Day 5 数据类型&编码

一、数据类型之-元组tuple

1.1 元组是什么类型?

一句话定义，元组是只读列表，一旦创建，不可修改。只可进行查询、切片操作。

1.2 元组的用法

用`()`来表示

```
>>> names = ("Alex","Jack","Rain")

>>> type(names)
<class 'tuple'>
>>> names
('Alex', 'Jack', 'Rain')
```

查询

仅有`count`和`index`2个内置方法，用法跟list一致

```
>>> names
('Alex', 'Jack', 'Rain')
>>> names.index("Jack")
1
>>> "Rain" in names
True
```

切片

切片与列表用法一致

删除

`del names`

1.3 什么情况会用元组?

其实没什么特别要求必须什么时候 用元组的场景，这基本上是一个程序员自行决择的问题。

二、数据类型之-字典dict

2.1 什么是dict类型

引子

我们学了列表， 现在有个需求， 把你们公司每个员工的姓名、年龄、职务、工资存到列表里，你怎么存?

```
staff_list = [
    ["Alex",23,"CEO",66000],
    ["黑姑娘",24,"行政",4000],
    ["佩奇",26,"讲师",40000],
    # [xxx,xx,xx,xxx]
    # [xxx,xx,xx,xxx]
    # [xxx,xx,xx,xxx]
]
```

这样存没问题，不过你要查一个人的工资的话， 是不是得把列表遍历一遍

```
for i in staff_list:
    if i[0] == '黑姑娘':
        print(i)
        break
```

但假如你公司有2万人，如果你要找的黑姑娘正好在列表末尾，那意味着你要遍历2万次，才能找到这个信息。列表越大，查找速度越慢。

好了，现在福音来了， 接下来学要的字典可以 查询数据又快、操作又方便，是日后开发中必备神器。

```
staff_list = {
    "alex": [23, "CEO", 66000],
    "黑姑娘": [24, "行政", 4000],
    # ....
}

print(staff_list["黑姑娘"]) # 即可取出来
```

2.2 dict定义

{key1:value1，key2:value2}

```
info = {
  "name":"Alex Li",
  "age" : 26,
  "name":"Jacke"
}

key -> value
```

`:`号左边是key, 右边是value

dict特性

1. key-value结构
2. key必须为不可变数据类型（字符串、数字、元组）， (hashtable)
3. key 必须唯一 ,(hashtable)
4. 一个key对应的value可存放任意数据类型，可修改、可以不唯一
5. 可嵌套，即value也可是dict
6. py3.7之前是无序的, 3.7开始变成有序的了， ordered_dict
7. 查询速度快，且不受dict的大小影响，至于为何快？我们学完hash再解释。=O(1)

2.3 dict用法

创建

```
# 方法1
>>> info2 = {
       "name":"BlackGirl",
       "age":23
       }

# 方法2
>>> info = dict(name="alex",age=22)
>>> info
{'name': 'alex', 'age': 22}
```

增

```
>>> staff_list
{'alex': [23, 'CEO', 66000], '黑姑娘': [24, '行政', 4000]}
>>>
>>> staff_list["XiaoYun"] = [26,"前端开发",19000]  # 新增

>>> staff_list
{'alex': [23, 'CEO', 66000], '黑姑娘': [24, '行政', 4000], 'XiaoYun': [26, '前端开发',
19000]}
```

检查式新增

向dict里新增一个key,val值，如果这个key不存中，如果这个key已存在，就返回已存在的key对应的value

```
>>> staff_list.setdefault("Celina",[22,"Accountant",12000])  # 新增一个k,v值
[22, 'Accountant', 12000]
>>> staff_list
{'alex': [23, 'CEO', 66000], '黑姑娘': [24, '行政', 4000], 'XiaoYun': [26, '前端开发',
19000], 'Celina': [22, 'Accountant', 12000]}


>>> staff_list.setdefault("XiaoYun",[22,"Accountant",12000])  # 新增一个，但是XiaoYun已经
存在了，所以返回原来的值
[26, '前端开发', 19000]
```

改

普通修改

```
>>> names["xiao_yun"] = [23,"前台", 6000]
>>> names["xiao_yun"][2] = 4500  # 改values列表里的值
>>> names
{'xiao_yun': [23, '前台', 4500], 'Celina': [23, 'UE', 9999]}
>>>
```

合并修改

把另外一个dict合并进来

```
staff_list = {
    "alex": [23, "CEO", 66000],
    "黑姑娘": [24, "行政", 4000],
    "xiao_yun":[22,"Student",2000],

}

names = {
    "xiao_yun":[25,"前端开发",12000],
    "Celina":[23,"UE", 9999]
}

staff_list.update(names) # 把names的每个k,v赋值给staff_list，相当于如下操作
```

```
# for k in names:
#     staff_list[k] = names[k]

print(staff_list)
```

输出：{'alex': [23, 'CEO', 66000], '黑姑娘': [24, '行政', 4000], 'xiao_yun': [22, 'Student', 2000], 'Celina': [23, 'UE', 9999]}

查

```
>>> names
{'xiao_yun': [23, '前台', 4500], 'Celina': [23, 'UE', 9999]}

# get 方法
>>> names.get("Jack")  # 如果没有，返回None
>>> names.get("xiao_yun") # 如果有，则返回值
[23, '前台', 4500]


# 直取
>>> names["Jack"]    #  如果没有，报错
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Jack'


>>> names["Celina"]
[23, 'UE', 9999]
```

判断是否在dict里有指定的key

```
>>> names
{'xiao_yun': [23, '前台', 4500], 'Celina': [23, 'UE', 9999]}
>>>
>>> "Alex" in names
False
```

```
dic.keys()     #返回一个包含字典所有KEY的列表；
dic.values()   #返回一个包含字典所有value的列表；
dic.items()    #返回一个包含所有（键，值）元组的列表；
```

删

```
names.pop("alex")    # 删除指定key
del names["black_girl"] # 删除指定key,同pop方法
names.popitem()  # 以LIFO的方式删除一对值
names.clear()        # 清空dict
```

循环

```
1、for k in  dic.keys()
2、for k,v in dic.items()
3、for k in dic     # 推荐用这种，效率速度最快

info = {
    "name":"路飞学城",
    "mission": "帮一千万b粉白嫖学好编程",
    "website": "https://luffycity.com"
}
for k in info:
    print(k,info[k])
```

输出

```
name 路飞学城
mission 帮一千万b粉白嫖学好编程
website https://luffycity.com
```

特殊方法

fromkeys : 批量生成多个k,v的dict

```
>>> n
['alex', 'jack', 'rain']
>>> dict.fromkeys(n, 0)
{'alex': 0, 'jack': 0, 'rain': 0}
```

Copy: 浅copy , 同列表的copy一样

求长度

```
len(info) # len()方法可同时用于列表、字符串
```

解释器自带函数

2.4 dict 为何查询速度快?

因为dict是基于hashtable实现的， hashtable的原理导致你查询速度就是O(1)， 意思就是你即使有1亿个数据，查询某个值也只需1次搞定 。。。。。

哈哈， 上边这句，对现在的你，肯定完全懵逼。 什么是hashtable, 什么是O(1)....

