# 记账程序

#

# 1. 假设在记账时,有一万元钱

# 2. 无论是开销还是收入都要进行记账

# 3. 记账内容包括时间、金额和说明等

# 4. 记账数据要求永久存储

# import os

# import time

# import pickle as p

#

# def save\_money(record,wallet): #存钱

# try:

# amount = int(input('anmount:'))

# comment = input('comment:')

# except:

# print('\nInvalid input.Return main menu.')

# return

# data = time.strftime('%Y-%m-%d')

#

# with open(wallet,'rb') as fobj:

# balance = p.load(fobj) + amount

# with open(wallet,'wb')as fobj:

# p.dump(balance,fobj)

# with open(record,'a') as fobj:

# fobj.write(

# '%-15s%-8s%-8s%-10s%-20s\n' % (data,'',amount,balance,comment)

# )

#

# def cost\_money(record,wallet): #花钱

# amount = int(input('anmount:'))

# comment = input('comment:')

# data = time.strftime('%Y-%m-%d')

# with open(wallet,'rb') as fobj:

# balance = p.load(fobj) - amount

# with open(wallet,'wb')as fobj:

# p.dump(balance,fobj)

# with open(record,'a') as fobj:

# fobj.write(

# '%-15s%-8s%-8s%-10s%-20s\n' \

# % (data,amount,'',balance,comment)

# )

#

# def query\_money(record,wallet): #查询

# print(

# '%-12s%-8s%-8s%-10s%-20s' % \

# ('data', 'save', 'cost', 'balace', 'comment'))

# with open(record) as fobj:

# for line in fobj:

# print(line,end='')

# with open(wallet,'rb') as fobj:

# print('\033[34;1m剩余余额：%d\033[0m' % p.load(fobj))

#

# def show\_menu():

# record = 'record.txt'

# wallet = 'wallet.data'

# if not os.path.exists(wallet): #当wallet文件不存在时，创建文件并写入

# with open(wallet,'wb')as fobj:

# p.dump(10000,fobj)

# cmds = {'0':save\_money,'1':cost\_money,'2':query\_money}

#

# prompt= '''

# (0) save

# (1) cost

# (2) query

# (3) exit

# 请选择(0/1/2/3)：'''

#

# while True:

# try:

# choice = input(prompt).strip()[0]

# except IndexError:

# continue

# except(KeyboardInterrupt,EOFError):

# print('\nBYE-bye')

# # choice = '3'

# if choice not in '0123':

# print('无效输入,请重试')

# continue

#

# if prompt == '3':

# break

# cmds[choice](record,wallet)

#

# if \_\_name\_\_ == '\_\_main\_\_':

# show\_menu()

1. 创建函数

函数不允许在函数未声明之前对其进行引用或者调用

2.内部函数

在函数体内创建另外一个函数是完全合法的,这种函数叫做内部/内嵌函数

1. 调用函数

# def foo():

# print('in foo')

#

# def bar():

# print('in bar')

# foo()

# bar()

# def get\_info(name,age):

# print('%s is %s years old' % (name,age))

#

# if \_\_name\_\_ == '\_\_main\_\_':

# get\_info('bob',25)

# def func1(\*args): #\*args表示是个元祖

# print(args)

#

#

# if \_\_name\_\_ == '\_\_main\_\_':

# func1()

# func1('hao')

# func1('hao',123)

# func1('hao',123,'hello')

#

# def func2(\*\*kwargs): #\*\*表示是个字典

# print(kwargs)

#

# if \_\_name\_\_ == '\_\_main\_\_':

# func2()

# func2(name='zhangsan')

简单的加减法数学游戏

1. 随机生成两个100以内的数字

2. 随机选择加法或是减法

3. 总是使用大的数字减去小的数字

4. 如果用户答错三次,程序给出正确答案

1.

from random import randint,choice

def add(x,y):

return x + y

def sub(x,y):

return x - y

def sh(x,y):

return x \* y

def cy(x,y):

return x / y

def exam():

cmds = {'+':add,'-':sub,'\*':sh,'/':cy}

nums = [randint(1,100) for i in range(2)]

nums.sort(reverse=True) #也可以写成nums.sort() ,nums.reverse() sort排序，reverse倒叙

op = choice('+-\*/')

result = cmds[op](\*nums) # (\*nums)有上面的循环得出两位数，\*nums是把列表拆开

prompt = '%s %s %s =' % (nums[0],op,nums[1])

couter = 0

while couter <3:

try:

answer = int(input(prompt))

except:

print()

continue

if answer == result:

print('对了')

break

else:

print('错了')

couter +=1

else:

print('%s %s' % (prompt,result) )

