Carleton University
SYSC 1005 – Introduction to Software Development – Fall 2017
Midterm Exam A Solutions

Name:_____

Student Number:_____

INSTRUCTIONS:

1. This exam is closed book. Simple (non-programmable) 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. Do not write anything on this page other than your name and student number.
4. Answer all questions on the question paper. You can use pen or pencil.
5. Do not detach any sheets from the question paper.
6. You are not permitted to use the calculator/clock apps on your cell phone. The time will be written on the blackboard and updated periodically throughout the exam.
7. An attendance sheet will be distributed after the exam begins. If you finish your exam early, do not leave until you have signed the attendance sheet.
8. If you need to go to the washroom during the exam, bring your question paper to the teaching console at the front of the classroom.
9. For all questions, assume that all required `import` statements are in the source code.
10. When drawing Python Tutor-style diagrams, remember to label each frame with its name. Remember to label each variable with its name and each object with its type and value (use `?` if the value is unknown). Use arrows to represent bindings/references. If a frame has been deallocated, instead of erasing it you can draw a line through it or write "Deallocated" in the frame.
11. The crib sheet at the end of this question paper summarizes some Python functions that you may find useful.
12. When you have finished your exam, give it to one of the proctors. Show your campus card to the proctor, who will verify that the name and student number on your card match what is written on this page. You must leave the exam room as soon as you have turned in your paper.
13. All pages of this question paper, including the crib sheet, must be turned in.

Q1: ____/8 Q2: ____/4 Q3: ____/6 Q4: ____/2 Q5-19: ____/15 Total: ____/35

Question 1 [8 marks]

Here is an incomplete transcript from a session with the Python shell running Python 3.6. 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`.

```
>>> 45 - 24 / 4 * 5 + 3
```

18.0 (Comment: not 18)

```
>>> 38 // 5 * 5 + 38 % 5
```

38_ (Comment: not 38.0)

```
>>> 16 / 2 ** 3
```

2.0 (Comment: not 2)

```
>>> 8 -- (5 - 3)
```

10 (Comment: not 10.0)

```
>>> (9 - 2)(4 + 3)
```

Error (no operator between (9 - 2) and (4 + 3); multiplication is not implied)

```
>>> 20 * 0.08 = tax
```

Error (this statement is not equivalent to tax = 20 * 0.08)

```
>>> p = 20
>>> q = 35
>>> p = p, q
>>> q
```

35

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

35

Question 2 [4 marks]

Put a checkmark, ✔, or an ✗, in the box, ☐, beside the correct answer. For each part, 2 marks will be awarded for selecting the correct answer and 0 marks will be awarded for selecting an incorrect answer. 0 marks will be awarded if you select more than one answer for each part, even if one of the answers is correct.

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

```
from Cimpl import *

def mystery_filter(first, second):
    coral = create_color(255, 127, 80)
    orchid = create_color(218, 112, 214)

    for x, y, col in second:
        set_color(second, x, y, coral)

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

    second = first
```

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

```
>>> image1 = load_image(choose_file())
>>> image2 = load_image(choose_file())
>>> mystery_filter(image2, image1)
>>> show(image1)
```

Solution*: suppose* `image1` *refers to image A and* `image2` *refers to image B. When* `mystery_filter` *is called,* `first` *is bound to B and* `second` *is bound to A.*

The first loop changes all the pixels in A to coral.

`first = copy(second)` *makes a copy of A and binds* `first` *to this image (call it C). The second loop changes all the pixels in C to orchid.*

Finally, `second = first` *makes* `second` *refer to the same image as* `first` *(C).*

When the function returns, `image1` *refers to A (which was modified) and* `image2` *refers to B (which was not modified.)*

Part A: The picture that was just displayed is

- ☐ entirely coral.
- ☐ entirely orchid.
- ☐ the original image that was bound to `image1`.
- ☐ the original image that was bound to `image2`.

```
>>> show(image2)
```

Part B: The picture that was just displayed is

- ☐ entirely coral.
- ☐ entirely orchid.
- ☐ the original image that was bound to `image1`.
- ☐ the original image that was bound to `image2`.

Question 3 [6 marks]

These statements are typed in the Python Tutor editor:

```
v = 10
w = 7       # Point A
v = v * 3
x = v + w
w = w - 14  # Point B
z = w * v
```

When answering parts (a) and (b), assume that Python Tutor has been configured the way we've been using it; that is: *hide exited frames*, *render all objects on the heap*, and *draw pointers as arrows*.

Comment: *execute the code using PyTutor and observe the diagram after* `w = 7` *and* `w = w - 14` *are executed.*

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

Frames Objects

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

Frames Objects

Question 4 [2 marks]

When this script was executed, we chose an image in which all the pixels have the same RGB colour, `(60, 120, 150)`. Briefly describe the image that was displayed when `show` was called.

```
def negative(image):
    for x, y, (r, g, b) in image:
        col = create_color(r - 255, g - 255, b - 255)
        set_color(image, x, y, col)

image = load_image(choose_file())
negative(image)
show(image)
```

Solution:

The image displayed by `show` is entirely black, or

Every pixel in the modified image has colour (0, 0, 0)

We weren't looking for a step-by-step explanation of how the filter works. All that was required was a brief description of what the modified image looks like. 0 marks were awarded if the student didn't answer the question that was asked.

