Assignment 4B

LILLEBAELT ACADEMY OF
PROFESSIONAL HIGHER EDUCATION

Author
Martin Grønholdt
mart80c7@edu.eal.dk

Sunday 11 December 2016

Table of Contents

Introduction.....1
8. Sum of Numbers.....4
9. Drawing pattern.....6
10. Drawing pattern.....7
Christmas tree.....8
Conclusion.....13

Introduction

The programs in this hand-in uses most of the things learned so far. Some programs have not been split into functions, since they were so simple that it would be an exercise in over complication.

All files for this hand in are available at:
https://github.com/deadbok/eal_programming/tree/master/Assignment 4B

Error handling

All programs, requesting user input, handle bad input by asking the user, to use only the correct data type, where after it exits.

```
Enter the amount of a purchase: 2hjjhg

Please use only numbers.
```

Example output of a program when the user enters an incorrect value.

7. Pennies for Pay

This program calculate the amount of money a person would earn over a period of time if his or her salary is one penny the first day, exponentially rising each day.

prog7.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
'''
Name: Program 7
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2016-12-10)

Calculate the amount of money a person would earn over a period of time if his
or her salary is one penny the first day, exponentially rising each day.
'''


def grow_or_pay(pay=1, days=2, day=1, total=0):
    '''
    Calculate the amount of pay, doubling the salary each day.

    :param pay: The salary to base the calculation on.
    :param days: The number of days to calculate the salary for,
    :param day: The current day.
    :return: The total salary.
    '''
    # Print the current results.
    print('{:12.2f}\t|{:4d}\t|{:16.2f}'.format(pay, day, total))

    # Check if we're done.
    if days == day:
        return (pay)

    # Next step.
    pay *= 2
    total += pay
    day += 1

    # Call again.
    return (grow_or_pay(pay, days, day, total))


def main():
    '''
    Main entry point.
    '''
    # Get number of days, and salary from the user.
    days = 0
    pay = 0
    try:
        pay = float(input('Input the amount of salary per day: '))
        days = int(input('Input the amount of days: '))
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers.')
        exit(1)
```

```
    print('\nDaily salary\t| Day\t|\tTotal salary')
    print('-------------------------------------------')
    total_pay = grow_or_pay(pay, days)
    print('-------------------------------------------')
    total_pounds = int(total_pay / 100)
    total_pennies = total_pay - (total_pounds * 100)
    print('{:.0f} pounds and {:2.0f} pennies in {} days.'.format(total_pounds,
                                                                 total_pennies,
                                                                 days))


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

Result

```
Input the amount of salary per day: 1
Input the amount of days: 12

Daily salary    | Day   |       Total salary
-------------------------------------------
        1.00    |   1   |             0.00
        2.00    |   2   |             2.00
        4.00    |   3   |             6.00
        8.00    |   4   |            14.00
       16.00    |   5   |            30.00
       32.00    |   6   |            62.00
       64.00    |   7   |           126.00
      128.00    |   8   |           254.00
      256.00    |   9   |           510.00
      512.00    |  10   |          1022.00
     1024.00    |  11   |          2046.00
     2048.00    |  12   |          4094.00
-------------------------------------------
20 pounds and 48 pennies in 12 days.
```

Output of the program when run from the command line.

8. Sum of Numbers

This program calculate the sum of numbers input by the user. The program uses a list and the *sum()* function to simplify the code..

prog8.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
'''
Name: Program 8
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2016-12-10)

Calculate the sum of numbers input by the user.
'''


def main():
    '''
    Program main entry point.
    '''
    # List to keep the numbers enteret by the user.
    numbers = list()
    # Get numbers from the user.
    try:
        print('\nInput a series of numbers, end inputting by entering a ' +
              'negative number')
        number = float(input('Input a number: '))
        while number > -1:
            numbers.append(number)
            number = float(input('Input a number: '))
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers.')
        exit(1)

    print('The sum of all numbers entered is: {:0.2f}'.format(sum(numbers)))


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

Result

```
Input a series of numbers, end inputting by entering a negative number
Input a number: 1
Input a number: 2
Input a number: 3
Input a number: 4
Input a number: 5
Input a number: 6
Input a number: 7
Input a number: 8
Input a number: 9
Input a number: -1
The sum of all numbers entered is: 45.00
```

Output of the program.

