in 运算符比较：

1.列表的in运算符计算时间会因列表中元素个数增加而增加

L = [3,7,0,1...10976] #大约有十万个

if 888 in L:

print('888在列表里') #慢

2.字典的in运算符计算时间不会因键值对的个数增加而增加，操作速度快于列表

d = {3:None,7:'xxx',0:None,....10976:None}

if 888 in d:

print('888在字典的键里') #快

字典的迭代访问：

容器是可迭代对象，字典只能对键进行迭代访问

>>>d = {'name': 'zhang','birthday': (2008,8,8)}

>>>for k in d:

... print(k)

birthday

name

可以用于字典的内建（built-in）函数

len(x) 返回字典的键值对的个数

max(x) 返回字典的键的最大值

min(x) 返回字典的键的最小值

sum(x) 返回字典的所有键的和

any(x) 对所有键进行真值测试，有一个为True结果为True

all(x) 对所有键进行真值测试，所有为True才为True

字典的方法：

见：

>>>help(dict)

或“python3笔记附件”内的“dict方法.html”

练习1：

输入一个整数代表星期几(0-6),0/日代表周日，1/'一'代表周一....依次类推

任意输入字符串，打印这个字符串是否代表星期几，如果不是以上字符

打印"字典内没有相应的数据"

见 code 内的lianxi1.py

字典推导式：

语法：

{键表达式:值表达式 for 变量 in 可迭代对象 [if 真值表达式]}

注：[]的内容可以代表可省略

示例：

生成一个字典，键为10以内的数字，值为键的平方

d = {x:x\*\*2 for x in range(10)}

练习2：

1.有字符串列表如下：

l = ['tarena','xiaozhang','hello']

用推导式生成如下字典：

d = {'tarena':6,'xiaozhang':9,'hello':5}

注：字典的值为键的长度

2.有两个列表：

nu = [1001,1002,1003,1004]

names = ['Tom','Jerry','Spike','Tyke']

用nu中的编码作为键，以names中的字符串作为值，生成相应的字典

见 code 内的lianxi2.py

字典推导式的嵌套：

语法同列表推导式嵌套

字典 vs 列表

1.都是可变对象

2.索引方式不同，列表用整数索引，字典用键索引

3.字典的查找速度可能会快于列表（重要）

4.列表的存储是有序的，字典的存储是无序的

集合 set （是可迭代对象）

集合是可变的容器

集合内的数据对象都是唯一的（不能重复多次的）

集合是无序的存储结构，集合中的数据没有先后顺序关系

集合内的元素必须是不可变对象

集合是可迭代对象

集合是相当于只有键没有值的字典（键则是集合的数据）

创建空的集合：

set()

创建非空的集合：

s = {1,2,3,4}

集合的构造函数 set

set() 创建一个空的集合对象（不能用{}来创建空集合）

set(iterable) 用可迭代对象创建一个新的集合对象

示例：

s = set() #s 为空集合

s = {3,2,5}

s = set('abcda') # s = {'a','b','c','d'}

s = set({1:'1',2:'2',3:'3'}) #s = {1,2,3}

s = set(range(1:10:3)) # s = {1,4,7}

集合的运算：

交集，并集，补集，子集，超集

运算符：

& | - ^ < <= > >= == !=

in / not in

& 生成两个集合的交集

>>>s1 = {1,2,3}

>>>s2 = {2,3,4}

>>>s1 & s2

{2,3}

| 生成两个集合的并集

>>>s1 = {1,2,3}

>>>s2 = {2,3,4}

>>>s1 | s2

{1,2,3,4}

- 生成两个集合的补集

>>>s1 = {1,2,3}

>>>s2 = {2,3,4}

>>>s1 - s2

{1}

^ 生成两个集合的对称补集

>>>s1 = {1,2,3}

>>>s2 = {2,3,4}

>>>s1 ^ s2

{1,4}

> 判断一个集合是另一个集合的超集

{1,2,3} > {1,2} #True

{2,3,1} > {3,2} #True

< 判断一个集合是另一个集合的子集

{2,3,4} < {1,2,3,4}

== / != 集合相同和不同

{1,2,3} == {3,2,1} #True

{1,2,3} != {2,3,4} #True

in / not in 运算符

等同于字典的in,用于集合中，当某个值存在于集合中，返回True,否则返回False

集合的in / not in 运算符的速度快于序列

能用于集合的函数：

len(x) 返回集合元素的个数

max(x) 返回集合元素的最大值

min(x) 返回集合元素的最小值

sum(x) 返回集合的所有元素值的和

any(x) 对所有元素进行真值测试，有一个为True结果为True

all(x) 对所有元素进行真值测试，所有为True才为True

python3 集合中常用方法

见：>>>help(set)

或：“python3笔记附件”中的“set方法.html”

集合推导式

集合推导式是用可迭代对象生成集合的表达式

语法：

{表达式 for 变量 in 可迭代对象 [if 真值表达式]}

注：[]内的内容可省略

示例：

numbers = (1,2,3,2,2,4,5,3)

s = {x for x in numbers} #去除重复的元素

固定集合 frozenset

固定集合是不可变的，无序的，含有不重复元素的集合

作用：

固定集合可以作为字典的键，还可以作为集合的值

创建空的固定集合

fs = frozenset()

创建非空的固定集合：

frozenset(iterable) 用可迭代对象创建新的固定集合

示例：

fs = frozenset([2,3,4,5])

运算：

& 交集

| 并集

- 补集

^ 对称补集

in / not in 运算

< <= > >= == !=

固定集合的方法：

相当于集合的全部方法（去掉修改集合的方法）

列表，字典，集合对象优缺点：

1.都为可变

2.列表的尾部插入速度比较快，查中间插入和头插入数据速度慢

3.列表有序

4.字典添加，删除，修改速度快于列表

5.集合的添加，删除快于列表，且不能存储重复元素

练习3：

1.输入一段字符串，打印所有输入过的字符串，

但重复的只打印一次（不要求打印的顺序）

s = input('输入字符串：')

for c in set(s):

print(c,end=' ')

2.输出一段字符串，打印出这个字符串中出现过的字符及出现过的次数

#方法一

s = input('输入字符串：')

d = {x:s.count(x) for x in set(s)}

for y in d.items():

print('%s:%d次' %y)

#方法二

for y in set(s):

print(y,':',s.count(y),'次')

#方法三

d = {}

for y in s:

if y not in d:

d[y] = 1

else:

d[y] += 1

练习4

1.有一只猴子，摘了很多桃子

第一天吃了全部桃子的一半，感觉不饱又吃一个

第二天吃了剩下桃子的一半，感觉不饱又吃了一个

...依次类推

到第十天，发现只剩下一个

请问第一天摘了多少个桃？

2.完全数

1+2+3=6（6为完全数）

完全数是指除自身以的所有的因数和等于本身

求4-5个完全数：

6

28

496

8128

3.任意输入一个数n代表三角形的高度，打印此形状的三角形

如输入4

打印如下：

1

121

12321

1234321

4.将第3题打印如下：

1

121

12321

1234321

12321

121

1

## Day11

函数 function

函数是可以重复执行的语句块，可以重复调用。

作用：

1.用于封装可重复执行的语句，提高语句的可重用性

2.定义用户级别的函数

函数定义语句 def语句的语法：

def 函数名（参数列表）:

语句块（代码块）

函数的语法说明：

1.函数的名字就是语句块的名称

2.函数名的命名规则与变量名命名规则相同（必须为标识符）

3.函数名是一个变量（不要轻易对其赋值）

4.函数的自己的名字空间，在函数内部可以访问外部的变量，但外部的语句不能

访问函数内部的变量

5.函数如果不需要传入参数，则参数列表可以为空

6.语句部分不能为空，如果为空需要填充pass语句

函数调用：

函数名（实际调用传递参数）

注：

实际调用传递参数简称‘实参’

调用说明：

1.函数调用是一个表达式

2.如果没有return语句，函数执行完成后返回None对象

3.如果函数需要返回其他的对象，需要用到return语句

练习1：

1.写一个函数mysum，此函数带有两个参数x,y

此函数功能是打印出两个参数x,y的和，即x+y

2.写一个函数print\_even,传入一个参数n代表终止整数（不包含n）

见 code 内的lianxi1

函数内部有独立运行空间，函数外部不能访问函数内部变量

示例见：

code 内的 function\_variable.py

return 语句：

语法：

return [表达式]

注:[]代表其中的内容可以省略

作用：

用于函数中，结束当前函数的执行，返回到调用函数的地方，

同时返回一个对象的引用关系

说明：

1.return语句后跟的表达式可以省略，省略后相当于return None

2.如果函数内没有return语句，则函数执行完最后一条语句后返回None

3.函数调用会返回一个对象的引用

示例见：

code 内的 return.py

练习2：

1.写一个函数mymax，返回两个数的最大值

2.写一个函数input\_number(),此函数用于读取用户输入的多个整数

（输入负数时结束输入），将用户输入的数形成列表返回给调用者

def input\_number():

...

L = input\_number()

print('输入的最大数是：'，max(L))

print('输入的这些数的和是：'sum(L))

见 code 内的lianxi2.py

python函数的参数传递

传递方式：

1.位置传参

2.序列传参

3.关键字传参

4.字典关键字传参

位置传参：

实际调用参数（实参）的对应关系与形式参数（形参）的对应关系是按位置

来依次对应的

序列传参：

序列传参是在函数调用过程中，用\*将序列拆解后按位置进行传递的传参方式

实参和形参通过序列传递和匹配

示例：

def fx(a,b,c):

pass

s1 = [11,22,33]

fx(\*s1)#将序列拆解后按顺序进行位置传参

fx(\*'abcd')#会报错，实参和形参个数不一致

>>>s = 'abcde'

>>>print(\*s)

a b c d e

关键字传参：

是指传参时，按着形参的名称给形参赋值

实参和形参按名称进行匹配

示例：

def fx(a,b,c):

pass

fx(b=22,c=33,a=11)

注：

实参和形参按形参名进行匹配，可以不按位置进行匹配

字典关键字传参：

实参为字典，用\*\*拆解字典后在进行关键字传参

示例：

def fx(a,b,c):

pass

d = {'c':33,'a':11,'b':22}

fx(\*\*d) #拆解字典再依次按关键字传参

说明：

字典的键名和形参必须一致

字典的键名必须为字符串

字典的键名要在形参中存在

综合传参：

函数的传参方式在能确定形参能唯一匹配到相应实参的情况下可以任意组合

注：

通常位置传参和序列传参先传递，其次是关键字传参和字典关键字传参

示例:

def fx(a,b,c,d,e,f):

pass

fx(10,\*[20,30],e=50,\*\*{'d':40,'f':60})

以下的传参方式错误

fx(e=50,\*\*{'d':40,'f':60},10,\*[20,30])

------以下讲函数的形参-------

函数的缺省参数

语法：

def 函数名（形参名1=默认实参1，形参名2=默认实参2，...）:

语句

示例：

def info(name,age=1,address='不详'):

print('我叫',name,'我今年：',age,'岁'，'家庭住址:',address)

info('张飞',30,'中原')

info('Tarena',10)

info('赵云')

说明：

1.缺省参数必须自右至左依次存在

def info(name,age=1,address):#会报错

2.缺省参数可以有0个，1个，多个，甚至全部都有缺省参数

3.缺省参数的绑定对象存在于函数内，同函数的生命周期一致

示例：

def fn(a,l=[]):

l.append(a)

print(l)

l = [1,2,3,4]

fn(5,l)#[1,2,3,4,5]

fn(6,l)#[1,2,3,4,5,6]

fn(1.1)#[1.1]

fn(2.2)#[1.1,2.2]

函数的形参定义方式：

1.位置形参

2.星号元组形参

3.命名关键字形参

4.双星号字典形参

位置形参：

def 函数名（形参名1，形参名2，...）:

语句块

星号元组形参：

语法：

def 函数名（\*元组形参名）：

语句块

作用：

收集多余的位置传参

示例见：

code 内的 star\_tuple\_argument.py

命名关键字形参：

语法：

def 函数名（\*，命名关键字形参）：

语句块

或

def 函数名（\*args，命名关键字形参）：

作用：

所有的命名关键字形参都强制调用者采用关键字传参或字典关键字传参的

方式传递

示例见：

code 内的 keyword\_arguments.py

双星号字典形参：

语法：

def 函数名(\*\*字典形参名):

语句块

作用：

收集多余的关键字传参

注：字典形参名通常叫 kwargs

示例见：

code 内的 dict\_keyword\_argument.py

函数参数说明：

位置形参，缺省参数，星号元组形参，，命名关键字形参，双星号字典形参可以

混合使用

函数参数从左至右的顺序为：

1.位置形参

2.星号元组形参

3.命名关键字形参

4.双星号字典形参

示例：

def fn(a,b,\*args,c,\*\*kwargs):

print(a,b,args,c,kwargs)

fn(1,2,3,4,\*'ab',\*\*{'d':'D'},c=100)

输出：1 2 (3,4,'a','b') 100 {'d':D}

fn(1,2,3,4,\*'ab',\*\*{'d':'D','c':'X'},e=100)

输出：1 2 (3,4,'a','b') X {'d':D,'e':100}

可以接受任意位置传参和关键字传参的函数：

def fn(\*args,\*\*kwargs):

pass

练习3：

1.写一个函数mysum,可以传入任意个实参的数字，返回所有实参的和

def mysum(...):

...

print(mysum(1,2,3,4)) #10

2.已知内建函数max的帮助文档为：

max(...)

max(interable) -> value

max(arg1,arg2,\*arg3) -> value

仿造max，写一个mymax函数，实现功能与max完全相同

测试示例：

print(mymax([6,8,3,5])) #8

print(mymax(100,200)) #200

3.写一个函数min\_max

可以给出任意个数字实参，返回这些实参的最小值和最大值，要求两个数字形成

元组后返回（小在前，大在后）

## Day12

函数变量

函数名是变量，他在创建时绑定一个函数

示例：

def f1(lst=[]):

print('f1函数被调用')

f1()

f1 = None

f1()#出错，f1绑定的是None

示例：

>>>def f1():

print('hello')

>>>def f2():

print('world')

>>>f1,f2 = f2,f1

>>>f1() #world

world

函数可以作为另一个函数的参数（实参）传递

示例：

见 code 内的 函数作为参数.py

函数可以作为另一个函数的返回值

函数可以返回另一个函数

示例：

见 code内的 函数作为返回值.py

函数嵌套定义：

def 语句的作用是用来创建一个函数

def 语句可以写在函数内部，在函数执行时可以动态创建一个函数

示例：

见 code 内的 函数嵌套.py

全局变量和局部变量

局部变量：

1.定义在函数内部的变量称为局部变量

（函数的形参也是局部变量）

2.局部变量只能在函数内部使用

3.局部变量在函数调用时才能够被创建，在函数调用之后会自动销毁

全局变量：

1.定义在函数外部，模块内部的变量称为全局变量

2.全局变量，所有函数都可以直接访问（但函数内部不能直接修改

全局变量的绑定关系）

说明：

1.在函数内首次对变量赋值是创建局部变量，再次为变量赋值是修改

局部变量的绑定关系

2.在函数内部的赋值语句不会对全局变量造成影响

python作用域：

作用域也叫命名空间，是访问变量时查找变量名的范围空间

python的四个作用域LEGB

作用域 英文解释 英文缩写

局部作用域（函数内） Local(function) L

外部嵌套函数作用域 Enclosing function locals E

函数定义所在模块作用域 Global(module) G

python内置模块的作用域 Builtin(python) B

示例：

见 code 内的 作用域.py

变量名的查找规则（顺序）：

1.查找本地变量

2.查找包裹此函数的外部嵌套函数内部的变量

3.全局变量

4.内置变量

globals() / locals() 函数

globals()返回全局作用域内变量的字典

locals() 返回局部作用域内变量的字典

示例：

见 code 内的 glabals\_locals\_function.py

练习1

1.写一个函数mysum，此函数的功能是返回：

1+2+3+4+...+n的和

def mysum(n):

...

print(mysum(100))

2.写一个函数mysum2,此函数可以传入一个参数，两个参数和三个参数：

1）当传入一个参数时，这个参数代表终止数

2）当传入两个参数时，第一个参数代表起始值，第二个参数代表终止值

3）当传入三个参数时，第三个参数代表步长

此函数的功能是返回开始到终止值的和

global 语句

作用：

1.告诉解释执行器，global语句声明的一个或多个变量，这些变量的

作用域为模块级的作用域的变量，也称作全局变量

2.全局声明（global）

将赋值变量映射到模块文件内部的作用域

语法：

global 变量1，变量2

示例：

见 code 内的 global语句.py

说明：

1.全局变量如果要在函数内部被赋值，则必须经过全局声明（否则

会被认为是局部变量）

2.全局变量在函数内部不经过声明就可以直接访问

3.不能先声明局部变量，再用global声明为全局变量，

此做法不符合规则

4.global变量列表里的变量名不能出现在此作用域内的形参列表

示例：

def f1(v):

global v #此时v不能被声明为全局变量

nonlocal 语句

作用：

告诉解释器，nonlocal声明的变量不是局部变量，也不是全局变量，

而是外部嵌套函数内的变量

语法：

nonlocal 变量1，变量2..

示例：

见 code 内的 nonlocal语句.py

说明：

1.nonlocal语句只能在被嵌套函数内部进行使用

2.访问nonlocal变量将对外部嵌套函数作用域内的变量进行操作

3.当有两层或两层以上函数嵌套时，访问nonlocal变量只对

最近一层的变量进行操作

4.nonlocal语句的变量列表的变量名不能出现在此函数的参数列表里

练习2：

1.创建一个列表L=[]

写一个函数input\_number读取数据放入列表L中

程序如下：

L=[]

def input\_number():

...

while True:

i = int(input('输入数字(-1结束)：'))

...

input\_number()

print('刚才输入的整数值是：',L)

2.写一个函数isprime(x)

判断是否为素数，如果为素数返回True,否则返回False

测试代码：

if isprime(5):

print('5素数')

3.写一个函数prime\_m2n(m,n)

返回从m开始到n结束范围内素数的列表，并打印

l= prime\_m2n(10,20)

print(l) #[11,13,17,19]

4.写一个函数primes(n),返回小于n的所有的素数的列表

l = prime(10)

print(l) #[2,3,5,7]

## Day13

lambda 表达式（又匿名函数表达式）

作用：

创建一个匿名函数对象

同def类似，但不提供函数名

语法：

lambda [形参1，形参2...]:表达式

示例：

def myadd(x,y):

return x+y

可以改写为：

>>>myadd = lambda x,y:x+y

>>>print(myadd(1,2))

3

语法说明：

1.lambda只是一个表达式，它用来创建一个函数对象

2.当lambda表达式调用时，先执行冒号后的表达式，并返回表达式的结果的引用

3.lambda表达式创建的函数只能包含一条表达式

4.lambda比函数简单，且可以随时创建和销毁，有利于减少程序的耦合度

练习1：

1.写一个lambda表达式，判断这个数的2次方+1能否被5整除，如果能返回True,否则

返回False

fx = lambda n:...

print(fx(4)) #False

print(fx(3)) #True

2.写一个lambda表达式，求两个变量的最大值：

mymax = lambda ...

print(mymax(55,63)) #63

见 code 内的lianxi1.py

看懂下面的程序在做什么：

def fx(f,x,y):

r = f(x,y)

print(r)

fx((lambda a,b:a+b),1,2) #3

fx((lambda x,y:x\*\*y),3,4) #81

eval 和 exec 函数

eval函数：

作用：

把一个字符串当成一个表达式执行，返回表达式执行后的结果

格式：

eval(source,globals=None,locals=None)

示例：

x = 1

y = 2

s = 'x+y'

v = eval(s)

print(v) #3

print(eval('x+y',{'x':1,'y':2})) #3

print(eval('x+y',{'x':1,'y':2},{'x':3,'y':4})) #7

优先调用locals内的x,y

print(eval('x+y',{'x':1,'y':2},{'x':3})) #5

exec函数：

作用：

把一个字符串当成程序来执行

格式：

exec(source,globals=None,locals=None)

示例：

>>>s = 'x=5;print('hello');x += 1;print(x)'

>>>exec(s)

hello

6

自己写一个程序，解释执行用户输入的任何语句

见 code 内的 exec语句.py

函数式编程：

是指用一系列函数解决问题

函数是一等公民

1.函数本身是对象，可以赋值给变量，赋值后变量绑定函数

2.允许将函数作为实参传入另一个函数

3.允许函数返回一个函数

好处：

1.每一个函数完成细小的功能，一系列函数的任意组合可以解决大的问题

2.函数仅接收输入并产生输出，不会影响其他全局变量的状态

练习：

求：1+1/2+1/4+...+1/2\*\*n

print(sum[1/2\*\*x for x in range(n+1)])

高阶函数：High Order Function

满足下列条件中的一个的函数即为高阶函数

1.函数接收一个或多个函数作为参数传入

2.函数返回一个函数

python内置（builtins）的高阶函数：

map, filter, sorted

map函数：

map(function,\*iterables) 用函数对可迭代对象中的每一个元素作为参数计算出

新的可迭代对象，当最短的一个可迭代对象不再提供数据时，此可迭代对象生成

结束

示例1：

生成一个可迭代对象，要求此可迭代对象可以生成1-9自然数的平方

def power2(x):

return x\*\*2

for x in map(power2,range(1,10)):

print(x)

求以上数据的和

print(sum(map(power2,range(1,10))))

示例2：

生成一个可迭代对象，要求此可迭代对象生成如下数据：

1\*\*4，2\*\*3，3\*\*2，4\*\*1

>>>list(map(pow,range(1,5),range(4,0,-1)))

[1,8,9,4]

练习：

1.求1\*\*2+2\*\*2+3\*\*2+...+9\*\*2的和

1)>>>def power2(x):

return x\*\*2

>>>sum(map(power2,range(1,10)))

2)>>>sum(map(lambda x: x\*\*2,range(1,10)))

2.求1\*\*3+2\*\*3+3\*\*3+...+9\*\*3的和

>>>sum(map(lambda x:x\*\*3,range(1,10)))

3.求1\*\*9+2\*\*8+3\*\*7+...+9\*\*1的和

>>>sum(map(pow,range(1,10),range(9,0,-1)))

filter函数：

格式：

filter(func,iterable)

作用：

筛选可迭代对象iterable中的数据，返回一个可迭代对象，此可迭代对象将

对iterable进行筛选

函数func将对iterable中的每个元素进行求值，返回False时将此数据丢弃，

返回True时，则保留此数据

示例：

#isodd函数 判断x是否为奇数，是奇数返回True

def isodd(x)：

return x%2 == 1

#打印10以内的奇数

for x in filter(isodd,range(1,10)):

print(x)

#生成10以内的偶数列表

list(filter(lambda x:x%2 == 0,range(1,10)))

sorted 函数：

作用：

将原可迭代对象的数据进行排序，生成排序后的列表

格式：

sorted(iterable,key=None,reverse=False)

参数说明：

iterable 可迭代对象

key 绑定函数，此函数用来提供一个排序的依据

reverse 标志用来设置是否降序排序（从小到大）

示例：

l = [5,-2,-4,0,3,1]

l1 = sorted(l)

l1 = [-4,-2,0,1,3,5]

l2 = sorted(l,reverse=True)

l2 = [5,3,1,0,-2,-4]

#如何排序如下：

[0,1,-2,3,4,5]?

l3 = sorted(l,key=abs) #[0,1,-2,3,4,5]

#按名字长度排序

name = ['zhang','zhao','li','sun']

sorted(name,key=len)

#以名字倒过来的顺序依据来排序

方法一

def f(x):

return x[::-1]

sorted(name,key=f)

方法二

sorted(name,key=lambda x:x[::-1])

递归函数 recursion

函数直接或间接的调用自身

示例：

#直接调用自身

def f():

f() #调用自己

f()

print('递归完成')

#间接调用自身

def fa():

fb()

def fb():

fa()

fa()

递归说明：

1.递归一定要控制递归的层数，当符合某一条件时要终止递归

2.几乎所有的递归都能用循环来代替

递归的优缺点：

优点：

递归可以把问题简单化，让路劲更为清晰，代码更为简洁

缺点：

递归因系统环境影响大，当递归深度太大时，可能会得到不可预知的结果

示例：

#会无限循环，崩溃

def story():

print('讲故事')

story()

print('讲完了')

story()

#此递归用参数来控制最大次数

def story(times):

print('第',times,'遍')

print('将故事')

if times >= 3:

return

story(times+1)

print('第',times,'遍故事讲完')

story(1)

递归函数的实现方法：

先假设函数已经实现

示例：

求：100+99+98+...+2+1的和

分析：

先假设mysum(x)已经完成，且能求x+(x-1)+...+1的和

def mysum(x):

#先判断终止条件

if x == 1:

return 1

return x + mysum(x-1)

print(mysum(100)) #5050

练习2：

1.编写函数 myfac(x),计算x阶乘x!

5!=5\*4\*3\*2\*1

print(myfac(5)) #120

2.写程序算出1-20的阶乘的和

思考：

能否用函数式编程中的高阶函数实现？

3.已知有列表：

l=[[3,5,8],10,[[13,14],15],18]

1)写一个函数print\_list(lst)打印出列表中所有数字

print\_list(l)

2)写一个函数sum\_list(lst) 返回列表中所有数字的和

print(sum\_list(l)) #86

提示：

type(x) 可以返回一个变量的类型

如：type(20) is int #True

type([1,2,3]) is list #True

4.开始学生信息管理项目

需要两个函数：

input\_student()返回学生信息的字典的列表

output\_student(lst) 以表格式打印学生信息

学生信息：

姓名(name) 字符串

年龄(age) 整数

成绩(score) 整数

input\_student()

调用时等待用户输入学生信息，当输入姓名为空时结束，最后将学生信息形

成字典后存入列表中返回

def input\_student():

...

l = input\_student()

请输入学生姓名：xiaozhang

请输入学生年龄：20

请输入学生成绩：98

请输入学生姓名：xiaoli

请输入学生年龄：23

请输入学生成绩：100

请输入学生姓名：<回车>输入结束

此时列表数据为：

l = [{'name':'xiaozhang','age':20,'score':98},

{'name':'xiaoli','age':23,'score':100}]

def output\_student(lst):

...

print(l)#打印出字典的列表

output\_student(l) #打印出学生信息的表格如下：

+---------------+-----------+----------+

| name | age | score|

+----------------+----------+----------+

| xiaozhang | 20 | 98 |

| xiaoli | 23 | 100 |

+----------------+----------+----------+

## Day14

闭包 closure

将内嵌函数的语句和这些语句的执行环境打包在一起后，得到的函数对象

称为闭包

闭包必须满足以下三个条件：

1.必须有一个内嵌函数

2.内嵌函数必须引用外部函数中的变量

3.外部函数返回值必须是内嵌函数

示例：

见 code 内的 closure.py

说明:

如果一个内嵌函数访问了外部嵌套函数作用域内的变量，则这个内嵌

函数叫做闭包函数

装饰器 decorators(专业提高篇)

1.装饰器是一个函数，这个函数的主要作用是包装另一个函数或类

2.包装的目的是在不改变原函数的情况下改变被包装对象的行为

函数装饰器 function decorators

函数装饰器是指装饰是一个函数，传入的是一个函数，

返回的也是一个函数

示例：

deco1.py

函数的文档字符串：

语法：

def 函数(参数列表)：

'''函数的文档字符串'''

语句块

说明：

文档字符串通常来用来说明本函数的功能和使用方法

在交互模式下，输入help(函数名)，可以查看函数的'文档字符串'

函数的\_\_doc\_\_属性：

函数内的第一次未赋值给任何变量的字符串是此函数的文档字符串，此字符

串会自动赋值给函数的\_\_doc\_\_属性

示例：

def cba():

'只是文档字符串'

pass

print(cba.\_\_doc\_\_) #只是文档字符串

函数定义语句的完整语法：

[@装饰器1]

[@装饰器1]

...

def 函数名([位置形参],[\*元组形参],[命名关键字形参],[\*\*字典形参]):

'文档字符串'

语句块

模块 module

什么是模块：

模块是一个包含有一系列变量，函数，类等组成的程序组

模块是一个文件，模块文件通常以.py结尾

作用：

1.让一些相关的变量，函数，类等有逻辑的组织在一起，

使逻辑结构更加清晰

2.模块中的变量，函数和类可提供给其他模块使用

模块的分类：

1.内置模块(biultins),在解释器的内部可以直接使用

2.标准库模块，安装python时已安装且可直接使用

3.第三方模块(通常为开源),需要自己安装

4.用户自己编写的模块(可以作为其他人的第三方模块)

模块的导入 import

import 语句

语法：

import 模块名1 [as 模块新名1][,模块名2 [as 模块新名2...]

示例：

import math #导入数学模块

import sys ,os #导入sys,os模块

作用：

将某模块整体导入到当前模块中

用法：

模块名.属性名

help(模块名) 查看模块内的变量，函数，类等

练习：

1.输入圆的半径，打印圆的面积

import math

r = int(input('半径：'))

area = math.pi \* math.pow(r,2)

2.输入圆的面积，打印圆的半径

import math

a = int(input('面积：'))

r = math.sprt(a/math.pi)

from import 语句

语法：

from 模块名 import 模块属性名1 [as 属性新名1],

模块属性名2 [as 属性新名2],...

作用：

将某模块中内的一个或多个属性导入到当前模块的作用域

示例：

from math import pi,sqrt

from math import factorial as fac

可以直接使用导入的函数名或变量名

from import \* 语句

语法：

from 模块名 import \*

作用：

将某模块的所有属性导入到当前模块

示例：

from math import \*

dir 函数

格式：

dir([对象]) -->返回一个字符串列表

说明：

如果没有参数调用，则返回当前作用域内的所有变量的列表

如果给定一个对象作为参数，则返回这个对象的所有变量的列表

1.

2.

3.

常见模块见：

python3笔记附件

练习1：

1.编写函数fun(n) 其功能是计算并输出下列多项式的和

Sn=1+1/1!+1/2!+1/3!+...+1/n!

