Character Arithmetic

- You can do "math" with characters

```
'd' – 'a'          % Produces 3
'9' – '8'          % Produces 1
'a' < 'd'          % Produces 1 (= true)
'd' < 'b'          % Produces 0 (= false)
'Z' < 'b'          % Produces 1 (= true)
                   % Because 90, the ASCII code for 'Z',
                   % is less than 98, the ASCII code for 'b'

'a' + 2            % Produces 99
char('a'+2)        % Produces 'c'
```

Extracting Substrings

```
s = 'abcdef';

x = s(3)           % x = 'c'
x = s(2:4)         % x = 'bcd'
x = s(length(s))   % x = 'f'
```

Using the Word "end"

- In Matlab, the work "end" is *overloaded*
  - Used to terminate an if-statement
  - Used to terminate a for-statement
  - Used to terminate a while-statement
  - Used to represent the *last index* of a vector

```
s = 'abcdef';

x = s(end)         % x = 'f'
y = s(3:end)       % y = 'cdef'
```

Replacing Substrings

```
s = 'abcde';

s(2:4) = 'xyz'     % s = 'axyze'


s = 'abcde';

s(2:4) = 'wxyz'    % Error
```

- Dimensions must match

Dot-Operators

- Matlab is especially set up for Linear Algebra
  - Thus, "*", "/", and "^" correspond to matrix operations
- Term-by-term operators use ".*", "./", and ".^"
  - Matlab documentation calls these "array operations" (as opposed to "matrix operations")
- Why doesn't Matlab include operators ".+" and ".-"?

Shapes Must Match

- Examples

```
a = [4 8 12]
b = [1; 2; 4]  % Column vector

a + b          % Error
a + b'         % [5 10 16]

a ./ b         % Error
a' ./ b        % [4; 4; 3]
```

- Exception to shape matching
  - Scalars follow special rules
  - "A scalar can operate into anything"
- Scalar examples

```
a + 1          % [5 9 13]
10 + a         % [14 18 22]
2 .* a         % [8 16 24]
a ./ 2         % [2 4 6]
24 ./ a        % [6 3 2]
a .^ 2         % [16 64 144]
```

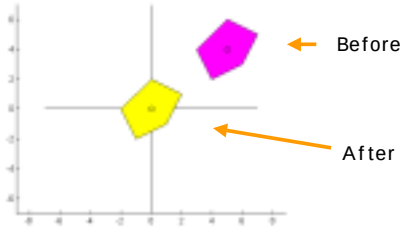
Relational Operators

- Comparison operators (e.g., “<”, “>”, “==”, etc.) also operate term-by-term, creating arrays of boolean values
- Examples

```
a = [7 0 5 2 4 6]
b = 1:6
a < b        % [0 1 0 1 1 0]
a == b       % [0 0 0 0 0 1]
```

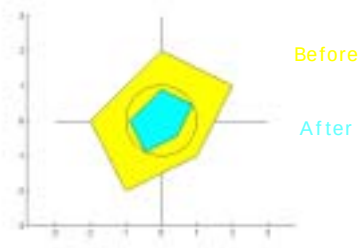
Operation 1: Centralize

- Move a polygon so that its center (the centroid of its vertices) is at the origin



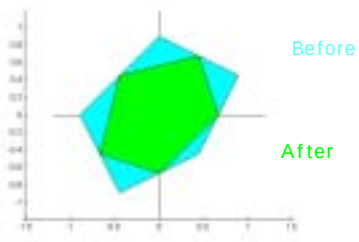
Operation 2: Normalize

- Shrink (or enlarge) the polygon so that the vertex furthest from the origin is on the unit circle



Operation 3: Smooth

- Create a new polygon by connecting the midpoints of the polygon edges

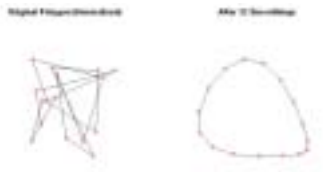


Proposed Simulation

Create a polygon with randomly located vertices

Repeat:

- Centralize
- Normalize
- Smooth



Random Walk Simulation

Start at the middle tile

Repeat until boundary reached:

- Pick a compass heading (N, E, S, W) at random
- Move one tile in that direction