Assignment 3A

Control structures, if-else, elif

LILLEBAELT ACADEMY OF
PROFESSIONAL HIGHER EDUCATION

Author
Martin Grønholdt
mart80c7@edu.eal.dk

Sunday 20 November 2016

Table of Contents

Introduction

The programs in this hand-in uses control structures control program flow depending on certain conditions, being met. The control structures used in these programs use if statements. In a few places these if statements have been replaced with a look up table made using a dictionary.

Error handling

All programs handle bad input by asking the user, to use only numbers, 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.

1. Roman Numerals

This program convert the numbers in the range 1 to 10 to Roman numerals. It uses a dictionary to look up the Roman representation of the numeral.

prog1.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 1
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (19/11-2016)

Convert to Roman numerals.
'''


def to_roman(number=1):
    '''
    Convert a number in to a string with Roman numerals.

    :param number: Number to convert (between, and including, 1 and 10)
    :return: A string with the with the Roman numeral representation.
    '''
    # Look up table for each number.
    roman_look_up = {1: 'I', 2: 'II', 3: 'III', 4: 'IV', 5: 'V',
                     6: 'VI', 7: 'VII', 8: 'VIII', 9: 'IX', 10: 'X'}
    # Return the Roman numeral using the look up.
    return (roman_look_up[number])


def main():
    '''
    Main entry point.
    '''
    # Get the amount of kilometres from the user.
    try:
        number = int(input('Input a number in the range of 1 through 10: '))
        if (number < 1) or (number > 10):
            raise ValueError
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers in the range of 1 through 10.')
        exit(1)

    print('\n{} is "{}" in Roman numerals.'.format(number, to_roman(number)))


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

Result

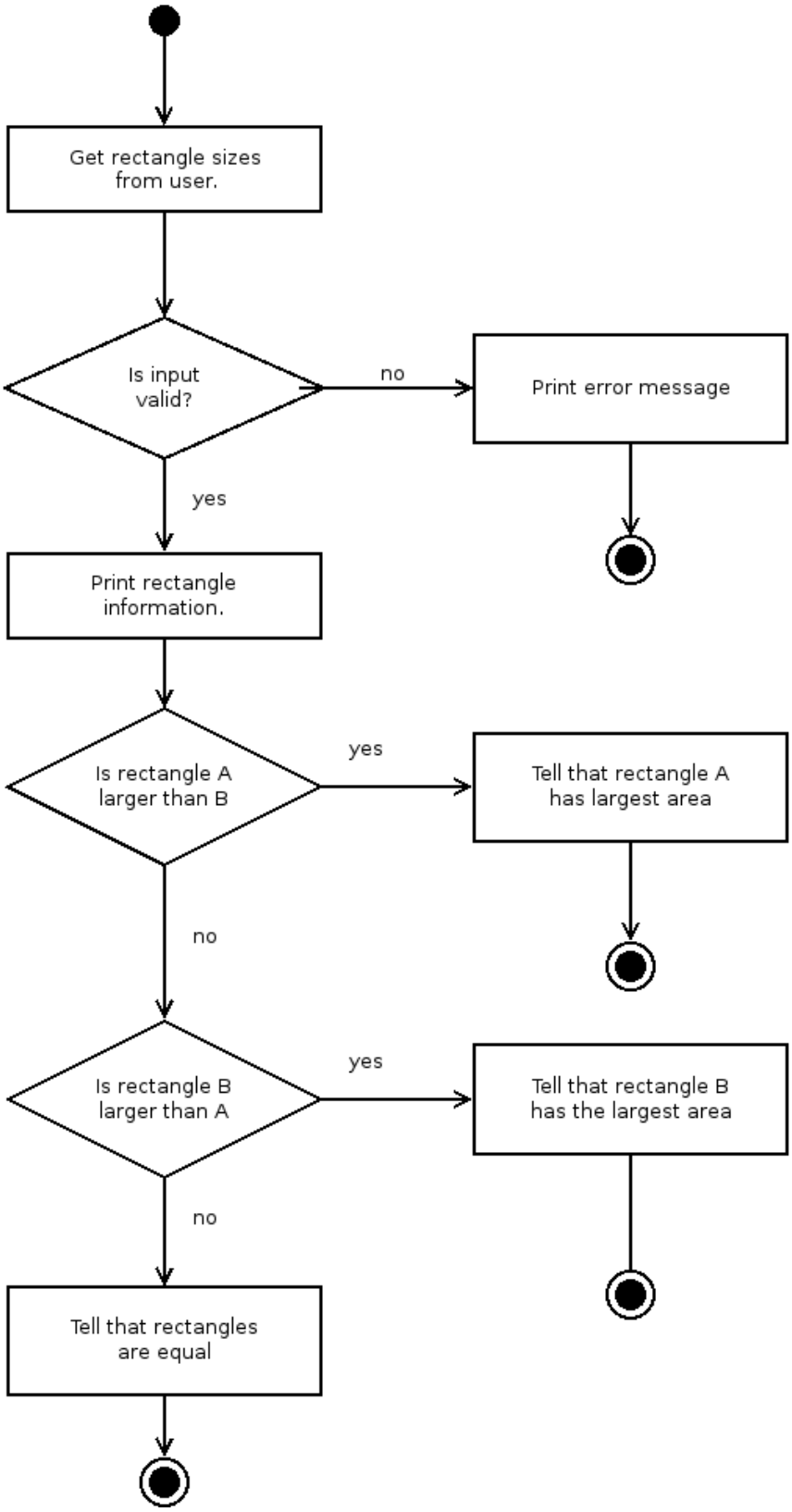
```
/usr/bin/python3.5 "/home/oblivion/Dokumenter/Skole/It-
et/2016/programming/Assignment 3A/prog1.py"
Input a number in the range of 1 through 10: 8

8 is "VIII" in Roman numerals.
```

