Fine 语言使用手册

前言

欢迎来到 fine 的世界！

这里只有数、字符串和二进制数据，三种数据：

不管是整数、小数、科学记数法，你都可以按照小学老师教的书写方法，输入数字。不用担心出错，fine 能领会你的意图、正确处理各类数据。

对于字符串，不论是单个字符、或是多个字符（汉字），fine 都把它们看做字符串，只要在输入字符串时，在其前后加上双引号（英式）即可。

二进制数据使用 Binary()内置函数输入，在文件操作、网络编程和串口收发时，会涉及到二进制文件或二进制数据操作。

由数字和字符串出发，fine 遵循“万物皆对象”的理念，简洁处理一切事物。

Fine 语言包含两个部分：编译器和虚拟机。你可以使用 fine 开发工具：编辑源文件（扩展名.fin 文件）、编译源文件（生成扩展名.fie 文件）、运行虚拟机（解释执行.fie 文件），得到你期望的结果。

Fine 的基础版本支持了：数字运算、字符串、列表、字典、函数、类与对象、文件处理、OS、模块加载、时间日期、多线程、人机交互（GUI）、网络编程、MySQL 数据库编程、Image 图片资源加载使用、Media 视频播放、绘图方法及指令集、串口通讯。

目前，已发布的版本经过大量测试，具备很好的稳定性，假如您在使用中发现问题，请提交到杭州发源软件有限公司官网(http://www.finelanguage.com.cn)的论坛上，或登录 https://gitee.com/birdzhaojd/fine-language 发表你的建议，我们将认真对待您的留言，定期升级更新，共同为 fine 的发展壮大努力！

视频教程参见抖音账号：dongfang445658

目录

第一章、开发工具安装及使用

1、 登录： http://www.finelanguage.com.cn

下载 fine 最新版本的安装包，双击安装程序后，默认安装位置为：C:\Program File(x86)\fine，安装后在桌面上会产生快捷键（fine），双击该快捷键，打开 fine 开发工具，开始代码编写、编译、运行。

2、 主菜单 File ->New　创建一个新的源文件

File ->Open ->New 创建新的源文件，或者

File ->Open　或 File ->Open->examples，选择一个已经存在的例程源文件

File ->Save　保存源文件（输入原文件名，扩展名必须为：.fin）

File ->Close　关闭当前打开的源文件

4、主菜单 Compile ->Compile　对源文件（扩展名.fin）进行编译，生成扩展名为 fie 的可执行文件，并自动保存在和源文件相同的目录下。如果编译过程出错，会提示并退出编译过程。

5、主菜单 Run ->Run　运行 finevirt.exe 虚拟机，对 fie 文件进行解释执行，如果执行过程出现错误，将提出错误信息并退出虚拟机。

断点设置：在点击 Run ->Run 菜单之前，在编辑器中，点击程序的行号（左边），此时，在行号区域会显示一个红圈，表示在该行设置断点，再次点击该行号，将取消断点。当程序执行到断点处时，程序将暂停执行，按任意键将继续执行下面的程序。最多限制设置三个断点。

断点清除：点击 Run ->Remove All Break Point 菜单，将取消所有已设置的断点。保存文件时，断点设置并不会被保存。

6、主菜单 ？ ->Manual，将弹出 fine 语言使用手册；

主菜单 ？ ->About，弹出 fineedit.exe 的版本号。

7、fine 语言在编译和运行过程中，都会主动检测代码错误，并提示错误代码的大致位置和错误类型。编译通过后，还需要尝试运行，进一步检测运行中可能的错误，最终获得可靠运行的程序。

8、注释符和续行符

fine 编辑器的“行注释符”为：“#”（其后的任何内容（本行内）均被 fine 编译器忽略）

fine 编辑器的“段略注释符”为：“/*　*/”，它们成对出现，在其中的所有内容，均被编译器忽略。

“行注释符”和“段略注释符”主要用于对代码的解释，便于今后理解记忆。

fine 编辑器的“续行符”为：“\” ，当一条语句太长，可以分成多行书写，只需插入续行符“\”，就可以另起一行继续书写。

注意事项：续行符之后必须立即换行，续行符之后不允许出现任何字符，包括空格。

9、如果需要用到数据库功能，请从本网站下载“mysql-5.7.38-winx64.rar”压缩文件，解压后，按照“mysql 安装教程.txt”教程安装 mysql 数据库服务程序，即可进行 mysql 编程。

10、fine 语言的源文件扩展名为：fin（例如 version.fin），编译产生的可执行文件的扩展名为：fie（例如 version.fie）。

11、当在 fineedit 开发工具上，完成代码编写、编译、运行，最终得到“字节码”文件（扩展名.fie），便可以脱离开发工具，直接双击你的可执行文件（例如 version.fie）。

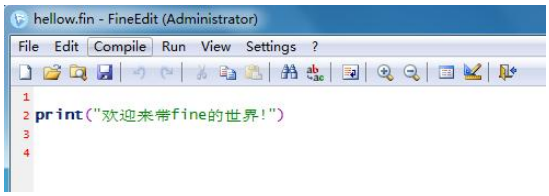
当双击扩展名为 fie 的文件时，Windows 会使用 finevirt.exe 解释“字节码”文件，实现程序功能。

第二章、fine 语言能干什么？

fine 能干什么？ 无所不能！还是让我们通过几个例程来体会。

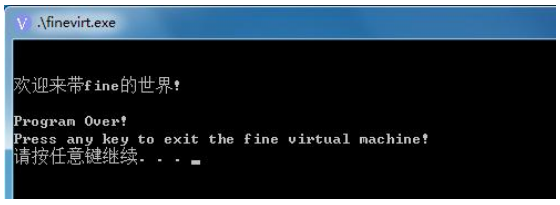
例程 1、写一个自己的程序 hellow.fin，在屏幕上显示：“欢迎来到 fine 的世界!”。

第一步，写代码——只有一行代码的程序!，如下：



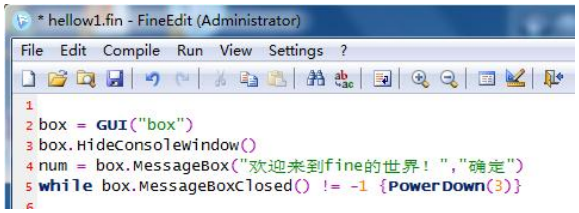
第二步，点击 File 保存文件(hellow.fin)之后，点击 Compile 编译，显示编译结果：

第三步，上图显示编译成功！再点击 Run 运行，就得到你的程序执行结果：

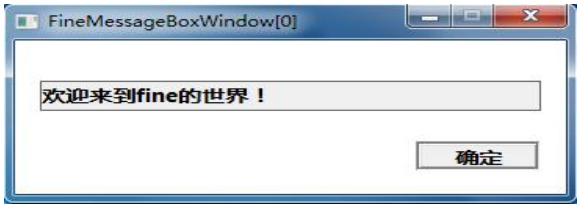


恭喜你！你已经正确的写出了你的以一个 fine 程序！

例程 2、如果你觉得不满意，希望写一个自己的窗口程序 hellow1.fin，在屏幕上创建自己的 Windows 窗口，运行结果显示在自建窗口上。3 行代码即可搞定！



编译成功后，点击 Run 运行，运行结果如下图：

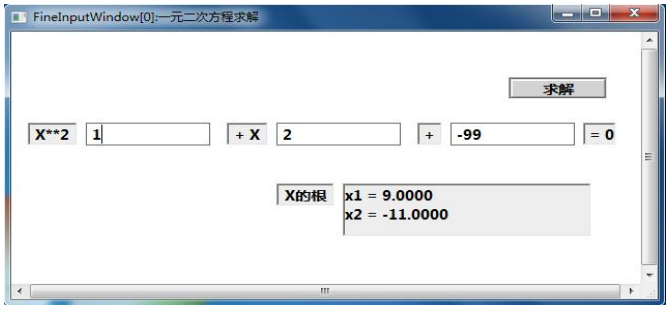


fine 如此简单！仅仅 3 行代码就是实现了自己的窗口，并将运行结果显示在自己的窗口上！

例程 3、如果你还不满意，觉得仅仅显示一句话太简单了，来点更复杂的！好吧，写一个求解一元二次方程的程序如何？solving_equation.fin 的代码如下：

```
# 例程16：《一元二次方程方程求解》
gui = GUI("fine")
m = MATH()
gui.HideConsoleWindow()
# 设计窗口界面
title = "一元二次方程求解"
size = [30,10,80,20]
edit1  = [" X**2 ","edit","f",9,6,15,1]
edit2  = [" + X ", "edit" ,"f",32,6,15,1]
edit3  = [" + ",   "edit","f",53,6,15,1]
text1  = [" = 0",  "text",74,6,0,1]
button = [" 求解 ","button",60,3,12,1]
text2  = [" X的根 ","text",40,10,30,3]
list = [title,size,edit1,edit2,edit3,text1,button,text2]
gui.Fine(list)
while gui.FineClosed() != -1
{
    PowerDown(3)
    if gui.FineReady() == -1 {continue}
    result = gui.FineRead()
    a = result[1]
    b = result[2]
    c = result[3]
    delt = b*b - 4*a*c
    if delt < 0
    {
        box = GUI("box")
        box.MessageBox("该方程无实数解！","确定")
        while box.MessageBoxClosed() != -1 {PowerDown(3)}
        continue
    }
    x1 = (-b + m.sqrt(delt))/(2*a)
    x2 = (-b - m.sqrt(delt))/(2*a)
    x1 = "x1 = " + sprint("%.4f",x1)
    x2 = "x2 = " + sprint("%.4f",x2)
    temp = x1 + "\n" + x2 + "\n"
    textlist = ["",temp]
    gui.SendText(textlist)
}
```

保存、编译成功后，点击运行，显示自己所设计的数据录入窗口，等待填写一元二次方程的参数 a、b、c，然后点击“求解”按钮，计算的结果显示在下面，如图：



solving_equation.fin 仅仅 40 行代码，我们就实现了一元二次方程求解的窗口程序，是否感觉 fine 确实 fine，简洁高效！

我知道，大家不会满足于此，希望看到网略编程、数据库编程等，更复杂的应用，那还是按部就班的、系统的学习后面的章节，相信用不了多久，大家都会成为 fine 编程“高手”！

Fine 语言在源程序书写上注意事项：

一、Fine 语言在书写源代码时，不需要行结束符（区别于 c 语言需要行结束符';'）。

二、Fine 语言在书写 if-elif-else（判断语句）、while（循环）、for（循环）、def（自定义函数声明）、class（自定义类声明）这些程序时，无需行结束符（区别于 Python 需要行结束符':'）。同时，在书写对应的程序体时，需要使用一对大括号将程序体包裹起来（区别于 Python 的行缩进法），在大括号内的程序体行缩进只是便于阅读，并不影响编译、运行。

三、Fine 语言在使用全局变量时，需要使用 global 声明，例如：

```
global  x
```

经 global 声明的的变量 x 的内容，在程序的任何地方都可以被修改，常用于不同模块之间、线程之间数据传递，同时，也谨慎使用，避免被错误修改。

上述声明语句仅仅声明了变量 x 是一个全局变量，并没有声明其数据类型，那么，在后续的程序中使用该全局变量是否会在成混乱呢？答案是不会造成混乱！当你需要使用全局变量 x 时，你可以直接向其赋值，Fine 虚拟机会智能记录其变量类型，并正确使用它。

四、除了全局变量需要声明（仅仅声明全局性）以外，Fine 语言并不需要对一般变量做特别声明，需要使用变量时，直接赋值即可，实际上 Fine 语言的赋值过程包含了变量类型申明。例如：

```
a = 1
b = 2
c = "Hello World!"
print("%     %\n", a+b,c)
```

上面一段程序中，使用了 a、b、c 三个变量，当虚拟机执行到前面两句时，会自动记录变量 a 和 b 为整数类型的数据，当执行第三行时，会自动记录变量 c 为字符串类型，当执行到第四句时，虚拟机会智能查询各个变量的类型，并根据其类型打印出正确的内容。

注意：第四句中%后面并没有指明要打印的变量类型，但是，程序是正确的，并不是书写错误，为什么没有像 c 语言或 Python 语言那样（需要使用 "%d　%s\n" ）特别指明被打印变量的类型呢？那是因为 Fine 语言的虚拟机知道所有变量的类型，虚拟机会根据变量类型做出正确的反应。

第三章、运算符号和赋值

第一节　初步认识 fine　例程《enter_fine.txt》

fine 的最底层数据有两种：数字、字符串。

数字：有包含整数、浮点数(小数、科学记数法)。程序员不用特别区分整数、小数、科学记数法，系统会自动识别。

字符串：不论是单个字符、或是多个字符，fine 都一视同仁地把它们当做字符串。在输入字符串时，仅要求在字符串的前后添加双引号，剩下的事情由 fine 帮你完成。

fine 支持的数字运算：

数字（浮点数）加、减、乘、除，乘方，求模

支持位操作（与、或、异或、非、左移、右移，位操作只适用于整数）

整数取值范围：-2**63　到 2**63 之间

浮点数范围：-4.940656458412465E-324 到-1.797693134862315E+308

```
#例程 1：数字输入和输出

a = 10                # 向变量 a 赋值，内容是数字 10
print("%\n",a)       # %是占位符，即，在显示时，将变量 a 的内容显示在占位符%的位置

a = 2.6               # 向变量 a 赋值，内容是数字 2.6，程序员不用关注是浮点数、或整数
print("%\n",a)       # fine 默认显示最大精度（2.600000000000000e+000）

a = 2.6               # 小数
print("%.2f\n",a)    # 占位符%后面的.2f 表明显示小数位数（这里显示 2 为小数）

a = 1.5e-25           # 科学记数法（e 或 E 等价，e、+、-前后不允许出现空格）
print("%\n",a)       # 显示 15 位小数

a = 1.23456789123E+200      # 科学记数法
print("%.5f\n",a)           # 显示 200 位整数和 5 位小数
print("%.5e\n",a)           # 显示科学记数法（保留 5 位小数）

a = 1.23456789123E-200      # 科学记数法
print("%.5f\n",a)           # 显示 0.00000
print("%.5e\n",a)           # 显示科学记数法（保留 5 位小数）

#   在系统内部小数也自动按科学记数法保存
#   在使用 print 显示时，默认输出为科学记数法
#   可以通过格式控制 "%.nf" 、或"%.ne"控制显示小数的位数
#   书写代码时，程序员正常书写即可，不必关心数字类型，系统会智能处理


# 例程 2：数字运算，对于数字运算的结果是整数还是浮点数，系统会智能处理，程序员不用关注。

b = 5
c = 4
d = -b*c+b*10    # -b 是取 b 的负数，它的优先级最高，先取 b 的负数，再乘以 c
print("%\n",d)    # 运算结果 d 为整数，因为参数均为整数，所涉及运算也不会产生浮点数

b = 5
c = 4
d = -b*c+b*10/2
print("%\n",d)    # 运算结果 d 为浮点数（除法运算后的数，默认为浮点数）

b = 5
c = 4
d = -b*c+b**3
print("%\n",d)    #运算结果 d 为浮点数（乘方运算的结果，默认为浮点数）

a = 5
b = 3
c = 2
```

```
d = a+5*(b<<1)**c    # 括号优先(左移 1 位，左移和右移操作符只适用于整数)
print("%.2f\n",d)      # 运算结果 d 为浮点数（乘方运算的结果，默认为浮点数）


#例程 3：浮点数取整、四舍五入取整

a = 8
b = 5
c = 4
d = -b*c+b*10/3+a
print("%\n",d)             # 运算结果 d 为浮点数

x = int(d)                 # 对 d 取整（丢弃小数部分）
print("%\n",x)             # x 为整数

y = int(d+0.5)             # 对 d 四舍五入取整
print("%\n",y)             # y 为整数

# 例程 4：字符串，字符串是被双引号""包含的一串字符组合，不能用单引号替代。
# 当字符串中出现 "\n" 时，它们被转义为一个换行字符
# 使用函数 len(str)计算字符串长度

a = "Hello world!"
print("%\n",a)          #  %是占位符，即，在显示的时候，变量 a 将显示在%的位置

b = len(a)              # 计算字符串 a 的长度，一个英文字符占一个字节，一个汉字占两个字节
print("%\n",b)          # 显示 12

a = "欢迎来自世界各国的人民 abc"
b = len(a)                          # 计算字符串 a 的长度，一个中文字占两个字节
print("%\n",b)                      # 显示 25


# 例程 5：取出字符串中指定位置的字符

a = "heLLo wORld!"
b = a[4]                            # 返回字符串 a 的第 4 个字符（从 0 开始计数），b = "o"
c = a[8]                            # 返回字符串 a 的第 8 个字符（从 0 开始计数），c = "R"
print("b = %       c = %\n",b,c)    # b 显示在第一个%的位置，c 显示在第二个%的位置


# 例程 6：两个字符串可以直接相加，结果是两个字符串拼接

a = "heLLo wORld!"
b = "世界，你好！ "
a = a + b                  # 将字符串 b 拼接到 a 的尾部（只支持字符串相加，不支持减法）
print("%\n",a)             # 字符串与数字之间不能进行加、减、乘、除等数学运算。


# 例程 7：支持两个字符串比较，比较比较运算有：<、<=、== 、!= 、>=  >。
# 不支持字符串与数字之间比较


a = "abc"
b = "ab"
if a > b
{
    print("%\n","a > b")
}
elif a == b
{
    print("%\n","a = b")
}
else
{
    print("%\n","a < b")
}


#例程 8：in   notin
a = "abcefghi"
b = "efg"
c = b in a             # 如果 b 存在于 a 中，返回 True，否则返回 False
print("%\n",c)       # 输出 True
c = "xy" in a        # 如果 xy 存在于 a 中，返回 True，否则返回 False
print("%\n",c)       # 输出 False
```

```
c = "xy" notin a
print("%\n",c)      # 输出 True


# 例程 8：列表（是一组数据的特定组合）

list = [1,2,"hello",3,"world",4,5]       # 列表赋值（中括号赋值），元素间用逗号隔开
print("%\n",list)


# 例程 9：列表

list = []                          # 列表赋值（定义一个空列表）
print("%\n",list)                  # 打印列表（此时为空列表）
list.append("hello")               # 向列表中添加一个字符串数据项
list.append(1)                     # 向列表中添加一个数字数据项
list.append(2)                     # 向列表中添加一个数字数据项
list.append("world")               # 向列表中添加一个字符串数据项
print("%\n",list)                  # 打印列表

list1 = []
list1.append(list)                 # 将列表 list 作为一个元素添加到列表 list1 中
list1.append("Welcome")
print("%\n",list1)                 # 输出列表


# 例程 10：list[n]取列第 n 项（从 0 计数），不改变原列表内容

a = [1,"hello",3,"world",4]
b = a[1]                           # 取出列表中位置 1 处的列表项，b = "hello"
print("%\n",b)
b = a[2]                           # 取出列表中位置 2 处的列表项，b = 3
print("%\n",b)
print("%\n",a)                     # 显示原列表 a


#例程 11：两个列表可以直接相加，结果是后面列表拼接在前面列表的尾部
a = [1,2,3,4]
b = [5,6,7]
c = a+b
print("%\n",c)              #   b 拼接在 a 后面，a 的内容变化，b 的内容不变
#   只支持两个列表相加，不支持减法；
#   列表与数字之间不能进行加、减、乘、除等数学运算。


# 例程 12：字典（是数据的另一种特定组合）

a = {"a":1,"b":2,"world":"世界","hello":"你好"}  # 字典赋值（大括号字典赋值）
print("%\n",a)


#例程 13：字典

a = {}                             #  字典赋值（定义一个空字典）
print("%\n",a)
a.setitem("a",1)                   # 向字典中添加词条
a.setitem("b",2)                   # 向字典中添加词条
a.setitem("world","世界")
a.setitem("hello","你好")
print("%\n",a)


# 例程 14：dict[KEY] ——取字典下标（由关键字取词条内容）

a = {"a":1,"b":2,"world":"世界","hello":"你好"}
b = a["world"]                                    #取字典中词条"world"对应的 value 值
print("%\n",b)


#例程 15：bool 量

# 在 fine 中，True 和 False 是关键字，分别代表 bool 量真、假
# 再 bool 运算中，数字 0、空字符串""、空列表[]、空字典{}被编译器翻译为 False，否则为 True

a = True
print("%\n",a)
```

```
a = False
print("%\n",a)
a = 3 > 2                    #遇到比较符号，先比较，把比较结果（真、假）赋值给 a
print("%\n",a)
print("%\n",1 == 2)
```

第二节 fine 运算及优先级

1、 全局变量定义符：global x

使用 global 定义了一个全局变量，在程序的任何地方，都可以对全局变量 x 赋值（不论是在主模块、或是在子模块），并立即更新该全局变量。多用于函数之间、模块之间传递数据。

2、 算术运算符

操作符	描述	示例
+	运算符两侧的数相加	a+b
-	左侧减去右侧	a-b
*	两数相乘	a*b
/	左侧除以右侧	a/b
%	左侧除以右侧求余数	a%b
**	幂：返回 a 的 b 次幂	a**b

3、比较运算符

操作符	描述	示例
<	小于	a < b
<=	小于等于	a <= b
==	相等	a == b
!=	不等	a != b
>=	大于等于	a >= b
>	大于	a > b
in	包含于	a in b
notin	不包含	a notin b

4、赋值符

fine 只提供一个赋值符，即，“=”。

fine 不提供任何“简记”赋值符，便于初学者学习。初学者不需要把精力浪费在记忆“简记”符上。

5、位运算符（位运算的数据必须是整数，不能是浮点数）

操作符	描述	示例
&	按位与	a & b (a,b 均为整数)
|	按位或	a | b (a,b 均为整数)
^	按位异或	a ^ b (a,b 均为整数)
~	按位非	~a (a 为整数)
<<	左移	a << b (a,b 均为整数)
>>	右移	a >> b (a,b 均为整数)

6、逻辑运算符

操作符	描述	示例
and	逻辑与	a and b
or	逻辑或	a or b
not	取反	not a

7、优先级（括号优先）

操作符	描述	优先级
~	单目运算符，按位取反	最高优先级 0
**	指数	1
&|^<< >>	按位运算符	2
*/%	乘法	3
+-	加法	4
< <= == != >= >	比较运算符	5
in notin	包含于、不包含	6
=	赋值运算符	7
not	逻辑取反（单目运算符）	8
and or	逻辑与、逻辑或	最低优先级 9

第三节 fine 注释符

1、 行注释符：# ——表明本行在#之后的部分为注释，编译时剔除。
2、 段落注释符：/* */——成对出现，它们之间的所有内容，都会被编译器忽略。
3、 续行符：\——如果一条语句太长，不便于阅读，可以用字符 \ ，并立即换行（之间不允许插入其它字符），表明后面一行属前一行的延续。

第四章、数据结构（字符串、列表和字典）

第一节、字符串　例程《string_method.txt》

```
#  字符串：
#  1、双引号" "之间的字符构成字符串，字符串赋值方式：a = "abcd"、或 a = input()
#  2、字符串长度不限；
#  3、字符串中如果遇到\n，会将其转义成一个换行符（占一个字节）
#  4、空字符串也是字符串，如：b = ""
#  5、字符串长度计算：x = len("abcd")，结果是 x = 4
#       x = len("ab\ncd")，结果是 x = 5，因为"\n"被翻译成了一个字节换行符（0x0a）
#  6、字符串可以相加（拼接）
#  7、字符串可以直接比较

转义符：在字符串内部遇到下列字符将被转义为功能字符
\a->0x00,BEL 响铃字符
\b->0x0c,退格字符
\f-> 0x0c,换页符
\r->0x0d ,回车键
\t->0x09，制表符
\v->0x0b，垂直制表符
\o->0x00，空字符
\n->0x0a，换行符
\'->'，还是单引号
\"->"，还是双引号
\?->?，　还是问号
\反斜杠\，转移后还是反斜杠
在排除上述转义符之后，\反斜杠之后遇到\r 转义为续行符，抛弃本行中\之后的所有字符，
    并与下一行字符拼接在一起
```

```
# 例程 1：str.append(x)在字符串的尾部追加字符或字符串

a = "hello world!"
print("%\n",a)
b = "ef"
a.append(b)             # 在字符串 a 的尾部追加字符串 b
print("%\n",a)          # 输出结果为："hello world!ef"


# 例程 2：str.index(x)返回字符或子字符串 x 在字符串 str 中的位置（起始值为 0）

a = "hello 世界你好 world! "
b = a.index("llo")                  # 返回结果 b = 2(一个英文字符占一字节)
print("%\n",b)
b = a.index("你")                   # 返回结果 b = 10(一个汉字占两个字节)
print("%\n",b)
print("%\n",a.index("or"))          # 打印结果：16


# 例程 3：str.contains(x)如果字符串中包含 x，返回 True(真)，否则返回 False(假）

a = "hello 你好世界 world! "
if "lo" in a
{
    print("%\n","True")
}
else
{
    print("%\n","False")
}

if "xyz" in a
{
    print("%\n","True")
}
else
{
    print("%\n","False")
}

if "好世" in a
{
    print("%\n","True")
```

```
}
else
{
      print("%\n","False")
}

print("%\n",a.contains("lo"))
print("%\n",a.contains("xyz"))
print("%\n",a.contains("好世"))



#  例程 4：str.count(x)计算字符串中包含 x 的次数

a = "hello  你好世界好  world! "
count = a.count("llo")                    # "llo"只出现一次，count = 1
print("%\n",count)

count = a.count("o")                      # "o"只出现两次，count = 2
print("%\n",count)

count = a.count("好")                     # "好"只出现两次，count = 2
print("%\n",count)


#  例程 5：str.upper()将字符串 str 中的所有字符都变成大写，数字、空格、标点不变

a = "hello world!"
b = a.upper()
print("%\n",b)          #  全变大写
print("%\n",a)          # a 的内容不变


#  例程 6：str.lower()将字符串 str 中的所有字符都变成小写，数字、空格、标点不变

a = "Hello world!"
b = a.lower()
print("%\n",b)           #  全变小写
print("%\n",a)           # a 的内容没变

#  例程 7：str.capitalize()将字符串 str 的第一个字符改为大写，其它改为小写

a = "hello World!"
b = a.capitalize()
print("%\n",b)                  #  首字符大写，其它不变
print("%\n",a)                  # a 的内容没变


#  例程 8：str.title()将字符串 str 中的每个单词首字符改为大写，其它改为小写

a = "hello wORld!"
b = a.title()
print("%\n",b)          #  每个单词首字符变大写
print("%\n",a)          # a 的内容没变


#  例程 9：
# str.replace(x,y)用 y 替换掉字符串 str 中的 x，原 str 发生变化
#  输入参数：参数 1 是老的字符或子字符串，参数 2 是新的字符或子字符串
#  输出内容是将字符串 str 中 x 用 y 替换掉
a = "heLLow  输出内容是将字符串  wORld!"
b = a.replace("he","He")   #"He"替换"he"
print("%          %\n",a,b)     #b 是替换后的字符串,a 保持不变
c = a.replace("内容是","好好好好")   # "好好好好"替换"内容是"
print("%        %\n",a,c)     #c 是替换后的字符串,a 保持不变
d = a.replace("OR","NONO")    # "NONO"替换"OR"
print("%        %\n",a,d)     #d 是替换后的字符串,a 保持不变


#  例程 10：str[n]返回字符串的第 n 个字符(英文字符占一个字节，汉字占两个字节)
#    输出字符串的第 n 个字符（如果要从字符串中获取一个汉字，输入参数 n 必须是
#    该汉字对应的第一个字节的序号，否则返回 False，并报错）
#    原来的字符串 str 保持不变

a = "hello  你好世界  wORld!"
b = a[4]                            #  返回 o
print("%\n",b)

b = a[8]                            #  返回“好”（第 8 个字节正好是“好”的第一个字节）
if   b==False
{
    print("输入参数错误！可能不是一个汉字的首字节！\n")
}
else
{
    print("%\n",b)
}
```

```
b = a[9]            # 9 对应汉字“好”的第二个字节，输入参数错误，返回 False
if   b==False
{
    print("输入参数错误！可能不是一个汉字的首字节！\n")
}
else
{
    print("%\n",b)
}

b = a[17]
print("%\n",b)              # 打印第 17 个字符（R）
print("%\n",a)              # a 是源字符串，没变


# 例程 11：两个字符串可以直接相加，结果是两个字符串拼接

a = "hello wORld!"
b = "世界，你好！"
print("%\n",a+b)          # b 拼接在 a 后面，a 的内容变化，b 的内容不变
#   只支持两个字符串相加，不支持减法；
#   字符串与数字之间不能进行加、减、乘、除等数学运算。


# 例程 12：支持两个字符串比较

a = "abc"
b = "ab"
if a > b
{
    print("%\n","a > b")
}
elif a == b
{
    print("%\n","a = b")
}
else
{
    print("%\n","a < b")
}


# 例程 13：str.afterchr(var)截取字符串中指定字符之后的部分（不包括 var 本身）

# 参数（var）是被查询字符（只能是一个字符或一个汉字），返回截取的字符串

# 历史原因\有特定含义，例如\n 代表一个字符(换行符)，\\n 代表两个字符：\和 n
# 在字符串中要输入字符\，正确的方法是输入\\，系统会将其翻译为\

a = "d:\\fin\\aa.py"      # \\是\的转义符
b = a.afterchr("\\")      # 得到的是第一个 \ 之后的部分
print("%\n",b)           # 运行结果："fin\aa.py"

a = "abcd 我爱北京天安门 efg"
b = a.afterchr("c")
print("%\n",b)              # 运行结果："d 我爱北京天安门 efg"

c = a.afterchr("爱")
print("%\n",c)              # 运行结果："北京天安门 efg"

c = a.afterchr("你")
print("%\n",c)              # 找不到与 var 匹配的字符（或汉字）时，返回空字符串""

c = a.afterchr("x")
print("%\n",c)              # 运行结果：""


# 例程 14：str.beforechr(var)截取字符串中，指定字符之前的部分（不包括 var 本身）

# 参数 var（只能是一个字符或一个汉字，不能为多个字符或多个汉字）
# 返回截取的字符串

a = "d:\\fin\\aa.py"
b = a.beforechr(".")
print("%\n",b)                  # 运行结果："d:\fin\aa"

# 例程 15：str.rafterchr(var)从后向前查询，截取字符串中，指定字符之后的字符串
# 与 str.afterchr(var)的区别是，一个从前向后查询，一个从后往前查询

a = "d:\\aa\\bb\\aa.py"
b = a.rafterchr(".")
print("%\n",b)                  #   运行结果："py"
b = a.rafterchr("\\")
print("%\n",b)                  #   运行结果："aa.py"


# 例程 16：截取字符串中，指定字符之后的前半部分，但是查找顺序是从后向前查找
# 字符串方法：a.rbeforechr(var)
```

```
# a 是字符串对象
# 参数（var）是被查询字符
# 返回截取的字符串，找不到与 var 匹配的字符时，返回空字符串

a = "d:\\aa\\bb\\aa.py"
b = a.beforechr("\\")
print("%\n",b)                    #   运行结果："d:"

b = a.rbeforechr("\\")
print("%\n",b)                    #   运行结果："d:\\aa\\bb"


# 例程 17   str = str.set(n,sub)将字符串 sub 复制 n 次赋值给 str

str = ""                          # 调用该方法之前，需要对 str 赋初值""(空字符串)
str = str.set(5,"abc")
print("%\n",str)                  # 输出“abcabcabcabcabc”


# 例程 18   str1 = str.firstpart(n)将字符串 str 的前 n 个字符赋值给字符串 str1。如果第 n 个字符是汉字的第一个字节，将取(n-1)个字符

str = "abcd 中国人民 efg"        # 注意一个汉字占用两个字节

str1 = str.firstpart(7)        # 第 7 个字符是“国”的第一个字节，输出不包含“国”
print("%\n",str1)                 # 输出“abcd 中”

str1 = str.firstpart(8)        # 第 8 个字符是“国”的第二个字节，输出包含“国”
print("%\n",str1)                 # 输出“abcd 中国”

str2 = str.firstpart(14)       # 第 14 个字符是“f”，输出“g”之前的字符串
print("%\n",str2)                 # 输出“abcd 中国人民 ef”


# 例程 19   str1 = str.secondpart(n) 返回从 str 字符串的第 n+1 个字符开始的后半部分，
# 如果第 n+1 个字符是汉字的第二个字节，将从第 n 个

字符开始返回

str = "abcd 中国人民 efg"         # 注意一个汉字占用两个字节

str1 = str.secondpart(7)       # 第 7 个字符是“国”的前一个字节，输出包含“国”
print("%\n",str1)                 # 输出“国人民 efg”

str1 = str.secondpart(8)       # 第 8 个字符是“国”的后一个字节，输出不包含“国”
print("%\n",str1)                 # 输出“人民 efg”

str2 = str.secondpart(14)      # 第 14 个字符是“f”，输出“g”后面的字符
print("%\n",str2)                 # 输出“g”


# 例程 20     str.pop()
# 无参数、无返回，将字符串 str 的最后一个字符或汉字抛弃

a = "abcdegf"
print("%\n",a)
a.pop()
print("%\n",a)

b = "中国杭州"
print("%\n",b)
b.pop()
print("%\n",b)


# 例程 21：二进制字符串输入和追加

a = Binary("x32x01x02x03x00")      # 输入二进制字符串（32 01 02 03 00）
a.appendBinary("x04x05x06")        # 在尾部追加二进制字符串(04 05 06)

print(a)              # a 的值是：32 01 02 03 00 04 05 06
```

第二节、列表　　例程《list_method.txt》

```
#   列表：
#   1、列表赋值方式一：a = [1,2,3,4,5]
#   2、列表赋值方式二：先将一个空列表赋值给 b，即：  b = []
#       再使用列表方法  b.append(x)、或  b.extend(list)添加列表项或子列表
#   3、列表项数计算：x = len(a)
```

```
# 例程 1： list.append(x)将 x 追加到原列表的尾部

a = [1,2,3,"ab","abc"]
a.append("abcd")
print("%\n",a)              # a 的尾部添加了一项字符串"abcd"
a.append(12.5)
print("%\n",a)              # a 的尾部添加了一项浮点数，显示最大精度 15 位小数
print("%.1f\n",a)           # a 的尾部添加了一项浮点数，保留 1 位小数


# 例程 2： list.extend(x)将列表 x 追加到原列表的尾部

a = [1,2,3,"ab","abc"]
b = [4,5,6,7]
a.extend(b)                      # 将列表 b 内的所有项
print("%\n",a)                   # a 包含了 b 的所有项
print("%\n",b)                   # b 不变

a = [1,2,3,"ab","abc"]
b = [4,5,6,7]
c = a + b
print("%\n",a)                   # a 不变
print("%\n",b)                   # b 不变
print("%\n",c)                   # c 包含了 a 和 b 的所有项


# 例程 3：list.insert(n,x)在第 n 项位置插入 x

a = [1,2,3,"ab","abc"]
a.insert(2,"XYZ")
print("%\n",a)                # 运行结果：a = [1,2, "XYZ",3,"ab","abc"]


# 例程 4：list.count(x)计算列表中包含 x 的次数

a = [1,2,3,"ab","abc",3,3,3,4]
b = a.count(3)                    # 计算列表 a 中有几个数值等于 3 的项
print("%\n",b)                    # 打印结果为 4


# 例程 5：list.index(x)查找内容等于 x 的项所在的位置，并返回位置（从 0 计数）
# 如果找不到与 x 匹配列表项，则返回 -1

a = [1,2,3,"ab","abc",3,3,3,4]
b = a.index(3)        # 查找内容为 3 的项，返回其位置(有多个时，只返回第一个)
print("%\n",b)
b = a.index("abc")
print("%\n",b)
b = a.index("xxxx")
print("%\n",b)              # 列表中没有"xxxx"项，返回-1


# 例程 6：list.pop(n)弹出第 n 项内容，并将删除列表项中对应的数据
# 输入参数 n 是列表中的第 n 项，也可以为空
# 输入参数为空时弹出并删除最后一项
# 输出第 n 项的内容，随即删除第 n 项

a = [1,2,3,"ab","abc",3,3,3,4]
b = a.pop(3)                     # 弹出第 3 项（"ab"），并将列表中的该项删除
print("%\n",b)                   # 显示第三项对应的内容
print("%\n",a)                   # 显示列表 a，此时少了一项"ab"
b = a.pop()                      # 无参数
print("%\n",b)                   # 显示末项的内容
print("%\n",a)                   # 显示列表 a，此时又少了末项 4


# 例程 7：list.remove(x)直接删除内容为 x 的项，无返回
# 如果有多项相同，只删除最前面的项

a = [1,2,3,"ab","abc",3,3,3,4]
a.remove(3)                      # 删除内容为 3 的项（多项相同时，只删除最前面的项）
print("%\n",a)                   # 显示列表 a，此时少了内容为 3 的第二项，与 b 相同


# 例程 8：list.reverse() 将整个列表顺序颠倒，改变顺序，不改变内容

a = [1,2,3,"ab","abc",3,3,3,4]
a.reverse()                   # 列表反序（非倒序）
print("%\n",a)                   # 显示列表 a，顺序颠倒


# 例程 9：list.sort() 将列表内容排序（升序），改变顺序，不改变内容

a = [1,5,2.5,8,3.3,4,9,2]
a = a.sort()                     # 列表升序（先 sort()，再 reverse()可以实现降序）
print("%.1f\n",a)             # 格式控制%.1f 表示列表中所有浮点数均保留 1 位小数
```

```
# 例程 10：list.replace(n,x)用内容 x 替换列表的第 n 项

a = [1,5,2,8,3,4,9,2]
a.replace(3,"hello")
print("%\n",a)                    # 用"hello"替换第三项（8）
```

第三节、字典　　例程《dict_method》

```
#  字典：
#  1、字典赋值方式一：a = {"a":1,"b":2,"world":"世界","hello":"你好"}
#  2、字典赋值方式二：a = {} ，首先得到一个空字典 a，
#       再用字典方法添加词条，如：a.setitem(k,y)

# 例程 1：dict.setitem(x,y)在字典 dict 中添加一个词条 x:y

a = {"a":1,"b":2,"world":"世界","hello":"你好"}
a.setitem("beijing","北京")                              # 字典 a 增加了一词条
print("%\n",a)                                           # 在%位置显示字典 a

a.setitem("shanghai","上海")
print("%\n",a)                                           # 继续增加词条


# 例程 2：dict.get_key(n)返回序号为 n 的词条的 KEY 值，n 大于词条数时，返回空""

a = {"a":1,"b":2,"world":"世界","hellow":"你好"}          # 录入字典
b = a.get_key(2)                 # 取出编号(从 0 计数)为 2 的词条的 KEY 值"world"
print("%\n",b)                   # "world"
x = a.get(b)                     # 取出 b 对应的值"世界"
print("%\n",x)

# 为避免输入参数大于字典词条数，可以先比较，再操作。例如参数为 10 时：
if   10 < len(a)
{
     c = a.get_key(10)       # 只有字典的总词条数大于 10，该语句才会被执行
}


# 例程 3：dict.get(k)在字典 dict 中,查找键值(key)为 k 的词条的值（value)

a = {"a":1,"b":2,"world":"世界","hello":"你好"}
b = a.get("world")     # 取出键值为"world"的词条对应的值，并赋值给变量 b
print("%\n",b)         # 在%位置显示"world"对应的值（"世界"）

c = a.get("b")
print("%\n",c)         # 打印"b"对应的值（2）

c = a.get("好")         # 字典中没有该词条，返回空字符串""
if   c == ""
{
     print("%\n","字典中没有该词条！")         #打印"b"对应的值（2）

}
print("%\n",a)         # 字典 a 不变

# 例程 4：dict.contains(k)在字典 dict 中,查找键值为 k 的词条是否存在
# 如果词条 k 存在，返回 True(真)，如果不存在，返回 False（假）

a = {"a":1,"b":2,"world":"世界","hello":"你好"}
b = a.contains("world")                                  # 存在词条"world",b = 1(True)
print("%\n",b)
x = a.contains("c")                                      # 没有词条"c"，x = 0(False)
print("%\n",x)
print("%\n",a)                                           # 不影响字典 a 内容

# 例程 5：dict.copy()复制字典 dict，原字典不变化

a = {"a":1,"b":2,"world":"世界","hello":"你好"}
b = a.copy()                                             # 复制字典 a，并存入变量 b 中
print("%\n",b)                                           # b 是一个与 a 相同的字典

c = a
print("%\n",b)            # 打印结果相同，但是，把 b 和 c 含义不同

# b 是复制了一份与 a 一样的字典，系统里面有两份字典
# c 只是保存了 a 字典的地址，系统里只有一份字典，a 和 c 都指向那个字典。


# 例程 6：dict.popitem(key)返回字典的最后一个词条内容，并删除该词条
```

```
a = {"a":1,"b":2,"world":"世界","hello":"你好"}
b = a.popitem("world")          # 将字典 a 中 KEY 值为"world"的词条删除
print("%\n",b)                  # b = ["world","世界"]
b = a.popitem("xxxxx")          # 字典 a 中不存在值为"xxxxx"的词条，返回-1
print("%\n",b)                  # b = -1 意味着词条"xxxxx"不存在
print("%\n",a)                  # a = {"a":1,"b":2, "hello":"你好"}

# 为避免试图删除一个不存在的词条，可以采用下面做法
a = {"a":1,"b":2,"world":"世界","hello":"你好"}
if   a. contains("xxxxx")
{
     b = a.popitem("xxxxx")        # 当词条"xxxxx"存在时，才执行删除操作
}
else
{
     print("%\n","该词条不存在！")
}

# 例程 7：dict.pop()无参数，返回字典末项的值，并删除字典末项

a = {"a":1,"b":2,"world":"世界","hello":"你好"}
b = a.pop()                  # b = ["hello","你好"]  是一个列表
print("%\n",b)              # 打印键值为"world"的词条内容
print("%\n",a)              # a = {"a":1,"b":2,"world":"世界"}


# 例程 8：dict.keys() 无参数，返回字典的所有 key 值（列表形式）,原字典不变
a = {"a":1,"b":2,"world":"世界","hello":"你好"}
b = a.keys()          # b = ["a","b","world","hello"]
print("%\n",b)      # 打印包含所有键值的列表
print("%\n",a)      # 原字典不变


# 例程 9：dict.values() 无参数，返回字典的所有 value 值（列表形式），原字典不变

a = {"a":1,"b":2,"world":"世界","hello":"你好"}
b = a.values()           # b = [1,2, "世界","你好"]
print("%\n",b)
print("%\n",a)         # 原字典不变


# 例程 10：dict.items() 无参数，返回字典的所有[key,value]值（列表形式）

a = {"a":1,"b":2,"world":"世界","hello":"你好"}
b = a.items()       # b = [["a",1],[ "b",2],[ "world","世界"],["hello","你好"]]双层列表
print("%\n",b)      # b 的每一项都是一个子列表
print("%\n",a)      # 原字典不变
```

第五章、流程控制　例程《if_while_for》

第一节、if 判断

```
#  if 语句的控制转移
#  在 fine 语言中，True 和 False 是关键字，分别代表真和假

#例程 1：if 语句
a = True
if  a   #意思是如果 a 为 True，执行｛｝内的函数体
{
    print("%\n","It is True")
}

a = False
if  a   #意思是如果 a 为 True，执行｛｝内的函数体
{
    print("%\n","It is True")
}
#  显然，该例 a 不为 True，所以不会执行｛｝内的函数体

#例程 2：if 语句
#  不等于 0 的数（包含小数）为 True（真）
#  等于 0 的数为 False（假）

a = 10

if  a    #意思是如果 a 为 True（真），执行下面{}内的函数体
{
    print("%\n","It is True")
}
#  凡是不等于 0 的数，都为 True（真），因此上述函数体得以执行

a = 0
if  a
{
    print("%\n","It is True")
}
#  只有等于 0 的数为 False(假)，因此上述函数体不会被执行
a = 0
if  a    #如果 a 为 True（真），执行下面{}内的函数体
{
    print("%\n","It is True")
}
else    #除了 a 为 True（真）以外的情况，执行下面{}内的函数体
{
    print("%\n","It is False")
}
#  本例中会输出："It is False"。
#  因为 a = 0(即，a 为 False），不满足第一个条件，所以不会执行第一个函数体
#  else 语句的意思：是除了 a 为 True（真）以外的情况，执行下面{}内的函数体
#  因此，会执行 print("%\n","It is False")语句，输出："It is False"

a = 0.2
if  a       #意思是如果 a 为 True（真），执行下面{}内的函数体
{
    print("%\n","It is True")
}
#  凡是不等于 0 的数(包含小数)，都为 True（真）
#  因此上述函数体得以执行


#例程 3：if 语句
#  不为空的字符串为 True（真）
#  空的字符串为 False（假）

a = "hello"
if  a       #意思是如果 a 为 True（真），执行下面{}内的函数体
{
    print("%\n",a)
}
#  凡是不为空的字符串，都为 True（真）
#  因此上述函数体得以执行

a = ""
if  a
{
    print("%\n","It is True")
}
#  空字符串为 False(假)，因此上述函数体不会被执行

#例程 4：if 语句
#  凡是不为空的列表、字典都为 True（真）
#  凡是为空的列表、字典都为 False（假）
```

```
a = [1,2,3,4]
if  a
{
    print("%\n",a)
}
#   不为空的列表为 True（真），执行上述函数体

b = {"a":1,"b":2,"c":3}
if  b
{
    print("%\n",b)
}
#   不为空的字典为 True（真），执行上述函数体

b = {}
if  b
{
    print("%\n",b)
}
#   b 为空字典，为 False（假），因此{}内的函数体不被执行

#   总结：凡是不为 0 的数，不为空的字符串、列表、字典都为 True（真）
#   凡是为 0 的数，为空的字符串、列表、字典都为 False（假）

#例程 5：if 语句
if 3>2    # 3>2 是成立的，3>2 这个表达式会为 True，函数体将被执行
{
    print("%\n","It is True!")
}

if 3>5    # 3>5 是不成立的，3>5 表达式会为 False，函数体将不被执行
{
    print("%\n","It is True!")
}

a = "abc"
b = "ab"
c = a>b    #字符串比较，如果 a>b 成立，c 被赋值 True，否则被赋值 False
if c    # a>b 成立，函数体将被执行
{
    print("%\n","It is True!")
}

c = "abc" > "ab"    # "abc" > "ab"成立，c 被赋值 True
if c
{
    print("%\n","It is True!")   #   打印结果："It is True!"
}

if "abc" > "ab"    # "abc" > "ab"成立，函数体将被执行
{
    print("%\n","It is True!")
}
#   上述三个例子只是写法不同，效果是相同的


#例程 6：if   elif    else
a = 5   #改变 a 的值，看看程序运行的不同结果
if a>5
{
    print("%\n","a>5")
}
elif a == 5
{
    print("%\n","a = 5")
}
else
{
    print("%\n","a < 5")
}

a = 2      #改变 a 的值，看看程序运行的不同结果
if a>5
{
    print("%\n","a > 5")
}
elif a == 5
{
    print("%\n","a = 5")
}
elif a < 5        #这里没有用到 else，但是运行结果相同
{
    print("%\n","a < 5")
}
#   在一个 if-elif-else 语句里面，必须以 if 开头，不能以 elif 开头
#   中间可以出现多个 elif 语句，处理不同情况
#   最后可以用 else，执行不满足上述所有条件的情况
#   也可以没有 else，意味着不满足上述条件的情况将被忽略
```