忘记它吧，为师先用最简单的例子让你理解:

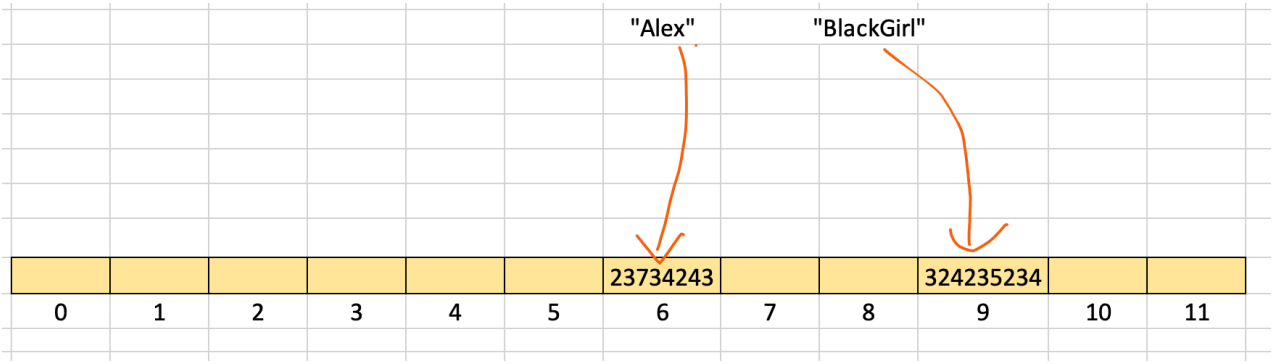
看以下列表nums, 让你用最快最高效的方式找出100这个数，你怎么找?

```
>>> nums
[125, 247, 53, 260, 33, 6, 132, 198, 151, 110, 95, 163, 116, 295, 289, 200, 142, 153,
126, 41, 294, 290, 31, 34, 241, 132, 191, 82, 246, 123, 19, 171, 37, 130, 139, 250,
53, 241, 84, 172, 14, 120, 138, 92, 294, 116, 269, 104, 278, 146, 12, 293, 135, 240,
289, 121, 295, 50, 60, 198, 10, 57, 248, 79, 169, 298, 292, 64, 298, 3, 9, 152, 268,
178, 0, 136, 165, 53, 146, 78, 270, 104, 116, 283, 177, 145, 142, 71, 176, 33, 235,
134, 188, 29, 39, 15, 136, 137, 16, 189]
```

1. 先排序
2. 再折半（2分）算法查找

粗识hashtable

hashtable(散列表) 用了个更nb的方式， 直接把你的key通过hash函数算出来个数字， 然后扔在hashtable的某个位置，



你再去通过dict.get("alex")获取的时候，它只需要把 `alex` 再变成一个hash值，去找在这个表里的值 就可以了。

这样， 无论你的dict多大， 你查询速度都不会受影响 .

2.3 dict练习题

1. 生成一个包含100个key的字典，每个value的值不能一样
2. {'k0': 0, 'k1': 1, 'k2': 2, 'k3': 3, 'k4': 4, 'k5': 5, 'k6': 6, 'k7': 7, 'k8': 8, 'k9': 9} 请把这个dict中key大于5的值value打印出来。
3. 把题2中value是偶数的统一改成-1
4. 把下面列表中的值进行分类， 变成dict,

```
Input : test_list = [4, 6, 6, 4, 2, 2, 4, 8, 5, 8]
Output : {4: [4, 4, 4], 6: [6, 6], 2: [2, 2], 8: [8, 8], 5: [5]}
需求 ：  值一样的要分类存在一个key里
```

5. 把一段话里重复的单词去掉

```
Input : Python is great and Java is also great
Output : is also Java Python and great
```

1. 写程序输出dict中values里唯一的值

```
dic = {'gfg': [5, 6, 7, 8], 'best': [6, 12, 10, 8], 'is': [10, 11, 7, 5], 'for': [1,
2, 5]}
结果 : [1, 2, 5, 6, 7, 8, 10, 11, 12]
```

1. 把所有下表中同字母异序词找出来

```
 arr = ['cat', 'dog', 'tac', 'god', 'act']
```

结果： `[ ['cat','tac','act'],['god','dog']]`

三、数据类型之-集合set

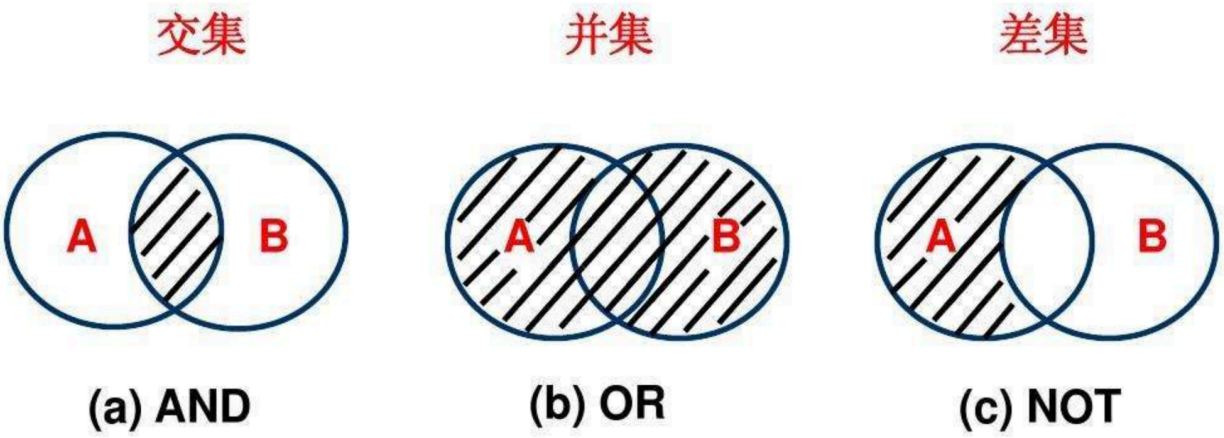
3.1 定义和作用

集合跟我们学的列表有点像，也是可以存一堆数据，不过它有几个独特的特点，令其在整个Python语言中占有一席之地，

1. 里面的元素不可变，代表你不能存一个list、dict 在集合里，字符串、数字、元组等不可变类型可以存
2. 天生去重，在集合里没办法存重复的元素

3. 无序，不像列表一样通过索引来标记在列表中的位置，元素是无序的，集合中的元素没有先后之分，如集合{3,4,5}和{3,5,4}算作同一个集合
4. 关系测试（作用）

基于上面的特性，我们可以用集合来干2件事，**去重和关系运算**



3.2 语法

创建

创建集合

注意也是`{}`，但不是dict, 不是k,v结构

```
>>> a = {1,2,3,4,2,'alex',3,'rain','alex'}
>>> a
{1, 2, 3, 4, 'alex', 'rain'}
```

