# CS150 – Sample Final

**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.**
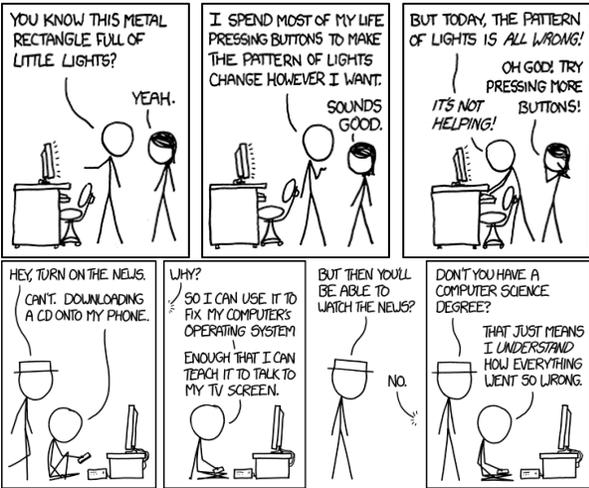
**Question 1: Odds and Ends [0 points]**

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

         i. _____ For all possible inputs merge sort has lower asymptotic time complexity than selection sort.

        ii. _____ 01101 in binary is equal to 13.

       iii. _____ All columns in a datascience `Table` must have the same type.

       iv. _____ 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. _____ 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.

(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?

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

(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!

**Question 2: Vector execution [0 points]**

(a) In the following Python code, assume `a` and `b` are lists of floats.

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

Rewrite the above code in Python using *only* built-in functions. If the code returns a vector, your implementation should return a list You will be evaluated based on the efficiency and conciseness of your approach.

(b) In the following Python code, assume `a` is a list of floats, and `thresh` a single float.

```python
import numpy as np
def mystery(a, thresh):
    b = np.array(a)
    return b[b < thresh]
```

Rewrite the above code in Python using *only* built-in functions. If the code returns a vector, your implementation should return a list You will be evaluated based on the efficiency and conciseness of your approach.

(c) Which of the above implementations, using NumPy or just Python built-ins, have the smallest Big-O asymptotic time complexity? Briefly explain your answer.

**Question 3: Objects [0 points]**

(a) What does the above code below print when executed with the green arrow?

```
class Foo():
    def __init__(self, a):
        self.a = a
        self.b = 2

    def mystery(self, d):
        return self.a + self.b + d

class Bar(Foo):
    def __init__(self, c):
        super().__init__(4)
        self.c = c

    def mystery(self, d):
        return self.a + self.b + self.c + d

a = Foo(6)
b = Bar(2)
print(a.mystery(3))
print(b.mystery(1))
```

(b) You are implementing an application to help the library track its collection. One desired feature is: "Check if a book is in the public domain, i.e., its publication date is more than 95 years ago.". Identify one likely class, method on that class and attribute in that class you would define to implement this feature. If there is more than one potential class, method or attribute you only need to identify one of each.

**Question 4: 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}
```

(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
```

**Question 5: 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)
```

(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:])
```

(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, $1 * 1 + 2 * 2 + 3 * 3 = 14$).

**Question 6: 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)
```

(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
```

**Question 7: 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.

(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'
```