CS 1112 Prelim 1 Review

What We'll Do Today

- **Review of these topics:**
  - Conditional (if-elseif-else) statements
  - Loops: for, while, nested
  - Functions
  - Vectors
  - Vectorized code & linear interpolation
- **Practice prelim questions** which involve several topics at once
- **Questions**

Yay!

Conditional statements

General form

```
if (condition1)
    % code to run if condition1 is true
elseif (condition2)
    % code to run if condition2 is true but
    % condition1 is false
else
    % code to run if all previous conditions were false
end % important to include this!
```

Conditional statements

You don’t need any more branches after the `if` branch:

```
if (condition1)
      % some code
end
```

Conditional statements

You don’t need any more branches after the `if` branch:

```
if (condition1)
      % some code
end
```

You don’t need `elseif` branches after the `if` branch:

```
if (condition1)
      % some code
else
      % ‘catch all’ condition
end
```

Conditional statements

There can be many `elseif` branches after the `if` branch:

```
if (condition1)
      % some code
elseif (condition2)
      % some code
elseif (condition3)
      % some code
else
      % 'else' not required
end
```

Conditional statements

There can be many `elseif` branches after the `if` branch:

```
if (condition1)
      % some code
elseif (condition2)
      % some code
elseif (condition3)
      % some code
else
      % 'else' not required
end
```

Can nest `if-elseif-else` branches inside any other conditional branch:

```
if (condition1)
      if (subcondition1)
          % code to run if condition1 and
          % subcondition1 are both true
      else
          % condition1 is true, subcondition1 is not
      end
elseif (condition2)
      if (subcondition2)
          % condition1 is not true, condition2
          % is true, subcondition2 is true
      elseif (subcondition3)
          % condition1 is not true, condition2 is
true,
          % subcondition2 is not true but
subcondition3
          % is true
      end
else
      % none of the previous conditions are true
end
```

Conditional statements

- Conditions must evaluate to true or false (equivalently, 1 or 0)
- Can join simple conditions together using `&&` (and), `||` (or)
- Check equality using `==` (not `=`, which is for assignment)
- Check inequality using **~=**

Conditional statements

- Conditions must evaluate to true or false (equivalently, 1 or 0)
- Can join simple conditions together using `&&` (and), `||` (or)
- Check equality using `==` (not `=`, which is for assignment)
- Check inequality using **~=**

Examples

Incorrect

```
if (a + b = 2)
     % do something if the sum
of
     % a and b is 2
end
```

Correct

```
if (a + b == 2)
     % do something if the sum
of
     % a and b is 2
```

Conditional statements

- Conditions must evaluate to true or false (equivalently, 1 or 0)
- Can join simple conditions together using `&&` (and), `||` (or) , `~` (not)
- Check equality using `==` (not `=`, which is for assignment)
- Check inequality using **~=**

Examples

Incorrect

```
if (a + b = 2)
      % do something if the sum
      % of a and b is 2
end
```

Correct

```
if (a + b == 2)
      % do something if the sum
      % of a and b is 2
end
```

```
if (a + b == 2)
      if (c + d == 3)
      % some code to run if the sum
      % of a and b is 2, and also if
      % the sum of c and d is 3
      end
end
```

The above code is equivalent to this:

```
if (a + b == 2) && (c + d == 3)
      % some code
end
```

for and while loops

I know exactly how many times I need to loop

Fixed iteration

for loop

I need to loop until some stopping condition(s)

Indefinite iteration

while loop

for and while loops

for loop

Iterates a fixed number of times

Syntax:

```
for variableName = start:stepSize:end
     % # of times this code will run:
     % floor((end-start)/stepSize) + 1
end
```

Example: Print the numbers 2, 4, 6, 8

```
for k = 2:2:8
    disp(k);
end
```

for and while loops

for loop

Iterates a fixed number of times

Syntax:

```
for variableName = start:stepSize:end
     % # of times this code will run:
     % floor((end-start)/stepSize) + 1
end
```

Example: Print the numbers 2, 4, 6, 8

```
for k = 2:2:8
    disp(k);
end
```

while loop

Iterates until a condition becomes false

Syntax:

```
while (condition is true)
    % need code that will eventually
    % cause the condition to become false
end
```

Example: Print the numbers 2, 4, 6, 8

```
k = 2;
while (k <= 8)
    disp(k);
    k = k+2;
end
```

Equivalence of for and while loops

- A while loop can do everything that a for loop can do
- The reverse is not always true (because you are not allowed to use `break` to end iteration in a `for` loop early)
- while loops are useful for not iterating more than is necessary (i.e. they can be more **efficient**) (efficiency has to do with code **speed**, not **length)**

Equivalence of for and while loops

Recall vectorQuery from lab 6: display 1 if the number r is within the first n elements of vector v; display 0 if not.

Equivalence of for and while loops

Recall vectorQuery from lab 6: display 1 if the number r is within the first n elements of vector v; display 0 if not.

Which of these is correct? If both are correct, which is better?

```
found = 0;
n = min(n, length(v));
for k = 1:n
     if(v(k) == r)
          found = 1;
     end
end
disp(found)
```

```
k = 1; found = 0;
while (k <= n && k <= length(v) &&
~found)
     if(v(k) == r)
          found = 1;
     end
     k = k+1;
end
disp(found)
```

Equivalence of for and while loops

Recall vectorQuery from lab 6: display 1 if the number r is within the first n elements of vector v; display 0 if not.

Which of these is correct? If both are correct, which is better?

```
found = 0;
n = min(n, length(v));
for k = 1:n
     if(v(k) == r)
          found = 1;
     end
end
disp(found)
```

```
k = 1; found = 0;
while (k <= n && k <= length(v) &&
~found)
     if(v(k) == r)
          found = 1;
     end
     k = k+1;
end
disp(found)
```

Answer: Both solutions are correct – however, the code on the right is more efficient because it iterates the minimum number of times necessary.

