Python: Getting Functional

Do a little analysis of the code below. How does it look? Choose as many of the options below as you like, or write in your own analysis:

```
dog_name = "Codie"
dog_weight = 40
if dog_weight > 20:
        print(dog_name, 'says WOOF WOOF')
else:
        print(dog_name, 'says woof woof’)
```

```
dog_name = "Jackson"
dog_weight = 12
if dog_weight > 20:
        print(dog_name, 'says WOOF WOOF’)
else:
        print(dog_name, 'says woof woof’)
```

```
dog_name = "Sparky"
dog_weight = 9
if dog_weight > 20:
        print(dog_name, 'says WOOF WOOF’)
else:
        print(dog_name, 'says woof woof’)
```

```
dog_name = "Fido"
dog_weight = 65
if dog_weight > 20:
print(dog_name, 'says WOOF WOOF’)
else:
print(dog_name, 'says woof woof')
```

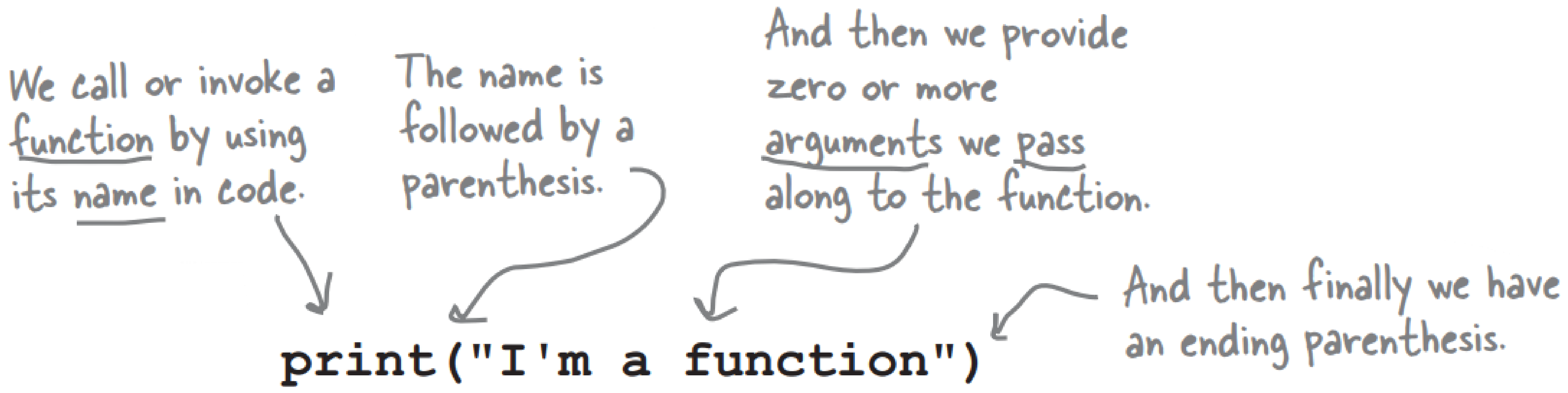
What’s wrong with the code, anyway?

BRAIN POWER

How can you improve this code? Take a few minutes to think of a few possibilities.

Turning a block of code into a <u>FUNCTION</u>

You've already *used* a few functions, like, `print`, `str`, `int`, and `range`.



We call or invoke a function by using its name in code.

The name is followed by a parenthesis.

And then we provide zero or more arguments we pass along to the function.

And then finally we have an ending parenthesis.

```
print("I'm a function")
```

To create a function, start with the def keyword, and follow it with a name to remember the function by.

Next you can have zero or more parameters. Think of these like variables that hold values that you will pass into the function when you call it.

And then we have a colon, which, as you know, starts a block of code in Python.

```
def bark(name, weight):
    if weight > 20:
        print(name, 'says WOOF WOOF')
    else:
        print(name, 'says woof woof')
```

Here's the block of code we are going to reuse.