2.编写函数fun(n),求：

s = 1+x\*\*1/1!+x\*\*2/2!+x\*\*3/3!+...+x\*\*n/n!

print(fun(3.1,10))

练习2：

1.已知有五位朋友在一起，第一位10岁，以后依次比前一位大2岁，求第五位年龄

2.改写之前的学生信息管理项目源码，要求带有操作界面：

+------------------------------------------------+

| 1)添加学生信息 |

| 2)显示所有学生的信息 |

| 3)删除学生信息 |

| 4)修改学生成绩 |

| 5)按学生成绩高-低显示学生信息|

| 6)按学生成绩低-高显示学生信息|

| 7)按学生年龄高-低显示学生信息|

| 8)按学生年龄低-高显示学生信息|

| 9)退出 |

+-------------------------------------------------+

请选择：

## Day15

时间模块 time

见 python3笔记附件 中的 “时间模块time.html”

练习1：

1.写一个程序，以电子时钟的显示时间：

HH:MM:SS

2.写一个程序，输入你生日

1）计算出你出生那天是星期几？

2）计算出你已经出生了多少天？

见 code 内的 lianxi1.py

系统模块 sys

见 python3笔记附件 中的 “系统模块sys.html”

自定义模块

示例见：

mymod.py #被导入模块

模块调用测试.py #主模块

导入模块时的路劲索引顺序：

1.索引程序的运行时路径（当前路径）

2.搜索sys,path里提供的路径

3.搜索内置模块

模块化编程的优点：

1.有利于多人合作开发

2.是代码更加易于维护

3.提高代码的复用率

4.模块化编程有助于解决函数名和变量名冲突（重名）问题，模块内的变量的作用域

为模块内全局

模块的加载过程

1.在模块导入时，模块的所有语句都会执行

2.如果一个模块已经导入，则再次导入是不会重新执行模块内的语句

模块的重新加载：

import imp

imp.reload(mymod) #重新加载已经加载过的mymod模块

模块被导入和执行的过程：

1.先搜索相关的路径找模块（.py）

2.判断是否由此模块对应的.pyc文件，如果没有此文件，则用.py文件生成

相应的.pyc文件再进行加载

3.如果文件已经存在，则判断.pyc文件和.py文件的修改时间，在决定是否

重新生成.pyc文件

模块的编译 compile

编译 解释执行

mymod.py --->> mymod.pyc ------>> python3

模块的文档字符串：

模块内第一行没有赋值给任何变量的字符串为文档字符

此文档字符串可以用help函数查看

示例见：

code 内的 mymod.py

\_\_file\_\_属性

用来绑定模块对应的文件路径名

示例：

import mymod

print(mymod.\_\_file\_\_)

模块的 \_\_name\_\_属性：

用来记录模块自身的名字

作用：

1.记录模块名

2.用来判断是否为主模块

\_\_name\_\_说明：

1.当此模块为主模块(也就是第一个运行的模块)运行时，\_\_name\_\_绑定'\_\_main\_\_'

2.当此模块不是主模块时，模块名为文件名去掉.py

模块的\_\_all\_\_列表

模块中的\_\_all\_\_列表是一个用来存放可导出属性的字符串列表

作用：

当用from import \*语句导入模块时，只导入\_\_all\_\_列表内的属性

示例见：

code 内的\_\_all\_\_属性.py

模块的隐藏属性：

模块中以'\_'开头的属性，在from import \*语句导入时，将不会导入，通常称这些属性

为隐藏属性

示例见：

code 内的 mymod2.py

标准库模块：

随机模块 random

作用：

用于模拟或生成随机数的模块

模块说明见

python3笔记附件 内的 随机模块random.html

练习2：

假设可以作为密码的字符有：

a-z

A-Z

0-9

下划线

写一个程序，随机生成六位密码

见 code 内的 lianxi2

包（模块包）package

包是将模块以文件夹的组织形式进行分组管理的方法

作用：

1.将一系列模块进行分类管理，有利于防止命名冲突

2.可以在需要时加载一个或部分模块而不是全部模块

包示例：

mypack/

\_\_init\_\_.py

menu.py

games/

\_\_init\_\_.py

contra.py #魂斗罗

supermario.py#超级玛丽

office/

\_\_init\_\_.py

excel.py

word.py

powerpoint.py

包的导入语法：

import 包名 [as 包别名]

import 包名.模块名 [as 模块新名]

import 包名.子包名.模块名 [as 模块新名]

其他两种方法和导入模块方法一样

练习3：

1.编写一个闹钟，启动时设置定时时间（小时和分钟）

到时间后打印'时间到...'然后退出程序

2.模拟斗地主发牌，扑克牌共54张：

花色：

黑桃（'\u2660'）,红桃（'\u2663'）

方块（'\u2665'）,梅花（'\u2666'）

数值：

A2-10JQK

大小王

三个人，每人发17张牌，底牌留三张：

输入回车，打印第一个人的17张牌

输入回车，打印第二个人的17张牌

输入回车，打印第三个人的17张牌

再输入回车，打印出三张底牌

## Day16

包（模块包）package

包是将模块以文件夹的组织形式进行分组管理的方法

作用：

1.将一系列模块进行分类管理，有利于防止命名冲突

2.可以在需要时加载一个或部分模块而不是全部模块

包示例：

package/

\_\_init\_\_.py

menu.py

games/

\_\_init\_\_.py

contra.py #魂斗罗

supermario.py#超级玛丽

office/

\_\_init\_\_.py

excel.py

word.py

powerpoint.py

说明：

当子包或子包内的模块被导入时，上一层的包也会被先导入

例如：

import package.games.contra

此时会先导入package，再导入games,再导入contra

\_\_init\_\_.py文件

是常规包内必须存在的文件，\_\_init\_\_.py会在导入包时被自动调用

作用：

1.编写此包的内容

2.在内部添加包的文档字符串

3.在\_\_init\_\_.py文件内可以加载此包所依赖的其它模块

示例见：

code 内的 package/\_\_init\_\_.py

包的\_\_all\_\_列表：

作用：

用来记录此包中有哪些子包或模块需要导入，当用'from 包 import \*'语句

导入时，只查找\_\_all\_\_中的模块或子包

此语句写在包内的\_\_init\_\_.py文件内

\_\_all\_\_ = ['包名1(或模块名1)'，'包名2(或模块名2)']

说明：

\_\_all\_\_列表只在from xxx import \*语句时起作用

示例见：

code 内的 \_\_all\_\_列表测试.py

包的相对导入：

包的相对导入是指包内的模块的相互导入

语法：

from 相对路径包或模块 import 属性或模块名

或

from 相对路径包或模块 import \*

相对路径：

在from xxx import 语句中，xxx部分可以使用相对路径

. 代表当前目录

.. 代表上一级目录

... 代表上二级目录

.... 依次类推

注：

1.相对导入时不能超出包的外部

2.相对导入避免循环导入

示例见：

code 内的 包的相对导入.py

异常（基础）exception

1.什么是错误

错误是指由于逻辑或语法等导致一个程序无法正常执行的问题

错误的特点：

有些错误无法预知

2.什么是异常

异常是程序出错时标识的一种状态，当异常发生时，程序不会再向下执行，而

转去调用此函数的地方等待处理错误并恢复为正常状态

异常的作用：

通知上层调用者有错误产生需要处理

用作信号

try-except语句的两种语法:

try:

可能触发异常的语句

except 错误类型1 [as 变量1]：

异常处理语句1

except 错误类型2 [as 变量2]：

异常处理语句2

except(错误类型3，错误类型4，...):

异常处理语句3

...

except: #除了以上的错误以外的任何错误都可以处理

异常处理语句other

[esle:

未发生异常的语句]

[finally:

最终语句]

作用：

尝试捕捉异常，将程序转为正常状态并继续执行

示例见：

code 内的 try\_except1.py

try-except语句语法说明：

1.as 子句是用于绑定错误对象的变量，可以省略

2.except 子句可有一个或多个，但至少要有一个

3.else 子句最多只能有一个，也可以省略

4.finally子句最多只能有一个，也可以省略

练习1：

写一个函数get\_score()来获取用户输入的学生成绩（0-100的整数），如果输入

出现错误，则此函数返回0，如果用户输入的数是0-100之间的数，返回这个数

def get\_score():

...

score = get\_score()

print('学生成绩是：',score)

见 code 内的 lianxi1.py

try-finally语句：

语法：

try:

可能触法异常的语句

finally:

最终语句

语法说明：

finally 子句不可以省略

一定不存在except子句

作用：

通常用try-finally语句来做触发异常时必须要处理的事件，无论异常是否

发生，finally子句都会被执行

注：

try-finally 语句不会改变程序的（正常/异常）状态

示例见：

code 内的 try-finally.py

raise 语句：

作用：

触发一个错误，让程序进入异常状态：

语法：

raise 异常类型

或

raise 异常对象

示例见：

code 内的 raise.py

assert 语句（断言语句）

语法：

assert 真值表达式，错误数据（通常是字符串）

当真值表达式为False时，用错误数据创建一个AssertionErroe类型的错误，

并进入异常状态

等同于：

if 真值表达式 == False:

raise Assertion(错误数据)

示例见：

code 内的 assert.py

为什么要用异常处理机制：

在程序调用层数较深时，向主调函数传递错误信息需要层层的return返回比较麻烦，

所以用异常处理机制

练习2：

1.一个球从100米的高度落下,每次落地后反弹高度为原高度的一半，在落下，

1）写程序算出皮球从10次落地后反弹高度是多少？

2）球共经过多少米

2.打印九九乘法表

3.分解质因数：

如90 = 2\*3\*3\*5

(质因数是指最小能被原整数整除的素数（不含1）)

## Day17

迭代器 iterator 和 生成器 generator

什么是迭代器：

1.迭代器是访问可迭代对象的一种方式，用迭代器可以访问可迭代对象

2.迭代器是指iter(可迭代对象)返回的对象

3.迭代器可以用next(it)函数获取可迭代对象的数据

迭代器函数iter和next

iter(iterable) 从可迭代对象中返回一个迭代器，iterable必须是能提供一个

迭代器的对象

next(iterator) 从迭代器iterator中获取下一个记录。如果无法获取下一条记录，

则触发StopIteration

迭代器说明：

迭代器只能往前取值，不可后退

用iter函数可以返回一个可迭代对象的迭代器

示例：

>>>l=[1,2,3,4]

>>>it = iter(l) #用iter返回一个迭代器用it绑定

>>>next(it) #用next(it)来获取l中的元素

1

>>>next(it)

2

>>>next(it)

3

>>>next(it)

4

>>>next(it)#出现异常

StopIteration

从迭代器获取数据

it = iter(l)

while True:

try:

x=next(it)

print(x)

except StopIteration:

break

练习1：

已知有一个集合：

s={'gs','js','zg','ny'}

1.用for语句遍历集合中的元素并打印

2.将上面的for语句改写为while语句实现上面同样的功能

提示：用iter和next函数实现

见 code 内的 lianxi1.py

生成器 generator(python2.5及之后版本)

生成器是能够动态提供数据的对象，生成器的对象也是可迭代对象

生成器有两种：

生成器函数

生成器表达式

生成器函数定义：

含有yield语句的函数是生成器函数，此函数被调用将返回一个生成对象

yield 翻译为（产生或生成）

yield 语句

语法：

yield 表达式

说明：

yield 只用于def函数中，目的是将此函数作为生成器函数使用

yield 用来生成数据，供next(it)函数使用

示例见：

code 内的 myyield.py

练习2：

#写一个生成器函数my\_integer(n)生成1到n的整数：

def my\_integer(n):

i = 1:

while i < n:

yield i

i += 1

for x in my\_integer(5):

print(x) #1,2,3,4

练习3：

写一个生成器函数myodd(start,stop)用于生成start开始到stop结束的所有奇数

l=[x for x in myodd(1,10)]

print(l) #[1,3,5,7,9]

见code 内的 lianxi3.py

生成器表达式：

语法：

（表达式 for 变量 in 可迭代对象 [if 真值表达式]）

注：[]内的if部分可以省略

作用：

用推导式形式生成一个新的生成器

示例：

gen = (x\*\*2 for x in range(1,5))

it = iter(gen)

next(it) #1

next(it) #4

next(it) #9

next(it) #16

next(it) #StopIteration

迭代工具函数：

迭代工具函数的作用是生成一个个性化的可迭代对象

zip(iter1[,iter2,...])

返回一个zip对象，此对象用于生成一个元组，

此元组的个数是最小的可迭代对象决定，元组内容是可迭代对象iter1和iter2

中元素的组合

示例：

>>>it=zip([1,2,3],('a','b','c','d'))

>>>for x in it:

print(x)

(1,'a')

(2,'b')

(3,'c')

>>>dict(it)

{1: 'a', 2: 'b', 3: 'c'}

enumerate(iterable[,start])

生成带索引的枚举对象，返回的迭代类型为索引-值对（index-value）对，

默认索引从零开始，也可以用start制定

示例：

>>>name = ('a','b','c','d')

>>>for i in enumerate(name):

print(i)

(0, 'a')

(1, 'b')

(2, 'c')

(3, 'd')

练习4：

写一个程序，输入任意行的文字数据，当输入空行时结束输入，打印带有行号的输入

结果：

见：code 内的 lianxi4.py

字节串和字节数组：

字节串（也叫字节序列）bytes

作用：

存储以字节为单位的数据

说明：

字节串是不可改变的序列

字节是0-255之间的整数

创建空的字节串的字面值

b''

b""

b''''''

b""""""

创建非空的字节串的字面值：

B=b'hello'

B=b"hello"

B='''abcd'''

B="""abcd"""

B=b'abc\n123'

B=b'\x41\42'

字节串的构造函数 bytes

bytes() 生成一个空的字节串，等同于b''

bytes(整数可迭代对象) 用可迭代对象初始化一个字节串

bytes(整数n) 生成n个值为0的字节串

bytes(字符串，encoding='utf-8') 用字符串的转换编码生成一个字节串

bytes 的运算

+ += \* \*=

< <= > >= == !=

in/not in

索引和切片

in / not in

B = b'ABCD'

0x41 in B #True

bytes 与 str 的区别：

bytes 存储字节（0-255）

str 存储字符（Unicode值）

bytes 与 str转换

编码(encode)

str ---------> bytes

b=s.encode(encoding='utf-8')

解码(decode)

bytes ---------> str

s=b.encode(encoding='utf-8')

示例：

b = '你好'.encode('utf-8')

print(b)

s = b.decode('utf-8')

print(s)

字节数组 bytearray

可变的字节序列

创建函数 bytearry

bytearray() 生成一个空的字节数组

bytearray(整数可迭代对象) 用可迭代对象初始化一个字节数组

bytearray(整数n) 生成n个值为0的字节数组

bytearray(字符串，encoding='utf-8') 用字符串的转换编码生成一个字节数组

操作：

+ += \* \*=

< <= > >= == !=

in/not in

索引和切片，可以索引、切片赋值

例：

>>>ba = bytearray(b'abcdefg')

>>>ba[0]=65 #ba=bytearray(b'Abcdefg')

>>>ba[1::2]=bytearray(b'BDF') #ba=bytearray(b'ABcDeFg')

练习5：

1.用生成器函数生成斐波那契数列的前n个数

1，1，2，3，5，8，13...

def fibonacci(n):

...

yield ..

1)输出前20个数

2)求前30个数的和

2.写程序打印杨辉三角（只打印6层）

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1. 5 10 10 5 1

## Day18

bytearray 方法

Ba = bytearray('abcde',encoding='utf-8')

Ba.clear()清空字节数组

Ba.append(n)追加一个字节（n为0-255的整数）

Ba.remove(value)删除第一个出现的字节，如果没有出现，则触发ValueError错误

Ba.reverse()字节的顺序反转

Ba.decode(encoding='utf-8')将bytearray转换为字符串（返回）

Ba.find(sub[,start[,end]])查找sub字节数组，返回索引

文件

什么是文件

文件是用于数据存储的单位

文件通常用来长期存储设置

文件中的数据是以字节为单位进行顺序存储的

内存的读写速度快，磁盘的读写速度慢

文件的操作步骤：

打开文件

读写文件

关闭文件

文件的打开函数 open

open(file,mode='rt')

用于打开一个文件，返回 文件流 对象，如果打开失败会触发IOError错误

文件的关闭方法

f.close() 关闭文件，释放系统资源

示例见：

file\_open.py

文件操作分为两种类型的操作：

文本文件操作

二进制文件操作

文本文件模式：

mode = 't'

1.默认文件中存储的内容为字符数据，以行为单位读取，采用系统默认的

换行符进分割

2.对于文本文件的读写，需要用字符串(str)进行读取和写入数据

各操作系统默认的换行符：

Linux 换行符 '\n'

windows 换行符 '\r\n'

旧的Macintosh 换行符 '\r'

新的Mac os 换行符 '\n'

练习1：

读取data.txt文件的内容，并将打印如下格式：

姓名：小王 电话：13777777777

见 code 内的 lianxi1

练习2：

1）写一个程序，读入任意行的文字信息，当输入空行是结束输入，将读入的字符串

存在列表中，然后将列表里的内容写入文件input.txt中

见 code 内的 lianxi2-1

2）写一个程序，从input.txt中读取之前输入的数据，存入列表中，再加上行号进行

打印显示

第1行：xxxxx

第2行：xxxxx

...

见 code 内的 lianxi2-2

二进制文件操作：

打开模式：'b'

默认文件中存储的都是以字节为单位的二进制数据，通常有人为规定的格式，

需要以字节为单位进行读写

对二进制文件的读写，需要用字节串(bytes)进行操作

f.read(n)

对二进制文件 返回字节串，n为最大字节数

f.readline()

对二进制文件 返回字节串

f.readlines()

对于二进制文件 返回字节数组

f.write(x)

对二进制文件 x必须为字节串

f.seek(offset,whence=0) 函数

offset 偏移量

大于0的数代表向文件末尾方向移动

小于0的数代表向文件头方向移动

whence 相对位置

0 代表从文件头开始偏移

1 代表从当前位置开始偏移

2 代表从文件尾开始偏移

示例：

f.seek(10,0) #将光标从文件头向右移动了10位

f.seek(5,1) #将光标从当前位置向右移动了5位

f.seek(-10,2)#将光标从文件尾向作移动了10位

汉字编码：

国标系列：

GB18030（2字节或4字节编码，共27533）

GBK (2字节编码，共21003个)

GB2312(2字节编码，共6763个汉字)

常用于windows

国际标准：

UNICODE16/UNICODE32 <<--->> UTF-8

常用于Linux /Mac OS X /IOS /Android

编码字符串:(用于encode和decode方法中)

'gb2312'

'gbk'

'gb18030'

'utf-8'

'ascii'

编码注释：

在源文件的第一行或第二行写入如下格式的内容

# -\*- coding:gbk -\*-

或

# -\*- coding:utf-8 -\*-

是告诉python解释执行器当前文件的编码是什么

练习3：

1.修改原来的学生管理系统的代码，将其分为模块

建议：

1.把显示菜单放在一个模块内

2.把对学生的操作放在一个模块内

3.把主事件循环while True 选择输入放在一个模块内

2.在菜单上添加两个功能：

9）保存学生信息到文件（si.txt）

10)从文件中读取数据（si.txt）

建议用'逗号'分隔数据：

#file: si.txt

xiaozhang,21,98

xiaoli,22,100

...

## Day19

标准输入输出文件

sys.stdin

sys.stdout

sys.stderr

模块名：sys

说明：

每一个python程序启动后都会有以上三个文件已经打开供我们使用

示例见：

stdin.py

stdout.py

linux下 Ctrl+D输入文件结束符

面向对象编程

类(class) 和 对象(object)

什么是对象：

对象是指现实中的物体或示例

什么是面向对象：

把一切看成对象（示例），对象和对象之间用方法（行为）建立关联关系

什么是类：

拥有相同属性和行为的对象分为一组，即为一个类，类是用来描述对象的工具

类的创建语法：

class 类名（继承列表）:

'''类文档字符串'''

实例方法（类内的函数method）的定义

类变量(class variable)定义

类方法(@classmethod)定义

静态方法(@staticmethod)定义

类的作用：

可以用类创建一个或多个此类的对象（实例）

类内的变量和方法能被此类所创建的所有实例所共同拥有

说明：

类名必须是标识符（与变量名命名规则相同，建议首字母大写）

类名实质上就是变量，它绑定一个类实例

类的定义最后面要加两个空格以告诉解释执行器，类的定义已经结束

示例：

class Dog:

pass

术语

类对 象 实例

class object instance

构造函数：

构造函数的调用表达式

类名([创建传参列表]) #[]里的内容可以省略

作用：

创建这个类的实例对象，并返回此实例对象的引用关系

示例：

class Dog: #定义一个类

pass

dog1 = Dog() #用类来创建一个对象用dog1绑定

print(id(dog1))

dog2 = Dog() #用类来创建一个对象用dog2绑定

print(id(dog2))

print(dog1 is dog2) #False

实例说明：

实例有自己的作用域或名字空间，可以为该示例添加实例变量(也叫属性)

实例可以调用类的方法

实例可以访问类中的类变量

实例变量(属性attribute)

每个实例可以有自己的变量，称为实例变量(属性)

实例的使用语法：

实例.属性名

属性的赋值规则:

(同变量规则相同)

1.首次为属性赋值则创建此属性

2.再次为属性赋值则改变属性的绑定关系

实例：

class Dog:

pass

dog1 = Dog()

dog1.kinds = '京巴' #为dog1绑定的实例添加kinds属性

dog1.color = '白色' #添加属性

print(dog1.kinds,dog1.color) #访问属性

dog1.color = '黄色' #修改dog1.color的绑定关系

print(dog1.color)

练习1：

写一个Student类，此类的对象有属性name,age,score用来保存学生的

姓名，年龄，成绩

1)写一个函数input\_student读入n个学生的信息，用对象来存储这些信息(不用字典)

并返回对象的列表

2)写一个函数output\_student打印这些学生信息(格式不限)

示意：

class Student:

pass

def input\_student():

...#获取学生信息，形成列表返回

def output\_student(lst):

#打印学生信息

def main():

l = input\_student()

output\_student(l)

main()

删除属性：

del 语句

语法：

del 对象.属性名

示例：

class Student:

pass

stu = Student()

stu.name = 'zhang'

print(stu.name)

del stu.name #删除此属性

print(stu.name) #属性错误，因为属性已经不存在

实例方法 method

语法：

class 类名(继承列表)：

def 实例方法名(self,参数1，参数2，...):

语句块

作用：

用于描述一个对象的行为，让此类型的全部对象都拥有相同的行为

说明：

实例方法的实质是函数，是定义在类内的函数

实例方法的第一个参数代表调用这个方法的实例，一般命名为'self'

实例方法的调用语法：

实例.实例方法名(调用参数)

或

类名.实例方法名(实例，调用参数)

示例见：

code 内的 instance\_method.py 和 instance\_method2.py

初始化方法：

作用：

对新创建的对象添加属性等必须的资源

语法形式：

class 类名：

def \_\_init\_\_(self[,参数列表])：#[]内的内容可以省略

语句块

说明：

1.初始化方法名必须为\_\_init\_\_不可改变

2.初始化方法会在构造函数创建实例后自动调用，且将实例自身通过第一个

参数self传入\_\_init\_\_方法

3.构造函数的实参将通过\_\_init\_\_方法的参数列表传到\_\_init\_\_方法中

4.初始化方法内如果需要return语句返回，则必须返回None

实例见：

code 内的 init\_method.py

析构方法：

语法形式：

class 类名：

def \_\_del\_\_(self):

语句块

作用：

在对象被销毁之前被调用，主要负责清理对象所占用的资源

说明：

python建议尽可能少的在析构方法内做事情，因为销毁时间难以确定

示例见：

code 内的 del\_method.py

预置的实例属性：

\_\_dict\_\_属性：

用于绑定一个存储此实例自身变量的字典

\_\_class\_\_属性

用于绑定创建此实例的类

实例：

class Dog:

pass

dog1 = Dog()

print(dog1.\_\_dict\_\_) #{}

dog1.color = '白色'

print(dog1.\_\_dict\_\_) #{'color': 'haung'}

print(dog1.\_\_class\_\_)

用于类的函数：

isintance(obj,class\_or\_tuple)

返回这个对象obj是否是某个类的对象，或者某些类中的一个类的对象，

如果是则返回True,否则返回False

type(obj)返回对象的类

对象：

属性(对象拥有的名词) 用实例变量存储

行为(对象拥有的动作) 用方法表示

练习2：

1.定义一个类Human(人类)

有三个属性：

姓名：name

年龄：age

家庭住址:address（可以省略）

有如下方法：

show\_info(self) 用来显示人的信息

updata\_age(self) 方法用来让这个人的年龄增加一岁

def input\_human():

输入此人的信息，新名为空结束

def main():

docs = input\_human()

for h in docs:

h.show\_info() #列出所有人的信息

for h in docs:

h.update\_age() #让所有人都长一岁

for h in docs:

h.show\_info() #再次列表所有人的信息

mian()

## Day20

类变量：

1.类变量是类的属性，此属性属于类，不属于此类创建的实例

2.类变量可以通过此类的实例直接访问

3.类变量可以通过此类的对象的'\_\_class\_\_'属性间接访问

示例：

class Human:

total\_count = 0 #类变量，用于记录对象的个数

print(Human.total\_count) #0

h1 = Human()

print(h1.total\_count) #0 不会出错会访问类变量

Human.total\_count = 1 #修改类变量

h1.total\_count = 1 #添加了自己的实例属性total\_count

h1.\_\_class\_\_.total\_count = 3 #间接修改类变量

类的 \_\_slots\_\_ 属性

作用：

限定一个创建的实例只能有固定的属性（实例变量）

说明：

1.\_\_slots\_\_属性是一个列表，字表的值是字符串

2.含有\_\_slots\_\_属性的类所创建的实例没有\_\_dict\_\_字典

示例见：

code 内的 slots.py

类方法 @classmethod

类方法是操作类的方法，类方法属于类，不属于该类创建的对象

说明:

1.类方法需要使用@classmethod装饰器定义

2.类方法的第一个参数用来绑定类，约定写为cls

3.类和对象实例都可以调用类方法

4.类方法不能访问此类创建的对象的属性

示例见：

code 内的 classmethod.py

静态方法 @staticmathod

静态方法是定义在类的内部的函数，此函数作用域是类的内部

说明：

静态方法需要使用@staticmethod装饰器定义

静态方法与普通函数的定义相同，不需要传入self和cls

静态方法只能凭借该类和实例来调用

静态方法不能访问类变量和实例变量

示例：

class A:

@staticmethod

def myadd(a,b):

return a+b

print(A.myadd(1,2))

a = A()

print(a.myadd(3,4))

练习1：

修改之前的Human类，让此类能够记录此类生成的对象的个数，创建一个类方法

get\_human\_count来获取'人对象'的个数

见 code 内的 lianxi1.py

继承 inheritance 和 派生 derived

什么是继承/派生

继承是从已有类中派生出新类，新类具有原类的数据属性和行为，

并能扩展新的能力

派生就是从一个已有的类衍生出新类，在新的类上添加新的属性和行为

作用：

1.用继承派生机制，可以将一些共有功能加在基类中，实现代码的共享

2.在不改变超类的代码的基础上，改变原有的功能

名词：

基类(base class)/超类(super class)/父类(father class)

派生类(derived class)/子类(child class)

继承说明：

任何类都是直接或间接的继承自object类

object类是一切类的基类

类内的\_\_base\_\_属性

此属性用来记录类的基类

单继承

语法：

class 类名(超类名):

语句块

示例：

见 code 内的 inherit.py

class Human:

def say(self,what)：

print('说',what)

def walk(self,distance):

print('走了',distance,'公里')

练习2：

list类里只有append向末尾添加一个元素的方法，但没有向列表头部添加元素的方法，试想

能否为列表在不改变原有功能的基础上添加一个insert\_head(n)的方法，在列表的头部添加

元素

见 code 内的 lianxi2.py

覆盖：override

什么是覆盖

覆盖是指在有继承派生类关系的类中，子类中实现了与基类（超类）同名的方法，在子类

实例调用方法时，实际调用的是子类中的覆盖版本，这种现象叫做覆盖

实例见:

code 内的 override.py

问题：

当覆盖发生时，子类能否调用父类的方法？

super函数

super(type,obj)

返回绑定超类的实例(要求必须为type类型的实例)

super()

返回绑定超类的实例，等同于super(\_\_class\_\_,实例的第一个参数)，且必须在方法内调用

实例见：

code 内的 super.py

显示调用基类的初始化方法：

见code 内的 super\_init.py

说明：

当子类实现了\_\_init\_\_方法后，父类的\_\_init\_\_方法将被覆盖，即不会再主动调用父类的

\_\_init\_\_方法，会引起父类的属性得不到初始化，此时需要显示调用父类的初始化

练习3：

1.看懂学生管理系统划分模块的依据

2.添加保存文件和读取文件的代码

3.把字典改为对象来存储数据

练习4:

写一个Bicycle(类),有run(骑行)方法，调用时显示骑行里程km

class Bicycle:

def run(self,km):

print('自行车骑行了',km,'公里')

在写一个电动自行车类继承自Bicycle，添加电池电量valume属性，同时拥有两个方法：

1.fill\_charge(vol),用来充电,vol为电量（度）

2.run(km) 方法用于骑行，每骑行10km消耗电量1度，当电量消耗尽是调用Bicycle的

run方法骑行并显示骑行结果

主程序：

b = EBicycle(5) #创建一个电动自行车,默认电量5度

b.run(10) #骑行10km

b.run(100) #骑行100km

b.fill\_charge(6) #充电6度

b.run(70) #又骑行70km

## Day21

issubclass 函数

issubclass(cls,class\_or\_tuple)

判断一个类是否是继承自其他的类，如果此类cls是class或tuple中的一个派生子类，

返回True,否则返回false

示例：

class A:

pass

class B(A):

pass

class C(B):

pass

class D(B):

pass

issubclass(B,A) #True

issubclass(C,B) #True

issubclass(D,C) #False

issubclass(C,A) #True

issubclass(C,(int,str)) #False

issubclass(C,(int,A)) #True

封装 enclosure

1.封装是指隐藏类的实现细节，让使用者不关心这些细节

2.封装的目的是让使用者通过尽可能少的使用实例变量名（属性）操作对象

私有属性和方法

python类中以双下划线('\_\_')开头，不以双下划线结尾的标识符为私有成员

私有成员只能被方法调用，不能在子类或其他地方使用

私有成员有两种：

私有属性

私有方法

示例：

class A:

def \_\_init\_\_(self):

self.\_\_p1 = 100 #创建是有属性

def \_\_m1(self):

print('A类的私有方法被调用！')

def test(self):

print(self.\_\_p1) #可以访问

self.\_\_m1() #可以访问

a = A()

print(a.\_\_p1) #出错，不可以访问

a.\_\_m1() #出错，在类外部不能调用类的私有方法

a.test() #用方法来操作私有属性和方法

多态 polymorphic

字面意思：多种状态

多态是指在有继承/派生关系的类中，调用基类对象的方法，实际能调用子类

的覆盖方法的现象叫多态

说明：

多态调用方法与对象相关，不与类相关

python的全部对象只有'运行时状态(动态)',没有'C++/Java'里的'编译时状态(静态)'

示例见：

code 内的 poly.py

多继承的缺陷：

标识符（名字空间）冲突的问题，要谨慎使用多继承

多继承的 MRO(Methid Resolution Order)问题

MRO 方法搜索顺序问题

Python3 广度优先

Python2 深度优先

函数重写 overwrite

在自定义的类中，通过添加特定的方法，让自定义的类生成的对象（实例）能像内建对象

一样进行内建函数操作

对象转字符串函数重新

repr(obj) 返回一个能代表此对象的字符串，通常:

eval(repr(obj)) == obj

str(obj) 通过给定的对象返回一个字符串（这个字符串通常是给人阅读的）

换句话说：

repr(obj) 返回的字符串是给python用的

str(obj) 返回的字符串是给人看的

重写方法

repr(obj) 函数的重写方法 def \_\_repr\_\_(self)

str(obj) 函数的重写方法 def \_\_str\_\_(self)

当对象没有\_\_str\_\_方法时，则返回\_\_repr\_\_(self)的值

内建函数重写

obj.\_\_abs\_\_() 方法对应 abs(obj)

obj.\_\_len\_\_() 方法对应 len(obj)

obj.\_\_reversed\_\_() 方法对应 reversed(obj)

obj.\_\_round\_\_() 方法对应 round(obj)

示例见：

code 内的 myinteger.py

数值转换函数重写

obj.\_\_complex\_\_() 对应 complex(obj)

obj.\_\_int\_\_() 对应 int(obj)

obj.\_\_float\_\_() 对应 float(obj)

obj.\_\_bool\_\_() 对应 bool(obj)

bool测试函数的重写

格式：

def \_\_bool\_\_(self):

...

作用：

1.用于bool(obj)函数取值

2.用于if语句真值表达式中

3.用于while语句真值表达式中

说明：

布尔测试方式的查找顺序是\_\_bool\_\_方式，其次是\_\_len\_\_方法，如果

没有以上方法则返回True

示例见：

code 内的 bool.py

迭代器（高级）

可以通过next(obj)函数取值的对象，就是迭代器

迭代器协议：

迭代器协议是指对象能够使用next函数获取下一项数据，在没有下一项数据

时触发一个StopIteration异常来终止迭代的约定

迭代器协议的实现方法：

在类内需要定义\_\_next\_\_(self)方法来实现迭代器协议

语法形式：

class MyIterator:

def \_\_next\_\_(self):

迭代器协议

return 数据

可迭代对象

是指能用iter(obj)函数返回迭代器的对象(实例)

可迭代对象的内部要定义\_\_iter\_\_(self)方法来返回迭代器对象

练习1：

1.完善学生管理系统读取学生信息功能

2.看懂类的封装的用意及功能的划分（如：save\_to\_file(self)方法等）

3.写一个实现迭代器协议的类Primes,让此类可以生成从b开始到n个素数

class Primes:

def \_\_init\_\_(self,b,n):

...

...

for x in Primes(10,20):

print(x) #11,13,17,19

4.扩展学生信息管理程序，试想能否在不改变原Student类的基础上，为每个学生添加一个家庭地址信息（提示用继承/派生创建一个新的类）

5.添加11）修改家庭住址功能

## Day22

属性管理函数

1.getattr(obj, name[, default])

从一个对象得到对象的属性；getattr(x, 'y') 等同于x.y; 当属性不存在时,

如果给出default参数,则返回default,如果没有给出default 则产生一个AttributeError错误

2.hasattr(obj, name)

用给定的name返回对象obj是否有此属性,此种做法可以避免在getattr(obj, name)时引发错误

3.setattr(obj, name, value)

给对象obj的名为name的属性设置相应的值value, setattr(x, 'y', v) 等同于 x.y = v

4.delattr(obj, name)

删除对象obj中的name属性, delattr(x, 'y') 等同于 del x.y

异常（高级）

with 语句：

语法：

with 表达式1 [as 变量名1],表达式2 [as 变量名2],...