第二节 while 循环

```
#例程 1：while 循环
a = 1
while a <= 10
{
    print("a = %\n",a)
    a = a + 1
}
#   当 while 后面的表达式 a <= 10 成立时，该比较表达式为 True，
#   函数体得以执行；当表达式不成立时，自动跳出循环体。

#例程 2：while 循环
a = 10
while a   # 当 a 不等于 0 时执行循环体，当 a 等于 0 时，退出循环体
{
    print("a = %\n",a)
    a = a - 1
}
#   当 while 后面的表达式 a 不为 0 时，该比较表达式为 True，
#   函数体得以执行；当 a = 0 时，自动跳出循环体。

#例程 3：while 循环
a = 10
while a   #当 a 不等于 0 时执行循环体，当 a 等于 0 时，退出循环体
{
    if a>5
    {
        a = a - 1   # 这里直接返回到循环起点，后面的同样语句得不到执行
        continue    # a>5 时，continue 指令强迫程序返回循环起点，忽略后面的程序
    }
     elif a == 2
    {
         break      # 当 a=2 时，break 指令强迫程序中途退出循环
     }
     else   # 既不满足 a>5 又不满足 a=2 时，打印 a 值
     {
         print("a = %\n",a)
     }
     a = a - 1   #控制循环的变量
}
#   可以使用多个 continue、break 来控制循环走向
```

第三节、for 循环

```
#例程 1：for 循环（遍历列表）
list = [1,2,3,4,5]
for i in list
{
    print("%\n",i)   #i 取值分别为列表的每一项，遍历列表
}

for i in [1,2,3,4,5]    #也可以直接使用列表，效果是一样的
{
    print("%\n",i)   #i 取值分别为列表的每一项，遍历列表
}

#例程 2：for 循环（遍历列表）
list = [1,2,3,4,5]
for i in list
{
    print("%\n",i)
    if i==3
    {
        break   #在 for 循环中，遇到 i=3 时，将退出循环
    }
}

#例程 3：for 循环（遍历列表）
list = [8,2,3,1,5]
for i in list
{
    if i<3
    {
       continue   # 满足条件 i<3 时，for 循环跳转到循环起点，后面的程序不被执行
    }
    print("%\n",i)   # 不满足 i<3 的条件时，打印 i 值
}
#   本例中不小于 3 的项：8、3、5 将被打印，小于 3 的项：1、2 将被跳过

#例程 4：for 循环（遍历列表）
for i in range(0,10,1)   # range(0,10,1)将返回一个列表[0,1,…9]
{
    print("%\n",i)
}
#   range(start,end,step)是一个内置函数，产生一个
```

```
#   [start,start+step,start+2*step,…,n*step]列表，step 是成员间隔
#   参数 start,end,step 均为整数（包含负整数）

for i in range(5,-5,-2)   #range()将返回一个列表[5,3,1,-1,-3],不含-5
{
    print("%\n",i)
}


#例程 5：for 循环（遍历字典）
d = {"a":1,"world":"世界","c":3,"hello":"你好"}
for k in d.keys()   #遍历字典的 key 值
{
    print("%\n",k)
}
#   d.keys()将返回一个由字典 key 值组成的列表，然后 for 循环遍历整个列表

#例程 6：for 循环（遍历字典）
d = {"a":1,"world":"世界","c":3,"hello":"你好"}
for v in d.values()   #遍历字典的 value 值
{
    print("%\n",v)
}
#   d.values()将返回一个由字典的 value 组成的列表，for 循环遍历整个列表

#例程 7：for 循环（遍历字典）
d = {"a":1,"world":"世界","c":3,"hello":"你好"}
for k,v in d.items()   #遍历字典的 key,value 值
{
    print("%          %\n",k,v)
}
#   d.items()将返回一个由字典的[[key1,value1], [key2,value2],…]组成的列表，
#   for 循环遍历整个列表


#例程 8：while 和 for 循环选择
#   while 和 for 都可以执行循环操作，那么，两者之间又有什么区别呢？
#   for 循环需要先建立一个列表，然后，遍历列表实现循环，看起来比较直观，不
#   需要在循环体内设置循环控制变量，因此，在循环次数不是很大时，常常使用；
#   但是，当循环次数很大时，必然需要创建一个很大的列表，这样不仅占用大量
#   的内存，还影响执行效率，这种情况下，建议使用 while 循环。


for i in range(0,200,1)
{
    print("%n",i)
}
#   当循环次数不是太大时，使用 for 或者 while 区别不大，看你喜欢。

i = 20000
while   i
{
    print("%n",i)
    i = i - 1
}
#   当循环次数巨大时，显然，使用 while 更合适，占用内存少，效率高。
```

第六章、自定义函数　例程《function.txt》

```
#   函数定义：
def func(var1,var2,…,varn)
{
    #函数体
}

#   1、关键字:def
#   2、函数名 func
#   3、参数：数字、字符串、列表、字典等，都可以作为函数参数
#   4、数字、字符串作为变量，即可立即数，也可以是其变量
#   5、形参和实参必须严格对应，顺序不能变

#例程 1：有返回函数
#   函数定义的关键字是 def，后面 add 是函数名，括号内是参数
def   add(x,y)                    # 定义了一个名为 add 的函数，有两个形参 x,y
{
    c = x+y                       # 函数功能（输入参数相加）
    return c                      # 返回结果
}
a = add(4,6)                      # 形参 x,y 被实参 4,6 替换，调用函数 add，把结果赋值给 a
print("%\n",a)                    # 打印 a


#例程 2：无返回数据的函数
#   函数定义的关键字是 def，后面是函数名，括号内是参数
def   add1(x,y)                   #定义了一个名为 add1 的函数，有两个形参 x,y，没有 return
{
    c = x+y                       #函数功能（输入参数相加）
    print("%\n",c)                #打印 c，函数不返回数值
}
add1(4,6)                         #调用函数 add，在函数内打印出结果，不返回值
#   函数定义时的形参，必须与函数调用时的实参一一对应，顺序不能错

def   add2(x,y)                   #定义了一个名为 add1 的函数，有两个形参 x,y，return 返回
{
    c = x+y
    if   c > 10
    {
        print("%\n",c)            # 两个数之和大于 10，打印结果
        return                    # 退出函数
    }
    print("%\n","两个数之和未超过 10！")    # 两个数之和未超过 10 执行该语句
}
add2(5,10)


#例程 3：无输入参数函数，无返回数据，只是执行事务
def   my_print()                  #定义了一个名为 my_print 的函数，没有参数
{
    print("%\n","My print")       #函数功能
}
my_print()                        #调用函数 my_print


#例程 4：列表作为参数使用
#   功能：将输入参数 1（列表）的每一项，加上参数 2 的值（数字），
#   成生一个新的列表，并输出该列表

def   add(x,y)                    # 定义了一个名为 add 的函数，有两个形参 x,y，其中 x 是一个列表
{
    list_out = []                 # 定义一个空列表，
    a = len(x)                    # 计算输入列表的项数
    for i in range(0,a,1)
    {
        list_out.append(x[i]+y)   # list 列表的每一项都+y，生产新列表
    }
    return list_out               # 输出返回一个新列表
}

list = [1,2,3,4,5]                # 列表必须事先赋值给变量
a = add(list,4)                   # 参数 1 是个列表
print("%\n",a)                    # 调用 add 返回一个新的列表，打印新的列表
#   def 定义的函数形参可以是数字、字符串，也可以是列表、字典；
#   函数的定义者须准确了解参数的类型，并在函数中正确使用。


#例程 5：字典作为参数使用
#   函数功能：在函数内部打印字典的每一项
def   print_dict(dict)            #函数只有一个参数（字典）
```

```
{
    a = len(dict)                       #计算字典的词条数目
    for i in range(0,a,1)               #按顺序循环取出每一个词条的“Key”和“Value”
    {
        b = dict.get_key(i)             #由词条的序号，按顺序取出词条的“键值-Key”
        c = dict.get(b)                  #由词条的“键值”，取出对应的“值-Value”
        print("%     %\n",b,c)
    }
}
d = {"a":1,"world":"世界","c":3,"hello":"你好"}
print_dict(d)
#   形成习惯：列表参数的形参用 list，字典参数的形参用 dict；
#   列表、或字典作为函数参数并无特殊要求；
#   函数调用者需要准确知道参数类型


#例程 6：变量作用域（局部变量只在函数内有效）
def   func()                            # 函数定义
{
    x = "hello!"                        # 函数内使用了一个局部变量 x
    print("%\n",x)                      # 打印局部变量 x
}
func()                                  # 执行 func()函数，会打印出："hello!"

print("%\n",x)                          # 在编译过程中会报错：“变量 x 没有定义！”
#   局部变量 x 在 func()函数外面无效。


#例程 7：变量作用域（内、外都定义了时，内部用局部变量，外部用全局变量）
x = "你好！"                            # 函数外部定义的变量（全局变量）
def   func()
{
    x = "hello!"                        # 函数内使用了一个局部变量 x
    print("%\n",x)                      # 打印局部变量 x
}
func()                                  # 执行 func()函数时，内部 x 有定义，因此打印出："hello!"
print("%\n",x)                          # 函数内部定义的变量，在外部无效，因此打印出："你好！"
#    函数内部定义的变量，在外部无效，因此，在外部只能使用外部定义的值，
#    即："你好！"

#例程 8：变量作用域(内部没定义时，使用外部的值）
x = "你好！"                            # 函数外部定义的变量（全局变量）
def   func()
{
    print("%\n",x)                      # 函数内部没有定义 x 值，会自动使用外部定义的值
}
func()                                  # 执行 func()函数时，因内部没有定义 x，因此打印出："你好！"
print("%\n",x)   #同样打印出："你好！"
#   如果函数外部定义了某个变量，而内部没有定义，函数会自动使用
#   外部的定义的全局变量。


#例程 9：函数返回关键字：return
#   函数可以返回一个数字、字符串、列表、字典
#   当希望函数返回多个数据时，可以将多个数据构造成一个列表，将列表返回

def   multi_value()
{
     r1 = "第一个返回值"
     r2 = "第二个返回值"
     r3 = "第三个返回值"
     list = []                          # 将空列表赋值给 list
     list.append(r1)                    # 添加列表项
     list.append(r2)                    # 添加列表项
     list.append(r3)                    # 添加列表项
     return list                        # 返回包含多个值的列表
}
a = multi_value()
print("%\n",a)
var1 = a[0]                             # 把返回值从列表中提取出来
var2 = a[1]                             # 把返回值从列表中提取出来
var3 = a[2]                             # 把返回值从列表中提取出来
print("%    %    %\n",var1,var2,var3)     # 打印多项返回值
```

第七章、内置函数　例程《buildin.txt》

内置函数：是一组由 fine 提供的常用的系统函数，便于程序编写。

```
#例程 1：input()输入函数
# 无输入参数
# 键盘输入函数，每调用一次，只能输入一个数据，不支持多个数据输入；
# 如果要输入的是字符串，必须在前、后输入双引号（"）,长度限制在 254 个字节
# 以内（不含引号），如果要输入的是数字（包括浮点数）时，不能加双引号。
# 如果输入格式错误（例如少了右边的引号、或数字中出现非法字符等）,会发出
# 嘟嘟声响，提醒重新输入。

t = 0                                   # 数字输入的例子
for i in range(0,10,1)
{
    print("a = ")
    a = input()
    print("显示输入  a = %\n",a)
    t = t + a
    print("累加和  t = %.10f\n",t)
}
b = input()                             # 输入字符串"abcde"时，前后带双引号
print("%\n",b)                          # 打印结果是："abcde"
#   在使用 input 输入数据时，如果输入的时数字，不加引号，直接输入，
#   如果输入的是字符串，必须在字符串前后添加双引号


#例程 2：无格式打印（显示）
#   print()显示（打印）函数
#   输入参数为需要打印的数据
#   没有返回值
a = 1                                   #a 是变量，1 是数字常量
print(a)                                #在当前光标位置打印变量内容
print(1)                                #直接打印数字常量 1
print("abc")                            #直接打印字符串常量"abc"
b = "I love you."
print(b)                                #先给变量 b 赋值，再打印变量 b
# 无格式控制打印，输出字符将显示在当前光标位置，如果是多个数据输出，它们之间没有间隔符。


#例程 3：格式打印（显示）
#   print()显示（打印）函数
#   第一个输入参数作为打印控制格式，控制变量、常量打印
#   没有返回值
a = 1                                   #数字常量赋值给变量 a
b = "I love you."                       #字符串常量赋值给变量 b
c = [1,2,3,4]                           #列表赋值给变量 c
d = {"a":1,"b":2,"c":3}                 #字典赋值给变量 d
print("%    %    %    %\n",a,b,c,d)
#   字符串"%    %    %    %\n"是打印控制格式
# 四个变量打印在同一行，四个替换符 %    %    %    % 分别对应变量 a,b,c,d
#   \n 控制打印换行


#例程 4：格式打印（显示）
#   print()显示（打印）函数
#   第一个输入参数作为打印控制格式，控制变量、常量打印
#   没有返回值
a = 1                                   #数字常量赋值给变量 a
b = "I love you."                       #字符串常量赋值给变量 b
c = [1,2,3,4]                           #列表赋值给变量 c
d = {"a":1,"b":2,"c":3}                 #字典赋值给变量 d
print("%\n    %\n    %\n    %\n",a,b,c,d)    #每个变量打印在不同的行
#   与例程 3 不同的是：每个变量打印在不同的行


#例程 5：格式打印（显示）
#   print()显示（打印）函数
#   第一个输入参数作为打印控制格式，控制变量、常量打印
#   没有返回值

a = 1                                   #数字常量赋值给变量 a
b = "I love you."                       #字符串常量赋值给变量 b
c = [1,2,3,4]                           #列表赋值给变量 c
d = {"a":1,"b":2,"c":3}                 #字典赋值给变量 d
format = "%    %    %    %\n"
print(format,a,b,c,d)                   #四个变量打印在同一行
# 与例程 3 的不同是：打印控制字符串是一个变量


#例程 6：浮点数打印（浮点数指小数或科学记数法）
math = MATH()
a = math.sqrt(3)                        # 3 的开平方赋值给变量 a
```

```
b = "I love you."                       # 字符串常量赋值给变量 b
c = 10                                  # 数字赋值给变量 c
print("%    %    %\n",a,b,c)           # 三个变量打印在同一行
#   在第一个%位置打印变量 a 的内容，是个浮点数，默认用最大精度表示（15 位小数），
#   在第二个%位置打印字符串变量 b 的内容，
#   在第三个%位置打印整数变量 c 的内容，
#   \n 控制打印换行。

#例程 7：浮点数打印（浮点数指小数或科学记数法）
math = MATH()
a = math.sqrt(3)                        # 3 的开平方赋值给变量 a
b = "I love you."                       # 字符串常量赋值给变量 b
c = 10                                  # 10 赋值给变量 c
print("%.3f    %    %\n",a,b,c)         # 三个变量打印在同一行
# 与例程 6 不同的是：打印第一个变量 a 时，进行了保留小数位限制，
# 本例只保留 3 位小数，其它非浮点数不需要改变。


#例程 8：浮点数打印（浮点数指小数或科学记数法）
math = MATH()
a = math.sqrt(3)
b = "I love you."
c = 10
print("%.5f    %.5f    %.5f\n",a,b,c)
#   本例对每个变量都进行了小数位数限制，但是，它只对浮点数有效，并不会
#   影响其它类型数据的输出结果。


#例程 9：int(x)，x 是小数，int(x)的功能是取 x 得整数部分
#   int(x)取整函数
#   输入参数是小数（也可以是整数）
#   输出是 x 的整数部分
math = MATH()
a = math.sqrt(3)
b = int(a)                              #对 a 取整
c = int(a+0.5)                          #对 a 四舍五入取整
print("%.4f    %    %\n",a,b,c)        #三个变量打印在同一行


#例程 10：len(x)，参数 x 可以是字符串、也可以是列表、也可以是字典
#   len(x)计算参数长度
#   输入参数 x 可以是字符串、也可以是列表、也可以是字典
#   如果 x 是字符串，则输出字符串中字符的个数（包含空格、标点符号一个汉字占两个字节）
#   如果 x 是列表，则输出列表中的项数
#   如果 x 是字典，则输出字典中的词条数目

a = "hello World!"                      #将一个字符串赋值给变量 a
print("字符串长度为：%\n",len(a))       #计算字符串的长度并打印
#格式打印，引号内的内容原样打印，但是在%的位置用 a 的内容替换，\n 换行

print("字符串长度为：%\n",len("hello World!"))    # 直接将字符串输入给函数 len()

b = [1,2,3,"abc","001","hello"]         #将列表赋值给变量 b
c = len(b)
print("列表项数为：%\n",c)

d = {"good":"好","world":"世界","apple":"苹果"}
print("字典的词条数目为：%\n",len(d))


#例程 11：range(start,end,step)
#   range(start,end,step)按要求返回一个列表
#   返回一个列表[start,start+step,start+2*step,+ … +n*step]
#   返回的列表中不包含 end 本身
a = range(0,10,1)
print("%\n",a)                  # [0,1,2,3,4,5,6,7,8,9]
a = range(0,10,3)
print("%\n",a)                  # [0, 3, 6, 9]
a = range(12,20,1)
print("%\n",a)                  # [12,13,14,15,16,17,18,19]
a = range(5,-5,-2)
print("%\n",a)                  # [5,3,1,-1,-3]


#例程 12：max(var1,var2,…,varn)（求最大值）
#   max(var1,var2,…,varn) 返回 var1,var2,…,varn、或列表中最大的那一个
#   输入参数组 var1,…,varn，可以是一组数(整数或浮点数)、或是一组字符串，还可以是一个列表
#   输入参数组、或列表内元素，要么全部是数，要么全部是字符串，字符串和数不能比较，
#   返回其中最大数，或最大的字符串。

a = max(1, 5, 3.5, 7.2, 3, 4, 0.2)
print("%.2f\n",a)

a = max("0","000","a","abc","ac")
print("%\n",a)

list = [1,2,3,4,5,6,7,8]
```

```
print("%\n",max(list))

#   字符串的比较规则：逐位比较，如果第一位相同，再比较第二位，直到
#   某位出现不同，此时该位字符较大的，对应的字符串为较大的字符串。
#   如："b" > "ab" > "a"
#   数字字符"0","1",…，"9"小于大写字母，大写字母小于小写字母

#例程 13：min(var1,var2,…,varn)
#   min(var1,var2,…,varn) 返回 var1,var2,…,varn、或列表中最小的那一个
#   输入参数组 var1,…,varn，可以是一组数(整数或浮点数)、或一组字符串，还可以是一个列表。
#   输入参数组，或列表元素，要么全部是数，要么全部是字符串，字符串和数不能比较，
#   返回其中最小的数，或最小的字符串

a = min(1, 5, 3.5, 7.2, 3, 4, 0.2)
print("%.2f\n",a)
a = min("0","000","a","abc","ac")
print("%\n",a)

list = [1,2,3,4,5,6,7,8]
print("%\n",min(list))


#例程 14：itoc(var)
# 输入参数 var 必须是（正、负）整数、或者（正、负）浮点数，浮点数包括小数和科学记数法。
# 返回十进制的字符串

a = itoc(505)
print("%\n",a + "abcd")          # 打印结果为："505abcd"

a = itoc(-123.456)
print("%\n",a)                    # 打印结果为："-123.456"


#例程 15：ctoi(var)
#   参数 var 可以是正整数、负整数的字符串、或含小数点的字符串、或是科学记数法的字符串。
#   返回十进制的数

a = ctoi("505")                 # 字符串
print("%\n",a)                  # 转化为整数：505

a = ctoi("-505")                # 字符串
print("%\n",a)                  # 转化为整数：-505

a = ctoi("-123.456")            # 字符串
print("%.3f\n",a)               # 转化为浮点数：-123.456

a = ctoi("-123.456e-3")         # 字符串
print("%.6f\n",a)               # 转化为浮点数：-0.123456

a = ctoi("123.456e-30")         # 字符串
print("%\n",a)                  # 转化为浮点数：1.23456000000000e-28
print("%.3e\n",a)               # 转化为浮点数：1.234e-28，并以科学计数法打印出来


#例程 16：version() 无输入参数，返回：版本号 ： 当前 fie 文件编译时间、 保留空间、文件长度
print("%\n",version())


# 例程 17：c = sprint() 功能：与 print 函数功能类似
# 区别在于，print 是将字符串打印在屏幕上，sprint()是将字符串赋值给变量，并不打印。

a = 5.6
b = "中国"
list = [1,"abcd",12.56,"张三"]
dict = {"a":1,"b":2,"c":3}
c = sprint("%.1f      %      %.2f      %\n",a,b,list,dict)      # 将变量转化为字符串，赋值给变量 c
print("%\n",c)
# 运行结果与 print("%.1f      %      %.2f      %\n",a,b,list,dict)的效果是一样的


# 例程 18：sprintTab(var,width,decim,way) 将变量 var 按参数 width,decim,way 要求转化为字符串

# 参数 1：是被转化变量，可以是整数、浮点数、字符串、列表、字典
# 参数 2：经 sprintTab 转换后，参数 1 所占位置宽度，不足部分自动补空格
# 参数 3：小数位数，对于浮点数，该参数控制显示小数的位数，该参数对于整数和字符串无效
# 参数 4：补空格方式，1 左对齐，2 居中，3 右对齐，其他值被默认左对齐
# 返回：返回一个占位 width 个字节的字符串，不足部分补空格

list_class = []

list = ["姓名","性别","年龄","就读学校"]
list_class.append(list)

list = ["张三","男","25","北京大学"]
list_class.append(list)

list = ["王二麻子","男","22","北京航天航空大学"]
```

```
list_class.append(list)

list = ["贝拉克.侯赛因.奥巴马","男","78","Harvard University"]
list_class.append(list)

# 像这样一组数据，直接 print 是对不齐的，显得混乱，用 sprintTab 预处理后，再显示，就整齐了。

# 左对齐
for i in list_class
{
    member = i                              # 从班级列表里面取出一位同学的信息
    name = sprintTab(member[0],30,0,1)      # 姓名占位 30 个字节，小数位 0，左对齐
    sex    = sprintTab(member[1],6,0,1)     # 性别占位 6 个字节，小数位 0，左对齐
    age    = sprintTab(member[2],6,0,1)     # 年龄占位 6 个字节，小数位 0，左对齐
    school = sprintTab(member[3],30,0,1)    # 就读学校占位 30 个字节，小数位 0，左对齐
    print(name,sex,age,school,"\n")         # 无格式 print，最后加一个换行
}
print("\n\n\n")                             # 打印三个换行

# 居中
for i in list_class
{
    member = i                              # 从班级列表里面取出一位同学的信息
    name = sprintTab(member[0],30,0,2)      # 姓名占位 30 个字节，小数位 0，居中
    sex    = sprintTab(member[1],6,0,2)     # 性别占位 6 个字节，小数位 0，居中
    age    = sprintTab(member[2],6,0,2)     # 年龄占位 6 个字节，小数位 0，居中
    school = sprintTab(member[3],30,0,2)    # 就读学校占位 30 个字节，小数位 0，居中
    print(name,sex,age,school)              # 无格式 print
    print("\n")                             # 最后加一个换行
}
print("\n\n\n")

# 右对齐
for i in list_class
{
    member = i                              # 从班级列表里面取出一位同学的信息
    name = sprintTab(member[0],30,0,3)      # 姓名占位 30 个字节，小数位 0，右对齐
    sex    = sprintTab(member[1],6,0,3)     # 性别占位 6 个字节，小数位 0，右对齐
    age    = sprintTab(member[2],6,0,3)     # 年龄占位 6 个字节，小数位 0，右对齐
    school = sprintTab(member[3],30,0,3)    # 就读学校占位 30 个字节，小数位 0，右对齐
    print(name,sex,age,school,"\n")         # 无格式 print
    print("\n")                            # 换行
}

# 例程 19    PowerDown(x)
# 参数是 1、2、3 其中之一，无返回。
# 功能：用于空闲等待时，降低 PC 功耗，例如：用在等待接收、或 GUI 交互的过程中
gui = GUI()                                                 # 定义对象 gui
boxnum = gui.MessageBox()                                   # 申请资源
gui.MessageBox(boxnum, "低功耗设置！")                       # 弹出提示
while gui.MessageBoxClosed(boxnum) != -1 {PowerDown(3)}     # 设置为低功耗



# 例程 21： 将一个二进制数据块赋值给变量 a

# x32、x01、x02 等，代表一个 16 进制的字节，小写 x、或大写 X 均可。

a = Binary("x32x01x02x03x00x04")    # 将 16 进制数据 32 01 02 03 00 04，保存至二进制字符串变量 a 中
print("%\n",a)                      # 打印该二进制变量
b = sprint("%",a)                   # 将二进制变量 a 转化为字符串
print(b)                            # 打印字符串 b
```

第八章、类与对象　例程《class.txt》

使用关键字 class 可以自定义类，自定义类成员包括：类属性、类方法
类方法函数定义时，有一个默认参数 self(类对象本身)，必须明示，但是，在调用时却不明明示

```
#   例程 1
class STUDENT(object)   #STUDENT 类名，object（基类）父类
{
    name                # 类属性定义，不用声明类型
    sex                 # 类属性定义，不用声明类型
    age                 # 类属性定义，不用声明类型
    yuwen               # 类属性定义，不用声明类型
    shuxue              # 类属性定义，不用声明类型
    yingyu              # 类属性定义，不用声明类型

    def average(self)       # 类方法定义时，参数 self(对象本身)必须有，不能省略
    {
        return (self.yuwen+self.shuxue+self.yingyu)/3    # 计算平均分
    }
    def score(self)         # 类方法定义时，参数 self(对象本身)必须有，不能省略
    {
        return (self.yuwen+self.shuxue+self.yingyu)
    }
}

a = STUDENT("黎明","男",18,95,99,80)      # 生成一个对象 a，并对其初始化

print("%\n",a.name)                       # 显示属性
print("%\n",a.sex)
print("%\n",a.age)
print("%\n",a.yuwen)
print("%\n",a.shuxue)
print("%\n",a.yingyu)
b = a.average()                           # 调用方法，计算该学生的平均分数
print("%.2f\n",b)                         # 显示平均分数
c = a.score()
print("%.2f\n",c)                         # 显示平均分数
#   生成类对象时，初始化参数顺序必须与类定义顺序一致，不能错位


#   例程 2
class STUDENT(object)   #STUDENT 类名，object（基类）父类
{
    name                # 类属性定义，不用声明类型
    sex                 # 类属性定义，不用声明类型
    age                 # 类属性定义，不用声明类型
    yuwen               # 类属性定义，不用声明类型
    shuxue              # 类属性定义，不用声明类型
    yingyu              # 类属性定义，不用声明类型

    def average(self)    #类方法，计算平均分数，方法可以直接使用属性值
    {
        return (self.yuwen+self.shuxue+self.yingyu)/3
    }
    def score(self)
    {
        return (self.yuwen+self.shuxue+self.yingyu)
    }
}

a = STUDENT("黎明","男",18,95,99,80)   #参数顺序必须与类定义顺序相同

#   生成一个对象 a，并对所有“属性”初始化；
#   不支持对属性单独赋值，例如：a.sex = "女"（错误）
#   不支持类定义之外，扩展属性，例如增加物理科目：a.wuli = 100（错误）
#   要增加类属性，必须在类定义中添加属性。

print("%\n",a.name)          #显示属性
print("%\n",a.sex)
print("%\n",a.age)
print("%\n",a.yuwen)
print("%\n",a.shuxue)
print("%\n",a.yingyu)
a.sex = "女"                 # 错误！不支持对属性单独赋值
print("%\n",a.wuli)          # 错误！因为类定义中没有 wuli 属性


#   例程 3
class STUDENT(object)   #STUDENT 类名，object（基类）父类
{
    name                # 类属性定义，不用声明类型
    sex                 # 类属性定义，不用声明类型
    age                 # 类属性定义，不用声明类型
    yuwen               # 类属性定义，不用声明类型
```

```
    shuxue              # 类属性定义，不用声明类型
    yingyu              # 类属性定义，不用声明类型

    # 定义类方法时，必须包含 self 参数，还可以使用外部参数
    # 调用类方法时，不需要显式出现 self 参数，只需要输入外部参数
    def average(self,x,y)     #类方法，self 是对象本身，x,y 是两个外部参数
    {
        return (self.yuwen+self.shuxue+self.yingyu)*y/x
    }

    def score(self,x)          #类方法，self 是对象本身，x 是外部参数
    {
        return (self.yuwen+self.shuxue+self.yingyu)*x
    }
}

a = STUDENT("黎明","男",18,95,99,80)   #参数顺序必须与类定义顺序相同

#   定义的方法中，有外部参数的情况
#   生成一个对象 a，并对所有“属性”初始化；

print("%\n",a.name)   #显示属性
print("%\n",a.sex)
print("%\n",a.age)
print("%\n",a.yuwen)
print("%\n",a.shuxue)
print("%\n",a.yingyu)

# 在定义 average()方法时，有一个参数 self，和两个外部参数
# 在调用方法时，不需要显式输入 self 参数，只需要输入外部参数即可
b = a.average(2,6)   # 调用时 self(对象 a)是内部参数省略，外部参数必须顺序填写
print("%.2f\n",b)    # 显示平均分数

#score()方法有一个外部参数
c = a.score(1.5)      # self 是内部参数 a，调用时省略，外部参数 x 必须填写
print("%.2f\n",c)    # 显示平均分数


#   例程 4：类继承（单继承）和多态
class   A(object)   #object 表明是基类
{
    def say(self)    #类 A 有方法 say()
    {
        print("%\n","I came from class A")
    }
}

class   B(A)           #类 B 继承了类 A
{
    def say(self)   #类 B 也定义了方法 say()
    {
        print("%\n","I came from class B")
    }
}

#   类 B 中方法 say()具有“多态性”，fine 遵循如下规则：首先在类内寻找方法，
#   只有在类内找不到对应的方法，再去父类寻找。

class   C(A)           #类 C 同样继承了类 A
{
    #   空类。类 C 中没有定义方法 say()
}

b = B()   #没有属性，生成对象 b 时，没有初值
b.say()   #显示："I came from class B"   （B 类内有方法 say()）

c = C()   #没有属性，生成对象 c 时，没有初值
c.say()   #显示："I came from class A"   （类的“继承性”,继承了父类方法）

#   在调用 c.say()时，首先在类 C 中寻找方法 say()，如果找到直接调用
#   如果在类 C 中没有找到方法 say()，查看 C 是否为基类，如果 C 是基类，
#   说明找不到方法 say()。如果 C 不是基类，去 C 的父类 A 中查找，如果在
#   父类中找到方法 say()，就可以调用。


#   例程 5
#   一个班有 5 名同学，这学期开了 3 门课（语文、数学、英语）
#   写一个程序，满足以下功能：显示每个同学的所有属性和平均分数、总分数

#定义 STUDENT 类
class STUDENT(object)   #STUDENT 类名，object（基类）父类
{
    name                # 类属性定义，不用声明类型
    sex                 # 类属性定义，不用声明类型
    age                 # 类属性定义，不用声明类型
    yuwen               # 类属性定义，不用声明类型
    shuxue              # 类属性定义，不用声明类型
```

```
    yingyu              # 类属性定义，不用声明类型

    def average(self)       #求平均分数   定义方法
    {
        return (self.yuwen+self.shuxue+self.yingyu)/3
    }
    def score(self)         #求总分数      定义方法
    {
        return self.yuwen + self.shuxue + self.yingyu
    }
}

list = []

a = STUDENT("张三","男",18,90,95,100)
list.append(a)
a = STUDENT("李四","女",18,60,70,60)
list.append(a)
a = STUDENT("王五","男",18,80,85,90)
list.append(a)
a = STUDENT("孙麻子","男",18,20,20,20)
list.append(a)
a = STUDENT("孙二娘","女",38,30,50,60)
list.append(a)

for i in list    #遍历列表（全班所有同学（对象））
{
    x1 = i.name          #姓名
    x2 = i.sex           #性别
    x3 = i.age           #年龄
    x4 = i.yuwen         #语文
    x5 = i.shuxue        #数学
    x6 = i.yingyu        #英语
    x7 = i.average()   #求平均分数
    x8 = i.score()      #求总分数
    print("%    %    %    %    %    %    %.2f    %\n",x1,x2,x3,x4,x5,x6,x7,x8)
}


#   例程 6
#   一个班有 5 名同学，这学期开了 3 门课（语文、数学、英语）
#   写一个程序，满足以下功能：
#   1、显示总分最高的同学姓名和总分
#   2、显示总分最低的同学姓名和总分

class STUDENT(object)   #STUDENT 类名，object（基类）父类
{
    name                # 类属性定义，不用声明类型
    sex                 # 类属性定义，不用声明类型
    age                 # 类属性定义，不用声明类型
    yuwen               # 类属性定义，不用声明类型
    shuxue              # 类属性定义，不用声明类型
    yingyu              # 类属性定义，不用声明类型

    def average(self)       #求平均分数   方法
    {
        return (self.yuwen+self.shuxue+self.yingyu)/3
    }
    def score(self)         #求总分数      方法
    {
        return self.yuwen + self.shuxue + self.yingyu
    }
}

# 逐个输入学生信息，并存入列表 list 中，列表的每一项都是一个 STUDENT 类对象
list = []    #定义一个空列表
a = STUDENT("张三","男",18,90,95,100)   #输入同学的所有属性，不包括方法
list.append(a)
a = STUDENT("李四","女",18,60,70,60)
list.append(a)
a = STUDENT("王五","男",18,80,85,90)
list.append(a)
a = STUDENT("孙麻子","男",18,20,20,20)
list.append(a)
a = STUDENT("孙二娘","女",38,30,50,60)
list.append(a)

# 计算最高分
score_max = 0   #记录最高分的变量，初始化为 0
for i in list    #遍历列表
{
    #每循环一次，取出一位同学的类对象，并赋值给 i
    x = i.score()                   # 用 i 的 score()方法计算出的结果，赋值给 x
    if x > score_max
    {
        score_max = x           #如果总分 x 大于 score_max，将 score_max 更新为 x
        stu_name = i.name       #同时，将该同学的姓名赋值给 stu_name
```

```
    }
}
print("分数最高的同学是：%      分数：%\n",stu_name,score_max)

# 计算最低分
score_min = 1000   #记录最低分的变量，初始化为 1000
for i in list     #遍历列表
{
    #每循环一次，取出一位同学的类对象，并赋值给 i
    x = i.score()                    # 用 i 的 score()方法计算出的结果，赋值给 x
    if x < score_min
    {
        score_min = x             #如果总分 x 小于 score_min，将 score_min 更新为 x
        stu_name = i.name         #同时，将该同学的姓名赋值给 stu_name
    }
}
print("分数最低的同学是：%      分数：%\n",stu_name,score_min)
```

第九章、操作系统方法　例程《os_method.txt》

“万物皆对象”——如果要使用操作系统的指令，首先要定义一个操作系统对象，然后，通过调用操作系统对象的方法，来使用操作系统指令。

os = OS()　　# 定义操作系统对象 os

```
#例程 1：os.getcwd() 获取当前执行文件的目录
os = OS()                       # 创建 os 对象（当然，你也可以使用 x = OS()）
path = os.getcwd()              # os 获取当前执行文件所在的目录的方法，无参数
print("%\n",path)               # 显示默认目录为 d:fine


#例程 2    os.chdir()切换目录
os = OS()                       # 创建 os 对象
a = os.chdir("d:\\")            # 从当前路径跳转到 d:\，成功返回 True，否则返回 False
print("%\n",a)                  # 显示执行结果正确与否
path = os.getcwd()              # 获取改变后的目录。
print("%\n",path)               # 显示改变后的默认目录为 d:


#例程 3    os.pause()程序暂停执行
os = OS()                       # 创建 os 对象
os.pause()                      # 无参数，使程序暂停，按任意键接着运行


#例程 4   os.rename(var1,var2)重命名方法
# 参数 1：旧文件名，参数 2：新文件名，文件名也可以是路径
# 如果指定路径新文件名已经存在，会覆盖新文件
# 如果前后路径不一样，相当于移动文件（旧文件被删除，在新的路径上产生新文件）

os = OS()                       # 创建 os 对象
a = os.rename("copy00.txt","b.test")    # 重命名，成功返回 True，否则返回 False.
print("%\n",a)                  # 重命名成功显示 True,否则显示 False

#例程 5    os.pathexists()判断文件夹或路径是否存在
#参数是字符串（文件夹或路径）
#如果路径存在返回 True,否则返回 False
os = OS()                       #创建 os 对象
print("%\n",os.pathexists("c:\\windows"))   #判断路径"c:\\windows"是否存在
print("%\n",os.pathexists("copy00.txt"))    #判断文件"copy00.txt"是否存在


#例程 6    os.pathisfile()判断参数（字符串）是否是个文件
#参数是字符串（文件名，可以带路径）
#如果是文件返回 True,否则返回 False
os = OS()                       #创建 os 对象
a = os.pathisfile("c:\\copy00.txt")     #参数是字符串（文件名）
print("%\n",a)

#例程 7    os.pathisdir()判断路径是否存在
# 输入参数（字符串）
# 如果路径存在，返回 True，否则返回 False
os = OS()                       #创建 os 对象
a = os.pathisdir("c:\\xwindows")
print("%\n",a)

#例程 8    os.mkdir() 创建单层目录
# 输入参数（字符串），是文件目录。
# 如果创建成功，返回 True，否则返回 False，
# 如果目录已经存在，放弃创建，返回 False。
os = OS()                       #创建 os 对象
a = os.mkdir("d:\\test")
print("%\n",a)
a = os.mkdir(".\\test")         #在当前目录下，创建一个子目录 test
print("%\n",a)

#例程 9    os.makedirs() 创建多层目录
# 输入参数（字符串），是多层目录
# 如果创建成功，返回 True，否则返回 False
# 如果目录已经存在，放弃创建，返回 False
os = OS()                       #创建 os 对象
a = os.makedirs("d:\\test\\test1\\test2")
print("%\n",a)
```

第十章、计算类方法　　例程《**math_method.txt**》

```
# 使用数学计算函数时：
# 第一步是调用 math = MATH()，定义一个 math 对象，
# math 对象包含两个属性：圆周率 pi 和自然数常数 e
# 第二步使用 math 的函数方法


#例程 1：打印 math 对象的两个属性：圆周率和自然数
math = MATH()
print("%.5f       %.5f\n",math.pi,math.e)      # 打印 math 对象的两个属性：圆周率和自然数


#例程 2：math.sqrt(x) 输入参数不能小于 0，输出为 x 的开平方值，是浮点数

math = MATH()                                  # 定义一个 MATH 对象 math
a = math.sqrt(9)                               # 计算 3 的开平方
print("%.0f\n",a)                               # 只显示整数部分


#例程 3：math.pow(var1,var2) 输入参数 var1、var2 可以是整数、浮点数，输出 var1 的 var2 次方

math = MATH()
a = math.pow(2.5,3.3)                          # 计算 2.5 的 3.3 次方
print("%.5f\n",a)


#例程 4：math.exp(x)，计算 e（自然数）的 x 次方，x 可以是整数或浮点数

math = MATH()
a = math.exp(5)
print("%.3f\n",a)


#例程 5：math.abs(x)，计算 x 的绝对值
# 输入参数可以为整数也可以是浮点数
# 输出 x 的绝对值

math = MATH()
a = math.abs(-10)
print("%\n",a)              # 在%位置显示 a 的值
a = math.abs(-2.5)
print("%\n",a)              # 在%位置显示 a 的值


# 例程 6：math.fabs(x)，计算浮点数的绝对值，输入参数为正负浮点数、或 0，输出 x 的绝对值
# 该方法完全可以被 abs()方法替代，是个冗余的方法，与 abs()的结果是相同的，区别在于 fabs()只适用于浮点数。

math = MATH()
a = math.fabs(-2.5)
print("%.1f\n",a)               #在%位置显示 a 的值，保留 1 位小数


#例程 7：math.log(x)，计算 x 以自然数 e 为底的对数，参数必须是大于 0 的实数(不能小于等于 0）

math = MATH()
a = math.log(3)
print("%.5f\n",a)             #在%位置显示 a 的值，保留 5 位小数


#例程 8：math.log10(x)，计算 x 以 10 为底的对数，参数必须是大于 0 的实数(不能小于等于 0）

math = MATH()
a = math.log10(10)
print("%.5f\n",a)             # 留 5 位小数

#例程 9：math.sin(x)，计算 x 的正弦值，参数必须是实数(弧度)，弧度 = 2*3.1415926*度数/360

math = MATH()
a = math.sin(10)
print("%.5f\n",a)          #在%位置显示 a 的值，保留 5 位小数


#例程 10：math.cos(x)，计算 x 的余弦值，参数必须是实数(弧度)，弧度 = 2*3.1415926*度数/360

math = MATH()
a = math.cos(10)
print("%.5f\n",a)


#例程 11：math.tan(x)，计算 x 的正切值，参数必须是实数(弧度)，弧度 = 2*3.1415926*度数/360

math = MATH()
a = math.tan(10)
```

```
print("%.5f\n",a)


#例程 12：math.rand(x1,x2)，获取 x1 到 x2 之间的伪随机整数
#   输入参数 x1 和 x2（0-32767）之间的正整数
#   输出 x1 到 x2 （包含 x1 和 x2）之间的伪随机正整数
math = MATH()
a = math.rand(1,45)
print("%\n",a)                  # 在%位置显示 a 的值
#   例如公司有 45 位员工，用 math.rand(1,45)产生一个 1 到 45 之间的随机数，作为中奖号码。


#例程 13：产生 10 个 0 到 100 之间的伪随机数
math = MATH()
for i in range(0,10,1)          #利用 for 循环，控制循环 10 次
{
    a = math.rand(0,100)        #每循环一次产生一个 0 到 100 之间的随机数
    print("%\n",a)              #显示随机数
}
```

第十一章、文件操作方法　例程《file_method.txt》

fp = FILEOPEN()　# 定义一个文件操作对象 fp，使用 fp 的方法操作文件。

#文件打开函数 FILEOPEN(var1,var2)是内置函数，有两个参数;
#var1 是字符串，是文件名（含路径，如果没有路径则默认为当前目录：finevirt.exe 所在目录位置）

#var2 是字符串，表明打开文件的方式（有多种方式）
#var2 = "r"，以只读方式打开文件，打开后文件指针指向文件头部
#var2 = "rb"，以二进制格式只读方式打开文件，打开后文件指针指向文件头部
#var2 = "r+"，打开文件用于读写，打开后文件指针指向文件头部
#var2 = "w"，打开文件，只用于写入。如果文件已存在，将覆盖原内容，如果不存在，则创建新文件
#var2 = "wb"，同上，区别在于以二进制格式打开文件
#var2 = "w+"，打开一个文件用于读写。如果文件已存在，将覆盖原内容，如果不存在，则创建新文件
#var2 = "wb+"，同上，区别在于以二进制格式打开文件
#var2 = "a"，打开一个文件用于追加内容，如果文件存在，指针指向尾部，如不存在，创建新文件
#var2 = "ab"，同上，区别在于以二进制格式打开文件
#var2 = "a+"，打开文件用于读写。如文件已存在，指向尾部；如不存在，创建新文件，用于读写
#var2 = "ab+"，同上，区别在于以二进制格式打开文件

```
# 例程 1 获取文件指针、设置文件指针、读取文件长度
fp = FILEOPEN("test.txt","r")
print("%\n",fp.getpointer())      # 读取设置后的文件指针，并打印它
fp.setpointer(18)                   # 设置文件指针
print("%\n",fp.getpointer())      # 读取设置后的文件指针，并打印它
length = fp.filelen()               # 读取文件长度（字节数），不影响此前的文件指针位置
print("%\n",length)


# 例程 2:  只读方式打开文件，如果文件不存在，则报错
# windows 平台上，保存文本文件时，每个换行符占两个字节\r\n，读出时会自动剔除掉\r，保留\n
# fp.read()读取全部文件
# fp.read(var)读取 var 个字节
fp = FILEOPEN("test.txt","r")           # 打开文件（文本、只读方式）
if fp == False                          # 如果打开文件出错，返回 False
{
    print("%\n","打开文件出错！")
}
else
{
    str = fp.read()          # 读全部文件
    print("%      %      %\n\n\n",str,len(str),fp.filelen())
    fp.setpointer(2)         # 设置文件指针
    str = fp.read(6)         # 返回的字符串中，换行符占一个字节\n（剔除了\r 字节）
    str1 = fp.read(8)        # 接着上次读出的位置，继续读出 8 个字节
    fp.close()               # 关闭已经打开的文件
    print("%      %\n",str,len(str))
    print("%      %\n",str1,len(str1))
}


# 例程 3:  只读方式打开二进制文件，如果文件不存在，则报错
# string = fp.read(var)参数是将要读取的字节数，返回所读取的字符串
fp = FILEOPEN("finecomp.exe","rb")       # 打开文件（文本、只读方式）
if fp == False                          # 如果打开文件出错，返回 False
{
    print("%\n","打开文件出错！")
```

```
}
else                                        #
{
    str = fp.read(200)      # 读取 200 个字节的内容
    fp.close()                  # 关闭已经打开的文件
    print("%\n",str)        # 显示读出的二进制数据块
    print("%\n")
}


# 例程 4:  从文件当前位置读取一行内容
# string = fp.readline()无参数 ，返回所读取的一行字符串
fp = FILEOPEN("test.txt","r")       # 打开文件（文本、只读方式）
if fp == False                              # 如果打开文件出错，返回 False
{
    print("%\n","打开文件出错！")
}
else
{
    str = fp.readline()                     # 读取一行内容，包含一个换行符\n
    str1 = fp.readline()
    fp.close()                              # 关闭已经打开的文件
    print("%     %\n",str,len(str))
    print("%     %\n",str1,len(str1))
}


# 例程 5:  读取全部文件，每一行作为一个列表项（保留换行符），保存在一个列表中
# list = fp.readlines()无参数，返回一个列表，包含全部文件，每一行作为一个列表项

fp = FILEOPEN("test.txt","r")       # 打开文件（文本、只读方式）
if fp == False                              # 如果打开文件出错，返回 False
{
    print("%\n","打开文件出错！")
}
else
{
    list = fp.readlines()                   # 读取全部文件，每一行作为一个列表项，返回该列表
    fp.close()                              # 关闭已经打开的文件
    print("len = %\n",len(list))
    for i in list                           # 遍历列表
    {
        print("%     %\n",i,len(i))                    # 打印出列表的每一项
    }
}


# 例程 6  写入字符串，如果有文件，覆盖原文件，如果没有，创建新文件
# 使用 fp.write(str)将字符串 str 写入文件 copy40.txt 中。返回：无返回

string = "abcdefg 中文\n 床前明月光，疑是地上霜；举头望明月，低头思故乡。"
fp = FILEOPEN("test1.txt","w+")   # 可写、可读
if fp == False
{
    print("打开文件错误！\n")
}
else
{
    fp.write(string)
```

```
    str = fp.read()
    print("%\n",str)
    fp.close()
}


# 例程 7

# fp.writelines(list)参数是列表，返回：无返回
list = ["床前明月光，\n","疑是地上霜；\n","举头望明月，\n","低头思故乡。\n"]
fp = FILEOPEN("test1.txt","w+")
if fp == False
{
    print("打开文件错误！\n")
}
else
{
    fp.writelines(list)
    str = fp.read()
    print("%\n",str)
    fp.close()
}


# 例程 8： 写入字符串，如果有文件，指向文件底部，如果没有，创建新文件，然后再读出显示
string = "床前明月光，疑是地上霜；举头望明月，低头思故乡。\n"
fp = FILEOPEN("test.txt","a+")          # 尾部添加模式
if fp == False                          # 如果文件能正确打开，说明文件已经存在
{
    print("打开文件错误！\n")
}
else                            # 如果打不开 copy00.txt 文件，说明文件不存在，下面写入文件
{
    fp.write(string)
    str = fp.read()
    fp.close()
    print("%\n",str)
}


# 例程 9：把二进制数据块写入文件
# 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 20 21      # 二进制数据块
b = Binary("x00x01x02x03x04x05x06x07x08x09x0ax0bx0cx0dx0ex0fx20x21")
fp = FILEOPEN("test2.txt","wb+")        # 打开文件（文本、读写方式）
if fp == False                          # 如果文件能正确打开，说明文件已经存在
{
    print("打开文件错误！\n")
}
else                            # 如果打不开 copy00.txt 文件，说明文件不存在，下面写入文件
{
    fp.write(b)
    str = fp.read()
    fp.close()
    print("%\n",str)
}
```

