```
###
          What follows are tiny CoffeeScript programs.
          The programs in this document introduce:
6 • Basic commands.
8 Primitive datatypes.
9 · Assignment and Variables
To use this document:
13 " 1. 'Go'to'<u>http://prototyp.in</u>-OR'download framer from <u>https://framer.com</u>
    2. Type out, by hand, each program, by itself.
15 ' ' ' (You can ignore any lines with instructions or commentary.)
      3. Make notes where anything is the least bit confusing. Experiment with specific examples to explore how it works.
```

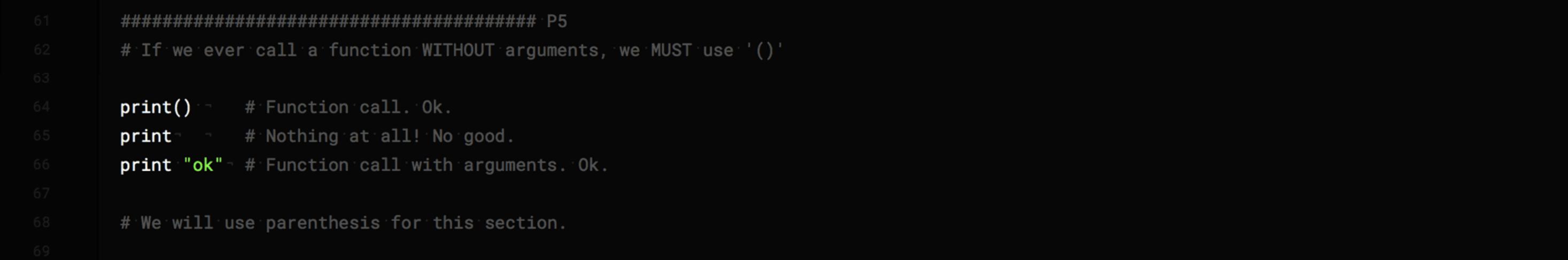
###

```
# Some words are like magic spells.
# We cast the spell by using a keyword, followed by parenthesis and a semi-colon.
# In programming we call spells "functions".
# Casting them is called "calling" them.
# Below, we "call" the "function" print.
print() **# A panel should pop up on the right side.
```

| ###################################### |
|---|
| #'If'you'try'to'use'a'function'that'doesn't'exist,'you'll'get'an'error. |
| |
| explode() |
| |
| |

```
# Some functions accept additional details.
# We call these details "arguments".
print(100)
print(100,200,300)
# When we call functions with arguments we sometimes say that we're: "passing" whatever-data to whatever-function
# Above, we're passing 100 to the print function. Next we 'pass three numbers to print'.
```

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

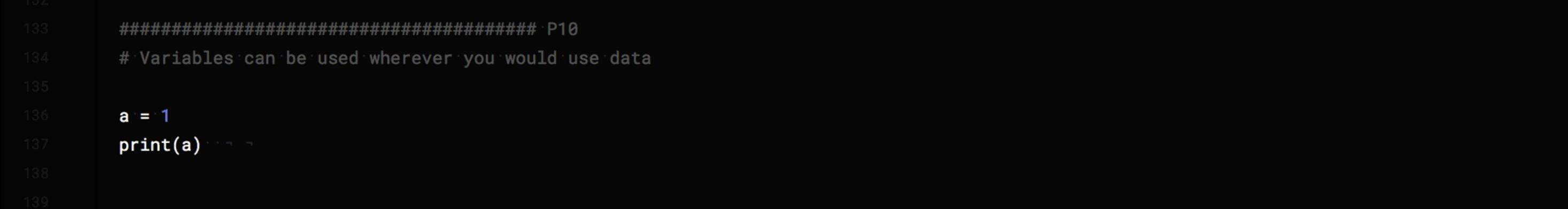


```
# Computers do math.
# +, -, /, * are called "operators" or "operations".
# A math expression is called an "expression".
# Expressions are always solved BEFORE function calls.
# For example, 'print 10+10' becomes 'print 20' and then the computer runs that code.
print(10+10)
print(10-10)
print(10*10)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

```
# Data comes in different types, or "data types".
# Some data types are "primitive" and are built into the language.
# CoffeeScript/Javascript has a few primitive data types:
# A "String" of characters.
print("1,000.0")
# A · "Number"
print(1000.0)
# A Boolean"
print(true)
# Tech Note: CoffeeScript and JavaScript do not distinguish between integers and floating point numbers like some other languages.
```

```
# In JavaScript and CoffeeScript, datatypes affect how operations work.
# Technically, the computer will try to convert the data to something that works.
print(2+2)
print("2"+"2")
print("2"+2)
print("2"*2)
# (Some languages play it safe and refuse to convert data unless told to.)
```

```
################ P9
# Data can be stored in memory.
# There are two basic places: long term (hard drive) and short term (RAM).
# Data in long term storage will remain even if the program stops. It is a topic for another day.
# Short term storage disappears after the program stops.
# Short term storage is both easy and essential. It works almost the same way in almost every language.
# In CoffeeScript, this is how it's done:
myData = "SomeData"
amIReallyAProgram = true
VALUE\_FOR\_PI = 3.141598
_A2d3f_ ' = ' "Bad ' Name"
herp_derp = 1
# The word on the left is called a "variable", the equal sign is called the "assignment operator".
# 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.
# The naming convention for variables in most languages is "camelCase". First word lowercase, next words uppercase.
# Note: For those who know JavaScript, 'var' is not allowed in CoffeeScript.
```



```
# 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'
```

```
# The same applies during assignment:
a = 2
         # This becomes be = 2+2+2' which becomes be = 6'
b = a+a+a
print(b + b) - # 'print 6+6' becomes 'print 12'
```

```
# 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)
```

```
# Look very carefully. This part is tricky.
a = 1
        "#"This 'line 'becomes 'b' = '1'
b = a
        *# There are now two places in memory (labeled 'a' and 'b').
        "#"'a' and 'b' each has it's own unique number '1'."
        "# The data for variable 'a' was duplicated. The copy was placed into variable 'b'.
        "# The variable 'b' does NOT point to the same memory that 'a' points to.
        # Changing the number in 'a' will not affect the data in 'b' or vice versa.
a = 2
print(a)
print(b)
# Note: This behavior holds for almost any programming language an interaction designer will encounter.
# Note: There is a huge exception to this rule that we will discuss in the section on Objects.
```

```
# CoffeeScript supports common assignment shortcuts or "mutating operators"
a = 1
a+= 2 · · -# · add · to · self, · e.g. · a · = · a · + · 2
     " # subtract from self: a = a - 2
      # increment by one, e.g. a = a + 1
a++
      # decrement by one, e.g. a = a - 1
a--
      # divide self by value, e.g. a = a/2
a*= 2 · · · # · multiply · self · by · value, · e.g. · a · = · a · * · 2
```

| ###################################### |
|---|
| ###################################### |
| ###################################### |
| # |
| # The print function is part of the FramerJS library. |
| # Normally, you'll want to use console.log(). |
| # Everything else in this document works anywhere you use CoffeeScript. |
| # |
| #·End! |