Explanation: for each pixel, the arguments passed to `create_color` will be negative. `create_colour` converts negative arguments to 0, so the colour returned by the function is (0, 0, 0) (black).

Multiple-choice Questions [15 marks]

For Questions 5 through 19, put a checkmark, ✔, or an ✗, in the box, ☐, beside the correct answer. For each question, 1 mark will be awarded for selecting the correct answer and 0 marks will be awarded for selecting an incorrect answer. 0 marks will be awarded if you select more than one answer for a question, even if one of the answers is correct.

Question 5

Which of the following is not a legal name for Python functions?

☐ `_is_too_large`
☐ `is_too_large?`
☐ `isTooLarge`
☐ `istoolarge`
☐ `is2large`

Comment: the character `?` cannot be used in a Python function name.

Question 6

Which of these statements about Python functions is correct?

☐ Executing a function definition executes the statements in the function body.
☐ Executing a function definition creates a function object.
☐ Executing a function definition creates a frame for the function's parameters and variables.
☐ Executing a function definition causes Python to display an error message.

Comment: a function definition is executed when a function is loaded into Python, which creates a function object and binds it to a variable with the same name as the function. Choices 1 and 3 describe what happens when a function is called (which is not the same as executing the function definition).

Question 7

After the statement `y = increment(z)` is executed (see Line A), what values are bound to `y` and `z`?

```
def increment(x):
    x = x + 1
    return x
z = 4
y = increment(z)   # Line A
x = y
```

☐ `y: 4  z: 4`
☐ `y: 4  z: 5`
☐ `y: 5  z: 4`
☐ `y: 5  z: 5`
☐ `y: None  z: 4`
☐ `y: None  z: 5`

Comment: execute the code using PyTutor. Note that the statement `x = x + 1` changes the binding in variable `x`, but this does not change the reference stored in the argument (`z`).

Question 8

After the statement `y = f(f(f(f(x))))` is executed (see Line A), what value is bound to `y`?

```
def f(x):
    x = x + 1
    return x
x = 5
y = f(f(f(f(x))))   # Line A
z = y
```

- ☐ 5
- ☐ 6
- ☐ 7
- ☐ 8
- ☐ 9
- ☐ None

Comment: execute the code using PyTutor. Function `f` is called four times. In the second, third and fourth calls, the argument is the value returned by the previous call.

Question 9

What values are printed by this code?

```
v = 3
for v in range(2, 6):
    print(v)
```

- ☐ 0, 2, 4
- ☐ 0, 2, 4, 6
- ☐ 1, 2, 3, 4, 5
- ☐ 1, 2, 3, 4, 5, 6
- ☐ 2, 3, 4, 5
- ☐ 2, 3, 4, 5, 6
- ☐ 3, 4, 5
- ☐ 3, 4, 5, 6

Comment: range returns an object that represents the sequence 2, 3, 4, 5.

Question 10

Which type contract should be added to this function's docstring?

```
def area_of_triangle(b, h):
    """
    Return the area of a triangle with base b and height h.
    """
    return b * h / 2
```

- ☐ This function doesn't need a type contract.
- ☐ `(int) -> int`
- ☐ `(int) -> float`
- ☐ `(int) -> number`
- ☐ `(number) -> int`
- ☐ `(number) -> float`
- ☐ `(number) -> number`
- ☐ `(float) -> int`
- ☐ `(float) -> float`
- ☐ `(float) -> number`
- ☐ We don't have enough information determine the type contract.

Comment: the arguments can be ints or floats. The return type is always a float. Some students noted that `(number, number) -> float` would be a better type contract.

Question 11

After the statement `x = f(x)` is executed (see Line A), what value is bound to `x`?

```
def f(x):
    x ** 2.0

x = 4
x = f(x)   # Line A
y = x
```

- ☐ `4.0`
- ☐ `8.0`
- ☐ `16.0`
- ☐ `4`
- ☐ `8`
- ☐ `16`
- ☐ `None`

Comment: the function doesn't have a `return` statement, so it always returns `None`.

Question 12

The execution of this script is visualized using Python Tutor.

```
def do_this(a):
    return 2 * a

def do_that(x, y):
    a = do_this(x) + y  # Line A
    return a

a = 3
b = 5
x = do_this(a)
y = do_that(a, b)
```

Immediately after `a = do_this(x) + y` is executed (see Line A), but before `return a` is executed, how many variables named `a` appear in the frames in the diagram displayed by Python Tutor?

- ☐ 0
- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4

Comment: execute this code using PyTutor, and observe the diagram when `a = do_this(x) + y` is executed. Two frames will be shown. The global frame will have a variable named `a`, as will frame `do_that`.

Question 13

When this script was executed, we chose an image in which the coordinates of the pixel in the lower-right corner are `(219, 99)`.

```
def make_aqua(image):
    aqua = create_color(0, 255, 255)
    for x, y, (r, g, b) in image:
        set_color(image, x, y, aqua)

image = load_image(choose_file())
make_aqua(image)
```

How many times was `set_color` called?