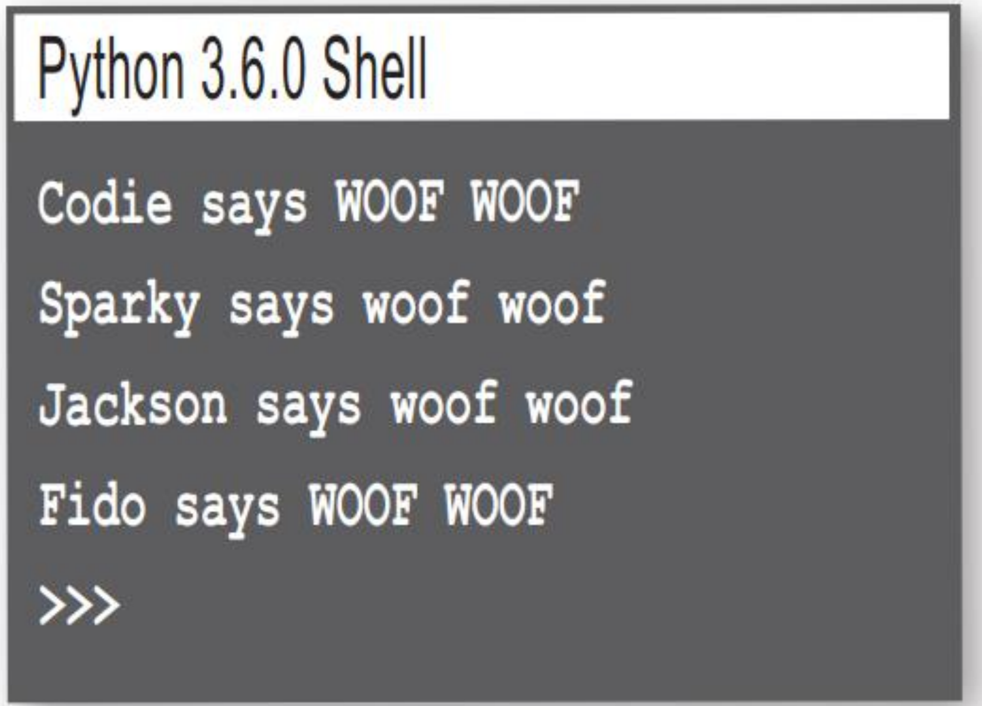
In Python and pretty much all languages, we call this code block the body of the function.

We created a function, so how do we use it?

```
bark('Codie', 40)
bark('Sparky', 9)
bark('Jackson', 12)
bark('Fido', 65)
```

Let's test bark with all the dogs we know about.

Great, that's the output we were expecting!


Python 3.6.0 Shell

```
Codie says WOOF WOOF
Sparky says woof woof
Jackson says woof woof
Fido says WOOF WOOF
>>>
```

Here we start with a print statement, just to get things going.

```
print('Get those dogs ready')

def bark(name, weight):
    if weight > 20:
        print(name, 'says WOOF WOOF')
    else:
        print(name, 'says woof woof')

bark('Codie', 40)

print("Okay, we're all done")
```

And then we have our bark function definition.

And then we're going to call the bark function with the arguments Codie and 40.

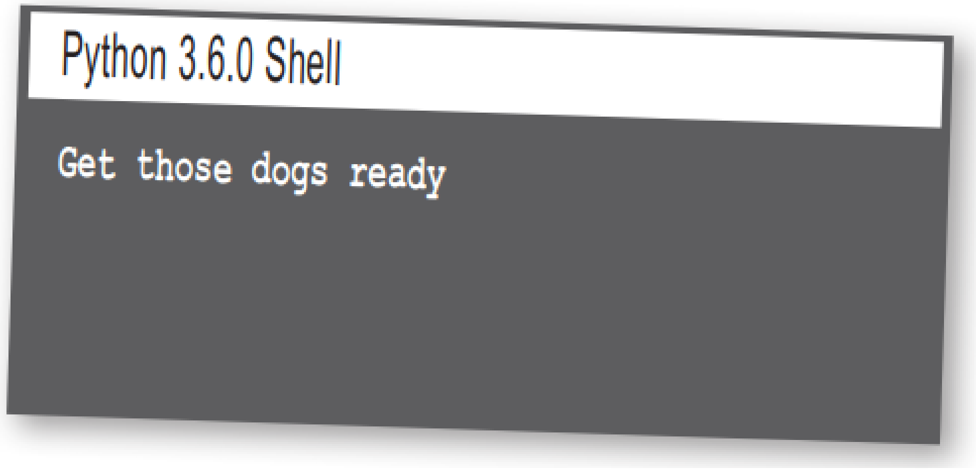
And finally we use a print statement to say we're done.

