1. 可以修改目录的SGID权限，并修改用户的umask来改变创建目录和文件时的默认权限，使得其他同组用户无法读写执行。
2. egrep -o "\b[[:alpha:]]+\b" filename | awk '{++count[$0]} END{for (word in count){ printf("%-15s%3d\n",word,count[word]);}}' | sort -n -r -k2,2 | head -10  
   在终端中运行后可以得到文件中词频前十的单词和词频。
3. 代码如下：

#!/bin/bash

a=1

b=1

num=0

echo -ne "$a\t$b\t"

while [ $num -lt 4 ]

do

let "a=a+b"

let "b=b+a"

let "num=num+1"

echo -ne "$a\t$b\t"

done

1. 代码如下：

def sum\_digit(num):

return sum(int(i) for i in str(num) if i.isdigit())

def main():

for num in range(100, 10000):

temp = num

while len(str(temp)) != 1:

temp = sum\_digit(temp)

if temp == 7:

print(num)

main()

1. Python代码如下：

def reverse\_num(num):

res = []

for i in str(num):

res.append(int(i))

rev = 1000000 \* res[6] + 100000 \* res[5] + 10000 \* res[4] + 1000 \* res[3] + 100 \* res[2] + 10 \* res[1] + res[0]

print(rev)

Bash代码如下：

#!/bin/bash

num=$1

n1=$num/1000000

n2=$num%1000000/100000

n3=$num%100000/10000

n4=$num%10000/1000

n5=$num%1000/100

n6=$num%100/10

n7=$num%10

let”r=$n1+$n2\*10+$n3\*100+$n4\*1000+$n5\*10000+$n6\*100000+$n7\*1000000"

echo $r

1. 简单列举以下一些常见的特殊方法：

\_\_init\_\_初始化函数

\_\_doc\_\_ 文档

\_\_dict\_\_字典

\_\_str\_\_ 和 \_\_repr\_\_输出形式，前者显示给用户，后者显示给开发

\_\_cmp\_\_排序函数

\_\_len\_\_元素个数

\_\_add\_\_、\_\_sub\_\_、\_\_mul\_\_、\_\_div\_\_等是运算符重载

1. 代码如下：

class NumList:

def \_\_init\_\_(self, num\_list):

self.num\_list = num\_list

def \_\_str\_\_(self):

return 'NumList({0})'.format(self.num\_list)

# 加法重载，且为正向加法

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

res = []

if isinstance(other, int) or isinstance(other, float):

res = [i + other for i in self.num\_list]

return NumList(res)

if isinstance(other, NumList):

if len(other.num\_list) == len(self.num\_list):

for i in range(len(self.num\_list)):

res.append(self.num\_list[i] + other.num\_list[i])

return NumList(res)

else:

if len(other.num\_list) < len(self.num\_list):

short\_list = other.num\_list

long\_list = self.num\_list

else:

short\_list = self.num\_list

long\_list = other.num\_list

j = 0

for i in range(len(long\_list)):

if j == 2:

j = 0

res.append(long\_list[i] + short\_list[j])

j += 1

return NumList(res)

# 加法重载，但是为反向加法，面对数值在前的情况

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

res = [i + other for i in self.num\_list]

return NumList(res)

# 乘法重载，且为正向乘法

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

res = []

if isinstance(other, int) or isinstance(other, float):

res = [i \* other for i in self.num\_list]

return NumList(res)

if isinstance(other, NumList):

if len(other.num\_list) == len(self.num\_list):

for i in range(len(self.num\_list)):

res.append(self.num\_list[i] \* other.num\_list[i])

return NumList(res)

else:

if len(other.num\_list) < len(self.num\_list):

short\_list = other.num\_list

long\_list = self.num\_list

else:

short\_list = self.num\_list

long\_list = other.num\_list

j = 0

for i in range(len(long\_list)):

if j == 2:

j = 0

res.append(long\_list[i] \* short\_list[j])

j += 1

return NumList(res)

# 乘法重载，但是为反向乘法，面对数值在前的情况

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

res = [i \* other for i in self.num\_list]

return NumList(res)

# 除法重载

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

if isinstance(other, int) or isinstance(other, float):

res = [i / other for i in self.num\_list]

return NumList(res)

a = NumList([1, 2])

b = NumList([2, 3, 4, 5, 6])

print(a + 2.1)

print(2.1 + a)

print(a + b)

print(a \* 3)

print(3 \* a)

print(a \* b)

print(a / 2)

运行结果：

NumList([3.1, 4.1])

NumList([3.1, 4.1])

NumList([3, 5, 5, 7, 7])

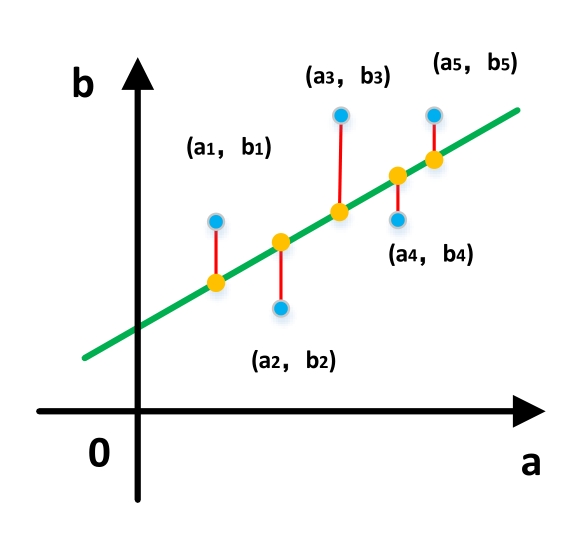
NumList([3, 6])