第十二章、时间和日期方法　例程《time_method.txt》

使用 time 方法之前，先要创建 time 对象：time = TIME()，使用完后系统自动回收，无需释放。

```
# 例程 1：time.sleep(t)　功能：让程序暂停运行 t 秒
# 输入参数是正整数，单位毫秒
# 无返回

time = TIME()
print("等待 3 秒，然后结束程序\n")
time.sleep(3000)　#睡眠 3 秒后继续执行
print("3 秒时间已过，程序结束\n")


# 例程 2：str = time.ZoneTime()无参数，获取本地时区
# 输入参数无，返回本地时区（字符串）

time = TIME()
zone = time.ZoneTime()
print("时区 = %\n",zone)


# 例程 3 UtcTime = time.GetUtcTime()　获取系统的 UTC 时间
# 无参数，
# 返回大整数（1601 年 1 月 1 日开始的系统时间，扩展到 15414 年），单位 100 纳秒。

time = TIME()
UtcTime = time.GetUtcTime()
print("UtcTime = %\n",UtcTime)


# 例程 4 LocalTime = time.GetLocalTime()　获取系统的本地时间
# 无参数，
# 返回本地时间，格式：nnnn-yy-rr HH:MM:SS DayOfWeek

time = TIME()
LocalTime = time.GetLocalTime()
print("LocalTime = %\n",LocalTime)


# 例程 5 LocalListTime = time.GetLocalListTime()　获取系统的本地时间（列表格式）
# 无参数，
# 返回本地时间，格式：[年,月,日,时,分,秒,星期,毫秒]
# 其中，星期 = 0——6，分别对应星期天——星期六

time = TIME()
LocalListTime = time.GetLocalListTime()
print("LocalListTime = %\n",LocalListTime)


# 例程 6 LocalListTime = time.UtcToLocalListTime(utctime)　将 UTC 时间转换为 local 时间
# 参数是 UTC 时间（大整数）
# 返回本地时间，格式：[年,月,日,时,分,秒,星期,毫秒]

time = TIME()

UtcTime = time.GetUtcTime()　　# 获取 UTC 时间

LocalListTime = time.UtcToLocalListTime(UtcTime)　　# 将 UTC 时间转化为本地时间
print("LocalListTime = %\n",LocalListTime)


# 例程 7 UtcTime = time.LocalListToUtcTime(list)　将本地 List 时间转换为 UTC 时间
# 参数是本地时间，格式：[年,月,日,时,分,秒]，不需要星期和毫秒，会产生尾差。
# 返回 UTC 时间（大整数）

time = TIME()

UtcTime = time.GetUtcTime()　　# 获取 UTC 时间
print("UtcTime = %\n",UtcTime)

LocalListTime = time.UtcToLocalListTime(UtcTime)　　# 将 UTC 时间转化为本地时间
print("LocalListTime = %\n",LocalListTime)

LocalListTime.pop()　# 抛弃毫秒项
LocalListTime.pop()　# 抛弃星期项
UtcTime1 = time.LocalListToUtcTime(LocalListTime)　　# 将本地时间转化为 UTC 时间
print("UtcTime1 = %\n",UtcTime1)
```

```
# 注意：UtcTime 和 UtcTime1 会有时间差异，差异来源于我们抛弃了 UtcTime 中的毫秒尾数。


# 例程 8    num = time.SetTimer(3000)    参数是定时时间，单位毫秒
#           b = time.TestTimer(num)      检测定时器 num 是否超时，超时返回 True,否则返回 False
# 返回申请到的定时器编号

time = TIME()

num = time.SetTimer(3000)                    # 设置定时器

print("申请到的定时器编号 = %\n",num)
print("等待定时器溢出……\n")

while True
{
    if(time.TestTimer(num))          # 检测定时器，一旦定时器溢出，将自动注销定该时器
    {
        print("定时器溢出，退出程序\n")
        break
    }
}


# 例程 9    time.KillTimer(num)   # 参数是定时器编号，注销定时器，无返回

time = TIME()

num = time.SetTimer(3000)                    # 设置定时器

print("申请到的定时器编号 = %\n",num)

time.KillTimer(num)             # 注销定时器 num
a = time.TestTimer(num)      # 检测定时器 num，因为该定时器已注销，再去检测将报错
```

第十三章、模块加载　例程《import》

在实际编程中，稍微复杂一点的程序，通常不会只写一个文件，而是按功能分成不同的模块，在主程序中，可以加载这些模块。

加载模块有两种方法：一种方法是直接加载整个模块，好处是函数名不会冲突，但是效率稍低；另一种方法是只加载模块中的部分函数或类，但是要避免函数名冲突。

```
# 首先写一个模块 bb.fin，作为后面的模块加载对象
# 有一个文件名为："\\fin\\bb.fin"的源文件，内含一个类 AA，和两个函数 iadd()、imut()

class AA(object)
{
    v1 = ""
    v2 = ""
    def add(self)
    {
        return self.v1 + self.v2
    }

    def show(self,x)
    {
        print("x = %\n",x)
    }
}

def iadd(x,y)
{
    return x+y
}

def imut(x,y)
{
    return x*y
}
```

```
# 例程 1  主程序 import1.fin 将加载模块 bb.fin

# 本例程 import 加载整个模块，该方法加载的模块内部的函数（或类），如果与主程序重名，不影响双方各自使用。

# 加载过程：首先对模块 bb 进行编译，生成 bb.fie 文件，保存在与源文件相同的目录中，然后使用 import 加载该模块

# 加载完整模块的指令如下：

import "C:\\fine\\fin\\examples\\import\\bb"          # 加载模块 bb 时，不需要写模块的扩展名

# 加载完成后，虚拟机将模块 bb 视作一个新添加的模块使用


a = bb.AA(3,4)                          # 使用模块 bb 内的类生成器 AA，生成类对象 a
print("%      %\n",a.v1,a.v2)    # 打印类 AA 的两个属性 v1 和 v2

b = a.add()
a.show(b)                               #利用类对象的方法 show()，来显示类对象的方法 add()的结果
print("\n\n")

c = bb.iadd(1,2)   #调用模块 bb 中的 iadd()函数
```

```
print("bb.iadd(1,2) = %\n\n",c)       #打印调用 iadd()的结果

d = bb.imut(3,4)
print("bb.imut(3,4) = %\n",d)   #在打印函数中，直接调用模块 bb 中的函数 imut()




#  例程 2  主程序 import2.fin 将加载模块 bb.fin

# from    mod    import func1,func2, ...   加载模块内部的函数

#  此方法加载的模块内的函数，如果与主程序定义的函数名重名，将发生冲突，后者将覆盖前者。

#  加载模块 bb 内部的类或函数的指令如下：

from "C:\\fine\\fin\\examples\\import\\bb" import AA,iadd,imut       #  从模块  bb  中，加载类 AA，和两个函数

#  由于类 AA 已经被直接加载到主程序中，因此，可以直接调用模块所定义的类、或函数

a = AA(5,6)

a.show(a.v1)

b = a.add()
a.show(b)

print("\n\n")

#  同样，可以直接调用函数 iadd()和 imut()

c = iadd(1,2)
print("%\n\n",c)

d = imut(3,4)
print("%\n",d)
```

第十四章、多线程　　例程《thread_method.txt》

```
# 创建一个线程、并操作该线程的步骤：

# 一、定义线程变量
  thread = THREAD()   #定义一个线程变量，THREAD()是内置函数

# 二、写线程函数，注意在线程函数中，不能使用 return 指令退出线程
  def ThreadFunc(x,y)   #线程函数名没有限制，x,y 是线程函数的参数
  {
      for i in range(0,500,1)
      {
          print("打印线程参数：  %      %\n",x,y)
      }
  }

# 三、利用线程方法创建线程
  thread.CreateThread(ThreadFunc,1,2)   #无返回
  # CreateThread 是个内置的线程创建方法，该方法的第一个参数必须是
  # 线程函数名，后面的参数是线程函数的参数，必须一一对应，如该例
  # 中，实参 1 对应形参 x，实参 2 对应形参 y，顺序不能错。

# 四、新创建的线程默认是处于悬挂状态，不会执行，需要启动线程
  thread.strat()   #无参数、无返回。
  # start()操作只是将 thread 对应的线程设置成执行状态，并不会立即退出当前主线程
  # 当线程队列执行到 thread 线程时，发现它是执行状态，就执行该线程

# 五、将线程设为悬挂状态，暂停执行
  thread.suspend()   #无参数、无返回。
  # suspend()操作只是将 thread 对应的线程设置成悬挂状态，并不会立即退出当前主线程
  # 当线程队列执行到 thread 线程时，发现他被悬挂，就跳过该线程，继续执行其它线程

# 六、线程同步
  thread.join()   #无参数、无返回。
  # join()只是将主线程设置为悬挂状态，并不会立即退出主线程，等到当前主线程时间片
  # 执行完毕，在下一个主线程的时间片到来时，才会生效。

# 七、线程互斥
  thread.locked()   #无参数、无返回。
  thread.unlock()   #无参数、无返回。
  # 当不同的线程需要操纵同一个变量、或事件，为避免“碰撞”破坏数据，
  # 使用 locked()方法，限制线程切换，直到当前线程“干完事件”，
  # 再使用 unlock()方法释放限制权，继续多线程工作

# 八、获取线程 ID
  thread.getlockID()    #无参数，返回当前线程的 ID 号。
  # 在使用 thread = THREAD()生成空的线程对象时，并没有分配线程 ID，当使用
  # thread = thread.CreateThread()创建线程时，才分配线程 ID，并将线程函数
  # 与线程对象 thread 进行绑定。

# 九、强制进行线程切换
  thread.threadchange()  # 无参数，无返回，并且，立即退出当前线程。

  # 十、如果子线程没有申请主线程等待，并且子线程工作量有很大，那么，允许主线程先结束，并不
会影响子线程的继续执行，主线程结束后，系统会等到所有的子线程结束后，才会退出程序。


# 例程 1   创建一个线程，并启动该线程
def ThreadFun()      #写线程函数，在线程里循环打印 100 行
{
```

```
    for i in range(0,500,1)
    {
        print("%       %\n","thread",i)
    }
}
thread = THREAD()          # 创建一个空的线程对象
    # 将线程对象与线程函数进行绑定，分配线程 ID 号，并将该线程放进线程队列
thread = thread.CreateThread(ThreadFun)   # 默认新创建的线程处于悬挂状态，不会执行
    # start()方法，将 thread 线程设置为运行状态，并不会立即退出当前线程，等到下一次线程
    # 队列执行到该 thread 线程时，发现处于运行状态，才执行线程任务。
thread.start()   # 线程的 start()方法，将 thread 线程设置为运行状态
# 主线程程序
for i in range(0,500,1)
{
    print("%       %\n","main",i)
}
print("\n")
print("%\n","主线程执行结束")


# 例程 2   线程同步。创建一个线程，让主线程等待子线程执行结束后
# 很多时候，我们希望在主线程中，等待子线程完成任务后，在继续主线程的
# 任务，这就需要线程同步——thread.join()线程等待
def ThreadFun()   #写线程函数，在线程里循环打印 100 行
{
    for i in range(0,100,1)
    {
        print("%       %\n","       thread                          ",i)
    }
}
thread = THREAD()   #定义一个线程变量
thread = thread.CreateThread(ThreadFun)   #创建线程，新线程处于悬挂状态
thread.start()   #启动当前 thread 子线程运行
thread.join()
# 上句的目的是要求主线程在此处等待，等到子线程执行完毕，再继续主线程
# 有了线程同步 join()，就能保证只有子线程 100 行打印完后，才会继续主线程
# 运行，即，打印"主线程执行结束"
# 但是，多线程的第一个时间片总是分配给主线程，因此，thread.join()只能在
# 第一个时间片结束后，才生效。
# 如果主线程占时很短，主线程结束，并不影响子线程执行。

for i in range(0,100,1)
{
    print("%       %\n","main                          ",i)
}
print("\n")
print("%\n","主线程执行结束")
print("\n")


# 例程 3   多线程同步
def ThreadFun()
{
    for i in range(0,100,1)
    {
        print("%       %\n","thread",i)
    }
}
def ThreadFun1()
```

```
{
    for i in range(0,100,1)
    {
        print("%       %\n","thread111111111",i)
    }
}
thread = THREAD()
thread = thread.CreateThread(ThreadFun)
thread1 = THREAD()
thread1 = thread1.CreateThread(ThreadFun1)
thread.start()
thread1.start()
for i in range(0,50,1)
{
    print("%       %\n","main          ",i)
}
thread.join()
thread1.join()
for i in range(0,50,1)
{
    print("%       %\n","main after join()",i)
}
print("\n")
print("%\n","主线程执行结束")
print("\n")


# 例程 4   携带参数的子线程
def ThreadFun(x,y)   #写线程函数，传递两个参数  x,y
{
    for i in range(0,100,1)
    {
        print("%       %       %\n","thread",x+y,i)   #打印两个参数之和
    }
}
thread = THREAD()   #定义一个线程变量
# 注意：CreateThread()函数的第一个参数是线程函数名，从第二个参数开
# 始，分别对应线程函数的参数。即 CreateThread()比 ThreadFun()多一个
# 参数，多出的这个参数（线程函数名 ThreadFun）放在最前面的位置。
# 线程函数名作为第一个参数，实参 5 对应形参 x，实参 7 对应形参 y，保证顺序
thread = thread.CreateThread(ThreadFun,5,7)   #创建线程，新线程处于悬挂状态
thread.start()   #启动当前 thread 子线程运行
thread.join()   #要求主线程等待子线程
for i in range(0,100,1)
{
    print("%       %\n","main",i)
}
print("\n")
print("%\n","主线程执行结束")
print("\n")
# 运行结果：子线程打印的是实参 5+7 的值


# 例程 5   同一个线程函数，可以创建多个独立运行的线程
def ThreadFun(x)   #写线程函数，传递一个参数  x
{
    for i in range(0,100,1)
    {
        print("%       %       %\n","                  thread",x,i)   #打印两个参数之和
        if i == 99
```

```
        {
            print("\n")
        }
    }
}
list = []
# 同一个线程函数 ThreadFun，可以创建多个线程，多个线程独立运行
for i in range(0,5,1)
{
    thread = THREAD()   #定义一个线程变量
    thread = thread.CreateThread(ThreadFun,i)   #创建线程，新线程处于悬挂状态
    list.append(thread)
}
for i in range(0,5,1)
{
    thread = list[i]
    thread.start()   #启动当前 thread 子线程运行
    thread.join()    #要求主线程等待子线程
}
# 主线程默认是立即开启的
for i in range(0,100,1)
{
    print("%       %\n","                        main",i)
}
print("\n")
print("%\n","主线程执行结束")
print("\n")
# 同一个线程函数可以建立多个独立运行的线程


# 例程 6   多个线程共用一个变量的情况下，防写数据冲突可以加锁
# 多个线程共用一个变量时，一个线程写数据时，为避免冲突可以加锁
# thread.locked()加锁，短时间独占变量，避免冲突
# thread.unlock()解锁。
global a = 0   #定义全局变量，多线程共用变量
thread = THREAD()
def ThreadFun(thread)   # 线程函数包含一个参数，即，线程对象 thread
{
    for j in range(0,100,1)
    {
        thread.locked()
        a = j
        thread.unlock()
        print("sub              a = %\n",a)
     }
     print("sub 结束            a = %\n",a)
}
thread = thread.CreateThread(ThreadFun,thread)
thread.start()
# 主线程开始执行
for i in range(0,100,1)
{
    thread.locked()
    b = 20000 + a
    thread.unlock()
    print("main            a = %\n",b)
}
print("\n")
print("%\n","主线程执行结束")
print("\n")
```

```
# 例程 7   获取线程 ID
def ThreadFun()
{
    for i in range(0,100,1)
    {
        print("%       %\n","thread",i)
    }
    print("\n")
    print("%\n","sub_子线程执行结束")
    print("\n")
}
thread = THREAD()                       # 此时生成的是一个空的线程对象，并未分配 ID
thread_ID = thread.get_ID()             # 此时调用 get_ID()的返回值是-1，表示未分配 ID
print("子线程 ID = %\n",thread_ID)
thread = thread.CreateThread(ThreadFun)     # 分配 ID，并将 thread 与 ThreadFun 绑定
thread_ID = thread.get_ID()
print("子线程 ID = %\n",thread_ID)
thread.start()
# 主线程程序
for i in range(0,100,1)
{
    print("%       %\n","main",i)
}
print("\n")
print("%\n","主线程执行结束")
print("\n")


# 例程 8   模拟一个线程切换信号，人为发生线程切换
# 无输入参数，无返回
# 用途：当主线程或子线程有循环查询状态（“慢动作”）时，如果没有查询到状态变化，
# 可以强制立即切换线程（不是结束线程，保留线程继续查询），避免空转浪费时间；
# 如果查询到状态变化，执行动作完成后，结束线程。
def ThreadFun(thread)
{
    for i in range(0,20,1)
    {
        print("%       %\n","       thread",i)
        thread.threadchange()                   # 每打印一行就切换线程，立即切换
    }
    print("\n")
    print("%\n","sub_子线程执行结束")
    print("\n")
}
thread = THREAD()
thread = thread.CreateThread(ThreadFun,thread)
thread.start()
# 主线程程序
count = 100
while count
{
    print("%       %\n","                  main",count)
    count = count - 1
}
print("\n")
print("%\n","主线程执行结束")
print("\n")
```

```
# 例程 9   挂起子线程
# 在多线程运行过程中，可以将一个、或多个子线程挂起，还可以重新启动
# thread.suspend()挂线程，thread.start()启动线程。
def ThreadFun()   #写线程函数
{
     for j in range(0,150,1)
     {
          print("sub                              = %\n",j)
     }
}
thread = THREAD()
thread = thread.CreateThread(ThreadFun)
thread.start()
# 主线程开始执行
for i in range(0,200,1)
{
    print("main                          = %\n",i)
    if i == 100
    {
        # suspend()操作只是将 thread 对应的线程设置成悬挂状态，并不会立即退出当前主线程
        # 当线程队列执行到 thread 线程时，发现他被悬挂，就跳过该线程，继续执行其它线程
        thread.suspend()   #   主线程运行到 100 时，将子线程挂起
        print("                                        悬挂\n")
    }
    elif i == 180
    {
        # start()操作只是将 thread 对应的线程设置成执行状态，并不会立即退出当前主线程
        # 当线程队列执行到 thread 线程时，发现它是执行状态，就执行该线程
        thread.start()      #主线程运行到 180 时，重启子线程
        print("                                        重启\n")
    }
}
print("\n")
print("%\n","主线程执行结束")
```

第十五章、人机交互　例程《gui_method》

GUI 方法：在使用 GUI 编程之前，请记住先定义一个 GUI 对象，然后使用 GUI 方法编程。

一、定义 GUI 对象 gui，以便调用 GUI 方法创建、并处理界面
gui = GUI("fine") 创建一个 Fine 窗口对象
gui = GUI("box") 创建一个 MessageBox 窗口对象

二、控制台窗口函数组
gui.HideConsoleWindow()　# 隐藏控制台窗口，建议程序开发、调试期间，不要隐藏，虚拟机报错在控制台窗口显示
gui.ShowConsoleWindow()　# 显示控制台窗口

三、MessageBox 函数组
box = GUI("box")　# 创建 MessageBox 窗口对象，通过该对象可以操作窗口
box.MessageBox(text,"确定|取消")　# 创建 MessageBox 窗口
box.MessageBoxClosed()　# 返回-1 表明 MessageBox 窗口关闭
box.MessageBoxRead()　# 关闭窗口后调用该方法，可以读出 MessageBox 关闭窗口的原因
box.CloseMessageBox()　# 关闭 MessageBox 窗口
box.SendMessageBox(str)　# 将字符串 str 发送到 MessageBox 窗口的 text 组件区

四、Fine 函数组
gui = GUI("fine")　# 创建 Fine 窗口的 GUI 对象，通过该对象操作 Fine 窗口
gui.Fine(list)　# 创建 Fine 窗口
gui.FineClosed()　# 返回-1 表明 Input 窗口关闭
gui.FineReady()　# 返回-1 表明无数据，返回 0 表明有数据
gui.FineRead()　# 读取数据，一次性将 Fine 窗口上的所有录入数据（edit 和 listbox）打包读出

# 前两个参数指定打印起始位置，第三各参数指定字体、字号、颜色(""空字符串使用默认字库)， 第四各参数是被打印字符串。
gui.TextOut(x,y,"|宋体|20|255|0|0","被打印字符串")
gui.CloseFine()　# 关闭 Fine 创建的窗口

gui.SendEdit(list)　# 将 list 列表项，发往 Fine 窗口的 edit 组件上，每一列表项按顺序应一个 edit 组件
gui.SendText(list)　# 将 list 列表项，发往 Fine 窗口的 text 组件上，每一列表项按顺序应一个 text 组件
gui.SendCombobox(list)　# 将 list 列表项，发往 Fine 窗口的 listbox 组件上，每一列表项对按顺序应一个 listbox 组件
gui.SendTextbox(list)　# 将 list 列表项，发往 Fine 窗口的 textbox 组件上，只允许有一个 textbox 组件
gui.SendEditbox(list)　# 将 list 列表项，发往 Fine 窗口的 editbox 组件上，只允许有一个 editbox 组件
gui.SendButton(list)　# 将 list 列表项，发往 Fine 窗口的 button 组件上，每一列表项按顺序应一个 button 组件
gui.SendImage(list)　# 将 list 列表项，发往 Fine 窗口的 image 组件上，每一列表项按顺序应一个 image 组件

gui.Drawing(list)　# 参数列表是绘图指令
gui.DrawPackage(list)　# 参数是一组绘图指令，使用该指令前，需使用 DrawPackageDone()检测 DrawPackage 绘图包指令的状态，只有 DrawPackageDone()返回 False，方可使用该指令。
gui.DrawPackageDone()　# 无参数，返回 True 表明上一个 DrawPackage 指令尚未执行完毕(忙)，返回 True 表明执行完毕(闲)。
gui.PlayMedia(str)　# str 是音视频文件名

注意：在一个 Fine 窗口中，可能有多个 edit、text、combobox 组件，在向 Fine 窗口发送数据时，需要分类发送。
SendEdit(list)向 edit 类组件发送数据，每次发送时，会同时给每一个 edit 组件发送数据，即，参数 2 是个列表参数，列表项数等于 edit 组件的数量，并按创建 Fine 窗口时的顺序排列。
SendText(list)向 text 类组件发送数据，每次发送时，会同时给每一个 text 组件发送数据，即，参数 2 是个列表参数，列表项数等于 text 组件的数量，并按创建 Fine 窗口时的顺序排列。
SendCombobox(list)向 combobox 类组件发送数据，每次发送时，会同时给每一个 combobox 组件发送数据，即，参数 2 是个列表参数，列表项数等于 combobox 组件的数量，并按创建 Fine 窗口时的顺序排列。

每个 Fine 窗口中，只允许配置一个 textbox、一个 editbox 和一个 listbox 组件，因此，SendTextbox(list)和 SendEditbox(list)中，列表参数 list 只有一项。不支持向 listbox 组件发送消息。

另外，也不支持向 menu 组件发送消息。

```
# 例程 1：gui.HideConsoleWindow()和 gui.ShowConsoleWindow()

# gui.HideConsoleWindow()，无参数，无返回，功能：用于隐藏控制台窗口
# gui.ShowConsoleWindow()，无参数，无返回，功能：如果关掉了控制台窗口，你可以用该函数重新显示控制台窗口。

gui = GUI("fine")　# 定于 GUI 对象 gui
gui.HideConsoleWindow()　# 隐藏控制台窗口，终于抛弃了那个“黑乎乎”的窗口

time = TIME()　# 定义 TIME 对象
time.sleep(2000)　# 等待两秒
gui.ShowConsoleWindow()　# “黑乎乎”的显示控制台窗口又回来了
```

```
# 例程 2：messagebox1.fin
# MessageBox 的应用举例，单按钮
# 参数 1 是警告信息，参数 2 是按钮组合字符串
# 返回所创建窗口的资源 ID
# 功能：用于弹出消息框，显示提示信息，按钮组合

box = GUI("box")                                    # 定义一个 MessageBox 窗口的 GUI 对象，通过对象 box 调用 MessageBox 的方法
#gui.HideConsoleWindow()                            # 建议程序开发、调试期间，不要隐藏控制台窗口（虚拟机报错都在该窗口）。

box.MessageBox("水里有鳄鱼！","确定")    # 创建 MessageBox 窗口，返回资源 ID
while box.MessageBoxClosed() != -1              # 等待关闭窗口消息
{
      PowerDown(3)                                  # 低功耗设置（分 1、2、3 级，基数越大、功耗越低）
}

# 当窗口关闭后，可以使用 gui.MessageBoxRead(box)读取关闭窗口的原因
x = box.MessageBoxRead()
if x == -1
{
    print("%\n","点击了窗口右上角的 X 按钮，窗口关闭！")
}
else
{
    print("%\n","点击了窗口中的‘确定按钮’，窗口关闭！")
}

# 该例程中，只有一个确定按钮，读出的 x 值只有-1、0 两种可能。
# 当 MessageBox 只有一个按钮时，通常不用调用 gui.MessageBoxRead(box)数据



# 例程 3：messagebox2.fin
# MessageBox 的应用举例，双按钮
# 参数 1 是警告信息，参数 2 是按钮组合字符串
# 返回所创建窗口的资源 ID
# 功能：用于弹出消息框，显示提示信息，按钮组合

box = GUI("box")                                    # 定义创建 MessageBox 的 GUI 对象 box
#gui.HideConsoleWindow()                    # 建议程序开发、调试期间，不要隐藏控制台窗口（虚拟机报错都在该窗口）。

box.MessageBox("水里有鳄鱼！","确定|取消")          # 创建 MessageBox 窗口，返回资源 ID
while box.MessageBoxClosed() != -1    {PowerDown(3)}    # 等待关闭窗口消息
x = box.MessageBoxRead()                                # 读取关闭窗口的原因

if x == 0
{
    print("%\n","你点击了‘确定’按钮")
}
elif x == 1
{
    print("%\n","你点击了‘取消’按钮")
}
elif x == -1
{
    print("%\n","你点击了窗口右上角的 X 按钮")
}

# 对于有两个按钮的 MessageBox 窗口，读取的值只可能是-1、0、1 三种之一



# 例程 4：messagebox3.fin
# MessageBox 的应用举例，三按钮
# 参数 1 是警告信息，参数 2 是按钮组合字符串
# 返回所创建窗口的资源 ID
# 功能：用于弹出消息框，显示提示信息，按钮组合

box = GUI("box")                                            # 定义 MessageBox 窗口的 GUI 对象
#gui.HideConsoleWindow()                                    # 建议程序开发、调试期间，不要隐藏控制台窗口（虚拟机报错都在该窗口）。

box.MessageBox("水里有鳄鱼！","是|否|取消")          # 创建 MessageBox 窗口，返回资源 ID
while box.MessageBoxClosed() != -1 {PowerDown(3)}       # 等待关闭窗口、或作出按钮选择，退出循环
x = box.MessageBoxRead()                                    # 读取关闭窗口的原因

if x == 0
{
    print("%\n","你点击了‘是’按钮")
}
elif x == 1
{
    print("%\n","你点击了‘否’按钮")
}
elif x == 2
{
    print("%\n","你点击了‘取消’按钮")
```

```
}
else
{
    print("%\n","点击了右上角 X 按钮")
}

# 对于三按钮读出的值为-1、0、1、2 数据之一
# 编程中，可以根据窗口关闭的原因，控制程序转向。



# 例程 5：messagebox4.fin
# MessageBox 的应用举例——SendMessageBox 和 CloseMessageBox，分别是：向消息框发送信息、关闭窗口

# 参数 1 是警告信息，参数 2 是按钮组合字符串，返回所创建窗口的资源 ID
# 功能：用于弹出消息框，显示提示信息，按钮组合

box = GUI("box")                                    # 定义 MessageBox 的 GUI 对象
#gui.HideConsoleWindow()                            # 建议程序开发、调试期间，不要隐藏控制台窗口（虚拟机报错都在该窗口）。

time = TIME()

box.MessageBox("水里有鳄鱼！","确定")               # 创建 MessageBox 窗口，返回窗口资源 ID
while box.MessageBoxClosed() != -1                  # 等待关闭窗口消息
{
    PowerDown(3)                                        # 低功耗设置（分 1、2、3 级，基数越大、功耗越低）
    time.sleep(2000)                                    # 等待 2 秒
    box.SendMessageBox("我是超人！我不怕！")    # 更新显示内容
    time.sleep(2000)                                    # 等待 2 秒
    box.CloseMessageBox()                               # 关闭 MessageBox 窗口
}
```

Fine 窗口提供了 10 个常用的组件，通过这些组件组合，可以满足大部分应用程序的需求。注意：Fine 中组件的含义与 windows 中的组件含义有所不同！

1、menu 组件，在一个 Fine 窗口中可以同时设置不超过 10 个主菜单，每个主菜单可以设置不超过 15 个子菜单功能键。

2、edit 组件，在一个 Fine 窗口中可以同时设置不超过 15 个 edit 组件，用于从界面录入数据（字符串、整数、浮点数）。

3、text 组件，在一个 Fine 窗口中可以同时设置不超过 10 个 text 组件，用于显示单行字符串。

4、button 组件，在一个 Fine 窗口中可以同时设置不超过 48 个 button 组件，点击 button 按钮时，会返回该按钮对应的序号，用于指示如何运用窗口录入的数据。

5、combobox 组件，在一个 Fine 窗口中可以同时设置不超过 5 个 combobox 组件，用于从 combobox 列表框中录入数据。

6、listbox 组件，在一个 Fine 窗口中只允许设置一个 listbox 组件，用于列表项数据录入，或用于文件列表。

7、textbox 组件，在一个 Fine 窗口中只允许设置一个 textbox 组件，用于显示多行数据，双击该区域可以返回被击中的“行”信息，该组件中的数据不可编辑。

8、editbox 组件，在一个 Fine 窗口中只允许设置一个 editbox 组件，用于显示多行数据，该组件中的数据可编辑，但不响应双击事件。

从控件行为来看：窗口输出触发事件有四类：菜单项（menu-m-n），button，textbox("Y") ，listBox，它们导致的输出数据如下：
textbox("Y")触发事件输出两项内容：第一项“textbox”(表明触发事件)，第二项是被击中的 textbox 框中的行信息。
listBox 触发事件输出两项内容：第一项是"listbox"(表明触发事件)，第二项是被击中的 listbox 框中的行信息
menu 触发事件输出项数不固定，第一项是菜单编号，后面由窗口中 edit 组件+combocox 组件+editbox 组件的总数确定，
#　　输出顺序先是所有的 edit 组件内容，其次是所有 combobox 组件内容，最后是 editbox 内容

