Carleton University
SYSC 1005 – Introduction to Software Development – Fall 2014
Midterm Exam – October 23, 2014 - Solutions

Name:_____

Student Number:_____

INSTRUCTIONS:

1. This exam is closed book. Calculators are permitted.
2. Exam questions will not be explained, and no hints will be given. If you think something is unclear or ambiguous, make a reasonable assumption (one that does not contradict the question), write it at the start of your solution, and answer the question.
3. Remember, indentation is important in Python. You may want to draw light vertical lines to guide you when indenting your Python code.
4. You do not have to write comments in your Python code; however, feel free to add comments to clarify anything you think requires further explanation. Please don't write comments that state the obvious; for example, there is no need for comments similar to this one:

```
x = x + 1  # Add 1 to x
```

5. The crib sheet on last two pages of this question paper summarizes some Python functions that you may find useful.
6. All pages of this question paper, including the crib sheet, must be turned in.

Question 1 _____**/ 8**

Question 2 _____**/ 4**

Question 3 _____**/ 8**

Question 4 _____**/ 3**

Question 5 _____**/ 7**

Total _____**/ 30**

Question 1 [8 marks]

Here is an incomplete transcript from a session with the Python shell running Python 3.x. On the ruled lines, write the values that Python displays after it evaluates each expression. If Python displays an error message, write "Error". (You don't need to write the actual error message Python displays.) If nothing is displayed, write "Nothing". Use the symbol, ␣, to denote a space in the output; e.g., `SYSC␣1005`.

```
>>> 12 - 7 // 3 + 4 * 2
```

`18`_____ (1 mark)

```
>>> 26 // 4 * 4 + 26 / 4
```

`30.5`_____ (1 mark)

```
>>> 3 * 2 ** 2
```

`12`_____ (1 mark)

```
>>> 5(3 + 6)
```

`Error`_____ (1 mark)

```
>>> 17 % 3
```

`2`_____ (1 mark)

```
>>> tip = 20 * 0.15
```

`Nothing`_____ (1 mark)

```
>>> j = 5
>>> k = 3
>>> k = j, k
>>> k
```

`(5, 3)`____ (1 mark. 0.5 marks were awarded if the `()`'s were omitted, or if the pair of integers was enclosed in brackets - `[]` - or braces - `{}`.)

```
>>> j  # Assume this command is typed immediately after the
       # previous four commands
```

`5`_____ (1 mark)

Question 2 [4 marks]

Suppose this function has been defined and loaded into the Python interpreter:

```
from Cimpl import *

def mystery_filter(first, second):
    indigo = create_color(75, 0, 130)
    brown = create_color(165, 42, 42)

    second = first
    for x, y, col in first:
        set_color(first, x, y, indigo)

    for x, y, col in second:
        set_color(second, x, y, brown)
```

The following statements, when executed in the Python shell, run without error.

```
>>> img1 = load_image(choose_file())
>>> img2 = load_image(choose_file())
>>> mystery_filter(img1, img2)
>>> show(img1)
```

Circle the correct answer: The picture that was just displayed is

.

B. entirely brown.

```
>>> show(img2)
```

Circle the correct answer: The picture that was just displayed is

D. the original image that was bound to `img2`.

Each correct answer was worth 2 marks. Marks were not deducted for incorrect answers.

Explanation: Suppose the two calls to `load_image` bind `img1` to image A and `img2` to image B. When `mystery_filter` is called, parameter `first` is bound to the same image as `img1`; i.e., A, and parameter `second` is bound to the same image as `img2`; i.e., B.

The statement `second = first` does not make a copy of A and bind it to `second`. Instead, `second = first` rebinds `second` to the same image that is bound to `first`, i.e. A. This means that `first`, `second` and `img1` are all bound to the same image (A). Variable `img2` remains bound to B.

The first `for` loop changes the colour of all of the pixels in A to indigo, because `first` is bound to A. The second `for` loop changes the colour of all of the pixels in A to brown, because `second` is bound to A. When `mystery_filter` returns, `img1` is still bound to A (which has been recoloured brown) and `img2` is still bound to B (which has not been changed).