9. Drawing pattern

This program uses nested loops to draw a pattern.

prog9.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
'''
Name: Program 9
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2016-12-10)

A program that uses nested loops to draw this pattern:

    *******
    ******
    *****
    ****
    ***
    **
    *
'''


def main():
    '''
    Program main entry point.
    '''
    # Loop for the height.
    for h in range(0, 7):
        # Loop for the width.
        for w in range(0, 7 - h):
            # Print the characters of the shape.
            print('*', end='')
        # New line.
        print()


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

Result

```
******
*****
****
***
**
*
```

Output of the program.

10. Drawing pattern

This program uses nested loops to draw a pattern.

prog10.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
'''
Name: Program 9
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2016-12-10)

A program that uses nested loops to draw this pattern:

    ##
    # #
    #  #
    #   #
    #    #
    #     #
'''


def main():
    '''
    Program main entry point.
    '''
    # Loop for the height
    for h in range(0, 6):
        # Print the leftmost line.
        print('#', end='')
        # Loop for the width.
        for w in range(0, h + 1):
            if w == h:
                # Print the rightmost line.
                print('#', end='')
            elif w < h:
                # Print the empty characters in between.
                print(' ', end='')
        print()


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

Result

```
# #
#  #
#   #
#    #
#     #
```

Output of the program when run from the command line.

Christmas tree

This program uses qute a bit of the Python language and some standard library functions to print a colourful Christmas tree. The colouring is done using an external library called *colorama*[1]. This library uses either ANSI calls on UNIX compatible terminals or Win32 calls on Windows. This functionality has not been verified, except on a Linux terminal.

christmas_tree.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Christmas Tree
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2016-12-11)

Print a Christmas tree.
"""
from random import randrange

from colorama import Fore


def stem(tree_width, stem_width):
    '''
    Print the stem of the Christmas tree.

    :param tree_width: Width of the christmas tree.
    :param stem_width: Width of the christmas tree stem.
    :return: Nothing.
    '''
    # Find the offset where the stem starts
    stem_start = int(tree_width / 2 - 1)
    for h in range(0, 2):
        # Loop until the end of the stem.
        for w in range(0, stem_start + stem_width):
            if w < stem_start:
                # Non stem position, print space.
                print(' ', end='')
            else:
                # Stem position, print #.
                print(Fore.RED + '#', end='')
        print()


def print_tree_char(dist):
    '''
    Print one character of the actual tree. Chooses randomly among the rest of
    the characters available.

    :param dist: List of characters and occurrences left.
    :return: The character printed.
    '''
    count = sum([c[1] for c in dist])
```

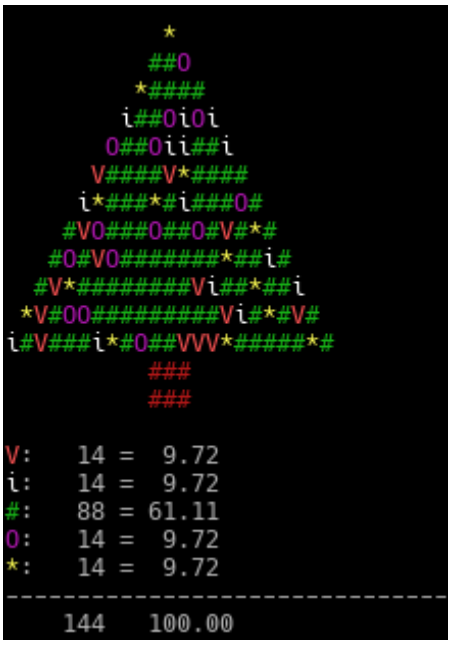
1 https://pypi.python.org/pypi/colorama

```
    # We're at the last character.
    if count == 1:
        i = 0
        # Don't waste time, just find it.
        while dist[i][1] == 0:
            i += 1