作用：

用于对资源的访问的场合，确保使用过程中不管是否发生异常，都会执行必要有'清理'操作

并释放资源

如：

文件打开后自动关闭，线程中锁的自动获取和释放

说明：

with语句与try-finally相似，并不会必变异常状态

as 子句用于绑定表达式创建的对象

示例见：

code 内的 with.py

环境管理器：

1.类内有\_\_enter\_\_和\_\_exit\_\_方法的类被称为环境管理器

2.能够用with进行管理的对象必须是环境管理器

3.\_\_enter\_\_将在进入with语句是被调用，并返回有as变量管理的对象

4.\_\_exit\_\_将在离开with语句时被调用，且可以用参数来判断离开with语句时是否出现

异常并做出相应的处理

示例见：

code 内的 filewrite.py

练习1:

实现文件的复制（建议使用二进制方式进行操作）

$ python3 mycp.py

请输入源文件：/etc/passwd

请输入目标文件: ./mypass.txt

提示：‘文件复制成功’或‘文件复制失败’

（建议使用with语句打开文件）

见 code 内的lianxi1.py

运算符重载：

什么是运算符重载

让自定义的类生成的对象（实例）能够使用运算符进行操作

作用：

让实例像数学表达式一样进行运算操作

让程序简洁易读

说明：

运算符重载方法的参数已经有固定的含义，不建议改变原有的含义

算术运算符：

方法名 运算符

\_\_add\_\_ 加法 +

\_\_sub\_\_ 减法 -

\_\_mul\_\_ 乘法 \*

\_\_truediv\_\_ 除法 /

\_\_floordiv\_\_ 地板除 //

\_\_mod\_\_ 求余 %

\_\_pow\_\_ 幂运算 \*\*

格式：

def \_\_xxx\_\_(self,other):

...

示例见：

code 内的mynumber.py

反向算术运算符：

方法名 运算符

\_\_radd\_\_ 加法 +

\_\_rsub\_\_ 减法 -

\_\_rmul\_\_ 乘法 \*

\_\_rtruediv\_\_ 除法 /

\_\_rfloordiv\_\_ 地板除 //

\_\_rmod\_\_ 求余 %

\_\_rpow\_\_ 幂运算 \*\*

格式：

def \_\_xxx\_\_(self,other):

...

练习2：

实现两个自定义表的相加

class Mylist:

def \_\_init\_\_(self,iterable):

self.data = [x for x in iterable]

...#类内以下的部分自己实现

l1 = Mylist([1,2,3])

l2 = Mylist(range(4,7))

l3 = l1 + l2

print('l3=',l3)

l4 = l1 \* 2

print(l4)

l5 = 2 \*l2

复合赋值运算符重载：

方法名 运算符

\_\_iadd\_\_ 加法 +=

\_\_isub\_\_ 减法 -=

\_\_imul\_\_ 乘法 \*

\_\_itruediv\_\_ 除法 /

\_\_ifloordiv\_\_ 地板除 //

\_\_imod\_\_ 求余 %

\_\_ipow\_\_ 幂运算 \*\*

示例见：

code 内的 mylist.py

问题：

#算法1

a = [100]

def test(x):

x = x + x #调用了\_\_add\_\_方法（重新生成一个新的对象绑定x）

print(x) #[100,100]

test(a)

print(a) #[100]

#算法2

a = [100]

def test(x):

x += x #调用了\_\_iadd\_\_方法（在原来的变量上原地操作）

print(x) #[100,100]

test(a)

print(a) #[100,100]

比较运算符的重载

\_\_lt\_\_ 小于 <

\_\_le\_\_ 小于等于 <=

\_\_gt\_\_ 大于 >

\_\_ge\_\_ 大于等于 >=

\_\_eq\_\_ 等于 ==

\_\_ne\_\_ 不等 !=

返回True 或 False

位运算符重载

\_\_invert\_\_ ~取反（一元运算符）

\_\_and\_\_ &位与（交集）

\_\_or\_\_ |位或（并集）

\_\_xor\_\_ ^位异或（对称补集）

\_\_lshift\_\_ <<左移

\_\_rshift\_\_ >>右移

反向位运算符重载

\_\_rinvert\_\_ ~取反（一元运算符）

\_\_rand\_\_ &位与（交集）

\_\_ror\_\_ |位或（并集）

\_\_rxor\_\_ ^位异或（对称补集）

\_\_rlshift\_\_ <<左移

\_\_rrshift\_\_ >>右移

为运算符复合赋值重载

\_\_iinvert\_\_ ~=取反（一元运算符）

\_\_iand\_\_ &=位与（交集）

\_\_ior\_\_ |=位或（并集）

\_\_ixor\_\_ ^=位异或（对称补集）

\_\_ilshift\_\_ <<=左移

\_\_irshift\_\_ >>=右移

一元运算符的重载：

\_\_neg\_\_ -负号

\_\_pos\_\_ +正号

\_\_invert\_\_ ~按位取反

in/not in 运算符的重载

格式：

def \_\_contains\_\_(self,e): #e代表元素

...

说明：

not in 相当于 in取反，所以只需重载in

示例见：

code 内的 in\_not\_in.py

索引index和切片slice运算符重载

重载方法：

\_\_getitem\_\_(self,i) 用于索引/切片取值

\_\_setitem\_\_(self,i) 用于索引/切片赋值

\_\_delitem\_\_(self,i) 用于del语句删除索引操作

作用：

让自定义类的对象能够支持索引和切片操作

示例见：

code 内的 index.py

练习3：

实现有序集合类OrderSet(),能实现两个集合的交集&,并集|,补集-,对称补集^,==,!=等操作

要求：

集合内部用list存储

class OrderSet:

def \_\_init\_\_(self,iterable):

self.data = [x for x in iterable]

...

测试用例：

s1 = OrderSet([1,2,3,4])

s2 = OrderSet([3,4,5])

print(s1 & s2) #OrderSet([3,4])

print(s1 | s2) #OrderSet([1,2,3,4,5])

print(s1 ^ s2) #OrderSet([1,2,5])

if OrderSet([1,2,3]) != OrderSet([1,2,3,4]):

print('不相同')

# MySQL部分

## Day23

王伟超

wangweichao@tedu.cn

一.什么是数据库

数据库是一个存储数据的仓库

二.都有哪些公司在用数据库？

金融机构、游戏网站、购物网站、论坛网站、...

三.提供数据库服务的软件

1.软件分类

MySQL、SQL\_Server、Oracle、Mariadb、DB2、MongoDB...

2.在生产环境中，如何选择使用哪个数据库？

1).是否开源

开源软件：MySQL、Mariadb、MongoDB

商业软件：Oracle、DB2、SQL\_Server

2).是否跨平台

不跨平台：SQL\_Server

跨平台：MySQL、Oracle、Mariadb、DB2、MongoDB

3).公司的类型

商业软件：政府部门、金融机构

开源软件：游戏网站、购物网站、论坛网站...

一.MySQL的特点

1.关系型数据库

1、关系数据库的特点

1）数据是以行和列的形式去存储的

2）这一系列的行和列称为表

3）表中的每一行叫一条记录

4）表中的每一列叫一个字段

5）表和表之间的逻辑关联叫关系

2、示例

1）关系型数据库存储

表1、学生信息表

姓名 年龄 班级

牛郎 25 AID1803

织女 23 AID1801

表2、班级信息表

班级 班主任

AID1803 卢大大

AID1801 孙大大

2）非关系型数据库存储

{"姓名":"牛郎","年龄":25,"班级":"AID1803","班主任":"卢大大"}

{"姓名":"织女","年龄":23,"班级":"AID1803"}

2.跨平台

可以在Unix, Linux, Windows上运行数据库服务

3.支持多种编程语言

Python, java, php...

二.MySQL的安装

1.Ubuntu安装MySQL服务

1、安装服务端

sudo apt-get install mysql-server

2、安装客户端

sudo apt-get install mysql-client

2.Windows安装MySQL服务

1、下载MySQL安装包（windows）

mysql-install\*\*\*5.7.\*\*\*.msi

2、双击、按照教程安装即可

三.启动和连接MySQL服务

1.服务端启动

1、查看MySQL服务状态

sudo /etc/init.d/mysql status

2、停止、启动、重启MySQL服务

sudo /etc/init.d/mysql stop(start,restart(reload))

reload 重载配置文件，保证线上业务不断

2.客户端连接

1、命令格式

mysql -h主机地址 -u用户名 [-P3306] -p密码

mysql -hlocalhost -uroot -p123456

-P3306 表示服务端口3306,默认值（可不写）

2、本地连接可以省略 -h选项

mysql -u用户名 -p123456

四.基本SQL命令

1.SQL命令的使用规则

1、每条SQL命令必须以“;”结尾

2、SQL命令不区分字母大小写

3、使用 \c 来终止命令的执行

2.库的管理

1、库的基本操作

1)查看已有的库

show databases;

2)创建库(指定默认字符集)

create database 库名[default charset=utf8];

3)查看创建库的语句

show create database 库名;

4)查看当前所在库

select database();

5)切换库

use 库名;

6)查看库中已有的表

show tables;

7)删除库

drop database 库名;

2、库名的命名规则

1)可以使用数字、字母、\_，但不能使用纯数字

2)库名区分字母大小写

3)库名具有唯一性

4)不能使用特殊字符和MySQL关键字

3.表的管理

1、表的基本操作

1)创建表

create table 表名(

字段名 数据类型,

字段名 数据类型,

...

)[default charset=utf8];

2)查看创建表的语句(字符集和存储引擎)

show create table 表名;

3)查看表结构

desc 表名;

4)删除表

drop table 表名;

2、表的命名规则

同库的命名规则一致

4.注意

1、所有的数据都是以文件的形式存放在数据库目录下

2、数据库目录:/var/lib/mysql

五.更改库、表的默认字符集

1.方法

通过更改MySQL服务的配置文件来实现

2.步骤

1、获取root权限

sudo -i

2、切换到配置文件所在路径

cd /etc/mysql/mysql.conf.d

3、备份

cp -p mysqld.cnf mysqld.cnf.bak

-p 选项表示将权限也一同复制

4、用vi打开mysqld.cnf,在[mysqld]处

character\_set\_server = utf8 保存退出

5、重启mysql服务

六.表记录管理

1.在表中插入记录

1、insert into 表名 values(值1),(值2),....,(值N);

2、insert into 表名(字段名列表) values(值1)，（值2），...,(值N);

2.查询表记录

1、select \* from 表名；

2、select 字段名1，字段名2，...，字段名N from 表名;

七.客户端把数据存储到数据库服务器上的过程

1.连接到数据库服务器： mysql -uroot -p；

2.选择一个库：use 库名；

3.创建表或者修改表：create ...

4.断开与数据库服务器的连接：exit | quit | \q

八.名词介绍

1.DB(database)

DB就是数据库，存储数据的仓库

2.DBMS(database management system)

数据库管理系统

管理数据库的软件:MySQL、Oracle、MongoDB、...

3.DBS(database system)

数据库系统

DBS = DB(存储) + DBMS(数据库软件) + 数据库应用(财务管理系统，人事管理系统...) + 用户

九.数据类型

1.数字类型（有符号signed和无符号unsigned）

1、整型

1)int(大整型，4个字节)

取值范围：0--2\*\*32-1(无符号unsigned)、-2\*\*31--2\*\*31-1(signed默认)

2)tinyint(微小整型，1个字节)

取值范围：-128--127(signed默认)、0--255(unsigned)

3)smallint(小整型,2个字节)

取值范围：0--2\*\*16-1(unsigined)、-2\*\*15--2\*\*15-1(signed)

4)bigint(极大整型，8个字节)

取值范围：0--2\*\*64-1(unsigined)、-2\*\*63--2\*\*63-1(signed)

2、浮点型

1)float(4个字节，最多显示7个有效位)

用法：

字段名 float(m,n) m:表示总位数 n:表示小数位位数

salary float(5,2) 取值范围： -999.99--999.99

注意：

1.浮点型插入整数时会自动补全小数位位数

2.小数位如果多于指定的位数，会对指定位的下一位进行四舍五入

float(5,2) # 23.128 -> 23.13

2)double(8个字节，最多显示15个有效位)

用法：

字段名 double(m,n)

3)decimal(m+2个字节，最多显示28个有效位)

用法：

字段名 decimal(28,5)

2.字符类型

1、char(定长)

取值范围：1--255

不给定宽度默认为：1

2、varchar(变长)

取值范围:1--65535

varchar没有默认宽度，必须给定一个宽度值

3、char 和 varchar 的特点

1)char 浪费存储空间，但性能高

2)varchar 节省存储空间，但是性能低

3.枚举类型

定义：字段的值只能在列举的范围内选择

1、enum(...)单选（最多有65535个不同的值）

字段名 enum(值1，值2，... 值N)

2、set(...)多选（最多有64个不同的值）

字段名 set(值1，值2，...值N)

插入记录时 "值1,值2,值3,...,值N"

4.日期时间类型

1、year:年 YYYY

2、date:日期 YYYYMMDD

3、time:时间 HHMMSS

4、datetime:日期时间 YYYYMMDDHHMMSS

5、timestamp:日期时间 YYYYMMDDHHMMSS

6、注意:

1)插入记录时datetime字段不给值，默认返回NULL

2)插入记录时timestamp字段不给值，默认返回系统当前时间

十.表字段的操作

1.语法：alter table 表名 执行动作；

1、添加字段

alter table 表名 add 字段名 数据类型 [first][after 字段名]；

first 表示在最前面添加新字段名

after 表示在某个字段名后添加新字段名

默认为添加在最后

2、删除字段

alter table 表名 drop 字段名；

3、修改字段数据类型

alter table 表名 modify 字段名 新数据类型；

注：在修改数据类型时会受到表中原有数据的限制

4、修改字段名

alter table 表名 change 旧名 新名 数据类型；

5、修改表名

alter table 表名 rename 新表名；

1.面试题

1.填空题

1.MySQL中的数据类型有：

1)数字类型、2）字符类型、3）枚举类型、4）日期时间类型

2.关系型数据库的核心内容是:\_\_关系\_\_即\_\_二维表\_\_

2.简答题

1.简述客户端把数据存储到数据库服务器上的过程

2.char和varchar的区别？，各自的特点？

## Day24

1.字符类型的宽度和数值类型的宽度的区别

1.数值类型的宽度仅仅为显示宽度，只用于select查询显示，和占用的

存储空间大小无关，可用zerofill(零填充)查看效果

2.字符类型的宽度超过则无法存储

2.where条件子句（配合查询、修改、删除操作）

1.语法格式

select \* from 表名 where 条件;

3.表记录管理

1.删除表记录

1.delete from 表名 where 条件;

2.delete from 表名;(删除表中的所有记录)

2.更新表记录

update 表名 set 字段名1=值1，字段名2=值2，... where 条件

注：如果不加'where 条件'子句会将更新表中所有记录

4.运算符操作

1.数值比较和字符比较

数值比较运算符：= != > < >= <=

字符比较运算符：= !=

2.逻辑比较

运算符：

and(两个或多个条件同时满足)

or(两个或多个条件有一个满足)

3.范围内比较

运算符：between and、in、not in

语法：

between 值1 and 值2

in(值1，值2，...,值N)

not in(值1，值2，...,值N)

4.匹配空、非空

1.空：is null

2.非空：is not null

注：

null:空值，必须用is 或者is not 去匹配

'':空字符串，用 = 或者 != 去匹配

5.模糊比较

where 字段名 like 表达式；

表达式：

\_:匹配单个字符

%:匹配0到多个字符（null不会被统计）

6.正则匹配查询（regexp）

语法：

where 字段名 regexp '正则表达式';

正则表达式符号:

^ : 以...开头

例：^[0-9] #匹配以数字开头

$ : 以...结尾

例：[0-9]$ #匹配以数字结尾

. : 匹配任意一个字符

[]: 包含...内容

[0-9]:匹配带数字的

[a-z]:匹配带小写字母的

[A-Z]:匹配带小写字母的

\*: 星号前面的字符出现0次或者多次

5.SQL查询

1.总结（执行顺序）

1)where ...

2)group by ...

3)select ... 聚合函数 from 表名

4)having ...

5)order by ...

6)limit ...

2.order by

作用：

给查询的结果进行排序

语法格式：

order by 字段名 排序方式;

asc(默认)：升序 desc：降序

3.limit (永远放在SQL语句的最后写)

作用：

限制显示查询记录的个数

用法：

limit n : 显示n条记录

limit m,n : 从第m+1条开始，显示n条记录

注：m的值是从0开始计数的

4.聚合函数

分类：

avg(字段名)：求字段的平均值

sum(字段名)：求和

max(字段名)：求最大值

min(字段名)：求最小值

count(字段名)：统计该字段记录的个数

示例：

1.攻击力最强值是

select max(gongji) as best from sanguo;

2.统计一下id 和 name两个字段分别有多少条记录

select count(id),count(name) from sanguo;

注：空值null不会被统计，空字符串''会被统计

3.统计蜀国英雄攻击值大于200的英雄数量

select count(\*) from sanguo where country='蜀国' and gongji>200;

5.group by（先分组，在聚合）

作用：给查询的结果进行分组

示例：

1.统计sanguo表中一共有几个国家

select country from sanguo group by country;

2.计算所有国家的平均攻击力

select country,avg(gongji) from sanguo group by country;

3.查找所有国家中英雄数量最多的前2名的国家名称及英雄数量

select country,count(\*) as numbers from sanguo

group by country

order by numbers desc

limit 2;

注意：

1.group by之后的字段名必须要为select之后的字段名

2.如果select之后的字段没有在group by语句之后，则必须要对 该字段进行聚合处理（聚合函数）

6.having

作用：对查询的结果进行进一步筛选

示例：

1.找出平均攻击力大于105的国家的前2名，显示国家名称和平均攻击 力

select country,avg(gongji) as pjgj from sanguo

group by country

having pjgj>105

order by pjgj desc

limit 2

注意：

1.having 语句通常与group by语句联合使用，用来过滤由group by

语句返回的记录集

2.having语句的存在弥补了where条件子句不能与聚合函数联合使用

的不足，where操作的是表中实际存在的字段，having操作的是聚合

函数生成的显示列

7.distinct

作用：

不显示字段的重复值

示例：

1.sanguo表中一共有多少哪些个国家？

select distinct country from sanguo;

2.计算蜀国一共有多少个英雄？

select count(distinct name) from sanguo

where

country = '蜀国';

注意：

1.distinct处理的是distincthe和from之间的所有字段，所有字段值

必须完全相同才可以去重

2.distinct不能对任何字段做聚合处理

8.查询表记录时可以做数学运算

1.运算符

+ - \* % /

示例：

查询时显示所有英雄的攻击力\*10

select name,gongji\*10,country as 攻击力 from sanguo;

6.约束

作用：

为了保证数据的完整性、一致性、有效性，可以限制无效的数据插入到数

据表中

约束分类：

1.默认约束（default）

作用：

在插入记录时，如果不给该字段赋值，则使用默认值

格式：

字段名 数据类型 default 默认值,

2.非空约束

作用：

不允许该字段的值有空值null记录

格式：

字段名 数据类型 not null,

7.索引

定义：

对数据库中表的一列或者多列的值进行排序的一种结构(mysql中索引

用Btree方式)

索引的优点：

可以加快数据的检索速度

索引的缺点：

1.当对表中的数据进行增加、修改、删除的时候，索引需要动态维护，降

低了数据的维护速度

2.索引需要占用物理存储空间(数据库目录/var/lib/mysql)

索引的示例：

1.运行insert\_.py 文件，插入100万条记录

2.开启性能分析

show variables like '%pro%';

set profiling = 1;

3.执行查询语句

select name from t1 where name='lucy9999';

4.查看性能分析结果

show profiles;

5.在name字段创建索引

create index name on t1(name);

6.执行查询语句

select name from t1 where name='lucy9999';

7.查看性能分析结果

show profiles;

8.关闭性能分析

set profiling = 0;

## Day25

1.索引分类

1.普通索引(index)

使用规则：

1.一个表中可以有多个index字段

2.字段的值可以有重复，也可以为NULL值

3.经常把做查询条件的字段设置为index字段

4.index字段的key标识为:MUL

创建普通索引：

1.在创建表时创建index

create table 表名(

id int,

name varchar(15),

index(id),

index(name)

);

2.在已存在的表中添加索引字段

create index 索引名 on 表名(字段名);

注：索引名一般和字段名一样

查看普通索引：

1.desc 表名：-->查看KEY标志为NULL

2.show index from 表名\G；

删除普通索引：

drop index 索引名 on 表名

注：删除普通索引只能一个一个删除

2.唯一索引(unique)

使用规则：

1.一个表中可以有多个unique字段

2.unique字段的值不允许重复，可以为空值NULL

3.unique的KEY标志为 UNI

创建唯一索引（基本等同index创建）

1.创建表时创建unique

1.unique(字段名)

2.字段名 数据类型 unique

2.在已有表中添加unique

create unique index 索引名 on 表名(字段名);

3.查看、删除唯一索引

desc 表名;

show index from 表名;

drop index 索引名 on 表名;

3.主键索引(primary key) && 自增长属性(auto\_increment)

使用规则：

1.一个表中只能有一个主键字段

2.对应字段的值不允许重复且不能为空值NULL

3.主键字段的KEY标志为 PRI

4.把表中能够唯一标识一条记录的字段设置为主键，通常把表中记录

编号的字段设置为主键

创建主键（PRI）:

1.创建表时创建PRI

方法一:字段名字段名 数据类型 primary key auto\_increment，

方法二：id int auto\_increment,

... ...,

primary key(id)

注：primary key(id,name) -->复合主键

create table 表名(

id int primary key auto\_increment,

name varchar(15)

)auto\_increment=1000;-->设置自增长属性的起始值

2.在已有表中添加主键

alter table 表名 add primary key(字段名);

删除主键：

1.先删除自增长属性(修改字段属性)

alter table 表名 modify id int;

2.再删除主键

alter table 表名 drop primary key;

4.外键索引

外键：让当前表字段的值在另一个表的范围内选择

语法格式：

foreign key(参考字段名)

references 被参考表名(被参考字段名)

on delete cascade(级联动作)

on update cascade(级联动作)

级联动作：

1.cascade:数据级联更新

当主表删除记录或者更新被参考字段的值时，从表会级联更新

2.restrict 默认

当删除主表记录时，如果从表中有相关联记录则不允许主表删除

更新同理

3.set null

当主表删除记录时，从表中相关联记录的参考字段值自动设置为

NULL

更新同理

4.no cation

on delete no action

on update no cation

同restrict,都是立即检查外键限制

案例：

表1、缴费信息表(财务)

学号 姓名 班级 缴费金额

1 唐伯虎 AID01 28000

2 秋香 AID01 20000

表2、学生信息表(班主任)

学号 姓名 缴费金额

1 唐伯虎 28000

2 秋香 20000

创建缴费信息表(主表)

create table jftab(

id int primary key,

name char(15),

class char(5),

money int

)default charset=utf8;

insert into jftab values

(1,'唐伯虎','AID01',28000),

(2,'秋香','AID01',20000),

(3,'祝枝山','AID01',22000);

创建学生信息表(从表)

create table bjtab(

stu\_id int,

name char(15),

money int,

foreign key(stu\_id) references jftab(id)

on delete cascade

on update cascade

);

删除外键：

alter table 表名 drop foreign key 外键名;

注：外键名的查看

show create table 表名；

在已有表中添加外键：

alter table 表名 add foreign key(参考字段名)

references 被参考表名(被参考字段名)

on delete 级联动作

on update 级联动作;

外键使用规则

1.两张表被参考字段和参考字段数据类型要一致

2.被参考字段必须是key的一种，通常是primary key

2.数据的导入

作用：

把文件系统的内容导入到数据库中

语法：

load data infile '文件名'

into table 表名

fields terminated by '分隔符'

lines terminated by '\n'

示例：

把/etc/passwd文件中的内容导入数据库中(indexdb库中的userinfo表)

tarena : x : 1000 : 1000 : tarena,,, : /home/tarena : /bin/bash

用户描述 登录权限

操作步骤

1.创建表

create table userinfo(

username char(20),

password char(1),

uid int,

gid int,

comment varchar(50),

homedir varchar(50),

shell varchar(50)

);

2.将要导入的文件拷贝到数据库的默认搜索路径中

1.查看数据库的默认搜索路径

show variables like 'secure\_file\_priv';

/var/lib/mysql-files

2.将要导入数据的文件拷贝到默认搜索路径里

sudo cp /etc/passwd /var/lib/mysql-files/passwd

3.执行数据库导入语句

load data infile '/var/lib/mysql-files/passwd'

into table userinfo

fields terminated by ':'

lines terminated by '\n';

3.数据的导出

作用：

将数据库表中的记录保存到系统文件里

语法格式：

select ... from 表名

into outfile '文件名'

fields terminated by '分隔符'

lines terminated by '分隔符'(一般为'\n')

示例：

把userinfo表中的username password 和uid 导出到文件user.txt

select username,password,uid from userinfo

into outfile '/var/lib/mysql-files/user.txt'

fields terminated by ','

lines terminated by '\n';

4.表的复制

语法格式：

create table 表名 select 查询命令;

示例：

复制userinfo 表中的全部记录,userinfo2

create table userinfo2 select \* from userinfo;

复制表的结构

create table 表名 select ... from 源表名 where false;

注意：

复制表的时候不会把原有表的key属性复制过来

5.嵌套查询(子查询)

定义：

把内层的查询结果作为外层的查询条件

示例：

把uid的值小于uid平均值的用户名和uid号显示出来

->select username,uid from userinfo

->where

->uid < select avg(uid) from userinfo;

6.连接查询

内连接

定义：从表中删除与其他被连接的表中没有匹配到的行

语法格式：

select 字段名列表 from 表1

inner join 表2 on 条件

inner join 表3 on 条件

...;

示例：

1.显示省市的详细信息

-> select 省.s\_name,市.c\_name from 省

-> inner join 市

-> on 省.s\_id = 市.cfather\_id;

2.显示省市县的详细信息

-> select 省.s\_name,市.c\_name from 省

-> inner join 市

-> on 省.s\_id = 市.cfather\_id

-> inner join 县

-> on 市.c\_id = 县.xfather\_id;

外连接

1.左连接

定义：以左表为主显示查询结果

语法格式：

select 字段名列表 from 表1

left join 表2 on 条件

left join 表3 on 条件

...;

示例：

1.以省表为主显示省市县详细信息

-> select 省.s\_name,市.c\_name from 省

-> left join 市

-> on 省.s\_id = 市.cfather\_id

-> left join 县

-> on 市.c\_id = 县.xfather\_id;

2.右连接

定义：以右表为主显示查询结果

语法格式：

select 字段名列表 from 表1

right join 表2 on 条件

right join 表3 on 条件

...;

7.多表查询

select 字段名列表 from 表名列表; #笛卡尔积

select 字段名列表 from 表名列表 where 条件 #等同内连接

## Day26

1.MySQL用户账户管理

1.开启MySQL远程连接

1.sudo -i

2.cd /etc/mysql/mysql.conf.d/

3.vi mysqld.cnf

#bind-address = 127.0.0.1

或者

bind-address = 0.0.0.0

4./etc/init.d/mysql restart

2.添加授权用户

1.使用root用户连接到服务器

mysql -uroot -p

2.添加新的授权用户

create user '用户名'@'IP地址' identified by '密码';

create user 'tiger'@'%' identified by '123456';

IP地址的表示方式：

1.% 表示用户可以从任何地址连接到服务器

2.localhost 用户只能从本地连接

3.指定一个IP 表示用户只能从此ip连接到服务器

3.给用户授权

grant 权限列表 on 库.表 to '用户名'@'IP地址' with grant option;

例：grant all privileges on \*.\* to 'tiger'@'%' with grant option;

权限列表：

select,update,delete,insert,alter,drop,create,...

all privileges 表示所有权限

库.表:

\*.\* 表示所有库的所有表

4.查看授权用户

select user,host from mysql.user;

5.删除授权用户

drop user '用户名'@'IP地址'；

2.数据备份(mysqldump,在Linux终端中操作)

1.命令格式：

mysqldump -uroot -p 源库名 > 路径/xxx.sql

例：mysqldump -uroot -p studb > /home/tarena/studb.sql

2.源库名的表示方式：

--all-databases 备份所有库

库名 备份单个库

-B 库1 库2... 备份多个库

库名 表1 表2... 备份指定库的指定表

3.练习：

1.备份所有库 all-mysql.sql 放到用户主目录下的mydata目录中

mysqldump -uroot -p --all-databases > mydata/all-mysql.sql

2.备份indexdb 库中的 省，市，县三张表scx.sql

mysqldump -uroot -p indexdb 省 市 县 > mydata/scx.sql

3.备份MOSHOU 和 indexdb 库，命名为MSind.sql

mysqldump -uroot -p -B MOSHOU indexdb > mydata/MSind.sql

3.数据恢复(在Linux终端中操作)

1.命令格式：

mysql -u用户名 -p 目标库名 < 路径/xxx.sql

4.备份种类

完全备份：mysqldump

增量备份：binlog日志、xbackup工具

5.从所有库的备份文件中恢复某一个库(--one-database)

mysql -u用户名 -p --one-database 目标库名 < all-mysql.sql

注意：

1.恢复是必须先创建空库

2.恢复库时库中新增的表不会删除

6.事务和事务回滚

定义：

一件事从开始发生到结束的整个过程

作用：

确保数据的一致性

事务和事务回滚的应用：

1.mysql中默认sql语句会自动commit到数据库

show variables like 'autocommit';

2.事务应用

1.开启事务

mysql> start transaction;

#此时autocommit被禁用，SQL命令不会对数据库中数据做修改

2.终止事务

mysql> commit;

或者

mysql> rollback

3.注意：

1.事务和事务回滚只针对于表记录的增，删，改，对创建库、创

建表的操作无效

7.存储引擎

定义：

是用来处理 表（table） 的处理器

存储引擎基本操作：

1.查看已有表的存储引擎

show create table 表名； #engine...

2.创建表时指定存储引擎

create table 表名(...)engine=myisam;

3.查看所有的存储引擎

mysql> show engines;

工作中使用的存储引擎：

innodb:

特点：

1.共享表空间

表名.frm 表结构

表名.ibd 表记录和索引信息

2.支持行级锁

myisam：

特点：

1.独享表空间

表名.frm 表结构

表名.myd 表记录

表名.myi 索引信息

2.支持表级锁

锁：

1.加锁的目的

解决客户端并发访问的冲突问题

2.锁的类型

读锁(select)：又叫共享锁

当发起select时，会自动加上锁

加读锁之后不能更改表中内容，但可以进行查询

写锁(insert,update,delete)：

又叫互斥锁或者排他锁

3.锁粒度

表级锁

行级锁

###操作完成后会自动释放锁

如何决定使用哪种存储引擎：

1.执行查询操作多的表使用myisam存储引擎(使用innnodb浪费资源)

2.执行写操作比较多的表使用innodb存储引擎

更改表的默认存储引擎：

修改配置文件/etc/mysql/mysql.conf.d/mysqld.cnf

在[mysqld]中添加以下语句

default-storage-engine = myisam(存储引擎名)

memory存储引擎：

memory:表记录存储在内存中

表名.frm 表结构

注：

服务重启之后表结构在，表记录都消失

8.MySQL调优

1.选择合适的存储引擎

经常用来读的表使用myisam存储引擎，其余的表都用innodb存储引擎

2.SQL语句调优(尽量避免全表扫描)

1.在select where order by 常涉及到的字段上建立索引

2.where子句中不使用 !=,否则将放弃使用索引进行全表扫描

3.尽量避免用NULL值普判断，否则会全表扫描

4.尽量避免用 or 来连接条件，否则会全表扫描

示例：

优化前：

select id from t1 where id=10 or id=20;

优化后：

select id from t1 where id=10

union all

select id from t1 where id=20;

5.模糊查询尽量避免使用前置 % ,导致全表扫描

6.尽量避免 in 和 not in,导致全表扫描

示例：

优化前：

select id from t1 where id in(1,2,3);

优化后：

select id from t1 where id between 1 and 3;

7.尽量避免使用select \* ...,要用具体的字段列表代替 \*,不要返回用不

到的字段

9.Python数据库编程

1.python数据库接口(python DB-API)

为开发人员提供的数据库应用编程接口

2.支持的数据库服务软件

mysql、Oracle、SQL\_Server、mongodb..