BRAIN POWER

Pretend you are the Python interpreter. Start at the top of the code and mentally step through and execute each line. Are there any steps that are unclear to you?

We start here.

```
print('Get those dogs ready')

def bark(name, weight):
    if weight > 20:
        print(name, 'says WOOF WOOF')
    else:
        print(name, 'says woof woof')

bark('Codie', 40)

print("Okay, we're all done")
```


Python 3.6.0 Shell

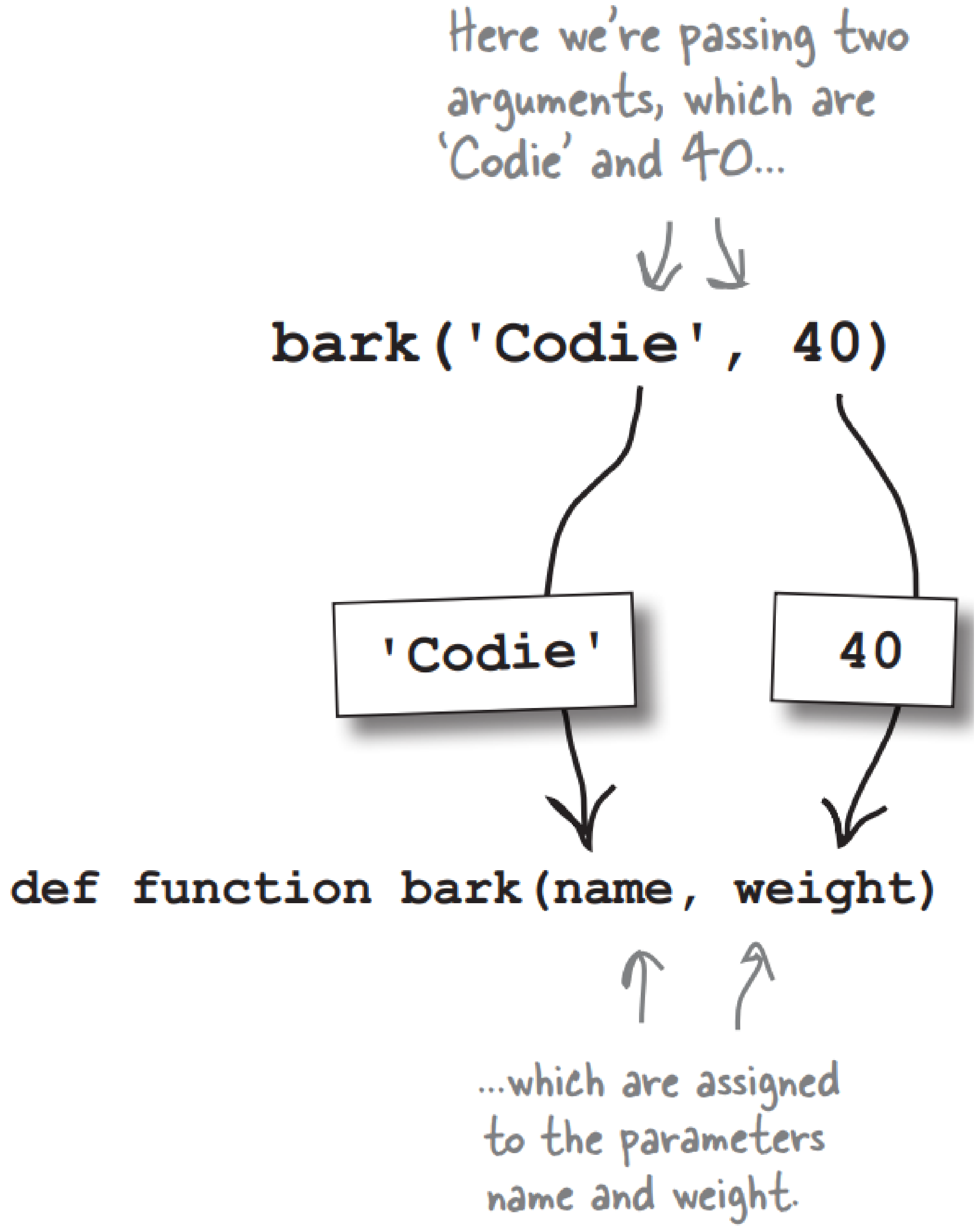
```
Get those dogs ready
```

```python
print('Get those dogs ready')

def bark(name, weight):
    if weight > 20:
        print(name, 'says WOOF WOOF')
    else:
        print(name, 'says woof woof')

bark('Codie', 40)

print("Okay, we're all done")
```

