CS 150 - Sample Final Exam

You may use the following material on this exam:

- The final exam "cheat sheet" which I will provide
- Up to two sheets of notes, letter-size, double-sided

Beyond that, this exam is closed book, closed notes, closed computer, closed calculator, etc.

You do not need to include comments or constants in your code.

---

1. Odds and ends

   (a) T/F: For each statement below, indicate whether the statement is true or false. If you make an assumption about something in the statement, please note it underneath your answer.

_______ For all possible inputs insertion sort is faster than selection sort.
False. On average, insertion sort does require less comparisons, however, for any given run, the actual run-time could be faster or slower than selection sort.

_______ 01101 in binary is equal to 13.
True.

_______ '*' in R for vectors is equivalent to '.*' in Matlab for vectors.
True.

_______ If we typed the two first statements below, s would have the value shown below:

```
>>> s = "this is a test"
>>> s.upper()
'THIS IS A TEST'
>>> s
'THIS IS A TEST'
```

False. Strings are immutable.

_______ For an $O(n^2)$ algorithm, if it takes 10 seconds to run on 1000 items we would expect it to take approximately 40 seconds on 2000 items.
True. For algorithms with quadratic run-times, doubling the size would result in roughly quadrupling the run-time.

(b) Add the following two binary numbers: `01101101` with `00111101`. Make sure to show your work (i.e. carries, etc.).

```
11111 1
01101101
00111101
--------
10101010
```

(c) Given the following code in a file called `my_file.py`:

```
# some code up here

if __name__ == "__main__":
    print("bananas " * 2)
else:
    print("apples"[:-3])

print("End of " + __name__)
```

What would happen if the file were run by clicking on the "run" button?

You'd see:
bananas bananas
End of __main__

What would happen if we then typed `import my_file` in the Python shell?

You'd see:
app
End of my_file

(d) The `xor` (exclusive or) of two boolean values is `True` if either value is `True`, but is `False` if they are both `True` or both `False`. The following is the "truth table" for `xor`:

a	b	xor(a,b)
F	F	F
F	T	T
T	F	T
T	T	F

Write a function `xor` that calculates the exclusive or of two boolean variables. Remember to use good boolean style!

```
def xor(a,b):
        return a != b
```

2. Matlab

(a) If we initialized `x` in Matlab to be the following matrix:

```
10 4  3
1  7  15
6  4  2
```

What would be displayed by the following:

i. `disp(x(:, 1))`
   10
   1
   6

ii. `disp(sum(x))`
   17 15 20

iii. `disp(sum(x > 5))`
   2 1 1

iv. `disp(sum(sum(x .* (x > 5))))`
   38

(b) Write a Matlab function that takes a matrix as a parameter and returns the difference between the largest and smallest values in that matrix.

function result = max_minus_min(m)
result = max(max(m)) - min(min(m));
end

3. Dictionaries

(a) Write a function called `dictionary_add` that takes two dictionaries whose *values* are numbers and returns a dictionary containing the keys found in **both** dictionaries. The value associated with these keys should be the sum of the values in the two dictionaries. If a key does not occur in BOTH dictionaries, then it should not occur in the returned dictionary. For example:

```
>>> d1 = {"a":1, "b":2, "c":3}
>>> d2 = {"a":4, "c":5, "d": 6}
>>> dictionary_add(d1, d2)
{'a': 5, 'c': 8}
```

```
def dictionary_add(dict1, dict2):
    result_dict = {}
    for key in dict1:
        if key in dict2:
            result_dict[key] = dict1[key] + dict2[key]
    return result_dict
```

(b) In one sentence, what does the following function do if `dict1` and `dict2` are dictionaries? Be concise, but precise.

```
def mystery(dict1, dict2):
    for key in dict1:
        if not key in dict2:
            return False

    return True
```

Returns True if all of the keys in dict1 are in dict2, False otherwise.

4. Recursion

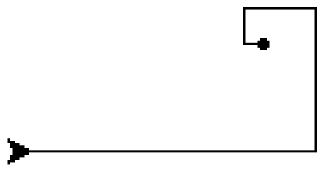
   (a) Draw what the following function would draw on the screen for the call `mystery(80, 5)` assuming the turtle started at the center of the screen (0,0) and was facing to the right. In addition, annotate your drawing with the final turtle location and orientation.

```
from turtle import *

def mystery(length, levels):
    if levels == 0:
        dot()
    else:
        forward(length)
        left(90)

        mystery(length/2, levels-1)

        right(90)
        backward(length)
```



(b) What does the following function return for the input "CS class":

```
def mystery(s):
  if s == "":
    return s
  else:
    if s[0].isupper():
      return s[0].lower() + mystery(s[1:])
    else:
      return s[0].upper() + mystery(s[1:])
```

cs CLASS

(c) Write a function called `sum_squared` that takes a list of numbers as a parameter and returns the sum of the each of the numbers squared. For example, `sum_squared([1, 2, 3])` would return `14` (that is, 1*1 + 2*2 + 3*3=14). If you write the function *recursively* you will receive up to 5 points. If do not use recursion you can only receive a maximum of 3.5 points.

```
def sum_squared(list):
    if len(list) == 0:
        return 0
    else:
        return list[0]*list[0] + sum_squared(list[1:])
```

5. We've got problems...

   (a) The following program expects two command-line arguments. The program is supposed to print the usage if the user doesn't supply exactly two arguments, or run the function `my_function(...)` if the user does supply two arguments. However, the program has **two** general problems. Fix the problems so that it works as expected. You don't have to rewrite the function, just mark-up the one below.

```
import sys

def my_function(a, b):
  # some function stuff

def print_usage():
  print("my_program.py <number> <number>")

number1 = sys.argv[0]
number2 = sys.argv[1]

if len(sys.argv) != 2:
  print_usage()
else:
  my_function(number1, number2)
```

   (1) sys.argv will have 3 entries with the first one being the name of the program, so all of the indices referring to sys.argv need to be increased by 1 (0→1, 1→2, 2→3).
   (2) Move the declaration of number1 and number2 inside the else statement. Otherwise you'll get an index out of bounds error.

   (b) The following function attempts to check if a number is prime or not, but it has a mistake. Correct the function:

```
import math

def isprime(num):
    """Returns True if the input is a prime number, False otherwise"""
    for i in range(2, int(math.sqrt(num)+1)):
        if num % i == 0:
            return False
        else:
            return True
```

   Move the "return True" to be *after* the for loop, not inside it.

6. Sequences

(a) Write a function called `unique` that takes a string as a parameter and returns `True` if all of the characters in the string are unique (i.e no repeats) or `False` if there are duplicate letters. You will only get partial credit if your function is not efficient.

Many ways to do it, but should be using a set for efficiency. Here's one simple way:

```
def unique(string):
    return len(set(string)) == len(string)
```

(b) Write a function `reverse_sentence` that takes a string representing a sentence and returns a new string where all the words in the sentence are in reverse order. You can assume a "word" is anything separated by a space. For example:

```
>>> reverse_sentence("this is a sentence")
'sentence a is this'
```

```
def reverse_sentence(sentence):
    words = sentence.split()
    reversed = ""
    for word in words:
        reversed = word + " " + reversed
    return reversed.strip()
```