Some common loop patterns

1. Find the maximum/minimum/“best” item in a set

Example: Given a vector v, display the smallest item in v

Some common loop patterns

1. Find the maximum/minimum/“best” item in a set

Example: Given a vector v, display the smallest item in v

```
minSoFar = v(1); % Initialize “best-so-far” variable
for k = 2:length(v)
    if (v(k) < minSoFar)  % Compare “best-so-far” variable to current
        minSoFar = v(k); % element in the set and update it if
needed
    end
end
disp(minSoFar)
```

Some common loop patterns

2. Accumulation: use iteration to compute a statistic from a set of values (e.g. a sum, product, average, etc.)

Example: given a vector v, display the product of all elements in v

Some common loop patterns

2. Accumulation: use iteration to compute a statistic from a set of values (e.g. a sum, product, average, etc.)

<u>Example</u>: given a vector v, display the product of all elements in v

```
productSoFar = v(1);    % Initial value of statistic
for k = 2:length(v)
     % Update statistic by "accumulating" it with the
     % current value in the set
     productSoFar = productSoFar*v(k);
end
disp(productSoFar)
```

Some common loop patterns

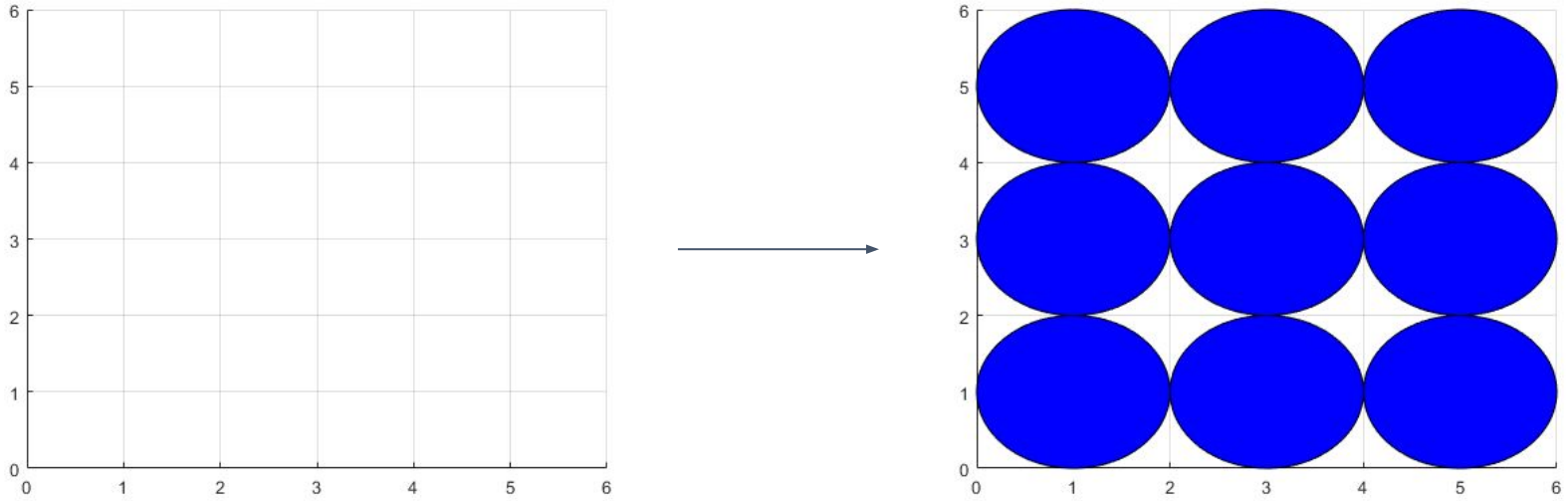
3. Iterate through all combinations of two variables with a nested loop

Example: Draw a disk of radius 1 at every other point in a n × n grid
(e.g. if n is 5, draw disks at at (1,1), (1,3), (1,5), …, (3,1), (3,3), (3,5)...)

Some common loop patterns

3. Iterate through all combinations of two variables with a nested loop

Example: Draw a disk of radius 1 at every other point in a n × n grid
(e.g. if n is 5, draw disks at at (1,1), (1,3), (1,5), (3,1), (3,3), (3,5)...)



Some common loop patterns

3. Iterate through all combinations of two variables with a nested loop

<u>Example</u>: Draw a disk of radius 1 at every other point in a n × n grid (e.g. if n is 5, draw disks at at (1,1), (1,3), (1,5), …, (3,1), (3,3), (3,5)...)

```
for x = 1:2:n % Iterate through all possible x-coordinates
    for y = 1:2:n % Iterate through all possible y-coordinates
        DrawDisk(x, y, 1, 'b')
    end
end
```

Some common loop patterns

4. Do something repeatedly until one or more conditions is/are met

Example: Generate random numbers (and display them) until we've generated 6 numbers or we get a random number greater than 0.9, *whichever happens first*.

Some common loop patterns

4. Do something repeatedly until one or more conditions is/are met

Example: Generate random numbers (and display them) until we've generated 6 numbers or we get a random number greater than 0.9, *whichever happens first*.

```
numGenerated = 1;
r = rand;
disp(r)
while (r <= 0.9 && numGenerated <= 5)  % 5 and not 6, because we already
      r = rand;                  % generated one random number before the loop
      disp(r)
      numGenerated = numGenerated + 1;
end
```

Some common loop patterns

4. Do something repeatedly until one or more conditions is/are met

Tip: It is often easier to think of a ***quitting condition*** instead of a ***continue condition*** when writing while loops. **Negate a quit condition to derive the continue condition**.

