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!

Poll: What do you want out of this?

- **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**

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

There can be no branches after the `if` branch:

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

Conditional statements

There can be no branches after the `if` branch:

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

There can be no `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) , `~` (not)
- 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) , `~` (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
```

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
     % Number 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
     % Number 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 to have 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;
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;
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. (For example, think about when r is found *before* the nth index of 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

Some common loop patterns

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

<u>Example</u>: 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)...)

```
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”

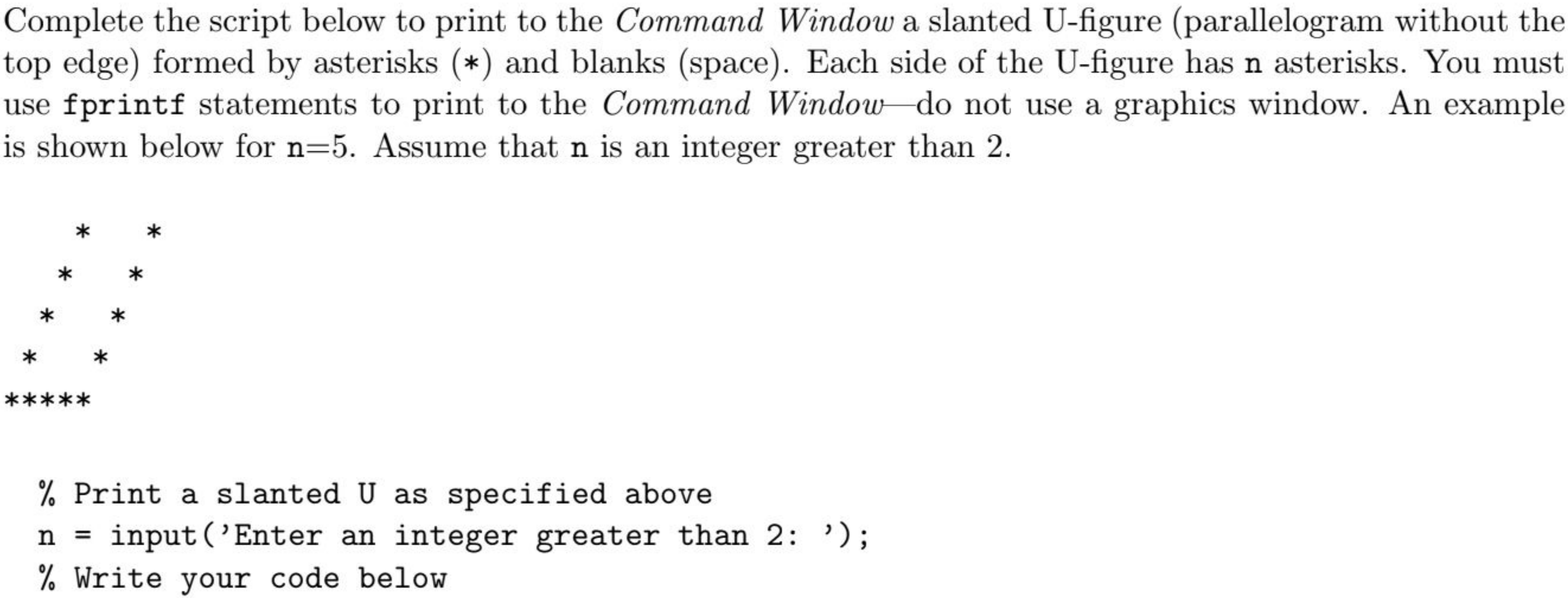
Continue condition: “continue while ~(x==0 && y==0 && z==0)”

