# 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

1

```
year = 2024;
2
   if year < 1859
3
       display('Pre-Cooper Days');
4
   elseif year == 1984
5
       display('Literaly 1984');
6
   elseif year < 2011
       display('Free Education');
   else
       display('Free Cooper Union');
10
   end
   For Loops
   factorial = 1;
2
   for i = 1:10
3
        if i == 1
4
            continue
       end
       if i == 9
            break
        end
       factorial = factorial * i;
12
   end
13
```

# 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;
```

# **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.<sup>1</sup>

We can create a function that squares like so:

```
sq = 0(x) x^2;
```

end

Anonymous functions also capture context:

 $<sup>^{1} \</sup>verb|https://www.mathworks.com/help/matlab/matlab_prog/anonymous-functions.html|$ 

# **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 =</pre>
```

121393

Now for a few notes about function definitions:<sup>2</sup>

- 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;
2
       sz = size(A);
   end
   We can then call the function like so:
   x = [1, 2];
   y = [3, 4];
2
3
   [sz, B] = outer_product(x, y)
4
   sz =
2
         2
                2
4
5
   B =
6
         3
                4
8
         6
                8
```

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

 $<sup>^2 \</sup>verb|https://www.mathworks.com/help/matlab/matlab_prog/types-of-functions.html|$