3.python 提供的操作mysql模块

python3:pymysql

python2:MySQLdb

4.pymysql模块使用流程

1.建立数据库连接

2.创建游标对象

3.使用游标对象的方法操作数据库

4.提交commit

5.关闭游标对象

6.关闭数据库连接

5.建立数据库连接

对象名 = pymysql.connect('主机地址','用户名','密码'[,'库名']

[,charset='utf8'])

2.connect连接对象支持的方法

1.cursor() 创建一个游标对象db.cursor()

2.commit() 提交到数据库执行(表记录增删改)

3.rollback() 回滚

4.close() 关闭数据库的连接

3.游标对象支持的方法

1.execute('SQL命令') 执行SQL命令

2.fetchone() 取得结果集的第一条记录

3.fetchmany(n) 取得结果集的 n 条记录

4.fetchall() 取得结果集的所有记录

5.close() 关闭游标对象

10.Mysql workbench(图形化界面管理工具)

11.ER模型&ER图

1.定义：

ER模型即 实体 - 关系 模型

ER图即 实体关系图

2.三个概念

1.实体

2.属性

3.关系

定义：

实体之间的关系

分类

一对一关系(1:1)

一对多关系(1:n)

多对多关系(m:n)

4.ER图的绘制

1.矩形框代表实体，菱形框代表关系，椭形代表属性

# 网络编程部分

## Day27

吕泽

lvze@tedu.com.cn

网络 ---》进程线程 ---》web服务器 ---》mongo ---》regex ---》项目

互联网 --》网络传输数据

ISO（国际标准化组织）--》网络体系结构标准(OSI模型)

OSI模型（七层模型）

1.应用层：提供用户服务，具体内容有特定程序规定

2.表示层：提供数据的压缩解压和加密等

3.会话层：建立程序级的连接，决定使用什么样的传输服务

4.传输层：提供合适的通信服务，流量控制

5.网络层：路由选择，网络互联，分组传输

6.链路层：提供链路交换，具体消息发送

7.物理层：物理硬件，接口

OSI模型优点：

将功能分开，降低网络传输中的耦合度，每一部分完成自己的功能，

开发更加清晰

高内聚：使模块功能单一，不混杂

低耦合：模块之间尽量减少相互的关联和影响

四层、五层模型（大多实际使用）

四层：

应用层：应用层，表示层，会话层

传输层：传输层

网络层：网络层

物理链路层：链路层和物理层

五层（TCP/IP模型）：

应用层：应用层，表示层，会话层

传输层：传输层

网络层：网络层

链路层：链路层

物理层：物理层

要求：能够顺序说出七层模型，知道如何对应四层五层模型，简单理解每一层的功能

，能理解数据传输过程

网络协议：

网络通信中，各方必须遵守的规定，包括建立什么连接，如何识别，消息结构等

应用层：TFTP，HTTP，SMTP，DNS...

传输层：TCP, UDP...

网络层：IP...

网络概念：

主机：计算机的名称localhost

获取主机名：

import socket

socket.gethostname() #'tedu'

获取ip地址：

socket.gethostbyname('tedu') #'127.0.1.1'

IP

IPv4:192.168.1.2

点分式十进制：三个点分为四个部分，每部分的值范围0-255

IPv6：ip地址更多

127.0.0.1：代表本地测试ip

0.0.0.0: 局域网可用ip

192.168.1.0：表示网段

192.168.1.1：表示网关(不是绝对)

192.168.1.255：广播地址

In [2]:socket.gethostbyaddr('www.baidu.com')

Out[2]:('127.0.0.1',[],['119.75.216.20'])

主机名 别名 IP地址

将ip地址转化为二进制

In [4]: socket.inet\_aton('127.0.0.1')

Out[4]: b'\x7f\x00\x00\x01'

将二进制转化为点分十进制

In [6]: socket.inet\_ntoa(b'\x7f\x00\x00\x01')

Out[6]: '127.0.0.1'

注：inet\_pton、inet\_ntop功能同上，只是第一个参数用

socket.AF\_IENT表示IPv4

socket.AF\_IENT6表示IPv6

端口号(1-65535)

端口号是地址的组成部分，在一个系统中区分应用层程序

1--255:代表众所周知的端口 http-->80

256--1023:代表系统端口

获取一个应用的端口

In [7]: socket.getservbyname('mysql')

Out[7]: 3306

In [8]: socket.getservbyname('http')

Out[8]: 80

子网掩码：和ip配合过滤网段

dns：域名解析

字节序

大端序：高字节存在低地址位

小端序：低字节存在低地址位

网络字节序：将字节序发送方式统一（采用大端序方式）

传输层服务

tcp协议：

面向连接的通信服务，可靠的数据传输服务

三次握手：

1.客户端向服务器发起连接请求(问是否可以连接)

2.服务器受到请求后进行确认(允许连接)返回报文

3.客户端收到许可，建立连接

四次挥手：

1.主动方发送报文告知被动方要断开连接

2.被动方返回报文，告知收到请求，准备断开

3.被动方发送报文给主动方告知准备就绪可以断开

4.主动方发送报文确定断开

可靠传输：无失序、无差错、无丢失、无重复

应用情况：

适用于传输内容较大，网络情况好，需要提供准确数据的情况

如：聊天信息，文件传输，邮件等

udp协议：

面向无连接的服务，不可靠的数据传输服务

发送时完全由发送端自主进行，不考虑接收端

适用情况：

网络较差，对传输准确性要求低，广播组播

如：视频会议，广播数据

套接字编程

套接字(socket)：进行网络通信的一种手段

1.流式套接字(SOCK\_STREAM)

传输层基于tcp协议进行通信

2.数据报套接字(sock\_dgram)

传输层基于udp协议进行通信

3.原始套接字(sock\_raw)

访问底层协议的套接字

tcp服务端：

from socket import \*

1.创建套接字

sockfd = socket(socket\_family=AF\_INET,socket\_type=SOCK\_STREAM,proto=0)

功能：创建套接字

参数：

socket\_family:选择地址族种类

socket\_type:套接字类型(SOCK\_STREAM,SOCK\_DGRAM)

proto:子协议类型通常默认为0

返回值：返回创建的套接字

2.绑定ip和端口号

sockfd.bind()

功能：绑定ip和端口

参数：是一个元组，第一项是字符串形式的ip,第二项是端口

3.让套接字具有监听功能

sockfd.listen(n)

功能：使套接字变为监听套接字，同时创建监听队列

参数：n 监听队列大小

4.等待客户端连接

sockfd.accept()

功能：阻塞等待客户端连接

返回值：1.返回一个新的套接字用来和客户端通信

2.返回连接的客户端的地址

5.消息的收发

recv(buffer)

功能：接收消息

参数：一次接收消息的大小，字节

返回值：返回接收到的内容

send(data)

功能：发送消息

参数：发送的内容(字节串)

返回值：返回发送了多少个字节

注：当没有接收端的时候send操作会导致broken pipe

6.关闭套接字

close()

功能：关闭套接字

tcp客户端：

1.创建套接字(须和服务端的套接字类型一致)

2.发起连接请求

connect()

功能：发起连接

参数：一个元组 第一个元素服务器ip，第二个元素服务器的port

3.收发消息

4.关闭套接字

问题：如何让一个服务器处理多个客户端任务？

练习：

1.什么是网络七层模型，每一层什么功能。tcp/ip模型又是什么

2.简述三次握手和四次挥手的过程

3.使用tcp完成一个简单的会话

## Day28

关于recv和send

发送接收缓冲区

1.发送和接收消息均先放到缓存区再进行处理

2.recv接收消息当一次接收不完的时候会下次继续接收

3.当recv阻塞时，如果客户端断开，则recv立即返回空字符串

tcp粘包处理

tcp中数据以数据流的方式发送接收，每次发送的数据间没有边界，在接收时可能造成数据的粘连即为粘包

粘包如何处理：

1.每次发送消息结束位置加标志

2.发送的消息添加结构描述

3.当连续发送时每次发送有一个短暂延迟sleep(0.1)

udp服务端

1.创建数据报套接字

sockfd = socket(AF\_INET,SOCK\_DGRAM,0)

2.绑定服务端地址

addr=('127.0.0.1',9999)

sockfd.bind(addr)

3.收发消息

data,addr=sockfd.recvfrom(buffersize)

功能：接收数据报套接字消息

参数：每次最多接收消息的大小，单位字节

返回值：data,接收的消息

addr,消息发送者的地址

\*recvfrom 一次接收一个数据包，如果数据报一次没有接收完则会丢失没有接收的内容

sendto(data,addr)

功能：发送消息

参数：data 要发送的消息

addr 发送给某个主机的地址

返回值：发送消息的字节数

4.关闭套接字

close()

示例见：code 内的 udp\_server.py和udp\_client.py

tcp 和 udp 的区别

1.tcp传输数据使用字节流的方式传输，udp是数据包

2.tcp会产生粘包现象，udp不会

3.tcp对网络条件要求高，udp更适合实时传输

4.tcp编程可以保证传输的可靠性，udp则不保证

5.tcp使用listen、accept来处理连接，udp不需要

6.消息收发tcp使用recv和send或sendall，udp使用recvfrom和sendto

补充：sendall()

用法同send()

发送成功返回None,失败产生异常

套接字属性

from socket import \*

s=socket()

s.fileno()

功能：获取套接字的描述符

\*描述法：每一个IO操作系统都会分配一个不同的整数与之对应，该整数即为此IO操作符的描述符

s.type:获得套接字类型

s.getsockname():获取套接字绑定的地址

s.getpeername()

功能：使用accept生成的套接字调用，获取该套接字对应的客户端的地址

s.setsockopt(level,optname,value)

功能：设置套接字选项

参数：

level： 要定义的选项类型，可选值有IPPROTO\_TCP,IPPROTO\_IP,SOL\_SOCKET等等

optname：根据level确定的子选项

value：根据选项设置的值

示例：s.setsockopt(SOL\_SOCKET,SO\_REUSEADDR,1)

见code 内的 sock\_attr.py

s.getsockopt(level,optname)

功能：获取套接字选项

参数：同setsockopt

返回值：返回相应选项的值

udp应用之广播

要将套接字设置为允许接收广播

将消息发送给局域网所有终端

广播地址：176.215.155.255

广播风暴：在一个网络中大量发送广播会占用大量带宽

tcp应用之http传输

http协议(超文本传输协议)

网站访问流程：

1.客户端(浏览器)发起http请求

2.传输层使用tcp协议建立连接，层层打包将请求内容发送给服务器

3.web服务器解包后解析http请求，交后端应用程序处理

4.后端应用得到结果，通过web服务器回发给前端

用途 ： 网站中网页的传输和数据传输

也可用作基于http协议的编程传输数据

特点 ： 1.应用层协议，传输层使用tcp连接

2.简单，灵活，接口使用方便

3.几乎支持所有的数据类型

4.是无状态的

5.http1.1 持续连接

请求（request）

格式 ：

请求行 确定具体的请求类型

请求头 对请求内容的信息描述

空行

请求正文 具体请求参数

请求行 ： GET /index.html HTTP/1.1

请求方法 请求资源 协议版本

请求方法：GET 获取网络资源（多用于简单获取）

POST 提交一定的附加数据，得到返回结果

HEAD 获取响应的头信息

PUT 更新服务器资源

DELETE 删除服务器资源

TRACE 用于测试

CONNECT 保留方法

OPTIONS 请求获取服务器性能和信息

请求头：

选项：值

请求体：

get请求----》get参数 &a=1&b=2

post请求----》 post提交的内容

响应(response)

格式：

响应行 反馈响应的情况

响应头 对响应的具体描述

空行

响应体 具体返回给用户的内容

响应行 ： HTTP/1.1 200 OK

协议版本 响应码 信息

1xx 提示信息，表示请求已经接收，正在处理

2xx 请求响应成功

3xx 重定向，完成任务需要其他操作

4xx 客户端错误

5xx 服务端错误

常见相应码：

200 成功

401 没有访问权限

404 资源不存在

500 服务器发生未知错误

503 服务器暂时无法执行

响应头 ：格式同请求头

响应体 ： 文件，图片。。。。

要求：

能够简述访问一个网站的流程

知道什么是http协议及基本特点

知道http请求种类，及每种请求的功能

知道http响应种类及常见的响应码

知道get请求和post请求的功能和区别

作业 ： 描述问题自己总结

通过tcp套接字发送一个文件

http协议进一步了解

## Day29

一、TCP应用 http服务器

1. 接受http请求
2. 给出一定的响应

二、IO（input output）

定义：

在内存中存在数据交换的操作就叫IO操作

1.内存和磁盘交换 如：文件读写、打印

2.内存和网络交换 如：recv、send

IO密集型程序：程序执行中执行了大量的IO操作，而较少需要cpu运算

特点：消耗cpu资源少，运行周期往往较长

cpu密集型程序：程序执行中需要大量的cpu运算，IO操作较少。

特点：占用cpu多

IO分类：

1. 阻塞IO：默认形态，是效率最低的一种IO

阻塞：

1. 因为等待某种条件达成再继续运行，如：accept、recv、input
2. 处理IO事件的时候耗时较长也会产生阻塞，如：文件的读写过程，网络数据的传输过程
3. 非阻塞IO：通过修改IO对象使其变为非阻塞状态（改变第一种阻塞形态

通常用循环不断判断阻塞条件，需要消耗更多cpu但是一定程度上提高了IO效率

s.setblocking(bool)

功能：将套接字设置的阻塞状态

参数：bool默认为True,设置为False则为非阻塞

超时等待（检查）

S.timeout(s)

功能：设置套接字的超时监测

参数：超时时间

所谓超时监测即对原本阻塞的函数进行设置，使其不再始终阻塞，而是阻塞等待一定时间后自动返回。在规定时间中如果正常结束阻塞则继续执行否则产生timeout异常

Cookie

更专业的打印异常信息(traceback)

import traceback

a = 10

try:

b = a/’i’

except TrpeError:

traceback.print\_exc()

1. IO多路复用

定义：同时监控多个IO事件，当哪个IO事件准备就绪就执行哪个IO事件，形成并发的效果

import select

select 此方法支持win,linux,unix

poll 此方法支持linux,unix

epoll 仅支持 linux，unix

r,w,x = select(rlist,wlist,xlist[,timeout])

功能：监控IO事件，阻塞等待IO事件发生

参数：rlist 列表 存放我们要监控等待处理的IO

wlist 列表 存放我们希望主动处理的IO

xlist 列表 存放如果发生异常需要我们处理的IO

timeout 数子 超时监测，默认一直阻塞

返回值：r 列表 rlist当中准备就绪的IO

w 列表 rlist当中准备就绪的IO

x 列表 xlist当中准备就绪的IO

IO多路复用注意点：

1. 在处理IO过程中不应该发生死循环（某个IO单独占有服务器）
2. IO多路复用是单进程程序，是一个并发程序
3. IO多路复用有较高的IO执行效率

poll

1. 创建poll对象 p = select.poll()
2. 加入关注的IO p.register(s[,（事件名1 | 事件名2…）]) p.unregister(s) 删除关注的IO
3. 使用poll函数监控 events = p.poll() 功能：阻塞等待register的事件只要有任意准备就绪即返回。返回值：events [(fileno,event),…]
4. poll io事件：POLLIN(rlist) POLLOUT(wlist) POLLUP(断开连接) POLLERR（xlist） POLLPRI(紧急处理) POLLVAL(无效数据)
5. 处理发生的IO事件

epoll

1. 效率上比poll和select稍微高
2. 只能用于linux和unix
3. 支持边缘触发 select和poll只支持水平触发
4. 事件驱动IO
5. 异步IO

作业：

复习tcp udp select三个代码能够自己完成

## Day30

**Cookie**

**os.unlink() 或者 os.remove()**

功能：删除某个文件

参数：一个路径文件

**os.path.exists()**

功能：判断一个文件是否存在

参数：一个路径文件

**本地套接字**

**Linux系统内的文件类型：**

b：块设备文件

c：字符设备文件

d：文件夹

-：普通文件

l：链接文件link

s：套接字文件

p：管道文件

本地套接字的意义：在linux/unix操作系统下，提供本地进程间通信的一种方式

**本地套接字（服务器端）创建流程：**

1. 创建本地套接字

sockfd = socket(AF\_UNIX,SOL\_STREAM,0)

1. 绑定套接字文件
2. 监听
3. 接收连接
4. 消息收发
5. 关闭套接字

**多任务编程**

通过应用程序利用多个计算机核心达到多任务同时执行的目的，以此来提升程序的执行效率。

**多进程，多线程：**

进程：程序在计算机中一次执行的过程

程序，是一个静态的描述，不占有计算机资源；进程，是一个动态的过程，占有cup内存的计算机资源，有一定的生命周期。

注：同一个程序，每次执行都是不同的进程。因为分配的计算机资源不同

**进程的创建流程：**

1. 用户空间运行程序发起进程创建申请
2. 调用操作系统内核接口创建进程
3. 分配计算机资源，确定进程状态
4. 将新的进程提供给用户使用

**多个进程如何占有cpu：**

1. 一个内核统一时刻只能运行一个任务
2. 多个进程对内核资源进行争夺，操作系统决定哪个进程占有计算机核心
3. 占有计算机核心的进程我们称为该进程占有cpu的时间片

**进程有哪些信息，如何保存：**

PCB(进程控制块)：在Linux或unix系统中进程创建后，会在内存开辟一块空间存放进程的相关信息，成为进程控制块

查看进程信息：ps –aux(-ajx)

STAT 表示进程状态：

D：等待态（不可中断等待）

S：等待态（可中断等待）

T：等待态（暂停）

R：运行态

Z：僵尸态

+：前台进程（默认不带+，即为后台进程）

<：高优先级

N：低优先级

l：有进程链接

s：会话组

**进程优先级：**

优先级决定了一个进程的执行权限和占有资源的优先程度

top：动态查看当前的进程的状态，q退出，< >可以翻页

linux中优先级范围 -20——19，-20最高

用户程序默认优先级为0

nice：以指定的优先级运行程序

例如：

nice -9 ./while.py 以9的优先级运行程序

nice –9 ./while.py 以-9的优先级运行程序

renice：改变某个进程的优先级

例如：

Sudo renice 2 进程号

信息：用户 PID 占有内存 优先级等

PID：在操作系统中进程的唯一标志，是大于0的整数，由系统自动分配

**进程特征：**

1. 进程是操作系统资源分配的最小单位
2. 每个进程单独占有4G虚拟内存
3. 进程之间相互独立，运行不受影响

**进程的状态：**

**三态：**

1. 就绪态：进程具备运行条件，等待系统分配处理器运行
2. 运行态：进程占有cpu处于运行的状态
3. 等待态：又称为阻塞态、睡眠态，指进程暂时不具备运行的条件，需要阻塞等待(sleep、accpet)

**五态：**

1. 新建态：创建一个进程，获取资源，直接表现为运行一个程序，或者在程序中创建新的进程
2. 就绪态：进程具备运行条件，等待系统分配处理器运行
3. 运行态：进程占有cpu处于运行的状态
4. 等待态：又称为阻塞态、睡眠态，指进程暂时不具备运行的条件，需要阻塞等待(sleep、accpet)
5. 终止态：进程执行结束，资源回收过程

**父子进程：**

在系统中除了初始化进程其他进程都有一个父进程，可能有多个子进程

进程树命令 ：pstree

要求：

1. 什么是进程
2. 进程状态，每种状态怎么回事，转换
3. 进程和线程的区别

需求：编写一个程序可以同时做多种事情

方案：使用两个进程分别完成预定事件

import os（os只能在Linux和unix系统中使用）

**os.fork()**

功能：创建一个新的进程

参数：无

返回值：三种情况，1.失败，返回一个负数，一般为-1；

2.成功，返回 0 在子进程中fork的返回值

3.成功，返回 >0的整数（新的进程的PID），在父进程中fork返回的值

注：

1. 父进程中fork之前的内容子进程同样会复制，但父子进程空间独立，fork之后的修改不会影响到对方
2. 父子进程在执行上互不影响，谁先执行，谁先执行完不确定
3. 子进程虽然复制父进程的空间，但是有自己的特性，比如自己的PID，进程PCB，进程栈空间等

**进程相关函数：**

**os.getpid()：**获取当前进程PID号

返回值：当前进程pid

**os.getppid()：**获取当前进程父进程的PID号

返回值：父进程PID

**os.\_exit(status)：**进程的退出

功能：结束一个进程

参数：表示进程的结束状态，是一个整数

**sys.exit([status])**

功能：结束一个进程，抛出异常

参数：传入一个正整数表示结束状态

传入字符串表示结束打印

**孤儿进程：**父进程先于子进程退出，此时子进程变为孤儿进程

注：孤儿进程会被系统指定的进程所‘收养’，即该进程成为孤儿进程的新的父进程。在孤儿进程退出时，“继父”会进行处理不会使其成为僵尸。

**僵尸进程：**子进程先于父进程退出，但是父进程没有处理子进程的退出状况，子进程就会变成僵尸进程。

注：僵尸进程会滞留PCB的部分信息在内存中，大量的僵尸进程会消耗系统资源，所以应该尽量避免僵尸进程的产生

**如何避免僵尸进程的产生：**

1. 让父进程先退出（不好控制）
2. 让父进程处理子进程的退出
3. 使用**wait** 或者 **waitpid** 函数

**os.wait()**

功能：等待子进程的退出进行处理

参数：无

返回值：一个二元元组，第一个值为退出的子进程PID，第二个值为子进程退出状态

**os.waitpid(pid,option)**

功能：处理子进程的退出

参数：pid：-1 表示等待任意的子进程退出

>0 表示等待相应PID号的子进程

option：0 表示阻塞等待

os.WNOHANG 表示非阻塞等待

返回值：同 wait

**os.waitpid(-1,0)** 等同于**os.wait()**

1. 使用信号处理
2. 创建二级子进程
3. 父进程创建子进程后等待子进程退出
4. 子进程创建二级进程后马上退出，二级子进程成为孤儿
5. 让父进程

**作业：**

1. 学习os.waitpid()的使用
2. 写一个群聊聊天室
3. 选择使用什么样的套接字
4. 功能描述
5. 类似于qq群聊，用户在加入聊天室有一个简单的登陆，输入用户名即可
6. 需要一个数据结构保持用户（列表或字典）
7. 当一个人发送消息，其他人可以接受消息

张三：xxxxx

1. 当一个人登陆，退出时给其他人一些提示

xxx login

xxx logout

1. 管理员发送消息，全部在线成员均可收到

## Day31

**multiprocessing 模块**

作用：创建进程

创建流程：

1. 需要将事件封装为函数
2. 使用multiprocessing提供的类创建新进程
3. 新的进程和对应的函数相关联，进程启动会自动执行函数，完成事件
4. 进程回收

创建子进程类

**multiprocessing.Process()**

功能：创建子进程

参数：name： 给创建的进程起一个名字

默认 process-1

target： 目标函数

args：元组 要给函数传递的参数 位置

kwargs：字典 要给函数传递的参数 键值

进程对象属性函数

**p.start()**

功能：启动子进程 此时进程真正创建

**p.join([timeout])**

功能：阻塞等待回收相应的子进程

参数：默认为阻塞,timeout为超时时间

**p.name：**进程名称

**p.pid：**创建的进程的PID号

**p.is\_alive()：**进程的状态

**p.daemon：**默认值为False 表示主进程结束后，不影响子进程的执行，如果设置为True，则主k进程执行完毕，所有的子进程一同退出。

**注：**

1. 设置必须在start()前
2. 一般使用daemon = True时不用加join
3. 该属性并不是linux/unix系统中所说的守护进程设置

守护进程：

1. 生命周期长，随系统创建随系统销毁
2. 不受前端控制，后台运行
3. 操作系统进程，或者是自动化运行进程居多

**作业：**编写一个程序，使用multiprocessing,要求创建两个进程来复制一个文件，各复制一半（以字节来分）

**cookie:**

size = os.path.getsize(‘文件名’)

功能：获取文件大小

## Day32

**创建自己的进程类**

示例见 code 内的 clock.py

**多进程优缺点：**

优点：并行多个任务，提高运行效率，空间独立，数据安全，创建方便

缺点：进程创建销毁的过程中消耗较多的计算机资源

**进程池**

使用条件：在需要频繁的创建删除较多进程的情况下，导致计算机资源消耗过多

进程池如何处理

1. 创建进程池，在池内放入适量的进程
2. 将事件加入进程池等待队列
3. 使用进程池中的进程不断处理事件
4. 所有事件处理后，回收关闭进程池

相关函数：

from multiprocessing import Pool

**Pool()**

功能：创建进程池

参数：processes 指定进程池中进程数量

返回值：得到进程池对象

**pool.apply\_async()**

功能：异步方式将事件放入进程池执行

参数：func 要执行的事件函数

args 同Process中args，给函数传参

kwargs同Process中kwargs，给函数传参

返回值：返回一个对象，该对象可以通过get()方法得到func函数的返回值

**pool.close()**

功能：关闭进程池，使其无法加入新的事件

**pool.join()**

功能：阻塞等待进程池退出（当所有事件处理完毕）

**pool.apply()**

用法和apply\_async一样，只是需要顺序执行，一个事件结束在执行另一个事件

**pool.map(func,iter)**

功能：类似于内建函数map，将第二个参数的迭代数传递给第一个参数的函数执行。同时兼容了使用进程池执行

返回值：返回func的返回值列表

**进程间通信**

1. 磁盘交互
2. 速度慢
3. 不安全
4. socket

本地套接字

1. 管道 （pipe）

在内存中开辟一块空间，对多个进程可见，通过管道，多进程进行通信

**multiprocessing -🡪 Pipe**

**fd1,fd2 = Pipe(duplex = True)**

功能：创建一个管道

参数：duplex 默认为True，表示双向管道

设置为False 则表示单向管道

返回值：返回两个管道流对象，表示管道的两端，如果是双向管道则两个均可读写，如果为单向管道，fd1只能读，fd2只能写

fd1.recv()

功能：接受消息（每次接收一条）

参数：无

返回值：接受到的消息

注：如果管道没有消息会阻塞

fd2.send(data)

功能：发送消息，可以是字符串或其他类型

参数：要发送的内容

注：如果没有接收端则管道破裂

1. 消息队列

队列：先进先出

在内存中开辟队列模型，用来存放消息。任何拥有队列的进程都可以存取消息

**创建队列：**

**q = Queue(maxsize = 0)**

功能：创建一个队列消息

参数：maxsize 默认为0 表示队列可存放消息由内存而定

>0 表示队列最多存放多少条消息

返回值：返回消息队列对象

**q.put()**

功能：向队列中存放消息

参数：要存的消息（字符串，整数，列表）

注：当队列满的时候会阻塞

**q.full()**

功能：判断队列是否为满，满返回True

**q.empty()**

功能：判断队列是否为空，空返回True

**q.qsize()**

功能：得到当前队列中的消息个数

**q.get()**

功能：从队列中取出消息

返回值：取出的消息

注：当队列为空时会阻塞

**q.close()：**关闭队列

put和get中均有可选参数block 和 timeout

block 默认为True 表示会阻塞，设置为False，则不会阻塞

timeout 只有block为True时才会起作用，才可设置超时时间

1. 共享内存

在内存中开辟一段空间，存储数据，对多个进程可见。每次写入共享内存的数据会覆盖之前的内容，由于对内存格式化较少，所以存取速度快

from multiprocessing import Value,Array

**obj = Value(ctype,obj)**

功能：开辟共享内存空间

参数：ctype 字符串形式参数，要转变的c类型（对照ctype表）

obj 写入共享内存的初始值

返回值：返回一个共享内存的对象（obj.value即可得到共享内存的值）

**obj = Array(ctype,obj)**

功能：开辟共享内存空间

参数：ctype 字符串形式参数，要转变的c类型（对照ctype表）

obj 存入到共享内存中的数据，如果是一个列表，要求列表中数据类型一致，如果是一个正整数，则表示开辟一个多大的序列空间

返回值：返回一个共享内存对象

**管道** **消息队列** **共享内存**

**开辟空间** 内存 内存 内存

**读写方式** 双向/单向 先进先出 操作覆盖内存

**效率** 一般 一般 快

**应用**  多用于亲缘进程 方便灵活广泛 较复杂

**是否需要互斥机制** 否 否 需要

1. 信号

一个进程向另一个进程通过信号传递某种讯息

kill –l 查看信号

**kill –signame PID** 给PID的进程发送一个信号

关于信号：

信号名称：系统定义，信号的名字

信号的含义：系统定义，信号的作用

信号的默认处理方法：系统定义，信号给接收进程带来的行为，一般有终止、暂停、忽略三种方法

python 如何操作信号

发送

**os.kill(pid,sig)**

功能：向一个进程发送一个信号

参数：pid 表示要发送信号的进程PID

sig 表示要发送的信号

**signal.alarm(sec)**

功能：向自身发送一个时钟信号SIGALRM

参数：时间秒

处理：

**signal.signal（signum,handler）**

功能：处理一个信号

参数：signum 表示要处理的信号

handler 对该信号的处理方法，有以下几种方法：

SIG\_DFL 采用默认方法

SIG\_IGN 忽略这个信号

func 自定义的方法处理

**注**：signal函数是一个异步处理信号函数，只要执行，在进程中就会按照指定方法处理信号

signal不能处理SIGSTOP和SIGKILL这两个信号

1. 信号量

给定一定的信号数量，对多个进程可见，并且多个进程均可操作。进程根据信号量的多少，可以有不同的行为

multiprocessing -🡪 Semaphore()

sem = Semaphore(num)

功能：定义信号量

参数：num 表示给定信号量的初始个数

返回值：返回信号量对象

sem.acquire() 将信号量减一，信号量为0时，会阻塞等待

sem.release() 将信号量加一

**cookie**

获取当前进程对象

**multiprocessing.current\_process()**

## Day33

使用处理僵尸进程

在父进程中，忽略子进程的发送信号

**signal(SIGCHLD,SIG\_IGN)**

**同步和互斥**

目的：对共有资源的操作会产生争夺，同步互斥是一种解决争夺的方案

临界资源：多个进程或这线程都可以操作的资源

临界区：操作临界资源的代码段

**同步**：同步是一种合作关系，为完成某个任务多进程或者多线程之间形成一种协调，按照条件次序执行，传递告知资源情况。这种协调可能是因为阻塞关系达成的

**互斥**：互斥是一种制约关系，但一个进程或线程进入到临界区会进行加锁操作，此时其他进程（线程）在企图操作临界资源就会阻塞。只有当资源被释放才能进行操作

**事件 Event**

创建事件对象：**e = Event()**

提供事件阻塞

**e.wait()**

对事件对象进程设置，此时wait判断如果事件被set，则结束阻塞

**e.set()**

清除对该事件对象的set

**e.clear()**

检测对象是否被设置set ，如果被设置返回True,否则返回False

**e.is\_set()**

**锁 Lock**

multiprocessing --🡪 Lock

lock = Lock()

lock.acquire() 上锁

lock.release() 解锁

with lock: --🡪 给with代码段上锁

….

….