same as

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

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

Use of loops
Spring 2018 Prelim: **Question 4**

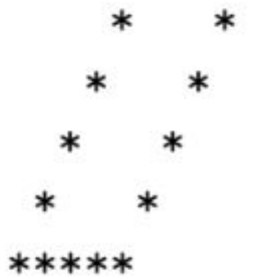


Complete the script below to print to the *Command Window* a slanted U-figure (parallelogram without the top edge) formed by asterisks (∗) and blanks (space). Each side of the U-figure has **n** asterisks. You must use **fprintf** statements to print to the *Command Window*—do not use a graphics window. An example is shown below for **n**=5. Assume that **n** is an integer greater than 2.

```
    *   *
   *   *
  *   *
 *   *
*****
```

```
  % Print a slanted U as specified above
  n = input(’Enter an integer greater than 2: ’);
  % Write your code below
```

Use of loops
Spring 2018 Prelim: **Question 4**

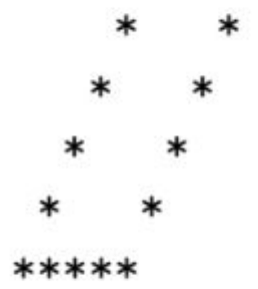


```
    *   *
   *   *
  *   *
 *   *
*****
```

Breaking down the problem:

- Think in structure first
- Then fill in the details
- What is important to the problem?
- Break into smaller problems
  - Assume you’ll be able to do a sub-task
  - Ask “What do I need to know for Task A?”
  - Then, “How do I write code for Task A?”

Use of loops
Spring 2018 Prelim: **Question 4**



```
    *   *
   *   *
  *   *
 *   *
*****
```

Breaking down the problem:

1. We need a loop. (over what?)
2. Loop over the lines*
   - Deciding what to do for each line will be manageable
3. Exactly n-1 lines: for loop

Use of loops
Spring 2018 Prelim: **Question 4**

```
    *   *
   *   *
  *   *
 *   *
*****
```

Breaking down the plroblem:

1. We need a loop. (over what?)
2. Loop over the lines*
   - ○ Deciding what to do for each line will be manageable
3. Exactly n-1 lines: for loop

```
n = input('Enter an integer greater than 2:
');

for line=1:(n-1)





end
```

Use of loops
Spring 2018 Prelim: **Question 4**

```
    *   *
   *   *
  *   *
 *   *
*****
```

Breaking down the problem:

1. We need a loop. (over what?)
2. Loop over the lines*
   - ○ Deciding what to do for each line will be manageable
3. Exactly n-1 lines: for loop
4. How do I print a given line?
   - ○ What do I need to know?

```
n = input(‘Enter an integer greater than 2:
’);

for line=1:(n-1)
    num_leading_spaces = n-line;
    num_middle_spaces = n-2;









end
```

Use of loops
Spring 2018 Prelim: **Question 4**

```
    *   *
   *   *
  *   *
 *   *
*****
```

Breaking down the problem:

1. We need a loop. (over what?)
2. Loop over the lines*
   - ○ Deciding what to do for each line will be manageable
3. Exactly n-1 lines: for loop
4. How do I print a given line?
   - ○ What do I need to know?
   - ○ How do I do it?

```
n = input(‘Enter an integer greater than 2:
’);

for line=1:(n-1)
    num_leading_spaces = n-line;
    num_middle_spaces = n-2;

    for i=1:num_leading_spaces
        fprintf(‘ ’)
    end
    fprintf(‘*’)
    for i=1:num_middle_spaces
        fprintf(‘ ’)
    end
    fprintf(‘*’)
end
```

Use of loops
Spring 2018 Prelim: **Question 4**

```
    *   *
   *   *
  *   *
 *   *
*****
```

Breaking down the problem:

1. We need a loop. (over what?)
2. Loop over the lines*
   - Deciding what to do for each line will be manageable
3. Exactly n-1 lines: for loop
4. How do I print a given line?
   - What do I need to know?
   - How do I do it?
5. Special case for final line.

```
n = input(‘Enter an integer greater than 2: ’);

for line=1:(n-1)
    num_leading_spaces = n-line;
    num_middle_spaces = n-2;

    for i=1:num_leading_spaces
        fprintf(‘ ’)
    end
    fprintf(‘*’)
    for i=1:num_middle_spaces
        fprintf(‘ ’)
    end
    fprintf(‘*\n’)
end

for i=1:n
    fprintf(‘*’)
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: **2017 spring Q 1(b)**

foo.m file

```
function z = foo(x,y)
    z = y + 1;
    x = x + 6;
    y = 2;
    fprintf(‘x is %d\n’, x)
    fprintf(‘z in %d\n’, z)
end
```

Note that:

- It is incorrect to initialize input variables inside the function.
- It is safe to first initialize return variables. If the loop doesn’t get executed, the return variable found never gets created and assigned.

User-defined functions: **Calling functions** Example: **2017 spring Q 1(b)**

foo.m file

```
function z = foo(x,y)
    z = y + 1;
    x = x + 6;
    y = 2;
    fprintf(‘x is %d\n’, x)
    fprintf(‘z in %d\n’, z)
end
```

script.m file

```
x = 4;
y = 12;
z = foo(x, x)

fprintf(‘z is %d\n’, z)
fprintf(‘x is %d\n’, x)
fprintf(‘y is %d\n’, y)
```

User-defined functions: **Calling functions**
Example: **2017 spring Q 1(b)**

foo.m file

```
function z = foo(x,y)
    z = y + 1;
    x = x + 6;
    y = 2;
    fprintf(‘x is %d\n’, x)
    fprintf(‘z in %d\n’, z)
end
```

script.m file

```
x = 4;
y = 12;
z = foo(x, x)

fprintf(‘z is %d\n’, z)
fprintf(‘x is %d\n’, x)
fprintf(‘y is %d\n’, y)
```

User-defined functions: **Calling functions** Example: **2017 spring Q 1(b)**

foo.m file

```
function z = foo(x,y)
    z = y + 1;
    x = x + 6;
    y = 2;
    fprintf('x is %d\n', x)
    fprintf('z in %d\n', z)
end
```

script.m file

```
x = 4;
y = 12;
z = foo(x, x)

fprintf('z is %d\n', z)
fprintf('x is %d\n', x)
fprintf('y is %d\n', y)
```

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

```
x is 10
z is 5
z is 5
x is 4
y is 12
```

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. 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).
- 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: **2018 spring Q2(a)**

Complete the following function:

```
function [ints, other] = getInts(v)
% Separate the integer values from non-integer values in vector v.
% v: a non-empty vector of type double
% ints: a vector storing only the integer values in v; ints may be empty.
% other: a vector storing only the non-integer values in v; other may be empty.
% Example: If v is [3 2.1 3 7] then ints is [3 3 7] and other is [2.1]
%
% Hint: A type double scalar x has an integer value if x divided by 1 results
% in a zero as the remainder.
%
% DO NOT use vectorized code.
```

Using Vectors: Building vectors Example: **2018 spring Q2(a)**

Complete the following function:

```
function [ints, other] = getInts(v)
% Separate the integer values from non-integer values in vector v.

ints = []; other = [];  % start with lengths 0, build as we go
intsIdx = 1; otherIdx = 1;
for idx=1:length(v)
    if rem(v(idx), 1) == 0  % then it’s an integer
        ints(intsIdx) = v(idx);  % builds the array
        intsIdx = intsIdx + 1;
    else
        other(otherIdx) = v(idx);
        otherIdx = otherIdx + 1;
    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

- You know f(x1) and f(x2)
- What are the values in between?

```
val1 = f(x1)
val2 = f(x2)
values = linspace(val1, val2, 300)  % linear interpolation
% spacing here is (val2-val1)/299
t = 0.3
value = t * val1 + (1-t) * val2  % also linear interpolation
```

