Introduction to Computer Science I
Fall 2014
Mid-term exam — Solutions

1. Question: Consider the following module of Python code...

```
def red_fish (x):
    y = 0
    if x == 1:
        y = x
        x = 2
    if x == 2:
        y = -x
        x = 3
    else:
        y = x
    print('x = ' + str(x))
    print('y = ' + str(y))

def blue_fish (m, c):
    i = 0
    new_m = ""
    while i < len(m):
        if m[i] == c:
            new_m = new_m + 'X'
        else:
            new_m = new_m + m[i]
        i = i + 1
    return new_m

def main ():

    x = 1
    s = "She sells sea shells by the seashore."
    red_fish(x)
    red_fish(x)
    t = blue_fish(s, 's')
    print(s)
    print(t)

main()
```

What output is printed when this module is run?

ANSWER: The output produced is...

```
x = 3
y = -2
x = 3
y = -2
She sells sea shells by the seashore.
She XellX Xea XhellX by the XeaXhore.
```

DISCUSSION: TBA

2. QUESTIONS: Provide short answers (no more than a few sentences) to each of the following questions:

   (a) What is the difference between a problem that is *intractable* and one that is *incomputable*? Given an example of each type of computation.

   (b) Why don't we use floating-point numbers for every computation? Why have an *int* type of data at all?

   (c) Recall the rules for determining a *leap year*: if the year is divisible by 4, but not if it is divisible by 100, unless it **is** divisible by 400.[1]
   Write a **single expression** that, given a variable `y` that holds a year number, evaluates to `True` if the year is a leapyear, and evaluates to `False` otherwise. Critically, note that you must write **only an expression, not a statement.** That is, no *if-then* statements or loops.

   ANSWERS:

   (a) A problem that is *incomputable* does not have, and cannot have, an algorithmic solution (e.g., *The Halting Problem*). An *intractable* problem is one for which there is an algorithmic solution, but that solution would require an unreasonable amount of time or memory to compute (e.g., ordering via random permutation).

   (b) Floating point numbers can suffer from *rounding errors*, where small fractions of the calculation results are inexact. This problem makes comparisons between the results of computations difficult. Integer math is always exact, making comparison straightforward.

   (c) `(y % 400 == 0) or ((y % 4 == 0) and (y % 100 != 0))`

   DISCUSSION: TBA

[1] By *divisible* I mean that the result of calculating division would leave no remainder.

3. QUESTION: Consider the *factorial function*, which, for a non-negative integer n, is defined as:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * fact(n-1) & \text{if } n \geq 1 \end{cases}$$

Write two functions to implement this definition:

(a) One version that is **iterative** (uses loops).

(b) Another version that is **recursive** (calls itself).

ANSWER:

(a) **Iterative:**

```
def fact_iterative (n):
    result = 1
    i = 1
    while i <= n:
        result = result * i
        i = i + 1
    return result
```

(b) **Recursive:**

```
def fact_recursive (n):
    if n == 0:
        return 1
    else:
        return n * fact_recursive(n - 1)
```

DISCUSSION: TBA

4. QUESTION: Consider two strings, `s` and `t`. If we assume that $len(s) \geq len(t)$, then it is possible that, in a particular sense, `s` *contains* `t`. That is, somewhere within `s`, starting at some position k, exists the continguous entirety of `t`. If `s` does contain `t`, we say that `t` is a *substring* of `s`.

   For example, if `s = 'I am the very model of a modern major general'`, then `t = 'very'` is contained in `s` starting at position 9. In contrast, another string, say `q = 'moam'` is **not** contained by `s`, even though that sequence of letters does appear in `s` in that order, but not contiguously.

   Write a function named `findSubstring` that accepts `s` and `t` as parameters. If `s` contains `t`, then `findSubstring` must return the position within `s` at which the substring `t` begins. If `t` **not** a substring of `s`, then `findSubstring` should return the special value -1.
   ANSWER:

```
def findSubstring (s, t):
    i = 0
    while i < len(s):
        shortened_s = s[i : i + len(t)]
        if shortened_s == t:
            return i
        i = i + 1
    return -1
```

   DISCUSSION: TBA