--------------------🡪 with 代码段结束时自动解锁

注：在lock对象处于上锁状态的时候，再企图上锁则会阻塞直到锁被释放，才能继续执行上锁操作。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**线程**

1. 线程也是多任务编程的一种的方法，可以使用计算机多核资源。是计算机核心分配的最小单位
2. 线程有称为轻量级的进程，再创建和删除时消耗的计算机资源

线程和进程关系

1. 一个进程中可以包含多个线程
2. 进程中的所有线程共享进程的空间资源（空间、全局变量、分配的内存等）
3. 进程也有自己的特有属性，比如 指令集、 TID等

**创建线程**

import threading

threading.Thread

**Thread()**

功能：创建线程

参数：target 线程函数

args 元组，给线程函数位置传参

kwargs 字典，给线程函数字典传参

name 给线程取名字(默认Thread-1)

返回值：线程对象

t.start() 启动线程

t.join([timeout]) 回收线程

t.is\_alive() 查看线程状态

t.name 查看线程名称

threading.currentThread() 得到线程对象

t.setName() 设置线程名称

t.daemon 属性

默认为False主线程执行完毕不会影响分支线程的执行，如果设置为True，则主线程执行完毕其他线程也会终止。

t.isDaemon()：判断daemon属性是True or False

设置方法：

t.daemon = True

t.setDaemon(True)

**线程的通信**

通过修改全局变量进行线程间的通信

**创建自定义线程类**

1. 继承Thread类
2. 重写run方法

**线程中同步互斥方法**

1. Event 线程事件

e.wait() e.set() e.clear()

1. Lock 线程锁

lock = Lock() 创建锁

lock.auqire() 加锁

lock.release() 解锁

1. Condition

**进程和线程的区别和联系**

1. 两者都是多任务编程方式，均可使用计算机的多核
2. 进程的创建要比线程消耗更多的资源
3. 进程空间独立，数据安全性更好操作，有专门的进程间通信方式
4. 线程使用全局变量通信，往往要和同步互斥机制配合防止产生资源的争夺
5. 一个进程可以包含多个线程，线程共享进程资源
6. 进程线程都有自己的特有资源

**使用场景：**

1. 需要创建较多的并发，任务比较简单，线程比较合适
2. 如果程序间数据资源使用重叠比较多，要考虑到线程锁是否需要更复杂的逻辑
3. 如果多个任务并无什么关联，不易用多线程将其融入到一个进程中
4. python线程不适用于计算密集型并发

作业：

信号通信，司机和售票员的故事

1. 创建父子进程，分别表示司机和售票员
2. 当售票员捕捉到SIGINT信号时，给司机发送SIGUSER1信号，此时司机打印“老司机开车了”
3. 当售票员捕捉到SIGQUIT时，给司机发送SIGUSER2信号，此时司机打印“系好安全带，车速有点快”
4. 当司机捕捉到SIGTSTP时，发送SIGUSER1，到战后售票员先下车（子进程退出），然后司机下车

温馨提示：键盘发送信号会给终端所有进程

TFTP文件服务器

文件的上传，下载和服务端文件库的查看

客户端

1. 查看文件库中有哪些文件
2. 下载文件到本地
3. 将本地的文件上次到文件库

确定技术：fork=====>多进程

tcp socket

**os.listdir(path)**

获取该文件夹下所有的文件，形成列表

**os.path.isfile(path)**

**os.path.isdir(path)**

判断一个文件是否是普通文件/文件夹，是返回True，否则返回False

**os.path.getsize(file)**

获取文件大小

## Day34

内核=====》操作系统的核心代码

**并发**

同时处理多个请求，但是内核采用轮循时间片的方式逐个访问，某一时间点实际只处理一个任务（单线程）

比如：IO多路复用、协程、循环服务器等

**并行**

使用多个内核，同时执行多个任务

比如：多进程、多线程

推荐书籍：

计算机原理、算法导论

公众号：

python程序员、python开发者

app：掘金

**创建条件变量对象**

con = threading.Condition()

con.acquire()：对资源加锁，加锁后其他位置再加锁则阻塞

con.release()：解锁

con.wait()：wait函数只能在加锁的状态下使用。wait函数会先解锁(release),然后让线程处于等待通知的阻塞状态

con.notify()：发送通知，线程接受到通知后，结束wait阻塞，并执行acquire加锁操作

**python线程之GIL（全局解释器锁）**

python ===》支持多线程===》同步互斥===》加锁===》超级锁===》在同一时刻，解释器只能解释一个线程===》大量python库为了省事沿用了这种方法===》python多线程效率低下

GIL问题：由于python的全局解释器锁造成python的多线程执行效率地下

解决方法：

1. 不使用线程，使用多进程
2. 不适用c c++ 做解释器
3. python线程适合高用时的IO操作，比如网络IO。不适合cpu密集型程序

**设计模式**

设计模式代表了一种最佳实践，是被开发人员长期总结，用来解决某一类问题的思路方法。这些方法保证了代码的效率也利于理解

生产者模式

生产者消费者

多个消费者和多个生产者对同一个仓库的资源进行操作

总结：

1. 进程线程区别和联系
2. 同步互斥的意义和实现方法
3. 进程线程使用什么样的方式通信
4. 进程线程的特点和选择
5. 简单的设计模式的理解
6. 僵尸进程，进程状态，GIL等概念的理解

服务器模型

硬件服务器：主机 集群

软件服务器：网络服务器 在后端提供网络功能，逻辑处理，数据处理的程序或者架构等

httpserver django flask…

**服务器架构：**c/s（客户端服务器） b/（浏览器服务器）c 服务器的组织形式

服务器追求：处理速度快，数据更安全，并发量大

硬件：更高配置，更多主机，集成，分布

软件：程序占有更少的资源，更流畅的运行，处理更多的并发

**基本的服务器模型**

并发：循环 并发模式 IO多路复用

循环：单线程程序，循环接收连接或者请求，然后处理，处理后继续循环。

缺点：不能同时处理多个客户端的并行，不允许某个客户端长期占有服务器

结构比较简单，适用于udp程序，要求处理请求可以很快完成

IO多路复用模型：通过同时监控多个IO来达到IO并发的目的。

缺点：也是单线程，不能够长期阻塞，不适合处理大量cpu占有高的程序

开销小，比较适合IO密集型的服务端程序

并行服务器：

每有一个客户端连接处理，就创建一个新的进程或者线程处理客户端的请求，而主进程/主线程可以继续接受其他客户端的连接

缺点：资源消耗比较大

适合客户端需要长期占有服务器的情况

**基于fork的多进程并发**

1. 创建套接字，绑定，监听
2. 接受客户端请求
3. 创建子进程处理客户端请求，父进程继续准备接受新的客户端连接
4. 客户端退出，则销毁相应的子进程

## Day35

**多线程并发**

**相比多进程并发：**

劣势：1.可能需要同步和互斥机制，2.受到GIL的影响

优势：只需要消耗较少的系统资源

**实现：**

threading 模块完成

步骤：

1. 创建套接字
2. 准备接收客户端连接
3. 每当有一个客户端连接进来就创建一个新的线程
4. 客户端退出后结束相应线程，关闭客户端套接字

**基于多线程的HttpServer**

思路：

1. 每有一个客户端就用一个线程为其服务
2. httpserver的功能（解析请求和处理响应）封装成类
3. 所有的静态页面收集到一个文件夹
4. 处理方法放在一个文件夹

**cookie:**

sys.path： 是一个列表，可以添加路径。该列表中的路经，对python下的所有环境可见

\_\_import\_\_()：参数为一个目录，等同于import导入

使用集成模块完成网络并发

python3 ===》 socketserver 模块

进程tcp并发

forkingMixIn, TCPServer, StreamRequestHandler

进程udp并发

forkingMixIn, UDPServer, DatagramRequestHandler

线程tcp并发

ThreadingMixIn, TCPServer, StreamRequestHandler

线程udp并发

ThreadingMixIn， UDPServer, DatagramRequestHandler

# MongoDB部分

## Day36

**数据存储阶段**

1. **文件管理阶段（.txt .doc .xls…）**

优点：数据可以长期保持，可以存储大量的数据，使用简单

缺点：数据一致性差，数据查找修改不方便，数据冗余度可能比较大

1. **数据库管理阶段**

优点：

1. 数据组织结构化降低了冗余度，
2. 提高了增删改查的效率，
3. 容易扩展，
4. 方便程序调用，做自动化处理

缺点：

1. 需要使用SQL或者其他特定的语句，相对比较复杂

**几个概念**

**数据**：能够输入到计算机中并被识别处理的信息集合

**数据结构**：研究一个数据集合中数据之间关系的

**数据库**：按照数据结构，存储管理数据的仓库。数据库是在数据库管理系统管理和控制下，在一定介质上的数据集合

**数据库管理系统**：管理数据库的软件，用于建立和维护数据库

**数据库系统**：由数据库和数据库管理系统，开发工具等组成的集合

**关系型数据库**

采用关系模型来组织数据结构的数据库（二维表）

Oracle DB2 SQLServer MySql SqLite(python标准库支持)

优点：

1. 容易理解，类似我们常见的表格
2. 使用方便，都是使用sql语句，SQL语句非常成熟
3. 数据一致性高，冗余度低，完整性好
4. 技术成熟，可以使用外部连接的复杂的操作

缺点：

1. 不能很好的满足高并发的需求，每次都需要进行SQL语句的解析
2. 针对海量数据的瞬间爆发读写性能不足，关系型数据库内部每步操作都需要加锁保证操作的原子性
3. 数据扩展普遍比非关系型困难
4. 数据一致性高，有时会浪费大量空间

**非关系型数据库（Nosql----> not only sql）**

优点：

1. 高并发，大数据读写能力强
2. 支持分布式，容易扩展
3. 弱化的数据结构，降低了数据的一致性

缺点：

1. 通用性差，没有像sql那样一致的操作
2. 操作灵活，容易混乱
3. 没有join，事务支持（并非所有数据库都没有事务）等操作

**Nosql的使用情况**

1. 数据一致性要求低
2. 数据库并发处理要求高
3. 数据库设计时对大小的估算不确定，需要分布拓展
4. 给定的数据比较容易建立起Nosql的模型

**Nosql分类：**

1. 键值型数据库

Redis oracle BDB Tokyo

1. 列存储数据库

HBase

1. 文档型数据库

MongoDB CouchDB

1. 图形数据库

**MongoDB（非关系型 ---->文档型数据库）**

1. 由c++编写的数据库管理系统
2. 支持非常丰富的增删改查数据操作
3. 支持非常丰富的数据类型
4. 使用方便，便于部署，支持分布，容易拓展
5. 支持众多的编程语言接口（python ruby c++ c# PHP）

**MongoDB安装**

**自动安装：**

sudo apt-get install mongodb

默认安装路径：/var/lib/mongodb

配置文件 /etc/mongodb.conf

命令集（默认） /usr/bin 或 /usr/local/bin

**手动安装：**

1. 下载MongoDB

[www.mongodb.com](http://www.mongodb.com) =====》 Download =====》 community server选择合适版本

1. 选择安装目录解压（/usr/local或者/opt）

tar解压后得到mongo文件夹

1. 将文件夹下的命令集目录(bin目录)添加到环境变量

PATH = $PATH:/opt/mongo…./bin

export PATH

将以上两句写在/etc/rc.local中

1. 重启

**mongodb命令**

设置数据库存储位置

命令：

**mongod –dbpath 目录**

设置端口号(默认27017)

命令：

**mongod –port 8888**

进入mongo shell界面，用来操作数据库

命令：

mongo

退出mongo shell

命令：

quit() 快捷键 Ctrl + C

**组成结构：**

键值对 -----》 文档 -----》 集合 -----》 数据库

**mysql 和 mongodb 概念对比**

**mysql mongo 含义**

database database 数据库

table collection 表/集合

column field 字段/域

row document 记录/文档

index index 索引

**创建数据库**

命令：

use 数据库名

说明：

1. 如果有该数据库，则直接切换到该数据库，如果没有则先创建再切换到该数据库
2. 使用use后数据库并不会马上被创建，而是需要插入数据后数据库才会创建

**查看数据库**

命令：

**show dbs**

**数据库名称规则**

1. 原则上是任意满足以下几条的utf-8字符
2. 不能是空字符，不能含有空格、点、‘/’、‘\’、‘\0’
3. 习惯上使用英文小写
4. 长度不超过64字节
5. 不能使用admin local config 这样的名字

**admin数据库**：存储用户

**local数据库**：存储本地数据

**config数据库**：存储分片配置信息

**db**：mongo系统全局变量，代表你当前正在使用的数据库

默认为test数据库，如果插入数据，即创建test数据库

**数据库的备份和恢复**

**备份：**

mongodump –h 数据库主机名（dbhost） –d dbname(要备份的数据库) –o dbdir（备份到的路径）

示例：

mongodump -h 127.0.0.1 -d stu -o ./

将本机下stu 数据库备份到当前目录中，并会在当前目录下自动生成一个stu 文件夹则为备份文件

**恢复：**

mongorestore –h 数据库主机名:端口号 –d 目标数据库 备份文件

示例：

mongorestore –h 127.0.0.1:27017 –d test ./stu

将当前目录下的备份文件stu恢复到本机的tset数据库中

**数据库的监测命令**

mongostat

insert query updata delete :表示每秒增查改删的次数

getmore command : 表示每秒运行命令次数

dir used flushes : 每秒操作磁盘的次数

vsize res 使用虚拟内存和物理内存

**监测每个数据库的读写时长**

mongotop

ns total read write

数据集合 总时长 读时长 写时长

**删除数据库**

db.dropDatabase()

删除db所代表的数据库

例如：

>use test

>db.dropDatabase()

删除test数据库

**集合的创建**

命令：

db.createCollection(collection\_name)

示例：

db.createCollection(‘class2’)

在当前数据库中创建一个名字为class2的集合

**集合的查看**

命令：

show tables

或

show collections

**集合的命名规则**

1. 不能为空字符串，不能有’\0’
2. 不能以sustem.开头，这是系统集合的保留前缀
3. 不能和保留字重复

**创建集合方法二**

当向一个集合中插入文档时，如果该集合不存在则自动创建

示例:

db.CollectionName.insert(data)

**删除集合**

db.collectionName.drop()

示例：

db.class0.drop()

删除class0集合

**集合重命名**

db.collectionName.renameCollection(‘new\_name’)

示例：

db.class2.renameCollection(‘class0’)

将class2重命名成class0

**文档**

mongodb中文档的组织形式

键值对组成文档 -----> 类似python中的字典

bson ----> json ------> JavaScript

mongodb中文档的数据组织形式为bson格式，类似python的字典，也是由键值对构成

文档中键 相当于域或字段

文档 相当于 记录

文档中键的命名规则：

1. utf-8格式字符串
2. 不能有‘\0’，习惯上不用点和$
3. 以“\_”开头的多为保留键，自定义时一般不以“\_”开头

**注：文档键值对是有序的，并且区分大小写**

**值：mongodb的支持数据类型**

**类型 值**

**=========================================**

整型 整数

布尔类型 true /false

浮点型 小数

Arrays 数组类型[1,2,3]

Timestamp 时间戳（当前时间点）

Data 时间日期

Object 内部文档

Null 空值

Symbol 特殊字符

String 字符串

Binary Data 二进制字串

code 代码

regex 正则表达式

ObjectId ObjectId字串

ObjectId：系统自定为每个文档生成的不重复的主键

键名称：\_id

值：ObjectId("5b03b83e5cad38143aaacf80")

24位16进制数

8 文档创建时间 6机器ID 4进程ID 6 计数器

**集合中文档的特点：**

1. 集合中的文档域不一定相同，不保证数据一致性
2. 集合中的文档结构不一定相同

**集合设计原则：**

1. 集合中的文档尽可能描述的数据类似
2. 同一类文档放在相同的集合，不同的文档分集合存放
3. 层次的包裹不宜太多

**插入文档**

db.collectionName.insert({})

注：当作为文档插入时键可以不加引号

**查看插入结果**

db.class0.find()

**插入多条文档**

db.collectionName.insert([{},{},{}….])

db.class0.insert([{name:'张三',age:20,sex:'男'},{name:'李四',age:22,sex:'男'}])

**注：\_id 为系统自动添加主键，如果自己写\_id域则会使用自己写的值，但是该值仍不允许重复**

**save 插入数据**

db.collection.save({})

注：

1. 在不加 \_id 时使用同 insert，
2. 如果使用save插入的时候加入 \_id ，则如果 \_id 值不存在，正常插入，如果该值存在，则修改原来内容，
3. save 无法一次插入多个文档

作业：

练习mongodb的数据库，集合创建删除

mongodb 插入练习

关系型数据库和非关系型数据库都有什么特点

mongodb的优点在哪里？

## Day37

db.collectionName 集合对象

**获取集合对象**

命令：

db.getCollection(‘collectionName’)

示例：

db.getCollection(‘class0’).insert({name:’zhang’,age:17})

**查找操作**

**db.collectionName.find() 等同于 select \* from tableName**

**find(query,field)**

功能：查找所有符合条件的文档

参数：query 筛选条件 相对于where字句

field 展示的域 相当于select的展示部分

返回：返回所有查找到的内容

**field 参数**： 选择要展示的域，传一组键值对

**键表示域名**

**值表示是否显示该域 0表示不显示，1表示显示**

**注**：如果某个域给定0，则表示不显示该域，其他的域均显示

如果某个域给定1，则表示显示该域，其他的域均不显示

\_id 永远默认认为显示，除非设置为0

除 \_id 外其他域必须拥有相同的设置，全为0或者全为1

如果不写该参数则表示显示所有域内容

**query 参数**：以键值对的形式给出查找条件

示例：查找Lucy的信息

db.class0.find({name:'Lucy'},{\_id:0})

注：如果不写第一个参数则表示查找所有内容

query的更多用法

操作符：使用$符号注明一个特殊字符串，表示一定的含义

例如：$lt 表示小于

**比较操作符：**

**$eq 等于**

例如：db.class0.find({age:{$eq:20}},{\_id:0})

筛选年龄等于20的

**$lt 小于**

例如：db.class0.find({age:{$lt:20}},{\_id:0})

筛选年龄小于20的

db.class0.find({name:{$lt:'Lucy'}},{\_id:0})

筛选名字小于’Lucy’的信息

**$lte 小于等于**

例如：db.class0.find({age:{$lte:18}},{\_id:0})

**$gt 大于**

例如：db.class0.find({age:{$gt:18}},{\_id:0})

**$gte 大于等于**

db.class0.find({age:{$gte:18}},{\_id:0})

**$ne 不等于**

db.class0.find({age:{$ne:18}},{\_id:0})

**注：**如果一个文档没有age域，则显示为不等于

**$in 包含**

db.class0.find({age:{$in:[16,17,18]}},{\_id:0})

筛选年龄是16，17，18的

**$nin 不包含**

db.class0.find({age:{$nin:[16,17,18]}},{\_id:0})

筛选年龄不包含16，17，18的

**逻辑操作符**

**$and 逻辑与**

例：db.class0.find({age:{$gt:19,$lt:22}},{\_id:0})

年龄大于19，小于22的

db.class0.find({age:20,name:'张三'},{\_id:0})

名字为张三，年龄为20

db.class0.find({$and:[{age:20},{name:'张三'}]})

名字为张三，年龄为20

**$or 逻辑或**

例：db.class0.find({$or:[{age:{$lt:19}},{name:'张三'}]})

年龄小于19或者名字为张三的文档

**$not 逻辑非**

例：db.class0.find({age:{$not:{$eq:20}}},{\_id:0})

年龄不是20的文档

**$nor 既不也不**

**findOne()**

功能参数和find（）完全相同，但只显示找到的第一条数据

**数组查找**

查看数组中包含某一项

例：db.class1.find({hobby:'唱歌'},{\_id:0})

**$all**

查找一个数组中同时包含多项的文档

例：db.class1.find({hobby:{$all:['玩火','唱歌']}},{\_id:0})

hobby中同时有 玩火 和 唱歌

**$size**

查找数组元素个数为指定个数的文档

例：db.class1.find({hobby:{$size:2}},{\_id:0})

**数组切片显示**

**$slice**

db.class1.find({hobby:{$size:2}},{\_id:0,hobby:{$slice:1}})

显示数组的前一项

db.class1.find({hobby:{$size:2}},{\_id:0,hobby:{$slice:[1,2]}})

显示数组中，跳过第一项，显示后两项的文档

**其他查找方法：**

**$exists**

判断一个域是否存在

例：db.class1.find({sex:{$exists:true}},{\_id:0})

db.class1.find({sex:{$exists:false}},{\_id:0})

**$mod**

做余数除数查找

例：db.class0.find({age:{$mod:[2,1]}},{\_id:0})

筛选年龄被2除余1的文档

db.class0.find({age:{$mod:[3,0]}},{\_id:0})

筛选年龄被3整除的文档

**$type**

查找指定数据类型的文档

例: db.class0.find({age:{$type:1}})

查找年龄是数字类型的文档

说明：1 表示的是数据类型的编码，其他数据类型的编码见官方文档说明。

例：db.class1.find({hobby:{$type:2}})

查找hobby数组中含有 字符串类型（2） 数据的文档

进一步的信息筛选

**distinct()**

功能：查看一个集合中某个域值的覆盖范围

例：db.class1.distinct('hobby')

查看集合中hobby域都有哪些元素（去重）

**pretty()**

功能：将查询结果格式化显示

例：db.class1.find().pretty()

**limit(n)**

功能：查询结果显示前n条记录

例：db.class1.find({},{\_id:0}).limit(2)

**skip(n)**

功能：显示时跳过前n条

例：db.class0.find({},{\_id:0}).skip(3)

**conut()**

功能：对查找结果计数统计

例：db.class0.find({sex:'男'}).count()

统计性别是“男”的文档的数量

**sort({键：1/-1})**

功能：对查找结果排序

1 表示按照升序排列，-1 表示按照降序排列

例：db.class1.find({sex:{$exists:true}},{\_id:0}).sort({age:1})

对查找结果按年龄升序排列

例：db.class0.find({},{\_id:0}).sort({age:1,name:-1})

当第一排序项相同时，再按第二排序项排序

**函数的连续使用**

例：db.class0.find({},{\_id:0}).sort({age:1,name:-1}).limit(3)

查找年龄最小的前三条信息

**删除文档**

**db.collectionName.remove(query,justOne)**

功能：删除指定的文档

参数：query 筛选要删除的文档，类似where字句，用法同查找操作

justOne 布尔值，默认false 表示删除所有筛选数据

true 则表示只删除第一条符合条件的文档

例：db.class0.remove({sex:'w'},true)

**删除集合中所有的文档**

db.collectionName.remove({})

练习：

1. 创建一个数据库grade

use grade

1. 数据库中创建一个集合class

db.createCollection(‘class’)

1. 集合中插入若干数据，文档格式如下：

{name:’zhang’,age:10,sex:’m’,hobby:[‘a’,’b’…]}

1. 查找练习
2. 查看班级所有人信息

db.class.find()

1. 查看班级中年龄为8岁的学生信息

db.class.find({age:8})

1. 查看年龄大于10岁的学生的信息

db.class.find({age:{$gt:10}})

1. 查看年龄在4---8岁之间的学生信息

db.class.find({age:{$gt:4,$lt:8}})

1. 年龄为6岁且为男生的学生

db.class.find({age:6,sex:’m’})

1. 年龄小于7或者大于10的学生

db.class.find({$or:[{age:{$lt:7}},{age:{$gt:10}}]})

1. 年龄是8或则是11岁的学生

db.class.find({$or:[{age:8},{age:11}]})

1. 找到兴趣爱好有两项的学生

db.class.find({hobby:{$size:2}})

1. 找到兴趣爱好有draw的学生

db.class.find({hobby:’draw’})

1. 找到既喜欢画画又喜欢跳舞的学生

db.class.find({hobby:{$all:[‘draw’,’dance’]}})

1. 统计爱好有三项的学生人数

db.class.find({hobby:{$size:3}}).count()

1. 找出本班年龄第二大的学生

db.class.find({},{\_id:0}).sort({age:-1}).skip(1).limit(1)

1. 查看学生的兴趣范围

db.class.distinct('hobby')

1. 找到年龄最大的三个学生

db.class.find().sort({age:-1}).limit(3)

1. 删除年龄大于12或者小于4岁的学生

db.class.remove({$or:[{age:{$gt:12}},{age:{$lt:4}}]})

## Day38

**修改数据**

**db.collectionName.update(query,update,upsert,multi)**

功能：修改一个文档

参数：1.query 筛选要修改的文档，相当于where子句，用法同查找

2.update 将内容更新成为什么内容，相当于set操作，需要使用修改器操作符

3.upsert 布尔值，默认值为false,表示如果query的文档不存在则无法修改，如果为true，则根据query和update参数插入新的文档

4.multi 布尔值，默认为false,表示如果有多条符合筛选条件的文档则只修改第一条，如果设置为true，则修改所有符合条件的文档

例：db.class0.update({name:'Lucy'},{$set:{sex:'女'}})

将Lucy的sex修改为‘女’

db.class0.update({name:'Lily'},{$set:{age:17,sex:'女'}},true)

修改Lily的年龄和性别，如果不存在Lily则插入此文档

db.class0.update({sex:'女'},{$set:{age:18}},false,true)

将所有性别为女的年龄修改成18

**修改器操作符**

**$set**

修改一个域的值，也可以增加一个域

**$unset**

删除一个域

例：db.class0.update({name:'Lucy'},{$unset:{hobby:0,age:0}})

删除Lucy的age域和hobby域（后面的数字随便写无特殊意义，一般写0或1）

**$rename**

修改域名

例：db.class0.update({},{$rename:{sex:'gender'}},false,true)

将所有文档的sex域名修改成gender

**$setOnInsert**

如果update操作插入了新的文档，则补充插入内容

例：db.class0.update({name:'王五'},{$set:{age:19},$setOnInsert:{gender:'男',hobby:'football'}},true)

如果插入王五数据，同时也插入age 域 和gender域

**$inc**

加减修改器（正、负、小数都可以）

例：db.class0.update({age:{$lt:19}},{$inc:{age:1}},false,true)

年龄小于19的全部加1

**$mul**

乘法修改器，用法同$inc，正数、负数、小数都可以

**$min**

设定最小值：如果筛选的文档指定的域值小于min值则不修改，如果大于min值，这修改成为min值

例：db.class0.update({},{$min:{age:20}},false,true)

将所有年龄大于20的全部修改成20

**$max**

设定最大值：如果筛选的文档指定值大于max值则不变，如果小于max值则修改成为max值

例：db.class0.update({},{$max:{age:20}},false,true)

将所有年龄小于20的修改成20

**数组修改器**

**$push**

向数组中添加一项

例：db.class1.update({name:'林动'},{$push:{hobby:'聪明'}})

向林东的hobby中添加 聪明 向

**$pushAll**

向数组中添加多项

例：db.class1.update({name:'牧尘'},{$pushAll:{hobby:['义气','游手好闲']}})

**$each**

逐个操作

例：db.class1.update({name:'林虎'},{$push:{hobby:{$each:['忠心','不怕死']}}})

**$position**

选择数据位置进行操作，必须和each合用

例：db.class1.update({name:'林虎'},{$push:{hobby:{$each:['愣头青','闷葫芦'],$position:0}}})

在数组的 0位置（第一项）处插入数据

**$sort**

对数组进行排序，必须和each合用

例：db.class1.update({name:'萧炎'},{$push:{hobby:{$each:['心机','蛇皮'],$sort:1}}})

在数组中添加数据后再进行排序

**$pull**

删除数组中的一个数据

例：db.class1.update({name:'萧炎'},{$pull:{hobby:'蛇皮'}})

**$pullAll**

删除数组中的多个元素

例：db.class1.update({name:'林虎'},{$pullAll:{hobby:['装逼','愣头青']}})

**$pop**

弹出数组中的一项

例：db.class1.update({name:'牧尘'},{$pop:{hobby:1}})

弹出最后一项

说明：-1表示弹出数组的第一项，1表示数组的最后一项

**$addToSet**

向数组中插入一个元素，但该元素不能和其他元素重复

例：db.class1.update({name:'萧炎'},{$addToSet:{hobby:'玩火'}})

插入 玩火 数据,如果存在则无法插入

可以防止数组中的数据重复

数据补充

**时间类型**

mongo中存储时间的格式：ISODate

方法一 自动生成当前时间

db.class2.insert({title:'Python入门',date:new Date()})

方法二 生成当前时间

db.class2.insert({title:'Python精通',date:ISODate()})

方法三 将当前时间变为字符串存储

db.class2.insert({title:'Python人工智能',date:Date()})

**指定时间的转换**

ISODate()

功能：生成mongo时间类型

参数：如果不加参数则生成当前时间，如果加参数，

格式为：’2018-05-24 11:11:11’

’20180525 11:11:11’

‘20181111’

**时间戳获取**

db.class2.insert({title:'PythonWeb前端',date:ISODate().valueOf()})

**Null**

1. 如果某个域存在却没有值可以设置为null

db.class2.insert({title:'Python秘籍',price:null})

1. 表示某个域不存在，可以用null进行匹配

db.class2.find({date:null},{\_id:0})

可以查找到没有date域的文档

**Object类型（值为一个文档**）

db.class2.find({'publication.publisher':'人民教育'},{\_id:0})

当使用外层文档引用内部文档的时候可以用 . 的方法引用，在使用时需要加上引用。

db.class2.update({title:'Python标准库'},{$set:{'publication.price':45.6}})

将publication内的price修改为45.6

**数组的下标引用**

使用一个数组时，可以使用 .序列下标 的方式使用数组具体的某一项。同样需要用引号

**文档查找结果的有序性**

可以通过[]取查找结果序列的某一项

db.class1.find({},{\_id:0})[2]

练习：

1. 将小红的年龄变为8岁，兴趣爱好变为跳舞，画画
2. 追加小明兴趣爱好 唱歌
3. 小王兴趣爱好增加吹牛，打篮球
4. 小李增加爱好，跑步和唱歌，但是不要和以前的重复

{$addToSet:{hobby:{$each:[‘run’,’sing’]}}}

1. 该班所有同学年龄加1
2. 删除小明的sex属性
3. 删除小李兴趣中的第一项

{$pop:{hobby:-1}}

1. 将小红兴趣中的画画爱好删除

{name:’小红’},{$pull:{hobby:’draw’}}

**索引**

指的是建立指定键值及所在文档中存储位置的对照清单。使用索引可以方便我们进行快速查找，减少遍历次数，提高查找效率

mongo中创建索引

**ensureIndex()**

功能：创建索引

参数：索引类别，索引选项

例：db.class1.ensureIndex({name:1})

说明：1表示为该域创建正向索引，-1表示逆向索引

\_id域会自动创建索引

**getIndexes()**

查看索引

例：db.class.getIndexes()

**dropIndex()**

功能：删除索引

参数：索引名

例：db.class.dropIndex('name\_1') 根据索引名

或

db.class.dropIndex({name:1}) 根据索引键值对

**dropIndexes()**

功能：删除所有索引，除了\_id

例：db.class.dropIndexes()

**复合索引**

db.class.ensureIndex({name:1,age:1})

根据多个域创建一个索引

**数组索引**

如果对某个数组域创建索引，那么表示对数组中的每个值均创建看索引，通过数组中单个值查询，也是索引查询

例：db.class.ensureIndex({hobby:1})

**子文档索引**

如果对一个域创建索引，值是一个文档则也会同时形成索引

如果对子文档某一个域进行索引创建，则只有通过子文档中的该域查找时才为索引查找

**覆盖索引**

查找时只获取索引项的内容，而不必取获取原数据中的其他内容，这样就不去连接原来的数据直接返回即可

例：db.class0.find({name:’Lily’},{\_id:0,name:1})

只显示{ "name" : "Lily" }

**唯一索引**

创建索引时希望索引域的值均不相同，也可以据此限制一个域的值

例：db.class.ensureIndex({name:1},{unique:true})

说明：当对某个域创建了唯一索引后，即不允许再插入相同的值的文档

**稀疏索引（间隙索引）**

只针对有指定域的文档创建索引表，没有该域的文档不会插入到索引表中

例：db.class2.ensureIndex({date:1},{sparse:true})

**索引约束**

1. 影响数据的插入，删除，修改操作。当数据发生改变时，索引表也必须同步更新
2. 索引也是需要占用一定的空间资源
3. 当数据库大量的操作是插入，修改，删除，而非查询操作时，不适合创建索引。
4. 数据量比较小时，考虑到空间成本也不适合创建索引。
5. 即使适合创建索引的情况，也不是索引越多越好

**聚合**

对数据文档进行整理统计

db.collectionName.aggregate()

功能：聚会函数，配合聚合条件进行数据整理统计

参数：聚合条件

**聚合操作符**

**$group** 分组 和分组操作符配合使用确定按什么分组

**分组操作符**

**$sum** 求和

统计结果名

例：db.class0.aggregate({$group:{\_id:'$gender',num:{$sum:1}}})

聚合 分组 按该域分组

结果显示

{ "\_id" : "男", "num" : 4 }

{ "\_id" : "女", "num" : 2 }

db.class0.aggregate({$group:{\_id:'$gender',num:{$sum:‘$age’}}})

**$avg** 求平均数

**$min** 求最小值

db.class0.aggregate({$group:{\_id:'$gender',num:{$min:'$name'}}})

按gender分组后求每组姓名的最小值

**$max** 求最大值

db.class0.aggregate({$group:{\_id:'$gender',最大值:{$max:'$name'}}})

**$first** 返回每组第一个文档的指定域值

db.class0.aggregate({$group:{\_id:'$gender',name:{$first:'$name'}}})

**$last** 返回每组最后一个文档的指定域值

db.class0.aggregate({$group:{\_id:'$gender',name:{$last:'$name'}}})

**$project**

用于修饰文档的显示结构

**db.class0.aggregate({$project:{\_id:0,name:1,age:1}})**

显示结果

{ "name" : "Lucy", "age" : 20 }

{ "name" : "张三", "age" : 20 }

{ "name" : "李四", "age" : 20 }

{ "name" : "回锅肉", "age" : 20 }

{ "name" : "Lily", "age" : 20 }

{ "name" : "王五", "age" : 20 }

**db.class0.aggregate({$project:{\_id:0,Name:'$name',Age:'$age'}})**

显示结果

{ "Name" : "Lucy", "Age" : 20 }

{ "Name" : "张三", "Age" : 20 }

{ "Name" : "李四", "Age" : 20 }

{ "Name" : "回锅肉", "Age" : 20 }

{ "Name" : "Lily", "Age" : 20 }

{ "Name" : "王五", "Age" : 20 }

**$match** 过滤数据

操作符的值同find的query

db.class.aggregate({$match:{sex:'m'}})

**$skip** 跳过前几条文档

db.class.aggregate({$skip:2})

**$limit** 显示前几条

db.class.aggregate({$limit:2})

**$sort** 排序 1表示升序， -1表示降序

db.class.aggregate({$sort:{age:1}})

**聚合管道**

将前一个聚合操作的结果给下一个聚合操作继续执行

db.collectionName.aggregate([聚合1，聚合2，…..])

