Name: _____ NetID: _____

(Legibly print **last name**, first name, middle name)

Statement of integrity:

It is a violation of the Code of Academic Integrity to look at any exam other than your own, to look at any reference material outside of this exam packet, to communicate with anyone other than the exam proctors, or to otherwise give or receive any unauthorized help during the exam.

Academic Integrity is expected of all students of Cornell University at all times. **By submitting this exam, you declare that you will not give, use, or receive unauthorized aid in this examination.**

Circle your discussion section:

Section	Day and Time	Instructor
201	W 10:10 AM - 11:00 AM	Kelly Cheng
202	W 11:20 AM - 12:10 PM	Xinran Zhu
203	W 12:25 PM - 1:15 PM	Blaire Yu
204	W 1:30 PM - 2:20 PM	Blaire Yu
205	W 2:40 PM - 3:30 PM	Claire Liang
206	W 3:45 PM - 4:35 PM	Claire Liang

Instructions:

- Check that this packet has 7 double-sided sheets.
- This is a 90-minute, closed-book exam; no calculators are allowed.
- The exam is worth a total of 100 points, so it's about one point per minute!
- Read each problem completely, including any provided code, before starting it.
- Do not modify any *given* code.
- Raise your hand if you have any questions.
- Use the back of the pages if you need additional space.
- Indicate your final answer. If you supply multiple answers, you may receive a *zero* on that question.
- Use only MATLAB code. No credit for code written in other programming languages.
- Assume there will be no input errors.
- Do not use `switch`, `try`, `catch`, `break`, `continue`, or `return` statements.
- Do not use built-in functions that have not been discussed in the course.
- You may find the following MATLAB predefined functions useful: `abs`, `sqrt`, `rem`, `floor`, `ceil`, `rand`, `zeros`, `ones`, `linspace`, `length`, `input`, `fprintf`, `disp`

Examples:
`rem(5,2)` → 1, the remainder from 5 divided by 2
`rand()` → a random real value in the interval (0,1)
`abs(-3)` → 3, absolute value
`floor(6.9)`, `floor(6)` → 6, rounds down to the nearest integer
`ceil(8.1)`, `ceil(9)` → 9, rounds up to the nearest integer
`length([2 4 8])` → 3, length of a vector
`zeros(1,4)` → 1 row 4 columns of zeros
`linspace(3,5,10)` → a vector of 10 real numbers evenly distributed in the interval [3,5]

Use this page for scratch work.

Question 1 (13 points)

Question 1.1 (8 points) Write in the box below the first 6 lines of output that would be produced by executing the following script. Write on the blank the total number of times the disp function is called. Note that disp([9, 8 ,7]) will display the vector as 9 8 7. If an error would occur during execution, write all the output that would be produced, if any, before the error would occur, followed by the word "error".

```
j = 1;
v = [9, 8, 7];
while j < 6
    j = j + 1;
    disp(j+1)
    v(j+1) = v(j) + 1;
    disp(v)
end
```

First 6 lines of output:

3

9 8 9

4

9 8 9 10

5

9 8 9 10 11

Total number of times disp is called: 10

Question 1.2 (5 points) What will be printed when the following script is executed? Please write your answer in the correct order that they would be printed. If an error would occur during execution, write all the output that would be produced, if any, before the error would occur, followed by the word "error".

Script	*Function* (in `foo.m`)	Answer:
`x = 2; y = 8;`	`function x = foo(z,x)`	x is 3
`z = foo(x,y);`	`y = x;`	z is 3
`fprintf('z is %d \n', z);`	`x = z + ceil(rand);`	x is 2
`fprintf('x is %d \n', x);`	`z = y;`	y is 8
`fprintf('y is %d \n', y);`	`fprintf('x is %d \n', x);`	

Use this page for scratch work.

Question 2 (8 points)

Rewrite the following code fragment without using the logical operators && or || such that the variable x is assigned the same value as the given fragment. (Do not use &, |, or xor either).

```
% assume that variables a, b, and c are each a numeric scalar

if a ~= b || a == b + c
    x = 1;
elseif c == b && a == b
    x = 2;
else
    x = 3;
end
```

Rewrite the code fragment here:

```
if a ~= b
    x = 1;
elseif a == b + c
    x = 1;
elseif c == b
    if a == b
        x = 2;
    else
        x = 3;
    end
else
    x = 3;
end
```

Use this page for scratch work.