        # And print it.
        dist[i] = (dist[i][0], dist[i][1] - 1)
        print(dist[i][0])
    # Some other character in the tree.
    else:
        # Pick a random character among the rest needed.
        choice = randrange(0, count)

        # Find the choice by distributing them among all the possible characters
        # in the tree.
        i = 0
        while choice > sum([c[1] for c in dist[0:i + 1]]):
            i += 1

        # Bad choice, all used up.
        if dist[i][1] == 0:
            # Make another choice
            return (print_tree_char(dist))
        else:
            # Print the character.
            dist[i] = (dist[i][0], dist[i][1] - 1)
            print(dist[i][0], end='')

    return (dist[i][0])


def tree(width, height, stem_width, dist):
    '''
    Print the tree.

    :param height: Height, in characters, of the tree.
    :param stem_width: Width, in characters, of the tree.
    :param dist: Characters and the number to be used in the tree.
    :return:
    '''
    # Get the area.
    n_chars = 0
    for i in range(0, 12):
        n_chars += i * 2 + 1

    # Dictionary of chars, and how many have been used.
    use = dict()
    # Create a dictionary to keep a record of the number of used characters.
    for char in dist:
        # Add all characters to the dictionary with 0 used.
        use[char[0]] = 0

    # Find the middle of the tree.
    tree_middle = int(width / 2)
    # Row width of the top.
    row_width = 1
    # Loop from the top down.
    for h in range(0, height):
        # New line.
```

```
        print()
        # Loop from left to right.
        for w in range(0, tree_middle + row_width):
            # Print the filling spaces before the tree.
            if w < (tree_middle - int(row_width / 2)):
                print(' ', end='')
            # Print the top.
            elif row_width == 1:
                # A star at the top.
                i = 0
                while '*' not in dist[i][0]:
                    i += 1

                # Print the character.
                dist[i] = (dist[i][0], dist[i][1] - 1)
                print(dist[i][0], end='')
                # Add to used
                use[dist[i][0]] += 1
            # Print the rows of the tree itself.
            elif w <= (tree_middle + int(row_width / 2)):
                use[print_tree_char(dist)] += 1

        # Next row is wider.
        row_width += 2

    stem(width, stem_width)
    return(use)


def main():
    '''
    Program main entry point.
    '''
    # Height of the tree, and stem width.
    height = 1
    stem_width = 3

    try:
        height = int(input('Input the height of the christmas tree: '))
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers.')
        exit(1)

    # Get the number of characters in the tree.
    n_chars = 0
    for i in range(0, height):
        n_chars += i * 2 + 1
    width = i * 2 + 1

    # Set number of characters for each type of decorations.
    tree_chars = [
        (Fore.MAGENTA + 'O', int((n_chars / 100) * 10)),
        (Fore.LIGHTWHITE_EX + 'i', int((n_chars / 100) * 10)),
        (Fore.LIGHTRED_EX + 'V', int((n_chars / 100) * 10)),
        (Fore.LIGHTYELLOW_EX + '*', int((n_chars / 100) * 10)),
        (Fore.GREEN + '#', int((n_chars / 100) * 60))]

    # Fix rounding errors by adding more green.
    tree_chars[4] = (tree_chars[4][0], tree_chars[4][1] + n_chars -
                     sum([c[1] for c in tree_chars]))
```

```
    # Print the tree after a new line.
    print()
    use = tree(width, height, stem_width, tree_chars)

    # Print statistics after a new line.
    print()
    total_number = 0
    total_percent = 0
    for char, number in use.items():
        percent = number / (n_chars / 100)
        print('{}'.format(char) + Fore.WHITE +
              ': {:4} = {:5.2f}'.format(number, percent))
        total_number += number
        total_percent += percent
    print(Fore.WHITE + '------------------------------')
    print(Fore.WHITE + '\t{:4}\t{:5.2f}'.format(total_number, total_percent))


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

Result



A 12 line high Christmas tree generated by the program.

A 24 line high Christmas tree generated by the program.

Conclusion

Especially the Christmas tree challenge was a fun one, giving an opportunity to use a wide range of what makes Python a really good and easy programming language to use.