练习：

使用之前的grade数据库

增加分数域score:{‘chinese’:88,’english’:78,’math’:98}

1. 按照性别分组统计每组人数

db.class.aggregate({$group:{\_id:'$sex',num:{$sum:1}}})

1. 按照姓名分组，过滤出有重名的同学

db.class.aggregate([{$group:{\_id:'$name',count:{$sum:1}}},{$match:{count:{$gt:1}}}])

1. 统计每名男生的语文成绩

db.class.aggregate([{$match:{sex:'m'}},{$project:{\_id:0,name:1,'sorce.chinese':1}}])

1. 将女生按照英语分数降序排序

db.class.aggregate([{$match:{sex:'w'}},{$sort:{'sorce.english':-1}},{$project:{\_id:0,name:1,'sorce.english':1}}])

## Day39

**固定集合**

mongo中可以创建大小固定的集合称为固定集合，固定集合的性能出色，适用于很多场景，比如：日志处理，临时缓存

特定：插入速度快，顺序查询速度快，能够淘汰早期数据，可以控制集合空间

**创建：**

**db.createCollection(collectionName,{capped:true,size:1000,max:number})**

size：设置固定集合的大小 单位kb

max：最多能容纳多少文档

例：db.createCollection('class3',{capped:true,size:10,max:3})

**文件存储**

数据库存储文件的方式

1. 在数据库中以字符串的方式存储文件在本地的路径

优点：节省数据库空间

缺点：当数据库或者文件位置发生变化即需要相应的修改数据库内容

1. 将文件已二进制数据的方式存放在数据库里

优点：文件存入数据库，数据库在，文件就不会丢失

缺点：当文件较大时，数据库空间占有大，提取困难

mongo中，使用GridFS方法 大文件存储

**GridFS：**是mongodb中大文件存储的一种方案，mongo中认为大于16M的文件为大文件

方案解释：

在mongodb数据库中，创建两个集合共同完成对文件的存储

**fs.files：**cunc文件的相关信息，比如：文件名，文件类型

**fs.chunks：**实际存储文件内容，以二进制方式分块存储。将大文件分为多个小块，每块占一个空间

存储文件到数据库

mongofiles –d dbname put file

数据库 要存储的文件

说明：如果数据库不存在则自动创建

获取数据库文件

mongofiles –d dbname get file

大文件存储优缺点：

优点：存储方便，方便数据库移植，对文件个数没有太多限制

缺点：读写效率低

游标

为什么使用游标

1. 防止网络拥塞，造成数据传输缓慢
2. 提高用户解析体验，可以后端解析

var c = db.class0.find() #创建游标

c.hasNext() #查看是否有下一个数据

c.next() #获取下一个数据

通过Python操作mongodb数据库

mongodb编程接口 pymongo

安装库

sudo pip3 install pymongo

操作步骤：

1. 创建mongo数据库的链接对象

conn = MongoClient(‘localhost’,27017)

1. 生成数据库对象

db = conn.stu

1. 生成集合对象

my\_set = db.class0

1. 增删改查索引聚合操作

**插入数据**

insert() insert\_many() insert\_one() save()

**删除数据**

remove({},multi = True)

mulit 默认为True 表示删除所有符合条件的数据

设置为false 表示只删除一条

**数据查找**

find()

功能：查找数据库内容

参数：同mongo shell find()

返回值：返回一个游标 -----> 迭代器

cursor = my\_set.find({},{'\_id':0})

cursor 为可迭代对象

next()

limit()

skip()

count()

max()

min()

sort()

mongoshell------>sort({name:1})

pymongo ------> sort([(‘name’,1)])

注：进行排序时游标必须是没有被访问过的

**数据修改**

update()

参数和mongoshell中的update相同

update\_many()：匹配到多个文档时全部修改

update\_one()：只修改匹配到的第一条文档

说明：编程中mongo的数据类型null可以用python中的None代替

**索引**

创建索引 ensure\_index() create\_index() create\_indexes()同时创建多个

查看集合中的索引 list\_indexes()

删除索引

drop\_index() 删除某一个索引

drop\_indexex() 删除所有索引

聚合

aggregate([])

参数：参数与mongoshell中聚合参数写法一致

返回值：迭代器

## Day40

**正则表达式**

动机

1. 处理文本成为计算机常见工作之一
2. 对文本内容的搜索提取是一项比较复杂困难的工作
3. 为了快速方便处理上述问题，正则表达式技术诞生，逐渐发展为一个被众多语言使用的独立技术

定义：即高级文本匹配模式，提供了搜索，替代等功能。本质是由一系列特殊符号和字符组成的字串，这个字串即使正则表达式。这个表达式描述了字符和字符的重复行为，可以匹配一类特征的字符串

目标：

1. 熟练使用正则表达式
2. 正确组合和理解一般的正则表达式
3. 能够使用python操作正则表达式

正则表达式特点

1. 方便进行检索和修改
2. 支持语言众多
3. 灵活多样
4. Mongo正则类型，django等框架作为url匹配，爬虫

正则表达式的使用

Python ----🡪re模块 处理正则表达式

re.findall(pattern,string)

功能：使用正则表达式匹配字符串

参数：pattren 正则表达式

string 目标字符串

返回值：一个列表 匹配到的所有内容

**元字符（即正则表达式中有特殊含义的符号）**

1. **普通字符**

元字符：abc

匹配规则：匹配相应的普通字符

例：>>>re.findall('abc','abcdfsdabc') #[‘abc’, ’abc’]

1. **或**

元字符：ab|cd

匹配规则：匹配 | 两边任意一个正则表达式符合的情况，两侧不要有没用的空格

1. **匹配单一字符**

元字符：.

匹配规则：匹配除了换行之外的所有字符

例：>>>re.findall('f.o','foo is not fao') #['foo', 'fao']

1. **匹配开始位置**

元字符：^

匹配规则：匹配一个字符串的开始位置

例：>>>re.findall('^hell','hello world,hello everyone') #['hell']

1. **匹配结束位置**

元字符：$

匹配规则：匹配目标字符串的结束位置

例：>>>re.findall('py$','python3 hello.py') #[‘py’]

1. **匹配重复**

元字符：\*

匹配规则：匹配前面的正则表达式重复0次或多次

例：>>>re.findall('fo\*','foo o f') #['foo', 'f']

1. **匹配重复**

元字符：+

匹配规则：匹配前面的正则表达式重复1次或多次

1. **匹配重复**

元字符：?

匹配规则：匹配前面的正则表达式重复0次或1次

例：>>>re.findall('ab?','abbcdac') #['ab', 'a']

1. **匹配重复**

元字符：{N}

匹配规则：匹配指定的重复次数

ab{3} ----🡪abbb

例：>>>re.findall('ab{3}','abbbbb') #[‘abbb’]

1. **匹配重复**

元字符：{m,n}

匹配规则：匹配前面的正则表达式重复m次到n次

ab{3,5} abbb abbbb abbbbb

例：re.findall('ab{2,5}','abdfsdabbcdsabbbbbbb') #[‘abb’, ’abbbbb’]

1. **匹配字符集合**

元字符：[]

匹配规则：匹配括号范围内的任意一个字符

[abc123d] a b c 1 2 3 d

[a-z]

[A-Z]

[0-9]

[123a-zA-Z]

1. **匹配字符集合**

元字符：[^…]

匹配规则：匹配除指定字符集之外的任意字符

例：>>>re.findall('[^0-9a-z]','230sfsAdfFS') #['A', 'F', 'S']

1. **匹配任意数字（或非数字）字符**

元字符： \d \D

匹配规则：\d 匹配任意数字字符 \D匹配任意非数字字符

例：>>>re.findall('1\d{10}','1353235322435') #['13532353224']

1. **匹配普通(非普通)字符**

普通字符：数字、字母、下划线

元字符：\w \W

匹配规则：\w匹配任意一个普通字符 \W匹配任意非普通字符

例：>>>re.findall('\w+','hello&\*33sdf') #['hello', '33sdf']

1. **匹配空字符或非空字符**

元字符：\s \S

匹配规则：\s匹配任意空字符[\n\t\r…] \S匹配任意非空字符

例：>>>re.findall('\s','hello world\r\n\t\v') #[' ', '\r', '\n', '\t', '\x0b']

1. **匹配起止位置**

元字符：\A \Z

匹配规则：\A匹配开始位置 \Z匹配结束位置

绝对匹配：\Aabc\Z ---🡪 abc(且字符串只是abc)

1. **匹配（非）单词边界位置**

元字符：\b \B

匹配规则：\b匹配单词的边界 \B匹配非单词的边界

单词边界：数字、字母、下划线和其他字符的交界位置为单词的边界

例：re.findall(r'\bis\b','this is a test') #[‘is’] 第二个is

re.findall(r'\Bis\b','this is a test') #[‘is’] this中的is

元字符总结

匹配单个字符：a . \d \D \w \W \s \S […] [^….]

匹配重复性：\* + ? {N} {m,n}

匹配某个位置：^ $ \A \Z \b \B

其他：| （） \

**匹配长度8-10位的密码，必须以数字开头，数字字母下划线组成**

**^[0-9]\w{7,9}$**

转义字符

正则表达特殊符号：

. \* ? $ ‘’ “” [] {} () \ ^

如果想匹配特殊符号则加转义

\”\.\” --🡪 “.”

r----🡪raw 原生字符串 不转义字符

**贪婪和非贪婪**

正则表达式默认的重复匹配模式：贪婪模式

尽可能多的向后匹配

+ \* ？ {m,n}这四种情况下产生贪婪模式

非贪婪模式：尽可能少的匹配内容，满足正则表达式含义即可

贪婪 -🡪 非贪婪 \*? +? ?? {m,n}?

**正则表达式分组**

使用（）可以为一个正则表达式建立一个子组，子组可以看做内部的整体

**子组的作用**

1. 增加子组后对正则表达式整体的匹配内容没有影响
2. 子组可以改变重复元字符的重复行为
3. 子组在某些操作中可以对子组匹配内容单独提取

**子组的注意事项**

1. 每个正则表达式可以有多个子组，有外到内由左到右为第一第二…子组
2. 子组通常不要交叉

**捕获组和非捕获组（命名组和非命名组）**

子组命名格式

(?P<name>abc)

1. 很多编程接口可以直接通过名字获取子组匹配内容
2. 捕获组中的正则表达式可以通过名字重复调用

（?P=name）

(?P<dog>ab)cdef(?P=dog) -----🡪abcdefab

匹配身份证号

re.search(‘\d{17}(\d|x)’).group()

re 模块

obj = compile(pattern,flags=0)

功能：获取正则表达式对象

参数：pattern 正则表达式

flags 功能标志位，提供更丰富的匹配

返回值：正则表达式对象

compile 对象属性方法（re模块没有）

print('flags:',obj.flags) #标志位常量

print('pattern:',obj.pattern) #正则表达式

print('groupindex:',obj.groupindex) #捕获组字典

print('groups:',obj.groups) #子组数量

obj.findall(string,pos,endpos)

功能：通过正则表达式匹配字符串

参数：string 目标字符串

pos 目标字符串的匹配开始位置

endpos 目标字符串的结束位置

返回值：匹配到的所有内容以列表返回

说明：如果正则表达式有子组则只显示子组匹配内容

obj.split(string)

功能：按照正则表达式切割目标字符串

参数：目标字符串

返回值：切割后的内容

obj.sub(replaceStr, string, count)

功能：替换正则表达式匹配到的内容

参数：replaceStr要替换的内容

string 目标字符串

count 最多替换几处

返回值：返回替换后的字符串

subn(replaceStr string count)

功能：替换正则表达式匹配到的内容

参数：replaceStr要替换的内容

string 目标字符串

count 最多替换几处

返回值：元组 返回替换后的字符串和实际替换的处数

re.finditer()

功能：匹配正则表达式匹配目标内容

参数：目标字符串

返回值：迭代对象，迭代的每个内容为一个match对象

re.match(pattern, string)

功能：匹配一个字符串,匹配内容必须在字符串开头，只能匹配一处

参数：pattern 正则表达式

string 目标字符串

返回值：如果匹配到，返回match 对象

如果没有匹配到，返回None

re.search(pattern,string)

功能：匹配一个字符串,匹配内容可以是任意位置，但只能匹配一处

参数：pattern 正则表达式

string 目标字符串

返回值：如果匹配到，返回match 对象

如果没有匹配到，返回None

注：通常match对象调用其属性时往往需要try异常处理

re.fullmatch(pattern,string)

功能：要求目标对象完全匹配

参数：pattern 正则表达式

string 目标字符串

返回值：如果匹配到，返回match 对象

如果没有匹配到，返回None

match 对象属性方法

属性：

print(match\_obj.pos) #目标字符串开头位置

print(match\_obj.endpos) #目标字符串结束位置

print(match\_obj.re) #正则表达式对象

print(match\_obj.string) #目标字符串

print(match\_obj.lastgroup) #最后一组的组名

print(match\_obj.lastindex) #最后一组是第几组

方法：

print(match\_obj.start()) #匹配到内容的开始位置

print(match\_obj.end()) #匹配到内容的结束位置

print(match\_obj.span()) #匹配到内容的起止位置

group(n)

功能：获取mathc对象匹配的内容

参数：默认为0，表示获取正则整体的匹配内容，如果传入大于0的正数则表示获取对应子组匹配内容

返回值：返回匹配到的内容

groups()

功能：得到所有子组匹配的内容

groupdict()

功能：得到所有捕获组匹配的内容

**作业**：读取一个文件的内容，将文件中所有以大写字母开头的单词匹配出来

**flags 参数**

作用：辅助正则表达式匹配

re.compile re.findall re.search re.match re.finditer re.split re.sub

re.subn re.fullmatch

I或IGNORECASE：匹配时忽略大小写

例：re.findall(‘h\w+’,’Hello World’,re.I) #[‘Hello’]

M或MULTILINE：对元字符^和$起作用，可以匹配每一行的开头和结尾

例：re.findall('^hello','hello world\nhello python',re.M) # ['hello', 'hello']

S或DOTALL：对元字符 . 起作用，让其可以匹配换行

例：re.findall('.+','hello world\nhello') # ['hello world', 'hello']

re.findall('.+','hello world\nhello',re.S) # ['hello world\nhello']

X或VERBOSE：可以为正则表达式添加注释

**多个flags同时使用**

**re.flags1 | re.flags2**

## Day41

什么是项目

**软件项目开发流程**

需求分析====》概要设计====》项目计划====》详细设计====》编码测试====》项目测试====》调试修改====》项目发布====》后期维护

需求分析：确定用户的真实需求，反复确认，使用用户能理解的表述进行表达，完成需求文档，用户确认

概要设计：对项目进行初步分析和整体设计

1. 确定功能模块
2. 确定技术思路和使用框架
3. 进行可行性分析，搭建整体架构图
4. 形成概要文档指导开发流程

项目计划：确定项目开发的时间轴和流程

1. 确定开法工作的先后顺序
2. 确定时间轴，事件里程碑
3. 人员分工
4. 形成甘特图和思维导图等辅助内容

详细设计：项目的具体实现手册

1. 形成详细设计文档：思路，逻辑流程，功能说明，技术点说明，数据结构说明，代码说明

编码测试：按照预定计划实现代码编写，并且做基本检测

1. 写代码
2. 写测试程序
3. 技术攻关

项目测试：对项目按照功能进行测试

1. 跨平台测试，使用测试
2. 完成测试报告

调试修改完善代码：

1. 根据测试报告进行代码修改

项目发布

1. 项目交付用户进行发布
2. 编写项目说明文档

**项目注意事项：**

1. 按时完成项目工作，和项目时间不足之间的冲突
2. 项目实施人员之间的冲突

项目工具的使用

编写文档：word ppt excel markdown LaTex

项目流程图：Mindmanager visio

项目管理：project

代码管理：Svn git

项目：电子词典

功能说明：

1. 用户能够登陆注册

登陆凭借用户名和密码即可

注册要求用户名和密码，用户名不能重复

1. 用户信息需要长期保存

使用Mysql或者mongo

1. 要求能够满足多用户同时登陆操作的情况
2. 用户开启客户端即进入登陆、注册、退出界面
3. 用户登陆后即进入用户的查词、查看历史记录、退出界面

查词：英英词典 可以循环查询，查询一个词会反馈给客户端词义

历史记录：查看自己曾经的查词记录，返回给用户查过哪些词、查询时间以及用户名

退出：退到上一级界面

1. 单词词义从单词本获取
2. 每个单词占一行
3. 单词和解释之间有空格
4. 单词按序排列

提示：

socket pymysql pymongo

服务器：注册 登陆 查词 历史记录

客户端：图形界面打印，提出请求，接受反馈，反馈展示

技术点：并发 数据库操作 数据表建立

**cookie**

import getpass

a = getpass.getpass()

功能：在输入时会自动有密码输入提示，而且加密输入

## Day42

pip python的标准第三方库管理工具

安装软件

pip3 install package

升级

pip3 install –upgrade package

卸载

pip3 uninstall package

查看软件包清单

pip3 list

查找软件包

pip3 search package

查看软件包信息

pip3 show package

记录python包软件环境

pip3 freeze > requirements.txt

根据环境文件进行环境安装

pip3 install –r requirements.txt

**协程**

并发编程：多进程 多线程 IO多路复用

定义：纤程、微线程，协程本质实际只有一个线程在运行

功能原理：通过应用层记录程序的上下文栈区，实现程序运行中的跳跃，进而实现选择代码段执行

优点：

1. 无需多线程切换的开销，资源消耗非常少
2. 无需进行同步互斥操作，多IO并发性极好

缺点：

1. 无法利用计算机多核资源
2. 如果程序阻塞会阻塞整个进程运行

yield =====》 实现python协程的基本原来

第三方库

greenlet evenless stackless gevent

gevent.spawn(func,argv)

功能：将func变为协程事件并启动

参数：func 事件函数 argv 多项，为func的参数

返回值：协程对象

gevent。join()

功能：回收协程

参数：要回收的协程对象

gevent.joinall()

功能：回收协程

参数：列表，里面可有多个协程对象

gevents.sleep()

功能：模拟IO阻塞的情况

参数：n 睡眠几秒

from gevent import monkey

monkey.patch\_all()

功能：在导入socket模块前使用 ，可将socket模块IO设置为非阻塞

## Day43

**前端** 前台 客户端 用户端

功能：

1. 和用户进行交互，获取用户请求
2. 和后端进行通信，整理用户请求个后端
3. 接受后端的内容，展现给用户

要求：

1. 良好的用户体验
2. 较全面的交互能力
3. 必要的优化
4. 良好的跨平台性能

**后端** 后台 服务端

功能：

1. 接受前端的请求
2. 进行逻辑处理和运算处理
3. 和磁盘进行交互（数据库 文件）
4. 进行数据的整理，更好的向前端返回结果

要求：

1. 良好的管理方案或者界面
2. 更好的并发性（并发方案）
3. 更快速的处理速度（良好的算法逻辑）
4. 更好的健壮性
5. 更好的可维护性和扩展性
6. 选择更加合适的技术搭配和框架

网站后端：

httpserver：

1. 接受http请求（socket通信）
2. 解析http请求（GET or POST）
3. 将具体的请求整理交给后端服务程序
4. 接受服务端返回的结果
5. 将结果发送给客户端response

后端服务程序：

1. 接受具体的请求内容
2. 逻辑处理请求（和数据库交互）
3. 将处理结果给httpserver

**cookie：**

1. 什么是框架

框架即开发模板，通过填写不同的内容可以完成快速的开发

1. \_\_call\_\_()的使用

在类中\_\_call\_\_()方法在实例对象使用()时会被自动调用执行，这使实例对象可以像函数一样使用并传参

例：class Calltest():

def \_\_call\_\_(self,a,b):

print('This is call test')

print(a,b)

test = Calltest()

test(1,2)

httpserver

技术点：

1. 使用并发效果，多进程或者多线程
2. 套接字 tcp套接字

后端框架：

1. 所有功能封装为类
2. 请求分为两种，一种是请求静态页面，一种是请求后端程序处理
3. 设置url列表来区分url

python的httpserver

python2 BaseHTTPServer

python3 http.server

**pdb调试**

通过pdb模块完成调试功能

功能：断点设置 单步调试 进入函数，查看当前代码段 查看变量值

break或b 设置断点

continue或c 继续执行

list或l 查看当前代码段

step或s 进入函数

exit或q 推出

next或n 执行下一行

pp 打印变量

help 帮助

pdb.set\_trace()

功能：设置初始断点，开始pdb调试模式

使用pdb模式运行程序

python3 –m pdb test.py

# Web部分

## Day44

赵旭

[zhaoxu@tedu.com.cn](mailto:zhaoxu@tedu.com.cn)

WEB

WEB Basic：HTML + CSS + JavaScript

Django：

**一、什么是web**

web就是互联网上的一种应用程序----网页

典型的应用：

1. c/s

client：客户端

server：服务端

1. b/s

browser：浏览器

server：服务器

**二、web的组成和运行流程**

由服务器，浏览器和通信协议组成

服务器：处理用户的请求（request）和响应（response）

浏览器：代替用户向服务器发送请求（User Agent）

通信协议：规范了数据是如何打包以及传递的----http

http：Hyper Text Transfer Protocal

超级 文本 传输 协议

**三、web服务器**

作用：接受用户的请求并给出响应，存储web信息，具备安全性功能

产品：APACHE TOMCAT IIS（internet information service） Nginx

技术：

1. JSP ---Java Server Page
2. PHP
3. ASP.NET
4. python web(Django , Flask,…..)

**四、web浏览器**

作用：

1. 代替用户向服务器发送请求
2. 作为响应数据的解释引擎

主流浏览器产品：

1. IE（Internet Explorer）
2. Google Chrome
3. Mozilla Firefox
4. Apple Safari
5. Opera Opera

浏览器靠内核处理数据，内核包含两部分：

1. 内容排版引擎 ----HTML CSS
2. 脚本解释引擎 ----JS

浏览器技术：

也称为“前端技术”

HTML CSS JavaScript

**HTML概述**

1. HTML介绍和基本语法

html：hyper text markup language

超级 文本 标记 语言

超文本：具备特殊功能的文本就是超文本

普通文本a：普通字符a

超文本a：表示的是超链接功能

普通文本b：普通字符b

超文本b：表示文字的加粗功能

标记：超文本的组成形式

普通文本a：a

超文本a：<a></a>

语言：语言有自己的语法规范

W3C：World Wide Web Consortium（万维网联盟）

1. HTML在计算机中的表现

所有的网页在计算机中都是以.html或.htm作为结尾的文件来进行表示的

开发工具：所有的文本编辑类软件都可以作为开发工具

1. 记事本
2. editplus, Sublime
3. Dreamweaver, WebStorm,…

运行工具：浏览器

推荐使用：google chrome

1. HTML基础语法（重点）

**标记的语法：**

1. 什么是标记：在网页中，用于表示功能的符号称为“标记/标签/元素”
2. 语法：所有的标记，在使用时必须用<>括起来，分为双标记和单标记

双标记：由开始标记和结束标记组成<标记>…</标记>

例：<a>…</a>

<div>…</div>

单标记：只有一个标记，既能表示开始，又能表示结束。单标记自己就是个独立的功能，没有文本或其他内容可以控制。

例：

<br>或<br/>：换行

<hr>或<hr/>：一条水平线

<img>或<img/>：显示图片

**标记的嵌套：**

在一对标记中，再出现另外一对标记，目的是为了实现功能的嵌套

例：加粗的超链接

<a><b>…</b></a> 或 <b><a>…</a></b>

注：为了良好的可读性，被嵌套的内容尽量独占一行，并添加缩进

<a>------父元素/父标记

<b>………</b>------子元素/子标记

</a>

**标记的属性和值**

1. 什么是属性

在标记中，用了修饰标记显示效果的东西就是属性

1. 语法
2. 属性的声明必须在开始标记中
3. 属性名和标记名之间要用空格隔开

<标记 属性名称></标记>

1. 属性名和属性值之间用=连接，属性值要用单引号或双引号引起来
2. 一个元素允许设置多个属性，多属性之间排名不分先后，但属性之间要用空格隔开

<标记 属性1=”值1” 属性2=”值2”></标记>

**HTML中的注释**

<!--注释内容-->

注：

1. 注释不能出现在<>中

<p<!--…-->>…</p>错误的

1. 注释不能嵌套

HTML中不区分大小写

<p></p> <P></P> <P></p>

1. HTML文档结构
2. 文档类型声明

出现在网页最顶端的第一个标记 <!doctype html>

作用：告诉浏览器使用HTML的哪个版本

1. HTML页面

在文档类型声明之下，使用一对html标记来表示网页的开始和结束。在html中，包含两对子元素

1.<head></head> 表示网页头部信息

2.<body></body> 表示网页主体信息

1. <head>标记

作用：描述网页的头部信息，对于网页起到控制的作用

子元素：

1.<title>标题内容</title>

2.指定网页编码 <meta charset=”utf-8”>,告诉浏览器用utf8的编码方式进行网页解析

注：要确保网页文件的编码方式也是utf8

1. 文本相关标记
2. HTML中特殊字符处理

1）&nbsp; 表示一个空格

2）&lt; 表示一个<

3）&gt; 表示一个>

4）&copy; 表示一个©

5）&yen; 表示￥

1. 文本样式标记

作用：修改文本在网页中的表现形式

标记：

1.<i></i>：斜体显示文本

2.<u></u>：下划线显示文本

3.<s></s>：删除线显示文本

4.<b></b>：加粗显示文本

5.<sup></sup>：上标方式显示文本

6.<sub></sub>：下标方式显示文本

特点：该组标记能够与其他的标记或文本在一行内显示

1. 标题标记

作用：以不同的文字大小以及加粗的方式显示文本

语法：<h#></h#> #：表示1—6的整数

特点：

1. 会改变文字的大小以及加粗效果
2. 每个标题都会具备垂直的空白距离
3. 每个标题都独占一行
4. 每个标题都会具备一个属性

属性：align

取值：

1. left：左对齐
2. center：居中
3. right：右对齐
4. 段落元素

作用：突出显示一段文本，每段文本独占一行/块，并且每个段落都会具备一小段的垂直空白距离

语法：<p></p>

属性：align

取值：left/center/right

**注意：p元素内不能嵌套块级元素**

1. 换行元素

<br>或<br/>

1. 分区元素
2. 块分区元素

标记：<div></div>

作用：布局（配合CSS）

特点：独占一行/一块

1. 行内分区元素

标记：<span></span>

作用：设置同一行文本的不同样式（配合CSS）

特点：允许在一行内显示多个span元素，也能够与其他的文本在一行内显示

1. 行内元素与块元素
2. 块元素

只要在网页中能独占一行/一块的元素都称为块级元素，简称为块元素

p,h1,h2,h3,h4,h5,h6,div

作用：都可以做布局

所有的块元素都会具备align属性

1. 行内元素

多个元素能够在一行内显示的，就是行内元素

span,i,b,s,u,sub,sup

1. 列表标记

**作用：**

按照从上到下的方式来进行数据排列，并能够显示列表的标识在内容的前面

**列表的组成：**

1. 列表的类型
2. 有序列表---<ol></ol>（Order List）
3. 无序列表---<ul><ul>（Unordered List）
4. 列表项

<li></li>（List Item）

例：

<ol>

<li>内容1</li>

<li>内容2</li>

</ol>

**列表的属性：**

1. 有序列表---ol
2. type

取值：

1. 1：按数字方式排列显示
2. A：按大写英文字符显示
3. a：按小写英文字符显示
4. I：按大写罗马数字显示
5. i：按小写罗马数字显示
6. start

指定有序列表从几开始

1. 无序列表---ul

type

取值：

1. disc：实心圆点，默认
2. circle：空心圆点
3. square：实心方块
4. none：不显示任何字符

**列表的嵌套：**

在一个列表项中，又出现一个列表

## Day45

1. **图像 和 超链接**
2. **URL（Uniform Resource Locator）**

统一资源定位器，用于标识网络中资源的位置，俗称路径

URL分类：

1. 绝对路径

访问网路资源时，使用绝对路径

例：http://[www.biadu.com](http://www.biadu.com)

http://www.baidu.com/img/bd\_logo1.png

1. 相对路径

从当前文件所在的位置处去开始查找资源文件所经过的路径

1. 根相对路径

从web程序所在的根目录处开始查找资源文件

**注：1.url中不能出现中文 2.url是严格区分大小写**

1. **图像**
2. 标记：<img>
3. 属性
4. src 指定要显示的图片的路径
5. width 宽度，以px为单位的数值（允许省略px）
6. height 高度，以px为单位的数值（允许省略px）

注：如果宽度和高度值设定一个值的话，那么另外一个值也跟着等比缩放

1. **超链接**
2. 什么是超链接

用户可以通过点击的操作来完成页面的跳转行为

1. 语法

标记：<a>内容</a>

属性：

1. href 表示条跳转的链接地址
2. target 指定要打开新网页的方式

取值：

1）\_self 默认值，在自身标签页中打开新网页

2）\_blank 在新的标签页中打开新网页

1. **表格**
2. **表格的语法**
3. 标记

表格：<table></table>

表行：<tr></tr>-------Table Row

单元格（列）：<td></td> ----Table Date

1. **属性**
2. table属性
3. width 指定表格的宽度，以px为单位数值（px可以省略）
4. height 指定表格的宽度，以px为单位数值（px可以省略）
5. border 指定边框宽度，以px为单位数值（px可以省略）
6. align 指定表格在其父元素中的水平对齐方式（left/center/right）
7. cellpadding 指定单元格内边距（内容与边框之间的距离）
8. cellspacing 指定单元格外边距（单元格之间的距离）
9. tr属性
10. align 控制当前行内容的水平对齐方式（left/center/right）
11. valign 控制当前行内容的垂直对齐方式（top/middle/bottom）
12. bgcolor 控制当前行内容的背景颜色（颜色的英文单词）
13. td属性
14. width
15. height
16. align
17. valign
18. bgcolor
19. colspan 跨列/合并列
20. rowspan 跨行/合并行
21. **不规则表格**
22. 单元格的跨列

从指定单元格的位置处开始，横向向右合并几个单元格（包含自己），被合并掉的单元格要删除

语法：colspan=’n’

1. 单元格的跨行

从指定单元格的位置处开始，纵向向下合并几个单元格（包含自己），被合并掉的单元格要删除

语法：rowspan=’n’

1. **行分组**
2. 什么是行分组

允许将表格中的若干行划分到一组中，方便管理

1. 语法
2. 表头行分组

允许将表格中最上方的若干行划分到一组中

<thead></thead>

1. 表尾行分组

允许将表格中最下方的若干行划分到一组中

<tfoot></tfoot>

1. 表主体行分组

表格中除了最上方和最下方之外的行们进行分组的话允许放在表主体行分行组中

<tbody></tbody>

1. **表单**
2. **作用**

用于接受用户的数据并提交给服务器

表单二要素：

1. form元素

表单，用于收集用户信息并提交给服务器

1. 表单控件

提供能够与用户交互的可视化组件

1. **form元素**
2. 作用

用于收集用户信息并提交给服务器

form元素在网页中是不可见的，但功能不能忽略

1. 语法
2. 标记

<form></form>

1. 属性
2. action 指定处理程序的地址，默认提交给本页
3. method 提交方式/方法，共提供了7-8个值，但常用只有2个
4. get（默认值）：通常是在向服务器索要数据时使用

