```
###

The programs in this document introduce:

    • (Custom) Functions


Interface programming is mostly about telling the computer to run some code when something happens.
We chunk together code and refer to it by writing custom functions.

###
```

```
12
13  ######################################## P0
14  # Before we get into details here's a tiny example of a basic interaction using a custom function.
15  # Don't fret about the bits we haven't talked about yet.
16
17  box = new Layer
18  box.onMouseDown( -> print("CLICK!") )   # When we press on the box, print "CLICK!"
19
20
```

```
21  ######################################### P1
22  # In CoffeeScript, a custom function or 'function literal' follows this pattern:
23  #
24  # -> CODE
25  #
26
27  # Example:
28  -> print("Hello?")
29
30  # The '->' means 'new function here'
31  # The code that comes after -> is the 'function body'
32  # All together this is called a 'function literal'. It is literally a function.
33  # The code of the function literal above will not run immediately. It's only stored for later. However, we have no way to refer to it.
34  # To use a custom function we need to store it in a variable; naming it in a sense.
35
36  shout = -> print("AH!") # Create and store a function in the variable shout.
37
38  shout() # Call our function by name,  "AH!"
39  shout() # We can call it many times,  "AH!"
40  shout() # ...                        "AH!"
41
42  # When we call the function shout, the computer jumps to the function code on line 36,
43  #   runs the code, then goes back to where it left off.
44  # In this way, our programs don't HAVE to run from top to bottom.
45  # We can use functions to make the computer jump around to different parts and even reuse chunks of code.
46  # Normally, our functions will have more code than just print.
47  # The shout function here is just a very simple example.
48
49
```

```
50      ######################################## P2
51      # You must declare a function before you use it.
52
53      shout()                  # ERROR: shout does not exist! Computer barfs and program stops executing.
54      shout = -> print("AH!") # Code not executed because program stopped.
55      shout()                  # Code not executed because program stopped.
56
```

```
57      ######################################### P3
58      # A function body can have multiple statements separated by semi-colons.
59
60      applaud = -> print("clap"); print("clap"); print("clap");
61      applaud()    # "clap"
62                   # "clap"
63                   # "clap"
64
65
```

```
66  ######################################## P4
67  # A big function on one line is hard to read.
68  # CoffeeScript allows you to write a multi-statement function by using indentation.
69  # This is much easier to read and we will use this approach moving forward.
70
71  applaud = ->        # The new function starts at -> and continues through the next three lines...
72      print("clap")   # Code INSIDE the function. Saved for later.
73      print("clap")   # Code INSIDE the function. Saved for later.
74      print("clap")   # Code INSIDE the function. Saved for later.
75
76  print("Hello")  # OUTSIDE the applaud function. This code runs immediately.    "Hello"
77  applaud()       # applaud called, "clap"
78                  #                 "clap"
79                  #                 "clap"
80
81
```

```
82  ######################################## P5
83  # You can write functions that accept 'input' or 'arguments' just like print() accepts arguments.
84
85  echo = (input)->  # 'input' is a variable that only exists inside of our echo function. Its value is set when the function is called later.
86                    # Note that the parenthesis are used in a different way here. On line 85 they do not indicate a function call.
87      print(input)  # On this line we use the parenthesis like normal to call the function print.
88      print(input)  # echo's input variable can only be used in its function body. It disappears once the function is complete.
89
90
91
92  echo("Hello!")    # The string "Hello!" will be assigned to echo's input variable when the echo function runs.   "Hello!"
93                    #                                                                                              "Hello!"
94  echo("Goodbye!")  # When the echo function runs this time, input will be set equal to "Goodbye!"                "Goodbye!"
95                    #                                                                                              "Goodbye!"
96  print(input)      # ERROR. Input doesn't exist outside of the echo function.
97                    # You can think of a function as an itty bitty program with its own variables.
98                    # The variables only exist in that function, just like your variables only exist in your program.
99                    # Once the function is done running the function variables disappear, just like with a program.
100                   # These kinds of variables are technically called 'parameters'.
101                   # The -data- that is passed to the parameter is called an 'argument'.