Output of the program when run from the command line.

2. Areas of Rectangles

This program uses if statements to decide which area of two rectangles is the largest.



Activity diagram for the program.

prog2.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 2
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (19/11-2016)

Compare the area of two rectangles and tell which is the largest.
'''
def get_rectangle(name='no name'):
    '''
    Get the width an height of a rectangle from the user.

    :param name: Name of the rectangle that the values belong to.
    :return: Tuple (width, height)
    '''
    #Get width.
    print('Enter the width of rectangle {}: '.format(name), end='')
    width = float(input())
    #Get height.
    print('Enter the height of rectangle {}: '.format(name), end='')
    height = float(input())
    #Return the values as a tuple.
    return((width, height))


def get_rect_area(rect):
    '''
    Get the area of a rectangle.

    :param rect: Tuple (width, height) values for the rectangle.
    :return: Area of the rectangle.
    '''
    # Return the area.
    return(rect[0] * rect[1])

def print_rect(rect, name='no name'):
    '''
    Print the width, height, and area of a rectangle.

    :param rect: Tuple (width, height) values for the rectangle.
    '''
    print('Rectangle {}'.format(name) +
          ' has a width of {:0.2f}'. format(rect[0]) +
          ', a height of {:0.2f},'.format(rect[1]) +
          ' and an area of {}'.format(get_rect_area(rect)))


def main():
    '''
    Program main entry point.
    '''
    # The rectangle sizes from the user.
    try:
        rect_a = get_rectangle('A')
        rect_b = get_rectangle('B')
```

```
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers.')
        exit(1)

    # Print rectangle info.
    print()
    print_rect(rect_a, 'A')
    print_rect(rect_b, 'B')

    # Tell which rectangle has the largest area.
    if (get_rect_area(rect_a) > get_rect_area(rect_b)):
        print('Rectangle A has a larger area than rectangle B.')
    elif (get_rect_area(rect_a) < get_rect_area(rect_b)):
        print('Rectangle B has a larger area than rectangle A.')
    else:
        print('Rectangles A and B have equal areas.')


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

Result

```
Enter the width of rectangle A: 4
Enter the height of rectangle A: 5
Enter the width of rectangle B: 6
Enter the height of rectangle B: 7

Rectangle A has a width of 4.00, a height of 5.00, and an area of 20.0
Rectangle B has a width of 6.00, a height of 7.00, and an area of 42.0
Rectangle B has a larger area than rectangle A.
```

Output of the program when rectangle B is larger.

```
Enter the width of rectangle A: 9
Enter the height of rectangle A: 8
Enter the width of rectangle B: 7
Enter the height of rectangle B: 6

Rectangle A has a width of 9.00, a height of 8.00, and an area of 72.0
Rectangle B has a width of 7.00, a height of 6.00, and an area of 42.0
Rectangle A has a larger area than rectangle B.
```

Output of the program when rectangle A is larger.