Question 3 (17 points)

The function `big()` takes a positive integer number as input and outputs the highest value digit in the input. For example, `big(7290)` returns 9. Using the function `big()` (do not write the function), finish the script below that prints the number in 1D array v that contains the largest digit. If multiple numbers in v contain the largest digit, only the first one should be printed. Assume v is already defined, only stores positive integers, and is not empty.

Example: if v = [820, 90, 2991, 7], the code should print the number 90.
Example: if v = [10, 58, 70], the code should print the number 58.

Be efficient for full credit: if your code finds a number with a 9 in it, it should not look at any other numbers in the vector because it is not necessary. (HINT: use a `while` loop.)

```
len = length(v);

maximum = 0;
index = 0;
i = 1;

while i <= len && maximum ~= 9
    val = big(v(i));
    if val > maximum
        maximum = val;
        index = i;

    end
    i = i + 1;
end

fprintf('%d', ___v(index)___ )
```

Use this page for scratch work.

Question 4 (40 points)

In this problem, you will study two different ways of finding the "greatest common divisor" (gcd) of two positive integers. The gcd of two positive integers, `m` and `n`, is defined as the largest integer `d`, such that `d` is a divisor of `m` (equivalently, `m` is a multiple of `d`) and `d` is a divisor of `n` (equivalently, `n` is a multiple of `d`).

Question 4.1 (10 points) The brute-force way for computing the gcd of two positive integers `m` and `n` is finding all the divisors of `m` and all the divisors of `n`, deciding which divisors they have in common, and setting gcd to be the largest integer among all the common divisors of `m` and `n`. For example:

`m = 12, n = 18`
Divisors of `m`: `1, 2, 3, 4, 6, 12`
Divisors of `n`: `1, 2, 3, 6, 9, 18`
Common divisors: `1, 2, 3, 6`
Greatest common divisor: `6`

Another example: the gcd of 4 and 3 is 1, because it is the largest number that is a divisor of 4 and 3.

Implement the function `findDivisors(m)` that takes a positive integer `m` as its input, and returns the number of positive divisors of `m` and a vector that contains all the positive divisors of `m`. (Hint: use the `rem()` function)

```
function [num_divisors, divisors] = findDivisors(m)
% Determine the positive divisors of a positive integer m.
% Return variables:
%    num_divisors: scalar storing the number of positive divisors of m
%    divisors: vector storing the positive divisors of m

divisors = [];
num_divisors = 0;
for k = 1:m
        if rem(m, k) == 0
                num_divisors = num_divisors + 1;
                divisors(num_divisors) = k;
        end
end
```

Question 4.2 (20 points) Now implement the function `gcdSlow(m, n)` that makes use of the `findDivisors(m)` function in (4.1) and finds the gcd of two positive integers `m` and `n`. No built-in MATLAB functions are allowed in this question.

Use nested loops: one to loop through the divisors of m and other to loop through the divisors of n. Use these loops to find the largest number that is in both sets of divisors. Efficiency is not a concern in this part. Just build a solution that works.

```
function d_slow = gcdSlow(m, n)
% This function finds the greatest common divisor using the brute-force
% algorithm and findDivisors from 4.1.
% Returns the greatest common divisor of m and n.
% Assumes m and n are positive integers.

common = [];
[num_mdiv, m_divisors] = findDivisors(m);
[num_ndiv, n_divisors] = findDivisors(n);
num_cdiv = 0; % number of common divisors
for i = 1:num_mdiv
        dm = m_divisors(i);
        found = false;
        j = 1;
        while (~found && j <= num_ndiv)  % okay if this is just a for loop
                if dm == n_divisors(j)
                        found = true;
                end
                j = j + 1;
        end
        if (found)
                num_cdiv = num_cdiv + 1;
                common(num_cdiv) = dm;
        end
end
d_slow = common(num_cdiv);
```

Question 4.3 (10 points) The recipe for computing the gcd in (4.1) and (4.2) is correct, but it can be slow when `m` and `n` are large integers. Therefore, you will implement an efficient way to compute the gcd using the Euclidean algorithm. The Euclidean algorithm for finding the greatest common divisor of `m` and `n` is listed in the three steps below (assuming $\texttt{m} \geq \texttt{n} > 0$). The only built-in MATLAB function you can use on this part is `rem`.

