最开始最直观的印象

1.定义一个变量时，不用 int a，或者 char c 这样，就直接 value = 123，不用声明类型，它的赋值机制与 c++不太一样，它其实是先开辟一个空间，存放这个数据后，value 相当于是一个引用。

还可以 x , y = 4 , 8 这样同时赋值。

2.
```
score = int(input("请输入分数:"))
if 90 <= score <= 100 :        #注意冒号
    print('A')                 #python 靠缩进来判断
elif 80 <= score < 90 :        #注意冒号
    print('B')
elif 60 <= score < 80 :        #注意冒号
    print('C')
elif 0 <= score < 60 :         #注意冒号
    print('D')
else :                             #注意冒号
    print("输入错误")
```

没有分号，没有分号，没有分号

if 、else 、for 、while 等后面有冒号，有冒号，有冒号

新增了 elif ，就相当于 else if

Python 不用｛｝来区分作用域，靠的是缩进，缩进，缩进

3. 除法　1/2 等于 0.5　1//2 等于 0

　幂方运算　2**5 等于 32

4.
```
for(i=0;i<4;i++)
{
    ......
}
```

在 Python 里为

```
for i in range(0,4) :
    ......
```

虽然 Python 是用 c 语言写的，但是不同之处还是有很多的，Python 更注重实用.....

一．数据类型

1.数字

2.字符串

3.元组

4.列表

5.字典

字符串、列表、元组都为序列类型的数据,序列就可以进行切片操作

1.数字

添加了复数类型，j 就相当于 i

```
>>> c=3.14
>>> type(c)
<class 'float'>
>>> d=3.14j
>>> type(d)
<class 'complex'>
>>> c+d
(3.14+3.14j)
```

没有 double,只有 float

2.字符串

（1）三重引号

```
>>> str1='''tom:
    hello world! '''
>>> str1
'tom:\n   hello world! '
>>> print (str1)
tom:
    hello world!
```

三重引号也可以用来多行注释

（2）索引

```
>>> str1="abcde"
>>> str1[0]
'a'
>>> str1[4]
'e'
>>> str1[5]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    str1[5]
IndexError: string index out of range
>>> str1[-1]    #-1 表示倒数第一个
'e'
>>> str1[-2]
'd'
>>> str1[0]+str1[-1]
```

```
31  'ae'
```

（3）切片操作

```
32  >>> str1[1:4]      #序列类型均有的切片操作
 'bcd'
```

（4）相关方法

操作	含义
+	连接
*	重复
<string>[]	索引
<string>[:]	切片
len(<string>)	求长度
<string>.upper()	将字符串中所有字母大写(但原字符串没有改变)
<string>.lower()	将字符串中所有字母小写
<string>.strip()	去两边空格及去指定字符
<string>.split()	按指定字符分割字符串为数组
<string>.join()	连接两个字符串序列
<string>.find()	搜索指定字符串
<string>.replace()	字符串替换
for <var> in <string>	字符串迭代

3.元组（tuple）

```
>>> t=123,
>>> type(t)
<class 'tuple'>
>>> t=123,456,'abc'   #t=（123,456,'abc'）括号可有可无
>>> type(t)
<class 'tuple'>
>>> t[2]
'abc'
>>> tl=789,234,t     #一个元组也可以作为另一个元组的一个元素
>>> tl[1]
234
>>> tl[2]
(123, 456, 'abc')
>>> tu=('a',345,(3.14,'b'))      #元组作为另一个元组的元素时要加括号避免歧义
```

```
>>> tu[2]
(3.14, 'b')
>>> tu[1:]
(345, (3.14, 'b'))        #元组属于序列的一种同样有切片操作
>>> tu*2
('a', 345, (3.14, 'b'), 'a', 345, (3.14, 'b'))
>>> tu
('a', 345, (3.14, 'b'))
```

元组定义后不能更改，也不能删除

与字符串一样，元组之间可以使用+号和*进行运算

4.列表（list）

列表与元组类似，但是列表可以随时修改

（1）列表相关方法

操作	含义
<list>.append(x)	将元素增加到列表的最后
<list>.sort()	将列表元素排序
<list>.reverse()	将列表元素反转
<list>.index(x)	返回第一次出现元素 x 的索引值
<list>.insert(i,x)	在位置 i 处插入新元素 x
<list>.count(x)	返回元素 x 在列表中的数量
<list>.remove(x)	删除列表中第一次出现的元素 x
<list>.pop(i)	取出列表中位置 i 的元素，并删除它