特点：

\*提交的数据会显示在地址栏上

\*安全性较低

\*提交数据最大为2KB

1. post：要将数据提交给服务器处理时使用

特点：

\*隐式提交，看不到提交数据

\*安全性较高

\*无提交数据的大小限制

1. **表单控件（重难点）**
2. 作用 & 注意

作用：提供了能够与用户交互的可视化组件

注意：

\*只有放在表单中的表单控件才被允许被提交

\*表单控件都是“行内块”元素

1. 表单控件详解（难点）
2. 文本框&密码框

文本框：<input type=’text’>

密码框：<input type=’password’>

属性：

1. name：定义控件名称

提交给服务器使用，如果没有name的话则无法提交

1. value：值

要提交给服务器的值，同时也是默认能够显示在控件上的值

1. maxlength

限制输入的最大字符数

1. placeholder：占位符

用户在未输入任何数据时所显示的内容

1. 按钮

1）提交按钮：<input type=’submit’>

将表单数据提交给服务器

2）重置按钮：<input type=’reset’>

将表单内容恢复到初始化的状态

3）普通按钮：<input type=’button’>

允许通过js自定义功能

属性：value 按钮上显示的文本

4）<button>内容</button>

1. 单选按钮 和 复选框

单选按钮：<input type=’radio’>

复选框：<input type=’checkbox’>

属性：

1. name

定义控件名称，除了定义名称之外，还起到分组的作用，一组中的单选按钮或复选框名称必须一致

1. value

值，尽量提前声明

1. checked

设置预选中，该属性无值

1. 隐藏域和文件选择框
2. 隐藏域

作用：想提交给服务器但不想给用户看的数据可以放在隐藏域中

语法：<input type = ‘hidden’>

属性：

name：定义控件的名称

value：定义控件的值

1. 文件选择框

作用：文件上传时使用

标记：<input type=’file’>

属性：

name：定义控件的名称

1. 多行文本域

标记：<textarea></textarea>

属性：

1. name：控件名称
2. cols：指定文本域默认显示的列数

注：一行能够显示的英文数量，中文减半

1. rows：指定文本域默认显示的行数
2. 下拉选择框（下拉列表）

<select name=’xxx’>

<option value=’值1’>显示文本内容</option>

<option value=’值2’>显示文本内容</option>

</select>

## Day46

1. **css介绍**

css：Cascading Style sheets

html：搭建网页结构

css：修饰和美化网页

1. **css的使用方式（重点）**
2. 内联方式

又称为：行内样式、内联样式

特点：将css的内容定义在单独的html的标签中

语法：

<标记 style=’样式声明’>

样式声明：就是要修饰的样式效果

1. 样式的声明是由样式属性和属性值来组成的
2. 属性和值之间使用冒号连接

<标记 style=’属性:值’>

1. 在一个style中允许出现多个样式声明，多个样式声明之间用分号隔开

**<标记 style=’属性:值;属性:值;’>**

常用的属性和值：

1. 文字大小

属性：font-size

取值：以px为单位的数值

ex：设置某div的文字大小为18px

<div style=’font-size:18px;’>xxx</div>

1. 文本颜色

属性：color

取值：取值为表示颜色的英文单词

ex：设置某div的文字颜色为红色（red）

<div style=’color:red;’>xxx</div>

1. 背景颜色

属性：background-color

取值：取值为表示颜色的英文单词

1. 内部样式表

作用：让定义好的样式能够使用在当前页面的多个元素上

语法：

**<head>**

**<style>**

**样式规则1**

**样式规则2**

**…..**

**样式规则n**

**</style>**

**</head>**

样式规则：

由 选择器 和 样式声明 组成的

目的：为了声明一组独立的样式

选择器：规范了页面中哪些元素能够使用声明好的样式

**选择器{**

**属性1:值1;**

**属性2:值2;**

**}**

1. 外部样式表

作用：将声明好的样式应用在多个网页中

用法：将样式规则声明在独立的css文件中（\*\*.css），在使用的网页中对css文件进行应用即可

使用步骤：

1. 声明

创建.css文集，并编写样式规则

1. 引用

<head>

<link rel=’stylesheet’ href=’.css文件路径’>

</head>

1. **样式表的特征**
2. 继承性

大部分的css属性是可以由父元素继承给子元素的

1. 层叠性

允许为一个元素定义多种的使用方式或多个样式规则

如果样式的声明之间不冲突，那么所有的样式声明都可以应用在元素上

1. 优先级

如果样式声明冲突，会按照不同方式的优先级来应用样式

优先级：

浏览器缺省设置 低

内部或外部样式表 中（就近原则：后定义者优先）

内联方式 高

1. **简易挑错**
2. Invalid property value

表示属性值错误

1. unknown property name

表示属性名称错误

1. **css选择器（重点）**
2. 作用：

规范了页面中哪些元素能够使用声明好的样式

目的：为了匹配页面的元素

1. 选择器的详解
2. **元素选择器**

特点：由标记名称作为选择器，主要匹配页面中指定标记所对应的所有元素

语法：

标记{

样式声明;

}

1. **类选择器**

特点：允许被任意元素所引用的选择器

语法：

1. 声明

.类名{

样式声明;

}

类名：字母，数字，下划线以及连接线(-)组成，不能数字开头

1. 引用

<标记 class=’类名’>（引用不加点）

特殊用法：

1. 分类选择器的定义方式

允许将元素选择器和类选择器结合到一起使用，为了实现对某种元素不同样式的细分控制

语法：

元素选择器.类选择器{样式声明;}

1. 多类选择器的引用方式

允许让一个元素同时引用多个类选择器，多个类选择器之间使用空格隔开

<标记 class=’c1 c2 c3’>

1. **id选择器**
2. ID的作用

在html中，每个元素都允许设置一个id属性，主要用于表示该元素在网页中独一无二的标识

<div id=’id名’></div>

1. id选择器

为了匹配页面中指定ID值的元素

语法：

#ID值{…}

ex:匹配id为main的文本设置为红色

#mian{color:red;}

1. **群主选择器**
2. 作用

定义多个选择器们的共有样式，定义方式是一个以 逗号 隔开的选择器列表

1. 语法：

选择器1，选择器2，…{… ….}

1. **后代选择器**
2. 作用

依托于元素的后代关系来匹配元素（不限制层级）

1. 语法

选择器1 选择器2{… ….}

ex: 匹配id为wang内所有span元素

#wang span{…}

1. **子代选择器**
2. 作用

依托于元素的子代关系来匹配元素（只有一层层级关系）

1. 语法

选择器1>选择器2{…}

ex: id为wang的下一级span元素

#wang>span{..}

1. **伪类选择器**
2. 作用

匹配元素不同状态的选择器

ex：超链接a 元素 有四种状态

状态1：链接未被访问时的状态

文本为蓝色，并有下划线

状态2：鼠标悬停在元素上的状态

鼠标的状态变成‘手’

状态3：当元素被激活时（鼠标点击时）

文本变为红色

状态4：当访问过后的状态

文本变为紫色

1. 语法

:伪类状态

通常会配合其他元素一起使用

选择器:伪类状态{… …}

1. 链接伪类

1）:link 匹配超链接未被访问时的状态

2）:visited 匹配超链接被访问后的状态

1. 动态伪类

1）:hover 匹配鼠标悬停时的状态

2）:active 匹配元素被激活时的状态

3）:focus 匹配获取焦点时的状态（文本框和密码框使用居多）

1. 选择器的优先级

当多个选择器能够同时应用到一个元素上时，并且样式发生冲突的话，则要按照不同选择器的优先级来应用样式

选择器的优先级看“权值”

**选择器取 值**

**元素选择器 1**

**类/伪类选择器 10**

**id选择器 100**

**内联方式 1000**

如果是复杂的选择器（后代，子代，分类）的计算方式是将选择器们的权值累加

1. **尺寸 与 边框**
2. 单位
3. 尺寸单位
4. px---像素
5. %---占据父元素对应属性的占比
6. in---英寸 1in=2.54cm
7. pt---磅 1pt=1/72in(在css中通常表示文字大小)

计算机中通常设置PPI（每英寸内的像素个数）为72

1. cm---厘米
2. mm---毫米

注：在css中，所有尺寸单位是不能省略的

1. 颜色单位（颜色取值）
2. rgb（r，g，b）

r：红色范围值，0-255

g：绿色范围值，0-255

b：蓝色范围值，0-255

ex：color:rgb(255,0,0)

1. rgba(a，g，b，alpha)

alpha：颜色透明度，取值0-1之间的数字

0：完全透明

1：完全不透明

1. #rrggbb

由6位16进制数字来组成颜色

ex：color:#ff0000

1. #rgb

#rrggbb的缩写，当每两位数字相同时，可以使用缩写的方式

#ff0000---#f00

#333---#333333

1. 英文单词

## Day47

1. **尺寸和边框**
2. 尺寸属性
3. 宽度

属性：width

取值：以px或%位单位的数值

1. 高度

属性：height

取值：以px或%位单位的数值

所有块级元素的尺寸，宽度占父元素100%的宽，高度以内容为准

所有行内元素的尺寸，宽度和高度都是以内容为准

**注：html 和 css中，除了img以外的所有行内元素的尺寸是不允许修改的**

1. 溢出处理

溢出处理的属性

属性：overflow

取值：

1. visable 溢出可见，默认值
2. hidden 溢出的内容会隐藏
3. scroll 显示滚动条，溢出时滚动条可用
4. auto 自动，溢出时产生滚动条并可用
5. 边框属性
6. 边框实现
7. 边框的简写方式

通过一个属性完成四个方向边框的所有效果设置（宽度、样式、颜色）

属性：border0

取值：width style color

width：边框的宽度，以px为单位的数值

style：边框的样式

solid：实线

dotted：虚线（点状）

dashed：虚线（线状）

color：边框的颜色

取值为合法的颜色值

可以取值为transparent(透明)

特殊用法：

boder：none；

1. 单边定义

定义某一条边框的宽度，样式，颜色

属性：border-方向：width style color

方向：top/bottom/left/right

1. 单属性定义

设置四个方向边框的某一个属性值

语法：border-属性：值；

属性：width/style/color

1. 单边单属性定义

设置某一方向边框的某一属性值

语法：border-方向-属性：值；

1. 轮廓

outline：none;

1. 边框倒角
2. 作用

将边框的四个直角变为圆角

1. 语法

属性：border-radius

取值：

1. 以px为单位的数值
2. 以%为单位的数值
3. 边框阴影

属性：box-shadow：h-shadow v-shadow blur spread color;

取值：

h-shadow：阴影的水平偏移距离，正数，阴影向右偏移，负数，阴影向左

v-shadow：阴影的垂直偏移距离，正数，阴影向下偏移，负数，阴影向上

blur：阴影的模糊大小，数值越大越模糊

spread：阴影的大小，取值为数值

color：阴影的颜色

1. **框模型 --- Box Model**
2. 什么是框模型

框：页面元素皆为框

框模型：定义元素的尺寸和距离的一种计算方式

Box Model：盒模型，方框属性

包含属性：尺寸，边框，外边距和内边距

当框模型属性介入到元素时，元素的整体占地尺寸会发生改变，计算方式如下：

**占地宽度=左右外边框+左右边框+左右内边距+宽**

**占地高度=上下外边框+上下边框+上下内边距+高**

1. 外边距

什么是外边距

围绕在元素边缘之外的空白距离就是外边距，外边距是不允许被其他元素占据的

语法：

1. margin

最多能够设置四个方向的外边距值

1. margin-top 上外边距的值
2. margin-right 右外边距的值
3. margin-bottom 下外边距的值
4. margin-left 左外边距的值

取值：

1. 以px为单位的数值
2. 以%为单位的数值

以父元素尺寸的占比作为外边距的值

1. 取值为负数

目的是为了移动元素

元素设置上外边距为正数，元素下移

元素设置上外边距为负数，元素上移

元素设置左外边距为正数，元素右移

元素设置左外边距为负数，元素左移

1. 取值为auto

自动计算左右外边距的值

注意：

1. auto只能应用在左右外边距上，上下无效
2. 只能为设置宽度的块级元素设置左右外边距为auto，目的是为了让块级元素水平居中

margin的简介写法

1. margin：value；

value表示的是上下左右四个方向外边距值

1. margin：v1 v2；

v1：表示上下外边距的值

v2：表示左右外边距的值

1. margin：v1 v2 v3；

v1：表示上外边距的值

v2：左右外边距的值

v3：下外边距的值

1. margin：v1 v2 v3 v4;

v1：上外边距

v2：右外边距

v3：下外边距

v4：左外边距

页面中具备外边距的元素：

body p h1~h6 ul ol

css重写：

通过元素选择器

1. 内边距
2. 什么是内边距

内容与元素边缘之间的距离

注意：内边距会扩大元素边框的所占区域

1. 语法

属性：

padding:四个方向的内边距值

padding-top/right/bottom/left：值

取值：

1. 以px为单位的数值
2. 以%为单位的数值

简洁写法：同margin

padding：value；上下左右

padding：v1 v2；上下 左右

padding：v1 v2 v3 上 左右 下

padding：v1 v2 v3 v4 上 右 下 左

1. 页面中具备默认内边距的元素
2. ul，ol
3. 文本框，密码框，按钮
4. box-sizing

作用：

指定框模型（尺寸，边框，内边距）的计算方式

语法：

属性：box-sizing

取值：

1. content-box

默认值，元素的width和height，只规定到元素的内容区域的宽和高。内边距和边框的尺寸是需要额外计算在加到当前元素上。

1. border-box

元素的width和height，能够规定到元素边框，内边距以及内容尺寸。

width=边框宽度+内边距+实际内容的宽度

height=边框高度+内边距+实际内容的高度

1. **背景属性**
2. 背景属性
3. 背景颜色

属性：background-color

取值：合法的颜色值

注意：背景颜色电话是从边框的位置处就开始绘制

1. 背景图像

属性：background-image

取值：url（’图片路径’）

1. 背景平铺

属性：background-repeat

取值：

1. repeat 默认值，横纵都平铺
2. no-repeat 不平铺
3. repeat-x 横向平铺
4. repeat-y 纵向平铺
5. 背景图片的尺寸

属性：background-size

取值：

1. width height 以px为单位的数值
2. width% height%

采用当前元素的尺寸占比作为背景图片尺寸

1. 背景图片的位置

改变背景图片在元素中的默认位置

属性：background-position

取值：

1）x y

以px为单位的数值，用空格隔开

x：背景图像水平偏移距离，值为正，向右偏移，负，向左偏移

y：背景图像上下偏移距离，值为正，向下偏移，负，向上偏移

2）x% y%

0% 0% 背景图在左上角

100% 100% 背景图在右下角

50% 50% 背景图在中间

3）关键字

x：left/center/right

y：top/center/bottom

1. 背景属性---简写方式

属性：background

取值：color url() repeat position；

ex：

background：red；

background：url(a.jpg) no-repeat 90% center;

## Day48

1. **文本格式化属性**
2. 字体属性
3. 指定字体

属性：font-family

取值：由逗号（，）隔开的字体列表

注意：如果字体中包含中文或空格的话，要用引号引起来

ex：

font-family：“微软雅黑”；

font-family：“微软雅黑”，“Arial”，“Helvetica”；

1. 指定字体大小

属性：font-size

取值：px或pt

1. 字体加粗

属性：font-weight

取值：

a．normal：非加粗显示

b．bold： 加粗显示

c．无单位的数值（400等同于normal 900等同于bold）

1. 字体样式

属性：font-style

取值：

normal：非斜体显示

italic：斜体显示

1. 字体属性（简写）

属性：font

取值：style weight size family

注意：

使用简写方式时，必须要设置family的值，否则无效

font：12px；=====》错误

font：12px “宋体”；

1. 文本属性
2. 文本颜色

属性：color

取值：合法的颜色值

1. 文本的排序方式

作用：控制某元素内的文本，图片和行内块元素的排列方式

属性：text-align

取值：left/center/right/justify（两端对齐）

1. 文字修饰

作用：指定线条修饰效果

属性：text-decoration

取值：

1. none：无任何线条显示
2. underline：下划线
3. overline：上划线
4. line-through：删除线
5. 行高

作用：指定一行文本数据的所占高度

特点：如果行高的高度大于文本的高度，那么文本内容将垂直居中行高内

使用场合：

1. 文本垂直居中
2. 行间距

属性：line-height

取值：

1. 以px为单位的数值
2. 无单位的数字，表示的是当前字体大小的倍数
3. **表格属性**
4. 表格属性
5. 表格的常用属性
6. 尺寸属性
7. 边框属性
8. 文本格式化
9. 背景属性
10. 框模型（margin不能用在表格td中）
11. 表格的特有属性
12. 表格合并

属性：**border-collapse**

取值：

1. separate

默认值，分离边框（双线边框）模式

1. collapse：边框合并
2. 边框边距

作用：设置每两个单元格之间的距离

属性：**border-spacing**

取值：

1. 指定一个数值

水平和垂直间距相等

1. 给定两个数值

第一个值：表示水平间距

第二个值：表示垂直间距

两个值之间用空格隔开

**注意：**

**必须要在分离边框模式下使用**

1. **过度效果---transition**
2. 什么是过渡

使得CSS的属性值在一段时间内平缓变化的一个效果

1. 语法
2. 指定过渡属性

作用：指定哪个属性值在变化的时候使用过渡效果

属性：**transition-property**

取值：

1. 属性名称
2. all 但凡能使用过渡效果的属性值在变化时一律都使用过渡来体现

允许使用过度效果的属性：

* 所有与颜色相关的属性
* 所有取值为数字的属性

1. 指定过渡时长

作用：指定在多长时间内完成过渡效果

属性：**transition-duration**

取值：以s或ms为单位的数值

1s=1000ms

300ms=0.3s=.3s

1. 指定过渡的速度时间曲线函数

作用：指定过渡效果变化速率

属性：**transition-timing-function**

取值：

1. ease

默认值，慢速开始，快速变快，慢速结束

1. linear

匀速

1. ease-in

慢速开始，加速结束

1. ease-out

快速开始，减速结束

1. ease-in-out

慢速开始和结束，中间先加速后减速

1. 指定过渡延迟

作用：当激发过渡效果后，等待多长时间再开始执行操作

属性：**transition-delay**

取值：以s或ms为单位的数值

1. 过渡属性：

属性：transition

取值：property duration timing-fun delay

ex：

transition：color 5s linear，backgroound 3s；（多个属性过渡）

1. 定位（重难点）
2. 什么是定位

表示的是元素在网页中的位置

1. 定位的分类

在CSS中，定位主要分为以下几类：

1. 普通流定位（默认定位方式）
2. 浮动定位（重难点）
3. 相对定位
4. 绝对定位
5. 固定定位
6. 普通流定位

普通流定位，又称为“文档流定位”。是页面中默认的定位方式。典型的“流式布局”。

特点：

1. 每个元素在页面中都有自己的位置，并占据一定的空间
2. 每个元素都是从父元素的左上角开始排列的
3. 每个元素基本上都是按照从左到右，从上到下的方式排列的

块级元素：从上到下，每个独占一行

行内元素&行内块元素：从左到右，排列不下时换行

1. 浮动定位
2. 浮动元素的特点

将元素设置为浮动定位，元素将具备以下特点：

* 浮动元素会被排除在文档流之外，脱离文档流后，元素将不再占据页面空间
* 剩余未浮动元素会上前占位
* 浮动定位的元素会停靠在父元素的左边或右边或其他已浮动元素的边缘上
* 浮动只能在当前行浮动
* 浮动解决的问题：让多个块级元素在一行内显示的问题

1. 语法

属性：**float**

取值：

* none 默认值，即无任何浮动效果
* left

浮动到父元素的左边，或停靠在左边已有的浮动元素的边缘上

* right

浮动到父元素的右边，或停靠在右边已有的浮动元素的边缘上

1. 浮动引发的特殊效果

* 如果父元素显示不下所有已浮动子元素的话，那么最后一个将换行，但有可能会被卡住
* 元素一旦浮动起来之后，就将变成块级元素

行内元素一旦浮动，就允许修改尺寸

* 元素一旦浮动起来之后，在未指定宽度的情况下，宽度将由内容来决定，主要针对块级元素
* 文字，图片，行内元素是采用环绕的方式来排列的，是不会被浮动元素压在底下的

1. 清除浮动

元素一旦浮动起来之后，后面的元素要上前占位，有可能被浮动元素压在底下。如果元素不想被压在底下的话，则可以通过清除浮动影响的方式来解决问题

1. 语法：

属性：clear

取值：

1. none 默认值，不做任何清除浮动的操作

2. left

清除当前元素前面元素左浮动所带来的影响，即不会被前面元素左浮动而压在底下

3. right

清除当前元素前面元素右浮动所带来的影响，即不会被前面元素右浮动而压在底下

4. both

清除当前元素前面元素任何一种浮动方式

1. 浮动元素对父元素高度的影响

由于浮动元素会脱离文档流，所以是不占据页面空间的，那么也就不占父元素的空间。那么父元素的高度是已未浮动的子元素的高度为准的。解决父元素的高度：

* + - 为父元素设置高度

弊端：不一定每次都清楚的知道父元素的高度

* + - 设置父元素也浮动

弊端：会对父元素后面的元素产生位置的影响

* + - 设置父元素的overflow为hidden或auto

弊端：如果有溢出要显示的元素，也一同被隐藏了

* + - 在父元素中追加空块级元素，并设置其clear属性为both

## Day49

1. 其他定位
2. 相关属性
3. 定位方式

属性：position

取值：

1. static 静态的，默认值
2. relative 相对定位
3. absolute 绝对定位
4. fixed 固定定位

注意：将元素的position设置为relative/ absolute/ fixed任意一种的话，那么该元素就称为“已定位元素”

1. 偏移属性（共4个）

作用：配合已定位元素实现位置的移动

属性：top/right/bottom/left

取值：以px为单位的数值

top：以元素的上边为基准边移动元素（值为正，元素下移；值为负，元素上移）

right：以元素的右边为基准边移动元素（值为正，元素左移；值为负，元素右移）

bottom：以元素的下边为基准边移动元素（值为正，元素上移；值为负，元素下移）

left：以元素的左边为基准边移动元素（值为正，元素右移；值为负，元素左移）

1. 定位方式----相对定位
2. 什么是相对定位

元素会相对原来的位置偏移某个距离

1. 场合

元素位置微调时使用相对定位

1. 语法

属性：position

取值：relative

配合偏移属性实现位置的移动

1. 定位方式----绝对定位（难点）
2. 什么是绝对定位&特点

* 绝对定位的元素会脱离文档流---不占页面空间
* 绝对定位的元素会相对于离他最近的，已定位的祖先元素去实现位置的初始化
* 如果元素没有已定位的祖先元素，那么元素将相对于body去实现位置的初始化

1. 语法

属性：position

取值：absolute

配合偏移属性实现位置的移动

1. 注意

绝对定位的元素会变为块级元素（可以直接修改尺寸）

1. 堆叠顺序
2. 什么是堆叠顺序

已定位元素们堆叠在一起的排序顺序

1. 语法

属性：z-index

取值：无单位的数字，数字越大元素越靠上，默认值为0

1. 注意：

* 如果堆叠顺序相同的话，则后来者居上
* 只有已定位的元素才能使用z-index relative/ absolute/ fixed
* 父子元素之间，永远都是父子

1. 定位方法---固定定位
2. 什么是固定定位

让元素固定在浏览器窗口的某个位置处，不会随着滚动条的滚动而发生位置的改变

1. 语法

属性：position

取值：fixed

配合着偏移属性实现元素位置的固定

1. 注意

* 固定定位的元素永远都是相对于body实现位置初始化的
* 固定定位的元素也会变成块级元素

1. 显示
2. 显示方式
3. 什么是显示方式

决定了元素是以什么样的方式显示在网页中（块级/行内/行内块）

1. 语法

属于：display

取值：

* none

让元素不显示----隐藏

脱离文档流---不占据页面空间

* block

将元素变为块级元素

* inline

将元素变为行内元素

* inline-block

将元素变为行内块元素

特点：

1. 多个元素能够在一行内显示
2. 允许修改尺寸
3. 显示效果
4. 可见性属性

属性：visibility

取值：

* **visible**

默认值，可见的

* **hidden**

隐藏的，未脱离文档流，所以还占据着页面空间

1. 透明度属性

属性：opacity

取值：0（完全透明）--1（完全不透明）

1. 垂直方向对齐方式

属性：vertical-align

取值：top/middle/bottom/baseline

* **放在td中**，相当于是valign的作用，取值top/middle/bottom，middle是默认值
* **放在图片和行内块元素中**，设置元素两端的文本相对于元素的垂直对齐方式，取值top/middle/bottom/baseline，baseline是默认值

1. 光标

作用：改变鼠标悬停在元素上时鼠标的样子

属性：cursor

取值：

1. default
2. pointer----小手
3. text-----文本插入光标
4. crosshair ------ +
5. wait----等待
6. help----帮助

## Day50

1. 列表
2. 列表特征
3. 上下外边距
4. 左内边距
5. 列表项标识
6. 纵向排序
7. 列表属性
8. list-style-type

取值：

* none 不显示任何标识
* disc 实心小圆点
* circle 空心小圆点
* square 实心方块

1. list-style

作用：列表的简写属性

常用用法：

list-style：none；

1. 转换属性 ---- transform
2. 什么是转换

改变元素在页面中的位置，尺寸，角度的一种方式

1. 属性
2. 转换属性

属性：transform

取值：

* none：默认值，无任何转换效果
* 转换函数

如果有多个转换函数的话，中间要用空格隔开

1. 转换原点

转换原点就是转换操作所围绕的一个点

语法：

属性：transform-origin

取值：使用空格隔开的两个值

1. 以px为单位的数值
2. 以%为单位的数值
3. 关键字（top/bottom/center/left/right）
4. 转换效果
5. 位移
6. 作用

改变元素在页面中的位置

1. 语法

属性：transform

取值（函数）：

* translate X（x）

x表示元素在x轴上的位移距离（横向）

x取值为正，元素右移

x取值为负，元素左移

* translateY（y）

x表示元素在x轴上的位移距离（横向）

y取值为正，元素下移

y取值为负，元素上移

* translate（x）

等同于 translateX（x）

* translate（x,y）

同时在x轴和y轴上作位移

1. 缩放
2. 作用

改变元素在页面中的大小

1. 语法

属性：transform

取值（函数）：

 scaleX（x）

x表示横向缩放比例

x默认值1，原始大小

大于1的数值：放大的比例

大于0小于1的数值：缩小的比例

小于0：物极必反

 scaleY（y）

y表示纵向缩放比例

y默认值1，原始大小

大于1的数值：放大的比例

大于0小于1的数值：缩小的比例

小于0：物极必反

 scale(value)

value表示x轴和y轴的缩放比例是相同的

1. 旋转
2. 作用

改变元素在网页中的角度

1. 语法

属性：transform

取值（函数）：

* rotate（ndeg）

n为旋转角度

n取值为正，顺时针旋转

n取值为负，逆时针旋转

1. 注意

* 转换原点会影响转换效果
* 旋转操作时，会连同坐标轴一起旋转

**JavaScript部分**

1. JavaScript概述
2. 什么是JavaScript

JavaScript简称“JS”，是一种运行于JS解释器或执行引擎中的解释性脚本语言

1. JS发展史：

1992年Nombas公司开发了一款脚本语言，能够运行在网页中，名称CMM（c--），更名为ScriptEase

1995年Netscape（网景）为自己的浏览器Navigator2.0开发了另一款脚本语言LiveScript，后更名为JavaScript

1996年微软为自己的浏览器IE3.0开发了一款JavaScript的克隆版本JScript

1997年Netscape找到了ECMA（欧洲计算机制造商联合会）。Netscape将JS的核心交给了ECMA，从此JS的核心被更名为ECMA Script，简称ES

1. 完整的JS组成：
2. 核心----ECMScript

包含了JS中最基本的语法规范

1. 浏览器对象模型---BOM模型

browser object model

提供了一系列的可操作方法和属性用于和浏览器进行交互

1. 文档对象模型---DOM模型

Document Object Model

提供了一系列的可操作方法和属性用于和文档进行交互

1. JS的使用方式
2. 在元素事件中编写JS代码

事件：用户在元素上所激发的行为操作

onclick：当用户单击元素时激发的操作

语法：

<标记 onclick=’JS执行代码’>

console.log(‘向控制台中输出一句话’);

1. 将代码嵌入在网页的<script></script>里

语法：

在网页的任意位置处，均可以嵌入

特点：

网页加载时就执行

document.write(“向网页中输出的一句话”)

注意：将document.write编写在按钮中的话，在执行时，会刷新网页中的内容

1. 将JS代码编写在外部的JS文件中
2. 创建一个js文件（xxx.js），编写JS代码
3. 在使用的网页中对js文件引入

<script src=’js文件路径’></script>

注意：在引入的JS标记中时不能编写JS脚本的

1. JS的基础语法
2. JS都是由语句来组成的
3. 由运算符，关键字和表达式构成的
4. JS中严格区分大小写
5. 每条语句必须以分号(；)结束
6. 所有的标识符，符号必须都是英文的
7. JS中的注释

单行：//

多行：/\* \*/

1. JS中的变量和常量
2. 变量
3. 声明变量

声明：var 变量名；

赋值：变量名=值；

声明并赋值：var 变量名=值；

例如：var name=’老王’；

console.log(name);

document.write(name);

window.alert(name);

1. 注意：

* 在声明变量时，尽量使用var关键字，如果省略var的话，也可以，但容易出问题
* 声明变量如果未赋值的话，默认值未undefined

1. 变量名的命名规范

* 不能是JS中的关键字和保留关键字
* 由字母，数字，下划线（\_）和$；

var $=35; 可以

* 数字不能开头
* 尽量不要重复
* 尽量要见名知意
* 如无特殊需求，尽量使用小驼峰命名法

var userName;

1. 变量的使用方式

* 为变量赋值

只要变量出现在赋值（=）符号的左边，一律都是赋值操作

* 获取变量的值

只要变量没出现在赋值符号的左边，一律都是取值操作

1. 常量
2. 什么是常量

一经声明就不允许被修改的数据就是常量

1. 语法

const 常量名=值；

注意：

* 常量声明好之后是不允许被修改的，所以一定要赋值
* 常量名一般采用全大写形式

1. 保留小数点n位

**l=l.toFixed(n)**

## Day51

1. 数据类型
2. 什么是数据类型

数据的类型，约束了数据在内存中所占的空间大小

1. JS的数据类型分类
2. 基本数据类型（原始类型）

* number类型

数字类型，可以表示32位（4字节）的整数或64位（8字节）的浮点数

整数：

允许表示十进制，八进制和十六进制

十进制：var num=123;

八进制：var num=011;(十进制9)

十六进制：var num=0x1A;(26)

小数：

小数计数法：var num=12.34；

指数计数法：var num=1.5e3；（1.5\*10\*\*3）

* string类型

字符串类型，由unicode的字符，数字和标点组成

注意：字符串在使用时必须用””或’’引起来。

查看字符的unicode码

var str=’张三丰’;

r=str.charCodeAt(); #得到“张“的十进制的数字

r=str.charCodeAt(1); #得到“丰“的十进制数字

r=r.toString(16); #将 r 转换成十六进制

r=r.toString(2); #将 r 转换成二进制

如何将unicode码转换为对应的字符（必须是4位的十六进制码，不足的话前面补零）

已知unicode码：5f20,转换成相应的字符

var str=’\u5f20’;

中文的范围：\u4e00 ---\u9fa5

转义字符：

\n：换行

\t：制表符

\”：”

\’：’

\\：\

* boolean类型

布尔类型，只用于表示真（true）或假（false）

注意：boolean类型是可以参与到数字运算中的，true当成1运算，false当成0运算

* undefined类型

产生undefined场合：

1. 声明变量未赋值
2. 访问对象不存在的属性时

* 查看数据类型

使用typeof（）或typeof参看变量的数据类型

var num = 35；

var t1 = typeof(num);

var t2 = typeof num;

1. 引用数据类型
2. 数据类型转换

不同的数据类型之间的数据在进行运算时是如何转换的。

1. 隐式转换
2. 数字 + 字符串

将数字先转换为字符串，再进行拼接

var num=15;

var str=’18’;

var r=num+str; #结果为’1518’

“15”+18+15 ===》 “151815”

15+18+”15” ===》 “3315”

1. 数字 + 布尔

将布尔类型转换为数字

1. 布尔 + 字符串

将布尔转换为字符串，再拼接

var b=true;

var str=”nihao”;

var s=b+str; ===》”truenihao”