由于它是天生去重的，重复的值你根本存不进去

帮列表去重

帮列表去重最快速的办法是什么？ 就是把它转成集合，去重完，再转回列表

```
>>> b
[1, 2, 3, 4, 2, 'alex', 3, 'rain', 'alex']
>>> set(b)
{1, 2, 3, 4, 'alex', 'rain'}
>>>
>>> b = list(set(b)) #一句代码搞定
>>> b
[1, 2, 3, 4, 'alex', 'rain']
```

增

```
>>> a = {1, 2, 3, 4, 5}
>>> a.add(9)
>>> a
{1, 2, 3, 4, 5, 9}
```

查

```
>>> a
{1, 2, 3, 4, 5, 9}
>>> 8 in a
False
```

改

不能修改，一旦创建，不可修改里面的值

删除

```
>>> a.pop()  # 随机删除一个值，并返回该值
2

>>> a.remove(3) # 删除3这个元素，若3不在，报KeyError

>>> a
{3, 4, 5}
>>> a.discard(8)  # 删除指定的值 ，若该值不存在,do nothing.
>>> a.discard(9)
>>> a.discard(5)
>>> a
{3, 4}
```

3.3 关系测试

> five_man_fight_1girl = {"Alex","peiqi","PyYu","OldVillageMaster","Egon","BlackGirl"}
>
> girls_game={"BlackGirl","XiaoYun","Celina","Alex"}

交集

```
>>> five_man_fight_1girl & girls_game   # & 符号
{'BlackGirl', 'Alex'}
>>> five_man_fight_1girl.intersection(girls_game)
{'BlackGirl', 'Alex'}
```

并集

```
>>> five_man_fight_1girl | girls_game  # | 符号
{'Celina', 'Alex', 'Egon', 'BlackGirl', 'peiqi', 'PyYu', 'OldVillageMaster',
'XiaoYun'}
>>> five_man_fight_1girl.union(girls_game)
{'Celina', 'Alex', 'Egon', 'BlackGirl', 'peiqi', 'PyYu', 'OldVillageMaster',
'XiaoYun'}
>>>
```

差集

```
>>> five_man_fight_1girl - girls_game  # - 符号代表 差集  , only in five_man_fight_1girl
{'PyYu', 'peiqi', 'OldVillageMaster', 'Egon'}
>>> girls_game - five_man_fight_1girl   # only in girls_game
{'Celina', 'XiaoYun'}
>>> five_man_fight_1girl.difference(girls_game)
{'PyYu', 'peiqi', 'OldVillageMaster', 'Egon'}
```

对称差集

```
>>> five_man_fight_1girl ^ girls_game   # ^ 符号,  把同时在2个电影里都出演过的人T出去
{'Celina', 'Egon', 'peiqi', 'PyYu', 'OldVillageMaster', 'XiaoYun'}
>>> five_man_fight_1girl.symmetric_difference(girls_game)
{'Celina', 'Egon', 'peiqi', 'PyYu', 'OldVillageMaster', 'XiaoYun'}
```

子集、父集

```
>>> five_man_fight_1girl.issubset(girls_game)  # 是不是girls_game的字集,  即five_man...里
的每个值都在girls_game里存在
False
>>> five_man_fight_1girl.issuperset(girls_game) # 是不是girls_game的父集
False
```

测试后修改

three_some = {'金角', '银角', '黑姑娘'}

four_p = {'金角', '银角',"黑姑娘", "XiaoYun"}

```
>>> four_p.difference(three_some)  #
{'XiaoYun'}
>>> four_p.difference_update(three_some) # 把差集的结果赋值给four_p
>>> four_p
{'XiaoYun'}

>>> four_p
{'XiaoYun'}
>>> four_p.update(three_some)  # 把合并后的结果赋值给four_p
>>> four_p
{'XiaoYun', '金角', '银角', '黑姑娘'}
```

四、二进制

4.1 引子

终于要讲2进制啦，讲之前，我们先讲个小故事，

大家知道古时候的中国是如何通信的么？

假如，战国时期两个国家要打仗了，我们垒了城墙，每隔一段就有兵镇守，现在有人来攻打我们了，然后我们是不是得通知其他人有人来打我们来了？怎么通知？

1. 派个人跑着去？等人回来，仗打完了
2. 飞鸽传书？不靠谱，鸽子会被敌人射下来做烧烤
3. 点狼烟信号，可行

好了，现在有5000精兵来打你了，你点了根狼烟搬救兵，从东边来了10个人，西边来了10个人，20个人来了，和你们一起战死了。

这怎么办？

我们不能这么保守了，只要我一点狼烟说有人来打我们了，先来他10000人，结果来了200个敌人，我们呼啦啦来一大堆人，是不是浪费资源啊？

我们是不是除了告诉人家要打仗了，还得告诉别人来了多少人啊？那我们怎么告诉？

来一个人点一根？来了5000人，点5000根，不用打了，自己给自己烧死了

那好我们就约定，来10个人点1根，来100个人点2根，来1000个人点3根，来5000个点4根，来10000个点5根。。。以此类推，恭喜你， 这样确实就能解决问题啦，粗略的能告诉友军来了多少敌人。

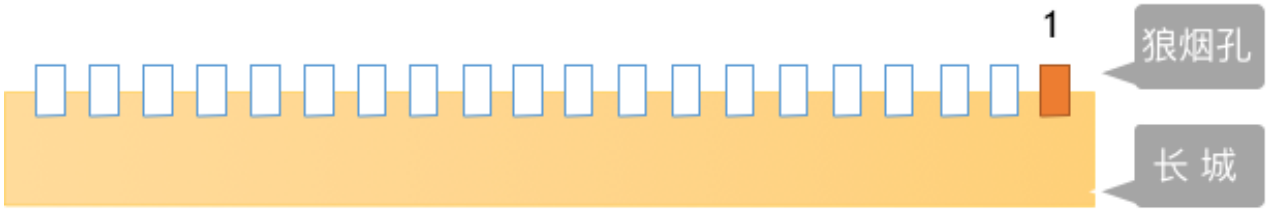
但现在友军将领提了一个变态的要求，你必须精确的告诉他一共来了多少敌人，他才安排来救援。 如何精确传送到底有多少个敌人?

