


```
##### P1
# In CoffeeScript, 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
```

```
38 ##### P3
39 # If we want to USE our function, we must store it in a variable so we can refer to it.
40 # We can use the variable name to call the function.
41
42 applaud = -> print("clapping") # Store the function in the variable 'applaud'
43 applaud() # Call the custom function
44 applaud() # Call the custom function again
45 applaud() # ...and again
46
47 # In a sense, we have named our function.
48
```

```
49 ##### P4
50 # NOTE: You MUST declare a function BEFORE you use it.
51
52 applaud() ~ ~ ~ ~ ~ # ERROR: applaud does not exist! Computer barfs and program may quit.
53 applaud = -> print("clapping") ~ # Code not executed because program quit
54 applaud() ~ ~ ~ ~ ~ # Code not executed because program quit
55
56
```

```
57 ##### P5
58 # Again, a function declaration DESCRIBES the function, it does not run the function code.
59 # Pay careful attention to the order that the print commands run.
60
61 print(1)
62 sayThree = -> print(3) - - - # Tell the computer to remember this code under the name 'applaud'
63 print(2) - - - - -
64 sayThree() - - - - - # Execute the applaud code.
65 - - - - - # You can think of a function call as meaning "go to and run that code"
66 print(4)
67
68
```

```
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
```

```
##### P7
# You can write each line by itself using a tab-indent.
# This is the more common way of writing functions in CoffeeScript.
```

```
applaud = ->
  print("clapping") # Code INSIDE the function body
  print("clapping") # Code INSIDE the function body
  print("clapping") # Code INSIDE the function body
print("Hello") # Code OUTSIDE the function body
applaud() # Code OUTSIDE the function body
```



```

88 ##### P8
89 # You can write functions that accept "arguments" or "input"
90
91
92 echo = (sound)-> # 'sound' is a variable that only exists inside of the function. The value is set when the function is called.
93   print(sound) # The sound variable dissapears once the function is complete.
94   print(sound) # You can think of a function as an itty bitty program. When it runs it may create some variables.
95   # And once the function is done running it cleans up it's variables, just like a full program does.
96   # To be technical, a variable that is part of a function declaration is called a "parameter".
97   # The -data- that is passed to the parameter during the function call is called the "argument".
98
99
100 echo("Hello!") # The string "Hello!" will be assigned to the parameter 'sound' when the echo function runs.
101   # But sound will only equal "Hello" this one time.
102
103 echo("Goodbye!") # When the echo funtion runs this time, 'sound' will be set euqal to "Goodbye!"
104

```

```
105 ##### P8b
106 # When we can call our function with an argument, we can use CoffeeScript's shortcut like normal.
107
108 echo = (sound) ->
109   print(sound)
110   print(sound)
111
112 echo "Hello!"      # Exactly the same as echo("Hello!")
113 echo "Goodbye!"    # Exactly the same as echo("Goodbye!")
114
```

```
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"
150
151
152
```

When printArea function runs
the 'obj' parameter in the function
will refer to the object literal we create here.

```
153 ##### P11b
154 # CoffeeScript shortcuts for objects and functions can be combined.
155 # This leads to very easy to write code, but the code may be ambiguous unless you
156 # understand the shortcuts.
157
158 printArea = (obj) ->
159   ~ print( obj.width * obj.height )
160   ~
161   ~
162 printArea({width:20,height:10}) ~ # "0,10,100"
163
164 printArea {width:20,height:10} ~# Exactly the same as above
165
166 printArea ~ ~ ~ ~ ~ # Exactly the same as above
167   ~ width:1
168   ~ height:1
169
170
171 # I'll avoid the shortcuts for a little longer.
172
173
```

```
174 #####
175 ##### ·Extra
176 #####
177
178 ##### ·P12
179 # Some functions, when called, turn into, or "return", data.
180 # For example, the built in function Number accepts primitive data, and returns a number.
181
182
183 print(Number("2")) · # Number("2") returns the number data '2'
184
185 # Function calls that return data can be used anywhere you use data, even as arguments.
186
```

```
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.
```



```
##### P14
```

```
# In most languages, you can return data from a custom function by using the "return" keyword.
```

```
bake = (ingredient)-> {  
  return ingredient + " pie"
```

```
pie = bake("apple") # Try making different kinds of pies!  
bake(pie)
```

```
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 #####
```