5.字典 (dict)

dict = {key1:value1,key2:value2.....}

key 应为一个常量

```
>>> d = {name:'123','age':20,3:456}
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    d = {name:'123','age':20,3:456}
NameError: name 'name' is not defined
>>> name = 'duan'
>>> dic = {name:'123','age':20,3:456}
>>> dic
{'duan': '123', 'age': 20, 3: 456}
```

```
>>> dic[name]
'123'
>>> name = 'lili'
>>> dic[name]
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    dic[name]
KeyError: 'lili'

>>> for i in dic:
        print (i,dic[i])


duan 123
age 20
3 456

>>> del dic[20]
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    del dic[20]
KeyError: 20
>>> del dic['age']
>>> dic
{'duan': '123', 3: 456}
>>> num = dic.pop(3)
>>> num
456

>>> dic['gender']='female'
>>> dic
{'duan': '123', 'gender': 'female'}
>>> dic.clear()
>>> dic
{}

>>> dic = {'duan': '123', 3: 456}
>>> str(dic)
"{'duan': '123', 3: 456}"
>>> type(dic)
<class 'dict'>
>>> num = dic.get(3,'error')
>>> num
456
>>> num = dic.get(4,'error')
>>> num
'error'
>>> dic.get(4,'error')
'error'
```

二．函数

基本形式

```
33  def add(x, y):
34      """Add two numbers"""
35      a = x + y
36      return a
```

因为 python 不用定义数据类型，所以函数最开始也不用写返回值类型，直接以 def 开头

与 c++的不同之处：

1.可以用关键词传参数

```
def fun(x=0, y=1, z=2):
    print(x, y, z)

fun(y=3)
```

```
0 3 2
Process finished with exit code 0
```

2.传入元组

```
def fun(x=0, y=1, z=2):
    print(x, y, z)

t = (4, 5, 6)
fun(*t)
```

```
4 5 6
Process finished with exit code 0
```

注意星号不能少，否则报错，且只有一个星号

2.传入字典

```
def fun(x=0, y=1, z=2):
    print(x, y, z)

d = {'y': 7, 'z': 8}
fun(**d)
```

```
0 7 8
Process finished with exit code 0
```

传字典需两个星号，且注意为‘y’，‘z’不能是y，z，因为之前说过key只能是常量，不能是变量

4.返回多个值

```
def fun(x, y):
    return y, x

num1 = 3
num2 = 5
num1, num2 = fun(num1, num2)      #实现交换
print(num1, num2)
```

```
5 3
Process finished with exit code 0
```

5.接受不定参数

```
def fun(x, *t):
    print(x)
    print(t)

fun(1, 2)
fun(3, 4, 5, 6)
```

```
1
(2,)
3
(4, 5, 6)
Process finished with exit code 0
```

多余参数相当于存放在元组t中

```
def fun(x, **d):
    print(x)
    print(d)

fun(1, a=2, b=3)
```

```
1
{'a': 2, 'b': 3}
Process finished with exit code 0
```

以关键词形式传入参数并存放在字典中

```
def fun(x, *args, **kwargs):
    print(x, args, kwargs)

fun(1, a=2, b=3, 4, 5)
```

```
  File "F:/Python/fun", line 4
    fun(1, a=2, b=3, 4, 5)                    ^
SyntaxError: positional argument follows keyword argument

Process finished with exit code 1
```

传入数据顺序应先按顺序赋值，多余的放在元组中，最后才能用关键字的形式传值，顺序不能乱。

```
def fun(x, *args, **kwargs):
    print(x, args, kwargs)

fun(1, 4, 5, a=2, b=3, x=8)
```

```
Traceback (most recent call last):
  File "F:/Python/fun", line 4, in <module>
    fun(1, 4, 5, a=2, b=3, x=8)
TypeError: fun() got multiple values for argument 'x'

Process finished with exit code 1
```

注意以关键字传参时不要和顺序传参的参数名重复

6.lambda 函数（匿名函数）

```
f = lambda x: x**3        # 相当于 def f(x):
                          #          return x**3

print(f(2))
```

```
8

Process finished with exit code 0
```