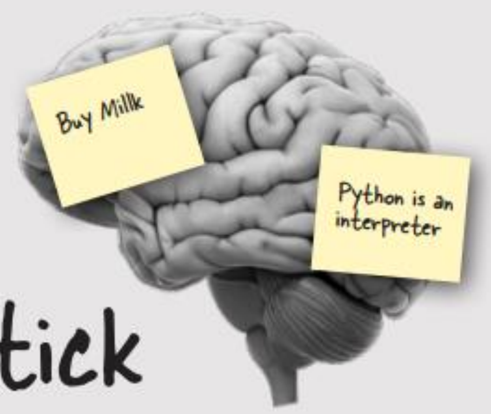
Next we have a call to the bark function.

It's time to invoke our function.

```
print('Get those dogs ready')

def bark(name, weight):
    if weight > 20:
        print(name, 'says WOOF WOOF')
    else:
        print(name, 'says woof woof')

bark('Codie', 40)

print("Okay, we're all done")
```


Here we're passing two arguments, which are 'Codie' and 40...
'Codie'
40
...which are assigned to the parameters name and weight.

```
bark('Codie', 40)

def function bark(name, weight)
```



Make it stick

We call, or **invoke**, a function.

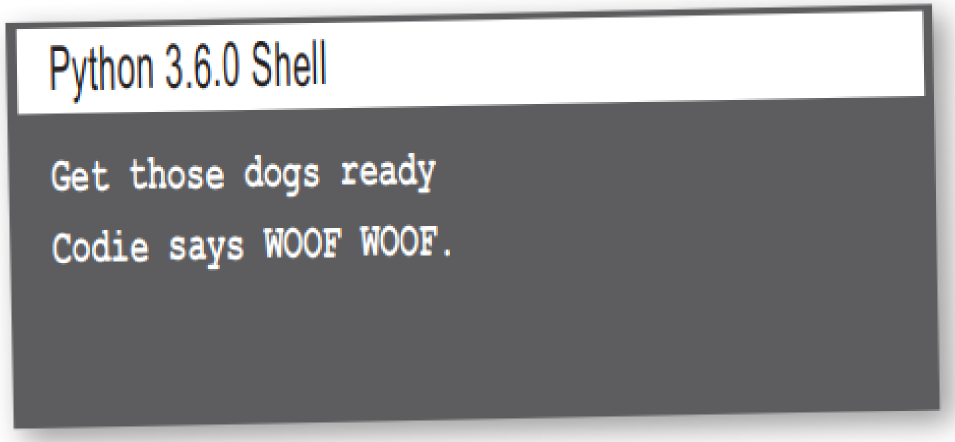
You pass **arguments** into your function calls.

A function has zero or more **parameters** that accept values from your function call.

Now we execute the function body.