- ☐ `0`
- ☐ `1`
- ☐ `99`
- ☐ `219`
- ☐ `256`
- ☐ `21,681`
- ☐ `22,000`
- ☐ `256 × 256 × 256 = 16,777,216`

*Comment: The image has 220 * 100 pixels, and `set_color` is called once for each pixel.*

Question 14

What does function `mystery` print?

```
def mystery(k, j):
    print(j, k)

j = 4
k = 7
mystery(j + 2, k + 3)
```

☐ `4 7`
☐ `4 + 2 7 + 3`
☐ `7 + 3 4 + 2`
☐ `42 73`
☐ `73 42`
☐ `10 6`
☐ `6 10`
☐ An error message is displayed.

Comment: execute this code using PyTutor. Variables `j` and `k` in the global frame are independent of parameters `j` and `k` in the frame for `mystery`.

Question 15

Given this code:

```
if x <= 20:
    print(5)
    if x < 10:
        print(4)
    else:
        print(3)
else:
    print(2)
print(1)
```

for what integer values of `x` does this script print a `5` followed by a `3` followed by a `1`?

☐ all values between 9 and 20, inclusive
☐ all values between 9 and 21, inclusive
☐ all values between 10 and 20, inclusive
☐ all values between 10 and 21, inclusive
☐ all values greater than 10
☐ no values
☐ none of the above

Note: "between *a* and *b*, inclusive" means the sequence *a*, *a*+1, *a*+2, …, *b*.

Comment: 5 is printed for all values of `x` <= 20. 3 is printed for all values of `x` <= 20 and >= 10. 1 is always printed.

Question 16

Consider this code fragment:

```
# Assume q and r have been bound to integers

if q <= r:
    p = r + q
else:
    p = q * r
print(p, q, r)
```

Which of the following code fragments is equivalent to this one? "Equivalent" means that both fragments would print the same values.

☐
```
if q == r:
    p = q * r
else:
    p = r + q
print(p, q, r)
```

☐
```
if q != r:
    p = q * r
else:
    p = r + q
print(p, q, r)
```

☐
```
if q >= r:
    p = q * r
else:
    p = r + q
print(p, q, r)
```

☐
```
if q > r:
    p = q * r
else:
    p = r + q
print(p, q, r)
```

☐
```
if q < r:
    p = q * r
else:
    p = r + q
print(p, q, r)
```

☐
```
if q <= r:
    p = q * r
else:
    p = r + q
print(p, q, r)
```

Question 17

Script A uses an `if-elif-else` statement to select and execute one of four actions:

```
a = 20       # Script A
if a == 10:
    a = 20
elif a == 20:
    a = 30
elif a == 30:
    a = 40
else:
    a = 50
```

Which of these scripts are functionally equivalent to Script A? For example, Scripts A and B are "equivalent" if variable `a` is bound to the same value when both scripts finish executing.

```
# Script B

a = 20
if a == 10:
    a = 20
if a == 20:
    a = 30
if a == 30:
    a = 40
else:
    a = 50
```

```
# Script C

a = 20
if a == 10:
    a = 20
    if a == 20:
        a = 30
        if a == 30:
            a = 40
        else:
            a = 50
```

```
# Script D

a = 20
if a == 10:
    a = 20
else:
    if a == 20:
        a = 30
    else:
        if a == 30:
            a = 40
        else:
            a = 50
```

- ☐ B
- ☐ C
- ☐ D
- ☐ both B and C
- ☐ both B and D
- ☐ both C and D
- ☐ all of B, C and D
- ☐ none of B, C and D

Comment: when Script A and D finish executing, `a` is bound to 30. When Script B is finished executing, `a` is bound to 40. When Script C is finished executing, `a` is bound to 20.

Question 18

Here is an incomplete script that "swaps" the integers bound to `j` and `k`.

```
j = 10
k = 15

# Missing code goes here

# j is now bound to 15 and k is now bound to 10
```

The missing code is:

☐
```
j = k
k = j
```

☐
```
t = j, k
j, k = t
```

☐
```
j = j, k
k, j = j
```

☐
```
t = j, k
k, j = t
```

☐ None of the choices is correct.

Comment: 1 mark was awarded for picking either the third or fourth choice. Both pack the values bound to `j` and `k` into a tuple, then unpack the tuple into, binding the values to `k` and `j`.

Question 19

The function $f : x \rightarrow f(x)$ where $f(x) = x^2 + 2x$ can be implemented as this Python function:

☐
```
def f(x):
    return x * x + 2x
```

☐
```
def f():
    return x * x + 2 * x
```

☐
```
def f(x):
    return x * (2 + x)
```

☐
```
def f(x):
    f(x) = x * x + 2 * x
```

☐ None of the choices is correct.

Comment: the first choice has a syntax error (`2x` should be `2 * x`). The second choice does not have a function parameter named `x`. In the fourth choice, the function body `f(x) = x * x + 2 * x` has a syntax error (`f(x)` is not a valid variable name); also, there is no `return` statement.

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

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

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

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

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

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