Question 3 [8 marks]

Suppose the following code has been typed in the Online Python Tutor editor:

```
def mystery(a, b):
    a = a + 2
    result = a * b
    b = b + 3          # Point B
    return result

a = 3
b = 7                  # Point A
result = mystery(a, a)
c = a * b              # Point C
```

When answering parts (a), (b) and (c), you do not need to show the bindings between function names and their function objects. Just show the bindings between variables and the associated data. Make sure you clearly indicate the frame where each variable is located.

(a) Draw a diagram similar to one that would be produced by the Online Python Tutor, depicting memory immediately after the assignment statement at Point A is executed, but before the next statement is executed.

This part was worth 1.5 marks. The The global frame (0.5 mark) has two variables, `a` and `b`, bound to integer objects 3 and 7 (0.5 mark each).

(b) Draw a diagram similar to one that would be produced by the Online Python Tutor, depicting memory immediately after the assignment statement at Point B is executed, but before the next statement is executed.

This part was worth 3.5 marks. The global frame (0.5 mark) has two variables, `a` and `b`, bound to integer objects 3 and 7 (0.5 mark each). The `mystery` frame (0.5 mark) has three variables, `a`, `b` and `result`, bound to 5, 6 and 15 (0.5 mark each).

(c) Draw a diagram similar to one that would be produced by the Online Python Tutor, depicting memory immediately after the assignment statement at Point C is executed.

This part was worth 3 marks. The global frame (0.5 mark) has four variables, `a`, `b`, `c` and `result`, bound to integer objects 3, 7, 21 and 15 (0.5 mark each). The `mystery` frame has been deleted (0.5 mark).

Checking the solutions: Type the code into the Online Python Tutor editor. Change one default option: change "inline primitives and nested objects (default)" to "render all objects on the heap". Now execute the code step-by-step, and observe the OPT diagrams immediately after the statements at Points A, B and C have been executed.

.

Question 4 [3 marks]

Here is a script containing the definitions of functions `sum_and_return` and `sum_and_print`:

```
def sum_and_return(a, b):
    return a + b

def sum_and_print(a, b):
    print(a + b)
```

After the script is loaded into Python, we type several commands in the shell. For each of the following commands, circle what Python will display:

```
>>> sum = sum_and_return(5, 3)
```

8 None **Nothing** An error message

```
>>> print(sum)
```

8 None Nothing An error message

```
>>> sum
```

8 None Nothing An error message

```
>>> sum = sum_and_print(5, 3)
```

8 None Nothing An error message

```
>>> print(sum)
```

8 **None** Nothing An error message

```
>>> sum
```

8 None **Nothing** An error message

Each correct answer was worth 0.5 marks. Marks were not deducted for incorrect answers.

Question 5 [7 marks]

Suppose we want to change an image so that every pixel with a red tone is converted to grayscale, but all other pixels are left unchanged. Here is an easy way to do this:

- First, calculate the pixel's brightness by calculating the average of its three components;
- Next, compare the pixel's red component to the brightness. If the red component is greater than 140% of the brightness, change the pixel's colour to its equivalent grayscale colour. For example, if a pixel's red component is 205 and the pixel's brightness is 110, the red component is greater than 154 (140% of the brightness), so the pixel's colour is changed.

Write a function called `red_to_grayscale` that implements this filter. The function header is:

```
def red_to_grayscale(img):
```

Parameter `img` is bound to the `Image` object that the filter will modify. Here is an example of how the function is called.:

```
>>> file = choose_file()
>>> img = load_image(file)
>>> red_to_grayscale(img)
>>> show(img)
```

Write your function on the **next** page.

Do not write your solution here,

or here,

or here,

and certainly not here.

Use Page 8.

Question 5 Solution

```
from Cimpl import *

def red_to_grayscale(img):
    for x, y, col in img:
        red, green, blue = col
        brightness = (red + green + blue) // 3

        if (red > brightness * 1.4):
            col = create_color(brightness, brightness, brightness)
            set_color(img, x, y, col)
```

Marking scheme: 1 mark for a loop that visits each pixel, binding the (x,y) coordinates and RGB components to variables. 1 mark for calculating the pixel's brightness. 2 marks for an `if` statement that correctly determines if the pixel's red component is greater than 140% of the brightness. 2 marks for creating the pixel's equivalent grayscale colour. 1 mark for calling `set_color` to change the pixel's colour to grayscale when the condition is true.

