模块 Module

模块是一个包含有一系列数据,函数,类等组成的程序组

模块是一个文件,模块文件名通常以.py结尾

作用:

1. 让一些相关的函数,数据,类有逻辑的组织在一起,使逻辑结构

更加清晰

1. 模块中的数据,函数和类等可以提供给其它模块使用

模块化编程的优点:

1. 有利于多人合作开发

2. 使代码更加利于维护

3. 提高代码的复用率

4. 模块化编程有助于解决函数名和变量名冲突(重名) 问题,模块内

的变量作用域为模块内全局

模块的分类:

内置模块(builtins) 在解析器的内部可以直接使用

标准库模块,安装python时已安装且可以直接使用

第三方模块(通常为开源), 需要自己安装

安装命令:

pip3 install 模块名

或

pip install 模块名(通常用来安装python3的模块)

用户自己编写的模块

模块的导入语句 import

import 语句

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

示例:

import math

import sys, time

作用:

当某模块整体导入到当前模块中

用法:

模块名.属性名

math.factorial(5) # 返回120

dir(obj) 函数,返回obj对象所有的属性的字符串列表

help(obj) 函数,可以查看模块的文档字符串

练习:

学习使用数学模块math来计算:

1. 输入一个圆的半径,打印出这个圆的面积

2. 输入一个圆的面积,打印出这个圆的半径

面积 = 圆周率 \* r \*\* 2

from import 语句

语法:

from 模块名 import 模块属性名1 [as 属性新名1],

模块属性名2 [as 属性新名2], ....

作用:

将某模块内的一个或多个属性导入到当前模块的作用域

from import \* 语句

语法:

from 模块名 import \*

作用:

将某模块的所有属性导入到当前模块作用域

dir函数

dir([对象])

作用:

如果没有参数调用,则返回当前作用域内所有变量的列表

如果给定一个对象作为参数,则返回这个对象的所有变量的列表

对于一个模块,返回这个模块的全部属性(变量)

对于一个类对象,返回这个对象的所有变量,并递归基类对象

的所有变量

对于其它对象返回所有变量,类变量和基类变量

内建模块

math

time

sys

math 模块

文档参见:

math.html

time 模块

文档参见:

time.html

sys 模块

运行时系统相关的信息 (runtime system)

文档参见:

sys.html

自定义模块并导入

要求:

模块文件后缀名必须以.py结尾

模块文件名必须是合法的标识符

避免名称和内建模块名冲突

导入方式:

import 语句

from import 语句

from import \*语句

正确的模块名:

mymod.py abcde.py abc123.py

错误的模块名:

123.py abc.cpp math.py

import 语句 搜索模块的路径顺序

1. 搜索内建模块

2. 搜索程序的运行时路径(当前工作目录)

3. sys.path 提供的路径

sys.path是一个列表,里面放的都是模块的搜索路径

模块的加载过程:

在模块导入时,模块的所有语句都会执行

如果一个模块已经导入,则再次导入时不会重新执行模块内的语句

模块新的重载加载

import mymod

import imp

imp.reload(mymod) # 重新加载已加载的mymod模块

注: 此做法通常只在开发阶段调式时使用

模块被导入的执行的过程

1. 先搜索相关的路径,找到.py文件

2. 判断是否有此文件对应的.pyc文件, 如果没有.pyc文件,则用

.py文件生成.pyc文件

3. 如果存在 .pyc文件,需要判断.py是否比.pyc文件新.如果新则

重新生成.pyc, 然后再加载.pyc文件

pyc文件 ,模块的编译文件

编译 解释执行

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

模块的属性

模块的文档字符串

\_\_doc\_\_属性

作用:

用来绑定模块的文档字符串

\_\_file\_\_属性

作用:

用来绑定模块对应的文件路径名

注:

内建模块没有\_\_file\_\_属性(如:math模块)

\_\_name\_\_属性

作用:

1. 用来记录模块的自身的名字

2. 用来判断是否为主模块

注: 主模块是指程序中第一个执行起来的模块

说明:

当此模块作为主模块运行时,\_\_name\_\_属性 绑定'\_\_main\_\_'

当此模块不作为主模块运行时,此\_\_name\_\_属性绑定模块名

如:mymod.py 的模块名为'mymod'

模块的 \_\_all\_\_列表

模块中 \_\_all\_\_列表是一个用来存放可导出属性的字符串列表

作用:

限定在用from xxx import \*导入时,只导入\_\_all\_\_列表

内的属性

注: 此\_\_all\_\_列表,只对from import \*语句有效

模块的隐藏属性

模块中以下划线(\_)开头的属性 在 from import \*语句导入时

将不被导入,通常称这些属性为隐藏属性

标准库模块

随机模块 random

作用:

用于模拟或生成随机数的模块

文档参见:

random.html

练习:

1. 猜数字游戏

随机生成一个0~100的整数,用变量x绑定

让用户输入一个数y,输出猜数字的结果

如果y大于x,则提示"您猜大了", 继续让用户输入

如果y小于x,则提示"您猜小了", 继续猜

如果y等于x,则提示"恭喜您猜对了", 打印出猜数字的次数

然后退出程序

注:

直到猜对才退出

2.剪刀石头布游戏

3.使用随机模块,随机生成6位的密码

可以作为密码的字符有:

a-z A-Z 0-9

写一个程序,每次调用时能随机生成一个密码

包 package

包的定义

包是将模块以文件夹的形式进行分组管理的方法

作用:

将一系列模块进行分类管理,有利于防止命名冲突

可以在需要时加载一个或部分模块而不是全部模块

包的导入:

同模块的导入规则相同

语法:

import 包名 [as 包别名]

import 包名.模块名 [as 模块新名]

import 包名.子包名.模块名 [as 模块新名]

...

from 包名 import 模块名 [as 模块新名]

from 包名.子包名 import 模块名 [as 模块新名]

from 包名.子包名.模块名 import 属性名 [as 属性新名]

...

from 包名 import \*

from 包名.模块名 import \*

import 语句搜包的路径顺序

1. 搜索当前运行时路径(当前工作目录)

2. 搜索sys.path 提供的路径

\_\_init\_\_.py 文件

是常规包内必须存在的文件

\_\_init\_\_.py 会在包加载时自动调用

作用:

编写此包的内容

在内部填写文档字符串

在\_\_init\_\_.py文件内可以加载此包所依懒的一些其它模块

\_\_init\_\_.py 内的 \_\_all\_\_列表

作用:

用来记录此包中有哪些模块或子包需要导入

当用from 包 import \*语句导入模块时,只查找\_\_all\_\_中

所有的模块或子包

说明:

\_\_all\_\_列表只对 from import \*语句起作用

包的相对导入

包的相对导入是指包内模块的相互导入

语法:

from 相对路径包或模块 import 属性名或模块名

或

from 相对路径包或模块 import \*

相对路径:

在用from xxx import yyy 语句中

xxx 部分可以使用相对路径

. 代表当前目录

.. 代表上一级目录

... 代表上二级目录

.... 以此类推

注: 相对导入时不能超出包的外部

异常 exception

什么是错误

错误是指由于逻辑或语法等导致一个程序无法正常执行的问题

错误的特点:

有些错误是无法预知的

什么是异常

异常是程序出错时标识的一种状态

当异常发生时,程序不会再下向执行,而转去调有此函数的地方待处理

此错误并恢复为正常状态

异常的作用:

用作信号,通知上层调用者有错误产生需要处理

try语句 的 两种语法:

try-except语句

try-finally语句

try-except语句的语法

try:

可以触发异常的语句

except 错误类型1 [as 变量名1]:

异常处理语句1

except 错误类型2 [as 变量名2]:

异常处理语法2

except (错误类型3, 错误类型4, ...) [as 变量名3]:

异常处理语句3

...

except:

异常处理语法

else:

未发生异常语句

finally:

最终语句

作用:

偿试捕获异常,得到异常通知,将程序由异常流程转为正常流程并

继续执行

说明:

as 子句是用于绑定错误对象的变量,可以省略

except 子句可以有一个或多个,但至少要有一个

except: 子句必须放在所有的except之后

else 子句最多只有能一个,也可以省略不写

finally 子句最多只能有一个,也可以省略不写

说明2:

else 子句里的语句会在当前try中没有发生任何异常时被执行

finally 子句 里的语句无论在异常状态还是正常状态都一定会

被执行

练习:

写一个函数 get\_score() 来获取学生输入的成绩(0~100的整数)

如果输入出现异常,则此函数返回0,否则返回用户输入的成绩

如:

def get\_score():

.... # 此处自己实现

score = get\_score()

print("学生的成绩是:", score)

python 全部的错误类型

查看所有异常的类型文档 的方式:

>>> help(\_\_builtins\_\_)

思考:

如何能让之前写的<<学生信息管理系统>> 无论用户如何操作都

不会发生异常而退出?

try-finally语句

语法:

try:

可能触发异常的语句

finally:

最终语句

说明:

finally子句不可以省略

一定不存在except 子句

作用:

通常用try-finally语句来做触发异常时必须要处理的事情,无论

异常是否发生,finally子句都会被执行

注:

try-finally语句不会改变程序的正常/异常状态

raise 语句

作用:

触发一个错误,让程序进入异常状态

发送错误通知给调用者

语法:

raise 异常类型

或

raise 异常对象

或

raise # << 重新触发上一次异常

说明:

raise 无参调用时,只能放在try-except 的except 子句里

用来触发刚收到且正在处理的错误信息

练习:

写一个函数 get\_age() 用来获取一个人的年龄信息

此函数规定用户只能输入 1~140之间的整数,如果用户输入其它

的数则直接触发ValueError类型的错误!

如:

def get\_age()

... # 此处自己实现

try:

age = get\_age()

print("用户输入的年龄是", age)

except ValueError as err:

print("获取年龄时发生错误,错误的原因是", err)

assert 语句(断言语句)

语法:

assert 真值表达式, 错误数据(通常是字符串)

作用:

当真值表达式为假(False)时,用错误数据创建一个

AssertionError 类型的错误,并进入异常状态

通常用来在编写代码时,故意抛出异常来发现错误

等同于:

if bool(真值表达式) == False:

raise AssertionError(错误数据)

为什么要用异常处理机制

在程序调用层数较深时,向主调函数传递错误信息要层层return

比较麻烦,所以用异常处理机制

异常小结:

语句:

try-except 接收异常通知(把异常状态转为正常状态)

try-finally 执行必须要执行的语句(无论正常/异常状态)

raise 语句 抛出(发送) 错误通知,将程序转为异常状态

assert 语句 根据条件决定是否抛出AssertionError类型

的错误,并转为异常状态