```
###

    The programs in this document introduce:

        • Functions


    Interface programming is mostly about telling the computer to run a custom function when some event happens.
    So we need to know how to create our own functions.

###

```

```
######################################## P1
# In CoffeeScropt, a function or "function literal" looks like this:
#
# -> FUNCTION CODE/BODY
#
# example:

-> print("I ran!")

# This code will have no result right now.
# The "->" means "new function starting here"
# The code that comes after is the "function body"
# All together this is called a "function declaration"
```

```
28  ######################################## P3
29  # The function does not run automatically though.
30  # It is set aside for running LATER.
31  # It is as if you -explained- to the computer how to do something instead of -telling- it to do something.
32
33  print("step 1")
34  -> print("I don't run")
35  print("step 2")
36
37
```

```coffeescript
######################################## P3
# If we want to USE our function, we must store it in a variable so we can refer to it.
# We can us the variable name to call the function.

applaud = -> print("clapping") # Store the function in the variable 'applaud'
applaud() # Call the custom function
applaud() # Call the custom function again
applaud() # ...and again

# In a sense, we have named our function.

```

```
######################################## P4
# NOTE: You MUST declare a function BEFORE you use it.

applaud()                  # ERROR: applaud does not exist! Computer barfs and program may quit.
applaud = -> print("clapping")  # Code not executed because program quit
applaud()                  # Code not executed because program quit
```

```
######################################## P5
# Again, a function declaration DESCRIBES the function, it does not run the function code.
# Pay careful attention to the order that the print commands run.

print(1)
sayThree = -> print(3)          # Tell the computer to remember this code under the name 'applaud'
print(2)
sayThree()                      # Execute the applaud code.
                                # You can think of a function call as meaning "go to and run that code"
print(4)
```

```
69  ######################################## P6
70  # A function can have multiple lines of code, separated by semi-colons.
71
72  applaud = -> print("clapping"); print("clapping"); print("clapping");
73  applaud()
74
75
```

```
76  ##################################### P7
77  # You can write each line by itself using a tab-indent.
78  # This is the more common way of writing functions in CoffeeScript.
79
80  applaud = ->
81      print("clapping")  # Code INSIDE the function body
82      print("clapping")  # Code INSIDE the function body
83      print("clapping")  # Code INSIDE the function body
84  print("Hello") # Code OUTSIDE the function body
85  applaud()  # Code OUTSIDE the function body
86
87
```

```coffeescript
######################################## P8
# You can write functions that accept "arguments" or "input"


echo = (sound)-> # 'sound' is a variable that only exists inside of the function. The value is set when the function is called.
    print(sound) # The sound variable dissapears once the function is complete.
    print(sound) # You can think of a function as an itty bitty program. When it runs it may create some variables.
                 # And once the function is done running it cleans up it's variables, just like a full program does.
                 # To be technical, a variable that is part of a function declaration is called a "parameter".
                 # The -data- that is passed to the parameter during the function call is called the "argument".


echo("Hello!") # The string "Hello!" will be assigned to the parameter 'sound' when the echo function runs.
               # But sound will only equal "Hello" this one time.

echo("Goodbye!") # When the echo funtion runs this time, 'sound' will be set euqal to "Goodbye!"

```

```coffeescript
######################################## P8b
# When we can call our function with an argument, we can use CoffeeScript's shortcut like normal.

echo = (sound)->
    print(sound)
    print(sound)

echo "Hello!"     # Exactly the same as echo("Hello!")
echo "Goodbye!"   # Exactly the same as echo("Goodbye!")

```

```
115 ######################################## P9
116 # Functions can be written to accept multiple arguments:
117
118 echo = (sound1,sound2)-> # 'sound1' and 'sound2' are both parameters.
119     print(sound1)
120     print(sound2)
121     print(sound1)
122     print(sound2)
123
124 echo("Hello","Sam")
125
126
```

```
127 ######################################## P10
128 # NOTE: You can pass more arguments to a function than it can accept
129 # The extra arguments will be ignored
130
131 echo = (sound1,sound2)->
132     print(sound1)
133     print(sound2)
134     print(sound1)
135     print(sound2)
136
137 echo("Hello","Sam","James","Bill") # Only the first two arguments are used by the function. The second two are ignored
138
139
140
```

```
141 ######################################## P11
142 # Functions can accept objects as arguments.
143 # This is a super common activity when using Framer. We will talk about it more in the animation section
144
145 printArea = (obj)->
146     print( obj.width * obj.height )
147
148
149 printArea({width:20,height:10})     # "0,10,100"  When printArea function runs
150                                                   the 'obj' parameter in the function
151                                                   will refer to the object literal we create here.
152
```

```coffeescript
######################################## P11b
# CoffeeScript shortcuts for objects and functions can be combined.
# This leads to very easy to write code, but the code may be ambiguous unless you
# understand the shortcuts.

printArea = (obj)->
    print( obj.width * obj.height )


printArea({width:20,height:10})    # "0,10,100"

printArea {width:20,height:10}  # Exactly the same as above

printArea                       # Exactly the same as above
    width:1
    height:1


# I'll avoid the shortcuts for a little longer.
```

```
########################################
######################################## Extra
########################################

######################################## P12
# Some functions, when called, turn into, or "return", data.
# For example, the built in function Number accepts primitive data, and returns a number.


print(Number("2"))  # Number("2") returns the number data '2'

# Function calls that return data can be used anywhere you use data, even as arguments.

```

```
187 ######################################## P13
188 # When using a function call as an argument, the calls are resolved from the inside out, like math.
189
190 print(Number("2"+"2"))  # The math is executed first, the result "22" is passed to the Number function,
191                         # and the result of that is passed to the print function.
192
```

```
193
194 ######################################## P14
195 # In most languages, you can return data from a custom function by using the "return" keyword.
196
197 bake = (ingredient)->
198     return ingredient + " pie"
199
200 pie = bake("apple") # Try making different kinds of pies!
201 bake(pie)
202 print(pie)   # “apple pie”
203
```

```
204 ######################################## P15
205 # In CoffeeScript, by default, a function returns whatever data is on the last line of its body.
206 # This can save us some typing:
207
208 bake = (ingredient)->
209     ingredient + " pie" # The math is executed and the results are automatically returned
210
211 print(bake("apple")) # I'm using the result of the bake function as the input the print function
212
213
214
215 ########################################
216 ######################################## End
217 ########################################
```