NumList([3, 6])

NumList([2, 6, 4, 10, 6])

NumList([0.5, 1.0])

1. 原理：对于一组数据，如果将每个数据与拟合直线对应的拟合点之间的距离称为误差，那么拟合最优就是误差平方和最小时，这就是最小二乘法原理。



对于线性回归而言，最小二乘法使得拟合结果对每个数据点误差平方和最小，此时拟合的效果最好。且线性回归的结果为一条直线，最小二乘法对这种情况的处理非常直观明了（如上图）。因此在线性回归中，我们最常用的就是最小二乘法拟合直线。

10.pandas将数组分为三种数据结构，分别是一维矩阵Series，二维矩阵DataFrame和三维面板Panel。

优势：

（1）可以轻易的处理浮点及非浮点数据类型的缺失值(NaN)

（2）大小可变：DataFrame和Panel都可以删除或插入列

（3）数据自动对齐

（4）灵活强大的分组功能，可对数据集进行拆分组合操作

（5）方便的将其他Python和NumPy数据结构中不同类索引的数据转换为DataFrame对象

（6）基于智能标签的切片，花式索引，轻易从大数据集中取出子集

（7）直观的合并，连接数据集

（8）轻易的重新定义数据集形状和转置

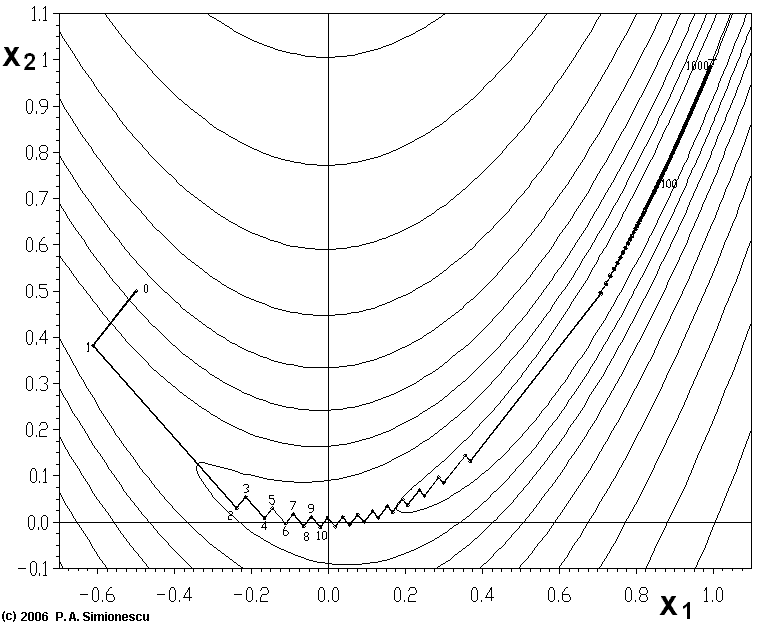
（9）轴(axes)的分层标签（使每个元组有多个标签成为可能）

11.**梯度下降法：**

**原理：**梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，因为该方向为当前位置的最快下降方向，所以也被称为是“最速下降法“最速下降法越接近目标值，步长越小，前进越慢。

**优点：**简单常用，理论提出早，更为完善。

**缺点：**（1）靠近极小值时收敛速度减慢，如下图所示;



　　 （2）直线搜索时可能会产生一些问题;

　　 （3）可能会“之字形”地下降。

（4）一般情况下，其解不保证是全局最优解，梯度下降法的速度也未必是最快的。

**牛顿法：**

**原理：**牛顿法是一种在实数域和复数域上近似求解方程的方法。方法使用函数f (x)的泰勒级数的前面几项来寻找方程f (x) = 0的根。

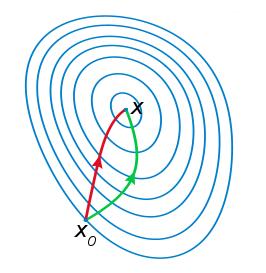
**优点：**（1）二阶收敛，收敛速度快

**缺点：**牛顿法是一种迭代算法，每一步都需要求解目标函数的Hessian矩阵的逆矩阵，计算比较复杂。

**二者区别：**

1. 从本质上去看，牛顿法是二阶收敛，梯度下降是一阶收敛，所以牛顿法就更快。
2. 从几何上说，牛顿法就是用一个二次曲面去拟合你当前所处位置的局部曲面，而梯度下降法是用一个平面去拟合当前的局部曲面，通常情况下，二次曲面的拟合会比平面更好，所以牛顿法选择的下降路径会更符合真实的最优下降路径。

**例子：（图形说明）**



图中左边为牛顿法迭代路径，右边为梯度下降法迭代路径

由上图可以看到，牛顿法迭代的次数少，每一次迭代时都会考虑下一步是否为最快下降，因此牛顿法要快于梯度下降法。