三．模块和包

1.模块（module）

任何一个.py 文件都可以作为一个模块，并使用 import 加载使用它（有点相当于 c++中的#include 一个文件，include 之后，这个文件里的函数以及数据成员都可以使用了），python 在寻找 import 文件时先从当前工作目录下寻找，没有的话再在安装 python 时自带的许多.py 文件中寻找，所以这也要求我们给文件命名时最好不要和自带的文件重名（如 random 等），否则会导致用不了官方提供的模块。

#module1.py

```
def Sum(lst):       #列表所有元素求和
    sum = 0
    for i in lst:
        sum = sum + i
    return sum


def isPrime(num):
    if num == 1:
        print("该数不是素数")
    else:
        for i in range(2, num):
            if num % i == 0:
                print("该数不是素数")
                break
        else:  # 当循环被 break 结束时, else 不执行;当循环正常结束时,即没有被 break, else 被执行
            print("该数是素数")

if __name__ == '__main__':
    isPrime(3)
```

```
该数是素数

Process finished with exit code 0
```

注意 name 属性，if __name__ == '__main__': 这一句代表只有当这个文件当作主文件被执行

时才会执行 isPrime(3)

使用 module1（方法一）

```
import module1
lst1 = [1, 2, 3, 4]
module1.Sum(lst1)
10
Process finished with exit code 0
```

注意使用时要用 module1.Sum(lst1)，不能直接 Sum（lst1），并且注意这里并没有执行 isPrime(3)，因为这里是将 module 导入作为模块使用，并不是主文件

使用 module1（方法二）

```
from module1 import Sum
lst1 = [1, 2, 3, 4]
Sum(lst1)
module1.isPrime(4)
10
Traceback (most recent call last):
  File "F:/Python/usemodule1.py", line 4, in <module>
    module1.isPrime(4)
NameError: name 'module1' is not defined

Process finished with exit code 1
```

注意这里 from module1 import Sum，相当于只导入了 module1 中的 Sum 函数，就算写 module1.isPrime(4)还是会报错，直接写 isPrime(4)照样会报错

```
from module1 import *
lst1 = [1, 2, 3, 4]
Sum(lst1)
isPrime(4)
10
该数不是素数
Process finished with exit code 0
```

这样写是将 module1 中的所有函数都导入了，且不用加 module1.，

```
from module1 import *

def Sum(x):
    print(x)

lst1 = [1, 2, 3, 4]
Sum(lst1)
isPrime(4)
```

```
[1, 2, 3, 4]
该数不是素数
Process finished with exit code 0
```

但如果和 module1 中的函数重名的话，module1 中的函数会被覆盖

使用 module1（方法三）

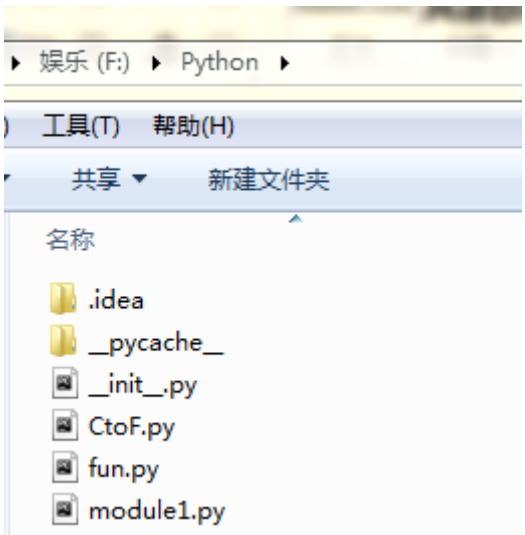
```
import module1 as m

lst1 = [1, 2, 3, 4]
m.Sum(lst1)
m.isPrime(4)
```

```
10
该数不是素数
Process finished with exit code 0
```

2.包

创建一个文件夹，在该文件夹下包含一个__init__.py 文件（这个文件内容可以为空），那么这个文件夹就为一个包了，该文件夹的名字即为包的名字。



所以该文件夹就为一个包了，文件夹下的所有.py 文件都可以作为一个模块导入，不仅仅局限于 module1

但当我们用时不能在该文件夹下创建.py 文件再 import Pyhton.module1 这样会报错，必须在这个文件夹**平级**的目录下才行

```
import Python.module1

lst1 = [2, 3, 4, 5]
Python.module1.Sum(lst1)
Python.module1.isPrime(5)
```

```
14
该数是素数

Process finished with exit code 0
```