Quit condition: “Quit when x==0 && y==0 && z==0”

Continue condition: “continue while ~(x==0 && y==0 && z==0)”

same as

x~=0 || y ~= 0 || z ~= 0

```
while (x~=0 || y ~= 0 || z ~= 0)
    …
end
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

An automotive consultant has determined that car buyers care most about cost, speed, and safety, which can be computed as scores (unitless point values) in a simplified model from the engine horsepower h, car frame weight w in kilograms, and passenger capacity c as follows:

$$\text{speed} = \frac{\log(h)}{cw^2} \qquad \text{cost} = \sqrt{hc} \qquad \text{safety} = 10\frac{w}{c}$$

Use of loops
Spring 2020 Prelim 1: **Question 3**

An automotive consultant has determined that car buyers care most about cost, speed, and safety, which can be computed as scores (unitless point values) in a simplified model from the engine horsepower h, car frame weight w in kilograms, and passenger capacity c as follows:

$$\text{speed} = \frac{\log(h)}{cw^2} \qquad \text{cost} = \sqrt{hc} \qquad \text{safety} = 10\frac{w}{c}$$

(a) Implement the following function as specified:

```
function [speed , cost , safety] = calc_scores(c, w, h)
% Compute the speed , cost , and safefy scores from c, w, h
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

An automotive consultant has determined that car buyers care most about cost, speed, and safety, which can be computed as scores (unitless point values) in a simplified model from the engine horsepower h, car frame weight w in kilograms, and passenger capacity c as follows:

$$\text{speed} = \frac{\log(h)}{cw^2} \qquad \text{cost} = \sqrt{hc} \qquad \text{safety} = 10\frac{w}{c}$$

(a) Implement the following function as specified:

```
function [speed , cost , safety] = calc_scores(c, w, h)
% Compute the speed , cost , and safefy scores from c, w, h
```

Solution:

```
speed = log(h)/(w^2*c); cost = sqrt(h*c); safety = 10*w/c;
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
% Iteratively find the combination of horsepower (best_h) and weight
(best_w) that achieves the highest speed score (max_speed) in the design
of a 4-passenger car with the the following constraints:
% - possible values of engine horsepower are 50, 51, ..., 200
% - possible values of car frame weight are 1500 , 1600 , ..., 3000
% - cost score of the design cannot exceed the target cost score
(target_c) by more than 20 points
% - safety score of the design cannot differ by more than 30 points from
the target safety score (target_s)

% If multiple combinations of horsepower and weight result in the
highest speed, any one of those combinations may be returned.

% If no combination of horsepower and weight can meet the constraints ,
then set all the return parameters to 0.
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
% Iteratively find the combination of horsepower (best_h) and weight
(best_w) that achieves the highest speed score (max_speed) in the design
of a 4-passenger car with the the following constraints:
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

```
% - possible values of engine horsepower are 50, 51, ..., 200
% - possible values of car frame weight are 1500 , 1600 , ..., 3000
```

⇒ Choose h from 50, 51, ..., 200,

⇒ choose w from 1500 , 1600 , ..., 3000

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

⇒ Choose h from 50, 51, ..., 200,

⇒ choose w from 1500 , 1600 , ..., 3000

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

⇒ Choose h from 50, 51, ..., 200,

⇒ choose w from 1500 , 1600 , ..., 3000

```
% - cost score of the design cannot exceed the target cost score
(target_c) by more than 20 points
```

⇒ The cost computed from a possible combination (h,w,c) <= target_c+20

```
% - safety score of the design cannot differ by more than 30 points from
the target safety score (target_s)
```

⇒ The safety computed from a possible combination (h,w,c) has to satisfy: abs(safety - target_s) <= 30.

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

⇒ Choose h from 50, 51, ..., 200,

⇒ choose w from 1500 , 1600 , ..., 3000

⇒ The cost computed from a possible combination (h,w,c) <= target_c+20

⇒ The safety computed from a possible combination (h,w,c) has to satisfy: abs(safety - target_s) <= 30.

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

⇒ Choose h from 50, 51, ..., 200,

⇒ choose w from 1500 , 1600 , ..., 3000

⇒ The cost computed from a possible combination (h,w,c) <= target_c+20

⇒ The safety computed from a possible combination (h,w,c) has to satisfy: abs(safety - target_s) <= 30.

```
% If multiple combinations of horsepower and weight result in the
highest speed, any one of those combinations may be returned.
```

⇒ No need to store the previous best combination, always maintain the current best combination.

```
% If no combination of horsepower and weight can meet the constraints ,
then set all the return parameters to 0.
```

⇒ A simple way is to initialize all return parameters to be 0.

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

⇒ Choose h from 50, 51, ..., 200,

⇒ choose w from 1500 , 1600 , ..., 3000

⇒ The cost computed from a possible combination (h,w,c) <= target_c+20

⇒ The safety computed from a possible combination (h,w,c) has to satisfy: abs(safety - target_s) <= 30.

⇒ No need to store the previous best combination, always maintain the current best combination.

⇒ A simple way is to initialize all return parameters to be 0.

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

⇒ Choose h from 50, 51, ..., 200,

⇒ choose w from 1500 , 1600 , ..., 3000

⇒ The cost computed from a possible combination (h,w,c) <= target_c+20

⇒ The safety computed from a possible combination (h,w,c) has to satisfy: abs(safety - target_s) <= 30.

⇒ No need to store the previous best combination, always maintain the current best combination.

⇒ A simple way is to initialize all return parameters to be 0.

```
Solution:
% Step 1: Starts with initialization and fixed constants.
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, ~~with c=4 fixed~~. We know [speed , cost , safety] = calc_scores(c, w, h).

⇒ Choose h from 50, 51, ..., 200,

⇒ choose w from 1500 , 1600 , ..., 3000

⇒ The cost computed from a possible combination (h,w,c) <= target_c+20

⇒ The safety computed from a possible combination (h,w,c) has to satisfy: abs(safety - target_s) <= 30.

⇒ No need to store the previous best combination, always maintain the current best combination.

~~⇒ A simple way is to initialize all return parameters to be 0.~~

Solution:

```
% Step 1: Starts with initialization and fixed constants.
capacity= 4; max_speed= 0; best_h= 0; best_w= 0;
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

⇒ Choose h from 50, 51, ..., 200,

⇒ choose w from 1500 , 1600 , ..., 3000

⇒ The cost computed from a possible combination (h,w,c) <= target_c+20

⇒ The safety computed from a possible combination (h,w,c) has to satisfy: abs(safety - target_s) <= 30.

⇒ No need to store the previous best combination, always maintain the current best combination.