# if op == '+':

# nums = nums[0] +nums[1]

# if op == '-':

# nums = nums[0] - nums[1]

if \_\_name\_\_ == '\_\_main\_\_':

while True:

exam()

try:

yn = input('是否要输入(y/n)?').strip()[0]

except IndexError: #当输入的不是y和n的话，重新输入

continue

except (KeyboardInterrupt,EOFError): #当用户输入ctrl +d 或c时，默认为用户输入的是n

yn = 'n'

if yn in 'nN':

print('\nBye-bye')

break

2.

#from random import randint,choice

def exam():

cmds = {'+':lambda x,y:x+y,'-':lambda x,y:x-y,'\*':lambda x,y:x\*y,'/':lambda x,y:x/y}

nums = [randint(1,100) for i in range(2)]

nums.sort(reverse=True) #也可以写成nums.sort() ,nums.reverse() sort排序，reverse倒叙

op = choice('+-\*/')

result = cmds[op](\*nums) # (\*nums)有上面的循环得出两位数，\*nums是把列表拆开

prompt = '%s %s %s =' % (nums[0],op,nums[1])

couter = 0

while couter <3:

try:

answer = int(input(prompt))

except:

print()

continue

if answer == result:

print('对了')

break

else:

print('错了')

couter +=1

else:

print('%s %s' % (prompt,result) )

# if op == '+':

# nums = nums[0] +nums[1]

# if op == '-':

# nums = nums[0] - nums[1]

if \_\_name\_\_ == '\_\_main\_\_':

while True:

exam()

try:

yn = input('是否要输入(y/n)?').strip()[0]

except IndexError: #当输入的不是y和n的话，重新输入

continue

except (KeyboardInterrupt,EOFError): #当用户输入ctrl +d 或c时，默认为用户输入的是n

yn = 'n'

if yn in 'nN':

print('\nBye-bye')