注意还是要加 Python.module1. ， 不然会报错

四．正则表达式

正则表达式是由字符和操作符构成

4.1 正则表达式常用操作符

操作符	说明	实例
.	表示任何单个字符（除换行符外）	
[ ]	字符集，对单个字符给出取值范围	[abc]表示 a、b、c，[a-z]表示 a 到 z 单个字符
[^]	非字符集，对单个字符给出排除范围	[^abc]表示非 a 或 b 或 c 的单个字符
*	前一个字符 0 次或无限次扩展	abc*表示 ab、abc、abcc、abccc 等
+	前一个字符 1 次或无限次扩展	abc+表示 abc、abcc、abccc 等
?	前一个字符 0 次或 1 次扩展	abc?表示 ab、abc
|	左右表达式任意一个	abc|def 表示 abc、def
{m}	扩展前一个字符 m 次	ab{2}表示 abbc
{m,n}	扩展前一个字符 m 至 n 次	ab{1,2}表示 abc、abbc
{:n}	扩展前一个字符 0 至 n 次	a{:3}b 表示 b、ab、aab、aaab
^	匹配字符串开头	^abc 表示 abc 且在一个字符串的开头
$	匹配字符串结尾	abc$表示 abc 且在一个字符串的结尾
()	分组标记，内部只能使用|操作符	(abc)表示 abc，(abc|def)表示 abc、def
\d	数字，等价于[0-9]	
\w	单词字符，等价于[A-Za-z0-9]	

4.2 经典正则表达式实例

^[A-Za-z]+$	表示 26 个字母组成的字符串（不含数字或其他）
^[A-Za-z0-9]+$	表示 26 个字母和数字组成的字符串
^-?\d+$	表示整数形式的字符串
^[0-9]*[1-9][0-9]*$	表示正整数形式的字符串（排除了 00……0）
[1-9]\d{5}	表示中国境内邮政编码形式，6 位（首位不能为 0）
[\u4e00-\u9fa5]	匹配中文字符（utf-8 编码）
\d{3}-\d{8}|\d{4}-\d{7}	表示国内电话号码形式，如 010-12345678
[1-9]?\d	表示 0-99
1\d{2}	表示 100-199

ip 地址如何表示？（ip 地址分 4 段，每段 0-255）

```
(([1-9]?\d|1\d{2}|2[0-4]\d|25[0-5]).){3}([1-9]?\d|1\d{2}|2[0-4]\d|25[0-5])
```

4.3 re 库主要功能函数

函数	说明
re.search()	在一个字符串中搜索匹配正则表达式的第一个位置，返回 match 对象
re.match()	从一个字符串的开始位置起匹配正则表达式，返回 match 对象
re.findall()	搜索字符串以列表类型返回全部能匹配的子串
re.split()	将一个字符串按照正则表达式匹配结果进行分割，返回列表类型
re.finditer()	搜索字符串，返回一个匹配结果的迭代类型，每个迭代元素是 match 对象
re.sub()	在一个字符串中替换所有匹配正则表达式的子串，返回替换后的字符串
re.compile()	将正则表达式的字符串形式编译成正则表达式对象

```
import re
r = r'\d{3}-\d{8}'
match = re.search(r, '123,010-12345678,027-12345678,')
print(type(match))
if match:
    print(match.group())
else:
    print("not match")
```

```
<class '_sre.SRE_Match'>
010-12345678

Process finished with exit code 0
```

从给定字符串中寻找第一个匹配正则表达式的位置，返回 match 对象，用 match.group() 取得数据

```
import re
r = r'\d{3}-\d{8}'
match = re.match(r, '123,010-12345678,027-12345678,')
print(type(match))
if match:
    print(match.group())
else:
    print("not match")
```

```
<class 'NoneType'>
not match

Process finished with exit code 0
```

re.match()从字符串的第一个位置开始匹配（就是从数字 1 开始），因为之后的不匹配，所以返回的是 NoneType

```
>>> import re
>>> re.split(r'[1-9]\d{5}','abc 123456shu 34 430080')
['abc ', 'shu 34 ', '']
```

```
>>> type(re.finditer(r'[1-9]\d{5}','abc 123456shu 34 430080'))
<class 'callable_iterator'>
>>> re.finditer(r'[1-9]\d{5}','abc 123456shu 34 430080')
<callable_iterator object at 0x02704C10>
>>> for m in re.finditer(r'[1-9]\d{5}','abc 123456shu 34 430080'):
        print(m.group())


123456
430080
```

```
>>> re.sub(r'[1-9]\d{5}','000000','abc 123456shu 34 430080')
'abc 000000shu 34 000000'
```

```
>>> type(re.compile(r'[1-9]\d{5}'))
<class '_sre.SRE_Pattern'>
>>> pat = re.compile(r'[1-9]\d{5}')
>>> match = pat.search('abc 123456shu 34 430080')
>>> print(match.group())
123456
```