```
Solution:
capacity= 4; max_speed= 0; best_h= 0; best_w= 0;
% Step 2: Simplify the task - in this case, remove constraints.
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
⇒  Find value of h and w to maximize speed, with c=4 fixed. We know
   [speed , cost , safety] = calc_scores(c, w, h).
⇒  Choose h from 50, 51, ..., 200,
⇒  choose w from 1500 , 1600 , ..., 3000
```

```
Solution:
capacity= 4; max_speed= 0; best_h= 0; best_w= 0;
% Step 2: Simplify the task - in this case, remove constraints.
% The task is to find best speed given some h and w choices.
% Loops? How many?
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
⇒  Find value of h and w to maximize speed, with c=4 fixed. We know
   [speed , cost , safety] = calc_scores(c, w, h).
⇒  Choose h from 50, 51, ..., 200,
⇒  choose w from 1500 , 1600 , ..., 3000
```

```
Solution:
capacity= 4; max_speed= 0; best_h= 0; best_w= 0;
% Step 2: Simplify the task - in this case, remove constraints.
for h = 50:200
  for w = 1500:100:3000
    % compute the speed
    [speed , cost , safety] = calc_scores(capacity , w, h);
  end
end
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know

```
   [speed , cost , safety] = calc_scores(c, w, h).
```

~~⇒ Choose h from 50, 51, ..., 200,~~

~~⇒ choose w from 1500 , 1600 , ..., 3000~~

Solution:

```
capacity= 4; max_speed= 0; best_h= 0; best_w= 0;
for h = 50:200
  for w = 1500:100:3000
    % compute the speed
    [speed , cost , safety] = calc_scores(capacity , w, h);
  end
end
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
```

⇒ Find value of h and w to maximize speed, with c=4 fixed. We know [speed , cost , safety] = calc_scores(c, w, h).

Solution:

```
capacity= 4; max_speed= 0; best_h= 0; best_w= 0;
for h = 50:200
  for w = 1500:100:3000
    % compute the speed
    [speed , cost , safety] = calc_scores(capacity , w, h);
    if (speed > max_speed)
       max_speed= speed; best_h= h; best_w= w;
    end
  end
end
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
Solution:
capacity= 4; max_speed= 0; best_h= 0; best_w= 0;
for h = 50:200
  for w = 1500:100:3000
    % compute the speed
    [speed , cost , safety] = calc_scores(capacity , w, h);
    if (speed > max_speed)
       max_speed= speed; best_h= h; best_w= w;
    end
  end
end
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
Solution:
capacity= 4; max_speed= 0; best_h= 0; best_w= 0;
for h = 50:200
  for w = 1500:100:3000
    % compute the speed
    [speed , cost , safety] = calc_scores(capacity , w, h);
    if (speed > max_speed)
       max_speed= speed; best_h= h; best_w= w;
    end
    % Step 4: complete the task, adding constraints:
    % cost <= target_c+20, abs(safety - target_s) <= 30.
  end
end
```

Use of loops
Spring 2020 Prelim 1: **Question 3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
Solution:
capacity= 4; max_speed= 0; best_h= 0; best_w= 0;
for h = 50:200
  for w = 1500:100:3000
    % compute the speed
    [speed , cost , safety] = calc_scores(capacity , w, h);
    if (speed > max_speed) && (cost <= target_c +20) && ...
abs(safety - target_s) <= 30
       max_speed= speed; best_h= h; best_w= w;
    end
end
end
```

User-defined functions

Syntax for writing a function (with 1 input, 1 output)

```
function returnVariable = FunctionName(inputVar)
    % code goes here
    returnVariable = something
```

User-defined functions

Syntax for writing a function (with 1 input, 1 output)

```
function returnVariable = FunctionName(inputVar)
    % code goes here
    returnVariable = something
```

Syntax for writing a function (with multiple inputs, multiple outputs)

```
function [return1, return2] = FunctionName(input1,input2)
    % code goes here
    return1 = something
    return2 = something
```

User-defined functions

Syntax for writing a subfunction

```
function [rV1,...] = FunctionName(IV1,...)
    % code goes here
    % use subfunction
end

function [srV1,...]  = SubfunctionName(sIV1,...)
   % code goes here
end
```

Note that:

- We need “end” at the end of each function.
- We can NOT directly access/call a subfunction from another file.

User-defined functions: **Calling functions** Example: **2020 Spring Q1(b)**

What will be printed when the following script is executed?

Script	*Function* (in `foo.m`)
`v = [4 5 10];`	`function a = foo(b,a)`
`k = [2 3 1];`	
`a = v(k(2));`	`b = a - b;`
`fprintf('a is %d\n', a)`	`a = b^2;`
`b= 6;`	
`c= foo(a,b);`	`fprintf('b is %d\n', b)`
`fprintf('c is %d\n', c)`	`fprintf('a is %d\n', a)`
`fprintf('a is %d\n', a)`	

User-defined functions: **Calling functions**
Example: **2020 Spring Q1(b)**

foo.m file

```
function a = foo(b,a)
    b = a - b;
    a = b^2;

    fprintf('b is %d\n', b)
    fprintf('a in %d\n', a)
```

Example: foo(2, 5)
Inside the function, b = 2, a = 5
a - b = 3, so b is changed to 3.
b^2 = 9, so a is changed to 9
It prints:
“b is 3”
“a is 9“
Output = a = 9

User-defined functions: **Calling functions**
Example: **2020 Spring Q1(b)**

foo.m file

```
function a = foo(b,a)
    b = a - b;
    a = b^2;

    fprintf('b is %d\n', b)
    fprintf('a in %d\n', a)
```

script.m file

```
v = [4,5,10];
k = [2,3,1];
a = v(k(2));
fprintf('a is %d\n', a)
b = 6;
c = foo(a,b);
fprintf('c is %d\n', c)
fprintf('a is %d\n', a)
```

k(2) → 3

v(3) → 10

a is 10

b = 6

User-defined functions: **Calling functions**
Example: **2020 Spring Q1(b)**

foo.m file

```
function a = foo(b,a)
    b = a - b;
    a = b^2;

    fprintf('b is %d\n', b)
    fprintf('a in %d\n', a)
```

script.m file

```
v = [4,5,10];
k = [2,3,1];
a = v(k(2));
fprintf('a is %d\n', a)
b = 6;
c = foo(a,b);
fprintf('c is %d\n', c)
fprintf('a is %d\n', a)
```

k(2) → 3

v(3) → 10

a is 10,

b = 6

foo(a, b)?

User-defined functions: **Calling functions**
Example: **2020 Spring Q1(b)**

foo.m file

