Lab Report of Python Programming

Lab 4: Functions and Recursion  
Credit hour: 2

Student Name: 徐政辉 Student ID: 2017329621139

## ￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣￣

1. Objective
   1. Know how to define and invoke functions and create modules for reusing functions;
   2. Know what a recursive function is and the application of using recursion.
2. Lab content
   1. (The MyTriangle module) Create a module named MyTriangle that contains the following two functions:

# Returns true if the sum of any two sides is

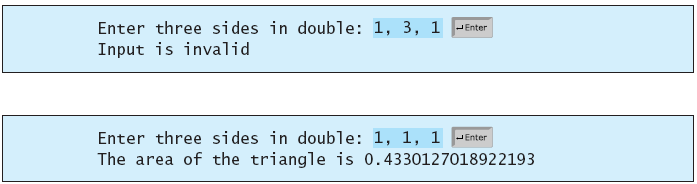
# greater than the third side.

def isValid(side1, side2, side3):

# Returns the area of the triangle.

def area(side1, side2, side3):

Write a test program that reads three sides for a triangle and computes the area if the input is valid. Otherwise, it displays that the input is invalid. Here are some sample runs:



* 1. (Palindromic prime) A palindromic prime is a prime number that is also palindromic. For example, 131 is a prime and also a palindromic prime, as are 313 and 757. Write a program that displays the first 100 palindromic prime numbers. Display 10 numbers per line and align the numbers properly, as follows:



* 1. (Turtle: filled rectangle and circle) Write the following functions that fill a rectangle with the specified color, center, width, and height, and a circle with the specified color, center, and radius.

# Fill a rectangle

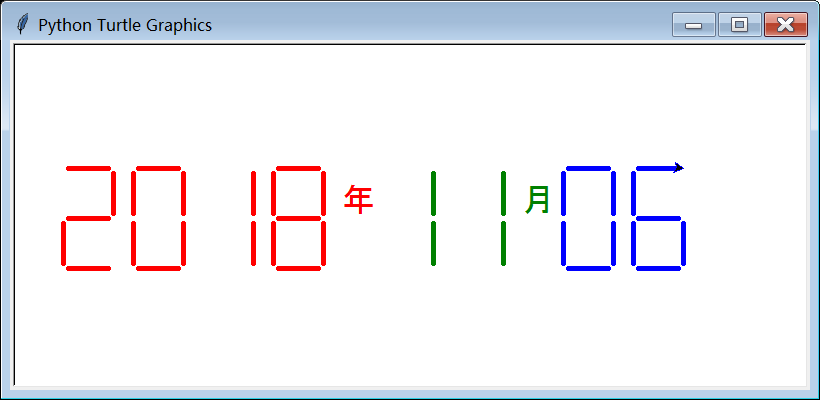
def drawRectangle(color = "black",x = 0, y = 0, width = 30, height = 30):

# Fill a circle

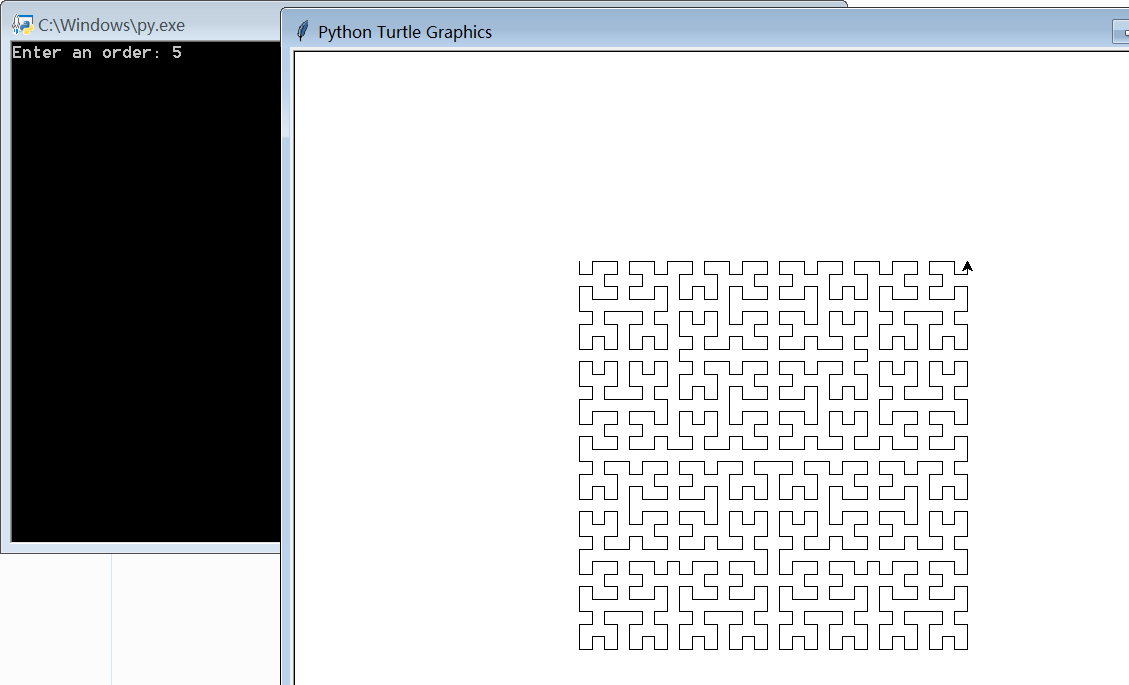
def drawCircle(color = "black", x = 0, y = 0, radius = 50):

