Carleton University
SYSC 1005 – Introduction to Software Development – Fall 2015
Midterm Exam – October 21, 2015

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 _____**/ 6**

Question 6 _____**/ 6**

Total _____**/ 35**

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`.

```
>>> 14 - 10 // 3 + 12 / 2
```

_____

```
>>> 32 % 3 + 32 // 3 * 3
```

_____

```
>>> 40 / 2 ** 2
```

_____

```
>>> 6 -- (7 + 4)
```

_____

```
>>> 24 % 5
```

_____

```
>>> markup = 1000 * 0.03
```

_____

```
>>> p = 22
>>> q = 33
>>> t = p, q
>>> j, k = t
>>> k
```

_____

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

_____

Question 2 [4 marks]

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

```
from Cimpl import *

def mystery_filter(first, second):
    cobaltgreen = create_color(61, 145, 64)
    cadetblue = create_color(95, 158, 160)

    for x, y, col in first:
        set_color(first, x, y, cobaltgreen)

    second = copy(first)
    for x, y, col in second:
        set_color(second, x, y, cadetblue)
    first = copy(second)
```

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

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

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

A. entirely cobaltgreen.

B. entirely cadetblue.

C. the original image that was bound to `img1`.

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

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

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

A. entirely cobaltgreen.

B. entirely cadetblue.

C. the original image that was bound to `img1`.

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

Question 3 [8 marks]

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

```
def mystery(p, q):
    p = p + 7
    r = p * q
    q = q + 5          # Point B
    return r

p = 6
q = 2                  # Point A
r = mystery(p, q)
s = p * q              # 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 label each frame with its name. 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 <u>after</u> the assignment statement at Point A is executed, but before the next statement is executed.

(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.

(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.

Question 4 [2 marks]

Here is the definition of a function that calculates the volume of a right-circular cone (a cone with a base that is a circle):

```
import math

def volume_of_cone(radius, height):
    return math.pi / 3 * height * radius ** 2
```

Considering the software engineering principles presented in this course, what is the best docstring for this function? (Circle A, B, C, or D, beside the correct answer.)

A
```
""" (number, number) -> float

Calculate the volume of a right-circular with the specified radius and height.

>>> volume_of_cone(3, 10)
94.2478
"""
```

B
```
""" (number, number) -> float

Return the volume of a right-circular with the specified radius and height.
This function uses the ** operator instead of Python's built-in pow
function to square the radius.

>>> volume_of_cone(3, 10)
94.2478
>>> volume_of_cone(3.0, 10.0)
94.2478
"""
```

C
```
""" (number, number) -> float

Return the volume of a right-circular with the specified radius and height.

>>> volume_of_cone(3, 10)
94.2478
>>> volume_of_cone(3.0, 10.0)
94.2478
"""
```

D
```
""" (number, number) -> float

Print the volume of a right-circular with the specified radius and height.

>>> volume_of_cone(3, 10)
94.2478
>>> volume_of_cone(3.0, 10.0)
94.2478
"""
```

Question 5 [6 marks]

Write a function named `darken`, which changes every pixel in an image by dividing each pixel's red, green and blue components by specified divisors. The function has four parameters, which should be listed in this order in the function header:

- the image that the function will modify
- a `int` representing the divisor for the red component of each pixel
- a `int` representing the divisor for the green component of each pixel
- a `int` representing the divisor for the blue component of each pixel

You can assume that the three divisors will be in the range 1 to 10; your function does not need to check this. For example, if a pixel's blue component is 80 and the divisor is 2, the division yields 40, so the blue component of the pixel's new colour will be 40. (Remember that the component values that are passed to `create_color` can be real numbers. See the description of `create_color` in the crib sheet.)

You have been provided with an incomplete implementation of this function. Complete the function by writing your solution on the ruled lines. (A correct solution requires fewer lines of code than the number of ruled lines.) You do not need to write a documentation string.

```
from Cimpl import *

def darken(_____________________________________________________________):
    for x, y, col in img:
        r, g, b = col
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Question 6 [6 marks]

Suppose we want to change an image so that every pixel's colour is changed to one of the three secondary colours in the RGB colour model (magenta, cyan or yellow). Here is one way to do this:

- If a pixel's red and green components are both greater than the blue component, change the pixel's colour to yellow (255, 255, 0).
- If a pixel's green and blue components are both greater than the red component, change the pixel's colour to cyan (0, 255, 255).
- Otherwise, change the pixel's colour to magenta (255, 0, 255). (This handles the case where the pixel's red and blue components are both greater than the green component, plus other cases.)

On the next page, you have been provided with an incomplete implementation of a function named `all_secondary` that implements this filter. Complete the function by writing your solution on the ruled lines. Notice that the function calls `create_color` three times, to create `Color` objects that represent the three secondary colours. Your solution should not create any other `Color` objects; i.,e, it can't call `create_color` inside the loop.

A correct solution requires fewer lines of code than the number of ruled lines; however, if you need more room, continue your solution on the back of a page.

Do not write your solution here,

or here,

or here,

and certainly not here.

Use Page 9.

Question 6 Solution

```
from Cimpl import *
def all_secondary(img):
    """ (Cimpl.Image) -> None

    Modify img so that each pixel is one of the secondary colours.

    >>> img = load_image(choose_file())
    >>> all_secondary(img)
    >>> show(img)
    """

    # Create the secondary colours once, outside the loop.

    yellow = create_color(_____________________________________________)

    cyan = create_color(_____________________________________________)

    magenta = create_color(_____________________________________________)

    for x, y, col in img:
        r, g, b = col
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

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.