1. Calculate the remainder `r` from `m` divided by `n`.
2. If `r` is zero then `n` is the gcd.
3. Otherwise, let `m = n` and then `n = r`. Repeat from step 1.

Hint: use a `while` loop.

Complete the following function which implements the Euclidean algorithm for finding the gcd.

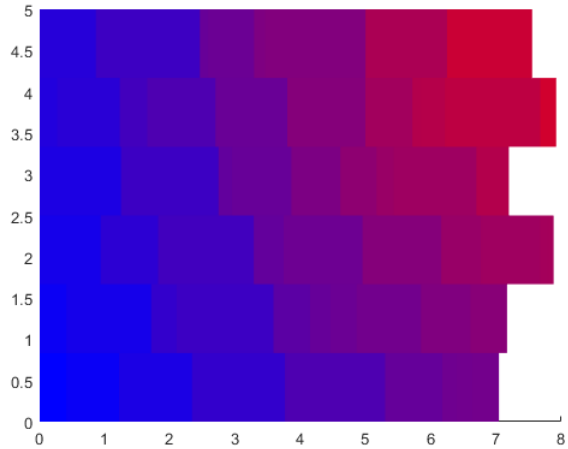
```
function d_fast = gcdFast(m, n)
% This function returns the gcd of m and n using the Euclidean algorithm.
% Assumes m and n are positive integers.
% Remember that the Euclidean algorithm requires m >= n but the inputs
% to this function do not necessarily already satisfy that criterion.


if m < n
    temp = m;
    m = n;
    n = temp;
end

r = rem(m,n);
while r ~= 0
    m = n;
    n = r;
    r = rem(m,n);
end

d_fast = n;
```

Example of the Euclidean algorithm in action when m = 21 and n = 6:

1. Divide 21 by 6 and get the remainder r = 3
2. r is not zero so m = 6, n = 3
3. Divide 6 by 3 and get the remainder r = 0
4. Since r = 0, n = 3 is the gcd

Use this page for scratch work.

Example plot from the completed script in problem 5. →



Question 5 (22 points)

Complete the script below to draw n rows of rectangles in the 2D space defined by x values [x_start, x_end] and y values [y_start, y_end]. Add code only to the boxes. See example on page 12.

All rectangles have the same vertical height of (y_end - y_start)/n. The horizontal width of each rectangle should be uniformly random in the range (0, s), where s = (x_end - x_start)/5.

Rows of rectangles are adjacent to each other. The bottom edge of the bottom-most row of rectangles is at y_start. Rectangles in the same row are adjacent to each other. In each row, start generating rectangles from left to right, with the first rectangle having its left edge at x_start. When you are drawing rectangles in any row, when you generate a rectangle width that would exceed x_end, do not draw that rectangle and start drawing the next row of rectangles.

For a rectangle with bottom left corner at (x,y), the color is determined by linear interpolation, depending on the distance between (x,y) and (x_end, y_end)—a rectangle with (x,y) = (x_start,y_start) would be blue, and a rectangle with (x,y) = (x_end, y_end) would be red. Recall the distance between arbitrary points (x1,y1) and (x2, y2) is d = sqrt((x1-x2)^2 + (y1-y2)^2).

Assume you can use function DrawRect_NB(x,y,L,W,c), which draws a rectangle with no border and with vertices (x,y), (x+L, y), (x+L, y+W) and (x, y+W).

```
close all
figure
hold on
red = [1 0 0]; blue = [0 0 1];
x_start = 0; y_start = 0;
x_end = 8; y_end = 5;
n = 6; s = (x_end - x_start)/5;

dist_max = sqrt((x_end-x_start)^2 + (y_end-y_start)^2);
y = y_start;

for row = 1:n

    x = x_start;
    length_cur = rand*s;

    while x + length_cur <= x_end
        dist = sqrt((x_end-x)^2 + (y_end-y)^2);
        colr = blue + (red - blue)*(dist - dist_max)/(0 - dist_max);
        % alternate color interpolation using textbook method
        % colr = (dist/dist_max)*blue + (1-dist/dist_max)*red;
        DrawRect_NB(x, y, length_cur, (y_end - y_start)/n, colr)
        x = x + length_cur;
        length_cur = rand*s;

    end
    y = y + (y_end - y_start)/n;

end
```

Use this page for scratch work.