```
function a = foo(b,a)
    b = a - b;
    a = b^2;

    fprintf('b is %d\n', b)
    fprintf('a in %d\n', a)
```

Variable scope means that changing a variable in a function doesn't affect its value outside

script.m file

```
v = [4,5,10];
k = [2,3,1];
a = v(k(2));
fprintf('a is %d\n', a)
b = 6;
c = foo(a,b);
fprintf('c is %d\n', c)
fprintf('a is %d\n', a)
```

k(2) → 3

v(3) → 10

a is 10

foo(a, b) is just foo(10,6)

User-defined functions: **Calling functions**
Example: **2020 spring Q1(b)**

Script	*Function* (in `foo.m`)
`v = [4 5 10];`	`function a = foo(b,a)`
`k = [2 3 1];`	
`a = v(k(2));`	`b = a - b;`
`fprintf('a is %d\n', a)`	`a = b^2;`
`b= 6;`	
`c= foo(a,b);`	`fprintf('b is %d\n', b)`
`fprintf('c is %d\n', c)`	`fprintf('a is %d\n', a)`
`fprintf('a is %d\n', a)`	

k(2) → 3

v(3) → 10

a is 10

c = foo(10,6);

b = 6-10 → b = -4

a = (-4)^2 = 16

b is -4

a is 16

User-defined functions: **Calling functions**
Example: **2020 spring Q1(b)**

Script	*Function* (in `foo.m`)
`v = [4 5 10];`	`function a = foo(b,a)`
`k = [2 3 1];`	
`a = v(k(2));`	`b = a - b;`
`fprintf('a is %d\n', a)`	`a = b^2;`
`b= 6;`	
`c= foo(a,b);`	`fprintf('b is %d\n', b)`
`fprintf('c is %d\n', c)`	`fprintf('a is %d\n', a)`
`fprintf('a is %d\n', a)`	

a is 10

b is -4

a is 16

Output is a → output = 16

c = output = 16

c is 16

a = v(k(2)) = 10

a is 10

User-defined functions: **Calling functions**
Example: **2020 spring Q1(b)**

Script	*Function* (in `foo.m`)
`v = [4 5 10];`	`function a = foo(b,a)`
`k = [2 3 1];`	
`a = v(k(2));`	`b = a - b;`
`fprintf('a is %d\n', a)`	`a = b^2;`
`b= 6;`	
`c= foo(a,b);`	`fprintf('b is %d\n', b)`
`fprintf('c is %d\n', c)`	`fprintf('a is %d\n', a)`
`fprintf('a is %d\n', a)`	

a is 10

b is -4

a is 16

c is 16

a is 10

User-defined functions: **Things to remember**

- Variables inside a function are local to that function. This means their values are not accessible outside the function, except for the return variable
- Make sure that the function output variable is assigned a value by the time the function ends

User-defined functions: **Things to remember**

- Variables inside a function are local to that function. This means their values are not accessible outside the function, except for the return variable
- Make sure that the function output variable is assigned a value by the time the function ends
- Not all functions have outputs (e.g. DrawDisk)
- Not all functions have inputs

User-defined functions: **Things to remember**

- Variables inside a function are local to that function. This means their values are not accessible outside the function, except for the return variable
- Make sure that the function output variable is assigned a value by the time the function ends
- Not all functions have outputs (e.g. DrawDisk)
- Not all functions have inputs
- *Display*/*print* and *return* are different. If a value is printed to the command window, its value is still lost *unless* it is assigned to the output variable (returned).

User-defined functions: **Things to remember**

- Variables inside a function are local to that function. This means their values are not accessible outside the function, except for the return variable
- Make sure that the function output variable is assigned a value by the time the function ends
- Not all functions have outputs (e.g. DrawRect)
- Not all functions have inputs
- *Display*/*print* and *return* are different. If a value is printed to the command window, its value is still lost *unless* it is assigned to the output variable (returned).
- Synonymous terms: Input variable, argument, parameter to a function
- Synonymous terms: Return variable, output variable

Built-in Functions

- abs, sqrt, rem, floor, ceil, round, rand, zeros, ones, linspace, length, input, fprintf, disp, plot, bar
- n = input('please input: ');
- y = linspace(x1,x2,n); generates n points. The spacing between the points is (x2-x1)/(n-1).
- rand: generate a random number in the range (0,1)
  - Need to know how to:
    - Generate a random number v in the range (a,b)

      v = a + rand*(b-a);

      % rand*(b-a) gives random numbers in the range (0,b-a)

    - Generate a random **integer v** in the range [a,b] without using randi

      v = ceil(a-1 + rand*(b-a+1));

      v = floor (a + rand*(b-a+1));

Vectors

One way of creating a vector:

```
a = [1, 2, 3];          % Dimension 1x3
b = [1; 2; 3];          % Dimension 3x1
c = 1:3;                % Same as c = [1, 2, 3];
d = linspace(1, 3, 3);     % Same as d =[1,2,3];
```

Vectors

One way of creating a vector:

```
a = [1, 2, 3];          % Dimension 1x3
b = [1; 2; 3];          % Dimension 3x1
c = 1:3;                % Same as c = [1, 2, 3];
d = linspace(1, 3, 3);     % Same as d =[1,2,3];
```

Another way: create an empty vector, then fill it. (useful if you don't know in advance how big the vector should be)

```
c = [];
c(1) = 1; c(2) = 2; c(3) = 3;
```

Vectors

One way of creating a vector:

```
a = [1, 2, 3];          % Dimension 1x3
b = [1; 2; 3];          % Dimension 3x1
c = 1:3;                % Same as c = [1, 2, 3];
d = linspace(1, 3, 3);     % Same as d =[1,2,3];
```

Another way: create an empty vector, then fill it. (useful if you don't know in advance how big the vector should be)

```
c = [];
c(1) = 1; c(2) = 2; c(3) = 3;
```

Useful vector functions:

```
d = zeros(1,3); % [0,0,0]
e = ones(1,3);   % [1,1,1]
f = length(d);   % f is 3
```

Vectors

One way of creating a vector:

```
a = [1, 2, 3];          % Dimension 1x3
b = [1; 2; 3];          % Dimension 3x1
c = 1:3;                % Same as c = [1, 2, 3];
d = linspace(1, 3, 3);     % Same as d =[1,2,3];
```

Another way: create an empty vector, then fill it. (useful if you don't know in advance how big the vector should be)

```
c = [];
c(1) = 1; c(2) = 2; c(3) = 3;
```

Useful vector functions:

```
d = zeros(1,3); % [0,0,0]
e = ones(1,3);   % [1,1,1]
f = length(d);   % f is 3
```

Accessing an index of a vector with a loop

```
% Add 1 to each element of c and display it
for k = 1:length(c)
    c(k) = c(k) + 1;   % not c = c+1
    disp(c(k))
end
```

Using Vectors: Building vectors
Example: **2019 fall Q2(b)**

Write a function named recomputeEvens that has one input parameter v (a vector) and returns a vector of the same length whose odd-indexed elements match those of v but whose even-indexed elements are equal to the average of that element's left and right neighbors. You may assume that v has an odd length of at least 3.

Example: recomputeEvens([1 -2 4]) should return [1 2.5 4].

Note: You must write the function header along with the function body, but you do not need to write the function comment.

Using Vectors: Building vectors
Example: **2019 fall Q2(b)**

Write a function named recomputeEvens that has one input parameter v (a vector) and returns a vector of the same length whose odd-indexed elements match those of v but whose even-indexed elements are equal to the average of that element's left and right neighbors. You may assume that v has an odd length of at least 3.

Example: recomputeEvens([1 -2 4]) should return [1 2.5 4].

Solution:

```
function v = recomputeEvens(v)
    for k = 2:2:length(v)
        v(k) = (v(k - 1) + v(k + 1))/2;
    end
```

Using Vectors: Building vectors
Example: **2019 fall Q2(b)**

Write a function named recomputeEvens that has one input parameter v (a vector) and returns a vector of the same length whose odd-indexed elements match those of v but whose even-indexed elements are equal to the average of that element's left and right neighbors. You may assume that v has an odd length of at least 3.

Alternate Solution:

```
function w = recomputeEvens(v)
    for k = 1:length(v)
        if rem(k, 2) == 1
            w(k) = v(k);
        else
            w(k)=(v(k-1)+v(k+1))/2;
        end
    end
```

Vectorized Code

- Operations on a whole vector that work element-wise

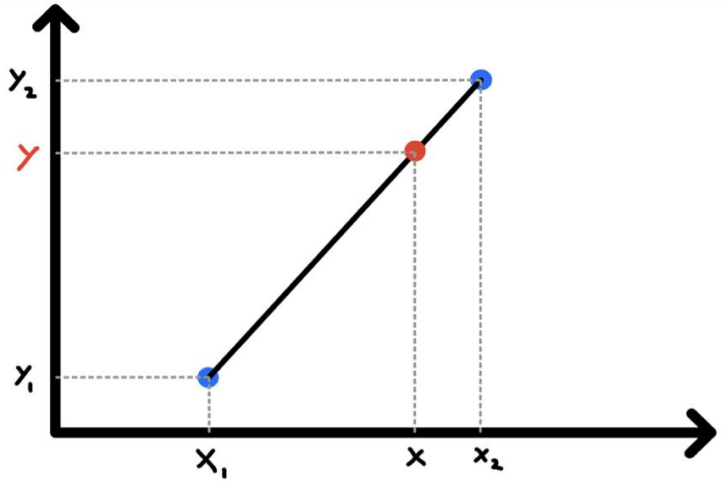
```
v = [1 2 3 4]
disp(-v)  % [-1 -2 -3 -4]
disp(v+v)  % [2 4 6 8]
disp(v.*v)  % [1 4 9 16]
disp(v.^2)  % [1 4 9 16]
disp(sin(v))  % [0.8415    0.9093    0.1411   -0.7568]
```

Linear Interpolation

Formula for linear interpolation

Given points (x1,y1) and (x2,y2), interpolate between these two points: given some new x in the interval (x1, x2), calculate the corresponding y.

How? Solve for y in terms of x:
Note: the slope from (x1, y1) to (x,y) is the same as (x1,y1) to (x2, y2).



$$\frac{y - y1}{x - x1} = \frac{y2 - y1}{x2 - x1}$$

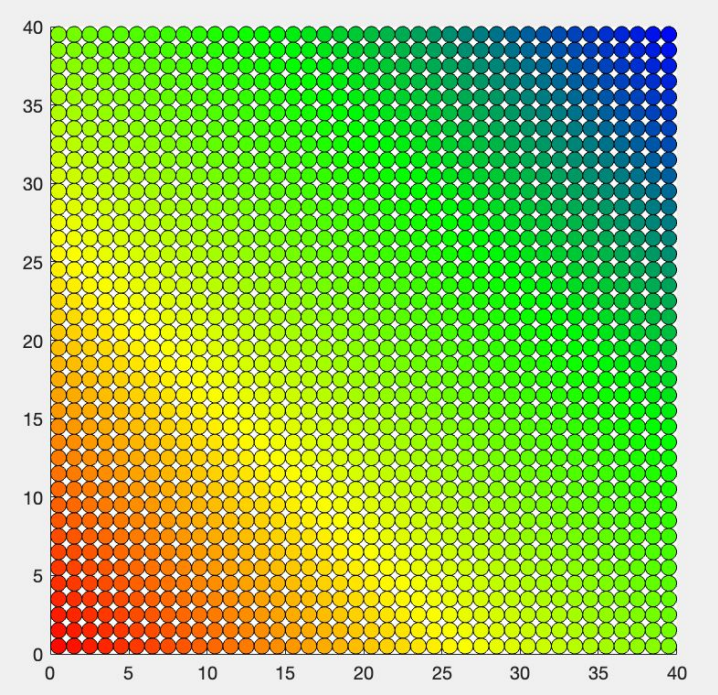
$$y = y1 + \frac{y2 - y1}{x2 - x1}(x - x1)$$

Linear Interpolation: Example

Draw an n*n (n positive integer) grid of disks in different colors interpolated between a few colors.

Each disk is drawn using the by-now familiar function DrawDisk(x, y, r, c), where the color c is determined by the value of (x + y)

The disks are centered at (i - 0.5, j - 0.5) for integers $1 \leq i, j \leq n$, and disks have r = 0.5

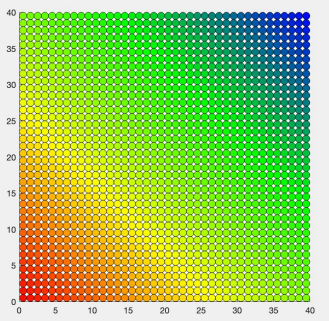


Linear Interpolation: Example

Write a function DrawRainbow(n, color1, color2, color3, color4)

The disk centered at (x, y) has color…

- color1 if x + y = 0
- A color interpolated between color1 and color2 if 0 < x + y < 2 * n / 3
- color2 if x + y = 2 * n / 3
- A color interpolated between color2 and color3 if 2 * n / 3 < x + y < 4 * n / 3
- color3 if x + y = 4 * n / 3
- A color interpolated between color3 and color4 if 4 * n / 3 < x + y < 2 * n
- color4 if x + y = 2 * n



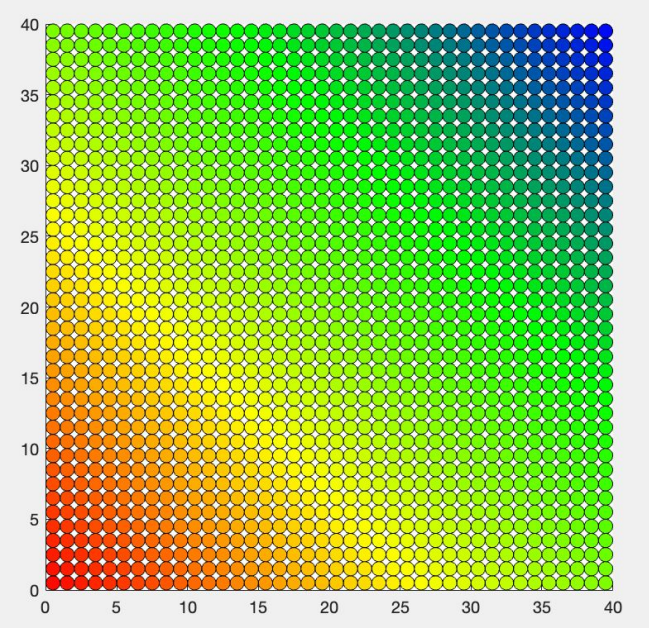
Linear Interpolation: Example

Write a function DrawRainbow(n, color1, color2, color3, color4)

First step: we should call DrawDisk in some nested for-loop!