Alternate solution:

```
from Cimpl import *

def red_to_grayscale(img):
    for x, y, col in img:
        red, green, blue = col
        brightness = (red + green + blue) // 3

        if (red > brightness * 1.4):
            red = brightness
            green = brightness
            blue = brightness
            col = create_color(red, green, blue)
            set_color(img, x, y, col)
```

Some students wrote solutions similar to this:

```
def red_to_grayscale(img):
    for x, y, col in img:
        red, green, blue = col
        brightness = (red + green + blue) // 3

        if (red > brightness * 1.4):
            red = brightness
            green = brightness
            blue = brightness
        col = create_color(red, green, blue)
        set_color(img, x, y, col)
```

Notice that the calls to `create_color` and `set_color` are outside the `if` statement, so these

functions will be called for every pixel, and not just the ones that are changed to grayscale. No marks were deducted for this, because the function correctly filters the image; however, the unnecessary calls will increase the function's execution time.

Some students wrote solutions similar to this:

```
def red_to_grayscale(img):
    for x, y, col in img:
        red, green, blue = col
        brightness = (red + green + blue) // 3

        if (red > brightness * 1.4):
            col = create_color(brightness, brightness, brightness)
            set_color(img, x, y, col)
        else:
            col = create_color(red, green, blue)
            set_color(img, x, y, col)
```

The `else` clause is completely unnecessary; it simply sets the pixel's colour to its current colour. Again, no marks were deducted for this, because the function correctly filters the image; however, the unnecessary calls will increase the function's execution time.

Crib Sheet

Built-in Python functions:

`abs(x)`
Return the absolute value of *x* (an `int` or a `float`).

`float(x)`
Convert argument *x* (a string or number) to a real number, if possible.

`int(x)`
Convert argument *x* (a string or number) to an integer, if possible. A real number argument will be truncated towards zero.

`max(a, b, c, ...)`
With two or more arguments, return the largest argument.

`min(a, b, c, ...)`
With two or more arguments, return the smallest argument.

`range(stop)`
Return an object containing the sequence of integers: `0, 1, 2, ..., stop - 1`.

`range(start, stop)`
Return an object containing the sequence of integers:
`start, start + 1, start + 2, ..., stop - 1`.

`input(prompt)`
Read a string from the keyboard and return the string. The *prompt* string, if provided, is printed without a trailing newline before reading the string.

`round(x, n)`
Return the `float` that results when the value of argument *x* (a `float`) is rounded to *n* digits after the decimal point.

`str(x)`
Return a printable string representation of argument *x*.

This crib sheet continues on the next page.

`Cimpl` module:

`choose_file()`
Launch a file chooser and return a string containing the name of the file that was selected.

`load_image(`*`filename`*`)`
Create an `Image` object from the contents of *`filename`* (a string) and return it.

`create_color(`*`red, green, blue`*`)`
Return a `Color` object with the specified *`red`*, *`green`* and *`blue`* components. Each component should be an `int` between 0 and 255; however, when the `Color` object is created, non-integer component values are converted, if possible, to `int`s; negative values are converted to 0, and values > 255 are capped at 255.

Functions that work with `Image` objects:

`copy(`*`img`*`)`
Return a new image containing a copy of the image *`img`*.

`get_color(`*`img, x, y`*`)`
Return a `Color` object containing the RGB components of the pixel at location (*x*, *y*) in the image *`img`*.

`get_height(`*`img`*`)`
Return the height of the image in pixels (an `int`).

`get_width(`*`img`*`)`
Return the width of the image in pixels (an `int`).

`set_color(`*`img, x, y, color`*`)`
Set the color of the pixel at location (*x*, *y*) in image *`img`*, to the RGB values stored in `Color` object *`color`*.

`show(`*`img`*`)`
Display the image *`img`* in a pop-up window.