各位同学可以自行思考研究5分钟，但我估计你想不出来哈哈。

好了，现在来看看我的方法。。。

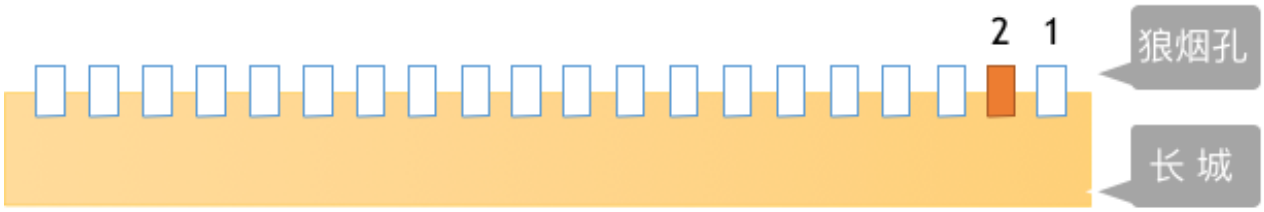
假如我们有20个狼烟孔，狼烟孔点燃了代表有人，没点燃代表没人。

这时候，1个敌人来了，点1根狼烟

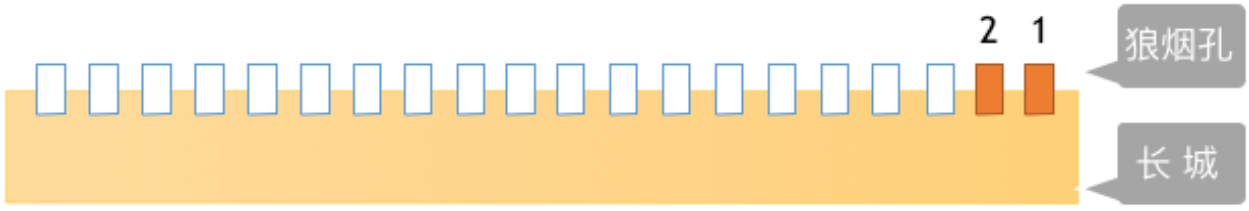


现在2个敌人来了，怎么办？再点一根狼烟，把20根狼烟都点上能表示20个人。。。这肯定不行。我们这样，把

第一个狼烟孔灭掉，点燃第二个，这样只点燃第二个孔就代表两个人



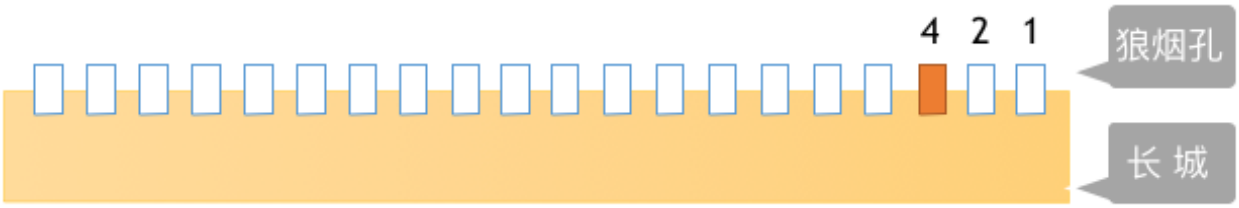
现在3个敌人来了，怎么办？把第一个狼烟孔点着了就表示3个人



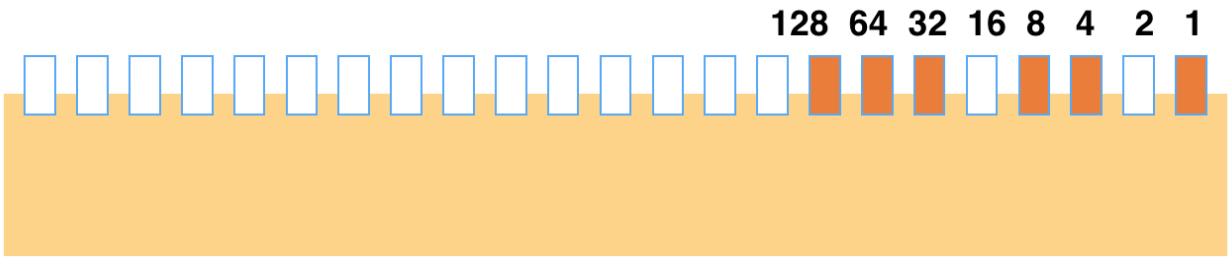
那如果来了4个人敌人，现在有两根狼烟都点着了只能表示3个人，表示4个人，做得到么？臣妾做不到啊～～～

不过还记得么？之前使用2根狼烟只能表示2个人，现在我们通过一些奇淫巧技是不是表达了4种状态(0,1,2,3)啦。。。

看看眼下这4个可恶的敌人吧，咱们用这两根狼烟已经装不下他们了，所以我们只好再点一根，同时我们还要灭掉前面的两根，因为第三根这一根狼烟就可以表示4个敌人



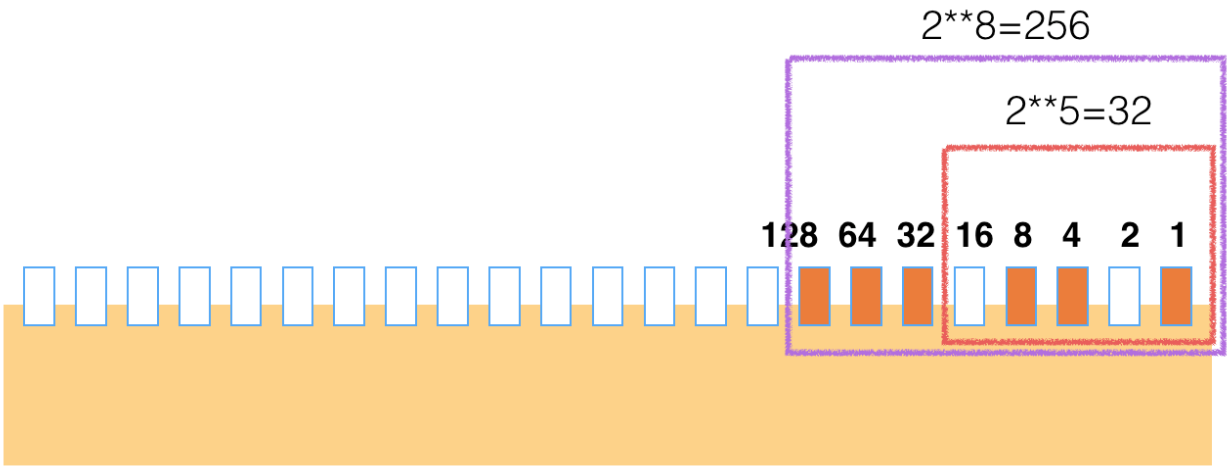
接下来我们以此类推，烟不够了就往后多点一根，最终就出现这样的情况了，各位算算这是多少敌人?



很简单，把红色柱子代表的值加起来就行了对吧,一共247个敌人。

到此，友军将领的变态需求终于满足啦。。。

最后补充下，算敌人个数时，你要把每个红柱子加起来，柱子越多，算的越慢。 其实有快速算法，你发现没有，每根柱子所代表的值 就是此柱及其前面柱子的多少次方。



好，同学们，快帮我算出来，如果来61352个敌人的话，狼烟如何排列?