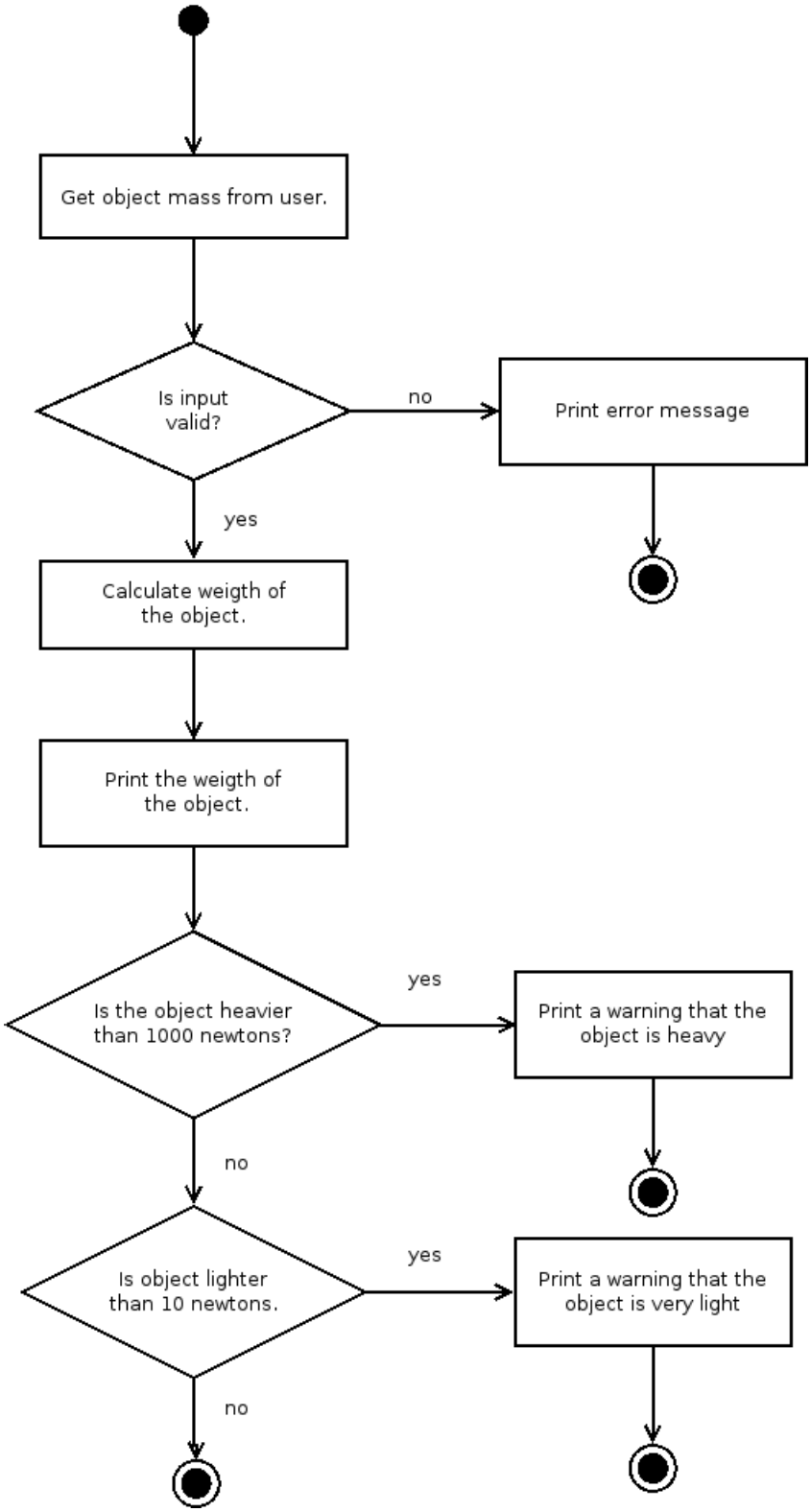
```
Enter the width of rectangle A: 2
Enter the height of rectangle A: 2
Enter the width of rectangle B: 2
Enter the height of rectangle B: 2

Rectangle A has a width of 2.00, a height of 2.00, and an area of 4.0
Rectangle B has a width of 2.00, a height of 2.00, and an area of 4.0
Rectangles A and B have equal areas.
```

Output of the program when rectangles are equal.