button 触发事件输出项数不固定，第一项是 button 编号，后面是由窗口中 edit 组件+combocox 组件+editbox 组件的总数确定，

#　　输出顺序先是所有的 edit 组件内容，其次是所有 combobox 组件内容，最后是 editbox 内容

text 组件只是显示作用，既不会触发事件，也不会输出数据。

#　另外两个组件放在其它章节讲。

9、image 组件，在 Fine 窗口设置图片，最多一个窗口可以放置 16 幅图片。

10、media 组件，在 fine 窗口设置音视频，一个窗口最多可以设置 4 个音视频。

```
# 下面通过例程逐一介绍 Fine 控件的功能。

# 例程 6：finewindow.fin
# Fine 界面函数组应用——创建 Fine 窗口

gui = GUI("fine")                          # 定义一个 Fine 窗口的 GUI 对象
gui.HideConsoleWindow()                    # 隐藏控制台窗口

# 创建一个 Fine 窗口至少需要两个窗口元素：窗口标题和窗口位置、窗口宽度和高度

title    = "我的 Fine 窗口"                # Fine 窗口标题
size     = [20,10,80,30]     # 指定窗口位置（单位：字节），参数 1 是水平起始点位置，2 是垂直起始位置，3 是窗口宽度，参数 4 是窗口高度

list = [title,size]                        # 将设计的窗口元素打包成一个列表

gui.Fine(list)                             # 创建 Fine 窗口，并返回窗口资源 ID 号
while gui.FineClosed() != -1               # 循环检测窗口关闭消息，如果 gui.FineClosed(id) = -1 表明窗口已关闭
{
    PowerDown(3)                           # 低功耗设置，这个很重要，在窗口界面任务线程中，大部分时间是空闲的的，该设置可以降低功耗
}

# PowerDown(n)的参数 n 取值只能为 1、2、3 中之一，PowerDown(3)设置功耗最低
# 该例程只是创建了一个 Fine 窗口，没有实现任何功能，下面将逐步添加功能



# 例程 7：menu.fin
# Fine 界面函数组应用——menu 组件和 TextOut 打印方法
# 将在窗口添加菜单按钮

gui = GUI("fine")
gui.HideConsoleWindow()

title    = "窗口组件的创建和使用"
winSize = [20,10,60,30]

menu1    = ["文件","menu","新建","打开","保存","退出"]    # 参数 1 是主菜单名（不能为空），参数 2 是菜单类控件，后面是子菜单功能键
menu2    = ["编辑","menu","撤销","复制","粘贴","删除"]

list = [title,winSize,menu1,menu2]                  # 将窗口设计元素打包成列表

gui.Fine(list)                                      # 创建 Fine 窗口，并返回窗口资源 ID 号 num
while gui.FineClosed() != -1                        # 循环检测窗口关闭消息（返回-1 表明窗口已关闭，退出循环）
{
    PowerDown(3)                                    # 低功耗设置
    if gui.FineReady() == -1 {continue}             # 检测是否准备好数据（-1 表明没有数据输出，0 表明有数据输出）
    x = gui.FineRead()              # 读取数据（列表），列表的第一项是指令，后面各项分别对应 edit\combobox\editbox 控件录入的数据

    # 该例程只展示了菜单控件，没有其他控件，因此，输出 x 列表只有一项，即：菜单编号

    str = sprint("列表项数：%\n",len(x))    # 构造字符串

    #gui.TextOut 用于在窗口显示内容，前两个参数是打印的起始位置；第三个参数为空，表示默认字库；第四个参数是被打印的字符串
    gui.TextOut(2,2,"",str)

    str = sprint("菜单编号：%\n",x[0])
    # 打印列表的第一项（如：menu-0-0，对应菜单->文件->新建；menu-1-2，对应菜单->编辑->粘贴；）
    gui.TextOut(2,4,"|微软雅黑|26|255|0|0",str)    #字体微软雅黑，字号 26，RGB(255,0,0) 红色
}



# 例程 8：text.fin
# Fine 界面函数组应用——text 控件应用示例
# 点击菜单按钮，对应的信息将显示在 text 框内

gui = GUI("fine")
#gui.HideConsoleWindow()

title    = "text 组件的创建和使用"
winSize = [20,10,60,30]

menu1    = ["文件","menu","新建","打开","保存","退出"]    # 参数 1 是主菜单名，参数 2 是 menu 类控件，后面是子菜单功能键
```

```
menu2    = ["编辑","menu","撤销","复制","粘贴","删除"]    # 参数 1 是主菜单名，参数 2 是菜单类控件，后面是子菜单功能键

text1    = ["  文件菜单  ","text",20,3,20,1]        # 参数 1 是 text 控件标题（可以为空""），参数 2 是 text 类控件，后面 text 控件的位置和大小
text2    = ["  编辑菜单  ","text",20,6,20,1]

list = [title,winSize,menu1,menu2,text1,text2]              # 将窗口设计元素打包成列表

gui.Fine(list)                                             # 创建 Fine 窗口，并返回窗口资源 ID 号 num
while gui.FineClosed() != -1                                # 循环检测窗口关闭消息（返回-1 表明窗口已关闭，退出循环）
{
    PowerDown(3)                                           # 低功耗设置
    if gui.FineReady() == -1 {continue}       # 检测是否准备好数据（-1 表明没有数据输出，0 表明有数据输出）
    x = gui.FineRead()      # 读取数据（列表），列表的第一项是指令，后面各项分别对应 edit\combobox\editbox 控件录入的数据

    # 该例程只展示了菜单控件，虽然增加了 text 控件，但是 text 控件并不录入数据，因此，输出 x 列表只有一项，即菜单及编号

    str1 = x[0].beforechr("-")                             # 提取字符串 x[0]中第一个“-”之前的部分（menu）
    str2 = x[0].afterchr("-")                              # 提取字符串 x[0]中第一个“-”之后的部分(0-n、或 1-n，n 是子菜单功能件编号)
    str3 = str2.beforechr("-")                             # 提取字符串 str2 中第一个“-”之前的部分（0 或 1）
    str    = str1 + "-" + str3                             # str 应该是 menu-0 或 menu-1

    if str == "menu-0"                                     # 如果点击的是“文件”类菜单
    {
        str1 = x[0]                                        # 将返回值显示在“文件菜单”控件中
        str2 = ""                                          # 清空“编辑菜单”控件
        gui.SendText([str1,str2])              # str1,str2 的顺序对应 text1 和 text2 的顺序
    }
    elif str == "menu-1"                                   # 如果点击的是“编辑”类菜单
    {
        str1 = ""                                          # 清空“文件菜单”控件
        str2 = x[0]                                        # 将返回值显示在“编辑菜单”控件中
        gui.SendText([str1,str2])              # str1,str2 的顺序对应 text1 和 text2 的顺序
    }
}


# 例程 9：Fine 界面函数组应用——edit 控件应用示例
# 在 edit 编辑框内录入数据，当点击菜单时，会把编辑框内的数据显示到文本框内，并清除编辑框

gui = GUI("fine")
gui.HideConsoleWindow()

title    = "edit 组件演示窗口"
winSize = [20,10,60,30]

menu1    = ["文件","menu","新建","打开","保存","退出"]    # 参数 1 是主菜单名，参数 2 是 menu 类控件，后面是子菜单功能键
menu2    = ["编辑","menu","撤销","复制","粘贴","删除"]    # 参数 1 是主菜单名，参数 2 是菜单类控件，后面是子菜单功能键

text0    = ["  指令  ","text",20,2,30,1]                   # 参数 1 是 text 控件标题，参数 2 是 text 类控件，后面 text 控件的位置和大小
text1    = ["  姓名  ","text",20,4,30,1]                   # 参数 5 可以为 0，则，只显示标题
text2    = ["  年龄  ","text",20,6,30,1]                   # 参数 1 是 text 控件标题，标题可以为空""，则，不显示标题
text3    = ["  身高  ","text",20,8,30,1]

edit1    = [" name ","edit","s",20,11,12,1]               # 控件标题可以为空("")，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（字符串）
edit2    = ["  年龄  ","edit","i",20,13,12,1]              # 参数 3 指定录入数据的类型（整数）
edit3    = ["  身高  ","edit","f",20,15,12,1]              # 参数 3 指定录入数据的类型（浮点数）

list = [title,winSize,menu1,menu2,text0,text1,text2,text3,edit1,edit2,edit3]          # 将窗口设计元素打包成列表

gui.Fine(list)                                             # 创建 Fine 窗口，并返回窗口资源 ID 号 num
while gui.FineClosed() != -1                                # 循环检测窗口关闭消息（返回-1 表明窗口已关闭，退出循环）
{
    PowerDown(3)                                           # 低功耗设置
    if gui.FineReady() == -1 {continue}       # 检测是否准备好数据（-1 表明没有数据输出，0 表明有数据输出）
    x = gui.FineRead()          # 读取数据（列表），列表的第一项是指令，后面各项分别对应 edit\combobox\editbox 控件录入的数据

    # 该例程增加了 3 个 edit 控件，因此，输出 x 列表有 4 项，即（指令 menu，edit1 录入内容，edit2 录入内容，edit3 录入内容）

    str1 = x[0]                                            # 取出第一项（指令）
    str2 = x[1]                                            # 取出第二项（edit1 录入的字符串）
    str3 = itoc(x[2])                                      # 取出第三项（edit2 录入的整数，并将该整数转换成字符串）
    str4 = itoc(x[3])                                      # 取出第四项（edit3 录入的浮点数，并将该浮点数转换成字符串）

    gui.SendText([str1,str2,str3,str4])      # 将四个字符串显示在四个 text 控件上
    gui.SendEdit(["","",""])                 # 清空所有 3 个 edit 控件
}

# menu 组件触发窗口返回数据：返回的数据是个列表，列表第一项表明返回数据的触发
# 信号（某一菜单项），后面跟随的是 edit 组件上的数据（edit 组件的输出顺序同 list
# 参数的打包顺序）。text 组件是静态组件，不会对外输出数据。


# 例程 10：Fine 界面函数组应用——button 控件应用示例
# 在窗口继续添加按钮 button，当点击按钮时，把编辑框内的数据转移到文本框内
```

```
gui = GUI("fine")
gui.HideConsoleWindow()

title     = "edit 窗口"
winSize = [20,10,60,30]

text0     = ["  指令  ","text",20,2,30,1]                          # 参数 1 是 text 控件标题，参数 2 是 text 类控件，后面 text 控件的位置和大小
text1     = ["  姓名  ","text",20,4,30,1]
text2     = ["  年龄  ","text",20,6,30,1]
text3     = ["  身高  ","text",20,8,30,1]

edit1     = [" name ","edit","s",20,11,12,1]        # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（字符串）
edit2     = ["  年龄  ","edit","i",20,13,12,1]      # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（整数）
edit3     = ["  身高  ","edit","f",20,15,12,1]      # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（浮点数）

button1    = ["录入","button",30,18,8,1]                          # 参数 1 是 button 控件标题，参数 2 是 button 类控件，后面是控件位置和大小
button2    = ["修改","button",40,18,8,1]

list = [title,winSize,text0,text1,text2,text3,edit1,edit2,edit3,button1,button2]              # 将窗口设计元素打包成列表

gui.Fine(list)                                                  # 创建 Fine 窗口，并返回窗口资源 ID 号 num
while gui.FineClosed() != -1                                    # 循环检测窗口关闭消息（返回-1 表明窗口已关闭，退出循环）
{
    PowerDown(3)                                                # 低功耗设置
    if gui.FineReady() == -1 {continue}       # 检测是否准备好数据（-1 表明没有数据输出，0 表明有数据输出）
    x = gui.FineRead()        # 读取数据（列表），列表的第一项是指令，后面各项分别对应 edit\combobox\editbox 控件录入的数据

    # 该例程增加了 3 个 edit 控件，因此，输出 x 列表有 4 项，即（指令 menu，edit1 录入内容，edit2 录入内容，edit3 录入内容）

    str1 = x[0]                                                 # 取出第一项（指令），可以按照指令去使用录入数据
    str2 = x[1]                                                 # 取出第二项（edit1 录入的字符串）
    str3 = itoc(x[2])                                           # 取出第三项（edit2 录入的整数，并将该整数转换成字符串）
    str4 = itoc(x[3])                                           # 取出第四项（edit3 录入的浮点数，并将该浮点数转换成字符串）

    gui.SendText([str1,str2,str3,str4])      # 将四个字符串显示在四个 text 控件上
    gui.SendEdit(["","",""])                 # 清空所有 3 个 edit 控件
    #gui.SendButton(["luru","xiugai"])         # 修改按钮上的标题
}

# 能触发窗口输出数据的事件有：点击菜单项，点击 button 按钮
# 输出的内容：第一项是触发输出的事件（点击了菜单项或按钮），如果没有 edit 组件
# 就只有这一项；如果有 edit 组件，后面每一项对应一个 edit 组件的内容。



# 例程 11：kousuan.fin
# 100 以内加减法口算，自动出题、批改程序

gui = GUI("fine")
gui.HideConsoleWindow()
math = MATH()

total = 0          # 记录做题总数
path = 0           # 记录答对题数
error = 0          # 记录答错题数

global isNewnum = 0           # 是否出国题目，0 未出题，1 已出题
global firstnum = 0           # 第一个运算符，全局变量
global secondnum = 0          # 第二个运算符，全局变量
global symbol = ""            # 运算符号，全局变量

def chuti()             # 出题函数
{
    firstnum = math.rand(0,100)        # 获取 0~100 的随机数，作为加数
    if math.rand(0,1) == 0             # 获取随机数 0 或 1，0 代表加法，1 代表减法
    {
        symbol = "+"
        # 如果是加法，则再产生一个 100-firstnum 的随机数，作为被加数，避免超出 100
        secondnum = math.rand(0,100-firstnum)       # 获取被加数
    }
    else
    {
        symbol = "-"
        # 如果是减法，则再产生一个 0~firstnum 的随机数，作为减数，避免超出结果<0
        secondnum = math.rand(0,firstnum)       # 获取被加数
    }
    isNewnum = 1       # 表明已出过题目
}

def pigai()             # 批改函数
{
    if symbol == "+"                # 如果是加法
    {
        result = firstnum + secondnum       # 两数相加
    }
    elif    symbol == "-"
```

```
    {
        result = firstnum - secondnum       # 相减
    }
    return result                           # 返回结果
}

# 设计界面
title = "100 以内加减法训练"
size  = [20,5,50,22]

text1 = ["总题目数          |宋体|20","text",22,2,10,1]    # "Courier New"   20
text2 = ["答对题目数        |宋体|20|0|200|0","text",22,4,10,1]    # 显示绿色
text3 = ["答错题目数        |宋体|20|255|0|0","text",22,6,10,1]    # 显示红色
text4 = ["","text",9,10,5,1]          # 显示被加数
text5 = ["","text",16,10,3,1]         # 显示运算符
text6 = ["","text",21,10,5,1]         # 显示加数
text7 = ["","text",28,10,3,1]         # 显示等号
edit  = ["","edit","i",33,10,5,1]     # 录入答案

button1 = ["出题","button",9,14,10,1]   # 出题按钮
button2 = ["批改","button",22,14,10,1]   # 批改按钮

list = [title,size,text1,text2,text3,text4,text5,text6,text7,edit,button1,button2]   #打包参数

gui.Fine(list)                      # 创建界面
gui.SendText([itoc(total),itoc(path),itoc(error),"","","","="])      # 显示界面初始数据

while gui.FineClosed() != -1                        # 检查窗口，如果窗口没有被关闭，执行 while 循环体
{
    PowerDown(3)                                    # 低功耗
    if gui.FineReady() == -1 { continue }      # 检查窗口是否有数据输出请求，如果没有跳转
    x = gui.FineRead()                              # 读出窗口输出的数据

    if x[0] == "button-0"                 # x[0]是窗口输出的指令，本窗口只有两个 button 按钮，button-0 表示出题按钮，button-1 表示批改按钮
    {
        chuti()                 # 调用出题函数，赋值 firstnum，secondnum，symbol
        gui.SendText([itoc(total),itoc(path),itoc(error),itoc(firstnum)," "+symbol,itoc(secondnum)," ="])      # 显示界面初始数据
        gui.SendEdit([""])    # 清除 edit 组件上的内容
    }
    elif x[0] == "button-1"       # button-1 表示点击了批改按钮
    {
        if isNewnum == 0            # 如果没有出题，先点击批改，则弹出对话框，提示错误
        {
            box = GUI("box")
            box.MessageBox("请先出题，再批改！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            continue          # 跳转程序，进入主窗口查询
        }
        total = total + 1
        result = pigai()        # 返回运算结果
        if result == x[1]       # 判断答案是否正确，窗口输出的第二项是 edit 组件录入的数据
        {
            path = path +1    # 答对  path +1
            # 更新答题总数、答对题数、答错题数
            gui.SendText([itoc(total),itoc(path),itoc(error),itoc(firstnum)," "+symbol,itoc(secondnum)," ="])      # 更新窗口显示
            box = GUI("box")
            box.MessageBox("恭喜你答对了！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
        }
        else
        {
            error = error + 1    # 答错  error + 1
            # 更新答题总数、答对题数、答错题数
            gui.SendText([itoc(total),itoc(path),itoc(error),itoc(firstnum)," "+symbol,itoc(secondnum)," ="])      # 更新窗口显示
            box = GUI("box")
            box.MessageBox("答错了，继续加油！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
        }
        # 清除上次的题目
        gui.SendText([itoc(total),itoc(path),itoc(error),"","","","="])      # 更新窗口显示
        gui.SendEdit([""])    # 清除 edit 组件上的内容
        isNewnum = 0              # 表明当前题目已批改过
    }
}



# 例程 12：Fine 界面函数组应用——textbox 控件应用示例（禁止 textbox 双击输出）
# 点击“确定”按钮，将编辑框内的数据转移到 textbox 框内，双击 textbox 框内的数据，没有相应。

gui = GUI("fine")
#gui.HideConsoleWindow()

title    = "我的数据录入窗口"
winSize = [20,10,60,30]
```

```
edit1    = [" name ","edit","s",20,2,16,1]      # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（字符串）
edit2    = [" 年龄 ","edit","i",20,4,10,1]      # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（整数）
edit3    = [" 身高 ","edit","f",20,6,12,1]      # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（浮点数）

button1   = ["确定","button",40,9,8,1]            # 参数 1 是 button 控件标题，参数 2 是 button 类控件，后面是控件位置和大小

# 注意：在一个 Fine 窗口中，只允许配置一个 textbox 控件。textbox 控件上的内容不可编辑。
textbox   = ["","textbox","N",2,12,38,9]          # 参数 1 是 textbox 控件标题，可以为空，2 是 textbox 类控件，3 是“N”表明不响应双击事件。

list = [title,winSize,edit1,edit2,edit3,button1,textbox]              # 将窗口设计元素打包成列表

gui.Fine(list)                                                      # 创建 Fine 窗口，并返回窗口资源 ID 号 num
while gui.FineClosed() != -1                                          # 循环检测窗口关闭消息（返回-1 表明窗口已关闭，退出循环）
{
    PowerDown(3)                                                    # 低功耗设置
    if gui.FineReady() == -1 {continue}       # 检测是否准备好数据（-1 表明没有数据输出，0 表明有数据输出）
    x = gui.FineRead()            # 读取数据（列表），列表的第一项是指令，后面各项分别对应 edit\combobox\editbox 控件录入的数据

    # 该例程增加了 3 个 edit 控件，因此，输出 x 列表有 4 项，即（指令 menu，edit1 录入内容，edit2 录入内容，edit3 录入内容）

    str1 = x[0]                                                     # 取出第一项（指令），可以按照指令去使用录入数据
    str2 = x[1]                                                     # 取出第二项（edit1 录入的字符串）
    str3 = itoc(x[2])                                             # 取出第三项（edit2 录入的整数，并将该整数转换成字符串）
    str4 = itoc(x[3])                                             # 取出第四项（edit3 录入的浮点数，并将该浮点数转换成字符串）

    gui.SendTextbox([str1 + "\n"])            # 将字符串 str1 添加到 textbox 控件上
    gui.SendTextbox([str2 + "\n"])            # 将字符串 str2 添加到 textbox 控件上
    gui.SendTextbox([str3 + "\n"])            # 将字符串 str3 添加到 textbox 控件上
    gui.SendTextbox([str4 + "\n"])            # 将字符串 str4 添加到 textbox 控件上

    gui.SendEdit(["","",""])                  # 清空所有 3 个 edit 控件
}


# 例程 13：textbox1.fin
# Fine 界面函数组应用——textbox 控件应用示例（允许 textbox 双击输出）
# 点击确定按钮将编辑框内的数据转移到 textbox 框内，与例程 11 不同的是，这里点击 textbox 框内的数据时，
# 会把击中的行内容，显示到控制台窗口。

gui = GUI("fine")
gui.HideConsoleWindow()

title     = "textbox 窗口"
winSize = [20,10,60,30]

edit1    = [" 姓名 ","edit","s",20,2,16,1]      # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（字符串）
edit2    = [" 年龄 ","edit","i",20,4,10,1]      # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（整数）
edit3    = [" 身高 ","edit","f",20,6,12,1]      # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（浮点数）

button1   = ["确定","button",40,9,8,1]            # 参数 1 是 button 控件标题，参数 2 是 button 类控件，后面是控件位置和大小

# 注意：在一个 Fine 窗口中，只允许配置一个 textbox 控件。textbox 控件上的内容不可编辑。但，双击该控件，可以输出数据
textbox   = ["","textbox","Y",2,12,40,10]         # 参数 1 是 textbox 控件标题，可以为空，2 是 textbox 类控件，参数 3 是“Y”表明响应双击事件。

list = [title,winSize,edit1,edit2,edit3,button1,textbox]              # 将窗口设计元素打包成列表

gui.Fine(list)                                                      # 创建 Fine 窗口，并返回窗口资源 ID 号 num
while gui.FineClosed() != -1                                          # 循环检测窗口关闭消息（返回-1 表明窗口已关闭，退出循环）
{
    PowerDown(3)                                                    # 低功耗设置
    if gui.FineReady() == -1 {continue}       # 检测是否准备好数据（-1 表明没有数据输出，0 表明有数据输出）
    x = gui.FineRead()          # 读取数据（列表），列表的第一项是指令，后面各项分别对应 edit\combobox\editbox 控件录入的数据

    # 该例程有 3 个 edit 控件，因此，输出 x 列表有 4 项，即（menu 或 textbox 指令，edit1 录入内容，edit2 录入内容，edit3 录入内容）

    if x[0] == "textbox"
    {
       gui.SendTextbox([x[1] + "\n"])            # 将字符串 str1 添加到 textbox 控件上
    }
    else
    {
        str1 = x[0]                                                 # 取出第一项（指令），可以按照指令去使用录入数据
        str2 = x[1]                                                 # 取出第二项（edit1 录入的字符串）
        str3 = itoc(x[2])                                         # 取出第三项（edit2 录入的整数，并将该整数转换成字符串）
        str4 = itoc(x[3])                                         # 取出第四项（edit3 录入的浮点数，并将该浮点数转换成字符串）

        gui.SendTextbox([str1 + "\n"])            # 将字符串 str1 添加到 textbox 控件上
        gui.SendTextbox([str2 + "\n"])            # 将字符串 str2 添加到 textbox 控件上
        gui.SendTextbox([str3 + "\n"])            # 将字符串 str3 添加到 textbox 控件上
        gui.SendTextbox([str4 + "\n"])            # 将字符串 str4 添加到 textbox 控件上

        gui.SendTextbox([str1 +"    "+ str2 +"    "+ str3 +"    "+ str4 +"\n"])
    }

    gui.SendEdit(["","",""])                  # 清空所有 3 个 edit 控件
```

```
}

# 窗口输出数据的事件：menu 菜单项、button 按钮、textbox 组件（第三个参数是“Y”）。
# menu 菜单项事件输出第一项是菜单编号，外加所有 edit 组件上的内容。
# button 按钮事件输出的第一项是按钮编号，外加所有 edit 组件上的内容。
# textbox 组件（“Y”）输出的第一项是字符串"textbox"，第二项是被双击击中的行信息。



# 例程 14：combobox.fin
# Fine 界面函数组应用——combobox 控件应用示例
# 窗口有一个编辑框和两个 combobox 下拉菜单框，点击确定将上述三个窗口的内容，显示在 textbox 框内，
# 再点击 textbox 框内信息时，将击中的行信息显示在 MessageBox 弹窗内。

gui = GUI("fine")
gui.HideConsoleWindow()

sex = "男,女"
subject = "语文,数学,英语,物理,化学,生物"

title     = "Combobox 窗口"
winSize = [20,10,60,30]

edit          = ["  姓名  ","edit","s",20,2,16,1]                  # 参数 1 是 edit 控件标题，参数 2 是 edit 类控件，参数 3 指定录入数据的类型（字符串）
combobox1   = ["  性别  ","combobox",sex,20,4,8,1]                 # 参数 1 是 edit 控件标题，2 是 edit 类控件，参数 3 指定录入数据的类型（字符串）
combobox2   = ["  科目  ","combobox",subject,20,6,12,1]      # 参数 3 指定录入数据的类型（字符串）

button        = ["确定","button",40,7,8,1]                        # 参数 1 是 button 控件标题，参数 2 是 button 类控件，后面是控件位置和大小

textbox    = ["","textbox","Y",2,10,38,12]          # 参数 3 是“Y”表明双击 textbox 区域，将输出数据。


list = [title,winSize,edit,combobox1,combobox2,button,textbox]                # 将窗口设计元素打包成列表

gui.Fine(list)                                                  # 创建 Fine 窗口，并返回窗口资源 ID 号 num
while gui.FineClosed() != -1                                    # 循环检测窗口关闭消息（返回-1 表明窗口已关闭，退出循环）
{
    PowerDown(3)                                                # 低功耗设置
    if gui.FineReady() == -1 {continue}        # 检测是否准备好数据（-1 表明没有数据输出，0 表明有数据输出）
    x = gui.FineRead()              # 读取数据（列表），列表的第一项是指令，后面各项分别对应 edit\combobox\editbox 控件录入的数据

    # 该例程有 1 个 edit 控件，2 个 combobox 控件，因此，点击 button 输出的 x 列表有 4 项，
    # 即（指令 button，edit、combobox1、combobox2 上的内容）
    # 还有一个 textbox 控件，并且其第三那个参数允许双击输出，因此，双击 textbox 区域，
    # 输出列表包含两项：第一项指令（textbox），第二项被击中的行信息。

    if x[0] == "textbox"                                        # 双击 textbox 区域产生的输出
    {
        str = "被击中的 textbox 控件的行信息是：" + x[1]              # 弹出消息框
        box = GUI("box")                                          # 定义一个 MessageBox 窗口对象 box
        box.MessageBox(str,"确定")                                 # 创建 MessageBox 窗口
        while gui.MessageBoxClosed() != -1 {PowerDown(3)}         # 检测 MessageBox 窗口关闭消息
    }
    elif x[0] == "button-0"                                     # 点击 button 按钮产生的输出
    {
        str = x[1] + "," + x[2] + "," + x[3]+ "\n"
        gui.SendTextbox([str])                          # 将字符串 str1 添加到 textbox 控件上
    }

    # SendCombobox 演示：不能清空 combobox 选项，但可以设置成任意(存在的)项，如果要设置的字符串不在列表中，则默认显示第一项
    gui.SendCombobox(["女","生物"])          # 设置 combobox 控件的值。
}

# 窗口输出数据的事件：menu 菜单项、button 按钮、textbox 组件（第三个参数是“Y”）。
# menu 菜单项事件输出第一项是菜单编号，外加所有 edit 组件上的内容+所有 Combobox 组件内容。
# button 按钮事件输出的第一项是按钮编号，外加所有 edit 组件上的内容+所有 Combobox 组件内容。。
# textbox 组件（“Y”）输出的第一项是字符串"textbox"，第二项是被双击击中的行信息。



# 例程 15：inputdata.fin
# 控件的字体、字号、显示颜色控制

gui = GUI("fine")
gui.HideConsoleWindow()
szlistbox1 = "支出,收入"
szlistbox2 = "生活用品,交通,通讯,衣服鞋帽,数码产品,健身,旅游,工资,奖金"

# 记录输入界面设计
title = "账单录入"
size    = [10,3,100,36]

text1       = ["|楷体|22|255|0|0","text",24,2,17,1]                                  # text 控件的标题为空，仍然可以定义字库
button     = ["  录入  |微软雅黑|24|255|255|0","button",55,2,10,1]                   # 给 button 控件设置字库
```

```
edit0       = [" ID |微软雅黑|22|0|0|255","edit","i",12,5,12,1]                          # 给 edit 控件设置字库
combobox1= ["  收入支出  |微软雅黑|22|0|255|0","combobox",szlistbox1,40,5,8,1]     # 给 combobox 控件设置字库

edit1       = ["  录入者  ","edit","s",66,5,18,1]
edit2       = ["  日期  ","edit","s",12,8,14,1]
combobox2= ["  商品分类  ","combobox",szlistbox2,12,10,28,1]
edit3       = ["  商品名称  ","edit","s",12,12,60,1]
edit4       = ["  数量  ","edit","f",12,14,20,1]
edit5       = ["  单价  ","edit","f",12,16,20,1]
text2       = ["  合计  ","text",12,18,20,1]
textbox    = ["  录入数据  |微软雅黑|22|0|0|255","textbox","N",12,20,55,7]              # 给 textbox 控件设置字库

# 打包设计元素
list = [title,size,text1,button,edit0,combobox1,edit1,edit2,combobox2,edit3,edit4,edit5,text2,textbox]

gui.Fine(list)                              # 创建账单录入窗口，并返回资源 ID
gui.SendText(["    记录输入",""])
while gui.FineClosed() != -1                # 检查窗口关闭消息
{
    PowerDown(3)                                    # 节能设置
    if gui.FineReady() == 0                 # 检查是否有数据录入
    {
        x = gui.FineRead()          # 读取录入数据

        # x[0]是 button-0(指令）
        # 优先输出 edit：x[1]ID，x[2]数据录入人，x[3]日期,x[4]商品名称，x[5]数量，x[6]单价
        # 其次输出 combobox：x[7]收入支出，x[8]商品分类

        tm = x[3]
        if tm == "" or len(tm) < 10 or tm[4] != "-"
        {
            box = GUI("box")
            box.MessageBox("时间不能为空！并且时间格式必须是：NNNN-YY-RR!","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            continue
        }

        y = x[5]*x[6]                                           # 计算数量乘以单价，合计
        Y = sprint("%.2f",y)                                    # 将合计转化为字符串
        gui.SendText(["    票据录入    ",Y])                    # 显示合计


        time1 = x[3]+" 08:00:00"                                # 拼接开票时间（录入的票据时间）

        str = itoc(x[1]) +","+x[7]+","+x[8]+","+x[4]+"," +itoc(x[5])+","+itoc(x[6])+","+time1+"\n"
        gui.SendTextbox([str])

        # 弹窗
        box1 = GUI("box")
        box1.MessageBox("成功录入了一条记录！","确定")
        while box1.MessageBoxClosed() != -1 {PowerDown(3)}
        gui.SendEdit(["",x[2],x[3],"","",""])                   # 将查询选中的所有 edit 项记录显示出来，便于修改
        gui.SendCombobox([x[7],x[8]])                           # 将查询选中的所有 Combobox 项记录显示出来，便于修改
        gui.SendText(["    记录输入",""])
    }
}

# menu 控件不接受字库(字体、字号、颜色)设置，使用系统默认配置。
# button 控件的标题响应字库(字体、字号、颜色)设置。
# text 和 textbox 控件的标题栏和显示栏，均响应字库(字体、字号、颜色)设置。
# 其它控件（edit、editbox、listbox 和 combobox）的标题，均响应字库(字体、字号、颜色)设置，
# 但是，它们的输入栏，只响应(字体、字号)设置，不响应颜色设置

# 设置方法示例：
#       只设置字体：  "标题|字体"。项目之间用"|"隔开，如果没有标题，则为："|字体"，竖线符号不能省略。
#       只设置字体和字号：  "标题|字体|字号"。项目之间用"|"隔开。
#       设置字体字号和颜色：  "标题|字体|字号|红|绿|蓝"。项目之间用"|"隔开。

# 注意：改变字号可能导致“标题”栏、或编辑框显示不下内容，遇到这种情况，可以在标题后面添加若干空格，即可解决。



# 例程 16：editbox.fin
# Fine 界面函数组应用——editbox 控件和 listbox 文件列表应用示例
# 在窗口设置菜单、一个文件列表框、一个 editbox 编辑框，窗口创建初始，列表框显示当前目录下的文件名、或子目录，
# 当双击中某行时，将对应的行信息显示在 editbox 框内，如果显示的是文件名，点击菜单可以对文件操作。

gui = GUI("fine")
gui.HideConsoleWindow()
os   = OS()
filename = ""

title       = "listbox 和 editbox 窗口"
winSize   = [10,2,120,40]

menu        = ["文件","menu","新建","打开","保存","退出"]                              # 菜单不接受指定字库，只能使用默认值
```

```
# 注意：在一个 Fine 窗口中，只允许配置一个 editbox 控件，editbox 控件不响应鼠标双击事件。editbox 控件上的内容可以编辑
editbox   = ["","editbox",0,0,88,27]      # 参数 1 是 textbox 控件标题，可以为空，参数 2 是 textbox 类控件，参数 3 是“N”表明不响应双击事。

list = [title,winSize,menu,editbox]                # 将窗口设计元素打包成列表

gui.Fine(list)                                     # 创建 Fine 窗口，并返回窗口资源 ID 号 num
while gui.FineClosed() != -1                       # 循环检测窗口关闭消息（返回-1 表明窗口已关闭，退出循环）
{
    PowerDown(3)                                   # 低功耗设置
    if gui.FineReady() == -1 {continue}        # 检测是否准备好数据（-1 表明没有数据输出，0 表明有数据输出）
    x = gui.FineRead()           # 读取数据（列表），列表的第一项是指令，后面各项分别对应 edit\combobox\editbox 控件录入的数据

    # 该例程有一个 listbox 组件，listbox 组件可以触发窗口输出，输出数据 x 的第一项可能是双击 listbox 指令，后面是鼠标双击击中的行
    # 该例程有一个 menu 组件，菜单项也可以触发窗口输出，输出 x 的第一项只可能是 menu 菜单编号，后面是窗口输出的数据
    # 该例程中，没有 edit 和 combobox 控件，只有一个 editbox 控件，因此，x[1]是 editbox 组件上的内容

    str0 = x[0]                                    # 取出第一项（指令），可以按照指令去使用录入数据
    str1 = x[1]                                    # 取出第二项（editbox 选取的内容）

    # 如果是点击了某一菜单项时，第一项是菜单编号，第二项是文件名或目录（来自 editbox 组件）
    if str0 == "menu-0-0"                          # 新建文件
    {
        gui.SendEditbox(["FINECLEAR"])                 # 清空 editbox 内容（"FINECLEAR"是清屏指令）
    }
    elif str0 == "menu-0-1"                        # 打开文件
    {
        # 创建二级窗口，使用 listbox 显示文件列表，从文件列表中选文件
        gui1 = GUI("fine")
        title       = "listbox 窗口"
        winSize    = [15,3,64,40]

       # 注意：在一个 Fine 窗口中，只允许配置一个 listbox 控件，双击列表项将输出所击中的列表项。
        listbox    = ["文件列表","listbox",0,0,28,30]       # 参数 1 是 listbox 组件标题，参数 2 是 listbox 类组件

        list = [title,winSize,listbox]                 # 将窗口设计元素打包成列表

        gui1.Fine(list)                                    # 创建 Fine 窗口，并返回窗口资源 ID 号 num
        while gui1.FineClosed() != -1
        {
            PowerDown(3)                                   # 低功耗设置
            if gui1.FineReady() == -1 {continue}           # 检测是否准备好数据（-1 表明没有数据输出，0 表明有数据输出）
            listboxout = gui1.FineRead()               # 输出内容包含两项：第一项"listbox"(表明是文件列表)，第二项是选中的文件或目录
            filename = listboxout[1]                       # 将选中的文件名赋值给 filename
            gui1.CloseFine()                               # 关闭二级 fine 窗口
        }

        if os.pathisfile(filename)                     # 判断 filename 是否为文件名，如果是文件名，打开文件，并显示到 editbox 控件上
        {
            fp = FILEOPEN(filename,"r")                    # 打开文件（文本、只写方式），如果文件已经存在，将覆盖源文件
            if fp == False                                 # 如果打开文件失败
            {
              box = GUI("box")                                 # 定义一个 MessageBox 窗口对象 box
              box.MessageBox("打开文件失败！ ","确定")      # 弹窗显示
              while box.MessageBoxClosed() != -1 {PowerDown(3)}
            }
            else
            {
                str = fp.read()                            # 读取全部文件内容（长度限制在 3 万字节以内）
                fp.close()                                   # 关闭文件
                gui.SendEditbox(["FINECLEAR"])             # 清空 editbox 内容（"FINECLEAR"是清屏指令）
                gui.SendEditbox([str])                     # 将读出的文件内容，显示在 editbox 控件上
            }
        }
        else                                                       # 如果不是文件，将字符串（路径）显示到 editbox 控件上
        {
            gui.SendEditbox(["FINECLEAR"])                         # 清空 editbox 内容
            str = filename + "可能不是文件名，无法打开该文件！ \n"
            gui.SendEditbox([str])                                 # 在 editbox 控件上显示路径
        }
    }
    elif str0 == "menu-0-2"                    # 保存文件
    {
        # 创建文件名输入窗口，获取文件名（fine 窗口嵌套，创建二级窗口），这个二级窗口目的是获取新的文件名
        gui2 = GUI("fine")                             # 定义一个 fine 窗口对象（每一个窗口都对应一个窗口对象）
        # 设计二级窗口
        title     = "文件名输入"
        size       = [40,10,60,10]
        edit       = ["请输入文件名","edit","s",20,2,32,1]
        button    = ["确定","button",40,5,12,1]
        list = [title,size,edit,button]              #打包设计元素

        # 创建二级窗口
        filename = ""                                  # 用于保存录入的文件名，初值赋值为空
        gui2.Fine(list)
        while gui2.FineClosed() != -1                  # 检查窗口关闭消息
```

```
        {
          PowerDown(3)
          if gui2.FineReady() == -1 {continue}    # 如果没有数据输出，跳转
          x = gui2.FineRead()                     # 列表 x 有两项：指令(button-0)和 edit 控件上的数据(文件名)
          filename = x[1]                         # 取出文件名
          gui2.CloseFine()                        # 关闭窗口
        }
        if filename != ""                         # 录入的文件名不为空时，尝试保存文件
        {
            # 检查 filename 文件是否存在
            fp = FILEOPEN(filename,"r")           # 尝试以只读方式打开文件 filename
            if fp == False                        # 打开文件失败，文件可能不存在，直接写入文件
            {
              fp = FILEOPEN(filename,"w")         # 打开文件（文本、只写方式），如果文件已经存在，将覆盖源文件
              fp.write(str1)                      # 把从 editbox 控件上读出的数据写入文件 copy1001.txt 中
              fp.close()                          # 关闭已经打开的文件
              # 弹窗 MessageBox 窗口
              box = GUI("box")            # 定义弹窗对象 box
              box.MessageBox(filename+"文件写入成功！","确定")          # 弹出消息提示框
              while box.MessageBoxClosed() != -1 {PowerDown(3)}         # 等待窗口关闭消息
            }
            else       # 如果文件已经存在，弹出消息框，询问是否要覆盖
            {
                # 弹窗
                box1 = GUI("box")
                box1.MessageBox(filename+"文件已经存在，是否要覆盖原文件？","确定|取消")      # 弹出消息提示框
                while box1.MessageBoxClosed() != -1 {PowerDown(3)}
                select = box1.MessageBoxRead()                          # 读出选择结果（0——确定，1——取消）

                # 根据询问结果，执行操作
                if select == 0                                          # 如果选择确定，覆盖原文件
                {
                    fp = FILEOPEN(filename,"w")       # 打开文件（文本、只写方式），如果文件已经存在，将覆盖源文件
                    fp.write(str1)                                      # 把从 editbox 控件上读出的数据写入文件 copy1001.txt 中
                    fp.close()                                          # 关闭已经打开的文件
                    # 弹窗
                    box2 = GUI("box")
                    box2.MessageBox(filename+"文件写入成功！","确定")          # 弹出消息提示框
                    while box2.MessageBoxClosed() != -1 {PowerDown(3)}        # 等待窗口关闭消息
                }
                elif select == 1                                        # 如果选择取消，放弃保存文件
                {
                    # 弹窗
                    box3 = GUI("box")
                    box3.MessageBox("放弃保存文件！","确定")                    # 弹出消息提示框
                    while box3.MessageBoxClosed() != -1 {PowerDown(3)}        # 等待窗口关闭消息
                }
            }
        }
    }
    else      # 其它指令（menu-0-3），退出程序
    {
        # 弹窗
        box4 = GUI("box")
        box4.MessageBox("确定要退出程序吗？","确定")                                  # 弹出消息提示框
        while box4.MessageBoxClosed() != -1 {PowerDown(3)}              # 等待窗口关闭消息
        gui.CloseFine()
    }
}


# 总结
# 我们已经讲完了全部全部四类窗口输出的触发事件：菜单项（menu-m-n），button，textbox("Y") ，listBox。
# textbox("Y")触发事件输出两项内容：第一项“textbox”(表明触发事件)，第二项是被击中的 textbox 框中的行信息。
# listbox 触发事件输出两项内容：第一项是"listbox"(表明触发事件)，第二项是被击中的 listbox 框中的行信息
# menu 触发事件输出项数不固定，第一项是菜单编号，后面由窗口中 edit 组件+combobox 组件+editbox 组件的总数确定，
#       输出顺序先是所有的 edit 组件内容，其次是所有 combobox 组件内容，最后是 editbox 内容
# button 触发事件输出项数不固定，第一项是 button 编号，后面是由窗口中 edit 组件+combocox 组件+editbox 组件的总数确定，
#       输出顺序先是所有的 edit 组件内容，其次是所有 combobox 组件内容，最后是 editbox 内容
# text 组件只是显示作用，既不会触发事件，也不会输出数据。



# 例程 17 计算器 calcullater.fin

gui = GUI("fine")
gui.HideConsoleWindow()

# 设计计算机界面
title   = "计算器"
size      = [40,8,38,20]

text        = ["|微软雅黑|22|255|0|0","text",8,2,20,1]

button0   = ["C|楷体|20|0|0|255","button",5,5,5,1]
```

```
button1    = ["<--|楷体|20|0|0|255","button",26,5,5,1]

button2    = ["1","button",5,7,5,1]
button3    = ["2","button",12,7,5,1]
button4    = ["3","button",19,7,5,1]
button5    = ["÷","button",26,7,5,1]

button6    = ["4","button",5,9,5,1]
button7    = ["5","button",12,9,5,1]
button8    = ["6","button",19,9,5,1]
button9    = ["×","button",26,9,5,1]

button10 = ["7","button",5,11,5,1]
button11 = ["8","button",12,11,5,1]
button12 = ["9","button",19,11,5,1]
button13 = ["-","button",26,11,5,1]

button14 = [".","button",5,13,5,1]
button15 = ["0","button",12,13,5,1]
button16 = ["+","button",19,13,5,1]
button17 = ["=","button",26,13,5,1]

# 设计元素打包
list        = [title,size,text,button0,button1,button2,button3,button4,\
              button5,button6,button7,button8,\
              button9,button10,button11,button12,\
              button13,button14,button15,button16,button17]

gui.Fine(list)                          # 创建界面，返回资源 ID
result = ""                                     # 输入的数字字符串
have_number = 0                                   # 0 表明没有被运算数，1 表示被运算数已经存在
symbol = ""                                     # 保存运算符号

while gui.FineClosed() != -1                  # 检测界面关闭消息
{
  PowerDown(3)                                      # 低功耗设置
  if gui.FineReady() == -1 {continue}         # 如果没有检测到字符输入，跳转到 while 循环
  xlist = gui.FineRead()                            # 至此说明有按钮被按下，读取按钮键值
  x = xlist[0]

  if x == "button-0"                                          # 如果按钮键值为 0，说明是 C 清除键
  {
    gui.SendText([""])                # 清除屏幕内容
    result = ""                       # 复位变量值
    symbol = ""                         # 复位变量值
    have_number = 0                     # 复位变量值
  }
  elif x == "button-1"                # 如果按钮键值为 1，对应退格健
  {
    result.pop()                        # 抛弃掉 result 的最后一个字符
    gui.SendText([result])            # 显示字符串
  }
  elif x == "button-2"                # 如果按钮键值为 2，对应数字 1
  {
    result = result + "1"             # 拼接字符串
    gui.SendText([result])            # 显示字符串
  }
  elif x == "button-3"                # 如果按钮键值为 3，对应数字 2
  {
    result = result + "2"
    gui.SendText([result])
  }
  elif x == "button-4"                # 如果按钮键值为 4，对应数字 3
  {
    result = result + "3"
    gui.SendText([result])
  }
  elif x == "button-6"                # 如果按钮键值为 6，对应数字 4
  {
    result = result + "4"
    gui.SendText([result])
  }
  elif x == "button-7"                # 如果按钮键值为 7，对应数字 5
  {
    result = result + "5"
    gui.SendText([result])
  }
  elif x == "button-8"                  # 如果按钮键值为 8，对应数字 7
  {
    result = result + "6"
    gui.SendText([result])
  }
  elif x == "button-10"                 # 如果按钮键值为 10，对应数字 7
  {
    result = result + "7"
    gui.SendText([result])
  }
  elif x == "button-11"                   # 如果按钮键值为 11，对应数字 8
  {
```

```
    result = result + "8"
    gui.SendText([result])
}
elif x == "button-12"                    # 如果按钮键值为 12，对应数字 9
{
    result = result + "9"
    gui.SendText([result])
}
elif x == "button-14"                    # 如果按钮键值为 14，对应小数点.
{
    result = result + "."
    gui.SendText([result])
}
elif x == "button-15"                    # 如果按钮键值为 15，对应数字 0
{
    result = result + "0"
    gui.SendText([result])
}
elif x == "button-5"                     # 如果按钮键值为 5，对应除法运算符÷
{
    if have_number == 0                  # 如果被运算数不存在
    {
        if result == ""    {continue}    # 如果在输入运算符之前，没有输入数字，跳转到 while 循环起点，等待输入数字
        if result[0] == "."
        {
            box = GUI("box")
            box.MessageBox("小数点前面的 0 不可省略！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            result = ""
            gui.SendText([""])
            continue
        }
        number1 = ctoi(result)           # 将当前 result 转化为数字
        symbol = "÷"                     # 赋值运算符变量
        result = ""
        have_number = 1                  # 标记被运算数已存在
    }
    else                                 # 如果被运算数已存在
    {
        if result == ""                  # 进一步检查字符串 result 是否为空
        {
            symbol = "÷"                 # 如果 number1 已经存在，并且 result 为空，赋值 symbol
        }
        else                             # 虽然 number1 已经存在，并且 result 不为空
        {
            number1 = ctoi(result)       # 将 result 转化为数字，替换掉原来的 number1
            symbol = "÷"                 # 赋值 symbol
            result = ""
        }
    }
}
elif x == "button-9"                     # 如果按钮键值为 9，对应乘法运算符×
{
    if have_number == 0
    {
        if result == ""    {continue}
        if result[0] == "."
        {
            box = GUI("box")
            box.MessageBox("小数点前面的 0 不可省略！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            result = ""
            gui.SendText([""])
            continue
        }
        number1 = ctoi(result)
        symbol = "×"
        result = ""
        have_number = 1
    }
    else
    {
        if result == ""
        {
            symbol = "×"
        }
        else
        {
            number1 = ctoi(result)
            symbol = "×"
            result = ""
        }
    }
}
elif x == "button-13"                    # 如果按钮键值为 13，对应减法运算符-
{
    if have_number == 0
    {
        if result == ""    {continue}
        if result[0] == "."
```

```
        {
            box = GUI("box")
            box.MessageBox("小数点前面的 0 不可省略！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            result = ""
            gui.SendText([""])
            continue
        }
        number1 = ctoi(result)
        symbol = "-"
        result = ""
        have_number = 1
    }
    else
    {
        if result == ""
        {
            symbol = "-"
        }
        else
        {
            number1 = ctoi(result)
            symbol = "-"
            result = ""
        }
    }
}
elif x == "button-16"                           # 如果按钮键值为 16，对应加法运算符+
{
    if have_number == 0
    {
        if result == ""    {continue}
        if result[0] == "."
        {
            box = GUI("box")
            box.MessageBox("小数点前面的 0 不可省略！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            result = ""
            gui.SendText([""])
            continue
        }
        number1 = ctoi(result)
        symbol = "+"
        result = ""
        have_number = 1
    }
    else
    {
        if result == ""
        {
            symbol = "+"
        }
        else
        {
            number1 = ctoi(result)
            symbol = "+"
            result = ""
        }
    }
}
elif x == "button-17"                           # 如果按钮键值为 17，对应等号运算符=
{
    if symbol == "" {continue}                  # 如果 symbol 运算符为空，按等号按钮不作任何反应，跳转到循环起点
    if result[0] == "."
    {
        box = GUI("box")
        box.MessageBox("小数点前面的 0 不可省略！","确定")
        while box.MessageBoxClosed() != -1 {PowerDown(3)}
        continue
    }
    number2 = ctoi(result)                      # 将字符串转化为第二个运算数
    if symbol == "÷"
    {
        if number2 == 0                         # 如果是除法，除数不能为 0，否则弹出提示，并清除 result 和屏幕
        {
            box = GUI("box")
            box.MessageBox("除数不能为 0！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            continue
        }
        number3 = number1 / number2             # 计算运算结果
    }
    elif symbol == "×"
    {
        number3 = number1 * number2
    }
    elif symbol == "+"
    {
        number3 = number1 + number2
    }
    elif symbol == "-"
```

```
        {
            number3 = number1 - number2
        }
        number1 = number3                           # 计算结果作为下一次的被运算数
        temp = sprint("%.10f",number3)              # 将运算结果转化为字符串
        gui.SendText([temp])                        # 显示在 text 控件上
        result = ""                                 # 复位 result 和 symbol
        symbol = ""
        have_number = 1                             # 计算结果作为被运算数
    }
}
```

第十六章、Image 图片操作　例程《image_method》

Fine 图片资源编程涉及两部分内容：

一、窗口组件——用于在 Fine 窗口设计布局图片资源，支持目前流行的图片格式。

image0　= ["test.bmp","image",5,1,50,30]　　# 图片资源组件说明

列表第一项是图片的路径+名称（没有路径默认在 finevirt.exe 同目录下寻找），必须包含文件扩展名（只限定.bmp 文件格式），如果找不到图片报错。

列表第二项是"image"表明该组件是显示一幅图片。

列表第 3~6 项是指定图片显示的位置（单位：字节），前两项是照片的起始位置，后两项分别是图片的宽度和高度。

功能：image 组件的功能是将指定的图片显示在指定的位置，如果图片大小与指定区域的大小不相等，则自动伸缩图片使其适配指定区域，

由于自动伸缩可能会产生畸变，使用时，尽可能事先将图片进行编辑，使图片的宽度和高度与指定区域相匹配，获得满意效果。

二、图片传递——程序控制向窗口布局的图片区域发送新的资源，替换原来的图片资源。

gui.SendImage(num,["test1.bmp","test2.bmp","test3.bmp"])

参数 1 是窗口资源 ID，参数 2 是一个列表，列表项数等于窗口布局的图片资源个数，列表的每一项对应一个图片资源的文件名（可以是 BMP、JPG、PNG、ICO 等）。

```
# 例程 1：在 Fine 窗口中，显示两幅图片
gui = GUI("fine")
#gui.HideConsoleWindow()
time = TIME()

title     = "image 组件测试程序"
winSize = [20,2,120,35]

image0    = ["exampleRes\\png1.png","image",6,3,50,26]
image1    = ["exampleRes\\fine10.jpg","image",61,3,50,26]

list = [title,winSize,image0,image1]

gui.Fine(list)
while gui.FineClosed() != -1
{
    PowerDown(3)
}



# 例程 2：影集
gui = GUI("fine")
gui.HideConsoleWindow()
time = TIME()

photos = ["exampleRes\\fine1.jpg","exampleRes\\fine2.jpg","exampleRes\\fine3.jpg","exampleRes\\fine4.jpg",\
          "exampleRes\\fine5.jpg","exampleRes\\fine6.jpg","exampleRes\\fine8.jpg","exampleRes\\fine20.jpg",\
          "exampleRes\\fine9.jpg","exampleRes\\fine10.jpg","exampleRes\\fine11.jpg","exampleRes\\fine12.jpg",\
          "exampleRes\\fine13.jpg","exampleRes\\fine14.jpg","exampleRes\\fine18.jpg","exampleRes\\fine19.jpg",\
          "exampleRes\\fine21.jpg"]


title     = "image 组件测试程序"
winSize = [20,2,90,40]

image0    = ["exampleRes\\fine21.jpg","image",0,0,90,40]     # 在 Fine 窗口放置一副图片

list = [title,winSize,image0]                                 # 打包设计元素
gui.Fine(list)                                                  # 创建 fine 窗口，并返回资源 ID
i = 0
while gui.FineClosed() != -1
{
    PowerDown(3)
    time.sleep(1500)                        # 间隔 1.5 秒换一副照片
    picture = photos[i]                     # 从列表中取出一副图片的文件名
    gui.SendImage([picture])                # 将取出的文件名发送到 Fine 窗口
    i = i + 1
    if i >= 30                              # 循环播放
    {
```

```
        i = 0
    }
}


# 例程 3：表单录入，在窗口添加两张图片
gui = GUI("fine")
gui.HideConsoleWindow()

szlistbox1 = "支出,收入"
szlistbox2 = "生活用品,交通,通讯,衣服鞋帽,数码产品,健身,旅游,工资,奖金"

# 记录输入界面设计
title = "账单录入"
size   = [10,3,150,40]

text1      = ["|楷体|22|255|0|0","text",24,2,17,1]                                    # text 组件的标题为空，仍然可以定义字库
button     = ["  录入  |微软雅黑|24|255|255|0","button",55,2,10,1]                        # 给 button 组件设置字库

edit0      = [" ID |微软雅黑|22|0|0|255","edit","i",12,5,12,1]                  # 给 edit 组件设置字库
combobox1= ["  收入支出  |微软雅黑|22|0|255|0","combobox",szlistbox1,40,5,8,1]      # 给 combobox 组件设置字库

edit1      = ["  录入者  ","edit","s",63,5,18,1]
edit2      = ["  日期  ","edit","s",12,8,14,1]
combobox2= ["商品分类       ","combobox",szlistbox2,12,10,28,1]
edit3      = ["  商品名称  ","edit","s",12,12,60,1]
edit4      = ["  数量  ","edit","f",12,14,20,1]
edit5      = ["  单价  ","edit","f",12,16,20,1]
text2      = ["  合计  ","text",12,18,20,1]
textbox    = ["  录入数据  |微软雅黑|22|0|0|255","textbox","N",12,20,74,9]              # 给 textbox 组件设置字库
image0     = ["exampleRes\\png1.png","image",84,2,28,16]
image1     = ["exampleRes\\fine5.jpg","image",116,2,28,16]

# 打包设计元素
list = [title,size,text1,button,edit0,combobox1,edit1,edit2,combobox2,edit3,edit4,edit5,text2,textbox,image0,image1]

t1 = TIME()

gui.Fine(list)                          # 创建账单录入窗口，并返回资源 ID
t1.sleep(200)
gui.SendText(["    记录输入",""])
while gui.FineClosed() != -1            # 检查窗口关闭消息
{
    PowerDown(3)                                # 节能设置
    if gui.FineReady() == 0             # 检查是否有数据录入
    {
        x = gui.FineRead()          # 读取录入数据

        # x[0]是 button-编号(指令）
        # 优先输出 edit：x[1]ID，x[2]数据录入人，x[3]日期,x[4]商品名称，x[5]数量，x[6]单价
        # 其次输出 listbox：x[7]收入支出，x[8]商品分类

        tm = x[3]
        if tm == "" or len(tm) < 10 or tm[4] != "-"
        {
            box = GUI("box")        # 创建 MessageBox 窗口对象
            box.MessageBox("时间不能为空！并且时间格式必须是：NNNN-YY-RR!","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            continue
        }

        y = x[5]*x[6]                                       # 计算数量乘以单价，合计
        Y = sprint("%.2f",y)                                # 将合计转化为字符串
        gui.SendText(["    票据录入    ",Y])                  # 显示合计


        time1 = x[3]+" 08:00:00"                            # 拼接开票时间（录入的票据时间）

        str = itoc(x[1]) +","+x[7]+","+x[8]+","+x[4]+"," +itoc(x[5])+","+itoc(x[6])+","+time1+"\n"
        gui.SendTextbox([str])

        box = GUI("box")
        box.MessageBox("成功录入了一条记录！","确定")
        while box.MessageBoxClosed() != -1 {PowerDown(3)}
        gui.SendEdit(["",x[2],x[3],"","",""])          # 将查询选中的所有 edit 项记录显示出来，便于修改
        gui.SendCombobox([x[7],x[8]])                  # 将查询选中的所有 Combobox 项记录显示出来，便于修改
        gui.SendText(["    记录输入",""])
    }
}
```

第十七章　　Media 视频播放　　例程《media_method》

音视频播放涉及一个组件和一个播放方法：

一、视频组件——用于在创建窗口的过程中，布局一个视频按钮，点击触发视频播放

media = ["C:\\fine\\exampleRes\\test.mp4","media",10,2,50,40]

参数 1：字符串，是视频文件名（绝对路径+文件名+扩展名）

参数 2：表明是视频文件

参数 3~6：是指定在窗口的播放位置

参数 3：位置的左上角 x 轴起始点（单位:字节宽度）

参数 4：位置的左上角 y 轴起始点（单位:字节高度）

参数 5：播放区域的宽度（单位字节宽度）

参数 6：播放区域的高度（单位字节高度）

注意事项：在使用音视频组件时，需要提供两个同名文件，即，格式为 mp4、avi、mp3 等的音视频文件，和同名的 bmp 图片文件。

例如：希望在 Fine 窗口布局一个视频（C:\\fine\\exampleRes\\test.mp4）组件，除了提供该视频文件外，还需要提供一个 C:\\fine\\exampleRes\\test.bmp 文件。在创建 Fine 窗口时，系统会寻找与视频源同名的 bmp 图片，用于生成图片按钮，点击该按钮将启动播放器。

二、播放方法——用于动态向窗口传递音视频文件，并立启动播放器，播放该文件。

gui.PlayMedia(num,filename)

filename 必须是绝对路径（例如 C:\\fine\\exampleRes\\test.mp4），相对路径可能导致播放器出错。同时不要忘记提供 C:\\fine\\exampleRes\\test.bmp 文件。

三、**播放实现**：Fine 内置了一个 VLCPortable.exe 播放器。

可以支持大多数常用的音视频格式，由于是直接使用了 VLCPortable 播放器，所以，程序对音视频的控制受到了局限。

```
#   例程 1：在窗口嵌入四个 avi 视频，点击图片创建窗口播放视频
#   一个 Fine 窗口最多可以嵌入 4 个视频
#   每一个视频必须提供两个同名文件：视频文件（青花.avi）和位图文件（青花.bmp）

gui = GUI("fine")
gui.HideConsoleWindow()

title       = "视频播放"
size        = [10,3,110,30]

media1 = ["C:\\fine\\exampleRes\\青花.avi","media",6,4,20,15]       # 必须使用绝对路径
media2 = ["C:\\fine\\exampleRes\\赛车.avi","media",31,4,20,15]      # 必须使用绝对路径
media3 = ["C:\\fine\\exampleRes\\职场.avi","media",56,4,20,15]      # 必须使用绝对路径
media4 = ["C:\\fine\\exampleRes\\赛车.avi","media",81,4,20,15]      # 必须使用绝对路径

# 打包设计元素
list = [title,size,media1,media2,media3,media4]

gui.Fine(list)                                  # 创建 Fine 窗口，并返回资源 ID
while gui.FineClosed() != -1                    # 检查窗口关闭消息
{
    PowerDown(3)                                        # 节能设置
}


#   例程 2：程序控制————向窗口动态发送音视频文件，并播放该文件
#   每一个视频必须提供两个同名文件：视频文件（青花.avi）和位图文件（青花.bmp）
gui = GUI("fine")
gui.HideConsoleWindow()
```

```
time = TIME()

# 视频列表，视频的文件名必须使用绝对路径
medias = ["C:\\fine\\exampleRes\\职场.avi","C:\\fine\\exampleRes\\赛车.avi","C:\\fine\\exampleRes\\青花.avi"]

title   = "视频播放"
size    = [10,3,100,40]

media = ["C:\\fine\\exampleRes\\职场.avi","media",20,5,30,20]

# 打包设计元素
list = [title,size,media]
i = 0
gui.Fine(list)                                  # 创建 Fine 窗口，并返回资源 ID
while gui.FineClosed() != -1                    # 检查窗口关闭消息
{
    PowerDown(3)                                    # 节能设置
    if i == 0
    {
        gui.PlayMedia(medias[i])                # 播放文件 0
    }
    elif i == 1
    {
        gui.PlayMedia(medias[i])                # 播放文件 1
    }
    elif i == 2
    {
        gui.PlayMedia(medias[i])                # 播放文件 2
    }
    i = i + 1
    if i >= 3
    {
        i = 0
    }
    time.sleep(30000)
}



# 例程 3：表单录入，在窗口添加两个视频

gui = GUI("fine")
gui.HideConsoleWindow()

szlistbox1 = "支出,收入"
szlistbox2 = "生活用品,交通,通讯,衣服鞋帽,数码产品,健身,旅游,工资,奖金"

# 记录输入界面设计
title = "账单录入"
size  = [10,3,158,40]

text1     = ["|楷体|22|255|0|0","text",24,2,17,1]                    # text 组件的标题为空，仍然可以定义字库
button    = ["  录入  |微软雅黑|24|255|255|0","button",55,2,10,1]     # 给 button 组件设置字库

edit0     = [" ID |微软雅黑|22|0|0|255","edit","i",12,5,12,1]         # 给 edit 组件设置字库
combobox1= ["  收入支出  |微软雅黑|22|0|255|0","combobox",szlistbox1,40,5,8,1]   # 给 combobox 组件设置字库

edit1     = ["  录入者  ","edit","s",66,5,18,1]
edit2     = ["  日期  ","edit","s",12,8,14,1]
combobox2= ["商品分类       ","combobox",szlistbox2,12,10,28,1]
edit3     = ["  商品名称  ","edit","s",12,12,60,1]
edit4     = ["  数量  ","edit","f",12,14,20,1]
edit5     = ["  单价  ","edit","f",12,16,20,1]
text2     = ["  合计  ","text",12,18,20,1]
textbox   = ["  录入数据  |微软雅黑|22|0|0|255","textbox","N",12,20,74,8]   # 给 textbox 组件设置字库
media0    = ["C:\\fine\\exampleRes\\青花.avi","media",90,2,28,16]          # 音视频文件需要绝对路径
media1    = ["C:\\fine\\exampleRes\\赛车.avi","media",122,2,28,16]         # 音视频文件需要绝对路径

# 打包设计元素
list = [title,size,text1,button,edit0,combobox1,edit1,edit2,combobox2,edit3,edit4,edit5,text2,textbox,media0,media1]

t1 = TIME()

gui.Fine(list)                          # 创建账单录入窗口，并返回资源 ID
t1.sleep(200)
```

```
gui.SendText(["    记录输入",""])
while gui.FineClosed() != -1                # 检查窗口关闭消息
{
    PowerDown(3)                                    # 节能设置
    if gui.FineReady() == 0                 # 检查是否有数据录入
    {
        x = gui.FineRead()          # 读取录入数据

        # x[0]是 button-编号(指令）
        # 优先输出 edit：x[1]ID，x[2]数据录入人，x[3]日期,x[4]商品名称，x[5]数量，x[6]单价
        # 其次输出 listbox：x[7]收入支出，x[8]商品分类

        tm = x[3]
        if tm == "" or len(tm) < 10 or tm[4] != "-"
        {
            box = GUI("box")                # 创建 MessageBox 窗口对象
            box.MessageBox("时间不能为空！并且时间格式必须是：NNNN-YY-RR!","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            continue
        }

        y = x[5]*x[6]                                           # 计算数量乘以单价，合计
        Y = sprint("%.2f",y)                                    # 将合计转化为字符串
        gui.SendText(["    票据录入    ",Y])                    # 显示合计


        time1 = x[3]+" 08:00:00"                                # 拼接开票时间（录入的票据时间）

        str = itoc(x[1]) +","+x[7]+","+x[8]+","+x[4]+"," +itoc(x[5])+","+itoc(x[6])+","+time1+"\n"
        gui.SendTextbox([str])

        box = GUI("box")                # 创建 MessageBox 窗口对象
        box.MessageBox("成功录入了一条记录！","确定")
        while box.MessageBoxClosed() != -1 {PowerDown(3)}
        gui.SendEdit(["",x[2],x[3],"","",""])           # 将查询选中的所有 edit 项记录显示出来，便于修改
        gui.SendCombobox([x[7],x[8]])                           # 将查询选中的所有 Combobox 项记录显示出来，便于修改
        gui.SendText(["    记录输入",""])
    }
}
```