After the values of your arguments have been assigned to each parameter, then it's time to start executing the body of your function.

```
print('Get those dogs ready')

def bark(name, weight):
    if weight > 20:
        print(name, 'says WOOF WOOF')
    else:
        print(name, 'says woof woof')

bark('Codie', 40)

print("Okay, we're all done")
```


Python 3.6.0 Shell

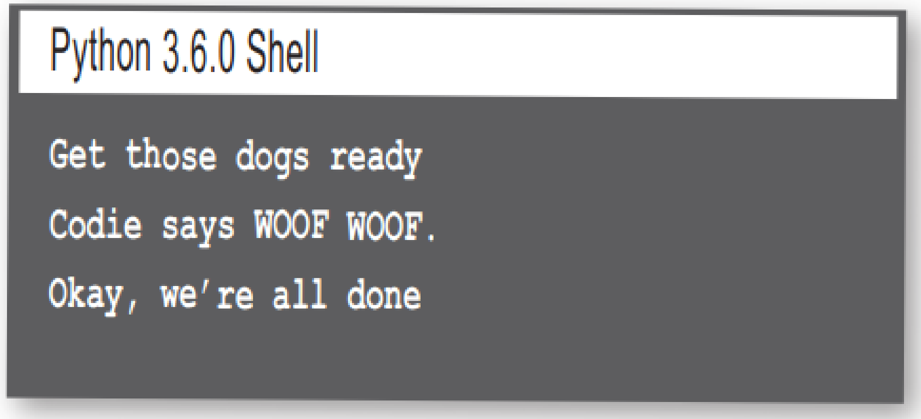
```
Get those dogs ready
Codie says WOOF WOOF.
```

```
print('Get those dogs ready')

def bark(name, weight):
    if weight > 20:
        print(name, 'says WOOF WOOF')
    else:
        print(name, 'says woof woof')

bark('Codie', 40)

print("Okay, we're all done")
```

Remember, when a function completes, the control of the program returns back to where the function was called, and the interpreter resumes execution there.

With the call to bark finished, the interpreter resumes execution right after the call to the bark function. So, we'll pick back up here.

```python
print('Get those dogs ready')

def bark(name, weight):
    if weight > 20:
        print(name, 'says WOOF WOOF')
    else:
        print(name, 'says woof woof')

bark('Codie', 40)

print("Okay, we're all done")
```


Python 3.6.0 Shell

```
Get those dogs ready
Codie says WOOF WOOF.
Okay, we're all done
```

Finally we reach the last line of our program.

A Test Drive

```
def bark(name, weight):
    if weight > 20:
        print(name, 'says WOOF WOOF')
    else:
        print(name, 'says woof woof')

bark('Codie', 40)
bark('Sparky', 9)
bark('Jackson', 12)
bark('Fido', 65)
```

Sharpen your pencil

```
bark('Speedy', 20)

bark('Barnaby', -1)

bark('Scottie', 0, 0)

bark('Lady', "20")

bark('Spot', 10)

bark('Rover', 21)
```

Q: **Do I need to define my function before the code that calls it?**

Or can I put my functions at the end of the file?

A: Yes, you need to define functions before they are *called* in your code. One thing to consider: say we have two functions, f1 and f2, where f1 calls f2 in its body. In this case, it is perfectly fine to define f2 after f1 in your code, so long as f1 does not get called before f2 is defined. That's because defining the function body of f1 does not invoke f2, until f1 is actually called. In terms of where to put functions, we suggest defining functions at the top of your file for better organization and clarity.

Q: **What happens if I mix up the order of my arguments, so that I’m passing the wrong arguments into the parameters?**

A: All bets are off; in fact, we’d guess you’re pretty much guaranteed either an error at runtime or incorrectly behaving code. Always take a careful look at a function’s parameters, so you know what arguments the function expects to be passed and in what order.

Q: **What are the rules for function names?**

A: The rules for naming a function are the same as the rules for naming a variable. Just start with an underscore or letter, and continue with letters, underscores, or numbers. Most Python programmers, by convention, keep their function names all lowercase with undescores between words, like **get_name** or **fire_cannon**

Q: **Can functions call other functions?**

A: Yes, happens all the time. Note you’re already doing this when you call the **print** function within the **bark** function code. Your own functions are no different; you can call them from your other functions.

Functions can **RETURN** things too

Here's a new function, get_bark, that returns the appropriate bark, given a dog's weight.

```
def get_bark(weight):
    if weight > 20:
        return 'WOOF WOOF'
    else:
        return 'woof woof'
```

If the weight is greater than 20 we return the string 'WOOF WOOF'.