break

1. 匿名函数

1.lambda

python允许用lambda关键字创造匿名函数

匿名是因为不需要以标准的def方式来声明

一个完整的lambda“语句”代表了一个表达式,这个表达式的定义体必须和声明放在同一行

2.filter()函数

filter调用一个布尔函数func来迭代遍历每个序列中的元素;返回一个使func返回值为true的元素的序列

# from random import randint

#

# def func1(x):

# return x % 2

# def func2(x):

# return x \* 2 +1

#

# if \_\_name\_\_ == '\_\_main\_\_':

# nums = [randint(1,100) for i in range(10)]

# print(nums)

# #filter的第一个参数是函数，这种函数叫做高阶函数

# #将nums列表中的每一项交给func1处理，返回True保留，返回False过滤掉

# print(list(filter(func1,nums)))

# print(list(filter(lambda x:x % 2, nums)))

#

# print(map(func2,nums)) #将nums列表中的每个数字交给func2加工

# print(map(lambda x: x % 2, nums))

三．函数高级应用

1.全局变量

标识符的作用域是定义为其声明在程序里的可应用范围,也就是变量的可见性

在一个模块中最高级别的变量有全局作用域

全局变量的一个特征是除非被删除掉,否则它们的存活到脚本运行结束,且对于所有的函数,他们的值都是可以被访问的

1. 局部变量

局部变量只时暂时地存在,仅仅只依赖于定义它们的函数现阶段是否处于活动

当一个函数调用出现时,其局部变量就进入声明它们的作用域。在那一刻,一个新的局部变量名为那个对象创建了

一旦函数完成,框架被释放,变量将会离开作用域

1. 名字空间

标识符的搜索顺序依次是局部、全局和内建

# x =20

# def bar():

# global x #global 全局变量,把x=20的值变为200

# x = 200

#python的窗口模式

# yum -y install tk-devel tcl-devel sqlite-devel

#cd /opt py3soft.tar.gz

#tar xzf python3.6.1xxxx.tar.gz

#cd Python-3.6.1

#./configure --prefix=/usr/local

#make

#make install

#

# import tkinter #

# root = tkinter.Tk()

# print(root)

四．函数式编程

1.偏函数

偏函数的概念是将函数式编程的概念和默认参数以及可变参数结合在一起

一个带有多个参数的函数,如果其中某些参数基本上固定的,那么就可以通过偏函数为这些参数赋默认值

# from functools import partial

#

# def add(a,b,c,d):

# return a+b+c+d

#

# if \_\_name\_\_ == '\_\_main\_\_':

# print(add(10,20,30,5))

# print(add(10,20,30,15))

# print(add(10,20,30,25))

# print(add(10,20,30,35))

# myadd = partial(add,10,20,30)

# print(myadd(5))

# print(myadd(15))

# print(myadd(25))

# print(myadd(35))

#

简单GUI程序

1. 窗口程序提供三个按钮

2. 其中两个按钮的前景色均为白色,背景色为蓝色

3. 第三个按钮前景色为红色,背景色为红色

4. 按下第三个按钮后,程序退出

#

# import tkinter

# from functools import partial

#

# root= tkinter.Tk()

# lb1 = tkinter.Label(root,text='hello world',font='Arial 15')

#text是指名字，font是指字体

# #b1 = tkinter.Button(root,bg='blue',fg='white',text='Button 1')

# MyButton = partial(tkinter.Button,root,bg='blue',fg='white')

# b1 = MyButton(text='Button 1')

# b2 = MyButton(text='Button 2')

# b3 = MyButton(text='Button 3',command = root.quit)

# lb1.pack()

# b1.pack()

# b2.pack()

# b3.pack()

# root.mainloop()

#

1. 递归函数

如果函数包含了对其自身的调用,该函数就是递归的

在操作系统中,查看某一目录内所有文件、修改权限等都是递归的应用

#

# def func(x):

# if x == 1:

# return x

# return x \* func(x - 1 )

# #3\*func(2)

# #3\*2\*func(1)

# #3\*2\*1

#

# if \_\_name\_\_ == '\_\_main\_\_':

# print(func(3))

# print(func(4))

#

# 快速排序

#1.随机生成10个数字

#2.利用递归,实现快速排序

# from random import randint

#

#

# def qsort(num\_list):

# if len(num\_list) < 2:

# return num\_list

#

# middle = num\_list[0]

# smaller = []

# larger = []

# for i in num\_list[1:]:

# if i < middle:

# smaller.append(i)

# else:

# larger.append(i)

# return qsort(smaller) + [middle] + qsort(larger)

#

#

# if \_\_name\_\_ == '\_\_main\_\_':

# nums = [randint(1,100) for i in range(10)]

# print(nums)

# print(qsort(nums))

1. 生成器

生成器是一个带yield语句的函数

一个函数或者子程序只返回一次,但一个生成器能暂停执行并返回一个中间的结果

yield 语句返回一个值给调用者并暂停执行

当生成器的next()方法被调用的时候,它会准确地从离开地方继续

# def mygen():

# yield 'hello world'

# a = 10 +20

# yield a

# yield [10,20]

#

#

#

# if \_\_name\_\_ == '\_\_main\_\_':

# # a = mygen()

# for item in mygen():

# print(item)

#

def block(fobj):

# content = []

# content = 0

# for line in fobj:

# content.append(line)

# content += 1

# if content == 10:

# yield content

# content = []

# content = 0

#

# if content:

# yield content

#

# if \_\_name\_\_ == '\_\_main\_\_':

#

# with open('/etc/passwd',)as fobj:

# for lines in block(fobj):

# print(lines)

# print()

#

#

装饰器

装饰器是在函数调用之上的修饰

这些修饰仅是当声明一个函数或者方法的时候,才会应用的额外调用

• 使用装饰器的情形有:

– 引入日志

– 增加计时逻辑来检测性能

– 给函数加入事务的能力

# def counter(start=0):

# count = start

# def incr():

# nonlocal count

# count +=1

# return count

# return incr

#

# if \_\_name\_\_ == '\_\_main\_\_':

# a = counter()

# print(a())

# print(a())

# b = counter(10)

# print(b())

# print(b())

# print(a())

# print(b())

#

# def colors(func):

# return '\033[31;1m%s\033[0m' % func()

# @colors

# def hello():

# return 'Hello world!'

#

# def welcom():

# return 'How are you?'

#

#

#

# if \_\_name\_\_ == '\_\_main\_\_':

# print(hello)

# # hello = colors(hello)

# print(hello())

#