Linear interpolation: Example

- Interpolate the colors between red [1 0 0] and blue [0 0 1]

```
figure; hold on;
n = 300;
for k=1:n
    f = ??
    col = (1-f)*[1 0 0] + f*[0 0 1];
    plot([k, k], [0, 1], ‘color’, col)
end
```

Linear interpolation: Example

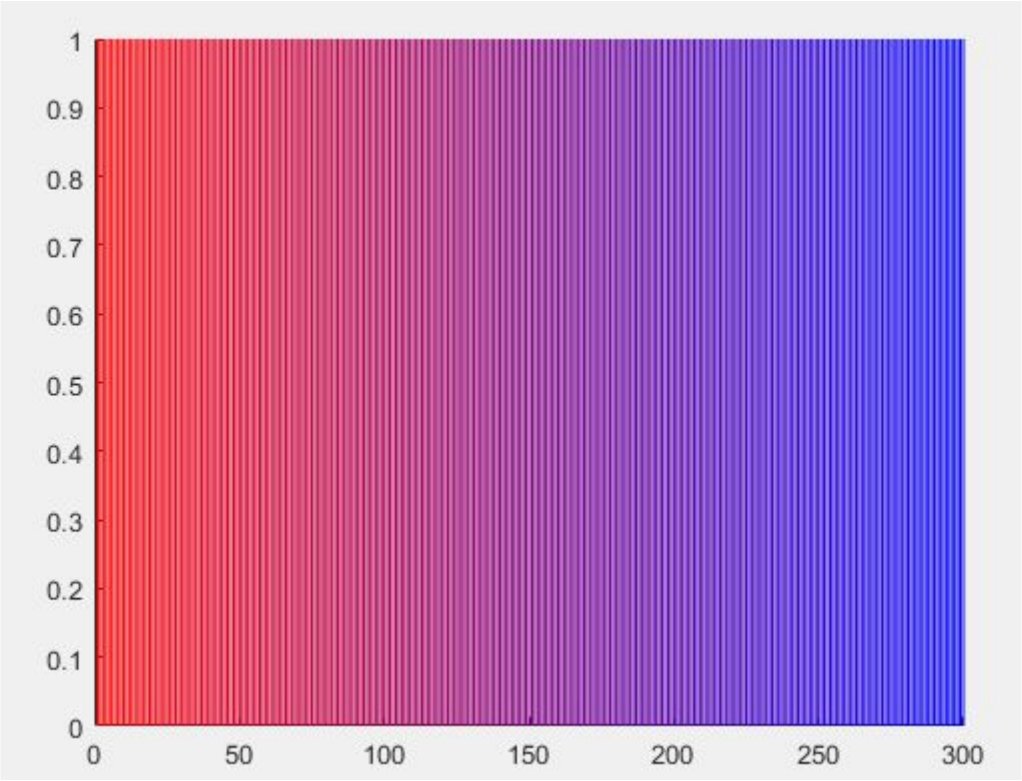
- Interpolate the colors between red [1 0 0] and blue [0 0 1]

```
figure; hold on;
n = 300;
for k=1:n
    f = (k-1)/(n-1);
    col = (1-f)*[1 0 0] + f*[0 0 1];
    plot([k, k], [0, 1], ‘color’, col)
end
```

Linear interpolation: Example

- Interpolate the colors between red [1 0 0] and blue [0 0 1]

```
figure; hold on;
n = 300;
for k=1:n
    f = (k-1)/(n-1);
    col = (1-f)*[1 0 0] + f*[0 0 1];
    plot([k, k], [0, 1], ‘color’, col)
end
```



Questions?