3. Mass and Weight

This program uses the is statement to warn if the weight of an object is within the expected range.



Activity diagram for the program

prog3.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 3
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (19/11-2016)

Calculate the weight of an object in newtons, warn if the object is lighter
than 10 newtons or heavier than 1000 newtons.
'''
def get_weight(mass=1):
    """
    Get the weight in newtons from the mass in kilograms.

    :param mass: Mass in kilograms.
    :return: Weight in newtons.
    """
    return(mass * 9.8)


def main():
    '''
    Program main entry point.
    '''
    # Get the amount of purchase from the user.
    try:
        mass = float(input('Enter the mass of the object in kilograms: '))
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers.')
        exit(1)

    # Calculate the weight.
    weight = get_weight(mass)
    # Print it.
    print('\nAn object with a mass of {:0.2f} kilograms'.format(mass) +
          ' has a weight of {:0.2f} newtons.'.format(weight))

    # Print a warning if object is larger or smaller than the min, max values.
    if weight > 1000:
        print('Warning: the object is heavier than 1000 newtons')
    if weight < 10:
        print('Warning: the object is lighter than 10 newtons')


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

Result

```
Enter the mass of the object in kilograms: 1

An object with a mass of 1.00 kilograms has a weight of 9.80 newtons.
Warning: the object is lighter than 10 newtons
```

Output of the program when the objects weight is in range.

```
Enter the mass of the object in kilograms: 10

An object with a mass of 10.00 kilograms has a weight of 98.00 newtons.
```

Output of the program when the objects weight is to light.

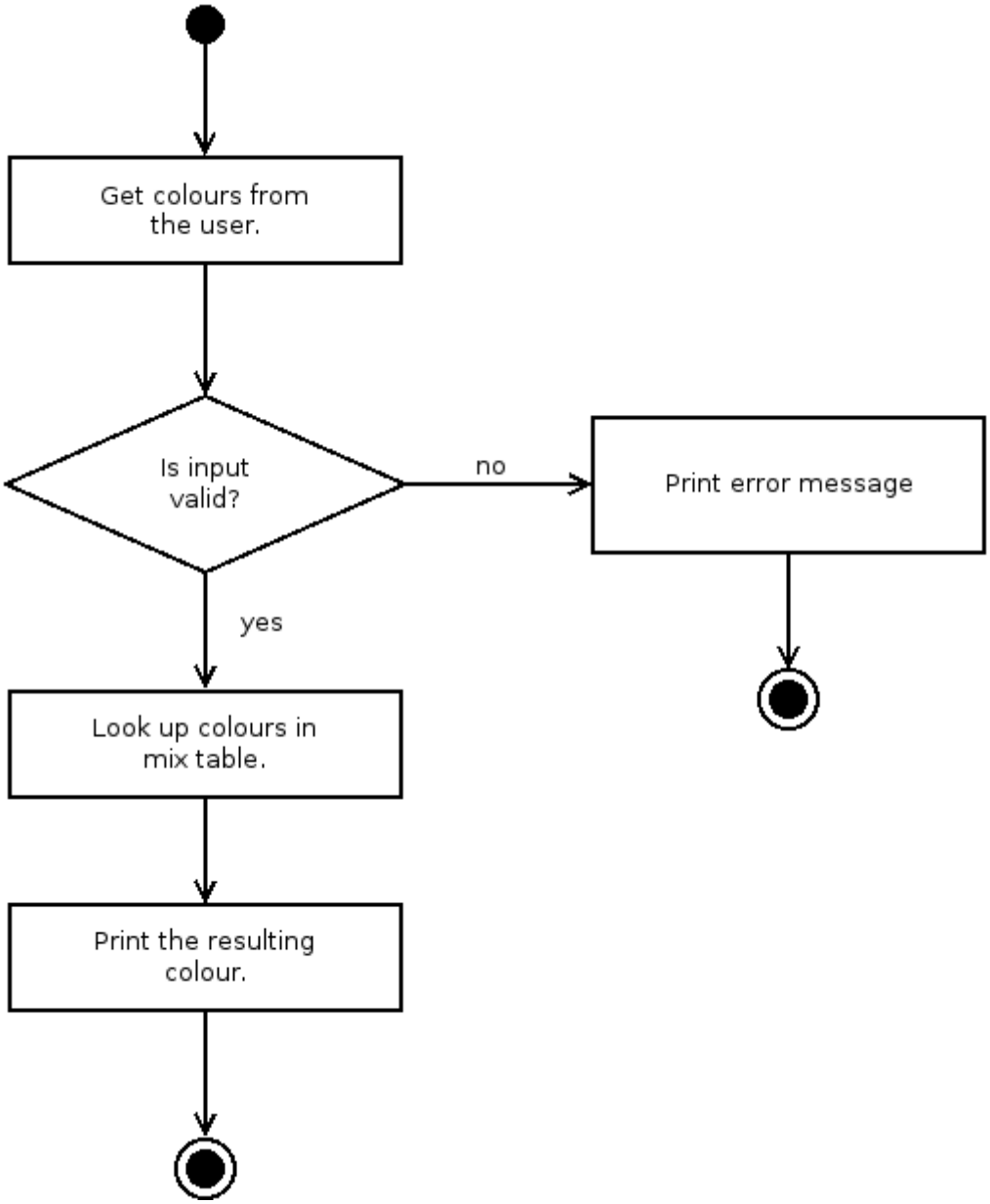
```
Enter the mass of the object in kilograms: 110

An object with a mass of 110.00 kilograms has a weight of 1078.00 newtons.
Warning: the object is heavier than 1000 newtons
```