Write a test program that draw a rectangle or a circle according to the specified color, center, width and height or radius.

* 1. (Seven-segment Indicator) Write a program that display the current date as follows:   
     (Hint: Turtle and Datetime module should be imported)



* 1. The Hilbert curve, first described by German mathematician David Hilbert in 1891, is a space-filling curve that visits every point in a square grid with a size of or any other power of 2. Write a program that using Turtle, prompt the user to enter the order and display a Hilbert curve for the specified order.



1. Code list

2.2.py

def isPalindromicPrime(x):

for i in range(2, x):

if x % i == 0:

return False

y = []

while x >= 1:

y.append(x % 10)

x //= 10

print(y)

l = len(y)

for i in range(l):

if y[i] != y[l-i-1]:

return False

return True

cnt = 0

x = 101\*[0]

number = 2

while cnt <= 100:

if isPalindromicPrime(number):

x[cnt] = number

cnt += 1

number += 1

for i in range(100):

print("%-8d" % x[i], end="")

if (i+1) % 10 == 0:

print()

2.3.py

import turtle

def drawRectangle(color="black",x=0,y=0,width=30,height=30):

turtle.color(color)

turtle.begin\_fill()

turtle.penup()

turtle.goto((x+width/2,y+width/2))

turtle.pendown()

turtle.right(90)

turtle.forward(height)

turtle.right(90)

turtle.forward(width)

turtle.right(90)

turtle.forward(height)

turtle.right(90)

turtle.forward(width)

turtle.end\_fill()

def drawCircle(color="black",x=0,y=0,radius=50):

turtle.begin\_fill()

turtle.color(color)

turtle.penup()

turtle.goto((x,y-radius))

turtle.pendown()

turtle.circle(radius)

turtle.end\_fill()

drawCircle(color="red")

drawRectangle()

2.4.py

import time

import turtle

turtle.pensize(4)

months = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

monthname = ["Jan.", "Feb.", "Mar.", "Apr.", "May.",

"Jun.", "Jul.", "Aug.", "Sep.", "Oct.", "Nov.", "Dec."]

currentTime = time.time()

totalSeconds = int(currentTime)

currentSecond = totalSeconds % 60

totalMinutes = totalSeconds//60

currentMinute = totalMinutes % 60

totalHours = totalMinutes//60

currentHour = totalHours % 24

totalDays = totalHours//24

def isleapYear(year):

if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):

return True

else:

return False

def getYearMonthDay(days):

year = 1969

while days > 0:

year += 1

if isleapYear(year):

days -= 366

else:

days -= 365

if isleapYear(year):

days += 366

else:

days += 365

month = 0

while days > 0:

month += 1

if month == 2:

if isleapYear(year):

days -= 29

else:

days -= 28

else:

days -= months[month-1]

if month == 2:

if isleapYear(year):

days += 29

else:

days += 28

else:

days += months[month-1]

return year, month, days

def draw\_number(number, x, y, length):

if number == 0 or number == 2 or number == 3 or number == 5 or number == 6 or number == 7 or number == 8 or number == 9:

turtle.penup()

turtle.setheading(0)

turtle.goto((x, y))

turtle.pendown()

turtle.forward(length)

if number == 2 or number == 3 or number == 4 or number == 5 or number == 6 or number == 8 or number == 9:

turtle.penup()

turtle.setheading(0)

turtle.goto((x, y-length))

turtle.pendown()

turtle.forward(length)

if number == 2 or number == 3 or number == 5 or number == 6 or number == 8 or number == 0 or number == 9:

turtle.penup()

turtle.setheading(0)