第十八章、绘图方法和指令集　例程《draw_method》

绘图方法：
gui.Drawing(drawlist)　　# drawlist 是一个列表，代表一个绘图指令
gui.DrawPackage(Multi_drawlist)　　# Multi_drawlist 是一个双重列表，每一个列表项是一个绘图指令

使用注意事项：
一、当只有一条绘图指令时，可以使用 gui.Drawing(drawlist) 执行绘图。
二、当一次性需要执行多条绘图指令时，可以将多条绘图指令打包，然后，使用 gui.DrawPackage(Multi_drawlist)，一次性执行多个绘图指令。例如，在游戏制作时，需要绘制背景，以及游戏部件图形，需要连续多次绘图，这种情况下，建议使用绘图指令包，执行速度快，占用资源少。

使用 gui.DrawPackage(Multi_drawlist)绘制多个图形指令时，可能需要花费比较多的时间，因此，可以使用 gui.DrawPackageDone()方法查询上一个绘图指令包是否执行完毕（执行完毕返回 True，否则返回 False）,如果尚未执行完毕，需要等待，直到返回 True，方可执行下一次绘图。
三、绘制图片的第一个指令必须是绘制画布指令，绘制完画布后，方可作画。

绘制画布的指令有三个，分别是：CanvasBrush、CanvasImage 和 CanvasPutImage。

当需要绘图时，需使用上述三个指令之一，设置画布，然后，再使用其它绘图指令，在画布上添加图形或图片，如果没有绘制画布，直接绘制其它指令，会终止程序，并报错。

绘图指令集：
1、 drawlist = ["CanvasBrush",20,20,400,300,255,0,0]

参数 1 是设置自制画布的指令，参数 2~3 是画布起点坐标(以窗口左上角位基准点)，参数 4~5 是画布终点坐标，参数 6~8 是 RGB 颜色。
2、 drawlist = ["CanvasImage",20,20,400,300,"D:\\fine\\exampleRes\\fine5.jpg"]

参数 1 是设置自制画布的指令，参数 2~3 是画布起点坐标(以窗口左上角位基准点)，参数 4~5 是画布终点坐标，参数 6 是作为画布的图片文件名。

注意事项 1：如果先前已设置画布，该命令会先清除屏幕内容，然后再设置画布。

注意事项 2：设置画布的命令中，参数 2~3 是以窗口的左上角位基准点，但是，以下的画图命令中的坐标位置是以画布的左上角为基准点。
3、 drawlist = ["Line",0,0,200,200,0,0,0,0,255]

参数 1 画线指令，参数 2~3 是线段的起点坐标，参数 4~5 是线段的终点坐标，参数 6 是线的类型，参数 7 是线宽，参数 8~10 是 RGB 颜色。参数 6：无间断实线 0，虚线 1，点画线 2，虚线+点 3，虚线+两点 4。
4、 drawlist = ["Text","欢迎你世界！",30,80,"微软雅黑",38,0,255,0]

参数 1 是在画布上书写文字指令，参数 2 是文字内容（单词传送不超过 512 字节），参数 3~4 是文字起始位置，参数 5 是字体，参数 6 是字号，参数 7~9 是 RGB 颜色.
5、 drawlist = ["Point",200,200,0,0,255]　　# 在(200,200)位置画一个蓝色的点

参数 1 是在画布上画点的指令，参数 2~3 是点的位置，参数 4~6 是 RGB 颜色.
6、 drawlist = ["Rectangle",200,200,300,200,0,4,0,255,0]

参数 1 画矩形，不填充颜色指令，参数 2~3 是矩形中心坐标，参数 4 是宽，参数 5 是高，参数 6 是线型，参数 7 是线宽，参数 8~10 是颜色。
7、 drawlist = ["RectangleFill",200,200,100,300,0,4,0,255,0,0,0,255]

参数 1 画矩形，并填充颜色指令，参数 2~3 是矩形中心坐标，参数 4 是宽，参数 5 是高，参数 6 是线型，参数 7 是线宽，参数 8~10 是颜色，参数 11~13 是矩形内填充色。
8、 drawlist = ["RoundRect",200,200,300,200,0,4,50,30,0,255,0]

参数 1 画圆角矩形、无填充指令，参数 2~3 是中心坐标，参数 4 是宽，参数 5 是高，

参数 6 是线型，参数 7 是线宽，参数 8 是椭圆角宽度，参数 9 是椭圆角高度，参数 10~12 是矩形颜色。

9、 drawlist = ["RoundRectFill",200,200,100,300,0,4,50,30,0,255,0,0,0,255]

参数 1 画圆角矩形、有填充指令，参数 2~3 是中心坐标，参数 4 是宽，参数 5 是高，参数 6 是线型，参数 7 是线宽，参数 8 是椭圆角宽度，参数 9 是椭圆角高度，参数 10~12 是矩形颜色，参数 13~15 是矩形内填充色。

10、drawlist = ["Polyline",10,20,10,80,40,80,100,30,0,4,0,255,0]

参数 1 是开放多边形指令，参数 2,3、4,5……是各点的坐标，第 2n+2 个参数是线形，依次是线宽、颜色

11、drawlist = ["Polygon",10,20,10,80,40,80,100,30,0,4,0,255,0]

参数 1 是闭合多边形、不填充颜色指令，参数 2,3、4,5……是各点的坐标，第 2n+2 个参数是线型，依次是线宽、颜色

12、drawlist = ["PolygonFill",10,20,10,80,40,80,100,30,0,4,0,255,0,0,0,255]

参数 1 是闭合多边形、并填充颜色指令，参数 2,3、4,5……是各点的坐标，第 2n+2 个参数是线形，依次是线宽、颜色、填充色

13、drawlist = ["Circle",100,100,50,0,4,0,255,0]

参数 1 是画圆、不填充指令，参数 2~3 是圆心，参数 4 是半径，参数 5 是线型，参数 6 是线宽，参数 7~9 是颜色

14、drawlist = ["CircleFill",100,100,50,0,4,0,255,0,0,0,255]

参数 1 是画圆、不填充指令，参数 2~3 是圆心，参数 4 是半径，参数 5 是线性，参数 6 是线宽，参数 7~9 是颜色，参数 10~12 是填充色

15、drawlist = ["Ellipse",100,200,150,100,3,4,0,255,0]

参数 1 是画椭圆、不填充指令，参数 2~3 是中心坐标，参数 4 是 x 轴宽度，参数 5 是 y 轴高度， 参数 6 是线型，参数 7 是线宽，参数 8~10 是颜色

16、drawlist = ["EllipseFill",100,200,150,100,3,4,0,255,0,0,0,255]

参数 1 是画椭圆、不填充指令，参数 2~3 是内接椭圆的矩形的左上角坐标，参数 4 是矩形宽度，参数 5 是矩形高度，参数 6 是线型，参数 7 是线宽，参数 8~10 是颜色，参数 11~13 是填充色

17、drawlist = ["Chord",100,100,150,100,30,120,3,4,0,255,0]

参数 1 是画闭合弦、不填充指令，参数 2~3 是内接椭圆矩形的中心坐标，参数 4 是矩形宽度，参数 5 是矩形高度，参数 6 是弧线起点（角度°），参数 7 是弧线终点（角度°），参数 8 是线型，参数 9 是线宽，参数 10~12 是颜色

18、drawlist = ["ChordFill",100,200,150,100,30,120,3,4,0,255,0,0,0,255]

参数 1 是画闭合弦、不填充指令，参数 2~3 是内接椭圆的矩形的中心坐标，参数 4 是矩形宽度，参数 5 是矩形高度，参数 6 是弧线起点（角度°），参数 7 是弧线终点（角度°），参数 8 是线型，参数 9 是线宽，参数 10~12 是颜色，参数 13~15 是填充色

19、drawlist = ["Pie",200,200,180,100,30,120,3,4,0,255,0]

参数 1 是画闭合弦、不填充指令，参数 2~3 是内接椭圆的矩形的中心坐标，参数 4 是矩形宽度，参数 5 是矩形高度，参数 6 是弧线起点（角度°），参数 7 是弧线终点（角度°），参数 8 是线型，参数 9 是线宽，参数 10~12 是颜色

20、drawlist = ["PieFill",400,200,100,180,30,120,3,4,0,255,0,0,0,255]

参数 1 是画闭合弦、有填充指令，参数 2~3 是内接椭圆的矩形的中心坐标，参数 4 是矩形宽度，参数 5 是矩形高度，参数 6 是弧线起点（角度°），参数 7 是弧线终点（角度°），参数 8 是线型，参数 9 是线宽，参数 10~12 是颜色，参数 13~15 是填充色

21、drawlist = ["Arc",300,300,150,100,30,120,3,4,0,255,0]

参数 1 是画弦指令，参数 2~3 是内接椭圆的矩形的中心坐标，参数 4 是矩形宽度，参数 5 是矩形高度，参数 6 是弧线起点（角度°），参数 7 是弧线终点（角度°），参数 8 是线型，参数 9 是线宽，参数 10~12 是颜色

22、drawlist = ["AddImage",100,100,200,200,"D:\\fine\\exampleRes\\mydraw.bmp"]

参数 1 是添加图片指令，参数 2~3 是图片左上角坐标，参数 4 是图片宽度，参数 5 是图片高度，参数 6 是添加的图片文件名（含扩展名）。

该指令每调用一次，都会读取硬盘，如果需要多次使用一个图片资源，应使用LoadImage+PutImage 指令组合，LoadImage 加载图片（只读取一次硬盘），可以使用 PutImage 多次使用该资源，确认不再需要该资源时，可以调用 DeleteImage 删除该资源。

23、drawlist = ["Save","PaintingArea","D:\\fine\\exampleRes\\mydarw.bmp"]

drawlist = ["Save","Screen","D:\\fine\\exampleRes\\screenshot.bmp"]

参数 1 是保存屏幕截图指令，参数 2 是"PaintingArea"时，将绘画区保存为图片；

参数 2 是"Screen"时，将整个屏幕内容保存为图片，参数 3 是文件路径名称（仅支持保存为 BMP 文件）。

24、drawlist = ["LoadImage","D:\\fine\\exampleRes\\fine.png","昵称"]

参数 1 是加载图片资源指令，参数 2 是加载的图片文件名（含扩展名），参数 3 是加载文件的昵称(方便后续调用)，如果昵称出现重复会提示。该指令将图片资源读入内存缓存，一个 Fine 窗口限制加载不超过 16 个图片资源。

25、drawlist = ["DeleteImage","昵称"]

参数 1 是删除图片资源指令，参数 2 是要删除的图片资源文件的昵称。

当不再使用某个图片资源时，可以调用该指令删除图片资源。

26、drawlist = ["PutImage","昵称",x,y]

(x,y)起点位置，宽度和高度使用图片宽度和高度

drawlist = ["PutImage","昵称",x,y,width,height]

(x,y)起点位置，width,height 是显示图片的宽度和高度（自动伸缩）。

参数 1 是放置图片资源指令，参数 2 是图片资源的昵称，参数 3~4 是图片起始位置，如果只有 4 个参数，表明显示图片时，宽度和高度使用图片本身的宽度和高度；如果有 6 个参数时，参数 5~6 表明是自适应显示的宽度和高度。

27、drawlist = ["RotatePutImage","昵称",angle,x,y]

angle 是旋转角度，(x,y)起点位置，宽度和高度使用图片宽度和高度

drawlist = ["RotatePutImage","昵称",angle,x,y,width,height]

angle 是旋转角度（单位度数），(x,y)起点位置，width,height 是显示图片的宽度和高度（自动伸缩）。

参数 1 是先旋转图片 angle 角度（单位度数），再放置图片资源指令，参数 2 是图片资源的昵称，参数 3 是旋转角度，参数 4~5 是图片起始位置， 如果只有 5 个参数，表明显示图片时，宽度和高度，使用图片本身的宽度和高度；

如果有 7 个参数时，参数 6~7 表明是自适应显示的宽度和高度。

28、drawlist = ["CanvasPutImage","昵称",x,y]　　将已经加载到内存中的图片资源，作为画布绘制到绘图区

(x,y)起点位置，宽度和高度使用图片宽度和高度(4 参数）

drawlist = ["CanvasPutImage","昵称",x,y,width,height]

(x,y)起点位置，宽度和高度使用图片宽度和高度，width,height 是指定的显示宽度和高度(5~6 参数）

注意：LoadImage 指令将资源读入内存，可以多次调用 PutImage 或 RotatePutImage 使用加载的资源(不再从硬盘读取文件)。

Add Image 指令是将文件直接读取硬盘(每次调用都会读取硬盘)，并将图片放到指定位置。

```
# 例程 1：绘图
gui = GUI("fine")
gui.HideConsoleWindow()
time = TIME()

title       = "绘画"                                    # 窗口标题
size        = [10,3,86,38]                              # 窗口尺寸
list = [title,size]
```

```
gui.Fine(list)                                                  # 创建 Fine 窗口，并返回资源 ID
while gui.FineClosed() != -1                                    # 检查窗口关闭消息
{
    PowerDown(3)                                                # 节能设置
      #图片画布
      drawlist = ["CanvasImage",20,20,640,510,"C:\\fine\\exampleRes\\fine5.jpg"]      # 使用图片作为画布
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Text","欢迎你世界！",0,0,"微软雅黑",38,0,255,0]                    # 在图片上写字
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Point",40,80,0,0,255]                        # 在(40,80)位置画一个蓝色的点
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Line",0,0,200,200,0,2,0,0,255]               # 画线 drawlist = ["Rectangle",100,100,300,300,0,4,0,255,0]
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Rectangle",200,70,60,80,0,4,0,255,0]     # 画矩形，不带填充色
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["RectangleFill",300,130,100,200,0,4,0,255,0,0,255,0]          # 画矩形，带填充色
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["RoundRect",420,70,100,60,1,4,50,30,0,255,0]                  # 画圆角矩形，不带填充色
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["RoundRectFill",540,120,60,130,3,4,50,30,0,255,0,0,0,255]     # 画圆角矩形，带填充色
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Polyline",50,200,60,300,90,250,70,180,0,4,0,255,0]           # 开放多边形
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Polygon",110,200,120,300,150,250,130,180,0,4,0,255,0]        # 封闭多边形，不填充
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["PolygonFill",170,200,180,300,210,250,190,180,0,4,0,255,0,0,0,255]      # 封闭多边形
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Circle",300,270,50,2,4,0,255,0]                                        # 画圆，不填充
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["CircleFill",420,200,50,0,4,0,255,0,0,0,255]                  # 画圆，填充颜色
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Ellipse",90,380,120,100,3,4,0,255,0]                         # 画椭圆，无填充颜色
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["EllipseFill",220,380,100,140,3,4,0,255,0,0,0,255]            # 画椭圆，有填充颜色
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Chord",340,400,100,70,30,120,3,4,0,255,0]                    # 画闭合弦，无填充颜色
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["ChordFill",430,400,100,70,30,120,3,4,0,255,0,0,0,255]        # 画闭合弦，有填充颜色
      gui.Drawing(drawlist)
      time.sleep(3000)

      drawlist = ["Pie",530,400,100,70,30,120,3,4,0,255,0]                      # 画扇形，无填充颜色
      gui.Drawing(drawlist)
      time.sleep(3000)
```

```
        drawlist = ["PieFill",510,320,70,120,0,120,3,4,0,255,0,0,0,255]            # 画扇形，有填充颜色
        gui.Drawing(drawlist)
        time.sleep(3000)

        drawlist = ["Arc",300,300,400,200,-20,150,3,4,0,255,0]                      # 画弧
        gui.Drawing(drawlist)
        time.sleep(3000)

        drawlist = ["AddImage",120,120,100,100,"C:\\fine\\exampleRes\\fine8.jpg"]   # 添加图片
        gui.Drawing(drawlist)
        time.sleep(3000)

        drawlist = ["Save","PaintingArea","D:\\mydraw.bmp"]          # 保存绘画区
        gui.Drawing(drawlist)
        time.sleep(3000)

        drawlist = ["Save","Screen","D:\\screen.bmp"]                # 保存全屏
        gui.Drawing(drawlist)
        time.sleep(3000)

        gui.CloseFine()
}



# 例程 2：使用 gui.DrawPackage(num,Multi_drawlist) 绘图
gui = GUI("fine")
gui.HideConsoleWindow()
time = TIME()

title       = "绘画"                                    # 窗口标题
size        = [10,3,86,38]                              # 窗口尺寸
list = [title,size]
gui.Fine(list)                                          # 创建 Fine 窗口，并返回资源 ID
while gui.FineClosed() != -1                            # 检查窗口关闭消息
{
        PowerDown(3)                                    # 节能设置
        pack = []                                       # 空列表
        drawlist = ["CanvasBrush",20,20,630,480,255,0,0]            # 自制画布
        pack.append(drawlist)

        drawlist = ["Text","欢迎你世界！",0,0,"微软雅黑",38,0,255,0]                  # 在图片上写字
        pack.append(drawlist)

        drawlist = ["Point",40,80,0,0,255]                  # 在(200,200)位置画一个蓝色的点
        pack.append(drawlist)

        drawlist = ["Line",0,0,200,200,0,2,0,0,255]         # 画线 drawlist = ["Rectangle",100,100,300,300,0,4,0,255,0]
        pack.append(drawlist)

        drawlist = ["Rectangle",200,70,60,80,0,4,0,255,0]   # 画矩形，不带填充色
        pack.append(drawlist)

        drawlist = ["RectangleFill",300,130,100,200,0,4,0,255,0,0,255,0]            # 画矩形，带填充色
        pack.append(drawlist)

        drawlist = ["RoundRect",420,70,100,60,1,4,50,30,0,255,0]                    # 画圆角矩形，不带填充色
        pack.append(drawlist)

        drawlist = ["RoundRectFill",540,120,60,130,3,4,50,30,0,255,0,0,0,255]       # 画圆角矩形，带填充色
        pack.append(drawlist)

        drawlist = ["Polyline",50,200,60,300,90,250,70,180,0,4,0,255,0]             # 开放多边形
        pack.append(drawlist)

        drawlist = ["Polygon",110,200,120,300,150,250,130,180,0,4,0,255,0]          # 封闭多边形，不填充
        pack.append(drawlist)

        drawlist = ["PolygonFill",170,200,180,300,210,250,190,180,0,4,0,255,0,0,0,255]      # 开放多边形
        pack.append(drawlist)

        drawlist = ["Circle",300,270,50,2,4,0,255,0]                                # 画圆，不填充
        pack.append(drawlist)

        drawlist = ["CircleFill",420,200,50,0,4,0,255,0,0,0,255]                    # 画圆，填充颜色
        pack.append(drawlist)
```

```
        drawlist = ["Ellipse",90,380,120,100,3,4,0,255,0]                    # 画椭圆，无填充颜色
        pack.append(drawlist)

        drawlist = ["EllipseFill",220,380,100,140,3,4,0,255,0,0,0,255]       # 画椭圆，有填充颜色
        pack.append(drawlist)

        drawlist = ["Chord",340,400,100,70,30,120,3,4,0,255,0]               # 画闭合弦，无填充颜色
        pack.append(drawlist)

        drawlist = ["ChordFill",430,400,100,70,30,120,3,4,0,255,0,0,0,255]   # 画闭合弦，有填充颜色
        pack.append(drawlist)

        drawlist = ["Pie",530,400,100,70,30,120,3,4,0,255,0]                 # 画扇形，无填充颜色
        pack.append(drawlist)

        drawlist = ["PieFill",510,320,70,120,0,120,3,4,0,255,0,0,0,255]      # 画扇形，有填充颜色
        pack.append(drawlist)

        drawlist = ["Arc",300,300,400,200,-20,150,3,4,0,255,0]               # 画弧
        pack.append(drawlist)

        drawlist = ["AddImage",120,120,100,100,"D:\\fine\\exampleRes\\fine8.jpg"]   # 添加图片
        pack.append(drawlist)

        gui.DrawPackage(pack)        # 将上述绘图指令打包一次绘制，速度快

        time.sleep(20000)
        gui.CloseFine()
}



# 例程 3：图片资源加载
# 一、使用 LoadImage 指令加载图片资源（一个 Fine 窗口最多限制不超过 16 个图片资源）
# 二、使用 PutImage 指令将图片放置到指定位置（可以原图，也可以自动伸缩）
# 三、使用 RotatePutImage 指令将图片旋转后放到指令位置（可以是原图，也可以自动伸缩）
# 四、当确定不需要某个图片资源时，可以使用 DeleteImage 指令删除对应图片资源。

gui = GUI("fine")
gui.HideConsoleWindow()
time = TIME()
os = OS()

title          = "绘画"                                    # 窗口标题
size             = [10,3,90,43]                          # 窗口尺寸
list = [title,size]
gui.Fine(list)                                           # 创建 Fine 窗口，并返回资源 ID

drawlist = ["LoadImage","C:\\fine\\exampleRes\\fine.png","image1"]      # 加载图片资源
gui.Drawing(drawlist)

drawlist = ["LoadImage","C:\\fine\\exampleRes\\fine8.jpg","myimage"]    # 加载图片资源
gui.Drawing(drawlist)

while gui.FineClosed() != -1                             # 检查窗口关闭消息
{
    PowerDown(3)                                         # 节能设置
    # 自制画布
    drawlist = ["CanvasBrush",0,0,680,620,255,0,0]       # 自制画布
    gui.Drawing(drawlist)
    time.sleep(300)

    drawlist = ["Text","欢迎你世界！ ",0,0,"微软雅黑",38,0,255,0]      # 在图片上写字
    gui.Drawing(drawlist)
    time.sleep(300)

    drawlist = ["Line",0,0,200,200,0,2,0,0,255]          # 画线 drawlist = ["Rectangle",100,100,300,300,0,4,0,255,0]
    gui.Drawing(drawlist)
    time.sleep(300)


    drawlist = ["PutImage","image1",100,160]             # 放置图片资源，图片的宽度和高度，使用自身的宽度和高度
    gui.Drawing(drawlist)
    time.sleep(1000)
```

```
    drawlist = ["RotatePutImage","image1",45,100,160]       # 放置图片资源，图片的宽度和高度，使用自身的宽度和高度
    gui.Drawing(drawlist)
    time.sleep(1000)

    drawlist = ["PutImage","image1",380,160]                # 放置图片资源，自适应调节宽度和高度
    gui.Drawing(drawlist)
    time.sleep(1000)

    drawlist = ["RotatePutImage","image1",45,380,160,100,400]       # 先旋转,再放置图片
    gui.Drawing(drawlist)
    time.sleep(1000)

    #drawlist = ["DeleteImage","image1"]                    # 把加载到内存中的资源删除
    #gui.Drawing(drawlist)
    #time.sleep(1000)

    drawlist = ["PutImage","myimage",200,250,200,200]       # 放置图片资源，自适应调节宽度和高度
    gui.Drawing(drawlist)
    time.sleep(2000)
}



# 例程 4：本例程是一个线程绘图程序，主线程向绘图子线程传递数据，子线程负责播放。

# 主线程
drawlist0  = ["CanvasBrush",20,20,640,510,255,0,0]                          # 自制画布
drawlist1  = ["CanvasImage",20,20,640,510,"exampleRes\\fine5.jpg"]           # 使用图片作为画布
drawlist2  = ["Text","欢迎你世界！",0,0,"微软雅黑",38,0,255,0]               # 在图片上绘制文字
drawlist3  = ["Point",40,80,0,0,255]                                        # 画一个蓝色的点
drawlist4  = ["Line",0,0,200,200,0,2,0,0,255]                               # 画线
drawlist5  = ["Rectangle",200,70,60,80,0,4,0,255,0]                         # 画矩形，不带填充色
drawlist6  = ["RectangleFill",300,130,100,200,0,4,0,255,0,0,255,0]          # 画矩形，带填充色
drawlist7  = ["RoundRect",420,70,100,60,1,4,50,30,0,255,0]                  # 画圆角矩形，不带填充色
drawlist8  = ["RoundRectFill",540,120,60,130,3,4,50,30,0,255,0,0,0,255]     # 画圆角矩形，带填充色
drawlist9  = ["Polyline",50,200,60,300,90,250,70,180,0,4,0,255,0]           # 开放多边形
drawlist10 = ["Polygon",110,200,120,300,150,250,130,180,0,4,0,255,0]        # 封闭多边形，不填充
drawlist11 = ["PolygonFill",170,200,180,300,210,250,190,180,0,4,0,255,0,0,0,255]  # 开放多边形
drawlist12 = ["Circle",300,270,50,2,4,0,255,0]                              # 画圆，不填充
drawlist13 = ["CircleFill",420,200,50,0,4,0,255,0,0,0,255]                  # 画圆，填充颜色
drawlist14 = ["Ellipse",90,380,120,100,3,4,0,255,0]                         # 画椭圆，无填充颜色
drawlist15 = ["EllipseFill",220,380,100,140,3,4,0,255,0,0,0,255]            # 画椭圆，有填充颜色
drawlist16 = ["Chord",340,400,100,70,30,120,3,4,0,255,0]                    # 画闭合弦，无填充颜色
drawlist17 = ["ChordFill",430,400,100,70,30,120,3,4,0,255,0,0,0,255]        # 画闭合弦，有填充颜色
drawlist18 = ["Pie",530,400,100,70,30,120,3,4,0,255,0]                      # 画扇形，无填充颜色
drawlist19 = ["PieFill",510,320,70,120,0,120,3,4,0,255,0,0,0,255]           # 画扇形，有填充颜色
drawlist20 = ["Arc",300,300,400,200,-20,150,3,4,0,255,0]                    # 画弧
drawlist21 = ["AddImage",120,120,100,100,"exampleRes\\fine8.jpg"]           # 添加图片

# 播放数据列表一（自制画布）
DataList0 = [drawlist0,drawlist2,drawlist3,drawlist4,drawlist5,drawlist6,drawlist7,drawlist8,\
                        drawlist9,drawlist10,drawlist11,drawlist12,drawlist13,drawlist14,drawlist15,drawlist16,\
                        drawlist17,drawlist18,drawlist19,drawlist20,drawlist21]

# 播放数据列表二（图片画布）
DataList1 = [drawlist1,drawlist2,drawlist3,drawlist4,drawlist5,drawlist6,drawlist7,drawlist8,\
                        drawlist9,drawlist10,drawlist11,drawlist12,drawlist13,drawlist14,drawlist15,drawlist16,\
                        drawlist17,drawlist18,drawlist19,drawlist20,drawlist21]


gui0 = GUI("fine")
gui0.HideConsoleWindow()
time = TIME()

global next                     # 全局变量，用于控制播放内容
global task                     # 全局变量，控制是否需要创建线程

def ThreadDrawing(drawList)     # 绘图线程携带一个参数，即，绘图指令列表
{
    gui = GUI("fine")
    title          = "绘画"                  # 窗口标题
    size           = [10,3,86,37]            # 窗口尺寸
    list = [title,size]
    gui.Fine(list)                           # 创建 Fine 窗口，并返回资源 ID
```

```
    while gui.FineClosed() != -1              # 检查窗口关闭消息
    {
        PowerDown(3)
        for i in    drawList                  # 遍历列表
        {
            gui.Drawing(i)                    # 绘制
            time.sleep(50)                    # 延迟
        }
        gui.CloseFine()                       # 关闭 Fine 窗口
    }
    task = True                               # 允许创建新的线程
    next = not next                           # 翻转 next 状态
    #time.sleep(500)
    # 线程结束
}

next = True          # True 表示绘制 DataList0 命令包，False 表示绘制 DataList1 命令包，
task = True          # True 表示需要创建线程，Fales 表示不需要创建线程

count = 10           # 计数器，线程创建的次数
while count
{
    PowerDown(3)
    if task                                   # True 表示需要创建线程
    {
        task = False                          # 清除 task，在子线程工作结束后，重置 True
        count = count -1                      # 计数器-1，当 count = 0 退出 while 循环，进而结束程序
        if next
        {
            thread = THREAD()                                       # 定义一个线程变量
            thread = thread.CreateThread(ThreadDrawing,DataList0)   # 默认新创建的线程处于悬挂状态，不会执行
            thread.start()                                          # 线程的 start()方法，将启动当前 thread 线程运行
            #thread.join()                                          # 通知主线程等待子线程
        }
        else
        {
            thread = THREAD()                                       # 定义一个线程变量
            thread = thread.CreateThread(ThreadDrawing,DataList1)   # 默认新创建的线程处于悬挂状态，不会执行
            thread.start()                                          # 线程的 start()方法，将启动当前 thread 线程运行
            #thread.join()                                          # 通知主线程等待子线程
        }
    }
}
```

第十九章、MySQL 数据库　例程《sql_method》

数据库对象生成函数：
sql = MYSQL()　　# 定义一个 MySQL 数据库对象 sql（未关联账号，查询结果为空）

sql 的方法（7 个类方法）：

sql.library_init()　# 初始化 mysql 库 无返回。

# 在后台以管理员身份事先创建了用户“zhaojd”，创建了数据库“dbzhaojd”。
sql = sql.connect("localhost","zhaojd","password","dbzhaojd",3306)　# 连接库,返回 sql 对象关联账号。

sql.set_character("utf8")　　# 设置数据库的字符编码，无返回。

sql = sql.query(SQL)　　# 执行 SQL 语句，返回执行结果对象（包含了查询结果）。

result = sql.use_result()　　# 读取查询结果，查询结果为双层列表。

sql.library_end()　　# 无参数，无返回，功能是释放 MySQL 资源

sql.close()　　# 无参数，无返回，释放 sql 对象

查询语句会产生查询结果 result，需要立即读取查询结果，查询结果为二维列表。
result 列表的每一项是一个子列表（即，把读出的每一行数据打包成子列表）。
不管 mysql 表格的字段是何种类型的数据，在读出的子列表中都呈现出“字符串”格式。
类型为数字（整数或浮点数）的字段，如果需要可以使用 ctoi()函数，将其转化为数字。

例程：大管家——账簿

大管家账簿使用 MySQL 数据库保存记录，具有数据录入、数据查询、数据修改、数据删除、统计报表，统计报表有月报表和周报表，报表内容：按月（按周）计算收入合计、支出合计、结余，并按金额降序显示当期账单。

