Lesson 04: Functions

Now that we've gotten past vectorization, we're ready to step into the world of abstraction. Functions in any programming language allow us to modularize and organize our code but, most of all, reduce repetition.

Control Flow

Since we can't vectorize every operation, sometimes we may need to resort to traditional control flow structures you might come to expect from a programming language.

If...Else Statements

```
year = 2024;

if year < 1859
    display('Pre-Cooper Days');
elseif year == 1984
    display('Literaly 1984');
elseif year < 2011
    display('Free Education');
else
    display('Free Cooper Union');
end
```

For Loops

```
factorial = 1;

for i = 1:10
    if i == 1
        continue
    end

    if i == 9
        break
    end

    factorial = factorial * i;
end
```


While Loops

```
four = -1;

while four ~= 4
    four = randi(2, 2);
    four = sum(four, 'all');
end
```

Try...Catch Blocks

```
try
    a = always_fails(3);
catch
    warning('Problem using function, assigning a value of 0.');
    a = 0;
end
```

Anonymous Functions

An anonymous function is a function that is not stored in a program file, but is associated with a variable whose data type is `function_handle`. Anonymous functions can accept multiple inputs and return one output. They can contain only a single executable statement.[1]

We can create a function that squares like so:

```
sq = @(x) x ^ 2;
```

Anonymous functions also capture context:

```
c = 3;
cap = @(x) c * x;
c = 2;

cap(1)
```

```
ans =

     3
```

[1] https://www.mathworks.com/help/matlab/matlab_prog/anonymous-functions.html

Function Handles

Function handles are essentially watered-down function pointers and allow us to do some pretty magical stuff:

```
trig = {@sin, @cos, @tan};

x = pi / 6;
y = zeros(1, numel(trig));

for i = 1:numel(trig)
    y(i) = trig{i}(x);
end

display(y);
```

```
y =

    0.5000    0.8660    0.5774
```

Standalone Functions

Functions in MATLAB can be defined in multiple ways, *however*, the most sane way is to have them stored in a standalone M-file. Take for example `fib.m`:

```
function x = fib(n)
    if n <= 1
        x = 1;
        return
    end

    x = fib(n - 1) + fib(n - 2);
end
```

If `fib.m` is in our MATLAB path we can then call it like any other function:

```
fib(25)
```

```
ans =

      121393
```


Now for a few notes about function definitions:[2]

- If you wish to define a function in a script file, you can only do so at the *end* of the file.
- For standalone functions, you may additionally define local functions *after* your standalone function.
- Nested functions lexically scope to the function in which they're defined and are free to access variables of the parent function.

Multiple Inputs/Outputs

```
function [sz, A] = outer_product(u, v)
    A = u' * v;
    sz = size(A);
end
```

Aside— If you're interested in how to define functions with variable-length inputs/outputs, feel free to check out `varargin` and `varargout`.

We can then call the function like so:

```
x = [1, 2];
y = [3, 4];

[sz, B] = outer_product(x, y)
```

```
sz =

     2     2


B =

     3     4
     6     8
```

Classes

Classes are beyond the scope of this course, but MATLAB's online documentation is a great starting point for the interested reader: `https://www.mathworks.com/help/matlab/object-oriented-programming.html`.

[2] `https://www.mathworks.com/help/matlab/matlab_prog/types-of-functions.html`