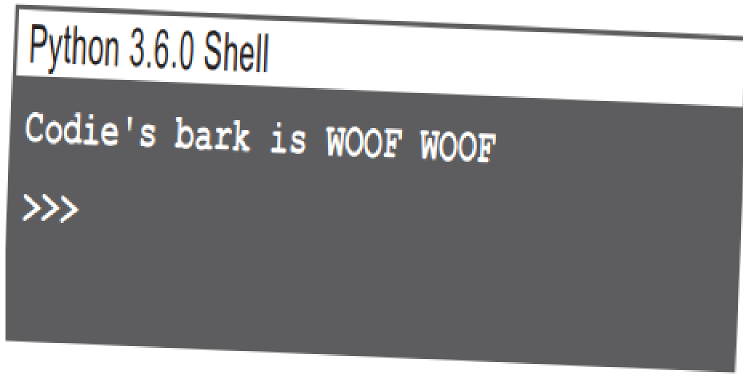
Otherwise, we return the string 'woof woof'

A function can have zero, one, or more return statements.

How to call a function that has a return value

You call the function like any other function, only this function returns a value, so let's set that value to a variable, codies_bark.

```
codies_bark = get_bark(40)
print("Codie's bark is", codies_bark)
```



```
Python 3.6.0 Shell
Codie's bark is WOOF WOOF
>>>
```

Sharpen your pencil

```
def make_greeting(name):
    return 'Hi ' + name + '!'
```

```
def compute(x, y):
    total = x + y
    if (total > 10):
        total = 10
    return total
```

```
def allow_access(person):
    if person == 'Dr Evil':
        answer = True
    else:
        answer = False

    return answer
```

```
make_greeting('Speedy')   ____________________

compute(2, 3)   ____________________

compute(11, 3)   ____________________

allow_access('Codie')   ____________________

allow_access('Dr Evil')   ____________________
```

I noticed in that last exercise that you declared some new variables right inside your functions, like total and answer.

You can declare new variables right inside of your function,

We call these *local variables*, because they are local to the function and only exist as long as the function execution does.

That's in comparison to the global variables we've been using so far, which exist as long as your entire program does.

Getting a little refactoring under our belts

```
hair = input("What color hair [brown]? ")
if hair == ‘’:
        hair = 'brown’
print('You chose', hair)

hair_length = input("What hair length [short]? ")
if hair_length == ‘’:
        hair_length = 'short’
print('You chose', hair_length)

eyes = input("What eye color [blue]? ")
if eyes == ‘’:
        eyes = 'blue’
print('You chose', eyes)

gender = input("What gender [female]? ")
if gender == ‘’:
        gender = 'female’
print('You chose', gender)

has_glasses = input("Has glasses [no]? ")
if has_glasses == ‘’:
        has_glasses = 'no’
print('You chose', has_glasses)

has_beard = input("Has beard [no]? ")
if has_beard == ‘’:
        has_beard = 'no’
print('You chose', has_beard)
```

Obviously our avatar code is in need of some abstraction. Use this space to work out how you think this code should be abstracted into a function (or functions) and what your function calls might look like.

For each attribute we're prompting the user and getting their response.

Each time we prompt we ask a different question.

And we have a different default value, like brown, short, blue.

```
hair = input("What color hair [brown]? ")
if hair == '':
    hair = 'brown'
print('You chose', hair)
```

```
hair_length = input("What hair length [short]? ")
if hair_length == '':
    hair_length = 'short'
print('You chose', hair_length)

eyes = input("What eye color [blue]? ")
if eyes == '':
    eyes = 'blue'
print('You chose', eyes)

    .
    .
    .
```

For every attribute we do exactly the same thing.

We're just showing the code for three attributes to save a little paper.

And we print the attribute back to the user.

Each attribute is also assigned to a variable, like hair or eyes.