程序共分为一个主程序模块，六个功能子模块。

主模块：daguanjia.fin，在主模块中加载其它 6 个子模块。

子模块 1：SJLR.fin，功能是显示录入界面，录入记录。

子模块 2：SJCX.fin，功能是显示查询界面，按条件查询，结果显示在 textbox 组件上。

子模块 3：SJXG.fin，功能是先显示查询界面，查出的符合条件的记录显示在 textbox 区域，可以双击选中某条记录，对其修改。

子模块 4：SJSC.fin，功能是先显示查询界面，查出的符合条件的记录显示在 textbox 区域，可以双击选中某条记录，对其删除。

子模块 5：YDBB.fin，功能是录入报表周期，按月度显示期间报表。

子模块 6：ZDBB.fin，功能是录入报表周期，按周显示期间报表（周日为一周的首日）。

```
# 主程序：daguanjia.fin

# 大管家记账——主程序
# 分为七个模块：主程序、数据录入、数据查询、数据修改、数据删除、月度报表、周度报表

from ".\\fin\\examples\\MySQL\\SJLR" import shujvluru                # 从模块 SJLR 中，加载 shujvluru 函数，    功能：数据录入
from ".\\fin\\examples\\MySQL\\SJCX" import shujvchaxun               # 从模块 SJCX 中，加载 shujvchaxun 函数，  功能：数据查询
from ".\\fin\\examples\\MySQL\\SJXG" import shujvxiugai               # 从模块 SJXG 中，加载 shujvxiugai 函数，  功能：数据修改
from ".\\fin\\examples\\MySQL\\SJSC" import shujvshanchu              # 从模块 SJSC 中，加载 shujvshanchu 函数，功能：数据删除
from ".\\fin\\examples\\MySQL\\YDBB" import yuebaobiao                # 从模块 YDBB 中，加载 yuebaobiao 函数，   功能：月度报表
from ".\\fin\\examples\\MySQL\\ZDBB" import zhoubaobiao               # 从模块 ZDBB 中，加载 zhoubaobiao 函数，  功能：周度报表

# 事先，在后台以管理员身份事先创建了用户“zhaojd”，创建了数据库“dbzhaojd”。
# 后台启动数据库（net start MySQL） use dbzhaojd 数据库，创建表：guanjiapo

sql = MYSQL()                                                        # 定义一个 sql 数据库变量，用于调用数据库方法
sql.library_init()                                                   # 初始化库
sql = sql.connect("localhost","zhaojd","password","dbzhaojd",3306)   #连接库,返回 sql 对象。
sql.set_character("utf8")                                            # 设置数据库的字符编码为"utf8"

/*
```

```
    # daguanjia 数据库的表定义：共有 9 个字段
    id INT PRIMARY KEY AUTO_INCREMENT,   # 票据记录的 ID 号，整数、主键、唯一不重复、自增
    in_out VARCHAR(4),                   # 收入、支出（4 字节）
    kind VARCHAR(24),                    # 票据分类（24 字节）
    name VARCHAR(60),                    # 商品名称（60 字节）
    quantity FLOAT(10,2),                # 数量浮点数（保留两位小数，整数部分最多 8 位）
    price FLOAT(10,2),                   # 单价浮点数（保留两位小数，整数部分最多 8 位）
    lururen VARCHAR(12),                 # 票据录入人姓名
    shijian DATETIME,                    # 发票时间（格式：2024-10-29 08:00:00），数据录入时只录入日期，自动补上时间(08:00:00)
    lurutime DATETIME                    # 自动获取票据录入时的系统时间（格式：2024-10-29 14:45:00）
*/

gui = GUI("fine")                        # 定义一个 gui 对象，用来调用 gui 方法
#gui.HideConsoleWindow()                 # 关闭控制台窗口
time = TIME()                            # 定义一个事件对象
os = OS()                                # 定义一个操作系统对象

global line_data                         # 定义一个全局变量，用于模块之间传递数据

# 主菜单设计
title = "管家婆"
size    = [30,10,50,22]
text    = ["","text",14,1,20,1]
button1 = ["票据录入","button",16,4,16,1]
button2 = ["数据查询","button",16,6,16,1]
button3 = ["查询修改","button",16,8,16,1]
button4 = ["查询删除","button",16,10,16,1]
button5 = ["统计报表","button",16,12,16,1]
button6 = ["退出程序","button",16,14,16,1]

list = [title,size,text,button1,button2,button3,button4,button5,button6]

gui.Fine(list)                           # 创建菜单窗口
gui.SendText(["          主菜单"])       # 显示菜单名称
while gui.FineClosed() != -1             # 循环检查窗口关闭消息
{
    PowerDown(3)                         # 低功耗设置
    if gui.FineReady() == -1 {continue}  # 如果菜单项没有被选中，继续循环等待
    x = gui.FineRead()                   # 获取菜单编号。

    dgjcd = x[0]                 # 将读取的第一项指令 x[0]，赋值 dgjcd 变量(大管家菜单)

    # 这里不要直接使用  if x[0] == "button-0" 进行判断，而是将 x[0]赋值变量后，对变量进行判断
    # 因为根据变量 dgjcd 的值，程序将转向其他模块，列表内的数据项，再虚拟机垃圾回收中，可能被回收掉，
    # 但是，不会回收掉一个有用的变量。
    if dgjcd == "button-0"           # 记录输入功能
    {
        shujvluru()      # 进入数据录入界面
    }
    elif dgjcd == "button-1"      # 记录查询功能
    {
        shujvchaxun("  记录查询  ","N")      # 进入数据查询界面，查询操作，不返回数据
    }
    elif dgjcd == "button-2"              # 查询修改功能
    {
        while True
        {
            line_data = ""                          # 给全局变量赋初值
            # 调用查询函数，参数分别是标题和是否允许 textbox 双击输出
            shujvchaxun("  查询修改  ","Y")         # 先查询，双击查询列表，选中查询项后，将其写入全局变量 line_data 中
            if line_data == ""    {break}           # 没有选中数据，退出循环，返回主菜单
            time.sleep(200)                         # 延迟 200 毫秒，等待上个窗口资源清空，避免新窗口收到旧窗口消息
            shujvxiugai(line_data)                  # 进入数据修改界面

            box = GUI("box")                        # 创建 MessageBox 窗口对象
            box.MessageBox("是否继续查询修改？","确定|退出")     # 创建 MessageBox 窗口
            while box.MessageBoxClosed() != -1 {PowerDown(3)}     # 检测窗口关闭消息，或多键选择
            select = box.MessageBoxRead()      # 读出的数据只可能是-1、0、1
            if select != 0 {break}                  # 读出数据不为 0 表明，退出查询，否则继续查询
        }
    }
    elif dgjcd == "button-3"      # 查询删除功能
    {
        while True
        {
            line_data = ""
            shujvchaxun("  查询删除  ","Y")          # 先查询，双击查询列表，选中查询项后，将所选中的条目返回
            if line_data == "" {break}
            time.sleep(200)                         # 延迟 200 毫秒，等待上个窗口资源清空，避免新窗口收到旧窗口消息
            shujvshanchu(line_data)                 # 进入数据修改界面

            box = GUI("box")                        # 创建 MessageBox 窗口对象
            box.MessageBox("是否继续查询删除？","确定|退出")     # 创建 MessageBox 窗口
            while box.MessageBoxClosed() != -1 {PowerDown(3)}     # 检测窗口关闭消息，或多键选择
            select = box.MessageBoxRead()      # 读出的数据只可能是-1、0、1
```

```
            if select != 0 {break}                  # 读出数据不为 0 表明，退出查询，否则继续查询
        }
    }
    elif dgjcd == "button-4"    # 统计报表功能
    {
        # 统计报表二级菜单设计
        title1    = "统计报表"
        size1     = [32,11,50,16]
        text1     = ["","text",   14,1,20,1]
        button11 = ["  月报表  ","button",16,4,16,1]
        button12 = ["  周报表  ","button",16,6,16,1]
        button13 = ["退出统计报表","button",16,8,16,1]
        list_tj = [title1,size1,text1,button11,button12,button13]

        gui1 = GUI("fine")              # 创建二级嵌套 fine 窗口对象（在 gui 窗口内嵌套 gui1 窗口）
        gui1.Fine(list_tj)                                  # 创建菜单窗口
        gui1.SendText(["                统计报表"])       # 显示菜单名称
        while gui1.FineClosed() != -1                       # 循环检查窗口关闭消息
        {
            PowerDown(3)                                    # 低功耗设置
            if gui1.FineReady() == -1 {continue}      # 如果菜单项没有被选中，继续循环等待
            x = gui1.FineRead()                             # 获取菜单编号。
            selectbb = x[0]                                 # 将读取的指令 x[0]，赋值给变量

            # 这里不要直接使用  if x[0] == "button-0"  进行判断，而是将 x[0]赋值变量后，对变量进行判断
            # 因为根据 selectbb 的值，程序将转向其他模块，列表内的数据项，再虚拟机垃圾回收中，可能被回收掉，
            # 但是，不会回收掉一个有用的变量。
            if selectbb == "button-0"           # 所选时间段的月报表：按月收入、支出、结余，当月单次消费金额前 5 项
            {
                yuebaobiao()
            }
            elif selectbb == "button-1"         # 所选时间段的周报表：按周收入、支出、结余，当周单次消费金额前 5 项
            {
                zhoubaobiao()
            }
            elif selectbb == "button-2"         # 退出统计报表二级菜单，返回一级菜单
            {
                gui1.CloseFine()
            }
        }
    }
    elif dgjcd == "button-5"      # 退出主程序
    {
        gui.CloseFine()
    }

}

 sql.library_end()              # 无参数，无返回，功能是释放 MySQL 资源
 sql.close()                    # 无参数，无返回，释放 sql 对象




# 数据录入模块

# 数据录入模块

time = TIME()
os = OS()
sql = MYSQL()

sql = MYSQL()                   # 定义一个 sql 数据库变量，用于调用数据库方法
sql.library_init()              # 初始化库
sql = sql.connect("localhost","zhaojd","password","dbzhaojd",3306)     #连接库,返回 sql 对象。
sql.set_character("utf8")   # 设置数据库的字符编码

szlistbox1 = "支出,收入"
szlistbox2 = ""
fp = FILEOPEN("C:\\fine\\fin\\examples\\MySQL\\shangpinleibie.txt","r")     # 从该文件中读出商品列表数据
str = fp.read()

# 将读出的文件转化为字符串，用“,”替换换行符
count = str.count("\n")
for i in range(0,count,1)
{
    szlistbox2 = szlistbox2 + str.beforechr("\n") + ","
    str = str.afterchr("\n")
}
szlistbox2 = szlistbox2 + str

# 记录输入操作
def shujvluru()
{
    gui3 = GUI("fine")
    # 记录输入界面设计
```

```
    title = "账单录入"
    size   = [10,3,80,25]
    text1      = ["","text",24,2,12,1]                    # 记录输入——子菜单名
    button     = ["  录入  ","button",55,2,12,1]
    edit0      = [" ID ","edit","i",12,5,12,1]
    listbox1 = ["收入支出","combobox",szlistbox1,36,5,8,1]       # 起点 24+2 空格+10 标题=44
    edit1      = ["  数据录入人  ","edit","s",60,5,12,1]                                    # 起点 44+4 空格+12 标题=60
    edit2      = ["  日期  ","edit","s",12,7,12,1]                                        # 20+3 空格+6 年份=29 开始，37 结束
    listbox2 = ["商品分类","combobox",szlistbox2,12,9,24,1]
    edit3      = ["商品名称","edit","s",12,11,60,1]
    edit4      = ["  数量  ","edit","f",12,13,20,1]
    edit5      = ["  单价  ","edit","f",12,15,20,1]
    text2      = ["  合计  ","text",12,17,20,1]

    # 打包设计元素
    list1 = [title,size,text1,button,edit0,listbox1,edit1,edit2,listbox2,edit3,edit4,edit5,text2]

    time = TIME()

    gui3.Fine(list1)             # 创建账单录入窗口，返回资源 ID
    gui3.SendText(["    记录输入    ",""])
    while gui3.FineClosed() != -1     # 检查窗口关闭消息
    {
        PowerDown(3)                                  # 节能设置
        if gui3.FineReady() == 0        # 检查是否有数据录入
        {
            x = gui3.FineRead()          # 读取录入数据，释放资源
            # x 是一个列表，列表项数 = 1(command) + edit 控件个数 + combobox 控件个数 + editbox 控件个数
            # 本例程中，有 6 个 edit 控件、两个 combobox 控件，外加一个 command 指令，所以，共有 9 个列表项
            # x[0]是使用这些数据的指令，只有一个 button，指令是明确的，不用分析它。
            # 优先输出 edit：x[1]ID，x[2]录入人，x[3]日期，x[4]商品名称，x[5]数量，x[6]单价
            # 其次输出 listbox：x[7]收入支出，x[8]商品分类

            # 判断日期格式是否正确
            tm = x[3]
            if tm == "" or len(tm) < 10 or tm[4] != "-"
            {
                box = GUI("box")
                box.MessageBox("时间不能为空！并且时间格式必须是：NNNN-YY-RR!","确定")
                while box.MessageBoxClosed() != -1 {PowerDown(3)}
                continue
            }

            # 计算：合计 = 数量*单价
            y = x[5]*x[6]                              # 计算数量乘以单价，合计
            Y = sprint("%.2f",y)                       # 将合计转化为字符串
            gui3.SendText(["    票据录入    ",Y])         # 显示合计

            if x[1] == 0
            {
                IDnum = "null"                         # 如果没有输入 ID 号，用 null 表示，系统会自动添加 ID 号
            }
            else
            {
                IDnum = sprint("%",x[1])               # 将整数 ID 转化为字符串，如果输入的 ID 号重复，会报错。
            }
            shuliang = sprint("%.2f",x[5])           # 将数量转化为字符串，保留 2 位小数
            danjia     = sprint("%.2f",x[6])         # 将单价转化为字符串，保留 2 位小数


            time1 = x[3]+" 08:00:00"                   # 拼接开票时间（录入的票据时间）

            autotime = time.GetLocalTime()             # 自动获取发票录入时的系统时间
            date = autotime.beforechr(" ")           # 截取系统时间中的日期部分
            time2 = autotime.afterchr(" ")         # 获取系统时间的后半部分
            time2 = time2.beforechr(" ")             # 截取系统时间的时间部分
            time2 = date+" "+time2                     # 拼接日期+时间

            # 拼接 SQL 指令：其中的数字项前后不加单引号，在字符串项的前后添加单引号（注意：不能用双引号！）
            # IDnum、shuliang、danjia 分别是整数、或浮点数，在其前后，不能添加单引号，其余字段为字符串，须在前后添加单引号
            temp = IDnum +","+x[7]+"","+x[8]+"","+x[4]+"","+shuliang+","+danjia+","+x[2]+"","+time1+"","+time2+""

            SQL = "insert into daguanjia values("+temp+");"      # 拼接 SQL 指令
            sql = sql.query(SQL)                                    #向 table1 表格中插入两条记录。

            box = GUI("box")
            box.MessageBox("成功录入了一条记录！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}

            gui3.SendEdit(["",x[2],x[3],"","",""])    # 将查询选中的所有 edit 项记录显示出来，便于修改
            gui3.SendCombobox([x[7],x[8]])                # 将查询选中的所有 listbox 项记录显示出来，便于修改
            gui3.SendText(["    记录输入    ",""])
        }
    }
}
```

```
# 数据查询模块：SJCX.fin

time = TIME()
os = OS()
sql = MYSQL()

sql = MYSQL()                    # 定义一个 sql 数据库变量，用于调用数据库方法
sql.library_init()               # 初始化库
sql = sql.connect("localhost","zhaojd","password","dbzhaojd",3306)     #连接库,返回 sql 对象。
sql.set_character("utf8")     # 设置数据库的字符编码

szlistbox = "支出,收入"

vars = ["","","","",""]
# 查询操作：参数 1 窗口标题，参数 2 是 textbox 是否响应鼠标双击消息
def shujvchaxun(menuname,doubleclickvalid)
{
    gui3 = GUI("fine")
    # 查询界面设计
    title = "账单查询"
    size     = [30,10,120,30]

    text1        = ["","text",45,2,12,1]                    # 账单查询界面
    button       = ["  查询  ","button",85,2,12,1]
    listbox1    = ["收入支出","combobox",szlistbox,12,5,8,1]
    edit1        = ["起始时间","edit","s",34,5,12,1]
    edit2        = ["终止时间","edit","s",58,5,12,1]
    edit3        = ["商品名称","edit","s",12,7,60,1]
    textbox     = ["查询结果","textbox",doubleclickvalid,12,10,70,12]

    list2 = [title,size,text1,listbox1,edit1,edit2,edit3,button,textbox]

    gui3.Fine(list2)                                  # 创建账单录入窗口,返回资源 ID
    gui3.SendText([menuname])
    gui3.SendEdit([vars[0],vars[1],vars[2])     # 显示上一次的输入内容
    gui3.SendCombobox([vars[3])                      # 显示上一次的输入内容
    while gui3.FineClosed() != -1                  # 检查窗口关闭消息
    {
        PowerDown(3)                                        # 节能设置
        if gui3.FineReady() == 0                     # 检查是否有数据录入
        {
            x = gui3.FineRead()                        # 读取录入数据
            # x[0]指令，x[1]起始时间，x[2]终止时间，x[3]商品名称，x[4]收入支出
            # 本例程中 button 和 textbox 两个控件可以返回数据，因此，需要区别处理

            # 如果数据查询模块禁止数据输出，所以，这里尽管有 textbox 控件，却不会出现 x[0] = "textbox"的指令
            if x[0] == "textbox"
            {
                line_data = x[1]
                gui3.CloseFine()
                return
            }

            # 既然不是 textbox 指令，那只可能是 button 指令，即，查询指令，构造查询的 SQL 指令
            # vars 列表项用于再次查询时的初始数据，避免重复输入、
            vars = [x[1],x[2],x[3],x[4]]

            if x[1] != ""
            {
                tm = x[1]
                if tm == "" or len(tm) < 10 or tm[4] != "-"
                {
                    box = GUI("box")
                    box.MessageBox("时间不能为空！并且时间格式必须是：NNNN-YY-RR!","确定")
                    while box.MessageBoxClosed() != -1 {PowerDown(3)}
                    continue
                }
                time1 = x[1] + " 00:00:00"
                time1 = """+ time1 + """
            }

            if x[2] != ""
            {
                tm = x[2]
                if tm == "" or len(tm) < 10 or tm[4] != "-"
                {
                    box = GUI("box")
                    box.MessageBox("时间不能为空！并且时间格式必须是：NNNN-YY-RR!","确定")
                    while box.MessageBoxClosed() != -1 {PowerDown(3)}
                    continue
                }
                time2 = x[2] + " 23:59:59"
                time2 = """+ time2 + """
            }

            if x[4] != ""
```

```
        {
            SQL = "select * from daguanjia where in_out = " + """+x[4]+"""
            if x[1 != ""
            {
                SQL = SQL + " and shijian >= " + time1
            }

            if x[2] != ""
            {
                SQL = SQL + " and shijian <= " + time2
            }
            if x[3] != ""
            {
                SQL = SQL + " and name = " + """+x[3]+"""
            }
        }
        else
        {
            if x[1] != ""
            {
                SQL = "select * from daguanjia where shijian > " + time1
                if x[2] != ""
                {
                    SQL = SQL + " and shijian <= " + time2
                }
                if x[3] != ""
                {
                    SQL = SQL + " and name = " + """+x[3]+"""
                }
            }
            else
            {
                if x[2] != ""
                {
                    SQL = "select * from daguanjia where shijian <= " + time2
                    if x[3] != ""
                    {
                        SQL = SQL + " and name = " + """+x[3]+"""
                    }
                }
                else
                {
                    if x[3] != ""
                    {
                        SQL = "select * from daguanjia where name = " + """+x[3]+"""
                    }
                    else
                    {
                        box = GUI("box")
                        box.MessageBox("请输入查询条件！ ","确定")
                        while box.MessageBoxClosed() != -1 {PowerDown(3)}
                        continue
                    }
                }
            }
        }

        SQL = SQL + ";"     # SQL 指令

        gui3.SendTextbox(["FINECLEAR"])     # 这里的 FINECLEAR 是清除 Textbox 显示区全部内容的指令
        sql = sql.query(SQL)                                    # 执行 SQL 指令
        result = sql.use_result()                               # 读取查询结果，查询结果为双层列表。
        ix = len(result)                                        # 计算列表项数，即查询到的行数
        if ix == 0
        {
            box = GUI("box")
            box.MessageBox("没有查到符合条件的记录！ ","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            continue
        }
        char_result = ""                    # 定义一个空字符串
        for i in range(0,ix,1)
        {
            char_result = char_result + sprint("%\n",result[i])         # 将所有符合条件的记录相加
        }
        gui3.SendTextbox([char_result])     # 一次性显示所有符合条件的记录，避免频繁显示，引起抖动
    } # end of if gui2.FineReady()
} #  关闭窗口
}




#  数据修改模块：SJXG.fin

time = TIME()
os = OS()
sql = MYSQL()

sql = MYSQL()                   # 定义一个 sql 数据库变量，用于调用数据库方法
```

```
sql.library_init()                # 初始化库
sql = sql.connect("localhost","zhaojd","password","dbzhaojd",3306)     #连接库,返回 sql 对象。
sql.set_character("utf8")    # 设置数据库的字符编码

szlistbox1 = "支出,收入"
szlistbox2 = ""
fp = FILEOPEN(".\\fin\\examples\\MySQL\\shangpinleibie.txt","r")
str = fp.read()
count = str.count("\n")
for i in range(0,count,1)
{
    szlistbox2 = szlistbox2 + str.beforechr("\n")+","
    str = str.afterchr("\n")
}
szlistbox2 = szlistbox2 + str

# 记录修改操作
def shujvxiugai(selectStr)
{
    gui3 = GUI("fine")
    # 输入参数拆包 selectStr(是一个字符串，包含了一个记录的所有项，用逗号将其隔开)
    ID0 = selectStr.beforechr(",")                    # 取出 ID
    temp = selectStr.afterchr(",")                    # 去掉","之前的字符
    in_out0 = temp.beforechr(",")                     # 取出 in_out
    temp = temp.afterchr(",")                         # 去掉","之前的字符
    kind0 = temp.beforechr(",")                       # 取出 kind
    temp = temp.afterchr(",")                         # 去掉","之前的字符
    name0 = temp.beforechr(",")                       # 取出 name
    temp = temp.afterchr(",")                         # 去掉","之前的字符
    quantity0 = temp.beforechr(",")                   # 取出 quantity
    temp = temp.afterchr(",")                         # 去掉第一个","之前的字符
    price0 = temp.beforechr(",")                      # 取出 price
    Y = ctoi(quantity0)*ctoi(price0)                  # 计算总价
    Y = sprint("%.2f",Y)                              # 转换成字符串
    temp = temp.afterchr(",")                         # 去掉","之前的字符
    luren0 = temp.beforechr(",")                      # 取出 luren
    temp = temp.afterchr(",")                         # 去掉","之前的字符
    kaipiaotime0 = temp.beforechr(",")                # 取出日期时间
    kaipiaotime0 = kaipiaotime0.beforechr(" ")   # 去掉时间，只保留日期

    # 录入时间(lirutime0)是系统自动产生的，这里无需提取改时间

    # 查询修改界面设计
    title      = "记录修改"
    size       = [10,3,80,25]
    text1      = ["","text",24,2,12,1]                # 票据录入——子菜单名
    button     = ["  修改  ","button",55,2,12,1]
    edit0      = [" ID ","edit","i",12,5,12,1]
    listbox1 = ["收入支出","combobox",szlistbox1,36,5,8,1]       # 起点 24+2 空格+10 标题=44
    edit1      = ["  记录修改人  ","edit","s",60,5,12,1]                                    # 起点 44+4 空格+12 标题=60
    edit2      = ["  日期  ","edit","s",12,7,12,1]                                         # 20+3 空格+6 年份=29 开始，37 结束
    listbox2 = ["商品分类","combobox",szlistbox2,12,9,24,1]
    edit3      = ["商品名称","edit","s",12,11,60,1]
    edit4      = ["  数量  ","edit","f",12,13,20,1]
    edit5      = ["  单价  ","edit","f",12,15,20,1]
    text2      = ["  合计  ","text",12,17,20,1]

    # 打包设计元素
    list1 = [title,size,text1,button,edit0,listbox1,edit1,edit2,listbox2,edit3,edit4,edit5,text2]

    t1 = TIME()
    gui3.Fine(list1)                                  # 创建账单录入窗口
    gui3.SendText(["    记录修改    ",Y])             # 显示标题

    # 将查询选中的所有 edit 项记录显示出来，便于修改
    gui3.SendEdit([ID0,luren0,kaipiaotime0,name0,quantity0,price0])
    gui3.SendCombobox([in_out0,kind0])        # 将查询选中的所有 listbox 项记录显示出来，便于修改

    while gui3.FineClosed() != -1                     # 检查窗口关闭消息
    {
        PowerDown(3)                                  # 节能设置
        if gui3.FineReady() == 0                      # 检查是否有数据录入
        {
            x = gui3.FineRead()                       # 读取录入数据

            # x[0]是 button 指令
            # 优先输出 edit：x[1]ID，x[2]数据录入人，x[3]日期,x[4]商品名称，x[5]数量，x[6]单价
            # 其次输出 listbox：x[7]收入支出，x[8]商品分类

            y = x[5]*x[6]                             # 计算数量乘以单价，合计
            Y = sprint("%.2f",y)                      # 将合计转化为字符串
            gui3.SendText(["    记录修改    ",Y])    # 显示合计

            if x[1] == 0
            {
                IDnum = "null"                        # 如果没有输入 ID 号，用 null 表示，系统会自动添加 ID 号
```

```
            }
            else
            {
                IDnum = sprint("%",x[1])            # 将整数 ID 转化为字符串，如果输入的 ID 号重复，会报错。
            }
            shuliang = sprint("%.2f",x[5])          # 将数量转化为字符串，保留 2 位小数
            danjia     = sprint("%.2f",x[6])        # 将单价转化为字符串，保留 2 位小数

            time1 = x[3]+" 00:00:00"                # 拼接开票时间（录入的票据时间）

            autotime = time.GetLocalTime()                  # 自动获取发票录入时的系统时间
            date = autotime.beforechr(" ")          # 截取系统时间中的日期部分
            time2 = autotime.afterchr(" ")          # 获取系统时间的后半部分
            time2 = time2.beforechr(" ")            # 截取系统时间的时间部分
            time2 = date+" "+time2                  # 拼接日期+时间

            # 拼接 SQL 指令：其中的数字项前后不加单引号，在字符串项的前后添加单引号（注意：不能用双引号！）
            # IDnum、shuliang、danjia 分别是整数、或浮点数，在其前后，不能添加单引号，其余字段为字符串，须在前后添加单引号

            temp = IDnum +","'"+x[7]+"'","'"+x[8]+"'","'"+x[4]+"'","+shuliang+","+danjia+","'"+x[2]+"'","'"+time1+"'","'"+time2+"'"'

            # 构造替换命令
            SQL = "replace into daguanjia (id,in_out,kind,name,quantity,price,lururen,shijian,lurutime) values(" + temp + ");"
            sql = sql.query(SQL)      # 替换原来的记录，id 编号保留原来的编号

            # 消息框提示修改成功
            box = GUI("box")
            box.MessageBox("成功修改了一条记录！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}

            gui3.CloseFine()
        }
    }
}




# 数据删除模块：SJSC.fin

time = TIME()
os = OS()
sql = MYSQL()

sql = MYSQL()                   # 定义一个 sql 数据库变量，用于调用数据库方法
sql.library_init()              # 初始化库
sql = sql.connect("localhost","zhaojd","password","dbzhaojd",3306)     #连接库,返回 sql 对象。
sql.set_character("utf8")   # 设置数据库的字符编码

szlistbox1 = "支出,收入"
szlistbox2 = ""
fp = FILEOPEN(".\\fin\\examples\\MySQL\\shangpinleibie.txt","r")
str = fp.read()
count = str.count("\n")
for i in range(0,count,1)
{
    szlistbox2 = szlistbox2 + str.beforechr("\n")+","
    str = str.afterchr("\n")
}
szlistbox2 = szlistbox2 + str

# 数据删除操作
def shujvshanchu(selectStr)
{
    gui3 = GUI("fine")
    # 输入参数拆包 selectStr(是一个字符串，包含了一个记录的所有项，用逗号将其隔开)
    ID0 = selectStr.beforechr(",")              # 取出 ID
    temp = selectStr.afterchr(",")              # 去掉","之前的字符
    in_out0 = temp.beforechr(",")               # 取出 in_out
    temp = temp.afterchr(",")                   # 去掉","之前的字符
    kind0 = temp.beforechr(",")                 # 取出 kind
    temp = temp.afterchr(",")                   # 去掉","之前的字符
    name0 = temp.beforechr(",")                 # 取出 name
    temp = temp.afterchr(",")                   # 去掉","之前的字符
    quantity0 = temp.beforechr(",")             # 取出 quantity
    temp = temp.afterchr(",")                   # 去掉第一个","之前的字符
    price0 = temp.beforechr(",")                # 取出 price

    y = ctoi(quantity0)*ctoi(price0)            # 计算总价
    Y = sprint("%.2f",y)                        # 转换成字符串

    temp = temp.afterchr(",")                   # 去掉","之前的字符
    luren0 = temp.beforechr(",")                # 取出 luren

    temp = temp.afterchr(",")                   # 去掉","之前的字符
    kaipiaotime0 = temp.beforechr(",")          # 取出日期时间
    kaipiaotime0 = kaipiaotime0.beforechr(" ")  # 去掉时间，只保留日期
```

```
    # 录入时间(lirutime0)是系统自动产生的，这里无需提取改时间

    # 查询修改界面设计
    title      = "记录修改"
    size       = [10,3,80,28]
    text1      = ["","text",24,2,12,1]                    # 记录删除——子菜单名
    button     = ["  删除  ","button",55,2,12,1]
    edit0      = [" ID ","edit","i",12,5,12,1]
    listbox1 = ["收入支出","combobox",szlistbox1,36,5,8,1]      # 起点 24+2 空格+10 标题=44
    edit1      = ["  记录修改人  ","edit","s",60,5,12,1]                                    # 起点 44+4 空格+12 标题=60
    edit2      = ["  日期  ","edit","s",12,7,12,1]                                         # 20+3 空格+6 年份=29 开始，37 结束
    listbox2 = ["商品分类","combobox",szlistbox2,12,9,24,1]
    edit3      = ["商品名称","edit","s",12,11,60,1]
    edit4      = ["  数量  ","edit","f",12,13,20,1]
    edit5      = ["  单价  ","edit","f",12,15,20,1]
    text2      = ["  合计  ","text",12,17,20,1]

    # 打包设计元素
    list1 = [title,size,text1,button,edit0,listbox1,edit1,edit2,listbox2,edit3,edit4,edit5,text2]

    t1 = TIME()
    gui3.Fine(list1)                                    # 创建账单录入窗口
    gui3.SendText(["    记录删除    ",Y])              # 显示标题
    gui3.SendEdit([ID0,luren0,kaipiaotime0,name0,quantity0,price0])    # 将查询选中的所有 edit 项记录显示出来，便于修改
    gui3.SendCombobox([in_out0,kind0])       # 将查询选中的所有 listbox 项记录显示出来，便于修改

    while gui3.FineClosed() != -1                       # 检查窗口关闭消息
    {
        PowerDown(3)                                                # 节能设置
        if gui3.FineReady() == 0                        # 检查是否有数据录入
        {
            x = gui3.FineRead()                         # 读取录入数据

            # x[0]是指令，只用一个 button 指令
            # 优先输出 edit：x[1]ID，x[2]数据录入人，x[3]日期,x[4]商品名称，x[5]数量，x[6]单价
            # 其次输出 combobox：x[7]收入支出，x[8]商品分类

            y = x[5]*x[6]                                           # 计算数量乘以单价，合计
            Y = sprint("%.2f",y)                                    # 将合计转化为字符串
            gui3.SendText(["    记录删除    ",Y])    # 显示合计

            if x[1] == 0
            {
                IDnum = "null"                          # 如果没有输入 ID 号，用 null 表示，系统会自动添加 ID 号
            }
            else
            {
                IDnum = sprint("%",x[1])            # 将整数 ID 转化为字符串，如果输入的 ID 号重复，会报错。
            }
            shuliang = sprint("%.2f",x[5])          # 将数量转化为字符串，保留 2 位小数
            danjia     = sprint("%.2f",x[6])        # 将单价转化为字符串，保留 2 位小数


            time1 = x[3]+" 00:00:00"                # 拼接开票时间（录入的票据时间）

            autotime = time.GetLocalTime()                  # 自动获取发票录入时的系统时间
            date = autotime.beforechr(" ")          # 截取系统时间中的日期部分
            time2 = autotime.afterchr(" ")          # 获取系统时间的后半部分
            time2 = time2.beforechr(" ")            # 截取系统时间的时间部分
            time2 = date+" "+time2                  # 拼接日期+时间

            # 拼接 SQL 指令：其中的数字项前后不加单引号，在字符串项的前后添加单引号（注意：不能用双引号！）
            # IDnum、shuliang、danjia 分别是整数、或浮点数，在其前后，不能添加单引号，其余字段为字符串，须在前后添加单引号

            # 构造删除命令（这里删除条件是 id 号）
            SQL = "delete from daguanjia where id = " + IDnum + ";"
            sql = sql.query(SQL)            # 替换原来的记录，id 编号保留原来的编号

            # 消息框提示删除成功
            box = GUI("box")                    # 创建 MessageBox 窗口对象
            box.MessageBox("成功删除一条记录！","确定")                 # 创建窗口
            while box.MessageBoxClosed() != -1 {PowerDown(3)}       # 检测窗口关闭消息

            gui3.CloseFine()
        }
    }
}



# 月报表模块：YDBB.fin

time = TIME()
```

```
os = OS()

sql = MYSQL()                    # 定义一个 sql 数据库变量，用于调用数据库方法
sql.library_init()               # 初始化库
sql = sql.connect("localhost","zhaojd","password","dbzhaojd",3306)     #连接库,返回 sql 对象。
sql.set_character("utf8")   # 设置数据库的字符编码

def yuebaobiao()
{
    gui3 = GUI("fine")
    # 查询界面设计
    title     = "月报表"
    size      = [30,10,120,35]
    text1     = ["","text",45,2,12,1]                    # 月报表界面
    button    = ["  报表  ","button",85,2,12,1]
    edit1     = ["起始时间","edit","s",22,5,12,1]
    edit2     = ["终止时间","edit","s",46,5,12,1]
    textbox = ["查询结果","textbox","N",12,8,70,16]

    list2 = [title,size,text1,button,edit1,edit2,textbox]
    gui3.Fine(list2)                                        # 创建账单录入窗口
    gui3.SendText(["    月报表    "])
    while gui3.FineClosed() != -1                          # 检查窗口关闭消息
    {
        PowerDown(3)                                                # 节能设置
        if gui3.FineReady() == 0                           # 检查是否有数据录入
        {
            x = gui3.FineRead()                             # 读取录入数据
            gui3.SendTextbox(["FINECLEAR"])         # 向一个 textbox 控件上发送"FINECLEAR"，会清空 Textbox 上原来显示的数据

            # x[0]指令
            # x[1]起始时间，x[2]终止时间

            tm0 = x[1]                                  # 起始时间
            tm1 = x[2]                                  # 截止时间

            if tm0 == "" or tm1 == "" or len(tm0) < 10 or tm0[4] != "-" or len(tm1) < 10 or tm1[4] != "-"
            {
                box = GUI("box")
                box.MessageBox("起始时间和截止时间都不能为空！并且时间格式必须为：nnnn-yy-rr！","确定")
                while box.MessageBoxClosed() != -1 {PowerDown(3)}
                continue
            }

            nc1    = tm1.beforechr("-")          # 截止年份（字符串）
            inc1 = ctoi(nc1)                      # 截止年份（数字）
            temp = tm1.afterchr("-")
            yc1    = temp.beforechr("-")         # 截至月份（字符串）
            iyc1 = ctoi(yc1)                      # 截止月份（数字）

            nc0    = tm0.beforechr("-")          # 起始年份（字符串）
            inc0 = ctoi(nc0)                      # 起始年份（数字）
            temp = tm0.afterchr("-")
            yc0    = temp.beforechr("-")         # 起始月份（字符串）
            iyc0 = ctoi(yc0)                      # 起始月份（数字）

            yueshu = (ctoi(nc1) - ctoi(nc0))*12 + ctoi(yc1) - ctoi(yc0) +1

            total_in    = 0
            total_out = 0

            for i in range(0,yueshu,1)
            {
                time1 = itoc(inc0) + "-" + itoc(iyc0) + "-" + "01" + " 00:00:00"     # 当月的第一天
                time1 = """+ time1 + """
                time2 = itoc(inc0) + "-" + itoc(iyc0) + "-" + "31" + " 23:59:59"     # 当月的最后一天
                time2 = """+ time2 + """

                # 计算总收入
                SQL = "select quantity,price from daguanjia where in_out = " + "'收入'"
                SQL = SQL +" and shijian >= " + time1 + " and shijian <= " + time2 + ";"
                sql = sql.query(SQL)                                  # 执行 SQL 指令
                result = sql.use_result()                           # 读取查询结果，查询结果为双层列表，所有数据均是字符串形式。
                ix = len(result)                                      # 计算列表项数，即查询到的行数
                zongshouru = 0                                          # 总收入
                for j in range(0,ix,1)
                {
                    single_list = result[j]
                    zongshouru = zongshouru + ctoi(single_list[0])*ctoi(single_list[1])
                }

                # 计算总支出
                SQL = "select quantity,price from daguanjia where in_out = " + "'支出'"
                SQL = SQL +" and shijian >= " + time1 + " and shijian <= " + time2 + ";"
                sql = sql.query(SQL)                                  # 执行 SQL 指令
                result = sql.use_result()                           # 读取查询结果，查询结果为双层列表，所有数据均是字符串形式。
                ix = len(result)                                      # 计算列表项数，即查询到的行数
```

```
    zongzhichu = 0                                          # 总收入
    for j in range(0,ix,1)
    {
        single_list = result[j]
        zongzhichu = zongzhichu + ctoi(single_list[0])*ctoi(single_list[1])
    }
    # 显示当月收、支、结余
    displayStr = ""                 # 显示缓冲区（标题行）
    displayStr = displayStr + itoc(inc0)+"年"+itoc(iyc0)+"月报表"
    displayStr = displayStr + "            收入："+sprint("%.2f",zongshouru)
    displayStr = displayStr + "            支出："+sprint("%.2f",zongzhichu)
    displayStr = displayStr + "            结余："+sprint("%.2f",zongshouru - zongzhichu)+"\n"
    gui3.SendTextbox([displayStr])          # 显示

    # 当月收入明细
    displayStr = ""                 # 显示缓冲区（标题行）
    displayStr = displayStr + "本月收入清单：\n"
    temp = sprintTab("商品名称",30,0,1)+sprintTab("数量",10,0,1)\
           +sprintTab("单价",10,0,1)+sprintTab("小计",10,0,1)+sprintTab("交易时间",12,0,1)+"\n"
    displayStr = displayStr + temp
    gui3.SendTextbox([displayStr])

    SQL = "SELECT name,quantity,price,shijian,quantity * price AS total_price from daguanjia where in_out = " + "'收入'"
    SQL = SQL +" and shijian > " + time1 + " and shijian < " + time2
    SQL = SQL +" ORDER BY total_price DESC" + ";"

    sql = sql.query(SQL)                                    # 执行 SQL 指令
    result = sql.use_result()                               # 读取查询结果，查询结果为双层列表。
    ix = len(result)
    displayStr = ""                 # 显示缓冲区
    for j in range(0,ix,1)                                  # 遍历所有消费记录
    {
        single_list = result[j]                             # 每循环一次，取出一个子列表
        spm = sprintTab(single_list[0],30,0,1)
        sl  = sprintTab(single_list[1],10,2,1)
        dj  = sprintTab(single_list[2],10,2,1)
        xj  = sprintTab(single_list[4],10,2,1)
        rq  = single_list[3]
        rq  = rq.beforechr(" ")                             # 只保留日期，抛弃时间
        rq  = sprintTab(rq,12,0,1)
        displayStr = displayStr + spm+sl+dj+xj+rq+"\n"      # 将所有记录信息，添加到字符串 displayStr 中
    }
    gui3.SendTextbox([displayStr+"\n"])     # 显示记录，并添加一个换行

    # 当月支出明细
    displayStr = ""                 # 显示缓冲区（标题行）
    displayStr = displayStr + "本月支出清单：\n"
    temp = sprintTab("商品名称",30,0,1)+sprintTab("数量",10,0,1)\
           +sprintTab("单价",10,0,1)+sprintTab("小计",10,0,1)+sprintTab("交易时间",12,0,1)+"\n"
    displayStr = displayStr + temp
    gui3.SendTextbox([displayStr])

    SQL = "SELECT name,quantity,price,shijian,quantity * price AS total_price from daguanjia where in_out = " + "'支出'"
    SQL = SQL +" and shijian > " + time1 + " and shijian < " + time2
    SQL = SQL +" ORDER BY total_price DESC" + ";"

    sql = sql.query(SQL)                                    # 执行 SQL 指令
    result = sql.use_result()                               # 读取查询结果，查询结果为双层列表。

    ix = len(result)
    displayStr = ""                 # 显示缓冲区
    for j in range(0,ix,1)                                  # 遍历所有消费记录
    {
        single_list = result[j]                             # 每循环一次，取出一个子列表
        spm = sprintTab(single_list[0],30,0,1)
        sl  = sprintTab(single_list[1],10,2,1)
        dj  = sprintTab(single_list[2],10,2,1)
        xj  = sprintTab(single_list[4],10,2,1)
        rq  = single_list[3]
        rq  = rq.beforechr(" ")                             # 只保留日期，抛弃时间
        rq  = sprintTab(rq,12,0,1)
        displayStr = displayStr + spm+sl+dj+xj+rq+"\n"      # 累加字符串
    }
    gui3.SendTextbox([displayStr+"\n\n"])     # 一次性显示累加字符串，

    if iyc0 + 1 > 12                                        # 计算下个月份
    {
        inc0 = inc0 +1
        iyc0 = 1
    }
    else
    {
        iyc0 = iyc0 + 1
    }
    total_in  = total_in  + zongshouru
    total_out = total_out + zongzhichu
}   # end of for i in range(0,yueshu,1)
```

```
            # 期间总收入、总支出、总结余
            displayStr = ""                          # 显示缓冲区
            displayStr = displayStr + "期间合计："
            displayStr = displayStr + "            合计收入："+sprint("%.2f",total_in)
            displayStr = displayStr + "            合计支出："+sprint("%.2f",total_out)
            displayStr = displayStr + "            结余："+sprint("%.2f",total_in - total_out)+"\n\n"
            gui3.SendTextbox([displayStr])
        }    # end of if gui3.InputReady() == 0
    }    # end of while gui3.InputClosed() != -1
}




# 周报表模块：ZDBB.fin

time = TIME()
os = OS()

sql = MYSQL()                    # 定义一个 sql 数据库变量，用于调用数据库方法
sql.library_init()               # 初始化库
sql = sql.connect("localhost","zhaojd","password","dbzhaojd",3306)     #连接库,返回 sql 对象。
sql.set_character("utf8")   # 设置数据库的字符编码

# 周报表
def zhoubaobiao()
{
    gui3 = GUI("fine")
    # 查询界面设计
    title     = "周报表"
    size      = [30,10,120,35]
    text1     = ["","text",45,2,12,1]                    # 月报表界面
    button    = [" 报表 ","button",85,2,12,1]
    edit1     = ["起始时间","edit","s",22,5,12,1]
    edit2     = ["终止时间","edit","s",46,5,12,1]
    textbox = ["查询结果","textbox","N",12,8,70,16]

    list2 = [title,size,text1,button,edit1,edit2,textbox]

    gui3.Fine(list2)                          # 创建账单录入窗口
    gui3.SendText["    周报表    "])
    while gui3.FineClosed() != -1             # 检查窗口关闭消息
    {
        PowerDown(3)                                    # 节能设置
        if gui3.FineReady() == 0                # 检查是否有数据录入
        {
            x = gui3.FineRead()                         # 读取录入数据
            gui3.SendTextbox(["FINECLEAR"])   # 清空 Textbox 上原来显示的数据

            # x[0]是 button 指令
            # x[1]起始时间，x[2]终止时间

            tm0 = x[1]                          # 起始时间
            tm1 = x[2]                          # 截止时间

            if tm0 == "" or tm1 == "" or len(tm0) < 10 or tm0[4] != "-" or len(tm1) < 10 or tm1[4] != "-"
            {
                box = GUI("box")
                box.MessageBox("起始时间和截止时间都不能为空！并且时间格式必须为：nnnn-yy-rr！","确定")
                while box.MessageBoxClosed() != -1 {PowerDown(3)}
                continue
            }

             # 计算终止年月日
            nc1    = tm1.beforechr("-")        # 截止年份（字符串）
            inc1 = ctoi(nc1)                    # 截止年份（数字）
            temp = tm1.afterchr("-")
            yc1    = temp.beforechr("-")       # 截至月份（字符串）
            iyc1 = ctoi(yc1)                    # 截止月份（数字）
            rc1    = temp.afterchr("-")       # 截至日期（字符串）
            irc1 = ctoi(rc1)                    # 截至日期（数字）

             # 计算起始年月日
            nc0    = tm0.beforechr("-")        # 起始年份（字符串）
            inc0 = ctoi(nc0)                    # 起始年份（数字）
            temp = tm0.afterchr("-")
            yc0    = temp.beforechr("-")       # 起始月份（字符串）
            iyc0 = ctoi(yc0)                    # 起始月份（数字）
            rc0    = temp.afterchr("-")        # 起始日期（字符串）
            irc0 = ctoi(rc0)                    # 起始日期（数字）

             # 计算期间周数
            listt1 = [inc1,iyc1,irc1,8,0,0]             # 终止时间列表
            shijianchuo1 = time.LocalListToUtcTime(listt1)      # 将终止时间转化为时间戳
            listt0 = [inc0,iyc0,irc0,8,0,0]             # 起始时间列表
```

```
shijianchuo0 = time.LocalListToUtcTime(listt0)      # 将起始时间转化为时间戳
zhoushu    = (shijianchuo1-shijianchuo0)/1000/1000/10/(24*60*60)/7     # 期间周数
zhoushu    = int(zhoushu) + 1                                  # 取整 +1，周数


# 计算起始周的周日，对应的日期
listt2 = time.UtcToLocalListTime(shijianchuo0)                    # 将时间戳转化为日期时间列表
week = listt2[6]

shijianchuo = shijianchuo0 - week*24*60*60*1000*1000*10        # 对应起始日期所在的周的星期日（周起始日）的时间戳
listt0 = time.UtcToLocalListTime(shijianchuo)                  # 起始周的周日的时间列表

total_in    = 0
total_out = 0

for i in range(0,zhoushu,1)
{
    time1 = itoc(listt0[0]) + "-" + itoc(listt0[1]) + "-" + itoc(listt0[2]) + " 00:00:00"     # 当周日第一天
    time1 = """+ time1 + """

    shijianchuo = shijianchuo + 6*24*60*60*1000*1000*10
    listt1 = time.UtcToLocalListTime(shijianchuo)                  # 当周六的时间列表
    time2 = itoc(listt1[0]) + "-" + itoc(listt1[1]) + "-" + itoc(listt1[2]) + " 23:59:59"     # 当月的最后一天
    time2 = """+ time2 + """
    zongshouru = 0

    # 计算总收入
    SQL = "select quantity,price from daguanjia where in_out = " + "'收入'"
    SQL = SQL +" and shijian >= " + time1 + " and shijian <= " + time2 + ";"
    sql = sql.query(SQL)                                    # 执行 SQL 指令
    result = sql.use_result()                               # 读取查询结果，查询结果为双层列表，所有数据均是字符串形式。
    ix = len(result)                                        # 计算列表项数，即查询到的行数
    zongshouru = 0                                          # 总收入
    for j in range(0,ix,1)
    {
        single_list = result[j]
        zongshouru = zongshouru + ctoi(single_list[0])*ctoi(single_list[1])
    }
    # 计算总支出
    SQL = "select quantity,price from daguanjia where in_out = " + "'支出'"
    SQL = SQL +" and shijian >= " + time1 + " and shijian <= " + time2 + ";"
    sql = sql.query(SQL)                                    # 执行 SQL 指令
    result = sql.use_result()                               # 读取查询结果，查询结果为双层列表，所有数据均是字符串形式。
    ix = len(result)                                        # 计算列表项数，即查询到的行数
    zongzhichu = 0                                          # 总收入
    for j in range(0,ix,1)
    {
        single_list = result[j]
        zongzhichu = zongzhichu + ctoi(single_list[0])*ctoi(single_list[1])
    }
    # 显示当周收、支、结余
    displayStr = ""                 # 显示缓冲区
    displayStr = displayStr + itoc(listt0[0])+"年"+itoc(listt0[1])+"月"+itoc(listt0[2])+"日报表"
    displayStr = displayStr + "            收入："+sprint("%.2f",zongshouru)
    displayStr = displayStr + "            支出："+sprint("%.2f",zongzhichu)
    displayStr = displayStr + "            结余："+sprint("%.2f",zongshouru - zongzhichu)+"\n"
    gui3.SendTextbox([displayStr])

    # 当周收入明细
    displayStr = ""                 # 显示缓冲区
    displayStr = displayStr + "本周收入明细：\n"
    temp = sprintTab("商品名称",30,0,1)+sprintTab("数量",10,0,1)\
          +sprintTab("单价",10,0,1)+sprintTab("小计",10,0,1)+sprintTab("交易时间",12,0,1)+"\n"
    displayStr = displayStr + temp
    gui3.SendTextbox([displayStr])

    SQL = "SELECT name,quantity,price,shijian,quantity * price AS total_price from daguanjia where in_out = "
    SQL = SQL + " '收入'"+" and shijian > " + time1 + " and shijian < " + time2
    SQL = SQL +" ORDER BY total_price DESC" + ";"

    sql = sql.query(SQL)                                    # 执行 SQL 指令
    result = sql.use_result()                               # 读取查询结果，查询结果为双层列表。
    ix = len(result)
    displayStr = ""                 # 显示缓冲区
    for j in range(0,ix,1)                                  # 遍历所有消费记录
    {
        single_list = result[j]                             # 每循环一次，取出一个子列表
        spm = sprintTab(single_list[0],30,0,1)
        sl    = sprintTab(single_list[1],10,2,1)
        dj    = sprintTab(single_list[2],10,2,1)
        xj    = sprintTab(single_list[4],10,2,1)
        rq    = single_list[3]
        rq    = rq.beforechr(" ")                           # 只保留日期，抛弃时间
        rq    = sprintTab(rq,12,0,1)
        displayStr = displayStr + spm+sl+dj+xj+rq+"\n"
    }
    gui3.SendTextbox([displayStr+"\n"])          # 显示当周明细，外加一空行，避免在循环体内显示，以防抖动
```

```
                # 当周支出明细
                displayStr = ""                 # 显示缓冲区
                displayStr = displayStr + "本周支出明细：\n"
                temp = sprintTab("商品名称",30,0,1)+sprintTab("数量",10,0,1)\
                      +sprintTab("单价",10,0,1)+sprintTab("小计",10,0,1)+sprintTab("交易时间",12,0,1)+"\n"
                displayStr = displayStr + temp
                gui3.SendTextbox([displayStr])

                SQL = "SELECT name,quantity,price,shijian,quantity * price AS total_price from daguanjia where in_out = "
                SQL = SQL + " '支出'"+" and shijian > " + time1 + " and shijian < " + time2
                SQL = SQL +" ORDER BY total_price DESC" + ";"
                sql = sql.query(SQL)                                    # 执行 SQL 指令
                result = sql.use_result()                               # 读取查询结果，查询结果为双层列表。
                ix = len(result)
                displayStr = ""                 # 显示缓冲区
                for j in range(0,ix,1)                                  # 遍历所有消费记录
                {
                    single_list = result[j]                             # 每循环一次，取出一个子列表
                    spm = sprintTab(single_list[0],30,0,1)
                    sl  = sprintTab(single_list[1],10,2,1)
                    dj  = sprintTab(single_list[2],10,2,1)
                    xj  = sprintTab(single_list[4],10,2,1)
                    rq  = single_list[3]
                    rq  = rq.beforechr(" ")                             # 只保留日期，抛弃时间
                    rq  = sprintTab(rq,12,0,1)
                    displayStr = displayStr + spm+sl+dj+xj+rq+"\n"
                }
                gui3.SendTextbox([displayStr+"\n\n"])                   # 显示记录，外加两个空行
                total_in  = total_in  + zongshouru
                total_out = total_out + zongzhichu
                # 计算下一周日的日期
                shijianchuo = shijianchuo + 24*60*60 *1000*1000*10      # 周六+1 天
                listt0 = time.UtcToLocalListTime(shijianchuo)           # 起始周的周日的时间列表
            }
            # 期间总收入、总支出、结余
            displayStr = ""                 # 显示缓冲区
            displayStr = displayStr + "期间合计："
            displayStr = displayStr + "          合计收入："+sprint("%.2f",total_in)
            displayStr = displayStr + "          合计支出："+sprint("%.2f",total_out)
            displayStr = displayStr + "          结余："+sprint("%.2f",total_in - total_out)+"\n\n"
            gui3.SendTextbox([displayStr])
        }
    }
}
```

第二十章、网络编程　例程《net_method》

net 方法集

net = NET() 功能：定义 NET 对象 net，无参数，新定义的 net 对象的 socket 属性为 NULL

net = net.socket(net.IPv4,net.STREAM,net.TCP)　功能：给 net 赋值
　　参数 1 是地址族，参数 2 是数据流格式，参数 3 是协议选择
　　返回一个 NET 对象 net，包含 net 的 socket 属性

net.timeout(3000) 功能：设置 net 的接收和发送超时时间,参数是超时时间，单位毫秒，无返回

net.clear()　功能：释放网络访问资源，无参数，无返回

net.close()　功能：释放一个 net 对象（包含释放其中的 socket），无参数，无返回，

local_ip = net.local_addr()　功能：返回本地主机地址，无参数
　　如果没有接入网络，返回 172.0.0.1，如果接入网络，返回本地网络地址

net.bind(net.IPv4,"172.0.0.1",8080)　更能：将服务器的 net 与服务器参数（地址族、地址、端口号）关联
　　参数 1 本地地址族，参数 2 本地地址，参数 3 服务的端口号，无返回

net.listen(max_clients)　功能：将服务器设置为被动监听模式
　　参数是服务器的最大并发服务个数（即，最多能支持的同时访问的客户端数量），无返回

以上方法都是即时方法，无需时间等待，下面的方法都是需要网络等待，因此，下述方法都需要创建线程来完成

连接命令组
net.connect(net.IPv4,"172.0.0.1",8080)　启动连接，将客户端的 net 与服务器的参数（地址族、地址、端口号）关联
net.connectOK()　　无参数，检测 connect()函数是否退出（返回-1 表明退出函数，超时设置不影响 connect 函数）
　　如果 net.connectOK() == 0 表明还处于连接过程中，等于 -1 表明连接终止，连接是否成功，
　　还需要通过 net.connectRead()读取结果，来判断是连接成功，还是连接失败
net.connectRead()　　无参数，连接成功返回 0，连接不成功返回-1
注意：connect()超时退出时间固定位 3*5 秒。connect 不受 net.timeout()超时设置影响。

net.timeout(ms)超时时间设置，对下面的函数都有效。

监听客户端接入命令组
net.accept()　　无参数，启动 accept()检测客户端接入
net.acceptOK()　　无参数，检测 accept()是否退出（结束返回-1，进行中返回 0）
　　net.acceptOK() == 0 表明还处于连接过程中，等于 -1 表示连接结束，
　　至于是否连接成功，还需要通过 net.acceptRead()读取结果，来判断。
net.acceptRead()　　无参数，返回列表包含三项：
　　第一项为-1 超时，为 0 有客户接入；第二项是客户端的 net 对象，第三项是客户端的地址

TCP 发送命令组
sendnet = net.sendNet()　　为一次 TCP 发送服务，定制一个网络发送对象（sendnet），本次发送基于 sendnet 操作

在发送数据时，我们并不区分缓冲区 sendbuf 中是字符串、或者二进制数据块，接收过程需要区分。
sendnet.send(sendbuf,length)　　创建线程，启动发送。参数 1 是待发送数据，参数 2 是发送数据的字节数
sendnet.sendOK()　　检测 send()线程是否退出（结束返回-1，进行中返回 0），如果退出线程，立即释放所占用资源。
　　sendnet.sendOK() = 0 表明还处于发送过程中，等于 -1 表明发送结束
　　至于是否发送成功，还需要通过进一步通过 sendnet.sendRead()读取结果来判断。
sendnet.sendRead()　　返回发送的字节数，字节数为-1 表明发送错误

TCP 接收命令组
recvnet = net.recvNet()　　为一次 TCP 接收服务，定制一个网络接收对象（recvnet），本次接收基于 recvnet 操作
recvnet.recv()　　创建线程，启动接收。
recvnet.recvOK()　　检测 recv()线程是否退出（结束返回-1，进行中返回 0），如果退出线程，立即释放所占用资源。
　　recvnet.recvOK() = 0 表明仍处于接收过程中，等于 -1 表明接收过程结束，
　　至于接收是否正确，尚需进一步通过 recvnet.recvRead()读取结果，进行判断。
在接收过程中，需要通过参数"string"、或"binary"，明确指明收到的是字符串，还是二进制数据块，便于虚拟机正确处理。
recvnet.recvRead("string")　　参数"string"表示接收的是文本字符串，"binary"表示接收的是二进制字符串。
返回列表包含二项：第一项为-1 表明接收错误，否则，表示收到的数据长度；第二项为收到的数据

UDP 发送命令组
udpsendnet = net.udpsendNet()　　为一次 UDP 发送服务，定制一个网络发送对象，本次发送基于 udpsendnet 操作

在发送数据时，我们并不区分缓冲区 sendbuf 中是字符串、或者二进制数据块，接收过程需要区分。
参数是：地址族、收信地址、端口号、待发数据、待发数据长度
udpsendnet.udpsend(net.IPv4,"192.168.232.195",54321,sendbuf,length)

udpsendnet.udpsendOK()　检测 udpsend()线程是否结束（返回-1 表明函数退出），如果退出线程，立即释放所占用资源。
udpsendnet.udpsendOK() = 0 表明仍处于发送过程中， -1 表明发送结束，
至于发送是否正确，还需要通过 udpsendnet.udpsendRead()读取结果，进行判断。
udpsendnet.udpsendRead()　返回发送的字节数（-1 表示发送错误）

UDP 接收命令组
udprecvnet = net.udprecvNet()　为一次 UDP 接收服务，定制一个网络接收对象，本次接收基于 udprecvnet 操作
udprecvnet.udprecv()　启动接收，启动接收。
udprecvnet.udprecvOK()　检测 udprecv()过程是否结束（结束返回-1，进行中返回 0）
udprecvnet.udprecvOK() = 0 表明仍处于接收过程中，等于 -1 表明接收结束，
至于接收是否正确，还需要通过 udprecvnet.udprecvRead()读取结果，进行判断。

在接收过程中，需要通过参数"string"、或"binary"，明确指明收到的是字符串，还是二进制数据块，便于虚拟机正确处理。
udprecvnet.udprecvRead("string")　参数"string"表示接收的是文本字符串，"binary"表示接收的是二进制字符串。
返回列表包含 4 项：第一项-1 表示接收失败，否则表示收到的数据长度；
第二项是是收到的数据，第三项是发信人地址，第四项是发信人端口号。

从上述函数的行为来分，可以分为两类：
1、即时方法：socket、timeout、close、clear、local_addr、bind、listen，这 7 个方法不需要等待时间，调用后立即生效。
2、耗时方法：connect、accept、send、recv、udpsend、udprecv 六个方法组。

2.1、connect 方法组使用举例：

```
net = NET()                                          # 定义 NET 对象 net
server_addr = "192.168.47.188"                       # 服务器地址
port = 5500                                          # 服务端口号
net = net.socket(net.IPv4,net.STREAM,net.TCP)        # 创建 socket,并赋值 net（IPv4、STREAM 数据流、TCP 协议）

net.connect(net.IPv4,"172.0.0.1",8080)               # 启动连接
where net.connectOK() != -1                          # 检测连接过程是否结束
{
    PowerDown(3)                                     # 低功耗设置
    # 可以在这里添加代码，利用等待期间执行其它工作
}

if net.connectRead() == -1          # 连接过程结束后，读取连接结果。
{
     print("连接失败！\n")
}
else
{
     print("连接成功！\n")
}
```

2.2、accept 方法组使用举例：

```
net = NET()                                          # 定义 NET 对象 net
server_addr = "192.168.47.188"                       # 服务器地址
port = 5500                                          # 服务端口号
net = net.socket(net.IPv4,net.STREAM,net.TCP)        # 创建 socket,并赋值 net（IPv4、STREAM 数据流、TCP 协议）
net.bind(net.IPv4,server_addr,port)                  # 将服务器：地址簇、地址和端口号与 net 绑定
net.timeout(3000)                                    # 设置超时时间 3000 毫秒
net.listen(10)                                       # 将 net 设置为监听状态，参数 10 是最大并发客户端数量

net.accept()                                         # 开始侦测过程，net.timeout()设置对 accept 有效
where net.acceptOK() != -1                           # 等待退出监听函数（返回-1 表示退出 accept 函数）
{
    PowerDown(3)                                     # 低功耗设置
    # 可以在这里添加代码，利用等待期间执行其它工作
}

x = net.acceptRead()              # 读取侦测结果。不论是否需要都必须读取结果，否则，将阻止再次调用 accept 方法。
if x[0] == -1                     # 表明 5*3000 毫秒时间内，没有侦测到客户端接入
{
     print("没有侦测到客户端接入！\n")
}
else
{
```

```
        print("侦测到客户端接入！ \n")
        print("客户端的地址是：%\n"，x[2])
    }
```