turtle.goto((x, y-length\*2))

turtle.pendown()

turtle.forward(length)

if number == 4 or number == 5 or number == 6 or number == 8 or number == 9 or number == 0:

turtle.penup()

turtle.setheading(0)

turtle.right(90)

turtle.goto((x, y))

turtle.pendown()

turtle.forward(length)

if number == 0 or number == 2 or number == 6 or number == 8:

turtle.penup()

turtle.setheading(0)

turtle.right(90)

turtle.goto((x, y-length))

turtle.pendown()

turtle.forward(length)

if number == 0 or number == 1 or number == 2 or number == 3 or number == 4 or number == 7 or number == 8 or number == 9:

turtle.penup()

turtle.setheading(0)

turtle.right(90)

turtle.goto((x+length, y))

turtle.pendown()

turtle.forward(length)

if number == 0 or number == 1 or number == 3 or number == 4 or number == 5 or number == 6 or number == 7 or number == 9 or number == 8:

turtle.penup()

turtle.setheading(0)

turtle.right(90)

turtle.goto((x+length, y-length))

turtle.pendown()

turtle.forward(length)

def drawTime(year, month, day):

x = -100

y = 30

turtle.pencolor("red")

turtle.penup()

turtle.goto((-50,0))

turtle.write("年",font=('Arial', 20, 'normal'))

while year >= 1:

draw\_number(year % 10, x, y, 30)

year //= 10

x -= 50

x=50

turtle.pencolor("green")

turtle.penup()

turtle.goto((100,0))

turtle.write("月",font=('Arial', 20, 'normal'))

cnt=0

while month >= 1:

draw\_number(month % 10, x, y, 30)

month //= 10

x -= 50

cnt+=1

if cnt<=1:

draw\_number(0,x,y,30)

cnt=0

x=200

turtle.pencolor("blue")

turtle.penup()

turtle.goto((250,0))

turtle.write("日",font=('Arial', 20, 'normal'))

while day >= 1:

draw\_number(day % 10, x, y, 30)

day //= 10

x -= 50

cnt+=1

if cnt<=1:

draw\_number(0,x,y,30)

turtle.speed(30)

year, month, day = getYearMonthDay(totalDays)

print(year,day,month)

drawTime(year, month, day)

turtle.hideturtle()

turtle.done()

2.5.py

import turtle

# 0上1下2左3右 4is not

turtle.speed(30)

def drawConncetLine(degree, x, y, direction, length):

if degree != 0:

return

turtle.penup()

turtle.goto((x, y))

turtle.setheading(0)

turtle.pendown()

if direction == 0:

turtle.left(90)

turtle.forward(length)

elif direction == 1:

turtle.right(90)

turtle.forward(length)

elif direction == 2:

turtle.left(180)

turtle.forward(length)

elif direction == 3:

turtle.forward(length)

print(direction)

def drawSquare(x1, y1, x2, y2, direction, length):

turtle.penup()

turtle.setheading(0)

turtle.goto((x1, y1))

if direction == 0:

turtle.right(90)

turtle.pendown()

turtle.forward(length)

if x1 < x2:

turtle.left(90)

turtle.forward(length)

turtle.left(90)

turtle.forward(length)

else:

turtle.right(90)

turtle.forward(length)

turtle.right(90)

turtle.forward(length)

elif direction == 1:

turtle.left(90)

turtle.pendown()

turtle.forward(length)

if x1 < x2:

turtle.right(90)

turtle.forward(length)

turtle.right(90)

turtle.forward(length)

else:

turtle.left(90)

turtle.forward(length)

turtle.left(90)

turtle.forward(length)

elif direction == 2:

turtle.pendown()

turtle.forward(length)

if y1 > y2:

turtle.right(90)

turtle.forward(length)

turtle.right(90)

turtle.forward(length)

else:

turtle.left(90)

turtle.forward(length)

turtle.left(90)

turtle.forward(length)

elif direction == 3:

turtle.left(180)

turtle.pendown()

turtle.forward(length)

if y1 < y2:

turtle.right(90)

turtle.forward(length)

turtle.right(90)

turtle.forward(length)

else:

turtle.left(90)

turtle.forward(length)

turtle.left(90)

turtle.forward(length)

def drawHilbertCurve(x1, y1, x2, y2, degree, direction, length, connect):

l = length/4

if degree == 0:

drawSquare(x1, y1, x2, y2, direction, length)

else:

# 0->左上上右 1->

if direction == 0:

if x1 < x2:

drawHilbertCurve(x1-l, y1+l, x1-l, y1-l,

degree-1, 2, length/2, 1)

drawConncetLine(degree-1, x1-l, y1-l, 1, length/2)

drawHilbertCurve(x1-l, y1-length+l, x1+l, y1 -

length+l, degree-1, direction, length/2, 3)

drawConncetLine(degree-1, x1+l, y1 - length+l, 3, length/2)

drawHilbertCurve(x2-l, y2-length+l, x2+l, y2 -

length+l, degree-1, direction, length/2, 0)

drawConncetLine(degree-1, x2+l, y2-length+l, 0, length/2)

drawHilbertCurve(x2+l, y2-l, x2+l, y2+l,

degree-1, 3, length/2, connect)

drawConncetLine(degree-1, x2+l, y2+l, connect, length/2)

else:

drawHilbertCurve(x1+l, y1+l, x1+l, y1-l,

degree-1, 3, length/2, 1)

drawConncetLine(degree-1, x1+l, y1-l, 1, length/2)

drawHilbertCurve(x1+l, y1-length+l, x1-l, y1 -

length+l, degree-1, direction, length/2, 2)

drawConncetLine(degree-1, x1-l, y1-length+l, 2, length/2)

drawHilbertCurve(x2+l, y2-length+l, x2-l, y2 -

length+l, degree-1, direction, length/2, 0)

drawConncetLine(degree-1, x2-l, y2-length+l, 0, length/2)

drawHilbertCurve(x2-l, y2-l, x2-l, y2+l,

degree-1, 2, length/2, connect)

drawConncetLine(degree-1, x2-l, y2+l, connect, length/2)

elif direction == 2:

if y1 > y2:

drawHilbertCurve(x1-l, y1+l, x1+l, y1+l,

degree-1, 0, length/2, 3)

drawConncetLine(degree-1, x1+l, y1+l, 3, length/2)

drawHilbertCurve(x1+length-l, y1+l, x1+length-l,

y1-l, degree-1, direction, length/2, 1)

drawConncetLine(degree-1, x1+length-l, y1-l, 1, length/2)

drawHilbertCurve(x2+length-l, y2+l, x2+length-l,

y2-l, degree-1, direction, length/2, 2)

drawConncetLine(degree-1, x2+length-l, y2-l, 2, length/2)

drawHilbertCurve(x2+l, y2-l, x2-l, y2-l,

degree-1, 1, length/2, connect)

drawConncetLine(degree-1, x2-l, y2-l, connect, length/2)

else:

drawHilbertCurve(x1-l, y1-l, x1+l, y1-l,

degree-1, 1, length/2, 3)

drawConncetLine(degree-1, x1+l, y1-l, 3, length/2)

drawHilbertCurve(x1+length-l, y1-l, x1+length-l,

y1+l, degree-1, direction, length/2, 0)

drawConncetLine(degree-1, x1+length-l, y1+l, 0, length/2)

drawHilbertCurve(x2+length-l, y2-l, x2+length-l,

y2+l, degree-1, direction, length/2, 2)

drawConncetLine(degree-1, x2+length-l, y2+l, 2, length/2)

drawHilbertCurve(x2+l, y2+l, x2-l, y2+l,

degree-1, 0, length/2, connect)

drawConncetLine(degree-1, x2-l, y2+l, connect, length/2)

elif direction == 1:

if x1 < x2:

drawHilbertCurve(x1-l, y1-l, x1-l, y1+l,

degree-1, 2, length/2, 0)

drawConncetLine(degree-1, x1-l, y1+l, 0, length/2)

drawHilbertCurve(x1-l, y1+length-l, x1+l, y1 +

length-l, degree-1, direction, length/2, 3)

drawConncetLine(degree-1, x1+l, y1+length-l, 3, length/2)

drawHilbertCurve(x2-l, y2+length-l, x2+l, y2 +

length-l, degree-1, direction, length/2, 1)

drawConncetLine(degree-1, x2+l, y2+length-l, 1, length/2)

drawHilbertCurve(x2+l, y2+l, x2+l, y2-l,

degree-1, 3, length/2, connect)

drawConncetLine(degree-1, x2+l, y2-l, connect, length/2)

else:

drawHilbertCurve(x1+l, y1-l, x1+l, y1+l,

degree-1, 3, length/2, 0)

drawConncetLine(degree-1, x1+l, y1+l, 0, length/2)

drawHilbertCurve(x1+l, y1+length-l, x1-l, y1 +

length-l, degree-1, direction, length/2, 2)

drawConncetLine(degree-1, x1-l, y2-l+length, 2, length/2)