102                   # We can say 'The echo function has one parameter named input' or 'The echo function accepts one argument'
103
104 # NOTE: A function's parameters can be named anything following normal variable naming rules.
105
```

```
106
107     ######################################## P6
108     # Functions can have multiple parameters, i.e. they can accept multiple arguments:
109
110     echo = (sound1,sound2)-> # 'sound1' and 'sound2' are both function variables, e.g. parameters.
111         print(sound1)        # They only exist in the echo function.
112         print(sound2)
113         print(sound1)
114         print(sound2)
115
116     echo("Hello","Sam")     # "Hello"
117                             # "Sam"
118                             # "Hello"
119                             # "Sam"
120
```

```
121     ######################################## P7
122     # Using a function name without parenthesis NORMALLY has no effect.
123     # For exceptions, see 01a-variables.pdf, P14
124
125     shout = -> print("AH!")
126     shout                   # Nothing! We only referred to the function. We didn't call it.
127     shout()                 # "AH!"
128
129
```

```
130 ######################################## P8
131 # There are reasons to refer to a function without calling it.
132 # This allows us to pass a function to another function.
133
134 shout = -> print("AH!")      # Create a function
135 print(shout)                 # <Function () { return print("AH!"); }>
136                              # Notice that we're treating the function as data, and passing it to the shout function.
137                              # Think of line 135 as handing a friend some written instructions that they read out loud but do not perform.
138
```

```
139     ######################################## P9
140     # Treating custom functions as data allows us to make logical connections between inputs and consequences.
141     # Framer layers have special built in functions that allow us to 'bind' an input event to a custom function.
142
143     box = new Layer()
144     shout = -> print("AH!")
145     sigh = -> print("Whew!")
146     box.onMouseDown(shout)    # When we mouse press on box, the function shout is called.
147     box.onMouseUp(sigh)       # When we mouse release on box, the function sigh is called.
148
149     # We'll talk about this more in future sections.
150
151
```

```
######################################## P10
# Instead of naming the function and then referring to it, we can also use a function literal to create behaviors.
# The code below creates the same interactive behavior as the previous code. (Though the program isn't -exactly- wired together the same.)

box = new Layer()
box.onMouseDown(-> print("AH!"))        # When we mouse press on box, the function shout runs.
box.onMouseUp(-> print("Whew!"))        # When we mouse release on box, the function sigh runs.
                                        # These unnamed literal functions are sometimes called an anonymous functions.
```

```
162 ######################################## P11
163 # We can pass objects to functions. i.e. Functions can accept objects as arguments.
164 # This is a common practice when using Framer and other UI frameworks.
165 # We will talk about it more in the animation section.
166
167 printArea = (input)->                    # When called on lines 170,171,and 172, the variable input becomes equal to whatever object was passed in.
168     print(input.width * input.height)    # Here .width and .height refer to the width and height of whatever object was passed in.
169
170 box1 = {width:20,height:10}
171 box2 = {width:10,height:10}
172 box3 = {width:2,height:10}
173
174 printArea(box1) # 200
175 printArea(box2) # 100
176 printArea(box3) # 20
177
178
```

```
179     ######################################### P12 [ Optional ]
180     # We can use CoffeeScript's function shorthand when calling our own functions.
181
182     echo = (input)->    # Note that the parenthesis around the function's parameter is still required.
183         print input
184         print input
185
186     echo "Hello!"       # Exactly the same as echo("Hello!"),   "Hello!"
187                         #                                       "Hello!"
188     echo "Goodbye!"     # Exactly the same as echo("Goodbye!")  "Goodbye!"
189                         #                                       "Goodbye!"
190
```

```
######################################## P13 [ Optional ]
# Shortcuts for objects and function calls can be combined.
# This allows for very quick writing but means that some very different expressions can look very similar...

print                    # Call the function print and pass it a literal object,    {width:1,height:1}
    width:1
    height:1

box =                    # Assign object data to the box variable
    width:1
    height:1

box                      # ERROR: Attempted to call a box function but box is not a function.
    width:1
    height:1

# This approach will come in handy later when coding animations.

########################################
######################################## END
########################################
```