2.3、send 方法组使用举例：

```
    # 发送一次数据之前，先为本次发送定制一个 NET 对象，本次发送的其他操作都基于该定制对象，
     # 发送结束后，系统会自动回收该定制对象和对应的资源。
    sendnet = net.sendNet()                                  # 定制一个发送 net 对象
    sendnet .send("发送方法测试示例！ ",18)                  # 使用定制的 net 对象，启动发送过程
    where sendnet .sendOK() != -1                             # 等待本次发送线程结束，线程结束后，立即释放资源。
    {
        PowerDown(3)                                         # 低功耗设置
        # 可以在这里添加代码，利用等待期间执行其它工作
    }

    x = sendnet .sendRead()          # 读取发送结果，释放资源，不论是否需要结果，都必须读取，否则资源无法释放。
    if x == -1                        # 表明发送失败
    {
         print("本次发送失败！ \n")
    }
    else
    {
         print("本次发送的字节数是：%\n"，x)
    }
```

2.4、recv 方法组使用举例：

```
    # 在接收数据之前，先为本次接收定制一个 NET 对象，本次接收的其他操作都基于该定制对象，接收结束后，
     # 系统会自动回收该定制对象和对应的资源。
    recvnet = net.recvNet()                  # 定制 NET 对象
    recvnet.recv()                            # 启动接收过程
    where recvnet.recvOK() != -1             # 等待本次接收线程结束，结束后立即释放资源
    {
        PowerDown(3)                          # 低功耗设置
        # 可以在这里添加代码，利用等待期间执行其它工作
    }

    x = recvnet.recvRead("string")            # 读取接收结果
    if x[0] == -1                             # 表明 5*t 毫秒内未收到数据
    {
         print("大约 5*t 毫秒内未收到数据\n")
    }
    else
    {
         print("本次收到的数据包内容：%       %\n",x[0],x[1])          # x[0]是收到的数据字节数，x[1]是收到的数据
    }
```

2.5、udpsend 方法组使用举例：

```
    # 在 udp 发送之前，先为本次发送定制一个 NET 对象，本次 udp 发送的其他操作都基于该定制对象，发送结束后，
     # 系统会自动回收该定制对象和对应的资源。
    udpsendnet = net.udpsendNet()                    # 申请发送资源
    udpsendnet.udpsend(net.IPv4,"192.168.232.195",54321,buf,length)      # 启动发送过程
    where udpsendnet.udpsendOK() != -1               # 等待本次发送过程结束，如果发送结束，立即释放资源。
    {
        PowerDown(3)                                 # 低功耗设置
        # 可以在这里添加代码，利用等待期间执行其它工作
    }

    x = udpsendnet.udpsendRead()                      # 读取发送结果
    if x == -1                                        # 表明发送失败
    {
         print("本次 udp 发送失败！ \n")
    }
    else
    {
         print("本次 udp 发送的字节数是：%\n"，x)
    }
```

2.6、udprecv 方法组使用举例：

```
    udprecvnet = net.udprecvNet()                    # 定制 udp 接收 NET 对象
    udprecvnet.udprecv()                             # 启动接收过程
```

```
    where    udprecvnet.udprecvOK() != -1              # 等待本次接收过程结束，接收完成后，立即释放资源。
    {
        PowerDown(3)                                   # 低功耗设置
        # 可以在这里添加代码，利用等待期间执行其它工作
    }

    x =    udprecvnet.udprecvRead("string")      # 读取接收结果
    if x[0] == -1                                # 表明 5*t 毫秒内未收到数据
    {
        print("大约 5*t 毫秒内未收到数据\n")
    }
    else
    {
        print("   数据包内容：%\n"，x[0])          # 读取的列表中的第一项内容是收到的数据字节数
        print("   数据包内容：%\n"，x[1])          # 读取的列表中的第二项内容是收到的数据
        print("   发信人地址：%\n"，x[2])          # 读取的列表中的第三项内容是收到的发信人地址
        print("发信人端口号：%\n"，x[2])           # 读取的列表中的第四项内容是收到的发信人端口号
    }
```

注意事项：不论是 TCP 发送、或接收，还是 UDP 发送、或接收，单次发送和接收的字节数不能超过 1024 个字节，
如果发送的内容超出限制，可以切片发送，要确保每次发送内容不超过 1024 字节。

例程 1：TCP 文件传输程序——服务器端，与例程 2 构成一堆

```
count = 0                          # 记录在线人数
os = OS()                          # 定义操作系统对象
time = TIME()                      # 定义 TIME 对象 time

def serverFunc(list)               # 定义线程函数
{
    count = count + 1
    print("在线人数：%\n",count)
    clientnet    = list[0]                                    # 拆包线程参数，列表第一项是客户端的 NET 对象 clientnet
    clientaddr = list[1]                                      # 列表第二项是客户端的 IP 地址
    # 服务器监听到客户端接入后，给客户端发送信息响应
    buf = "请发送文件名,包含扩展名！"          # 响应信息内容

    sendnet = clientnet.sendNet()                  # 定制单次发送的 NET 对象
    sendnet.send(buf,len(buf))          # 基于定制 NET 对象，启动发送（发送长度不允许超过 1024 字节，超出部分丢弃）
    while sendnet.sendOK() != -1    {PowerDown(3)}      # 等待发送函数退出（-1 表示退出），
                                                        # 退出原因：有超时退出和发送完成退出
    xsend = sendnet.sendRead()                          # 读出发送结果
    if xsend == -1                                      # 超时
    {
        count = count - 1                               # 在线人数计数器 -1
        print("在线人数：%\n",count)
        clientnet.close()                               # 关闭 clientnet
        return                                          # 结束文件服务线程
    }

    # 服务器发送完应答信息后，转而等待客户端发送过来的文件名（包含扩展名），
    # 该等待时间不确定，如果 10 分钟内没有收到文件名，则，结束线程。
    timescount = 0          # 计数器用于控制服务器超时退出伺服。
    while True
    {
        recvnet = clientnet.recvNet()                   # 定制本次接收的 NET 对象
        recvnet.recv()                                  # 基于定制 NET 对象，启动接收
        while recvnet.recvOK() != -1    {PowerDown(3)}     # 等待接收函数退出(退出原因：超时、货收到信息)，释放资源。

        # 接收结果是一个列表，第一项是-1 表示接收超时，否则表示收到的数据包字节数，
        # 第二项是收到的数据（最大不超过 1024 字节）
        xrecv = recvnet.recvRead("string")                  # 读取接收结果，参数只能是"string"或"binary"
        if xrecv[0] == -1        # 在初始化时我们设置的 tiemout = 3 秒，系统自动重新连接 5 次，每次超时时间大约 15 秒
        {
            timescount = timescount + 1          # 每超时一次计数器（timescount）+1
            if timescount > 40               # 40*15 = 600 秒，即 10 分钟，如果如果 10 分钟，结束对当前客户端的伺服服务。
            {
                count = count - 1                       # 在线人数计数器 -1
                print("在线人数：%\n",count)
                clientnet.close()                       # 关闭 clientnet
                return                                  # 结束文件服务线程
```

```
            }
            continue                                    # 没有超过设置时间，继续接收
        }
        recvbuf = xrecv[1]                              # 获取接收到的数据
        break
    }

    # 解析收到的信息
    if recvbuf != "exit"                                # 如果收到的不是"exit"，应该是文件名
    {
        fp = FILEOPEN(recvbuf,"r")                      # 收到文件名后，服务器尝试打开文件
        if fp == False                                  # 如果文件打不开，通知客户端，然后结束线程
        {
            fp.close()
            buf = "打开文件失败，请检查文件名、或路径是否错误！"

            sendnet = clientnet.sendNet()               # 定制发送 NET 对象
            sendnet.send(buf,len(buf))                  # 启动发送，返回资源 ID 号
            while sendnet.sendOK() != -1   {PowerDown(3)}     # 等待发送函数退出（-1 表示退出），释放资源
            xsend = sendnet.sendRead()                  # 读取发送结果
            if xsend == -1                                      # 超时，网络有问题
            {
                count = count - 1                               # 在线人数计数器 -1
                print("在线人数：%\n",count)
                clientnet.close()                               # 关闭 clientnet
                return                                          # 结束文件服务线程
            }
        }
        else    # 正确打开文件，准备发送文件
        {
            list = fp.readlines()     # 将文件按行读出，打包入列表 list（如果读出的行字节数大于 1024，需要切片发送）
            fp.close()                                  # 关闭文件
            time.sleep(500)                             # 延迟（500ms)发送，避免客户端来不及处理，而丢失数据包
            for line in list                            # 遍历列表(文件的每一行)
            {
                sendnet = clientnet.sendNet()           # 定制发送 NET 对象
                sendnet.send(line,len(line))                    # 基于定制 NET 对象，启动发送
                while sendnet.sendOK() != -1   {PowerDown(3)}     # 检测发送函数直到退出，释放资源
                xsend = sendnet.sendRead()                      # 读取发送结果
                if xsend == -1                                  # 超时，结束服务。
                {
                    count = count - 1                           # 在线人数计数器 -1
                    print("在线人数：%\n",count)
                    clientnet.close()                           # 关闭 clientnet
                    return                                      # 结束文件服务线程
                }
                time.sleep(500)    # 延迟（500ms)发送，避免客户端来不及处理，而丢失数据包
            }

            # 至此，文件发送结束，再发送"exit"，通知客户端，文件全部发送完毕。
            sendnet = clientnet.sendNet()               # 定制发送 NET 对象
            sendnet.send("exit",4)                              # 基于定制对象，启动发送
            while sendnet.sendOK() != -1   {PowerDown(3)}       # 检测发送函数直到退出，释放资源
            xsend = sendnet.sendRead()                  # 读取发送结果
            if xsend == -1                                      # 超时，结束本次连接
            {
                count = count - 1                               # 在线人数计数器 -1
                print("在线人数：%\n",count)
                clientnet.close()                               # 关闭 clientnet
                return                                          # 结束文件服务线程
            }
        }
    }

    # 至此，要么是客户端主动下线了，要么是申请的文件全部发送结束了
    count = count - 1                       # 在线人数计数器 -1
    print("在线人数：%\n",count)
    clientnet.close()                       # 关闭 clientnet，定制的 NET 对象，系统会自动回收
    # 线程执行完毕，自动结束线程
}

server_addr = "192.168.111.195"             # 服务器地址
port = 5500                                 # 服务端口号
```

```
net = NET()                                          # 定义 NET 对象 net
net = net.socket(net.IPv4,net.STREAM,net.TCP)        # 创建 socket,并赋值 net（IPv4、STREAM 数据流、TCP 协议）
net.bind(net.IPv4,server_addr,port)                  # 将服务器：地址簇、地址和端口号与 net 绑定
net.timeout(3000)                                    # 设置超时时间 3000 毫秒
net.listen(10)                                       # 将 net 设置为监听状态，参数 10 是最大并发客户端数量
while True
{
    print("服务器伺服接收！\n")
    net.accept()                    # 开始监听，net.timeout()设置对 accept 有效
    while net.acceptOK() != -1 {PowerDown(3)}        # 等待退出监听函数（返回-1 表示退出 accept 函数）
    #读出的监测结果是一个列表，列表第一项是-1 表明超时，如果不是-1，表明监测到客户接入，
    #列表第二项是客户端的 net，第三项是客户端地址
    xaccept = net.acceptRead()                       # 读出结果是一个列表
    if xaccept[0] == -1 {continue}                   # 超时，继续监测
    list = [xaccept[1],xaccept[2]]                   # 读出监听到的客户端的 net 和客户端地址，打包成一个列表
    thread = THREAD()                                # 定义线程对象
    thread = thread.CreateThread(serverFunc,list)    # 创建子线程处理该连接，列表作为线程参数
    thread.start()                                   # 启动子线程
}
net.clear()   #清除网络资源



# 例程 2：TCP 文件传输程序——客户端，与例程 1 构成一对

gui = GUI("fine")                                    # 定义 FIne 窗口的 GUI 对象，用于图形界面显示
time = TIME()
#gui.HideConsoleWindow()             # 隐藏控制台窗口

serveraddr = "192.168.72.188"      # 服务器地址
port = 5500                                          # 该服务的端口号

net = NET()                                          # 定义一个 NET 对象
net = net.socket(net.IPv4,net.STREAM,net.TCP)        # 地址族:IPv4、数据流格式:STREAM、通讯协议:TCP
net.timeout(3000)                                    # 设置 net 的接收和发送超时时间，connect()不受 net.timeout()超时设置影响。

# 连接服务器
count = 0
box = GUI("box")                                     # 创建 MessageBox 窗口对象
box.MessageBox("开始连接……","确定")
while box.MessageBoxClosed() != -1
{

    net.connect(net.IPv4,serveraddr,port)            # 参数是服务器端的:地址族、网址、端口号
    while net.connectOK() != -1
    {
        PowerDown(3)
        count = count + 1
        box.SendMessageBox("连接中：" + itoc(count))      # 显示连接中提示
    }
    # 至此，表明以退出连接状态
    box.CloseMessageBox()                            # 关闭 box 窗口

    x = net.connectRead()                            # 读取连接的结果
    if x == -1                                       # 如果连接不成功，创建嵌套 MessageBox 提示。
    {
        box1 = GUI("box")                                # 创建 MessageBox 嵌套（二级 MessageBox 窗口）
        box1.MessageBox("未能连接到服务器，将退出程序！","确定")
        while box1.MessageBoxClosed() != -1 {PowerDown(3)}
        return                                       # 退出主线程（退出程序）
    }
}

# 连接到服务器后，等待服务器应答
count = 0                                            # 尝试接收次数
while True                                           # 进入接收循环
{
    recvnet = net.recvNet()                          # 定制单次接收 NET 对象
    recvnet.recv()                                       # 基于定制对象启动接收
    while recvnet.recvOK() != -1 {PowerDown(3)}      # 等待接收函数退出
    xrecv =recvnet.recvRead("string")                    # 读取结果，释放资源
    if xrecv[0] == -1                                    # 超时(大约 18 秒)
    {
        box = GUI("box")                             # 创建 MessageBox 窗口
```

```
            box.MessageBox("超时！未接收到服务器应答，将退出程序！","确定")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}
            return                                          # 退出主线程（退出程序）
        }
        recvbuf = xrecv[1]                      # 接收到服务器应答，读取数据，并释放接收资源
        break                                           # 退出接收循环
}

# 设计 Fine 窗口，输入要下载的文件名
title     = "TCP 文件传输——客户端"                          # 窗口标题
size      = [30,5,80,20]                                          # 窗口位置、尺寸

text      = ["服务器应答","text",12,3,62,1]               # 显示区域
edit      = ["文件名","edit","s",12,7,62,1]                # 录入文件名组件
button    = ["  确定  ","button",55,11,12,1]              # “确定”按钮

list      = [title,size,text,edit,button]                   # 打包窗口参数

gui.Fine(list)                                                  # 创建 Fine 窗口，并返回资源 ID 号
gui.SendText([recvbuf])                                 # 窗口创建后，稍微延迟一会再显示，否则显示不出来
filename = ""                                                   # 初始化文件名为空
while gui.FineClosed() != -1                    # 检测文件录入窗口是否关闭
{
    PowerDown(3)                                        # 降低数据录入期间的功耗
    # 等待输入待下载文件名
    if gui.FineReady() == -1 {continue}       # 检测是否有数据录入（0 有数据，-1 无数据）
    x = gui.FineRead()                              # 读取数据，第一项是指令(button-0)，第二项是文件名(对应 edit 组件)
    filename = x[1]                                     # 拆包，把文件名取出
    if filename == ""
    {
        # 弹窗提示错误
        box = GUI("box")                                # 创建 MessageBox 对象
        box.MessageBox("文件名不能为空！","确定")                   # 创建 MessageBox 窗口
        if box.MessageBoxClosed() != -1 {PowerDown(3)}    # 等待关闭窗口
        continue                                        # 跳转，等待下次数据录入
    }
    gui.CloseFine()                             # 获取文件名后，关闭文件名录入窗口
}

# 没有文件名，将退出程序。但是，由于已经建立了连接，退出程序前，需要通知服务器，
# 以便服务器及时关闭对应的服务
if filename == ""
{
    buf = "exit"                                        # 客户端退出指令
    sendnet = net.sendNet()                 # 定制单次发送的 NET 对象
    sendnet.send(buf,len(buf))              # 启动发送，并返回发送资源
    while sendnet.sendOK() != -1 {PowerDown(3)}     # 等待发送函数退出（返回-1 线程退出），退出后，立即释放资源。
    net.close()                                         # 清除网络资源
    return                                              # 退出程序
}

# 文件名录入成功后，向服务器发送文件名

sendnet = net.sendNet()                                 # 定制单次发送 NET 对象
sendnet.send(filename,len(filename))            # 启动发送，并返回发送资源
while sendnet.sendOK() != -1 {PowerDown(3)}     # 等待发送函数退出（返回-1 表示发送函数退出）
xsend = sendnet.sendRead()                              # 读发送结果，同时释放资源
if xsend == -1                                                  # 发送字节数为 0，表明发送错误
{
    box = GUI("box")                                    # 创建 MessageBox 窗口对象
    box.MessageBox("发送文件名超时，将退出程序！","确定")          # 弹出提示信息
    while box.MessageBoxClosed() != -1 {PowerDown(3)}       # 等待关闭 MessageBox 窗口
    net.close()                                             # 清除网络资源
    return                                                  # 程序返回，退出程序
}

recv_finished = 0                       # 文件接收完毕标志（0 未接收完，1 接收完毕）

# 设计 Fine 窗口，动态显示收到的数据
title           = "服务器下载文件接"
size            = [10,1,120,40]
menu      = ["文件","menu","保存"]
editbox = ["","editbox",0,0,116,36]
list            = [title,size,menu,editbox]
```

```
gui = GUI("fine")          # 创建 Fine 窗口对象（每次创建窗口都需要先创建窗口对象，一个窗口对象对应一个窗口）
gui.Fine(list)
while gui.FineClosed() != -1
{
    PowerDown(3)
    while not recv_finished                              # 如果文件未收完，持续循环直到文件接收完毕
    {
        # 冗余处理，检测窗口状态，如果用户中途关闭接收窗口，终止接收，关闭 net，退出程序
        if gui.FineClosed() == -1
        {
            net.close()                                  # 清理网络资源
            box = GUI("box")                        # 创建 MessageBox 窗口对象，弹窗提示。
            box.MessageBox("客户端退出接收程序！","确定")                    # 弹出提示信息
            while box.MessageBoxClosed() != -1 {PowerDown(3)}       # 等待关闭 MessageBox 窗口
            return
        }

        # 循环接收服务器端发送的数据包
        recvnet = net.recvNet()              # 每次接收数据包之前，需要定制一个接收 Net 对象，基于此对象操作
        recvnet.recv()                                           # 启动接收
        while recvnet.recvOK() != -1 {PowerDown(3)}         # 等待接收过程结束，并释放资源
        xrecv = recvnet.recvRead("string")                                      # 读取结果
        if xrecv[0] == -1                            # 超时次数超限
        {
            box = GUI("box")                        # 弹窗提示错误
            box.MessageBox("接收文件超时，将退出程序！","确定")          # 弹出提示信息
            while box.MessageBoxClosed() != -1 {PowerDown(3)}           # 等待关闭 MessageBox 窗口

            gui.CloseFine()                                      # 关闭 fine 窗口
            net.close()                                             # 关闭 net，清理网络资源
            return                                                  # 程序返回，退出程序
        }
        recvbuf = xrecv[1]                          # 如果接收正确，获取数据
        if recvbuf == "exit"                        # 收到的数据包为“exit”时，表明文件发送完毕、服务器已退出
        {
            recv_finished = 1                       # 置位数据接收完毕标志
            box = GUI("box")                        # 弹窗提示
            box.MessageBox("文件接收完毕！","确定")                                 # 弹出提示信息
            while box.MessageBoxClosed() != -1 {PowerDown(3)}       # 等待关闭 MessageBox 窗口
            break                                   # 退出发送循环
        }
        else        # 收到数据包
        {
            gui.SendEditbox([recvbuf])          # 将收到的数据显示在显示区
        }
    }

    # 数据接收完毕之前，程序无法流转到此处，按钮消息被阻滞（不响应按钮和菜单），之后才响应菜单或按钮
    if gui.FineReady() == 0                         # 有界面数据输出
    {
        x = gui.FineRead()                          # x[0]是指令（menu-0-0 来自 menu）,x[1]是文件内容（来自 editbox）
        # 检查同名文件是否存在
        fp = FILEOPEN(filename,"r")         # 尝试打开文件
        if fp == False                                  # 打开失败，表明文件不存在
        {
            is_writefile = 1                            # 置位写文件标志，可以写入文件标志
        }
        else                                                    # 如果能打开文件，说明文件存在，询问是否同意覆盖原文
        {
            fp.close()                                  # 关闭文件

            box = GUI("box")                            # 创建弹窗 GUI 对象
            box.MessageBox(filename+"已经存在，是否要覆盖原文件？","确定|取消")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}           # 等待做出选择
            select = gui.MessageBoxRead(box)                                        # 读取选择结果
            if select == 0                                      # 0 表明是确定
            {
                is_writefile = 1                                # 置位写文件标志，将写入文件
            }
            else                                                        # 表明选择的是取消、或直接关闭弹窗
            {
                is_writefile = 0                                # 清 0 写文件标志，将不会写文件
            }
        }
```

```
        if is_writefile == 1                                    # 如果写文件标志是 1，进行写文件操作（将覆盖原文件）
        {
            fp = FILEOPEN(filename,"w")          # 只写方式打开文件，
            fp.write(x[1])                                      # 将从 editbox 组件上读出的内容，写入文件
            fp.close()                                          # 关闭文件
        }
    }
}
net.close()   #清除网络资源



# 例程 3：TCP 单聊程序——服务器端，与例程 4 构成一堆

os = OS()
time = TIME()
max_clients = 10                    # 最大并发访问的客户端数量

list_friend = ["刘备","关羽","张飞","曹操","孙权"]      # 朋友圈

serveraddr = "192.168.111.195"                          # 服务器地址
port = 10000                                            # TCP 群聊程序的端口号
dict_name_net = {}                                      # 客户端网名——客户端 net 的对象关系（字典）
count = 0                                               # 聊天室在线人数

# 线程函数(为每一个接入聊天室的用户开辟一个子线程，伺服该用户)
def serverFunc(list)                                    # list 是向线程函数传递的列表参数
{
    clientnet    = list[0]                              # list[0]是客户端 socket
    clientaddr = list[1]                                # list[1]是客户端地址
    clientnet.timeout(3000)                             # 设置 clientnet 的超时时间为 3 秒

    count = count + 1                                   # 在线人数+1
    print("在线人数：%\n",count)                        # 打印在线人数

    buf = "all"+" "+"all"+" "+ "欢迎加入聊天室！ "      # 构造服务器应答信息

    # 发送应答信息
    sendnet = clientnet.sendNet()          # 生成发送 NET 对象，本次发送均基于新生成的对象操作
    sendnet.send(buf,len(buf))                                  # 启动发送
    while sendnet.sendOK() != -1 {PowerDown(3)}     # 等待发送过程结束，并释放资源
    x = clientnet.sendRead()                                    # 读取发送结果
    if x == -1                                                  # 发送失败
    {
      clientnet.close()                                         # 关闭 net
      return                                                    # 结束伺服线程
    }

    out_times = 0               # 超时次数计数器，用于控制：当某个用户连续 10 分钟不发言，将结束伺服线程

    # 服务器子线程伺服————接收和转发
    while True
    {
        recvnet = clientnet.recvNet()                   # 生成一个接收 NET 对象，负责本次接收
        recvnet.recv()                                  # 启动接收过程
        while recvnet.recvOK() != -1    {PowerDown(3)}      # 等待接收函数退出(如果超过大约 15 秒收不到，超时退出)
        x = recvnet.recvRead("string")                  # 读取接收结果（包含两个列表项）

        if x[0] == -1                                   # 接收失败，每收到一次接收失败，大约是 15 秒
        {
            out_times = out_times + 1
            if out_times >= 40              # 如果连续 40 次失败，大约 10 分钟，没有收到客户端信息，结束线程伺服
            {
              count = count -1
              print("在线人数：%\n",count)                  # 打印在线人数
              clientnet.close()                             # 关闭 net
              return                                                # 结束伺服线程
            }
            continue
        }
        out_times = 0               # 如果收到信息，超时计数器清 0
        # 接收成功，读取接收数据
        recvbuf = x[1]                                          # 将收到的数据赋值给 recvbuf
```

```
#   recvbuf 数据包结构：收信人+" "+发信人+" "+信息，即数据包分成三部分，
# 收信人、发信人、信息，其间用空格分割
# 从收到的数据 recvbuf 中提取信息
recvname = recvbuf.beforechr(" ")                # 获取发信人名字
recvbuf    = recvbuf.afterchr(" ")                # 获取数据包剩余部分
sendname = recvbuf.beforechr(" ")                # 获取发信人名字
recvbuf    = recvbuf.afterchr(" ")                # 获取数据包剩余部分，即，信息内容

# 检查发信人是否在朋友圈，在朋友圈执行下面程序
if list_friend.contains(sendname)                 # list_friend 列表记录的是朋友圈名字
{
    # dict_name_net 字典记录的是在线的网友的网名————net 对应词条
    if not dict_name_net.contains(sendname)       # 如果 dict_name_net 字典里没有，该网友将其添加到字典里面
    {
        dict_name_net.setitem(sendname,clientnet)       # 向在线字典中添加词条(sendname:clientnet)
    }

    # 检查收信人是否在朋友圈，如果不在朋友圈内，回复发信人“不在朋友圈内”，并停止对其服务
    if not list_friend.contains(recvname)
    {
        # 服务器组织回复信息：服务器向发信人反馈，收信人不是朋友圈内人，无法与收信人聊天。
        sendbuf = sendname+" "+"服务器"+" "+recvname+"不在朋友圈内!"

        sendnet = clientnet.sendNet()                                    # 定义本次发送所需的 NET 对象
        sendnet.send(sendbuf,len(sendbuf))                               # 启动发送
        while sendnet.sendOK() != -1 {PowerDown(3)}       # 等待发送过程结束，并释放资源
        x = sendnet.sendRead()                                           # 读取发送结果
        continue                                                                 # 跳转至伺服接收
    }

    # 检查在线字典中是否包含接收人，如果不包含，通知发信人，聊天对象不在线
    if not dict_name_net.contains(recvname)
    {
        # 服务器组织回复信息：服务器应答发信人，收信人不在线！
        sendbuf = sendname+" "+"服务器"+" "+recvname+"不在线上!"

        sendnet = clientnet.sendNet()                                    # 获取发送线程 ID
        sendnet.send(sendbuf,len(sendbuf))                               # 启动发送
        while sendnet.sendOK() != -1 {PowerDown(3)}          # 等待发送过程结束
        x = sendnet.sendRead()
        if x == -1                                                       # 发送失败
        {
          clientnet.close()                                              # 关闭 clientnet
          count = count -1
          print("在线人数：%\n",count)                                  # 打印在线人数
          return                                                                 # 结束线程
        }
        continue                                                         # 如果通知成功，跳转至伺服接收
    }

    # 如果收到的信息是“exit”，即发信人下线通知
    if recvbuf == "exit"                  # 如果收到"exit"，表明 clientname 离开聊天室
    {
        # 为调试方便，允许自己与自己聊天，也可以禁止
        if sendname != recvname           # 如果不是自己和自己聊天，才需要通知聊天对象，
                                          # 否则，无须通知（sendname 已退出 sendname）
        {
            # 服务器通知收信人，发信人已下线
            sendbuf = recvname+ " " + "服务器" + " " + sendname + "已离开聊天室！ "

            recvnet = dict_name_net.get(recvname)          # 从在线字典中，查出收信人 net

            sendnet = recvnet.sendNet()          # 使用收信人的 net，定制本次发送的 NET 对象
            sendnet.send(sendbuf,len(sendbuf))                           # 启动发送
            while sendnet.sendOK() != -1 {PowerDown(3)}          # 等待发送函数退出，释放资源
            x = sendnet.sendRead()                                       # 读取数据
        }

        n = dict_name_net.popitem(sendname)          # 聊条对象下线了，把在线字典中 sendname 词条删除
        # 无论发送成功与否，均结束线程
        count = count -1
        print("在线人数：%\n",count)                  # 打印在线人数
        return                                        # 结束子线程
```

```
                }
                else    # 如果收到的信息不是 "exit",则，转发信件
                {
                    sendbuf = recvname+ " " + sendname + " " + recvbuf          # 构造发送信息
                    recvnet = dict_name_net.get(recvname)                        # 查找收信人 net
                    # 转发送信息
                    sendnet = recvnet.sendNet()                                  # 定制一个属于收信人的 NET 对象
                    sendnet.send(sendbuf,len(sendbuf))                           # 启动发送
                    while sendnet.sendOK() != -1 {PowerDown(3)}            # 等待发送函数退出
                    x =   sendnet.sendRead()                                     # 读取数据，释放资源
                    if x == -1                      # 如果转发不成功（聊天对象下线了），关闭 net，结束线程
                    {
                        n = dict_name_net.popitem(recvname)            # 把在线字典中 recvname 词条删除
                        clientnet.close()                                       # 关闭 clientnet(线程参数)
                        count = count -1
                        print("在线人数：%\n",count)                          # 打印在线人数
                        return
                    }
                }
            }
            else        #登陆人不在朋友圈
            {
                # 向发信人发送"：不在朋友圈，没有权限参与聊天！"，结束线程
                sendbuf = sendname+ " " + "服务器" + " " + sendname + "：不在朋友圈，没有权限参与聊天！"
                sendnet = clientnet.sendNet()
                sendnet.send(sendbuf,len(sendbuf))                                  # 启动发送
                while sendnet.sendOK() != -1 {PowerDown(3)}         # 等待发送函数退出，释放资源
                x = sendnet.sendRead()                                      # 读取数据
                if x == -1                                                   # 网络不通，结束线程
                {
                  clientnet.close()                                          # 关闭 clientnet
                  count = count -1
                  print("在线人数：%\n",count)                          # 打印在线人数
                  return
                }

                # 向发信人发送"exit",强制结束线程
                sendbuf = sendname+ " " + sendname + " " + "exit"

                sendnet = clientnet.sendNet()
                sendnet.send(sendbuf,len(sendbuf))                  # 启动发送
                while sendnet.sendOK() != -1 {PowerDown(3)}         # 等待发送函数退出
                x = sendnet.sendRead()                                      # 读取数据，释放资源

                # 无论发送成功与否，均结束线程，因此不用判断是否超时
                clientnet.close()
                count = count - 1
                print("在线人数：%\n",count)
                return
            }
        }
}

# 主线程
net = NET()                                                  # 定义 NET 对象
net = net.socket(net.IPv4,net.STREAM,net.TCP)          # 给 NET 对象赋值（包含 socket）
net.bind(net.IPv4,serveraddr,port)                          # 将该 net 与地址簇、地址、端口号绑定
net.listen(max_clients)                                      # 将该 net 设置为监听模式，并且设置同时在线的最多人数

while True              # 主程序循环
{
    print("伺服——检测客户端连接请求\n")
    net.accept()                                             # 进入监听状态
    while net.acceptOK() != -1 {PowerDown(3)}          # 检测是否退出监听函数
    x = net.acceptRead()                                     # 读取侦测结果

    if x[0] == -1 {continue}                                 # 如果是超时退出、或其它错误退出，跳转之循环起点

    list = [x[1],x[2]]                  # 监听到新的客户端信息：list[0]是客户端的 net,list[1]是客户端地址

    thread = THREAD()                                       # 定义线程对象
    thread = thread.CreateThread(serverFunc,list)      # 创建子线程
    thread.start()                                           # 启动子线程
}
net.clear()   #清除网络资源
```

```
# 例程 4：TCP 单聊程序——客户端，与例程 3 构成一对

gui = GUI("fine")                                   # 定义 fine 窗口的 GUI 对象
time = TIME()
#gui.HideConsoleWindow()          # 隐藏控制台窗口
friend = "刘备,关羽,张飞,曹操,孙权,诸葛亮"                          # 用于下拉菜单

serveraddr = "192.168.111.195"
port = 10000

net = NET()                                                 # 定义一个 NET 对象
net = net.socket(net.IPv4,net.STREAM,net.TCP)   # 地址族:IPv4、数据流格式:STREAM、通讯协议:TCP
net.timeout(3000)                                           # 设置 net 的接收和发送超时时间

# 设计登录界面
title          = "TCP 单聊程序——客户端登录"       # 登录界面窗口标题
size           = [30,10,36,14]                                  # 窗口位置、大小
edit0      = ["  账号  ","edit","s",8,2,20,1]       # 账号录入编辑组件
edit1      = ["  密码  ","edit","s",8,5,20,1]       # 密码录入编辑组件，本例程没有处理密码功能，因此，密码可以随便填写
button = ["  确定  ","button",14,8,12,1]          # 按键组件（点击表明完成数据录入）
list           = [title,size,edit0,edit1,button]       # 参数打包

myname = ""
gui.Fine(list)                                      # 创建登录窗口
while gui.FineClosed() != -1                        # 检测登录窗口关闭消息
{
    PowerDown(3)                                    # 降功耗，参数 1、2、3，越大越省电
    if gui.FineReady() == -1 {continue}         # 检测是否有数据录入
    x = gui.FineRead()                              # 读取数据包

    # 为了演示程序简单化，这里没有验证账号密码
    command = x[0]
    myname     = x[1]                               # 拆包，取出录入的账号
    mima           = x[2]                           # 拆包，取出录入的密码（该程序没有做密码验证）
    if myname == ""
    {
        box = GUI("box")                            # 创建 MessageBox 窗口对象
        box.MessageBox("账号不能为空！","确定")
        while box.MessageBoxClosed() != -1 {PowerDown(3)}
        continue                                    # 继续等待输入账户、密码
    }
    # 成功登陆后，关闭登录窗口
    gui.CloseFine()                                 # 关闭输入窗口
}

# 如果没有输入账号密码就退出程序，myname 必然为空
if myname == ""                         # 如果登录名为空，退出程序
{
    net.close()                         # 关闭 net
    return                              # 退出程序
}

# 登陆成功后，尝试连接服务器
count = 0
box = GUI("box")                                    # 创建登录窗口 GUI 对象 box
box.MessageBox("正在连接服务器：","确定")
while box.MessageBoxClosed() != -1                  # 等待 MessageBox 的窗口关闭消息
{
     PowerDown(3)
    # 连接服务器
    net.connect(net.IPv4,serveraddr,port)           # 参数是服务器端的:地址族、网址、端口号
    while net.connectOK() != -1                     # 等待连接
    {
        # 等待连接期间，可以执行代码，显示连接进度
        box.SendMessageBox("正在连接服务器：" + itoc(count))    # 显示执行进度
        count = count + 1
        PowerDown(3)                                            # 低功耗设置
    }

    if net.connectRead() == -1                                  # 读取连接的结果，-1 表明超时
    {
```

```
        box.SendMessageBox("未能连接到服务器，将退出程序！")
        net.close()                                                    # 关闭 net
        time.sleep(2000)                                               # 显示 2 秒
        box.CloseMessageBox()                                          # 关闭 MessageBox 窗口
        return                                                         # 退出程序
    }
    box.CloseMessageBox()                                              # 连接成功 关闭 MessageBox 窗口
}

# 设计单聊窗口界面
title    = "TCP 单聊程序——客户端"                                      # 连天界面窗口标题
size     = [30,5,80,30]                                                # 聊天窗口位置、尺寸
combobox = ["选择聊天对象","combobox",friend,14,1,20,1]                # 列表第三项是 listbox 项目字符串
textbox  = ["聊天内容","textbox","N",10,3,66,15]                       # "N"表示不响应双击 textbox 区域事件
edit     = ["发送内容","edit","s",10,19,64,3]                          # 录入发送内容的编辑组件
button   = ["发送","button",60,24,12,1]                                # 发送按钮
list     = [title,size,combobox,textbox,edit,button]                   # 参数打包


# 开启聊天室
gui = GUI("fine")                                                      # 创建 fine 窗口对象
gui.Fine(list)                                                         # 创建聊天窗口

recvnet = net.recvNet()                                                # 创建 NET 接收对象
recvnet.recv()                                                         # 启动接收过程，服务器进入接收伺服状态

while gui.FineClosed() != -1                                           # 检测聊天窗口是否关闭
{
    PowerDown(3)                                                       # 节能设置

    if recvnet.recvOK() == -1                                          # 等待接收过程结束，释放资源
    {
        # 至此，表明本次接收过程结束
        x = recvnet.recvRead("string")                                 # 读取接收结果
        if x[0] > 0                                                    # 如果收到数据，处理、显示数据
        {
            recvdata = x[1]                                            # 读取收到的内容后
            # 从收到的数据中，提取信息
            recv_name = recvdata.beforechr(" ")                        # 从信息内容中提取收信人网名
            recvdata = recvdata.afterchr(" ")                          # 将余下部分重新赋值 temp
            send_name = recvdata.beforechr(" ")                        # 提取发信人网名
            recvdata = recvdata.afterchr(" ")                          # 提取信件内容

            # 如果信件内容是"exit"，退出循环，结束聊天程序
            if recvdata == "exit"
            {
                str = send_name + "退出聊天室！"
                gui.SendTextbox([str])                                 # 显示发言人名字：和发言内容
                time.sleep(2000)                                       # 延迟 2 秒，关闭窗口，并退出程序
                gui.CloseFine()                                        # 关闭聊天窗口
                return                                                 # 退出程序
            }

            if send_name == "all"                                      # 是服务器响应信息
            {
                str = "服务器：" + recvdata+"\n"                       # 打包显示内容
            }
            else                                                       # 是聊天对象发过来的信息，前面加上发言人姓名
            {
                str = send_name + "：" + recvdata+"\n"                 # 收到的内容之前，加上发言人姓名
            }
            gui.SendTextbox([str])                                     # 显示发信人名字+发言内容
        }
        # 处理完上次接收到的信息后，立即创建接收 NET 对象，进入接收伺服状态
        recvnet = net.recvNet()                                        # 创建 NET 接收对象
        recvnet.recv()                                                 # 启动接收过程，服务器进入接收伺服状态
    }

    # 至此，表明：要么还处于接收中，要么已收到、并处理完数据包
    # 检查 Fine 窗口是否有数据录入
    if gui.FineReady() == 0                                            # 检查是否有数据录入
    {
        # 如果有数据录入，读取录入数据
        x = gui.FineRead()                                             # 读取录入数据包
```

```
            gui.SendEdit([""])                                          # 清除输入框内容，以备用户再次输入
            # 解析录入的数据包
            command       = x[0]                                        # 使用数据包的指令
            input_data    = x[1]                                        # 发言内容（来自 edit 组件）
            recv_name     = x[2]                                        # 聊天对象的名字（来自 combobox 组件）
            buf = recv_name + " " + myname + " " + input_data           # 构造发送数据包，它们之间用空格隔开
            gui.SendTextbox(["我："+input_data+"\n"])                    # 显示发言人名字：和发言内容

            # 发送发言内容
            sendnet = net.sendNet()                                     # 定制发送 NET 对象
            sendnet.send(buf,len(buf))                                  # 启动发送过程
            while sendnet.sendOK() != -1 {PowerDown(3)}                 # 检查发送过程是否结束，释放资源
            x = sendnet.sendRead()                                                  # 读取发送结果
            if x == -1                                                              # 如果发送超时，提示并退出程序
            {
                box = GUI("box")
                box.MessageBox("发送超时！请检查网络连接状态！","确定")
                while box.MessageBoxClosed() != -1 {PowerDown(3)}

                gui.CloseFine()                                                     # 关闭 Input 窗口
                net.close()                                                         # 关闭 net
                return                                                              # 退出程序
            }
        }
}

# 当客户端主动关闭 Fine 窗口，结束聊天时，应通知服务器！
buf = recv_name + " " + myname + " " + "exit"                           # 构造发送数据包，它们之间用空格隔开
sendnet = net.sendNet()                                                 # 申请发送资源
sendnet.send(buf,len(buf))                                              # 启动发送过程
# 检查发送过程是否结束（最多尝试 5 次，每次超时时间由 net.timeout(t)设置）
while sendnet.sendOK() != -1 {PowerDown(3)}                             # 检查发送过程是否结束，释放资源
x = sendnet.sendRead()                                                  # 读取发送结果

net.close()                                                             # 关闭 net



# 例程 5：UDP 群聊程序——服务器端，与例程 6 构成一对

net = NET()                                                             # 定义一个 NET 对象
port = 4321

net = net.socket(net.IPv4,net.DGRAM,net.UDP)   # 地址族:IPv4、数据流格式:DGRAM、通讯协议:UDP
# 把服务器地址与上述 socket 绑定
net.bind(net.IPv4,"192.168.111.195",port)          # 参数分别是服务器的：地址族、网址、端口号
net.timeout(3000)

dict_addr_name = {}             # 定义一个客户端地址——游客名字的空字典
number = 300                    # 游客编号
count    = 0                    # 在线人数

print("服务器伺服接收！\n")
while True
{
    recvnet = net.udprecvNet()                                          # 创建一个 udp 接收 NET 对象
    recvnet.udprecv()                                                   # 服务器端等待接收来自客户端的请求
    while recvnet.udprecvOK() != -1 {PowerDown(3)}                      # 等待 udprecvOK 退出，释放资源
    x = recvnet.udprecvRead("string")                                   # 读取数据
    if x[0] == -1 {continue}                                            # 第一项为-1 表明超时，继续接收
    RecvDatas = x[1]                                                    # 收到的数据
    remote_addr = x[2]                                                  # 发送端的地址

    # 新进游客处理
   if   not dict_addr_name.contains(remote_addr)                        # 如果字典里包含该地址，返回 1，否则返回 0
    {
        count = count + 1
        print("在线人数：%\n",count)

        number = number + 1
        name = "游客" + itoc(number)
        dict_addr_name.setitem(remote_addr,name)                        # 将新进游客添加入字典中
```

```
            # 群发欢迎词
            databack = "欢迎" + name + "进入聊天室！"     # 构造发送内容
            for k,v in dict_addr_name.items()                  #   k 是客户端地址，v 是游客名字
            {
                sendnet = net.udpsendNet()                                  # 创建 udp 发送 NET 对象
                sendnet.udpsend(net.IPv4,k,port,databack,len(databack))     # 发送信息时，客户端地址，端绑定的端口号
                while sendnet.udpsendOK() != -1 {PowerDown(3)}              # 等待 udpsend 返回,，释放资源
                xsend = sendnet.udpsendRead()                               # 读取数据
            }

        }

        # 在线游客处理
        if RecvDatas == "exit"     # 如果客户端请求退出
        {
            v = dict_addr_name.get(remote_addr)                     # 查找退出游客的名字
            dict_addr_name.popitem(remote_addr)                     # 将游客 v 从字典中删除
            count = count - 1
            print("在线人数：%\n",count)

            # 群发离线通知
            databack = v + "离开聊天室！"                           # 构造游客离线内容
            for k,v in dict_addr_name.items()                   # k 是客户端地址，v 是游客名字
            {
                sendnet = net.udpsendNet()                      # 创建 udp 发送 NET 对象
                sendnet.udpsend(net.IPv4,k,port,databack,len(databack))     # 发送信息时，客户端地址，端口号
                while sendnet.udpsendOK() != -1 {PowerDown(3)}              # 等待 udpsend 返回，释放资源
                xsend = sendnet.udpsendRead()                               # 读取数据
            }

        }
       else
        {
            # 构造群发字符串(服务器转发)
            v = dict_addr_name.get(remote_addr)                     # 查找该游客名字
            databack = v + "：" + RecvDatas                         # 构造游客发言内容

            # 群发信息
            for k,v in dict_addr_name.items()                       # k 是客户端地址，v 是游客名字
            {
                sendnet = net.udpsendNet()
                sendnet.udpsend(net.IPv4,k,port,databack,len(databack))     # 发送信息时，客户端地址，端口号
                while sendnet.udpsendOK() != -1 {PowerDown(3)}              # 等待 udpsend 返回
                xsend = sendnet.udpsendRead()                               # 读取数据，释放资源
            }
        }

}
net.close()  # 清除网络资源



# 例程 6：UDP 群聊程序——客户端，与例程 5 构成一对

gui = GUI("fine")                               # 定义 GUI 对象，用于图形界面显示
#gui.HideConsoleWindow()        # 隐藏控制台窗口

net = NET()                                     # 定义一个 NET 对象

localaddr = net.local_addr()            # 自动获取本地的 IP 地址
port = 4321                                     # 端口号

net = net.socket(net.IPv4,net.DGRAM,net.UDP)  # 给 net 赋值（地址族、数据流格式、UDP 传输）
net.bind(net.IPv4,localaddr,port)               # 将 net 与本地地址"192.168.47.195"和端口 4321 绑定
net.timeout(3000)                                       # 设置 net 的接收和发送超时时间

server_addr = "192.168.111.195"                 # 服务器地址，


# 设计群聊窗口界面
title     = "UDP 聊天程序——客户端"
size       = [30,5,80,30]
testbox = ["群聊内容","textbox","N",10,1,66,17]
edit       = ["发送内容","edit","s",10,19,64,3]
```

```
button    = ["发送","button",60,24,12,1]
list      = [title,size,testbox,edit,button]

gui.Fine(list)                          # 创建聊天 Input 窗口，返回资源 ID

recvnet = net.udprecvNet()        # 创建 udp 接收 NET 对象
recvnet.udprecv()                   # 启动接收服务，接收长度限制在 1024 字节以内

while gui.FineClosed() != -1            # 检测聊天窗口是否关闭
{
    PowerDown(3)                            # 节能设置

    if recvnet.udprecvOK() == -1      # udprecv 返回-1，表明退出 udprecv 函数，释放资源
    {
        xrecv = recvnet.udprecvRead("string")      # 读取接收结果
        if xrecv[0] != -1                   # 如果超时返回-1，否则，返回接收到的字节数
        {
            recvdada = xrecv[1]               # 接收数据
            str = recvdada+"\n"
            gui.SendTextbox([str])          # 显示收到的内容
        }

        # 处理完上次收到的数据后，再次进入伺服接收状态
        recvnet = net.udprecvNet()          # 创建 udp 接收 NET 对象
        recvnet.udprecv()                     # 启动接收服务，接收长度限制在 1024 字节以内
    }

    if gui.FineReady() == 0                         # Fine 窗口有输出
    {
        xfine = gui.FineRead()                      # 读取数据包（包含两项数据）
        command = xfine[0]                          # 使用输出数据的指令
        buf       = xfine[1]                                # 仅仅发送内容，服务器转发时，自动加上游客名。

        gui.SendEdit([""])                              # 清除输入框内容

        if len(buf) == 0 {continue}                 # 如果录入数据为空，跳转回 Fine 窗口检测

        sendnet = net.udpsendNet()                                                      # 分配发送资源
        sendnet.udpsend(net.IPv4,server_addr,port,buf,len(buf))       # 启动发送，地址是服务器地址
        while sendnet.udpsendOK() == -1
        {
            xsend = sendnet.udpsendRead()           # 读取发送结果，释放资源
            if xsend == -1                          # 发送失败
            {
                net.close()                         # 关闭 net
                gui.CloseFine()                     # 关闭 Fine 窗口
                return                              # 退出程序
            }
            # 发送后不用直接显示，服务器会向所有在线用户分(包含自己)发该信息，等收到信息后，再显示
        }
        if buf == "exit"
        {
            net.close()             # 关闭 net
            gui.CloseFine()         # 关闭 Fine 窗口
            return                  # 退出程序
        }
    }
}

# 没有发送 exit 直接关闭聊天窗口时，需要在这里自动向服务器发送"exit"
buf = "exit"
sendnet = net.udpsendNet()                                                      # 分配发送资源
sendnet.udpsend(net.IPv4,server_addr,port,buf,len(buf))       # 地址是服务器地址
while sendnet.udpsendOK() != -1 {PowerDown(3)}                  # 检查 udpsend 是否返回
xsend = sendnet.udpsendRead()                                                   # 读取发送结果，释放资源

net.close()   #清除网络资源



# 例程 7：UDP 单聊程序——服务器端，与例程 8 构成一对

# 服务器程序没有自动程序退出机制，如果要停止服务，需要手动操作，关闭该程序
```

```
# 该聊天程序的数据包格式：收信人网名+" "+发信人网名+" "+信件内容，其间使用单个空格隔开
# 收信人网名、发信人网名内部不允许使用空格


net = NET()                              # 定义一个 NET 对象
server_addr = "192.168.111.195"    # 服务器地址
port = 4321                              # 服务端口号

net = net.socket(net.IPv4,net.DGRAM,net.UDP)   # 给 net 复制：地址族:IPv4、数据流格式:DGRAM、通讯协议:UDP
net.bind(net.IPv4,server_addr,port)              # 把服务器地址与 net 绑定.参数分别是服务器的：地址族、网址、端口号
net.timeout(3000)                                    # 设置网络收发超时时间，如果不设置默认一直等待

name_list = ["刘备","关羽","张飞","曹操","孙权"]    # 聊天“朋友圈”，不在该“圈子”的人将不能参与聊天
dict_addr_name = {}       # 定义空字典：用于保存“网址-网名”对应关系(同一个网名在不同的地方上网，会有不同的网址)
dict_name_addr = {}       # 定义空字典：用于保存“网名-网址”对应关系
count   = 0                  # 记录并显示服务器服务的次数

print("服务器伺服状态……\n")
while True                          # 伺服接收
{
    PowerDown(3)
    recvnet = net.udprecvNet()      # 创建 udp 接收 NET 对象
    recvnet.udprecv()                 # 服务器端等待接收来自客户端的信息
    while recvnet.udprecvOK() != -1 {PowerDown(3)}      # 循环等待，推出循环的条件有：收到信息、或超时，释放资源
    x = recvnet.udprecvRead("string")                                                  # 读取接收结果(包含 4 项内容)
    if x[0] == -1 {continue}                 # 超时后，跳转到伺服状态
    temp          = x[1]                     # 接收到的数据包（包含发信人名字、收信人名字、信息内容）
    send_addr = x[2]                         # 发信人地址
    send_port = x[3]                         # 发信端口号

    recv_name = temp.beforechr(" ")        # 拆包，获取收件人网名（取空格前面部分）
    if name_list.contains(recv_name)       # 查看收件人是否在“朋友圈”内，如果在“朋友圈”内，进行下面处理
    {
        # 如果收信人在朋友圈内，继续提取发信人名字，所发数据
        temp = temp.afterchr(" ")            # 继续拆包，获取第一个空格之后的内容，再次赋值给 temp
        send_name = temp.beforechr(" ")      # 继续拆包，获取发信人网名
        RecvDatas = temp.afterchr(" ")       # 继续拆包，获取信件内容

        if dict_addr_name.contains(send_addr)    # 查看 dict_addr_name 字典中 send_addr（发件人地址）是否存在
        {
            temp_name = dict_addr_name.get(send_addr)    # 如果 send_addr（发件人地址）存在，找出该地址对应的网名
            dict_name_addr.popitem(temp_name)    # 删除另一个子字典 dict_name_addr 中，名字等于 temp_name 的词条
        }

        dict_addr_name.setitem(send_addr,send_name)       # 更新、或添加词条（可以适应同一个网址用不同的网名登录）
        dict_name_addr.setitem(send_name,send_addr)       # 添加词条（可以适应同一个网址用不同的网名登录）

        if RecvDatas == "exit"                            # 表明 send_name 准备退网
        {
            # 回复发信人
            databack = send_name + " " + send_name + " " + RecvDatas    # 构造转发信息包，回复发信人
            recv_addr = dict_name_addr.get(send_name)                   # 由发信人网名查出其网址

            sendnet = net.udpsendNet()                                          # 创建 udp 发送对象
            sendnet.udpsend(net.IPv4,recv_addr,port,databack,len(databack))     # 回复发信人
            while sendnet.udpsendOK() != -1 {PowerDown(3)}                      # 等待发送过程结束，释放资源
            x = sendnet.udpsendRead()                                           # 读取发送结果

            # dict_name_addr 字典中包含 recv_name，说明其在线，将信息转发给收信人
            if dict_name_addr.contains(recv_name)
            {
                # 构造转发信息包，通知收信人，聊天对象已离开
                databack = recv_name+" "+"服务器"+" "+send_name+"已离线！"
                recv_addr = dict_name_addr.get(recv_name)                       # 由发信人网名查出其网址

                sendnet = net.udpsendNet()                                          # 创建 udp 发送 NET 对象
                sendnet.udpsend(net.IPv4,recv_addr,port,databack,len(databack))     # 回复发信人
                while sendnet.udpsendOK() != -1 {PowerDown(3)}                      # 等待 udpsend 发送结束
                x = sendnet.udpsendRead()                                           # 读取发送字节数，释放资源
            }
            n = dict_addr_name.popitem(send_addr)       # 删除发信地址对应的词条
            n = dict_name_addr.popitem(send_name)       # 删除发信人对应的词条
            count = count + 1                           # 服务次数
            print("转发次数：%\n",count)
```

```
        }
        else    # 不是退网请求，即，正常聊天信息，向收信人转发信息
        {
            # dict_name_addr 字典中包含 recv_name，说明其在线，将信息转发给收信人
            if dict_name_addr.contains(recv_name)
            {
                databack = recv_name + " " + send_name + " " + RecvDatas    # 构造转发信息包
                recv_addr = dict_name_addr.get(recv_name)                    # 由收信人网名查出其网址

                sendnet = net.udpsendNet()                                   # 创建 udp 发送 NET 对象

                # 发送信息时，使用收信人地址，和客户端绑定的端口号
                sendnet.udpsend(net.IPv4,recv_addr,port,databack,len(databack))
                while sendnet.udpsendOK() != -1 {PowerDown(3)}               # 等待 udpsend 结束，释放资源
                x = sendnet.udpsendRead()                                    # 必须读取字节数
            }
            else        # dict_name_addr 字典中没有网名为 recv_name 时，说明其不在线，通知发信人
            {
                databack = send_name + " " + "服务器" + " " + recv_name + "不在线上"      # 构造回复信息
                recv_addr = dict_name_addr.get(send_name)                    # 由发信人网名查找发信人地址

                sendnet = net.udpsendNet()                                   # 创建 udp 发送 NET 对象

                # 发送信息时，要使用客户端地址，和客户端绑定的端口号
                sendnet.udpsend(net.IPv4,recv_addr,port,databack,len(databack))
                while sendnet.udpsendOK() != -1 {PowerDown(3)}          # 等待 udpsend 返回
                x = sendnet.udpsendRead()
            }
            count = count + 1                                                # 服务次数
            print("转发次数：%\n",count)
        }
    }
    else     # 如果该网名不在列表中，说明该网名没有权限（非朋友），通知发信人
    {
        databack = send_name + " " + "服务器" + " " + recv_name + "没有权限！"    # 构造回复信息
        recv_addr = dict_name_addr.get(send_name)                        # 由发信人网名查找发信人地址，回复发信人

        sendnet = net.udpsendNet()                                       # 创建 udp 发送 NET 对象

        # 发送信息时，要使用客户端地址，和客户端绑定的端口号
        sendnet.udpsend(net.IPv4,recv_addr,port,databack,len(databack))
        while sendnet.udpsendOK() != -1 {PowerDown(3)}                   # 等待 udpsend 返回
        x = sendnet.udpsendRead()
    }
}
net.close()  # 清除网络资源



# 例程 8：UDP 单聊程序——客户端，与例程 7 构成一对

# 该客户端程序的退出有两条途径：
#       1、发送"exit"信息，通知服务器，服务器通知收信人（发信人已退出），同时，回复发信人（回复信息仍为"exit"），
#          清除客户端在线记录，客户端收到回复后，立即退出聊天程序；
#       2、客户端可以直接关闭聊天窗口，退出聊天程序。这种情况下，服务器无法立即知道该客户端是否脱网。

# 发送数据包格式：收信人网名 +" "+ 发信人网名 +" "+ 信息内容
# 发信人网名、收信人网名可以是数字、字符组成的字符串（中间不允许有空格）
# 发信人网名、发信人网名和信息内容之间用单个空格隔开

gui = GUI("fine")                                       # 定义 fine 窗口的 GUI 对象
#gui.HideConsoleWindow()                                # 隐藏控制台窗口
friend = "刘备,关羽,张飞,曹操,孙权,诸葛亮"

net = NET()                                             # 定义一个 NET 对象

serveraddr = "192.168.111.195"                          # 服务器地址，
localaddr = net.local_addr()                            # 自动获取本地地址
port = 4321

net = net.socket(net.IPv4,net.DGRAM,net.UDP)    # 给 net 赋值（地址族、数据流格式、UDP 传输）
net.bind(net.IPv4,localaddr,port)               # 将 net 与地址"192.168.47.195"和端口 4321 绑定

net.timeout(3000)      # udp 只有操作系统内核缓冲区不足，才会出现超时（多次发送，由于没有网络，导致数据阻塞）
```

```
# 设计登录界面
title = "UDP 单聊程序——客户端"                # 登录界面窗口标题
size    = [30,10,34,14]                                  # 窗口位置、大小
list0 = ["  账号  ","edit","s",8,2,20,1]              # 账号录入编辑组件
list1 = ["  密码  ","edit","s",8,5,20,1]              # 密码录入编辑组件
list2 = ["  确定  ","button",14,8,12,1]              # 按键组件（点击表明完成数据录入）
list    = [title,size,list0,list1,list2]              # 参数打包

myname = ""
gui.Fine(list)                                        # 创建登录窗口，返回资源 ID 号
while gui.FineClosed() != -1                          # 检测登录窗口关闭消息
{
    PowerDown(3)                                      # 降功耗，参数 1、2、3，越大越省电
    if gui.FineReady() == -1 {continue}               # 检测是否有数据录入
    x = gui.FineRead()                                # 读取数据
    command = x[0]
    myname    = x[1]                                     # 拆包取出录入的账号
    mima            = x[2]                               # 拆包取出录入的密码（该程序没有做密码验证）
    if myname == ""
    {
        box = GUI("box")
        box.MessageBox("登录名不能为空！","确定")              # 创建 MessageBox 窗口
        while box.MessageBoxClosed() != -1 {PowerDown(3)}     # 查询 MessageBox 窗口关闭消息
        continue                                      # MessageBox 窗口关闭后，跳转到 Input 窗口循环
    }
    gui.CloseFine()                                   # 获取登录信息后，关闭输入窗口
}

if myname == ""
{
    box = GUI("box")
    box.MessageBox("登录名为空，将退出程序！","确定")      # 创建 MessageBox 窗口，返回资源 ID 号
    while box.MessageBoxClosed() != -1 {PowerDown(3)}     # 检测 MessageBox 窗口关闭消息
    return                                                # 退出程序
}


# 设计单聊窗口界面
title      = "UDP 单聊程序——客户端"                                          # 连天界面窗口标题
size       = [30,5,80,32]                                                  # 聊天窗口位置、尺寸
combobox = ["选择聊天对象","combobox",friend,14,1,20,1]          # 选择聊天对象组件
textbox   = ["聊天内容|新宋体","textbox","N",10,3,66,15]                                  # 显示聊天内容组件
edit       = ["发送内容","edit","s",10,19,64,4]                                  # 录入发送内容的编辑组件
button    = ["发送","button",60,25,12,1]                                    # 发送按钮
list       = [title,size,combobox,textbox,edit,button]                     # 参数打包

recv_name = ""
is_exit    = 0                                    # 退网标志

# 创建 Fine 窗口的 GUI 对象
gui = GUI("fine")
gui.Fine(list)                              # 创建聊天窗口

udpnet = net.udprecvNet()                   # 创建接收 NET 对象
udpnet.udprecv()                            # 启动伺服接收

while gui.FineClosed() != -1                # 检测聊天窗口是否关闭
{
    PowerDown(3)                                  # 节能设置

    if udpnet.udprecvOK() == -1           # 检测本次接收过程是否结束，释放资源
    {
        x = udpnet.udprecvRead("string")              # 读出接收结果（包含四项内容）
        if x[0] != -1                               # 如果收到数据（-1 表明接收出错）
        {
            # 拆解数据包
            temp              = x[1]
            send_addr = x[2]
            send_port    = x[3]
            # 解析数据包
            recv_name = temp.beforechr(" ")                 # 从信息内容中提取收信人网名（只能是 myname）
            temp = temp.afterchr(" ")                             # 将余下部分重新赋值 temp
            send_name = temp.beforechr(" ")                 # 提取发信人网名(聊天对象)
            recvdada = temp.afterchr(" ")                       # 提取信件内容
            if recvdada == "exit"
```

```
            {
                recvdada = "已经离线，再见！"
            }
            # 显示收到的信息
            str = send_name + "：" + recvdada+"\n"          # 聊天对象的发言：打包显示内容
            gui.SendTextbox([str])        # 显示发送数据，并不意味着对方一定能收到（网络断开时）
        }
        # 处理完上一次接收到的信息后，立即设置继续伺服接收
        udpnet = net.udprecvNet()                  # 创建接收 NET 对象
        udpnet.udprecv()                           # 启动伺服接收
    }

    # 检查 Fine 窗口是否有数据录入
    if gui.FineReady() == 0                        # 表明 Fine 窗口有数据录入
    {
        x = gui.FineRead()                         # 读取数据包(包含三项)
        command       = x[0]                       # 使用录入数据的指令
        input_data    = x[1]                       # 对应 edit 组件上的数据
        recv_name     = x[2]                       # 对应 combobox 组件上的数据

        gui.SendEdit([""])                         # 清除 edit 组件上的内容

        # send_name 是收到的聊天对象的名字，就是我要发送的对象
        buf = recv_name + " " + myname + " " + input_data   # 构造发送数据包，它们之间用空格隔开

        sendnet = net.udpsendNet()                 # 创建发送对象
        sendnet.udpsend(net.IPv4,serveraddr,port,buf,len(buf))   # 启动 udpsend 发送
        while sendnet.udpsendOK() != -1 {PowerDown(3)}         # 检查 udpsend 是否退出（函数退出返回-1），释放资源
        x = sendnet.udpsendRead()                  # 读取结果
        # 发送结束后，检查一下，发送内容
        if input_data == "exit"                    # 如果录入的内容是"exit"，该信息发送给服务器后，自动退出程序
        {
            is_exit = 1                            # 置位退网标志
            net.close()
            gui.CloseFine()                        # 关闭聊天窗口
            return                                 # 退出程序
        }

        # 如果不是退网指令，但是，发送失败
        if x == -1                                 # 超时，聊天对象退网
        {
            box = GUI("box")
            box.MessageBox("发送失败！请检查网络！","确定")          # 创建 MessageBox 窗口
            while gui.MessageBoxClosed() != -1 {PowerDown(3)}          # 检查 MessageBox 窗口关闭消息
            continue                   # 可以回到 Fine 界面重新选择聊天对象继续聊天
        }
        str = "我：" + input_data + "\n"
        gui.SendTextbox([str])        # 显示发送数据，并不意味着对方一定能收到（网络断开时）
    }
}

#可能存在游客直接关闭了 Fine 窗口，退出前没有发送"exit"的情况下，需要自动向服务器发送"exit",通知服务器"人已离开"
if is_exit == 0                            # 没有发送 exit 指令的标志
{
    if recv_name == ""
    {
        recv_name = myname
    }
    buf = recv_name + " " + myname + " " + "exit"          # 构造发送数据包，它们之间用空格隔开
    sendnet = net.udpsendNet()                                                  # 创建发送 NET 对象
    sendnet.udpsend(net.IPv4,serveraddr,port,buf,len(buf))       # 服务器地址、端口、数据包
    while sendnet.udpsendOK() != -1 {PowerDown(3)}          # 检查 udpsend 是否返回，释放资源
    x = sendnet.udpsendRead()
}
net.close()   #清除网络资源



# 例程 9：TCP 文件传输程序——服务器端，与例程 10 构成一对
# 客户端申请下载 finecomp.exe 二进制文件，服务器端读出该文件的前 300 字节内容，发送给客户端

count = 0                                  # 记录在线人数
os = OS()                                  # 定义操作系统对象
time = TIME()                              # 定义 TIME 对象 time
```

```
def serverFunc(list)                    # 定义线程函数
{
    count = count + 1
    print("在线人数：%\n",count)
    clientnet   = list[0]                          # 拆包线程参数，列表第一项是客户端的 NET 对象 clientnet
    clientaddr = list[1]                           # 列表第二项是客户端的 IP 地址
    # 服务器监听到客户端接入后，给客户端发送信息响应
    buf = "请发送文件名,包含扩展名！"       # 响应信息内容

    sendnet = clientnet.sendNet()                  # 定制单次发送的 NET 对象
    sendnet.send(buf,len(buf))          # 基于定制 NET 对象，启动发送（发送长度不允许超过 1024 字节）
    while sendnet.sendOK() != -1   {PowerDown(3)}    # 等待发送函数退出（-1 表示退出）
    xsend = sendnet.sendRead()                     # 读出发送结果
    if xsend == -1                                 # 超时
    {
        count = count - 1                          # 在线人数计数器 -1
        print("在线人数：%\n",count)
        clientnet.close()                          # 关闭 clientnet
        return                                     # 结束文件服务线程
    }

    # 服务器发送完应答信息后，转而等待客户端发送过来的文件名（包含扩展名），
    # 该等待时间不确定，如果 10 分钟内没有收到文件名，则，结束线程。
    timescount = 0          # 计数器用于控制服务器超时退出伺服。
    while True
    {
        recvnet = clientnet.recvNet()              # 定制本次接收的 NET 对象
        recvnet.recv()                             # 基于定制 NET 对象，启动接收
        while recvnet.recvOK() != -1   {PowerDown(3)}   # 等待接收函数退出(退出原因：超时、货收到信息)，释放资源。

        # 接收结果是一个列表，第一项是-1 表示接收超时，否则表示收到的数据包字节数，
        # 第二项是收到的数据（最大不超过 1024 字节）
        xrecv = recvnet.recvRead("string")             # 读取接收结果
        if xrecv[0] == -1          # 在初始化时我们设置的 tiemout = 3 秒，系统自动重新连接 5 次，每次超时时间大约 15 秒
        {
            timescount = timescount + 1      # 每超时一次计数器（timescount）+1
            if timescount > 40             # 40*15 = 600 秒，即 10 分钟，如果如果 10 分钟，结束对当前客户端的伺服服务。
            {
                count = count - 1                      # 在线人数计数器 -1
                print("在线人数：%\n",count)
                clientnet.close()                      # 关闭 clientnet
                return                                 # 结束文件服务线程
            }
            continue                                   # 没有超过设置时间，继续接收
        }
        recvbuf = xrecv[1]                             # 获取接收到的数据
        break
    }

    # 解析收到的信息
    if recvbuf != "exit"                               # 如果收到的不是"exit"，应该是文件名
    {
        fp = FILEOPEN(recvbuf,"rb")                    # 收到文件名后，服务器尝试打开文件
        if fp == False                                 # 如果文件打不开，通知客户端，然后结束线程
        {
            fp.close()
            buf = "打开文件失败，请检查文件名、或路径是否错误！"
            print("%\n",buf)                           # 报错
            sendnet = clientnet.sendNet()              # 定制发送 NET 对象
            sendnet.send(buf,len(buf))                 # 启动发送，返回资源 ID 号
            while sendnet.sendOK() != -1   {PowerDown(3)}      # 等待发送函数退出（-1 表示退出），释放资源
            xsend = sendnet.sendRead()                 # 读取发送结果
            if xsend == -1                             # 超时，网络有问题
            {
                count = count - 1                      # 在线人数计数器 -1
                print("在线人数：%\n",count)
                clientnet.close()                      # 关闭 clientnet
                return                                 # 结束文件服务线程
            }
        }
        else      # 正确打开文件，准备发送文件
        {
            data = fp.read(200)                        # 读出 300 个字节的二进制文件
```

```
            fp.close()                                      # 关闭文件
            # 先发送通知：客户端收到该通知后，切换到二进制数据接收模式
            sendnet = clientnet.sendNet()                   # 定制发送 NET 对象
            sendnet.send("binary",6)                        # 转向二进制接收的通知
            while sendnet.sendOK() != -1    {PowerDown(3)}      # 检测发送函数直到退出，释放资源
            xsend = sendnet.sendRead()                      # 读取发送结果
            if xsend == -1                                  # 超时，结束服务。
            {
                count = count - 1                           # 在线人数计数器 -1
                print("在线人数：%\n",count)
                clientnet.close()                           # 关闭 clientnet
                return                                      # 结束文件服务线程
            }
            time.sleep(500)

            # 发送二进制发送数据
            sendnet = clientnet.sendNet()                   # 定制发送 NET 对象
            sendnet.send(data,200)                          # 基于定制 NET 对象，启动发送
            while sendnet.sendOK() != -1    {PowerDown(3)}      # 检测发送函数直到退出，释放资源
            xsend = sendnet.sendRead()                      # 读取发送结果
            if xsend == -1                                  # 超时，结束服务。
            {
                count = count - 1                           # 在线人数计数器 -1
                print("在线人数：%\n",count)
                clientnet.close()                           # 关闭 clientnet
                return                                      # 结束文件服务线程
            }
            time.sleep(500)

            # 至此，文件发送结束，再发送"exit"，通知客户端，文件全部发送完毕。
            sendnet = clientnet.sendNet()                   # 定制发送 NET 对象
            sendnet.send("exit",4)                          # 基于定制对象，启动发送
            while sendnet.sendOK() != -1    {PowerDown(3)}      # 检测发送函数直到退出，释放资源
            xsend = sendnet.sendRead()                      # 读取发送结果
            if xsend == -1                                  # 超时，结束本次连接
            {
                count = count - 1                           # 在线人数计数器 -1
                print("在线人数：%\n",count)
                clientnet.close()                           # 关闭 clientnet
                return                                      # 结束文件服务线程
            }
        }
    }

    # 至此，要么是客户端主动下线了，要么是申请的文件全部发送结束了
    count = count - 1                                       # 在线人数计数器 -1
    print("在线人数：%\n",count)
    clientnet.close()                                       # 关闭 clientnet，定制的 NET 对象，系统会自动回收
    # 线程执行完毕，自动结束线程
}

server_addr = "192.168.111.195"                             # 服务器地址
port = 5500                                                 # 服务端口号

net = NET()                                                 # 定义 NET 对象 net
net = net.socket(net.IPv4,net.STREAM,net.TCP)               # 创建 socket,并赋值 net（IPv4、STREAM 数据流、TCP 协议）
net.bind(net.IPv4,server_addr,port)                         # 将服务器：地址簇、地址和端口号与 net 绑定
net.timeout(3000)                                           # 设置超时时间 3000 毫秒
net.listen(10)                                              # 将 net 设置为监听状态，参数 10 是最大并发客户端数量
while True
{
    print("服务器伺服接收！\n")
    net.accept()                                            # 开始监听，net.timeout()设置对 accept 有效
    while net.acceptOK() != -1 {PowerDown(3)}               # 等待退出监听函数（返回-1 表示退出 accept 函数）
    #读出的监测结果是一个列表，列表第一项是-1 表明超时，如果不是-1，表明监测到客户接入，
    #列表第二项是客户端的 net，第三项是客户端地址
    xaccept = net.acceptRead()                              # 读出结果是一个列表
    if xaccept[0] == -1 {continue}                          # 超时，继续监测
    list = [xaccept[1],xaccept[2]]                          # 读出监听到的客户端的 net 和客户端地址，打包成一个列表
    thread = THREAD()                                       # 定义线程对象
    thread = thread.CreateThread(serverFunc,list)           # 创建子线程处理该连接，列表作为线程参数
    thread.start()                                          # 启动子线程
}
```

```
net.clear()   #清除网络资源



# 例程 10：TCP 文件传输程序——客户端，与例程 9 构成一对

gui = GUI("fine")                          # 定义 FIne 窗口的 GUI 对象，用于图形界面显示
time = TIME()
#gui.HideConsoleWindow()                   # 隐藏控制台窗口

serveraddr = "192.168.111.195"         # 服务器地址
port = 5500                                # 该服务的端口号

net = NET()                                              # 定义一个 NET 对象
net = net.socket(net.IPv4,net.STREAM,net.TCP)        # 地址族:IPv4、数据流格式:STREAM、通讯协议:TCP
net.timeout(3000)                          # 设置 net 的接收和发送超时时间，connect()不受 net.timeout()超时设置影响。

# 连接服务器
count = 0
box = GUI("box")                                    # 创建 MessageBox 窗口对象
box.MessageBox("开始连接……","确定")
while box.MessageBoxClosed() != -1
{
    net.connect(net.IPv4,serveraddr,port)          # 参数是服务器端的:地址族、网址、端口号
    while net.connectOK() != -1
    {
        PowerDown(3)
        count = count + 1
        box.SendMessageBox("连接中：" + itoc(count))      # 显示连接中提示
    }
    # 至此，表明以退出连接状态
    box.CloseMessageBox()                          # 关闭 box 窗口

    x = net.connectRead()                          # 读取连接的结果
    if x == -1                                                  # 如果连接不成功，创建嵌套 MessageBox 提示。
    {
        box1 = GUI("box")                                  # 创建 MessageBox 嵌套（二级 MessageBox 窗口）
        box1.MessageBox("未能连接到服务器，将退出程序！","确定")
        while box1.MessageBoxClosed() != -1 {PowerDown(3)}
        return                                         # 退出主线程（退出程序）
    }
}

# 连接到服务器后，等待服务器应答
count = 0                                          # 尝试接收次数
while True                                         # 进入接收循环
{
    recvnet = net.recvNet()                                # 定制单次接收 NET 对象
    recvnet.recv()                                             # 基于定制对象启动接收
    while recvnet.recvOK() != -1 {PowerDown(3)}      # 等待接收函数退出
    xrecv =recvnet.recvRead("string")                      # 读取结果，释放资源
    if xrecv[0] == -1                                          # 超时(大约 18 秒)
    {
        box = GUI("box")                                   # 创建 MessageBox 窗口
        box.MessageBox("超时！ 未接收到服务器应答，将退出程序！","确定")
        while box.MessageBoxClosed() != -1 {PowerDown(3)}
        return                                         # 退出主线程（退出程序）
    }
    recvbuf = xrecv[1]                     # 接收到服务器应答，读取数据，并释放接收资源
    break                                              # 退出接收循环
}

# 设计 Fine 窗口，输入要下载的文件名
title    = "TCP 文件传输——客户端"                          # 窗口标题
size     = [30,5,80,20]                                                # 窗口位置、尺寸

text     = ["服务器应答","text",12,3,62,1]                # 显示区域
edit     = ["文件名","edit","s",12,7,62,1]                 # 录入文件名组件
button   = ["  确定  ","button",55,11,12,1]               # “确定”按钮

list     = [title,size,text,edit,button]                       # 打包窗口参数

gui.Fine(list)                                                 # 创建 Fine 窗口，并返回资源 ID 号
```

```
gui.SendText([recvbuf])                                  # 窗口创建后，稍微延迟一会再显示，否则显示不出来
filename = ""                                                  # 初始化文件名为空
while gui.FineClosed() != -1                          # 检测文件录入窗口是否关闭
{
    PowerDown(3)                                               # 降低数据录入期间的功耗
    # 等待输入待下载文件名
    if gui.FineReady() == -1 {continue}        # 检测是否有数据录入（0 有数据，-1 无数据）
    x = gui.FineRead()                          # 读取数据，第一项是指令(button-0)，第二项是文件名(对应 edit 组件)
    filename = x[1]                              # 拆包，把文件名取出
    if filename == ""
    {
        # 弹窗提示错误
        box = GUI("box")                                       # 创建 MessageBox 对象
        box.MessageBox("文件名不能为空！","确定")          # 创建 MessageBox 窗口
        if box.MessageBoxClosed() != -1 {PowerDown(3)}     # 等待关闭窗口
        continue                                               # 跳转，等待下次数据录入
    }
    gui.CloseFine()                          # 获取文件名后，关闭文件名录入窗口
}

# 没有文件名，将退出程序。但是，由于已经建立了连接，退出程序前，需要通知服务器，
# 以便服务器及时关闭对应的服务
if filename == ""
{
    buf = "exit"                                               # 客户端退出指令
    sendnet = net.sendNet()                          # 定制单次发送的 NET 对象
    sendnet.send(buf,len(buf))                       # 启动发送，并返回发送资源
    while sendnet.sendOK() != -1 {PowerDown(3)}      # 等待发送函数退出（返回-1 线程退出），退出后，立即释放资源。
    net.close()                                                #清除网络资源
    return                                                     # 退出程序
}

# 文件名录入成功后，向服务器发送文件名

sendnet = net.sendNet()                                    # 定制单次发送 NET 对象
sendnet.send(filename,len(filename))                # 启动发送，并返回发送资源
while sendnet.sendOK() != -1 {PowerDown(3)}        # 等待发送函数退出（返回-1 表示发送函数退出）
xsend = sendnet.sendRead()                                 # 读发送结果，同时释放资源
if xsend == -1                                                 # 发送字节数为 0，表明发送错误
{
    box = GUI("box")                                       # 创建 MessageBox 窗口对象
    box.MessageBox("发送文件名超时，将退出程序！","确定")          # 弹出提示信息
    while box.MessageBoxClosed() != -1 {PowerDown(3)}          # 等待关闭 MessageBox 窗口
    net.close()                                                # 清除网络资源
    return                                                     # 程序返回，退出程序
}

recv_finished = 0                          # 文件接收完毕标志（0 未接收完，1 接收完毕）

# 设计 Fine 窗口，动态显示收到的数据
title          = "服务器下载文件接"
size           = [10,1,120,40]
menu       = ["文件","menu","保存"]
editbox = ["","editbox",0,0,116,36]
list             = [title,size,menu,editbox]

gui = GUI("fine")                # 创建 Fine 窗口对象（每次创建窗口都需要先创建窗口对象，一个窗口对象对应一个窗口）
gui.Fine(list)
while gui.FineClosed() != -1
{
    PowerDown(3)
    while not recv_finished                                    # 如果文件未收完，持续循环直到文件接收完毕
    {
        # 冗余处理，检测窗口状态，如果用户中途关闭接收窗口，终止接收，关闭 net，退出程序
        if gui.FineClosed() == -1
        {
            net.close()                                        # 清理网络资源
            box = GUI("box")                           # 创建 MessageBox 窗口对象，弹窗提示。
            box.MessageBox("客户端退出接收程序！","确定")                  # 弹出提示信息
            while box.MessageBoxClosed() != -1 {PowerDown(3)}        # 等待关闭 MessageBox 窗口
            return
        }

        # 循环接收服务器端发送的数据包
        recvnet = net.recvNet()                # 每次接收数据包之前，需要定制一个接收 Net 对象，基于此对象操作
```

```
    recvnet.recv()                                              # 启动接收
    while recvnet.recvOK() != -1 {PowerDown(3)}         # 等待接收过程结束，并释放资源
    xrecv = recvnet.recvRead("string")                                  # 读取结果
    if xrecv[0] == -1                           # 超时次数超限
    {
        box = GUI("box")                    # 弹窗提示错误
        box.MessageBox("接收文件超时，将退出程序！","确定")          # 弹出提示信息
        while box.MessageBoxClosed() != -1 {PowerDown(3)}           # 等待关闭 MessageBox 窗口

        gui.CloseFine()                                         # 关闭 fine 窗口
        net.close()                                                 # 关闭 net，清理网络资源
        return                                                      # 程序返回，退出程序
    }
    recvbuf = xrecv[1]                      # 如果接收正确，获取数据
    if recvbuf == "exit"                    # 收到的数据包为“exit”时，表明文件发送完毕、服务器已退出
    {
        recv_finished = 1                   # 置位数据接收完毕标志
        box = GUI("box")                    # 弹窗提示
        box.MessageBox("文件接收完毕！","确定")                                 # 弹出提示信息
        while box.MessageBoxClosed() != -1 {PowerDown(3)}       # 等待关闭 MessageBox 窗口
        break                               # 退出发送循环
    }
    elif recvbuf == "binary"    # 转向二进制接受模式
    {
        # 使用二进制接收模式，循环接收，直到收到"exit"发送结束标志，置 recv_finished = 1，退出程序
        while True
        {
            recvnet = net.recvNet()       # 每次接收数据包之前，需要定制一个接收 Net 对象，基于此对象操作
            recvnet.recv()                                          # 启动接收
            while recvnet.recvOK() != -1 {PowerDown(3)}         # 等待接收过程结束，并释放资源
            xrecv = recvnet.recvRead("binary")                      # 读取结果
            if xrecv[0] == -1                           # 超时次数超限
            {
                box = GUI("box")                    # 弹窗提示错误
                box.MessageBox("接收文件超时，将退出程序！","确定")          # 弹出提示信息
                while box.MessageBoxClosed() != -1 {PowerDown(3)}           # 等待关闭 MessageBox 窗口

                gui.CloseFine()                                         # 关闭 fine 窗口
                net.close()                                                 # 关闭 net，清理网络资源
                return                                                      # 程序返回，退出程序
            }
            recvbuf = xrecv[1]                      # 如果接收正确，获取数据
            if recvbuf == "exit"                    # 收到的数据包为“exit”时，表明文件发送完毕、服务器已退出
            {
                recv_finished = 1                   # 置位数据接收完毕标志
                box = GUI("box")                    # 弹窗提示
                box.MessageBox("文件接收完毕！","确定")                 # 弹出提示信息
                while box.MessageBoxClosed() != -1 {PowerDown(3)}   # 等待 MessageBox 窗口关闭
                break                                                   # 关闭
            }
            else        # 收到数据包
            {
                str = sprint(recvbuf)
                gui.SendEditbox([str])          # 将收到的数据显示在显示区
            }
        }
    }
    else        # 收到数据包
    {
        str = sprint(recvbuf)
        gui.SendEditbox([str])          # 将收到的数据显示在显示区
    }
}

# 数据接收完毕之前，程序无法流转到此处，按钮消息被阻滞（不响应按钮和菜单），之后才响应菜单或按钮
if gui.FineReady() == 0                     # 有界面数据输出
{
    x = gui.FineRead()                      # x[0]是指令（menu-0-0 来自 menu）,x[1]是文件内容（来自 editbox）
    # 检查同名文件是否存在
    fp = FILEOPEN(filename,"r")             # 尝试打开文件
    if fp == False                          # 打开失败，表明文件不存在
    {
        is_writefile = 1                    # 置位写文件标志，可以写入文件标志
    }
```

```
        else                                    # 如果能打开文件，说明文件存在，询问是否同意覆盖原文
        {
            fp.close()                          # 关闭文件

            box = GUI("box")                    # 创建弹窗 GUI 对象
            box.MessageBox(filename+"已经存在，是否要覆盖原文件？","确定|取消")
            while box.MessageBoxClosed() != -1 {PowerDown(3)}      # 等待做出选择
            select = gui.MessageBoxRead(box)                       # 读取选择结果
            if select == 0                                          # 0 表明是确定
            {
                is_writefile = 1                                    # 置位写文件标志，将写入文件
            }
            else                                                    # 表明选择的是取消、或直接关闭弹窗
            {
                is_writefile = 0                                    # 清 0 写文件标志，将不会写文件
            }
        }

        if is_writefile == 1                    # 如果写文件标志是 1，进行写文件操作（将覆盖原文件）
        {
            fp = FILEOPEN(filename,"w")         # 只写方式打开文件，
            fp.write(x[1])                      # 将从 editbox 组件上读出的内容，写入文件
            fp.close()                          # 关闭文件
        }
    }
}
net.close()   #清除网络资源



# 例程 11：UDP 二进制传输——服务器端，与例程 12 构成一对

time = TIME()
net = NET()                     # 定义一个 NET 对象
port = 4321

net = net.socket(net.IPv4,net.DGRAM,net.UDP)   # 地址族:IPv4、数据流格式:DGRAM、通讯协议:UDP
# 把服务器地址与上述 socket 绑定
net.bind(net.IPv4,"192.168.111.195",port)       # 参数分别是服务器的：地址族、网址、端口号
net.timeout(3000)

databuf = Binary("x00x11x22x33x44x55x66x77x88x99xaaxbbxccxddxeexff")

print("服务器伺服接收！\n")
while True
{
    recvnet = net.udprecvNet()                  # 创建一个 udp 接收 NET 对象
    recvnet.udprecv()                           # 服务器端等待接收来自客户端的请求
    while recvnet.udprecvOK() != -1 {PowerDown(3)}   # 等待 udprecvOK 退出，释放资源
    x = recvnet.udprecvRead("string")           # 读取数据
    if x[0] == -1 {continue}                    # 第一项为-1 表明超时，继续接收
    RecvDatas = x[1]                            # 收到的数据
    remote_addr = x[2]                          # 发送端的地址

    while True
    {
        sendnet = net.udpsendNet()                          # 创建 udp 发送 NET 对象
        sendnet.udpsend(net.IPv4,remote_addr,port,databuf,16)   # 发送信息时，客户端地址，端口号
        while sendnet.udpsendOK() != -1 {PowerDown(3)}      # 等待 udpsend 返回,，释放资源
        xsend = sendnet.udpsendRead()
        time.sleep(1000)
    }
}
net.close()   # 清除网络资源



# 例程 12：UDP 二进制接收——客户端，与例程 11 构成一对

gui = GUI("fine")               # 定义 GUI 对象，用于图形界面显示
#gui.HideConsoleWindow()        # 隐藏控制台窗口

net = NET()                     # 定义一个 NET 对象
time = TIME()
```

```
localaddr = net.local_addr()            # 自动获取本地的 IP 地址
port = 4321                             # 端口号

net = net.socket(net.IPv4,net.DGRAM,net.UDP)   # 给 net 赋值（地址族、数据流格式、UDP 传输）
net.bind(net.IPv4,localaddr,port)              # 将 net 与本地地址"192.168.47.195"和端口 4321 绑定
net.timeout(3000)                              # 设置 net 的接收和发送超时时间

server_addr = "192.168.111.195"                # 服务器地址，


# 设计群聊窗口界面
title     = "UDP 聊天程序——客户端"
size      = [30,5,80,30]
testbox = ["群聊内容","textbox","N",10,1,66,17]
edit      = ["发送内容","edit","s",10,19,64,3]

list      = [title,size,testbox,edit]

gui.Fine(list)                          # 创建聊天 Input 窗口，返回资源 ID

while True
{
    sendnet = net.udpsendNet()              # 分配发送资源
    sendnet.udpsend(net.IPv4,server_addr,port,"连接服务器……",len("连接服务器……"))  # 启动发送，地址是服务器地址
    while sendnet.udpsendOK() != -1 {PowerDown(3)}
    xsend = sendnet.udpsendRead()           # 读取发送结果，释放资源
    if(xsend > 0) {break}
    gui.SendTextbox(["连接服务器……\n"])     # 显示收到的内容
    time.sleep(500)
}

recvnet = net.udprecvNet()        # 创建 udp 接收 NET 对象
recvnet.udprecv()                 # 启动接收服务，接收长度限制在 1024 字节以内

while gui.FineClosed() != -1            # 检测聊天窗口是否关闭
{
    PowerDown(3)                        # 节能设置
    if recvnet.udprecvOK() == -1      # udprecv 返回-1，表明退出 udprecv 函数，释放资源
    {
        xrecv = recvnet.udprecvRead("binary")     # 读取接收结果
        if xrecv[0] > 0                           # 如果超时返回-1，否则，返回接收到的字节数
        {
            recvdada = xrecv[1]                   # 接收到的二进制数据
            str = sprint(recvdada) + "\n"         # 转换成字符串
            gui.SendTextbox([str])                # 显示收到的内容
        }

        # 处理完上次收到的数据后，再次进入伺服接收状态
        recvnet = net.udprecvNet()          # 创建 udp 接收 NET 对象
        recvnet.udprecv()                   # 启动接收服务，接收长度限制在 1024 字节以内
    }
}
net.close()   #清除网络资源
```