re.complie(r’[1-9]\d{5}’)是将里面的字符串编译成一个正则表达式对象（即 Pattern 对象），然后这个 Pattern 对象可以调用上述的 6 个函数

4.4 Match 对象

4.4.1 Match 对象的属性

属性	说明
.string	待匹配的文本
.re	匹配时使用的 patter 对象（正则表达式）
.pos	正则表达式搜索文本的开始位置
.endpos	正则表达式搜索文本的结束位置

```
>>> pat = re.compile(r'[1-9]\d{5}')
>>> match = pat.search('abc 123456shu 34 430080')
>>> print(match.group())
123456
>>>
>>>
>>> match.string
'abc 123456shu 34 430080'
>>> match.re
re.compile('[1-9]\\d{5}')
>>> match.pos
0
>>> match.endpos
23
```

4.4.2 Match 对象的方法

方法	说明
.group(0)	获得匹配后的字符串
.start()	匹配字符串在原是字符串的开始位置
.end()	匹配字符串在原是字符串的结束位置
.span()	返回(.start(),.span())

```
>>> match.group(0)
'123456'
>>> match.start()
4
>>> match.end()
10
>>> match.span()
(4, 10)
```

4.5 match 的贪婪匹配和最小匹配

```
>>> match = re.search(r'PY.*N','PYANBNCNDN')
>>> match.group(0)
'PYANBNCNDN'
```

按正则表达式来看‘PYAN’、‘PYANBN’、‘PYANBNCN’、‘PYANBNCNDN’都可以，但是输出的是‘PYANBNCNDN’，说明 re 默认的是贪婪匹配，也就是会输出最长的那个。

那么如何实现最小匹配呢?
可以在加上?

```
>>> match = re.search(r'PY.*?N','PYANBNCNDN')
>>> match.group(0)
'PYAN'
```

最小匹配操作符

操作符	说明
*?	前一个字符 0 次或无限多次扩展的最小匹配
+?	前一个字符 1 次或无限多次扩展的最小匹配
??	前一个字符 0 次或 1 次扩展的最小匹配
{m,n}?	前一个字符 m 次或 n 次扩展的最小匹配

只要长度输出可能不同的，都可以通过在操作符后添加？变成最小匹配

```
>>> match = re.search(r'abc+','abccccc')
>>> match.group(0)
'abccccc'
>>> match = re.search(r'abc+?','abccccc')
>>> match.group(0)
'abc'
```

```
>>> match = re.search(r'abc?','abccccc')
>>> match.group(0)
'abc'
>>> match = re.search(r'abc??','abccccc')
>>> match.group(0)
'ab'
```

```
>>> match = re.search(r'abc{1,3}','abccccc')
>>> match.group(0)
'abccc'
>>> match = re.search(r'abc{1,3}?','abccccc')
>>> match.group(0)
'abc'
```

五．异常处理

Python标准异常总结

AssertionError	断言语句（assert）失败
AttributeError	尝试访问未知的对象属性
EOFError	用户输入文件末尾标志EOF（Ctrl+d）
FloatingPointError	浮点计算错误
GeneratorExit	generator.close()方法被调用的时候
ImportError	导入模块失败的时候
IndexError	索引超出序列的范围
KeyError	字典中查找一个不存在的关键字
KeyboardInterrupt	用户输入中断键（Ctrl+c）
MemoryError	内存溢出（可通过删除对象释放内存）
NameError	尝试访问一个不存在的变量
NotImplementedError	尚未实现的方法
OSError	操作系统产生的异常（例如打开一个不存在的文件）
OverflowError	数值运算超出最大限制

