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

```matlab
1   year = 2024;
2
3   if year < 1859
4       display('Pre-Cooper Days');
5   elseif year == 1984
6       display('Literaly 1984');
7   elseif year < 2011
8       display('Free Education');
9   else
10      display('Free Cooper Union');
11  end
```

For Loops

```matlab
1   factorial = 1;
2
3   for i = 1:10
4       if i == 1
5           continue
6       end
7
8       if i == 9
9           break
10      end
11
12      factorial = factorial * i;
13  end
```

While Loops

```
1  four = -1;
2
3  while four ~= 4
4      four = randi(2, 2);
5      four = sum(four, 'all');
6  end
```

Try...Catch Blocks

```
1  try
2      a = always_fails(3);
3  catch
4      warning('Problem using function, assigning a value of 0.');
5      a = 0;
6  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:

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

Anonymous functions also capture context:

```
1  c = 3;
2  cap = @(x) c * x;
3  c = 2;
4
5  cap(1)
```

```
1  ans =
2
3        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:

```
1  trig = {@sin, @cos, @tan};
2
3  x = pi / 6;
4  y = zeros(1, numel(trig));
5
6  for i = 1:numel(trig)
7      y(i) = trig{i}(x);
8  end
9
10 display(y);
```

```
1  y =
2
3      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`:

```
1  function x = fib(n)
2      if n <= 1
3          x = 1;
4          return
5      end
6
7      x = fib(n - 1) + fib(n - 2);
8  end
```

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

```
1  fib(25)
```

```
1  ans =
2
3      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

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

We can then call the function like so:

```
1   x = [1, 2];
2   y = [3, 4];
3
4   [sz, B] = outer_product(x, y)
```

───────────────────────────

```
1   sz =
2
3       2      2
4
5
6   B =
7
8       3      4
9       6      8
```

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

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`