





```
31 ##### P2
32 # If you try to use a function that doesn't exist, you'll get an error.
33
34 explode()
35
36
```

```
37 ##### P3
38 # Some functions accept additional details.
39 # We call these details "arguments".
40
41 print(100)
42 print(100,200,300)
43
44 # When we call functions with arguments we sometimes say that we're: "passing" whatever-data to whatever-function
45 # Above, we're passing 100 to the print function. Next we 'pass three numbers to print'.
46
47
```

```
48 ##### P4
49 # In CoffeeScript, if we "call" a function "with arguments", we can leave out the parenthesis.
50 # This is common shortcut in CoffeeScript. It looks like this:
51
52 print 100
53 print 100,200,300
54
55 # The spaces between the function name and the argument are meaningful symbols to CoffeeScript, just like '=' or '(' or '"', "
56 # This is called "significant whitespace".
57 # Python is another common language that uses significant whitespace.
58 # Most languages are NOT like this though (C, C++, C#, Java, Javascript, Swift, Objective-C)
59 # Significant whitespace is a polarizing paradigm among developers.
60
```

```
61 ##### P5
62 # If we ever call a function WITHOUT arguments, we MUST use '()'
63
64 print() ~ # Function call. Ok.
65 print ~ # Nothing at all! No good.
66 print "ok" ~ # Function call with arguments. Ok.
67
68 # We will use parenthesis for this section.
69
```



```
70 ##### · P6
71 # Computers do math.
72 # +, -, /, * are called "operators" or "operations".
73 # A math expression is called an "expression".
74 # Expressions are always solved BEFORE function calls.
75 # For example, 'print 10+10' becomes 'print 20' and then the computer runs that code.
```

```
76
77 print(10+10)
78 print(10-10)
79 print(10*10)
80 print(10/10)
```

```
81
82
83
```

```
84 ##### P7
85 # Data comes in different types, or "data types".
86 # Some data types are "primitive" and are built into the language.
87 # CoffeeScript/Javascript has a few primitive data types:
88
89 # A "String" of characters.
90 print("1,000.0")
91
92 # A "Number"
93 print(1000.0)
94
95 # A "Boolean"
96 print(true)
97
98 # Tech Note: CoffeeScript and JavaScript do not distinguish between integers and floating point numbers like some other languages.
99
100
```



```
101 ##### P8
102 # In JavaScript and CoffeeScript, datatypes affect how operations work.
103 # Technically, the computer will try to convert the data to something that works.
```

```
104
105 print(2+2)
106 print("2"+"2")
107 print("2"+2)
108 print("2"*2)
```

```
109
110 # (Some languages play it safe and refuse to convert data unless told to.)
111
112
```

```
113 ##### P9
114 # Data can be stored in memory.
115 # There are two basic places: long term (hard drive) and short term (RAM).
116 # Data in long term storage will remain even if the program stops. It is a topic for another day.
117 # Short term storage disappears after the program stops.
118 # Short term storage is both easy and essential. It works almost the same way in almost every language.
119 # In CoffeeScript, this is how it's done:
120
121 myData = "SomeData"
122 amIReallyAProgram = true
123 VALUE_FOR_PI = 3.141598
124 _A2d3f_ = "Bad Name"
125 herp_derp = 1
126
127 # The word on the left is called a "variable", the equal sign is called the "assignment operator".
128 # Variables may be named whatever you like! So long as the name contains no spaces or math operators and doesn't start with a number.
129 # The naming convention for variables in most languages is "camelCase". First word lowercase, next words uppercase.
130 # Note: For those who know JavaScript, 'var' is not allowed in CoffeeScript.
```

132

133

```
##### P10
```

134

```
# Variables can be used wherever you would use data
```

135

136

```
a = 1
```

137

```
print(a)
```

138

139

```
##### P11
```

```
# When variables are used with a math operator, the variable names are replaced with their data before the math is resolved.
```

```
a = 2
```

```
print(a+a)    # This becomes 'print 2+2' which becomes 'print 4'
```

```
print((a+a)*a) # This becomes 'print (2+2)*2' which becomes 'print 8'
```

```
148 ##### P12
149 # The same applies during assignment:
150
151 a = 2
152 b = a+a+a      # This becomes 'b = 2+2+2' which becomes 'b = 6'
153 print(b + b)   # 'print 6+6' becomes 'print 12'
154
```

```
##### P13
```

```
# When primitive data is assigned to a variable, the variable's old data is overwritten.
```

```
a = 1
```

```
a = 2      # The 1 in memory is overwritten with the number 2.
```

```
a = a + a  # becomes 'a = 2 + 2', then becomes 'a = 4'. Finally, the 2 that was in memory is overwritten with the number 4
```

```
print(a)
```



```
164 ##### P14
165 # Look very carefully. This part is tricky.
166
167 a = 1
168 b = a    # This line becomes 'b = 1'
169         # There are now two places in memory (labeled 'a' and 'b').
170         # 'a' and 'b' each has it's own unique number '1'.
171         # The data for variable 'a' was duplicated. The copy was placed into variable 'b'.
172         # The variable 'b' does NOT point to the same memory that 'a' points to.
173 a = 2    # Changing the number in 'a' will not affect the data in 'b' or vice versa.
174
175 print(a)
176 print(b)
177
178 # Note: This behavior holds for almost any programming language an interaction designer will encounter.
179 # Note: There is a huge exception to this rule that we will discuss in the section on Objects.
180
181
```

