CS150 – Sample Final Solution

Name: ________________________________ **Section: A / B**

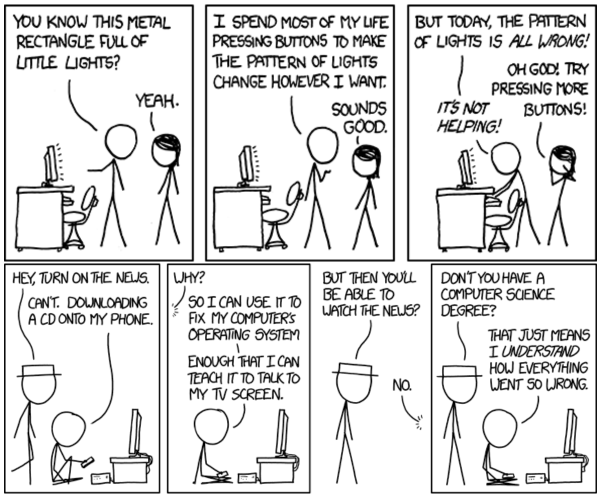
Date: ________________ **Start time:** ______________ **End time:** ______________

Honor Code:

Signature: ____________________________

This exam is closed book, closed notes, closed computer, closed calculator, etc. You may only use (1) the final "cheat sheet" from the course page, and (2) a single double-sided sheet of notes. **You have 3 hours.** Read the problem descriptions carefully and write your answers clearly and *legibly* in the space provided. Circle or otherwise indicate your answer if it might not be easily identified. You may use extra sheets of paper, stapled to your exam, if you need more room, as long as the problem number is clearly labeled and your name is on the paper. If you attached extra sheets indicate on your main exam paper to look for the extra sheets for that problem.

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


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!
HEY, TURN ON THE NEWS.
CAN'T. DOWNLOADING A CD ONTO MY PHONE.
WHY?
SO I CAN USE IT TO FIX MY COMPUTER'S OPERATING SYSTEM
ENOUGH THAT I CAN TEACH IT TO TALK TO MY TV SCREEN.
BUT THEN YOU'LL BE ABLE TO WATCH THE NEWS?
NO.
DON'T YOU HAVE A COMPUTER SCIENCE DEGREE?
THAT JUST MEANS I UNDERSTAND HOW EVERYTHING WENT SO WRONG.

Question 1: Odds and Ends [0 points]

(a) For each of the statements below state whether they are **T**rue or **F**alse.

i. __**T**__ For all possible inputs merge sort has lower asymptotic time complexity than selection sort.

ii. __**T**__ 01101 in binary is equal to 13.

iii. __**F**__ All columns in a datascience `Table` must have the same type.

iv. __**F**__ 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'
```

v. __**T**__ For an $O(n^2)$ algorithm, if the algorithm takes 10 seconds to run on 1000 items we would expect it to take approximately 40 seconds on 2000 items.

(b) Add the following two binary numbers: `01101101` with `00111101` assuming they both are unsigned positive integers. Show your work (including all carries) and verify your answer by converting both numbers and their sum to decimal.

Solution:

```
  11111 1
  01101101    109
+ 00111101     61
----------    ---
  10101010    170
```

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

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

i. What would be printed if the file were run by clicking on the green arrow?

Solution:
```
bananas bananas
End of __main__
```

ii. What would be printed if we executed `import my_file` in the Python shell?

Solution:
```
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)`
False	False	False
False	True	True
True	False	True
True	True	False

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

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

Question 2: Vector execution [0 points]

(a) In the following R code, assume `a` and `b` are equal-length vectors of numbers.

```
mystery <- function(a, b) {
    sum(a * b)
}
```

i. Rewrite the above code in Python using NumPy and/or pandas as appropriate assuming that `a` and `b` are lists of floats. You will be evaluated based on the efficiency and conciseness of your approach.

Solution:
```
import numpy as np
def mystery(a, b):
    return np.sum(np.array(a) * np.array(b))
```

ii. Rewrite the above code in Python using *only* built-in functions assuming that `a` and `b` are lists of floats. You will be evaluated based on the efficiency and conciseness of your approach.

Solution:
```
def mystery(a, b):
    result = 0
    for i n range(len(a)):
        result += a[i] * b[i]
    return result
```

(b) Which of the above implementations has the smallest Big-O asymptotic time complexity?

Solution: All the implementations have $O(n)$ time complexity. The implementations may take different amounts of time for the same input, but that doesn't change the asymptotic time complexity, which describes how execution time grows as the input grows.

Question 3: Dictionaries [0 points]

(a) Write a function named `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}
```

Solution:

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

(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
```

Solution:

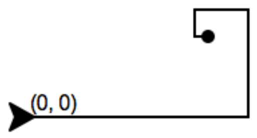
Returns True if all of the keys in `dict1` are in `dict2`, False otherwise.

Question 4: Recursion [0 points]

(a) Draw the image that would be created by the above code if invoked as `mystery(80, 5)`. Assume that the turtle is initially at the origin, facing right. 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)
```



Solution:

(b) What does the following function return when invoked with `"CS class"` as its argument?

```
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:])
```

Solution: `"cs CLASS"`

(c) Write a recursive function named `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, $1 * 1 + 2 * 2 + 3 * 3 = 14$).

Solution:

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

Question 5: We've got problems... [0 points]

(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 a runtime error and logical error. Fix both 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)
```

Solution:

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, e.g. `number1 = sys.argv[1]`.
2. Move the declaration of `number1` and `number2` inside the `else` statement. Otherwise you'll get an index out of bounds error when an incorrect number arguments are provided.

(b) The following function attempts to check if a number is prime or not, but it has a logical error. 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
```

Solution: Move the `return True` to be *after* the `for` loop (without an `else`), not inside it, i.e.:

```
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
    return True
```

Question 6: Sequences [0 points]

(a) Write a function named `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 be evaluated based on the efficiency and conciseness of your approach.

Solution:

```
# One of many potential approaches, but your solution should use a
# set for efficiency
def unique(string):
    return len(set(string)) == len(string)
```

(b) Write a function name `reverse_sentence` that takes a string representing a sentence as a parameter 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'
```

Solution:

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