4.2 二进制定义

二进制是计算技术中广泛采用的一种数制。二进制数据是用0和1两个数码来表示的数。它的基数为2，进位规则是“逢二进一”，由18世纪德国数理哲学大师莱布尼兹发现。当前的计算机系统使用的都是二进制系统，数据在计算机中主要是以补码的形式存储的。计算机中的二进制则是一个非常微小的开关，用“开”来表示1，“关”来表示0。

我们发现刚刚我们讲述的狼烟的故事和现在这个新理论出奇相似。假设狼烟点燃用1表示，狼烟灭掉用0表示，那么刚刚我们用狼烟表示百万雄师的理论就可以用在计算机上，这种表示数字的方式就叫做二进制。

你可能会觉得发明计算机的人思路轻奇，为什么要多此一举的用这种方式来表达数字，但事实上计算机不像我们这样智能，CPU是一个包含上亿个精巧的晶体管的芯片集合，晶体管表达感情的方式很简单，就是通过高低电压(有电没电)，低电压的时候表示0，高电压的时候表示1，因此最终能让计算机理解的就只有0和1而已。

4.3 二进制与十进制转换

其实刚刚在无形中我们已经将10进制转换成2进制了，现在我们要再总结一遍。

刚才我们已经发现，二进制的第n位代表的十进制值都刚好遵循着2的n次方这个规律

填位大法：

先把他们代表的值依次写出来，然后再根据10进制的值把数填到相应位置，就好了～～～

十进制转二进制方法相同，只要对照二进制为1的那一位对应的十进制值相加就可以了。

	128	64	32	16	8	4	2	1
20				1	0	1	0	0
200	1	1	0	0	1	0	0	0

4.4 二进制位运算

现在知道二进制如何转十进制了， 那你知道，计算机如何实现2个数之间的运算么?

程序中的所有数在内存中都是以二进制的形式储存的。位运算就是直接对整数在内存中的二进制位进行操作

下表中变量 a 为 60，b 为 13，二进制格式如下：

```
a = 0011 1100

b = 0000 1101
```

运算符	描述	实例
&	按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0（快记法：与运算， 位相乘）	(a & b) 输出结果 12 ，二进制解释： 0000 1100
|	按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1。（快记：或运算， 位相加）	(a | b) 输出结果 61 ，二进制解释： 0011 1101
^	按位异或运算符：当两对应的二进位相异时，结果为1（快记：位相减）	(a ^ b) 输出结果 49 ，二进制解释： 0011 0001
~	按位取反运算符：对数据的每个二进制位取反,即把1变为0,把0变为1 。（快记：~x=-(x+1)	(~a) 输出结果 -61 ，二进制解释： 1100 0011，在一个有符号二进制数的补码形式。
<<	左移动运算符：运算数的各二进位全部左移若干位，由 **<<** 右边的数字指定了移动的位数，高位丢弃，低位补0。	a << 2 输出结果 240 ，二进制解释： 1111 0000
>>	右移动运算符：把">>"左边的运算数的各二进位全部右移若干位，**>>** 右边的数字指定了移动的位数	a >> 2 输出结果 15 ，二进制解释： 0000 1111

> 按位取反深入:https://blog.csdn.net/Coder__CS/article/details/79186677?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_baidulandingword-8&spm=1001.2101.3001.4242

五、初识编码

5.1 文字来了-ASCII码

通过上一节讲的二进制的知识，大家已经知道计算机只认识二进制，生活中的数字要想让计算机理解就必须转换成二进制。十进制到二进制的转换只能解决计算机理解数字的问题，那么文字要怎么让计算机理解呢?

于是我们就选择了一种曲线救国的方式，既然数字可以转换成十进制，我们只要想办法把文字转换成数字，这样文字不就可以表示成二进制了么?

可是文字应该怎么转换成数字呢？就是强制转换啊，简单粗暴呀。 我们自己强行约定了一个表，把文字和数字对应上，这张表就相当于翻译，我们可以拿着一个数字来对比对应表找到相应的文字，反之亦然。

十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符
0		16	▶	32		48	0	64	@	80	P	96	`	112	p	128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
1	☺	17	◀	33	!	49	1	65	A	81	Q	97	a	113	q	129	ü	145	æ	161	í	177	▓	193	┴	209	╤	225	ß	241	±
2	☻	18	↕	34	"	50	2	66	B	82	R	98	b	114	r	130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
3	♥	19	‼	35	#	51	3	67	C	83	S	99	c	115	s	131	â	147	ô	163	ú	179	▓	195	├	211	╙	227	π	243	≤
4	♦	20	¶	36	$	52	4	68	D	84	T	100	d	116	t	132	ä	148	ö	164	ñ	180	▓	196	─	212	╘	228	Σ	244	⌠
5	♣	21	§	37	%	53	5	69	E	85	U	101	e	117	u	133	à	149	ò	165	Ñ	181	▓	197	┼	213	╒	229	σ	245	⌡
6	♠	22	▬	38	&	54	6	70	F	86	V	102	f	118	v	134	å	150	û	166	ª	182	▓	198	╞	214	╓	230	µ	246	÷
7	•	23	↨	39	'	55	7	71	G	87	W	103	g	119	w	135	ç	151	ù	167	º	183	▓	199	╟	215	╫	231	τ	247	≈
8	◘	24	↑	40	(	56	8	72	H	88	X	104	h	120	x	136	ê	152	ÿ	168	¿	184	▓	200	╚	216	╪	232	Φ	248	°
9	○	25	↓	41	)	57	9	73	I	89	Y	105	i	121	y	137	ë	153	Ö	169	⌐	185	▓	201	╔	217	┘	233	Θ	249	•
10	◙	26	→	42	*	58	:	74	J	90	Z	106	j	122	z	138	è	154	Ü	170	¬	186	▓	202	╩	218	┌	234	Ω	250	·
11	♂	27	←	43	+	59	;	75	K	91	[	107	k	123	{	139	ï	155	¢	171	½	187	▓	203	╦	219	▓	235	δ	251	√
12	♀	28	∟	44	,	60	<	76	L	92	\	108	l	124	|	140	î	156	£	172	¼	188	▓	204	╠	220	▄	236	∞	252	n
13	♪	29	↔	45	-	61	=	77	M	93	]	109	m	125	}	141	ì	157	¥	173	¡	189	▓	205	═	221	▌	237	φ	253	²
14	♫	30	▲	46	.	62	>	78	N	94	^	110	n	126	~	142	Ä	158	Pts	174	«	190	▓	206	╬	222	▐	238	∈	254	■
15	☼	31	▼	47	/	63	?	79	O	95	_	111	o	127	⌂	143	Å	159	ƒ	175	»	191	┐	207	╧	223	■	239	∩	255	ÿ

ASCII（American Standard Code for Information Interchange，美国信息交换标准代码）是基于拉丁字母的一套电脑编码系统，主要用于显示现代英语和其他西欧语言。它是现今最通用的单字节编码系统，并等同于国际标准ISO/IEC 646。

由于计算机是美国人发明的，因此，最早只有127个字母被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为 `ASCII` 编码，比如大写字母 `A` 的编码是 `65` ，小写字母 `z` 的编码是 `122` 。后128个称为扩展ASCII码, 主要是一些特殊字符。

那现在我们就知道了上面的字母符号和数字对应的表是早就存在的。那么根据现在有的一些十进制，我们就可以转换成二进制的编码串。

比如

```
一个空格对应的数字是0            翻译成二进制就是0（注意字符'0'和整数0是不同的）
一个对勾√对应的数字是251         翻译成二进制就是11111011
```

提问：假如我们要打印两个空格一个对勾 写作二进制就应该是 0011111011， 但是问题来了，我们怎么知道从哪儿到哪儿是一个字符呢?