第二十一章、串口通讯　　例程《serial_method.txt》

串口方法：

1、serial = SERIAL("COM3",115200)
参数 1 是串口名称，参数 2 是波特率（9600、14400、19200、38400、57600、115200、128000、256000 选一）。返回一个串口操作对象。

2、writeBytes = serial.Write(buffer,12)
参数 1 是待发送数据缓冲区，参数 2 是待发送数据的长度。待发送数据中间允许出现 0x00 的数据（适用于传送二进制数），因此，需要第二个参数来指明需要发送的字节数，而不是计算字符串的长度。返回，实际发送的字节数。

3、result = serial.IsReady()
无参数，返回 0 表明收到收据，返回-1 表明未收到数据

4、list = serial.Read()　参数要么是"string"，要么是"binary"，前者表明收到的是字符串，后者表明收到的是二进制数据块，返回一个列表，列表第一项是收到的数据长度，第二项是收到的数据。

5、serial.Close()
无参数，无返回，清除串口对象，关闭窗口。

注意事项一：收、发缓冲区默认为 1024 字节，如果超出限制，需要分成多次收发。
注意事项二：由于串口收发采用的是异步方式，因此，在读取接收数据时，需要调用 serial.IsReady()查询是否收到数据，只有当该方法的返回结果是 0 时，读取的数据才有效。serial.IsReady()返回-1 表明未收到数据。
注意事项三：可以同时支持 4 个串口同时工作。
注意事项四：串口配置——异步串口、8bit 数据、1bit 停止位、无奇偶校验，禁止 RTS/CTS 数据流控。

```
# 例程 1：串口发送

serial = SERIAL("COM3",9600)
time = TIME()

count = 0
while True
{
    bytesWritten = serial.Write("你好！ Hello World!",18)
    print("%\n",bytesWritten)
    count = count + 1
    if count >= 20 {break}
    time.sleep(2000)
}
serial.Close()



# 例程 2：串口接收

serial = SERIAL("COM7",9600)

while True
{
    while serial.IsReady() != 0 {PowerDown(3)}    # serial.IsReady() = 0 表明收到数据，没有收到数据返回-1
    list = serial.Read("string")                  # 查询到新数据后，退出循环，读取数据
    print("%\n",list)
}
serial.Close()                                    # 关闭串口，注意，用完串口需要关闭。
```

```
# 例程 3：串口收发，调试该程序，需要事先用数据线连接两个串口，并对两个串口进行配置，两个端口的波特率要相同。

gui = GUI("fine")                       # 定义一个 GUI 对象
#gui.HideConsoleWindow()                # 隐藏控制台窗口
serial = SERIAL("COM7",9600)

title    = "串口收发"                    # Fine 窗口标题
Size     = [20,10,80,30]                # 指定窗口位置和宽度高度

edit     = ["发送数据","edit","s",12,2,60,4]
button   = [" 发送 ","button",3,4.5,8,1]
textbox = ["","textbox","N",12,8,42,12]

list = [title,Size,edit,button,textbox]       # 将设计的窗口元素打包成一个列表

gui.Fine(list)                                  # 创建 Fine 窗口，并返回窗口资源 ID 号
while gui.FineClosed() != -1                    # 循环检测窗口关闭消息，如果 gui.FineClosed(id) = -1 表明窗口已关闭
{
    PowerDown(3)                                # 低功耗设置
    if gui.FineReady() == 0                     # 如果点击了“发送”按钮
    {
        x = gui.FineRead()                      # x 是个列表，第一项是指令，这里就是发送指令，第二项是 edit 录入数据
        bytesWritten = serial.Write(x[1],len(x[1]))         # 串口发送

        str = "发送数据：" + x[1] + "\n"
        gui.SendTextbox([str])                              # 显示在 textbox 中
        gui.SendEdit([""])                                  # 清空 edit 组件
    }
    if serial.IsReady() == 0                                # 串口收到数据
    {
        data = serial.Read("string")
        str = "接收数据：" + data[1] + "\n"
        gui.SendTextbox([str])                              # 显示在 textbox 中
    }
}


# 例程 4：串口收发，调试该程序，需要事先用数据线连接两个串口，并对两个串口进行配置，两个端口的波特率要相同。
# 收发二进制数据

gui = GUI("fine")                       # 定义一个 GUI 对象
#gui.HideConsoleWindow()                # 隐藏控制台窗口
serial = SERIAL("COM7",9600)

title    = "串口收发"                    # Fine 窗口标题
Size     = [20,10,80,30]                # 指定窗口位置和宽度高度

edit     = ["发送数据","edit","s",12,2,60,4]
button   = [" 发送 ","button",3,4.5,8,1]
textbox = ["","textbox","N",12,8,42,12]

list = [title,Size,edit,button,textbox]       # 将设计的窗口元素打包成一个列表

gui.Fine(list)                                  # 创建 Fine 窗口，并返回窗口资源 ID 号
while gui.FineClosed() != -1                    # 循环检测窗口关闭消息，如果 gui.FineClosed(id) = -1 表明窗口已关闭
{
    PowerDown(3)                                # 低功耗设置
    if gui.FineReady() == 0                     # 如果点击了“发送”按钮
    {
        x = gui.FineRead()                      # x 是个列表，第一项是指令，这里就是发送指令，第二项是 edit 录入数据
        binarydata = Binary(x[1])               # x[1]是界面录入的数据（x01x02x03 格式），通过函数 Binary 转换成二进制数据
        length = int(len(x[1])/3)               # x01 三个字符表示一个二进制的字节，所以，转换成二进制的长度是 len(x[1])/3
        bytesWritten = serial.Write(binarydata,length)      # 串口发送

        str = "发送数据：" + x[1] + "\n"
        gui.SendTextbox([str])                              # 显示在 textbox 中
        gui.SendEdit([""])                                  # 清空 edit 组件
    }
    if serial.IsReady() == 0                                # 串口收到数据
    {
        data = serial.Read("binary")                        # 参数表明接收的是二进制数据
        if data[0] > 0
        {
            str1 = sprint(data[1])                          # 是收到的二进制数据，通过 sprint 函数转换成字符串
            str = "接收数据：" + str1 + "\n"
            gui.SendTextbox([str])                          # 显示在 textbox 中
        }
    }
}


# 例程 5：串口收发，调试该程序，需要事先用数据线连接两个串口，并对两个串口进行配置，两个端口的波特率要相同。
# 打开一个二进制文件，读出前 300 个字节，串口发送出去
# 该例程作为发送端，例程 4 作为接收端。
```

```
serial = SERIAL("COM7",115200)

fp = FILEOPEN("finecomp.exe","rb")          # 打开文件（文本、只读方式）
if fp == False                               # 如果打开文件出错，返回 False
{
    print("%\n","打开文件出错！")
}
else
{
    binarydata = fp.read(300)         # 读取 100 个字节的二进制文件内容
    fp.close()                          # 关闭已经打开的文件

    bytesWritten = serial.Write(binarydata,300)          # 串口发送

    print("%\n",binarydata)                    # 显示读出的二进制数据块
    print("%\n")
}
```