Let's call our function get_attribute.

```
def get_attribute(question, default):
```

And we'll use two parameters, question for the question to ask, like "What color hair", and default, to supply the default value, like "brown".

```
def get_attribute(query, default):
    question = query + ' [' + default + ']? '
    answer = input(question)
```

Then we'll prompt the user and get their input. We'll assign the answer variable to their input.

```python
def get_attribute(query, default):
    question = query + ' [' + default + ']? '
    answer = input(question)
    if (answer == ''):
        answer = default
    print('You chose', answer)
```

```
def get_attribute(query, default):
    question = query + ' [' + default + ']? '
    answer = input(question)
    if (answer == ''):
        answer = default
    print('You chose', answer)
    return answer
```

```
def get_attribute(query, default):
    question = query + ' [' + default + ']? '
    answer = input(question)
    if (answer == ''):
        answer = default
    print('You chose', answer)
    return answer
```

```
hair = get_attribute('What hair color', 'brown')
hair_length = get_attribute('What hair length', 'short')
eye = get_attribute('What eye color', 'blue')
gender = get_attribute('What gender', 'female')
glasses = get_attribute('Has glasses', 'no')
beard = get_attribute('Has beard', 'no')
```

BRAIN POWER

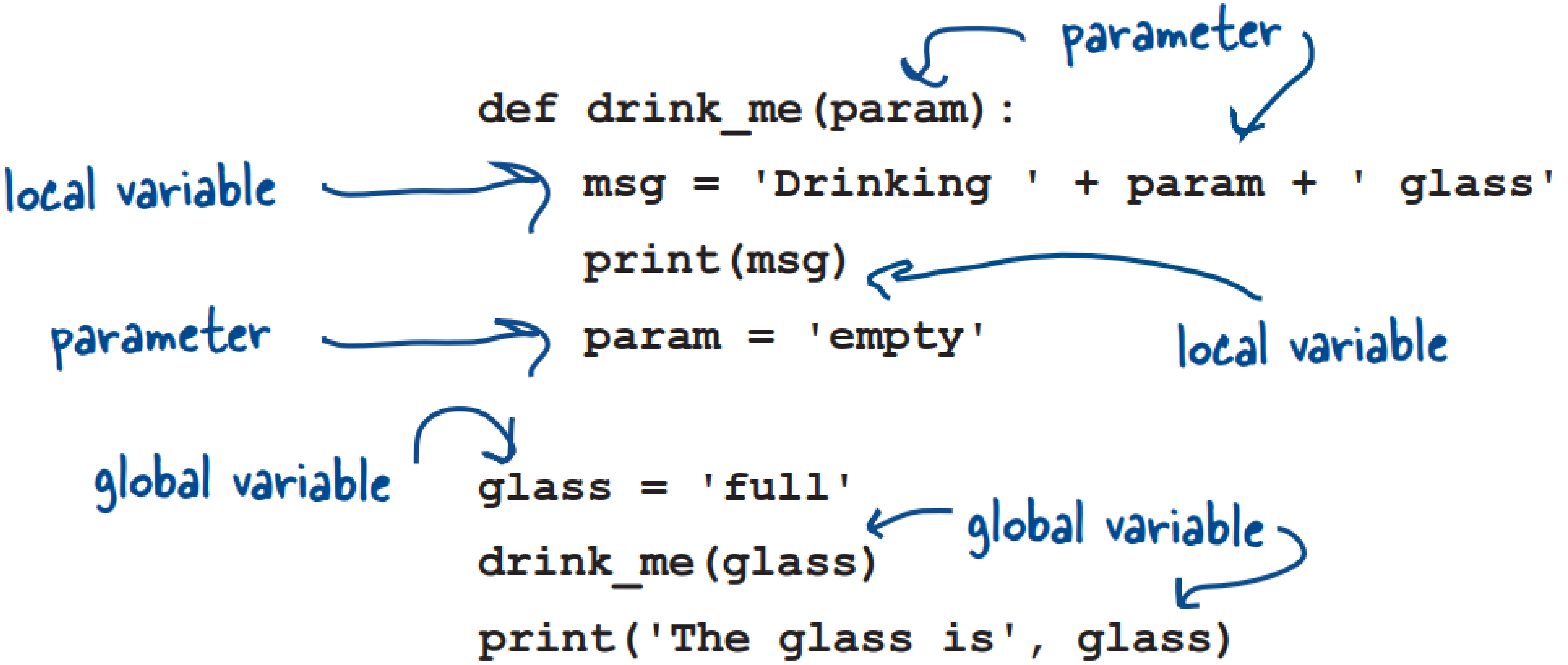
What does this code output? Are you sure? Why did you get that result? Was it what you expected?

```
def drink_me(param):
     msg = 'Drinking ' + param + ' glass’
     print(msg)
     param = 'empty’


glass = 'full’
drink_me(glass)
print('The glass is', glass)
```