drawHilbertCurve(x2+l, y2+length-l, x2-l, y2 +

length-l, degree-1, direction, length/2, 1)

drawConncetLine(degree-1, x2-l, y2-l+length, 1, length/2)

drawHilbertCurve(x2-l, y2+l, x2-l, y2-l,

degree-1, 2, length/2, connect)

drawConncetLine(degree-1, x2-l, y2-l, connect, length/2)

elif direction == 3:

if y1 > y2:

drawHilbertCurve(x1+l, y1+l, x1-l, y1+l,

degree-1, 0, length/2, 2)

drawConncetLine(degree-1, x1-l, y1+l, 2, length/2)

drawHilbertCurve(x1-length+l, y1+l, x1-length+l,

y1-l, degree-1, direction, length/2, 1)

drawConncetLine(degree-1, x1-length+l, y1-l, 1, length/2)

drawHilbertCurve(x2-length+l, y2+l, x2-length+l,

y2-l, degree-1, direction, length/2, 3)

drawConncetLine(degree-1, x2-length+l, y2-l, 3, length/2)

drawHilbertCurve(x2-l, y2-l, x2+l, y2-l,

degree-1, 1, length/2, connect)

drawConncetLine(degree-1, x2+l, y2-l, connect, length/2)

else:

drawHilbertCurve(x1+l, y1-l, x1-l, y1-l,

degree-1, 1, length/2, 2)

drawConncetLine(degree-1, x1-l, y1-l, 2, length/2)

drawHilbertCurve(x1-length+l, y1-l, x1-length+l,

y1+l, degree-1, direction, length/2, 0)

drawConncetLine(degree-1, x1-length+l, y1+l, 0, length/2)

drawHilbertCurve(x2-length+l, y2-l, x2-length+l,

y2+l, degree-1, direction, length/2, 3)

drawConncetLine(degree-1, x2-length+l, y2+l, 3, length/2)

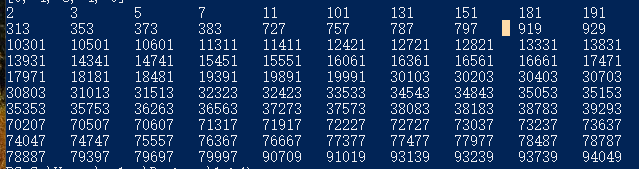
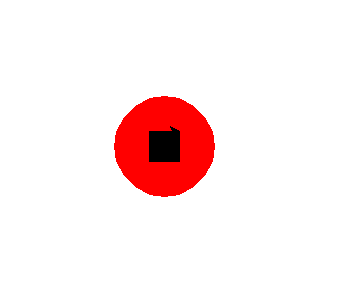
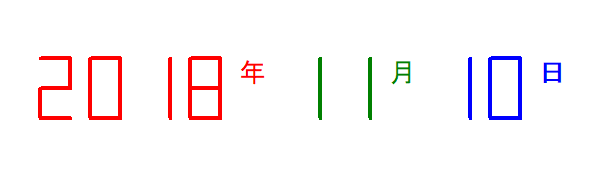
drawHilbertCurve(x2-l, y2+l, x2+l, y2+l,

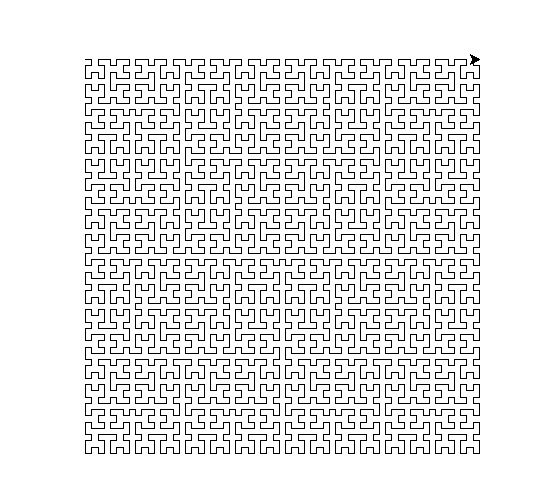
degree-1, 0, length/2, connect)

drawConncetLine(degree-1, x2+l, y2+l, connect, length/2)

drawHilbertCurve(-100, 100, 100, 100, 4, 0, 200, 4)

turtle.done()

1. Output



1. Analysis and conclusions

Recursion in this lab is very important. And the last problem cost me a lot of time to solve. May be my solution is very complex. But it’s true and it’s a way to solve it by use recursion.

In another way, I have learned that use the code that I coded before is very important , it can simplify your work and make your code more easier.