Chapter5_Exercises

May 28, 2017

1 Chapter 5 Exercises

1.1 Exercise 5.1

The time module provides a function, also named time, that returns the current Greenwich Mean Time in "the epoch", which is an arbitrary time used as a reference point. On UNIX systems, the epoch is 1 January 1970.

Write a script that reads the current time and converts it to a time of day in hours, minutes, and seconds, plus the number of days since the epoch.

```
In [18]: import time
         timey = time.time()

         hours = timey / 3600
         minutes = timey / 60
         seconds = timey

         daysSince = timey / (3600*24)

         print('Time() returns {:.2f}'.format(timey))
         print('Days since the epoch: {:.2f}'.format(daysSince))
         print('Hours since the epoch: {:.2f}'.format(hours))
         print('Minutes since the epoch: {:.2f}'.format(minutes))
         print('Seconds since the epoch: {:.2f}'.format(seconds))
```

```
Time() returns 1496001224.27
Days since the epoch: 17314.83
Hours since the epoch: 415555.90
Minutes since the epoch: 24933353.74
Seconds since the epoch: 1496001224.27
```

1.2 Exercise 5.2

Write a function named check_fermat that takes four parameters—a, b, c and n—and checks to see if Fermat's theorem holds. If n is greater than 2 and a n + b n = c n the program should print, "Holy smokes, Fermat was wrong!" Otherwise the program should print, "No, that doesn't work."

Write a function that prompts the user to input values for a, b, c and n, converts them to integers, and uses check_fermat to check whether they violate Fermat's theorem.

```
In [27]: def check_fermat(a,b,c,n):
             if n > 2 and a**n+b**n==c**n:
                 print("HOLY SMOKES, FERMAT was WRONG!")
             else:
                 print("...Dang, that doesn't work")

         def fermatUserCheck():
             a = int(input('Input a value for a: '))
             b = int(input('Input a value for b: '))
             c = int(input('Input a value for c: '))
             n = int(input('Input a value for n: '))
             check_fermat(a,b,c,n)
```

```
In [29]: fermatUserCheck()

Input a value for a: 3
Input a value for b: 4
Input a value for c: 5
Input a value for n: 123
...Dang, that doesn't work
```

Nuts.

1.3 Exercise 5.3

If you are given three sticks, you may or may not be able to arrange them in a triangle. For example, if one of the sticks is 12 inches long and the other two are one inch long, you will not be able to get the short sticks to meet in the middle. For any three lengths, there is a simple test to see if it is possible to form a triangle:

If any of the three lengths is greater than the sum of the other two, then you cannot form a triangle. Otherwise, you can. (If the sum of two lengths equals the third, they form what is called a "degenerate" triangle.)

Write a function named is_triangle that takes three integers as arguments, and that prints either "Yes" or "No", depending on whether you can or cannot form a triangle from sticks with the given lengths.

Write a function that prompts the user to input three stick lengths, converts them to integers, and uses is_triangle to check whether sticks with the given lengths can form a triangle

```
In [32]: def is_triangle(a,b,c):
             if c < a+b and a < b+c and b < c+a:
                 print('Yes')
             else:
                 print('No')
```

```
        def triangleUserCheck():
            a = int(input('Input a value for a: '))
            b = int(input('Input a value for b: '))
            c = int(input('Input a value for c: '))
            is_triangle(a,b,c)

        triangleUserCheck()
```

```
Input a value for a: 30
Input a value for b: 1
Input a value for c: 1
No
```

1.4 Exercise 5.4

What is the output of the following program?

```
In [33]: def recurse(n, s):
             if n == 0:
                 print(s)
             else:
                 recurse(n-1, n+s)

         recurse(3, 0)
```

```
6
```

What would happen if you called this function like this: recurse(-1, 0)?

```
In [59]: recurse(-1,0)
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)

<ipython-input-59-5ee6508d2be3> in <module>()
----> 1 recurse(-1,0)


<ipython-input-33-e1dc0aea2150> in recurse(n, s)
      3         print(s)
      4     else:
----> 5         recurse(n-1, n+s)
      6
```

```
      7 recurse(3, 0)


    ... last 1 frames repeated, from the frame below ...


    <ipython-input-33-e1dc0aea2150> in recurse(n, s)
      3         print(s)
      4     else:
----> 5         recurse(n-1, n+s)
      6
      7 recurse(3, 0)


    RecursionError: maximum recursion depth exceeded in comparison
```

Write a docstring that explains everything someone would need to know in order to use this function (and nothing else).

```
In [58]: """
         Computes the nth triangle number with an additional s offset added
         n must be an integer, s may be a float/double or int
         """

Out[58]: '\nComputes the nth triangle number with an additional s offset added\nn m
```