Output of the program when the objects weight is to heavy.

5. Colour Mixer

This program uses a look up table in nested dictionaries, to find the secondary colour resulting from mixing to primary colours.



Activity diagram for the program.

prog5.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 5
Author: Martin Bo Kristensen Grønholdt.
```

```
Version: 1.0 (13/11-2016)

Get the name of a secondary colour created from mixing to primary colours.
"""


def get_color():
    """
    Get the name of a primare color from the user.

    :return: 'red', 'blue', or 'yellow'.
    """
    # Get the colour from the user, convert to lower case.
    color = input('Enter the name of a primary colour: ').lower()
    # Raise an exception if the input is invalid.
    if color not in ['red', 'blue', 'yellow']:
        raise ValueError
    # Return the color.
    return (color)


def mix_colors(first_color, second_color):
    """
    Return the secondary colour resulting from mixing two primare ones.

    :param first_color: The first primary colour.
    :param second_color: The second primary colour.
    :return: The resulting secondary colour.
    """
    # Create a look up table for all colour mixes.
    color_look_up = {'red':
                         {'red': 'red',
                          'blue': 'purple',
                          'yellow': 'orange'},
                     'blue':
                         {'red': 'purple',
                          'blue': 'blue',
                          'yellow': 'green'},
                     'yellow':
                         {'red': 'orange',
                          'blue': 'green',
                          'yellow': 'yellow'}
                     }
    # Return the secondary colour.
    return (color_look_up[first_color][second_color])


def main():
    """
    Program main entry point.
    """
    # Get the colours from the user.
    try:
        first_color = get_color()
        second_color = get_color()
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only "red", "blue", and "yellow".')
        exit(1)

    # Print the result
```

```
    print('\nMixing the primary colours ' + first_color + ' and ' +
          second_color + ' results in the secondary colour ' +
          mix_colors(first_color, second_color))


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

Result

```
Enter the name of a primary colour: Red
Enter the name of a primary colour: blue

Mixing the primary colours red and blue results in the secondary colour purple
```

Output of the program when run from the command line.

```
Enter the name of a primary colour: very red

Please use only "red", "blue", and "yellow".
```

Output of the program when given incorrect input.

Conclusion

In some programming languages there is a case statement that might have been fitting to use. In python the elif statement makes the same thing possible. I have used another approach, looking up the result in a predefined dictionary, after validating the keys. I think this implementation is easier to maintain.