Options:

- Questions
- More practice prelim problems

Using Vectors
Example: **2018 spring Q3**

Complete the following function:

```
function n = howMany(v, s)
% Find the largest n such that the first n components in vector v have a sum
% strictly less than s. v is a non-empty vector with positive values; s is a
% scalar. Note that n may be zero.
% Example: if v is [5 1 4 6] and s is 10 , then n should be 2.
% DO NOT USE ANY BUILT-IN FUNCTIONS OTHER THAN length.
```

Using Vectors
Example: **2018 spring Q3**

Complete the following function:

```
function n = howMany(v, s)
% Find the largest n such that the first n components in vector v have a sum
% strictly less than s. v is a non-empty vector with positive values; s is a
% scalar. Note that n may be zero.
% Example: if v is [5 1 4 6] and s is 10 , then n should be 2.
% DO NOT USE ANY BUILT-IN FUNCTIONS OTHER THAN length.
```

If you’re not sure how to start, do an example by hand:

s = 10

[5 1 4 6]

total = 0

Using Vectors
Example: **2018 spring Q3**

Complete the following function:

```
function n = howMany(v, s)
% Find the largest n such that the first n components in vector v have a sum
% strictly less than s. v is a non-empty vector with positive values; s is a
% scalar. Note that n may be zero.
% Example: if v is [5 1 4 6] and s is 10 , then n should be 2.
% DO NOT USE ANY BUILT-IN FUNCTIONS OTHER THAN length.
```

If you’re not sure how to start, do an example by hand:

s = 10

[5 1 4 6]

⇧

idx = 1

total = 5

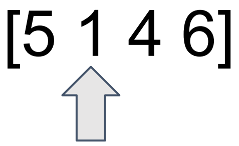
Using Vectors Example: **2018 spring Q3**

Complete the following function:

```
function n = howMany(v, s)
% Find the largest n such that the first n components in vector v have a sum
% strictly less than s. v is a non-empty vector with positive values; s is a
% scalar. Note that n may be zero.
% Example: if v is [5 1 4 6] and s is 10 , then n should be 2.
% DO NOT USE ANY BUILT-IN FUNCTIONS OTHER THAN length.
```

If you're not sure how to start, do an example by hand:

s = 10



[5 1 4 6]

idx = 2

total = 6

Using Vectors
Example: **2018 spring Q3**

Complete the following function:

```
function n = howMany(v, s)
% Find the largest n such that the first n components in vector v have a sum
% strictly less than s. v is a non-empty vector with positive values; s is a
% scalar. Note that n may be zero.
% Example: if v is [5 1 4 6] and s is 10 , then n should be 2.
% DO NOT USE ANY BUILT-IN FUNCTIONS OTHER THAN length.
```

If you’re not sure how to start, do an example by hand:

s = 10

[5 1 4 6]

⇧

idx = 3

total = 11>10 **STOP!**

Using Vectors
Example: **2018 spring Q3**

Complete the following function:

```
function n = howMany(v, s)
% Find the largest n such that the first n components in vector v have a sum
% strictly less than s. v is a non-empty vector with positive values; s is a
% scalar. Note that n may be zero.
% Example: if v is [5 1 4 6] and s is 10 , then n should be 2.
% DO NOT USE ANY BUILT-IN FUNCTIONS OTHER THAN length.
```

s = 10
[5 1 4 6]
↑
idx = 3
total = 11>10 **STOP!**

Indefinite iteration →	while loop
total →	accumulator
idx →	index
stop when total > s →	while condition

Using Vectors
Example: **2018 spring Q3**

Complete the following function:

```
function n = howMany(v, s)
% Find the largest n such that the first n components in vector v have a sum
% strictly less than s. v is a non-empty vector with positive values; s is a
% scalar. Note that n may be zero.
% Example: if v is [5 1 4 6] and s is 10 , then n should be 2.
% DO NOT USE ANY BUILT-IN FUNCTIONS OTHER THAN length.

idx=1;
total=0;
while total < s && idx <= length(v)
    total = total + v(idx);
    idx = idx + 1;
end
n = idx - 1;
```

Indefinite iteration	⇨	while loop
total	⇨	accumulator
idx	⇨	index
stop when total > s	⇨	while condition