论断句的重要性与必要性：

上次在网上看到个新闻，讲是个小偷在上海被捕时高喊道：“我一定要当上海贼王！”

正是由于这些字符串长的长，短的短，写在一起让我们难以分清每一个字符的起止位置，所以聪明的人类就想出了一个解决办法，既然一共就这255个字符，那最长的也不过是11111111八位，不如我们就把所有的二进制都转换成8位的，不足的用0来替换。

这样一来，刚刚的两个空格一个对勾就写作0000000000000000011111011，读取的时候只要每次读8个字符就能知道每个字符的二进制值啦。

在这里，每一位0或者1所占的空间单位为bit(比特)，这是计算机中最小的表示单位

每8个bit组成一个字节，这是计算机中最小的存储单位(毕竟你是没有办法存储半个字符的)orz～

```
bit            位，计算机中最小的表示单位
8bit = 1bytes  字节，最小的存储单位，1bytes缩写为1B
1KB=1024B
1MB=1024KB
1GB=1024MB
1TB=1024GB
1PB=1024TB
1EB=1024PB
1ZB=1024EB
1YB=1024ZB
1BB=1024YB
```

5.2 中文怎么搞?

英文问题是解决了， 我们中文如何显示呢？ 美国佬设计ASSCII码的时候应该是没考虑中国人有一天也能用上电脑， 所以根本没考虑中文的问题，上世界80年代，电脑进入中国，把砖家们难倒了，妈的你个一ASSCII只能存256个字符，我常用汉字就几千个，怎么玩？？？勒紧裤腰带还苏联贷款的时候我们都挺过来啦，这点小事难不到我们， 既然美帝的ASCII不支持中文，那我们自己搞张编码表不就行了， 于是我们设计出了GB2312编码表，长成下面的样子。一共存了6763个汉字。

code	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
B4A0	础	储	矗	搐	触	处	揣	川	穿	椽	传	船	喘	串	疮	
B4B0	窗	幢	床	闯	创	吹	炊	捶	锤	垂	春	椿	醇	唇	淳	纯
B4C0	蠢	戳	绰	疵	茨	磁	雌	辞	慈	瓷	词	此	刺	赐	次	聪
B4D0	葱	囱	匆	从	丛	凑	粗	醋	簇	促	蹿	篡	窜	摧	崔	催
B4E0	脆	瘁	粹	淬	翠	村	存	寸	磋	撮	搓	措	挫	错	搭	达
B4F0	答	瘩	打	大	呆	歹	傣	戴	带	殆	代	贷	袋	待	逮	

code	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
B5A0	怠	耽	担	丹	单	郸	掸	胆	旦	氮	但	惮	淡	诞	弹	
B5B0	蛋	当	挡	党	荡	档	刀	捣	蹈	倒	岛	祷	导	到	稻	悼
B5C0	道	盗	德	得	的	蹬	灯	登	等	瞪	凳	邓	堤	低	滴	迪
B5D0	敌	笛	狄	涤	翟	嫡	抵	底	地	蒂	第	帝	弟	递	缔	颠
B5E0	掂	滇	碘	点	典	靛	垫	电	佃	甸	店	惦	奠	淀	殿	碉
B5F0	叼	雕	凋	刁	掉	吊	钓	调	跌	爹	碟	蝶	迭	谍	叠	

code	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
B6A0	丁	盯	叮	钉	顶	鼎	锭	定	订	丢	东	冬	董	懂	动	
B6B0	栋	侗	恫	冻	洞	兜	抖	斗	陡	豆	逗	痘	都	督	毒	犊
B6C0	独	读	堵	睹	赌	杜	镀	肚	度	渡	妒	端	短	锻	段	断
B6D0	缎	堆	兑	队	对	墩	吨	蹲	敦	顿	囤	钝	盾	遁	掇	哆
B6E0	多	夺	垛	躲	朵	跺	舵	剁	惰	堕	蛾	峨	鹅	俄	额	讹

这个表格比较大，像上面的一块块的文字区域有72个，这导致通过一个字节是没办法表示一个汉字的(因为一个字节最多允许256个字符变种，你现在6千多个，只能2个字节啦，2**16=65535个变种)。

有了gb2312，我们就能愉快的写中文啦。

但我们写字竟然会出现中英混杂的情况，比如“我是黑姑娘，我的英文名叫Black Girl”， 这种你怎么办？这就要求你必须在gb2312里同时支持英文，但是还不能是2个字节表示一个英文字母。人家ASCII用一个字符，你用2个，那一个2mb大小的英文文档只要一改编码，就立刻变成4mb, 太坑爹，中国人你有钱也不能这么造呀。 所以中国砖家们又通过神奇手段兼容了ASSCII, 即遇到中文用2个字节，遇到英文直接用ASCII的编码。怎么做到的呢?

如何区别连在一起的2个字节是代表2个英文字母，还是一个中文汉字呢？ 中国人如此聪明，决定，**如果2个字节连在一起，且每个字节的第1位(也就是相当于128的那个2进制位)如果是1，就代表这是个中文，这个首位是128的字节被称为高字节。 也就是2个高字节连在一起，必然就是一个中文。**你怎么如此笃定？因为0-127已经表示了英文的绝大部分字符，128-255是ASCII的扩展表，表示的都是极特殊的字符，一般没什么用。所以中国人就直接拿来用了。

自1980年发布gb2312之后，中文一直用着没啥问题，随着个人电脑进入千家万户，有人发现，自己的名字竟然打印不出来，因为起的太生僻了。

于是1995年， 砖家们又升级了gb2312, 加入更多字符，连什么藏语、维吾尔语、日语、韩语、蒙古语什么的统统都包含进去了，国家统一亚洲的野心从这些基础工作中就可见一斑哈。 这个编码叫GBK，一直到现在，我们的windows电脑中文版本的编码就是GBK.

5.3 编码的战国时代

中国人在搞自己编码的同时，世界上其它非英语国家也得用电脑呀，于是都搞出了自己的编码，你可以想得到的是，全世界有上百种语言，日本把日文编到**Shift_JIS**里，韩国把韩文编到**Euc-kr**里，

各国有各国的标准，就会不可避免地出现冲突，结果就是，在多语言混合的文本中，显示出来会有乱码。之前你从玩个日本游戏，往自己电脑上一装，就显示乱码了。为什么电子邮件常常出现乱码？也是因为发信人和收信人使用的编码方式不一样。

这么乱极大了阻碍了不同国家的信息传递，于是 **ISO**（国际标谁化组织）的国际组织决定着手解决这个问题。他们采用的方法很简单：废了所有的地区性编码方案，重新搞一个包括了地球上所有文化、所有字母和符号 的编码！他们打算叫它”Universal Multiple-Octet Coded Character Set”，简称 **UCS**, 俗称 “**unicode**“。