ReferenceError	弱引用（weak reference）试图访问一个已经被垃圾回收机制回收了的对象
RuntimeError	一般的运行时错误
StopIteration	迭代器没有更多的值
SyntaxError	Python的语法错误
IndentationError	缩进错误
TabError	Tab和空格混合使用
SystemError	Python编译器系统错误
SystemExit	Python编译器进程被关闭
TypeError	不同类型间的无效操作
UnboundLocalError	访问一个未初始化的本地变量（NameError的子类）
UnicodeError	Unicode相关的错误（ValueError的子类）
UnicodeEncodeError	Unicode编码时的错误（UnicodeError的子类）
UnicodeDecodeError	Unicode解码时的错误（UnicodeError的子类）
UnicodeTranslateError	Unicode转换时的错误（UnicodeError的子类）
ValueError	传入无效的参数
ZeroDivisionError	除数为零

```
>>> 1+'1'
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    1+'1'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
>>> try:
        a = 1 + '1'
        print("----------------")
except TypeError as reason:
        print("This is a TypeError:",reason)


This is a TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
>>> try:
        num = 2 / 0
        a = 1 + '1'
except ZeroDivisionError as reason:
        print("This is a ZeroDivisonError:",reason)
except TypeError as reason:
        print("This is a TypeError:",reason)


This is a ZeroDivisonError: division by zero
```

```
>>> try:
        num = 2 / 0
        a = 1 + '1'
except TypeError as reason:
        print("This is a TypeError:",reason)


Traceback (most recent call last):
  File "<pyshell#24>", line 2, in <module>
    num = 2 / 0
ZeroDivisionError: division by zero
```

```
>>> try:
        num = 2 / 0
        a = 1 + '1'
except TypeError as reason:
        print("This is a TypeError:",reason)
finally:
        print("--------------------")


--------------------
Traceback (most recent call last):
  File "<pyshell#30>", line 2, in <module>
    num = 2 / 0
ZeroDivisionError: division by zero
```

```
>>> try:
        num = 2 / 0
        a = 1 + '1'
except TypeError as reason:
        print("This is a TypeError:",reason)
except ZeroDivisionError as reason:
        print("This is a ZeroDivisonError:",reason)
finally:
        print("--------------------")


This is a ZeroDivisonError: division by zero
--------------------
```

```
>>> raise ZeroDivisionError("除数为零的异常")
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    raise ZeroDivisionError("除数为零的异常")
ZeroDivisionError: 除数为零的异常
```

```
>>> try:
        print(int("123"))
except ValueError as reason:
        print("出错啦：", reason)
else:
        print("没有任何异常")
finally:
        print("执行完毕")


123
没有任何异常
执行完毕
```

```
>>> try:
        print(int("abc"))
except ValueError as reason:
        print("出错啦：", reason)
else:
        print("没有任何异常")
finally:
        print("执行完毕")


出错啦： invalid literal for int() with base 10: 'abc'
执行完毕
```

六．文件操作

```
"r"  以读方式打开，只能读文件 ， 如果文件不存在，会发生异常

"w" 以写方式打开，只能写文件， 如果文件不存在，创建该文件

                        如果文件已存在，先清空，再打开文件

"rb"  以二进制读方式打开，只能读文件 ， 如果文件不存在，会发生异常

"wb" 以二进制写方式打开，只能写文件， 如果文件不存在，创建该文件

                        如果文件已存在，先清空，再打开文件

"rt"  以文本读方式打开，只能读文件 ， 如果文件不存在，会发生异常

"wt" 以文本写方式打开，只能写文件， 如果文件不存在，创建该文件

                        如果文件已存在，先清空，再打开文件

"rb+"  以二进制读方式打开，可以读、写文件 ， 如果文件不存在，会发生异常

"wb+" 以二进制写方式打开，可以读、写文件， 如果文件不存在，创建该文件
```

七．面向对象编程

1.类的定义

```
>>> class Ball:
        def SetName(self,name):
                self.name = name
        def kick(self):
                print("kick ball %s" % self.name)


>>> ball = Ball()
>>> ball.SetName("123")
>>> ball.kick()
kick ball 123
>>> ball1 = Ball
>>> ball1.kick()
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    ball1.kick()
TypeError: kick() missing 1 required positional argument: 'self'
>>> ball1 = Ball()
>>> ball1.kick()
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    ball1.kick()
  File "<pyshell#5>", line 5, in kick
    print("kick ball %s" % self.name)
AttributeError: 'Ball' object has no attribute 'name'
```

