Lab 7

June 25, 2024

Dylan Liesenfelt

1 Exercise:

Write a function to calculate the area of a triangle given its three sides a, b, and c using Heron's formula.

Input: a, b, c

Output: area

[ ]:
```
import math

a = 7; b = 11; c = 5 # Setting variables for side lengths, could also use input() to take user inputs, but to keep things short and sweet hardcoding my variables.
# Values of a, b, and c must be less than the sum of the other two variables. Ex b can not = 12 if a = 7 and c = 5, a+c = 12.
# If this function were to take in user input, would have to have input validation to to account for this.

def areaOfTriangle(a,b,c): # Defining the function to calc area of triangle with the parameters of a,b,c that we will pass later
    s = (a + b + c) / 2 # Assigning value to var s (semi-perimeter in Herons formula) = 1/2 (a + b + c)
    return math.sqrt(s * (s - a) * (s - b) * (s - c)) # Herons formula A = sqrt of s (s-a)(s-b)(s-c), in this case we are just returning the value of the formula to the function.

area = areaOfTriangle(a,b,c) # Calling my function with the parameters of are vars a, b, and c, that we defined earlier, and assigning its returned value to variable 'area'.

print(f'The area of your triangle is: {area:.4f} units') # Printing the result to within four decimal places
```

```
The area of your triangle is: 12.9687 units
```

2 Exercise:

Implement a function to calculate the future value of monthly savings replaced from an expense into a savings account over x years at an annual interest rate of i, compounded monthly.

Input: Monthly savings amount (P), annual interest rate as decimal (i), number of years (x)

Output: Future value (VP)

Tip: You can use the formula FV = P * (((1 + i/12) ** (12*x) - 1) / (i/12))

[ ]:
```
p = 500; i = 0.05; x = 10 # Hardcoding variables once again.

def calcFutureValue(p,i,x): # taking in arguments for the monthly contribution, interest rate, and time in years
    return p * (((1 + i / 12) ** (12 * x) - 1) / (i / 12)) # Returning the value of the given calculation using our parameters from the given formula.

fv = calcFutureValue(p,i,x) # Assigning that value to var fv

print(f'Monthly savings of ${p:,.2f}, Interest Rate of {i * 100:.2f}% over {x} years.\nThe future value of your account is: ${fv:,.2f}') # Printing out the values of our vars and the future value.
```

```
Monthly savings of $500.00, Interest Rate of 5.00% over 10 years.
The future value of your account is: $77,641.14
```

3 Exercise:

Create a function that estimates the number of communicative extraterrestrial civilizations in the Milky Way galaxy based on the Drake Equation parameters.

Input: Use input statements to ask the user for: R_star, f_p, n_e, f_l, f_i, f_c, L

Output: Number of communicative extraterrestrial civilizations in our galaxy (N)

[ ]:
```
# Taking in user inputs
R_star = int(input('Stars formed per year in galaxy: '))
f_p = float(input('The fraction of those stars that have planets: '))
n_e = int(input('The average number of those planets that can support life: '))
f_l = float(input('The fraction of those planets that will actually develope life: '))
f_i = float(input('The fraction of those planets that will go on to develope civilization: '))
f_c = float(input('The fraction of those civilizations that develope communicative technologies detectable from space: '))
L = int(input('The length of time those civilizations release detectable signals'))
```

```
def calcDrakeFormula(R, p, e, l, i, c, L): # Define our function and assign␣
 ↪parameters that we will pass in later from our user inputs
    return (R * p * e * l * i * c * L) # Drakes equation N = R* x fp x ne x fl␣
 ↪x fi x fc x L

N = calcDrakeFormula(R_star, f_p, n_e, f_l, f_i, f_c, L) # Call the function␣
 ↪and assign its value to N

print(f'Number of communicative extraterrestrial civilizations in our galaxy:␣
 ↪{N}')
# I will be using the following inputs R* = 1, fp = 0.3, ne = 3, fl = 0.1, fi =␣
 ↪0.1, fc = 0.2, L = 10000.
```

```
Number of communicative extraterrestrial civilizations in our galaxy: 18.0
```

4 Exercise:

Develop a function that reads a text file and returns the total word count.