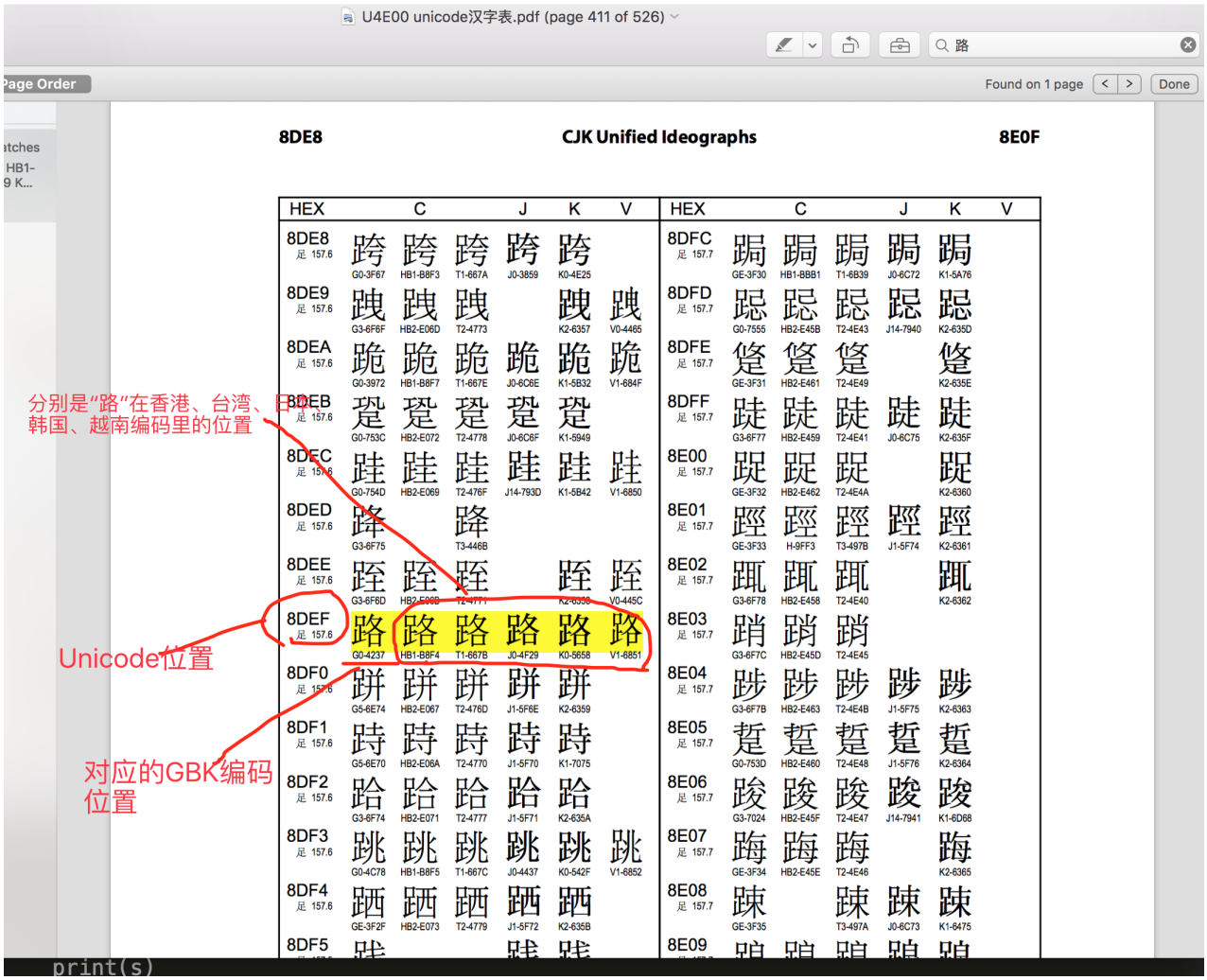
Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。Unicode 2-4字节 已经收录136690个字符，并还在一直不断扩张中…

Unicode标准也在不断发展，但最常用的是用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要**4**个字节）。

Unicode有2个特点：

1. 支持全球所有语言
2. 包含跟各种语言的映射关系，所以可以跟各种语言的编码自由转换，也就是说，即使你gbk编码的文字 ，想转成unicode很容易。

为何unicode可以跟其它语言互相转换呢？ 因为有跟所有语言都有对应关系哈，这样做的好处是可以让那些已经用gbk或其它编码写好的软件容易的转成unicode编码 ， 利于unicode的推广。 下图就是unicode跟中文编码的对应关系



5.4 大秦一统天下后的新问题

帝国(unicode)统一后，新的问题又出现了：如果统一成**Unicode**编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用**Unicode**编码比**ASCII**编码需要多一倍的存储空间，由于计算机的内存比较大，并且字符串在内容中表示时也不会特别大，所以内容可以使用unicode来处理，但是存储和网络传输时一般数据都会非常多，且互联网刚出现时，网速又慢，这个数据占用空间大一倍将是无法容忍的！！！

为了解决存储和网络传输的问题，出现了Unicode Transformation Format，学术名UTF，即：对unicode字符进行转换，以便于在存储和网络传输时可以节省空间!

- UTF-8： 使用1、2、3、4个字节表示所有字符；优先使用1个字符、无法满足则使增加一个字节，最多4个字节。英文占1个字节、欧洲语系占2个、东亚占3个，其它及特殊字符占4个
- UTF-16： 使用2、4个字节表示所有字符；优先使用2个字节，否则使用4个字节表示。
- UTF-32： 使用4个字节表示所有字符；

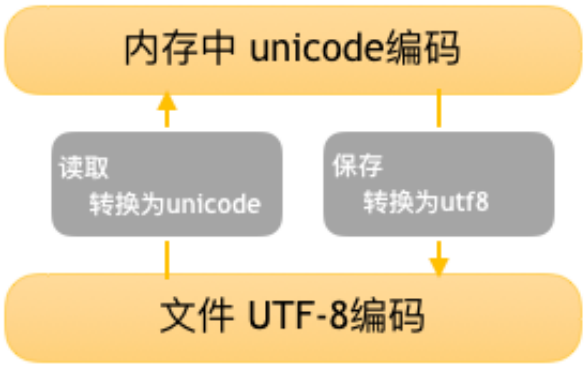
注意：**UTF 是为unicode编码 设计 的一种 在存储 和传输时节省空间的编码方案，它是Unicode的一种实现方式.**

如果你要传输的文本包含大量英文字符，用**UTF-8**编码就能节省空间：

字符	ASCII	Unicode	UTF-8
A	01000001	00000000 01000001	01000001
中	x	01001110 00101101	11100100 10111000 10101101

从上面的表格还可以发现，**UTF-8**编码有一个额外的好处，就是**ASCII**编码实际上可以被看成是**UTF-8**编码的一部分，所以，大量只支持**ASCII**编码的历史遗留软件可以在**UTF-8**编码下继续工作。

搞清楚了**ASCII**、**Unicode**和**UTF-8**的关系，我们就可以总结一下现在计算机系统通用的字符编码工作方式：



> unicode官网：https://home.unicode.org/
>
> unicode 中文表: http://www.chi2ko.com/tool/CJK.htm

5.4 常用编码介绍一览表

编码	制定时间	作用	所占字节数
ASCII	1967年	表示英语及西欧语言	8bit/1bytes
GB2312	1980年	国家简体中文字符集，兼容ASCII	2bytes
Unicode	1991年	国际标准组织统一标准字符集	2bytes
GBK	1995年	GB2312的扩展字符集，支持繁体字，兼容GB2312	2bytes
UTF-8	1992年	不定长编码	1-3bytes

