CS 111: Final Exam Extra Practice Problems

As we get closer to the exam, solutions will be posted under **Other Content** *on Blackboard.*

Question #1

What is the output of each of the following programs?

Part A

```
num = 30
if num > 20:
    print("do")
    if num < 15:
        print("go")
    print("no")
elif num < 0:
    print("lo")
    if num == 30:
        print("mo")
elif num // 3 == 10:
    print("so")
if num > 5:
    print("to")
```

Part B

```
x = 15
y = x
z = x // 2
w = x / 2
x = x + 2
print(x, y, z, w)
```

Part C

```
for i in range(3, 5):
    for j in range(2, i):
        print(i, j)
    print(i + j)
print(i * j)
```

Part D

```
def foo(a, b):
    while b > 0:
        a += 1
        b -= 1
    print(a, b)
    return a

a = 7
b = 3
foo(b, a)
print(a, b)
```

Question #2

Part A

Use a loop to write a Python function **`is_prime(n)`**, which takes in an integer **`n`** and returns **`True`** if **`n`** is prime and **`False`** if **`n`** is composite. You may assume that **`n`** will be greater than **`1`**.

Part B

Use recursion (no loops!) to write a Python function **`add_primes(vals)`**, which takes in a list **`vals`** of integers (each of which will be greater than **`1`**) and it returns ***the sum of only the prime numbers*** in the list **`vals`**. *Hint:* Use the function you wrote for Part A!

Question #3

Consider the following recursive function:

```
def foo(n):
    if n < 2:
        return 1
    else:
        result1 = foo(n-1)
        result2 = foo(n-2)
        return result1 + result2
```

If you were to evaluate the following at the Python prompt:

```
>>> foo(5)
result: 8
```

How many times would **`foo`** be called in this evaluation of **`foo(5)`**?

Question #4

Use recursion (no loops!) to write a Python function **`uniquify(vals)`** that takes in any list **`vals`** and returns ***a list of the distinct elements*** in the list **`vals`**. The order of the elements may be preserved, but they do not have to be. For example:

```
>>> uniquify([42, 'spam', 42, 5, 42, 5, 'spam', 42, 5, 5, 5])
result: ['spam', 42, 5]

>>> uniquify([0, 1, 2, 3, 0, 1, 2])
result: [3, 0, 1, 2]
```

Hint: Your function may make use of the **`in`** operator.

Question #5

Write a *recursive* Python function named **`merge`** that will merge two sorted lists of integers and return the merged sorted list. For example:

```
>>> merge([1, 4, 7, 11, 14], [2, 3, 6, 11, 13, 17])
result: [1, 2, 3, 4, 6, 7, 11, 11, 13, 14, 17]
```

Question #6

Your managers at Acme Composite Materials have decided to implement primality-checking in hardware with digital circuits. They've asked you to prototype a 4-bit primality tester:

Part A

Create a truth table with four bits of input (the binary representation of the values from 0 to 15, inclusive). For each of these sixteen possible inputs, indicate the appropriate output: 1 in the cases that the input is prime, and 0 in the cases that the input is composite. Acme Composites does not consider 0 or 1 to be primes.

Part B

Using the minterm expansion principle, sketch a circuit that implements the truth table from Part A.

Question #7

Write a Python function **`symmetric(grid)`**, which takes in a 2-D list of numbers, **`grid`**. You should assume that **`grid`** is a square array, with an equal number of rows and columns. Then, **`symmetric`** should return **`True`** if the values of **`grid`** are symmetric across the NW-SE diagonal—i.e., if the values "mirror" each other on either side of the diagonal going from the upper-left corner to the lower-right corner (see below)—and should return **`False`** if the values of **`grid`** are not symmetric across the NW-SE diagonal.

```
>>> symmetric([[1]])
result: True

>>> symmetric([[1, 2],          # is symmetric because the 2s match
               [2, 5]])
result: True

>>> symmetric([[1, 2],          # not symmetric because 1 != 2
               [1, 1]])
result: False

>>> symmetric([[1, 2, 3],       # is symmetric because 2s, 3s and 5s match
               [2, 4, 5],
               [3, 5, 6]])
result: True
```

<u>Question #8</u>

Below is the start of a **`Matrix`** class that initializes each object's data to a 2-D list of all zeros:

```
class Matrix:
    def __init__(self, nrows, ncols):
        self.nrows = nrows
        self.ncols = ncols
        self.data = [[0]*ncols for r in range(nrows)]
```

Write a method **`max(self, other)`** that takes in a second **`Matrix`** object **`other`**. This method should return a matrix with as many rows as are found in the shorter of **`self`** and **`other`**, and with as many columns as are found in the narrower of **`self`** and **`other`**. Each entry of the returned matrix should be the larger (the **`max`**) of the corresponding entries in **`self`** and **`other`**. Neither **`self`** nor **`other`** should change.

<u>Question #9</u>

Construct a finite state machine that accepts exactly those input strings of 0's and 1's that have ***at most*** two consecutive bits that are identical. Input strings with three or more consecutive identical bits should be rejected. For example, "0", "1", "00", "11", "01", "10", "010", and "00100110" should all be accepted. However, "111", "000", "01110", and "10101000" should all be rejected.