Input: filename

Output: total word count

[ ]:
```
filePath = input(r'Enter the file path to the file you wish read: ') # Taking␣
 ↪user input for where on system file is stored

def countWords(filePath): # Def function with param for the future file path
    with open(filePath, mode='r') as file: # Opening passed file path and␣
 ↪assigning the object to var file

        text = file.read() # Reading the object and assigning the new string␣
 ↪value to var text
        text.strip().split() # Using built in methods to remove white space and␣
 ↪break string into separate words instead of one long string

        wordCount = 0 # Defining the counter that will hold the word count␣
 ↪value of the file
        for word in text: # For loop, will iterate for each word in the text␣
 ↪string
            wordCount += 1 # for each word encountered during iteration wil add␣
 ↪1 to the value of our counter var
    return wordCount # Finally return the value of our word count to the␣
 ↪function.

finalCount = countWords(filePath)

print(f'The number of words in the given file is: {finalCount:,} words')
```

```
# As with tradition (and because iI already have it saved in this directory), I␣
 ↪have used the Bee Movie script once again for this exercise
```

```
The number of words in the given file is: 86,050 words
```

5 Exercise:

- (a) Develop a function that calculates how many Rubik's cubes fit in a box of dimensions x, y, z.

Input: e, x, y, z (e: length dimension of the Rubik cube)

Output: number of cubes

- (b) Develop a function that calculates how many boxes of dimensions x, y, z fit into a standard 40-foot-long shipping container.

Input: x, y, z

Output: number of boxes

- (c) Develop a function that integrates the previous two functions to find out how many Rubik's cubes fit in a large ship that holds x containers.

Input: x containers

Output: number of cubes

[ ]:
```
# a
def rubikCubeCubed(e,x,y,z):
    # Calc how many cubes can fit on each dimension of our box (Length, Width,␣
 ↪Height)
    d1 = (x//e) # Using floor division because you can not have something like␣
 ↪0.5 cubes
    d2 = (y//e)
    d3 = (z//e)
    return (d1 * d2 * d3) # Now we have number of cubes that can fit in each␣
 ↪dimension we can get the 'pseudo-volume', which will be our cubes that fill␣
 ↪the box

numberOfCubes = rubikCubeCubed(3, 12, 20, 40) # Hardcoding parameters for ease,␣
 ↪just assume units are inches
print(f'{numberOfCubes} Rubik Cubes can fit in a box {12} x {20} x {40} box')
```

```
312 Rubik Cubes can fit in a box 12 x 20 x 40 box
```

[ ]:
```
# b
def boxesInShippingBox(x,y,z):
```

```
    # Same idea for previous function but in this case we have a rectangle␣
 ↪instead of cube, assume measurements in meters
    length = (12.19 // x) # Calc how many boxes fit along the length dimension,␣
 ↪Length = 40ft 12.2 meters
    width = (2.44 // y) # Width = 8ft or 2.44 meters
    height = (2.59 // z) # Height = 8ft 6 in or 2.59 meters
    return (length * width * height)

numberOfBoxes = int(boxesInShippingBox(0.3, 0.5, 1.0)) # Hardcoding parameters␣
 ↪for ease, assume meters for units
print(f'{numberOfBoxes} boxes of dimensions {0.3} x {0.5} x {1.0} meters can␣
 ↪fit in a standard size shipping container.')
```

320 of dimensions 0.3 x 0.5 x 1.0 meters can fit in a standard size shipping container.

[ ]:
```
# c
sideLengthOfRubikCube = 0.0762 # Standard rubik cube is 3x3x3in cube, 3in is 0.
 ↪0762 meters
containersOnShip = int(input('How many containers on this ship: '))

boxL = float(input('Length measurement of box in inches: ')) / 39.37 # Convert␣
 ↪in to meters by dividing by 39.37, Will be 18in
boxW = float(input('Width measurement of box in inches: ')) / 39.37# Will be␣
 ↪14in
boxH = float(input('Height measurement of box in inches: ')) / 39.37# Will be␣
 ↪12in


def numOfRubikOnShip(e,x,y,z,c):
    totalCubes = rubikCubeCubed(e, x, y, z) * boxesInShippingBox(x, y, z) * c #␣
 ↪(Number of Rubik Cubes per Box) * (Number of Boxes per Standard Container) *␣
 ↪(Number of Containers on the ship)
    print(f'If you have a ship with {c} standard size shipping␣
 ↪containers,\nfilled with boxes measuring {x*39.37:.0f} x {y*39.37:.0f} x␣
 ↪{z*39.37:.0f} inches,\nfilled with standard size Rubik Cubes,\nThat ship␣
 ↪would have {totalCubes:,.0f} Rubik cubes onboard!')

numOfRubikOnShip(sideLengthOfRubikCube, boxL, boxW, boxH, containersOnShip)
```

If you have a ship with 3000 standard size shipping containers,
filled with boxes measuring 18 x 14 x 12 inches,
filled with standard size Rubik Cubes,
That ship would have 359,424,000 Rubik cubes onboard!