5.5 PY2 VS PY3编程

Python 在89年刚生下来的时候，还没有Unicode呢，所以龟叔选用的默认编码只能是ASCII, 一真到py2.7，用的还是ASCII, 导致Py默认只支持英文，想支持其它语言，必须单独配置。

`py2_test.py`

```
name = "路飞学城"
print(name)
```

执行报错：

```
alex@Alexander-MBP day5 % python py2_test.py
  File "py2_test.py", line 2
SyntaxError: Non-ASCII character '\xe8' in file py2_test.py on line 2, but no encoding
declared; see http://python.org/dev/peps/pep-0263/ for details
```

py2写中文的正确姿势

在代码文件第一行，做声明，告诉py解释器，后面的代码是用utf-8编码的，你用utf-8编码来解析它

```
# -*- encoding:utf-8 -*-
name = "路飞学城"
print(name)
```

不过注意如果你的电脑 是windows系统， 你的系统默认编码是GBK ,你声明的时候要声明成GBK, 不能是utf-8, 否则依然是乱码，因为gbk自然不认识utf-8.

在Py2里编码问题非常头疼，若不是彻底理解编码之间的各种关系，会经常容易出现乱码而不知所措。

到了Py3推出后,终于把默认编码改成了unicode, 同时文件存储编码变成了utf-8，意味着，不用任何声明，你就可以写各种语言文字在你的Python程序里。 从此，程序们手牵手过上了快乐的生活。

六、十六进制

6.1 什么是16进制

16进制，英文名称Hexadecimal(简写Hex)， 在数学中是一种逢16进1的进位制。一般用数字0到9和字母A到F（或a~f）表示，其中:A~F表示10~15，这些称作十六进制数字，比如十进制13用16进制表示是D, 28用16进制是1C。

```
0 1 2 3 4 5 6 7 8 9  A  B  C  D  E  F
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

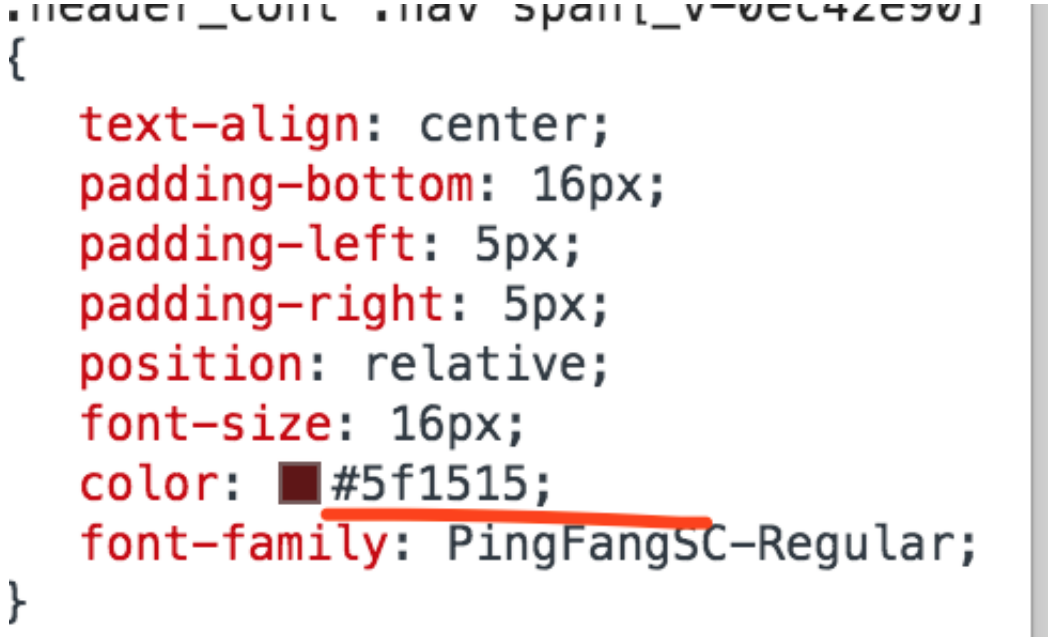
每个16进制，占4-bit, 所以要用16进制来表示一个字节的话，正好2个16进制数字就可以啦。

比如拿234这个数来算：

128	64	32	16	8	4	2	1	2进制
1	1	1	0	1	0	1	0	234
8	4	2	1	8	4	2	1	2个4bit
E				A				16进制

展示起来，更简洁，应该是16进制在计算机里被广泛应用的主要原因.

16进制在计算机领域应用普遍，常见的有html\css的颜色表、mac地址、字符编码等都用16进制来表示。 这是因为将4个位元（Bit）化成单独的16进制数字不太困难。1字节可以表示成2个连续的16进制数字, 也可以表示8个2进制数。这种混合表示法就容易令人混淆，因此需要一些字首、字尾或下标来显示这个值到底是16进制还是2进制 ， 所以在C语言、C++、Shell、Python、Java语言及其他相近的语言使用字首“0x”来标示16进制，例如“0x5A3”代表1443。



```
.header_cont .nav span[_v-0ec42e90]
{
    text-align: center;
    padding-bottom: 16px;
    padding-left: 5px;
    padding-right: 5px;
    position: relative;
    font-size: 16px;
    color: #5f1515;
    font-family: PingFangSC-Regular;
}
```

七、三元运算

又称三目运算，是对if...else...的一种简写

假设现在有两个数字，我们希望获得其中较大的一个，那么可以使用 if else 语句，例如：

```
if a>b:
  max = a
else:
  max = b
```

用三元运算的写法就是

```
max = a if a > b else b
```

上面语句的含义是：

- 如果 a>b 成立，就把 a 作为整个表达式的值，并赋给变量 max；
- 如果 a> b 不成立，就把 b 作为整个表达式的值，并赋给变量 max。

三元嵌套

Python 三元运算符支持嵌套，就可以构成更加复杂的表达式。在嵌套时需要注意 if 和 else 的配对，例如：

```
>>> a,b,c,d = 3,5,7,9
>>>
>>> a if a > b else c if c > d else d
9
```

应该理解为

```
 a if a > b else (c if c > d else d)
```

八、作业

8.1 三级菜单

数据源

```
menu = {
    '北京':{
        '海淀':{
            '五道口':{
                'soho':{},
                '网易':{},
                'google':{}
            },
            '中关村':{
                '爱奇艺':{},
                '汽车之家':{},
                'youku':{},
            },
            '上地':{
                '百度':{},
            },
        },
        '昌平':{
            '沙河':{
                '路飞学城':{},
```

```
                    '北航':{},
                },
                '天通苑':{},
                '回龙观':{},
            },
            '朝阳':{},
            '东城':{},
        },
        '上海':{
            '闵行':{
                "人民广场":{
                    '炸鸡店':{}
                }
            },
            '闸北':{
                '火车站':{
                    '携程':{}
                }
            },
            '浦东':{},
        },
        '山东':{},
    }
```

需求：

- 可依次选择进入各子菜单
- 可从任意一层往回退到上一层
- 可从任意一层退出程序

> 所需新知识点：列表、字典