1. 布尔 + 布尔

注意：

* 任意类型与字符串做加法运算时，都是先将其转换成字符串再做拼接
* 字符串与数字进行运算时

加法：拼接

其余操作（-，\*，/，%）：先将字符串转换成数字，再做运算

1. 显示转换（转换函数、强制转换）
2. toString()

将任意类型的数据类型转换为字符串，并返回转换后的结果

语法：

var r = 变量.toString();

1. parseInt()

将任意类型的数据类型转换成整数，如果实在无法转换的话，则返回NaN（Not a Number）

语法：

var r =parseInt(数据)；

例：var r=parseInt(‘13’);===》13

var r=parseInt(’13.5’);===》13

var r=parseInt(‘13你好’);===》13

var r=parseInt(‘你好13’);===》NaN

1. parseFloat()

作用等同于parseInt(),允许转换为小数

例：var r = parseFloat(’12.5’); #12.5

var r = parseFloat(’12.5abc’); #12.5

var r = parseFloat(’abc12.5’); #NaN

1. Number()

将任意类型数据转换为数字，只要包含非法字符的话，结果就为NaN

例：var r = Number(‘3.5’); #3.5

var r = Number(‘3’); #3

var r = Number(‘3abc’); #NaN

1. 运算符
2. 算术运算符

+， -， \*， /， %， ++， --

++：自增运算符，在自身数据的基础上只做+1运算

--：自减运算符，在自身数据的基础上只做-1运算

例：var a = 15;

console.log(a++); #先输出15，再变为16

console.log(a); #输出16

console.log(++a); #先变为17，再输出17

console.log(a); #输出17

++做前缀：对变量先自增，再使用

++做后缀：先使用，再对变量自增

1. 关系运算符（比较运算符）

>, <, >=, <=, ==, !=,===,!==

如果运算符的两端有一个操作数是数字的话，那么另外一个会自动通过Number()进行转换，然后再进行比较。

NaN只有做 不等 操作时才是true，其余都是false

NaN == NaN #false

NaN != NaN #true

NaN != ‘zhang’ #true

===：比较数据的数值和数据类型都相等的情况下，结果才为真

!==：比较的数据的数值和数据类型只要有一个不等，结果就是true

1. 逻辑与运算符

!：非，对现有条件取反，等同于python not

&&：逻辑与，等同于 python and

||：逻辑或，等同于python or

1. 位运算符

**&：按位与** （全1取1，有0取0）

3（011） & 5（101）；===》1（001）

使用场合：判断一个数字的奇偶性，任意数字与1做按位与操作，结果为1是奇数，结果为0是偶数。

**|：按位或**（有1取1，全0取0）

任意小数与0做按位或操作，快速取整（抛弃小数位，保留整数位）

**^：按位异或**（不同为1，相同为0）

在不借助第三方变量的前提下，交换两个数字的位置

var a=5;

var b=3;

a=a^b; ===》a=6

b=b^a; ===》b=5

a=a^b; ===》a=3

1. 条件运算符

单目运算符（一元运算符）：只有一个操作的运算符

++ -- ! typeof

双目运算符（二元运算符）：有两个操作数的运算符

+ - \* % … && || & |

三目运算符（三元运算符）：有三个操作数的运算符

？：

条件表达式 ？ 表达式1 ： 表达式2；

先判断条件表达式的值，true执行表达式1，false执行表达式2

表达式1和表达式2的位置处，还可以是一个条件表达式

1. 赋值运算符

## Day52

1. 流程结构：
2. 顺序结构
3. 分支结构（选择结构）
4. 循环结构
5. 分支结构
6. 作用

根据条件，选择某一段代码去执行

1. if 结构
2. if语法：

if(条件){

满足条件要执行的代码块

}

注意：

* if后的{}是可以省略的，如果省略的话，则只控制该结构下的第一条语句
* 条件尽量是boolean类型的，如果不是，以下条件当成假运算：

1. if（0）{}
2. if（0.0）{}
3. if（””）{}
4. if（NaN）{}
5. if（undefined）{}
6. if（null）{}
7. if\_else语法：

if(条件){

语句块1

}else{

语句块2

}

1. if\_else\_if语法：

if(条件){

语句块1

}else if(条件){

语句块2

}else{

语句块3

}

1. switch结构

场合：只有在等值判断时使用

语法：

switch(变量){

case 值1:

语句块1

break; //可以省略

case 值2:

语句块2

break;

… …

default: //只有所有的case块都未匹配上时，才执行，可省略

语句块n

}

注意：

* 变量和各个case块后面的值，是使用===的方式来判断的
* 如果case后不增加break的话，则从匹配的case开始，向下依次执行（不判断），知道结束或碰到break为止

1. 循环结构
2. 循环的作用

重复的执行相同或相似的代码

1. 循环二要素
2. 循环条件
3. 循环操作（循环体）
4. while循环
5. 语法

while(条件){

循环体

}

1. do…while循环
2. 语法

do{

循环体

}while(条件);

1. 流程
2. 先执行循环体
3. 判断循环条件
4. 条件为真，继续执行循环体，否则跳出循环
5. 循环的流程控制
6. break

用在循环体内，用于跳出整个循环结构

1. continue

用在循环体内，用于跳出本次循环，继续执行下次循环

1. for循环
2. 语法

for(表达式1；表达式2；表达式3){

循环操作

}

表达式1：循环条件的声明

表达式2：循环条件的判断

表达式3：更新循环条件变量

1. 流程
2. 先执行表达式1，声明循环条件（只执行一次）
3. 判断表达式2的值，true或false
4. 如果为true，则执行循环操作，false则退出循环
5. 执行完循环操作后再执行表达式3
6. 再判断表达式2的值
7. for 和while

* 相同点：

先判断循环田间，再执行循环操作的结构

* 不同点：

while：优先用在不确定循环次数的场合下

for：优先用在确定循环次数的场合下

## Day53

1. 循环的嵌套

允许在一个循环的内部再出现循环

1. 函数
2. 声明是函数

函数，即function，是一段预定义好，可以独立执行并包含多条执行语句的代码块

优点：

1. 多条执行语句的封装
2. 实现了代码的复用
3. 再JS中创建函数

function 函数名（参数列表）{

函数体

[返回值]

}

参数列表：

如果没有参数，可以为空

如果有参数，就编写参数列表，各参数用逗号隔开。

有参数的函数，在调用时就要传参，如果未传参的话，参数的值就是undefined

返回值：

在函数体内，经过运算后，需要传递给函数调用者的一个值，就是返回值。

返回值是可选的，有返回值的话，需要使用return进行返回，最多只能返回一个值

1. 函数的调用

在任意的JS合法的位置处，都允许做函数的调用

语法：

var ret = 函数名（参数列表）；

有参数，则传参，否则为空

有返回值，可以接收，否则返回undefined

1. 变量的作用域
2. 什么是变量的作用域

变量的作用域指的是变量可访问范围

1. 作用域的分类
2. 局部变量

使用var关键字声明在某个函数内的变量，都是局部变量

局部变量只能在声明的函数内使用，出了函数则无法使用

1. 全局变量

在function之外声明的变量都是全局变量

声明变量不使用var关键字的时候，都是全局变量

注意：全局变量，推荐放在所有的function之外，使用var关键字去声明

1. 数组----array
2. 什么是数组

数组（Array）是一个用于保存批量数据的数据结构

数组是按照线性结构来保存数据的

1. 创建数组
2. 创建有一个空数组

var 数组名 = []；

var 数组名 = new Array（）；

1. 创建数组并初始化元素

var 数组名 = [元素1，元素2，… …]；

var 数组名 = new Array（元素1，元素2，… …）；

1. 注意：

var 数组名 = new Array（length）；

length表示是初始化的长度，length为number类型

1. 数组的使用

获取获设置数组的元素值，一律使用下标

1. 获取数组的长度

属性：length

用法：数组名.length

1. 使用场合

* 能够找到数组中，最新要插入元素的位置（即向数组尾部增加元素时使用）
* 清空数组，将数组的length属性设置为0，即可清空数组

array.length = 0;

* 配合循环，遍历数组中的每一个元素

for(var i=0;i<arr.length;i++){

console.log(arr[i]);

}

1. 常用API
2. toString（）

作用：将数组转换成字符串，将转换后的字符串进行返回

1. join（seperator）

作用：返回一个由seperator连接的数组元素的字符串

var a=[‘h’,’e’,’l’]

str = a.join(‘-’)

console.log(str) ===》’h-e-l’

1. reverse()

作用：反转，会改变现有数组的结构

1. sort（）

作用：对现有数组的内容进行排序，默认是按照元素的unicode码升序排序，会改变现有数组的结构

允许通过自定义的排序规则（排序函数）来指定数字的排序方式

语法：arr.sort(排序函数);

* 升序的排序函数：

function sortAsc(a,b){

return a-b; #（return b-a,则为降序）

}

数组会自动将相邻的两个数字传递到参数的位置处

如果返回值是大于0，则交换两个数字的位置，否则不变

指定排序函数：

arr.sort(sortAsc);

* 使用匿名函数指定排序规则

匿名函数：是一个没有名称的函数，只为某一功能而存在

arr.sort(function(a,b){return a-b;});

## Day54

1. 数组
2. 进出栈操作

栈式操作：提供了快速操作数组头部和尾部元素的方法

1. push（）

入栈，压栈，向数组尾部增加新元素，并返回新数组的长度

1. pop（）

出栈，弹栈，删除并返回数组尾部的元素

1. unshift（）

向数组的头部添加新元素并返回数组的长度

1. shift（）

删除并返回数组头部的元素

1. 二维数组
2. 什么是二维数组

数组中的每一个元素又是数组

1. 声明二维数组

var arr = [[1,2,3],[‘1’,’2’,’3’]];

1. 字符串（string）
2. 声明字符串

var str1 = ‘字符串1’；

var str2 = String(‘字符串2’);

var str3 = new String(‘字符串3’);

1. length属性

作用：返回当前字符串中字符的个数

1. 常用函数----String API
2. 大小写转换函数

toUpperCase（）

返回当前字符串的完全大写形式

toLowerCase（）

返回当前字符串的完全小写形式

1. 获取指定位置的字符 或 字符的unicode码
2. 获取指定位置的字符

函数：str.charAt(index)

1. 获取指定位置的字符的Unicode码

函数：str.charCodeAt(index)

1. 检索字符串
2. indexOf(value,fromIndex)

value：要查找的子字符串

fromIndex：从哪个下标位置处开始查找，如果省略，则从第一个字符处开始查找

返回值：返回value第一次出现的下标，如果没有查询到子字符串的话，则返回-1

1. lastIndexOf（value，fromIndex）

作用：查找value最后一次出现的下标

1. 截取子字符串

函数：substring（start，end）

作用：返回从start到end-1之间的字符串，如果省略end的话，则截取到整个字符串的尾部。

1. 分割字符串

作用：将一个由指定连接符的字符串，按照连接符给拆开，拆分成一个字符串数组

函数：split（分隔符）

1. 模式匹配
2. 作用

配合着正则表达式完成字符串的查找和替换

1. 正则表达式

语法：/正则格式/修饰符

修饰符：

i：ignorcase（忽略大小写）

g：global（全局匹配）

m：multiple（允许多行匹配）

1. 函数
2. replace（substr/regexp，replacement）

作用：在一个字符串中，将substr或满足regexp格式的字符串替换成replacement

1. match（substr/regexp）

按照指定的子字符串或正则表达式进行匹配，并返回满足格式的子字符串（数组）

1. 其他内置对象
2. JS中对象分类
3. 内置对象 --- ES提供
4. 外部对象

* window（BOM）
* document（DOM）

1. 自定义对象 --- function

对象：包含属性和方法（函数）

对象.属性

对象.方法（）

1. RegExp 对象

RegExp -- Regular Expression

1. 创建regexp对象
2. var regExp = /匹配模式/修饰符；

var reg = /垃圾/igm

1. var regExp = new RegExp(‘匹配模式’，”修饰符”)；

var reg = new RegExp（‘垃圾’，‘g’）；

1. RegExp对象的方法
2. test（string）

string：要验证的字符串

作用：验证string是否满足当前正则表达式对象的格式。如果string满足，返回true，否则返回false

1. Math 对象
2. 作用

执行与数学相关的运算和数据

1. 属性

Math.PI：π

Math.E：表示自然对数

1. 方法

Math.sin()

Math.cos()

Math.tan()

Math.sqrt(x):开平方

Math.log(x):求对数

Math.pow(x,y):求x的y次方

Math.abs(x):求x的绝对值

Math.max(a,b,c,…):求最小值

Math.min():求最小值

Math.random():生成0-1之间随机数

Math.round(x):将x四舍五入

1. Date对象
2. 作用

获取客户端的日期时间

1. 创建Date对象
2. 获取当前日期时间

var now = new Date();

1. 初始化自定义日期时间对象

var date = new Date(‘2018/05/01 00:12:13’)

1. 方法
2. 读取或设置当前时间的毫秒
3. getTime（）

返回自1970-1-1 00：00：00到date对象所经过的毫秒数

1. setTime（毫秒数）

根据给定的毫秒数，结合1970-1-1计算日期

1. 读取时间分量
2. getFullYear（）

获取当前日期对象的四位年份数

1. getYear（）

返回自1900年以来，到当前日期对象所经过的年份

1. getMonth()

返回0-11的数字来表示1-12月

1. getDate（）

返回当前日期对象的日

1. getDay（）

返回当前日期对象的星期

返回0-6 表示星期日到星期六

1. 获取时间

getHours（）：获取小时

getMinutes（）：获取分

getSeconds（）：获取秒

getMilliseconds（）：获取毫秒

1. 转换为字符串

toString（）

toLocaleString（）

toLocaleTimeString（）

toLocaleDateString（）

## Day55

1. 外部对象
2. BOM 和 DOM

BOM：Brower Object Model（浏览器对象模型）

将浏览器比喻成一个对象—window（网页初始化时会自动创建），可以通过该对象灵活的操作浏览器

DOM：Document Object Model（文档对象模型）

将HTML文档比喻成一个对象—document，可以灵活的控制网页上的内容，该对象属于window的属性之一，使用时不用声明

1. window对象（BOM模型）
2. 作用

在JS中表示浏览器窗口

window下的属性和方法在使用的过程中，可以省略window。而直接使用属性和方法

window.alert() ===》 alert()

window.document ===》 document

window.history ===》 history

1. window中的对话框
2. 警告框 window.alert()/alert()
3. 输入框 window.prompt()/prompt()
4. 确认框 window.confirm(‘’)/confirm(‘’)

点击“确定”按钮的话，返回true

其余的所有操作，返回false

1. window中的定时器
2. 周期性定时器

每间隔一段时间后，就执行一遍程序，反复执行

声明定时器：

var ret = setInterval（fun，duration）

fun：要周期性执行的操作，可以是匿名函数

duration：要间隔的时间周期，以ms为单位

ret：返回创建好的定时器对象（清除定时器时使用）

清除定时器：

clearInterval（timer）

timer：要清除的定时器对象

1. 一次性定时器

特点：在指定的时间间隔后，只执行一次操作

声明一次性定时器：

var ret = setTimeout（fun，time）

fun：等待一定时间后执行的操作

time：要等待的时长

ret：已启动的定时器对象

清除定时器：clear Timeout（timer）；

1. window中的属性
2. screen属性

作用：获取客户端显示器的相关信息

属性：

1. width
2. height
3. avaliWidth
4. avaliHeight
5. history属性

作用：包含当前窗户所访问过的url地址们

属性 & 方法

属性：length 所访问过的url的数量

方法：

back（）：后退

forward（）：前进

go（num）：在当前网页的基础上前进或后退，num正则前进

1. location属性

表示浏览器上地址栏的信息

属性 & 方法

属性：href

表示当前窗口中正在浏览的网页的地址，如果为href赋值，相当于实现网页跳转功能

方法：reload()

重新加载当前网页，相当于刷新

1. navigator属性



1. document对象（DOM模型）
2. document的概述

document对象，是DOM模型中的顶层对象，封装了所有和HTML元素相关的属性，方法以及事件

网页在加载的时候，会在内存中生成一棵节点树（DOM树）。DOM树会封装网页上所有的内容。网页上的每一个元素，每一个属性，每一段文本（包括注释），都会被封装成DOM树上的一个单独的节点

节点分为以下类型：

* 元素节点：表示网页中的一个元素
* 属性节点：表示元素中的一个属性
* 文本节点：表示元素中的文本内容
* 注释节点：表示网页中的注释内容
* 文档节点：表示整个HTML文档

DOM中所提供的操作：

* 节点查找
* 读取节点信息
* 修改节点信息
* 删除节点
* 创建节点

1. 查找节点
2. 通过元素id查找节点

前提：元素一定要具备id属性，否则无法查找

**var elem = document.getElementById(‘元素id’)**；

elem：对应id的元素在JS中表现---DOM对象/DOM元素

DOM属性：

* + innerHTML

修改/获取/设置当前DOM对象的文本

* + innerText

修改/获取/设置当前DOM对象的普通文本

* + value

该属性只针对表单控件，允许获取或设置表单控件的值

1. 读取节点信息
2. 节点的类型

属性：nodeType

值：

返回1：元素节点

返回2：属性节点

返回3：文本节点

返回8：注释节点

返回9：文档节点

1. 节点的名称

属性：nodeName

元素节点和属性节点返回：元素名 或 属性名

文本节点返回：#text

文档节点返回：#document

注释节点返回：#comment

1. 获取 或 设置 元素节点的属性值
2. getAttribute（attrName）

作用：获取指定属性的值

返回值：attrName属性对应的值

1. setAttribute（attrName，attrValue）

作用：修改或设置指定属性的值

attrName：要修改或设置的属性名

attrValue：要修改或设置的值

1. removeAttribute（attrName）

作用：将attrName属性从节点中移除出去

1. 通过 “dom对象.属性名” 获取或设置属性值

注意：class属性不适用于该方式

1. 元素的样式（CSS）
2. 使用setAttribute 设置class属性的值
3. 使用元素的className属性修改class值
4. 使用内联方式设置属性值

elem.style.css属性名 = 值；

例：elem.style.color = ‘red’;

注意：如果css属性名中包含“-”，那么“-”要取消，并且后面单词的第一个字符变大写

font-size

elem.style.fontSize = ‘red’;

1. DOM—查询节点
2. 根据ID查询节点

document.getElementById()

1. 根据层级节点结构查询
2. parentNode

根据当前的节点的层级结构，返回父元素节点

1. childNodes

根据当前的节点的层级结构，返回当前元素的所有子元素数组

注意：能够获取所有的文本节点和元素节点

1. children

根据当前的节点的层级结构，返回当前元素的所有子元素节点数组

1. nextSibling

获取当前节点的下一个兄弟节点

1. nextElementSibling

获取当前节点的下一个兄弟元素节点

1. previousSibling

获取当前节点的上一个兄弟节点

1. previousElementSibling

获取当前节点的上一个兄弟元素节点

1. 通过标签名称获取页面元素

语法：document|elem.getElementsByTagName(‘标签名’)

返回值：返回包含指定标记的元素数组们

1. 通过元素的name属性值获取元素

语法：document.getElementsByName(‘name’)

返回值：包含指定name属性值的元素的数组

1. 通过元素的class属性值获取元素

语法：document|elem.getElementsByClassName(‘class’)

返回值：返回包含指定class属性值的所有元素

1. 增加节点
2. 创建节点

语法：

var elem = document.createElement(‘元素名’)

1. 增加节点

将创建好的元素增加到网页中

1. document.body.appendChild(elem)

向body中追加elem新元素

1. parentNode.appendChild(elem)

向parentNode的内部追加elem新元素

parentNode可以表示页面上的任意一个节点

1. parentNode.insertBefore（newElem,oldElem）

将newElem插入到parentNode中oldElem之前

1. 删除节点

在DOM中，删除节点的行为只能由父元素发起

1. 删除body中的子元素

document.body.removeChild(elem);

1. 删除其他元素的子元素

parentNode.removeChild(elem);

删除parentNode中的elem子元素

## Day56

1. 事件
2. 什么是事件

通常由用户的行为来激发的操作

1. 触发事件的行为

所有的事件在绑定的时候，前面要加on

1. 鼠标事件

click：鼠标单击事件

mouseover：鼠标移入元素激发的事件

mouseout：鼠标移除元素激发的事件

mousemove：鼠标在元素内移动时的事件

1. 键盘事件

keydown：键位按下时事件

keypress：键位按下时事件

keyup：键位抬起时事件

1. 状态改变事件

load：当元素加载完成时激发的事件

change：当元素发生改变时激发的事件

focus：当元素获取焦点时触发的事件

blur：当元素失去焦点时触发的事件

submit：当表单被提交时触发的事件

1. 绑定事件的方式
2. 在元素中绑定事件

语法：<标记 on事件名=’执行函数（）’>

例：<div onmouseover=’执行函数（）’></div>

<body onload=’执行函数（）’></body>

1. 在js中动态的为元素绑定事件

var div = document.getElementById(‘d1’);

div.on事件名 = function(){

事件操作

}

注意：在动态绑定事件中，在事件处理函数内，允许通过this关键字，来表示当前的元素（DOM元素）

1. 事件行为
2. load事件

常用方式：为body绑定load事件，目的是为了在所有内容都加载完成之后再执行的操作要放在此处

* <body onload=’’></body>
* JS中动态绑定（可放在head中）

window.onload = function(){

$(‘d1’).onclick = function(){

}

}

1. submit事件

只有在表单被提交时才会触发

注意：该事件需要一个boolean的返回值来通知表单是否要提交。返回为true，可以提交表单，返回为false则阻止表单提交

1. 事件对象（事件参数对象，event对象）
2. 什么是事件对象

任何一个事件在触发后，都会自动产生一个event对象

event对象中包含与当前事件相关的一些属性和方法

1. 获取event对象
2. 在html元素中绑定事件

<标记 onclick=”btn(evnet)”>

function btn(event){

事件处理函数

}

1. 在JS中动态为元素绑定事件

var d1 = $(‘d1’);

d1.onclick = function(event){

事件处理

}

1. 事件对象的常用属性
2. 事件源（触发当前事件的元素是谁）
3. 获取事件源

通过event.target获取事件源

事件源是一个DOM对象

1. 鼠标事件（事件对象的常用属性）

鼠标事件：

click，mouseover，mouseout，mousemove

1. offsetX，offsetY

获取鼠标在元素上的坐标点

以元素的左上角为（0，0）

1. clientX，clientY

获取鼠标在网页上的坐标点

1. screenX，screenY

获取鼠标在屏幕上的坐标点

1. 键盘事件（事件对象的常用属性）

键盘事件：keypress，keydown，keyup

1. keypress（只有在输入字符的前提下，才会被激发）

which：当前按下字符的ascii码

key：当前按下的字符

注意：该事件允许通过一个返回值，通知元素是否要处理该事件。返回值为true，则正常显示输入的字符，返回值为false，则终止显示

1. keydown（只有按下键盘键位时就会被触发，无论是否有字符的输入）

which：当前按下键位的键位码，只区分按键，并不区分大小写

返回值：同keypress

1. 事件冒泡
2. 什么是事件冒泡

当激发子元素事件时，同时把父元素对应的事件也给执行了

场合：

1. 必须是父子结构或具备层级关系的元素
2. 必须同时设置了相同的事件
3. 阻止事件冒泡

让该事件在当前元素中执行，不向上冒泡

event.stopPropagation();

## Day57

1. jQuery
2. jQuery介绍

jQuery是一个轻量级的JS库 - 是一个被封装好的JS文件，提供了更为简便的元素操作方式

jQuery封装了DOM

jQuery核心理念：Write Less Do More

jQuery版本：

jQuery 2.x 不再兼容IE6,7,8

jQuery 1.11.x

1. 使用jQuery
2. 引入jQuery文件

<script src="jquery-1.11.3.js"></script>

注意：该文件的引入操作必须要放在其他的jQuery操作之前

1. 使用jQuery
2. jQuery对象
3. 什么是jQuery对象

jQuery对象是由jQuery对页面元素进行封装后的一种体现

jQuery中所提供的所有操作都只针对jQuery对象其他对象(DOM对象)无法使用

1. 工厂函数 - $()

想要获取jQuery对象的话，则必须使用工厂函数$()

在$()中允许传递一个选择器/DOM对象作为参数，$()能够将选择器 和 DOM对象全部封装成jQuery对象再进行返回

1. DOM对象 和 JQuery对象之间的转换

DOM对象：不能使用jQuery提供的操作

jQuery对象：不能使用DOM提供的操作

* 将DOM对象转换为jQuery对象

语法：var 变量=$(DOM对象)

注意：所有的jQuery对象在起名的时候，最好在变量前加 $,主要用于和DOM对象的区分

* 将jQuery对象转换为DOM对象

1.var dom对象=jQuery对象[0];

2.var dom对象=jQuery对象.get(0);

1. jQuery选择器
2. 作用

获取页面上的元素们，返回值都是由jQuery对象所组成的数组

语法：$("选择器")

1. 常用选择器
2. 基本选择器
3. ID选择器

$(“#id”)

返回：返回页面中指定ID值的元素

1. 类选择器

$(“.className”)

返回：页面中指定className的所有元素

1. 元素选择器

$("element")

返回：页面中指定标记的所有元素

1. 群组选择器/复合选择器

$("selector1,selector2,...")

返回：返回满足函数内所有选择器的函数们

1. 层级选择器
2. $("selector1 selector2") 后代选择器
3. $("selector1>selector2") 子代选择器
4. $("selector1+selector2") 相邻兄弟选择器

作用：匹配紧紧跟在selector1后面且满足selector2选择器的元素

1. $("selector1~selector2") 通用兄弟选择器

作用：匹配selector1后面所有满足selector2选择器的元素

1. 基本过滤选择器

过滤选择器通常都会配合着其他的选择器一起使用

1. :first

只匹配一组元素中的第一个元素

$("p:first")

1. :last

只匹配一组元素中的最后一个元素

1. :not("selector")

在一组元素中，将满足selector选择器的元素排除

1. :odd

匹配 偶数行 元素(奇数下标)

1. :even

匹配 奇数行 元素(偶数下标)

1. :eq(index) -- equals

匹配 下标等于 index 的元素

1. :gt(index)

匹配 下标大于 index 的元素

1. :lt(index)

匹配 下标小于 index 的元素

1. 属性过滤选择器

依托于html元素的属性来进行元素过滤的

1. [attribute]

作用：匹配包含指定属性的元素

ex：

div[id]:匹配具备id属性的div元素

1. [attribute=value]

作用：匹配attribute属性值为value的元素

ex：

input[type=text]

input[type=password]

1. [attribute!=value]

作用：匹配attribute属性值不是value的元素

1. [attribute^=value]

作用：匹配attribute属性值是以value字符开头的元素

ex：

p[class^=col]

1. [attribute$=value]

作用：匹配attribute属性值是以value字符结尾的元素

1. [attribute\*=value] [attribute\*=value]

作用：匹配attribute属性值中包含value字符的元素

1. 子元素过滤选择器
2. :first-child

匹配属于其父元素中的首个子元素

1. :last-child

匹配属于其父元素中的最后一个子元素

1. :nth-child(n)

匹配属于其父元素中第n个子元素

1. jQuery操作DOM
2. 基本操作
3. html()

作用：获取 或 设置 jQuery对象中的html内容

ex：

console.log($("#main").html());

$("#main").html("");

1. text()

作用：获取 或 设置 jQuery对象中的text内容

1. val()

作用：获取 或 设置 jQuery对象中的value值(表单控件)

1. 属性操作

* attr()

作用：读取 或 设置jQuery对象的属性值

ex:

$obj.attr("id"); 获取 $obj 的id属性值

$obj.attr("id","main"); 设置$obj对象的id属性值为main

* removeAttr(“attrName”)

删除jQuery对象的attrName属性

ex：

$obj.removeAttr("class");

1. 样式操作
2. attr（）

$obj.attr("class","redBack");

1. addClass(“className”)

作用：将className 添加到元素的class值之后

ex：

$obj = $obj.addClass("c1");

$obj = $obj.addClass("c2");

连缀调用：

$obj.addClass("c1").addClass("c2");

1. removeClass("className")

如果无参的话，则清空类选择器

如果有参数的话，则删除对应的类选择器

ex：

$obj.removeClass("c1") 将c1类选择器从$obj移除出去

$obj.removeClass() 清空$obj的所有类选择器

1. toggleClass("className")

切换样式：

元素如果具备className选择器，则删除

元素如果没有className选择器，则添加

1. css("属性名")

$obj.css("width"); 获取$obj对象的width属性值

1. css("属性名","属性值")

$obj.css("background-color","yellow");

设置$obj对象的background-color的属性值为yellow

1. css(JSON对象)

JSON对象：

是一种约束了格式的对象表现形式

JSON:JavaScript Object Notation

JSON对象的表示方式：

* JSON对象必须使用{}括起来
* 使用键值对的方式来声明数据(表示属性和值)
* 所有的属性在使用时必须使用""括起来，值如果是字符串的话，也必须使用""括起来
* 属性和值之间使用（“:”）连接
* 多对属性和值之间使用 （“,”） 隔开

ex：

$obj.css({

“color”:”red”,

“font-size”:”32px”,

“float”:”left”

});

1. 遍历节点
2. children（）/children("selector")

获取某jQuery对象的所有子元素 或 带有指定选择器的子元素

注意：只考虑子代元素，不考虑后代元素

1. next() / next("selector")

获取某jQuery对象的下一个兄弟元素 / 满足selector的下一个兄弟元素

1. prev() / prev("selector")

获取某jQuery对象的上一个兄弟元素 / 满足selector的上一个兄弟元素

1. siblings() / siblings(selector)

获取某jQuery对象的所有兄弟元素 / 满足selector的所有兄弟元素

1. find("selector")

查找满足selector选择器的所有后代元素

1. parent()

查找某jQuery对象的父元素

1. 创建对象

语法：$("创建的标记")

ex：

1. 创建一对div

var $div = $("<div></div>");

$div.html("动态创建的div");

$div.attr("id","container")

$div.css("color","red");

1. 创建一对div

var $div = $("<div id='container' style='color:red;'>动态创建的div</div>");

1. 插入元素

作用：将创建好的元素插入到网页中

1. 内部插入

作为元素的子元素插入到网页中

* $obj.append($new); 将$new元素插入到$obj元素中的最后一个子元素位置处(追加)
* $obj.prepend($new); 将$new元素插入到$obj元素中的第一个子元素位置处

1. 外部插入

作为元素的兄弟元素插入到网页中

* $obj.after($new); 将$new元素作为$obj的下一个兄弟元素插入进来
* $obj.before($new); 将$new元素作为$obj的上一个兄弟元素插入进来

1. 删除元素

$obj.remove(); 将$obj元素删除出去

1. jQuery中的事件处理
2. 页面加载后的执行

类似于window.onload 但不同于 window.onload

jQuery加载后执行的特点：在DOM树加载完毕的时候就开始执行

$(document).ready(function(){

//页面的初始化操作

//DOM树加载完成后就开始运行

});

$().ready(function(){

//页面的初始化操作

//DOM树加载完成后就开始运行

});

$(function(){

//页面的初始化操作

//DOM树加载完成后就开始运行

});

1. jQuery的事件绑定
2. 方式1

$obj.bind("事件名称",事件处理函数);

ex：

$obj.bind(“click”,function(){

//事件的行为操作

console.log("... ....");

});

1. 方式2

$obj.事件名称（function(){

//事件处理函数

}）;

ex：

$obj.click(function(){

//通过 this 来表示触发该事件的DOM对象

});

1. 事件对象——event

在绑定事件的时候，允许传递 event 参数来表示事件对象

1. 大幅度

$obj.bind("click",function(event){

//event表示当前事件的事件对象

});

1. 大幅度

$obj.click(function(event){

//event表示当前事件的事件对象

});

event的使用方式与原生JS事件中的event使用方式一致

event.stopPropagation() : 阻止事件冒泡

event.offsetX:

event.offsetY:

event.target:获取事件源

1. jQuery动画
2. 基本显示/隐藏

语法：

$obj.show() / $obj.show(执行时间)

$obj.hide() / $obj.hide(执行时间)

1. 滑动式显示/隐藏

语法：

显示：$obj.slideDown() / $obj.slideDown(执行时间)

隐藏：$obj.slideUp() / $obj.slideUp(执行时间)

1. 淡入淡出式显示/隐藏

语法：

显示：$obj.fadeIn() / $obj.fadeIn(执行时间)

隐藏：$obj.fadeOut() / $obj.fadeOut(执行时间)