```
axisLims = [0 n];
figure
axis equal
axis([axisLims, axisLims])
hold on
% radius of the disks
r = 0.5;
....
for i = 1:n
    for j = 1:n
        xcenter = i - 0.5;
        ycenter = j - 0.5;
        color = ...
        DrawDisk(xcenter, ycenter, r, color);
    end
end
hold off
```

What's the color here?



Linear Interpolation: Example

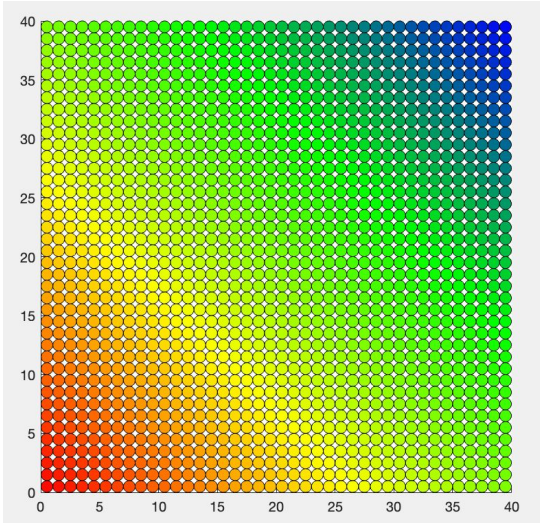
Write a function DrawRainbow(n, color1, color2, color3, color4)

Second step: determine the color of each disk!

There are 3 cases!

```
minxy = 0;
divider1 = 2 * n / 3;
divider2 = 4 * n / 3;
maxxy = 2 * n;
```

```
xysum = xcenter + ycenter;
if (xysum <= divider1)
    color = color1 + (color2 - color1) / (divider1 - minxy) * (xysum - minxy);
elseif (xysum <= divider2)
    color = color2 + (color3 - color2) / (divider2 - divider1) * (xysum - divider1);
else
    color = color3 + (color4 - color3) / (maxxy - divider2) * (xysum - divider2);
end
```



Questions?

Options:

- Questions
- More practice prelim problems

Using Vectors
Example: **2020 fall Q1.2**

Implement the following function as specified:

```
function s = maxAdjacentSum (v)
% Return the largest adjacent sum in vector v. An adjacent sum is the sum
% of two adjacent elements in a vector.
% v: a numeric vector with at least two elements
% s: the largest adjacent sum in vector v
% Example: maxAdjacentSum ([4 1 7 -1]) returns 8 since it is the
% value among the adjacent sums 4+1, 1+7, and 7+(-1)
% The only function allowed is length.
```

Using Vectors
Example: **2020 fall Q1.2**

Implement the following function as specified:

```
function s = maxAdjacentSum (v)
% Return the largest adjacent sum in vector v. An adjacent sum is the sum
% of two adjacent elements in a vector.
% v: a numeric vector with at least two elements
% s: the largest adjacent sum in vector v
% Example: maxAdjacentSum ([4 1 7 -1]) returns 8 since it is the
% value among the adjacent sums 4+1, 1+7, and 7+(-1)
% The only function allowed is length.
```

What kind of loops?

Using Vectors
Example: **2020 fall Q1.2**

Implement the following function as specified:

```
function s = maxAdjacentSum (v)
% Return the largest adjacent sum in vector v. An adjacent sum is the sum
% of two adjacent elements in a vector.
% v: a numeric vector with at least two elements
% s: the largest adjacent sum in vector v
% Example: maxAdjacentSum ([4 1 7 -1]) returns 8 since it is the
% value among the adjacent sums 4+1, 1+7, and 7+(-1)
% The only function allowed is length.
```

What kind of loops?

Fixed iteration ⇒ for loop

Using Vectors
Example: **2020 fall Q1.2**

Implement the following function as specified:

```
function s = maxAdjacentSum (v)
% Return the largest adjacent sum in vector v. An adjacent sum is the sum
% of two adjacent elements in a vector.
% v: a numeric vector with at least two elements
% s: the largest adjacent sum in vector v
% Example: maxAdjacentSum ([4 1 7 -1]) returns 8 since it is the
% value among the adjacent sums 4+1, 1+7, and 7+(-1)
% The only function allowed is length.
```

What kind of loops?

Fixed iteration ⇒ for loop

What loop pattern?

Using Vectors
Example: **2020 fall Q1.2**

Implement the following function as specified:

```
function s = maxAdjacentSum (v)
% Return the largest adjacent sum in vector v. An adjacent sum is the sum
% of two adjacent elements in a vector.
% v: a numeric vector with at least two elements
% s: the largest adjacent sum in vector v
% Example: maxAdjacentSum ([4 1 7 -1]) returns 8 since it is the
% value among the adjacent sums 4+1, 1+7, and 7+(-1)
% The only function allowed is length.
```

What kind of loops?

Fixed iteration ⇒ for loop

What loop pattern?

Find the maximum/minimum/"best" item in a set

Using Vectors
Example: **2020 fall Q1.2**

Implement the following function as specified:

```
function s = maxAdjacentSum (v)
% Return the largest adjacent sum in vector v. An adjacent sum is the sum
% of two adjacent elements in a vector.
% v: a numeric vector with at least two elements
% s: the largest adjacent sum in vector v
% The only function allowed is length.
```

Solution:

```
s= -inf; % or -realmax or the first adjacent sum
for k= 2: length(v)
     aSum= v(k-1) + v(k);
     if aSum > s
          s= aSum;
     end
end
```

Takeaway:
Maintain a "current best value" s
Each time a new item is obtained, compare it to the "current best value" s

Multi-Part Example: **2020 Spring Q3**

$$\text{speed} = \frac{\log(h)}{cw^2} \qquad \text{cost} = \sqrt{hc} \qquad \text{safety} = 10\frac{w}{c}$$

```
function [speed, cost, safety] = calc_scores (c, w, h)
% Compute the speed, cost, and safety scores from the
% passenger capacity
% ‘c’, car frame weight ‘w’, and engine horsepower `h`
% according to the consultant 's model.
```

Multi-Part Solution: **2020 Spring Q3**

$$\text{speed} = \frac{\log(h)}{cw^2} \qquad \text{cost} = \sqrt{hc} \qquad \text{safety} = 10\frac{w}{c}$$

```
function [speed, cost, safety] = calc_scores (c, w, h)
% Compute the speed, cost, and safety scores from the
% passenger capacity
% 'c', car frame weight 'w', and engine horsepower `h`
% according to the consultant 's model.

speed = log(h)/(c*w^2)
cost = sqrt(h*c)
safety = 10*w/c
```

Multi-Part Solution: **2020 Spring Q3**

```
function [max_speed, best_h, best_w] = optimize_car(target_c, target_s)
% Iteratively find the combination of horsepower (best_h) and weight
% (best_w)that achieves the highest speed score (max_speed) in the
design
% of a 4 - passenger car with the following constraints:
% - possible values of engine horsepower are 50 , 51 , ... , 200
% - possible values of car frame weight are 1500 , 1600 , ... , 3000
% - cost score of the design cannot exceed the target cost score
%   (target_c) by more than 20 points
% - safety score of the design cannot differ by more than 30 points from
%   the target safety score (target_s)
% If multiple combinations of horsepower and weight result in the
highest
% speed, any one of those combinations may be returned.
% If no combination of horsepower and weight can meet the constraints,
```

Multi-Part Solution: **2020 Spring Q3**

1. Iterate
   a. Definite iteration
   b. Iterate through h & w, according to the increments they gave

```
for h = 50:200
     for w = 1500:100:3000
```

```
     end
end
```

Multi-Part Solution: **2020 Spring Q3**

1. ~~Iterate~~
2. Initializations
   - Capacity (need for calling calc scores)
   - Max speed
   - Best h & w
   - If the algorithm doesn't find any combo, you should return 0 → so what should you initialize things as??

```
c = 4; % capacity
max_speed = 0;
best_h = 0;
best_w = 0;
for h = 50:200
     for w = 1500:100:3000




     end
end
```

Multi-Part Solution: **2020 Spring Q3**

1. ~~Iterate~~
2. ~~Initializations~~
3. Call function calc_scores
    - Inputs: Capacity, weight, & horsepower
    - Outputs: Speed, cost, & safety

```
c = 4; % capacity
max_speed = 0;
best_h = 0;
best_w = 0;
for h = 50:200
     for w = 1500:100:3000
          [speed,cost,safety] = calc_scores(c,w,h);
     end
end
```

Multi-Part Solution: **2020 Spring Q3**

1. ~~Iterate~~
2. ~~Initializations~~
3. ~~Call function~~
4. Best-so-far algorithm
   - Check if the speed is “better” (greater) than the max speed
   - Check if cost is under the target + 20
   - Check is safety score is within 30 points above or below target
   - If all these conditions are met, replace the values with the current best value

```
c = 4; % capacity
max_speed = 0;
best_h = 0;
best_w = 0;
for h = 50:200
     for w = 1500:100:3000
          [speed,cost,safety] = calc_scores(c,w,h);
          if (speed > max_speed ) && …
             (cost <= target_c +20) && …
             (safety <= target_s +30) &&
             (safety >= target_s -30)
             % OR : abs ( safety - target_s ) <= 30
               max_speed = speed;
               best_h = h;
               best_w = w;
          end
     end
end
```