Previous class:
- User-defined function
- Nested loops

Now:
- Working with colors
- 1-dimensional array—vector
- Algorithm for finding the best item in a set

Generating tables and plots

<table>
<thead>
<tr>
<th>x</th>
<th>sin(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.784</td>
<td>0.707</td>
</tr>
<tr>
<td>1.571</td>
<td>1.000</td>
</tr>
<tr>
<td>2.357</td>
<td>0.707</td>
</tr>
<tr>
<td>3.142</td>
<td>0.000</td>
</tr>
<tr>
<td>3.927</td>
<td>-0.707</td>
</tr>
<tr>
<td>4.712</td>
<td>-1.000</td>
</tr>
<tr>
<td>5.498</td>
<td>-0.707</td>
</tr>
<tr>
<td>6.283</td>
<td>0.000</td>
</tr>
</tbody>
</table>

x = linspace(0, 2*pi, 9);
y = sin(x);
plot(x, y);

Note: x, y are shown above as columns due to screen space; they are rows.

Built-in function linspace

x = linspace(1, 3, 5)

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
</tr>
</tbody>
</table>

x = linspace(0, 1, 101)

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
</tr>
</tbody>
</table>

How did we get all the sine values?

Built-in functions accept arrays

<table>
<thead>
<tr>
<th>x</th>
<th>sin(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.0</td>
</tr>
<tr>
<td>1.57</td>
<td>1.0</td>
</tr>
<tr>
<td>3.14</td>
<td>0.0</td>
</tr>
<tr>
<td>4.71</td>
<td>-1.0</td>
</tr>
<tr>
<td>6.28</td>
<td>0.0</td>
</tr>
</tbody>
</table>

and return arrays

| 0.00 | 1.00 | 0.00 | -1.00 | 0.00 |

Vectorized addition

| x    | 2 | 1.5 | 8 |
| y    | 1 | 2   | 0 | 1 |
| z    | 3 | 3.5 | 9 |

Matlab code: z = x + y

Vectorized multiplication

| a    | 2 | 1.5 | 8 |
| b    | 1 | 2   | 0 | 1 |
| c    | 2 | 2   | 0 | 8 |

Matlab code: c = a .* b
Vectorized element-by-element arithmetic operations on arrays:

+ \rightarrow 
\cdot \rightarrow 
\div \rightarrow 
A dot (.) is necessary in front of these math operators.

Shift:

\[ x + y = z \]

Matlab code: \( z = x + y \)

Reciprocate:

\[ \frac{x}{y} = z \]

Matlab code: \( z = x ./ y \)

Vectorized element-by-element arithmetic operations between an array and a scalar:

\[ \cdot \rightarrow \]
\[ .^2 \rightarrow \]
\[ .^\star \rightarrow \]
\[ .\div \rightarrow \]
A dot (.) is necessary in front of these math operators.

The dot in not necessary but OK.

Color is a 3-vector, sometimes called the RGB values:

- Any color is a mix of red, green, and blue.
- Example: \( \text{colr} = [0.4 \ 0.6 \ 0] \)
- Each component is a real value in \([0,1]\).
- \([0 \ 0 \ 0]\) is black.
- \([1 \ 1 \ 1]\) is white.
- \([0.2 \ 0.2 \ 0.2]\) is dark gray.
- \([0.4 \ 0.6 \ 0.1]\) is a colorized hue.

Mix two colors

Implement this function:

```matlab
function newc = mixEqual(c1,c2)
% Average colors c1 and c2.
% c1, c2, and newc are vectors representing colors.
% Display the three colors.
```
An array is a named collection of like data organized into rows or columns. A 1-d array is a row or a column, called a vector. An index identifies the position of a value in a vector.

Array index starts at 1

Let k be the index of vector x, then
- k must be a positive integer
- \(1 \leq k \leq \text{length}(x)\)
- To access the \(k^{th}\) element: \(x(k)\)

Accessing values in a vector

Given the vector \(\text{score}\) ...

```matlab
score(4) = 80;
score(5) = (score(4)+score(5))/2;
k = 1;
score(k+1) = 99;
```

A few different ways to create a vector

(More later!)

- \(\text{count} = \text{zeros}(1,6)\)
- \(\text{x} = \text{linspace}(10,30,5)\)
- \(\text{y} = [3 \ 7 \ 2 \ 1]\)
- \(\text{z} = [3; \ 7; \ 2]\)

Drawing a single line segment

```matlab
a = 0; \% x-coord of pt 1
b = 1; \% y-coord of pt 1
c = 5; \% x-coord of pt 2
d = 3; \% y-coord of pt 2
plot([a c], [b d], '-*')
```

Drawing a polygon (multiple line segments)

```matlab
% Draw a rectangle with the lower-left corner at (a,b), width w, height h.
x = [ ]; \% x data
y = [ ]; \% y data
plot(x, y)
```

Fill in the missing vector values!
Example
- Write a program fragment that calculates the cumulative sums of a given vector \( v \).
- The cumulative sums should be stored in a vector of the same length as \( v \).

\[ 1, 3, 5, 0 \quad v \]
\[ 1, 4, 9, 9 \quad \text{cumulative sums of } v \]

Common loop pattern to process a vector

```matlab
% v is a given vector
for k = 1:length(v)
    % work with v(k)
end
```

Algorithm: Finding the best in a set

```matlab
Init bestSoFar
Loop over set
    if current is better than bestSoFar
        bestSoFar = current
    end
end
```