注意：1.定义 class 里面的函数时，第一个参数一定为 self（就相当于 c++中的 this 指针）不加不行，如下

```
>>> class A:
        def show():
                print("123")


>>> a1 = A()
>>> a1.show()
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    a1.show()
TypeError: show() takes 0 positional arguments but 1 was given
```

2.实例化一个对象时，一定要 a1 = A()，括号不能掉，但是定义的时候 class A：这里是没有括号的

```
>>> class Ball:
        def __init__(self,name):
                self.name = name
        def __init__(self,name,size):
                self.name = name
                self.size = size
        def show(self):
                print("Ball: %s,Size:%s" %(self.name,self.size))


>>> ball = Ball("A")
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    ball = Ball("A")
TypeError: __init__() missing 1 required positional argument: 'size'
>>> ball = Ball("A","big")
>>> ball.show()
Ball: A,Size:big
```

注意：1.当一个对象被创建时，类定义里的左右双下划线的函数是被自动调用的，如__init__就相当于c++中的构造函数

2.从以上例子可以看出，python 不支持函数重载：

```
>>> class Ball:
        def __init__(self,name,size):
                self.name = name
                self.size = size
        def __init__(self,name):
                self.name = name
        def show(self,num):
                print("Ball: %s,Num: %d" %(self.name,num))
        def show(self):
                print("123")


>>> ball = Ball("A","big")
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    ball = Ball("A","big")
TypeError: __init__() takes 2 positional arguments but 3 were given
>>> ball.show()
123
```

他其实是后面的覆盖前面的，并且我们也不一定要用 c++的思维来看待这个问题，python 支持键值对的形式赋值，当我们给定默认参数时，可用键值对的形式赋值，也就没有函数重载的必要了

```
>>> class Ball:
        name = "ABC"
        def show(self):
                print(self.name)


>>> ball = Ball()
>>> ball.name
'ABC'
>>> ball.show()
ABC
>>>
>>> ball.name = '123'
>>> ball.show()
123
```

ball.name 可以直接访问或修改 Ball 的数据成员 name，说明 name 是共有的，但是 python 并没有 public 和 private 关键字，那么 python 如何区分公有和私有呢？

默认的数据成员和函数均为共有的
但当数据成员和函数的名称前有两个下划线，就变为私有的了

```
>>> class Ball:
        __name = "ABC"
        def show(self):
                print(self.__name)


>>> ball = Ball()
>>> ball.name
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    ball.name
AttributeError: 'Ball' object has no attribute 'name'
>>> ball.__name
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    ball.__name
AttributeError: 'Ball' object has no attribute '__name'
>>> ball.show()
ABC
```

注意不要写成 self.name，要与定义的名字一致，为__name
Python 使用“名字重整”方法，他其实是把前面带有两个下划线的成员名字进行了改动

```
>>> ball._Ball__name
'ABC'
>>> ball.__Ball__name
Traceback (most recent call last):
  File "<pyshell#89>", line 1, in <module>
    ball.__Ball__name
AttributeError: 'Ball' object has no attribute '__Ball__name'
```

Python 把前面带有两个下划线的成员名字改成了“_类名__变量名”的形式（前面为一个下划线），函数也是如此，如下：

```
>>> class Ball:
        __name = "ABC"
        def __show(self):
                print(self.__name)


>>> ball = Ball()
>>> ball._Ball__name
'ABC'
>>> ball._Ball__show()
ABC
```

八．应用小程序

（1）温度转换

```
temperature = input("请输入温度及单位：")
if temperature[-1] == ('c' or 'C') :   #注意括号一定要打，比较操作符比逻辑操作符优先级高
    tran = float(temperature[0:-1]) * 1.8 +32
    print ("转换成华氏温度为：",tran, 'F')
elif temperature[-1] in （'f','F'） :  #上面的和这种写法都可以
    tran = ( float(temperature[0:-1]) - 32 ) / 1.8
    print ("转换成摄氏温度为:%.2fC"%tran)   #注意这里
else :
    print ("输入有误")
```

（2）猴子吃桃问题

```
num = 1
for i in range (1,6) :
    num = (num+1)*2
print(num)

n = 1
for i in range (5,0,-1) :
    n = (n+1)<<1
print(n)

```

九. 附录

1.Python 3.x 保留字列表（34 个）

and	or	not	if
else	elif	for	in
break	continue	True	False
def	return	del	while
global	class	is	from
import	raise	as	assert
except	finally	lambda	nonlocal
with	from	yield	pass
None	try		

2.字符串处理方法