Let’s talk about variables a little more…

Make sure we can recognize each type of variable. Annotate this code by identifying local and global variables as well as any parameters.

```
def drink_me(param):
     msg = 'Drinking ' + param + ' glass’
     print(msg)
     param = 'empty’

glass = 'full’
drink_me(glass)
print('The glass is', glass)
```

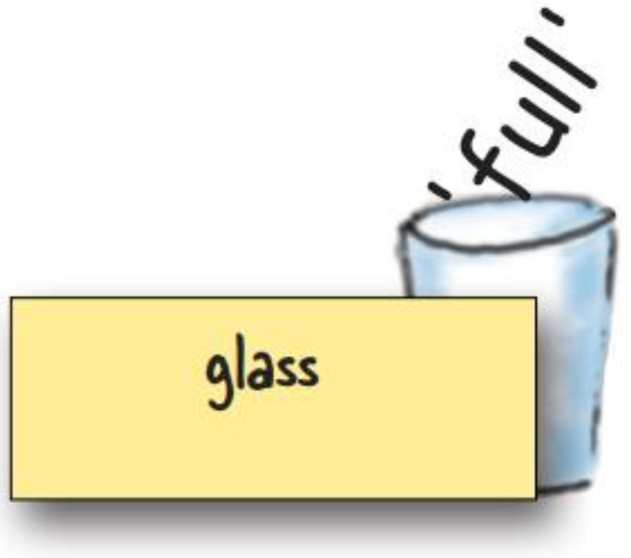


```
def drink_me(param):
    msg = 'Drinking ' + param + ' glass'
    print(msg)
    param = 'empty'

glass = 'full'
drink_me(glass)
print('The glass is', glass)
```

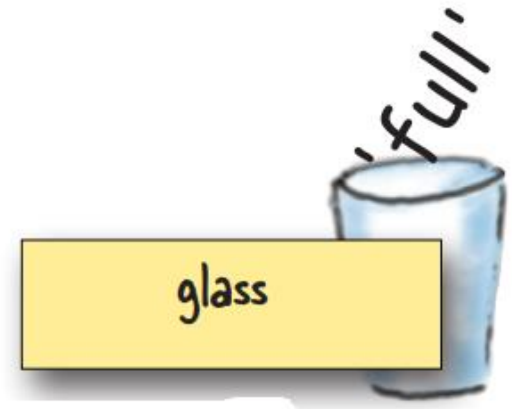
Here's our drink_me code again. Let's step through it and see how the glass remains full after drink_me is called.

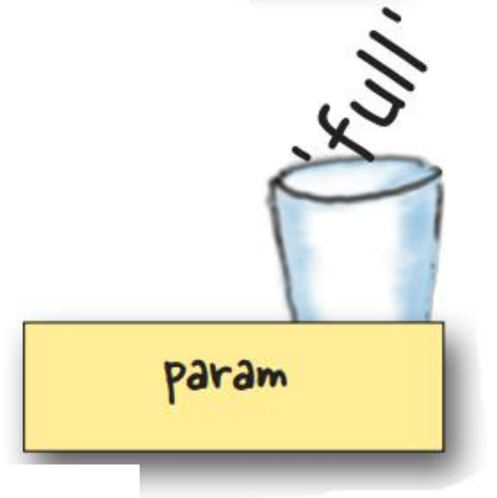
```
def drink_me(param):
    msg = 'Drinking ' + param + ' glass'
    print(msg)
    param = 'empty'

glass = 'full'
drink_me(glass)
print('The glass is', glass)
```

In this code, after the function is defined, we assign the string value 'full' to the variable glass.

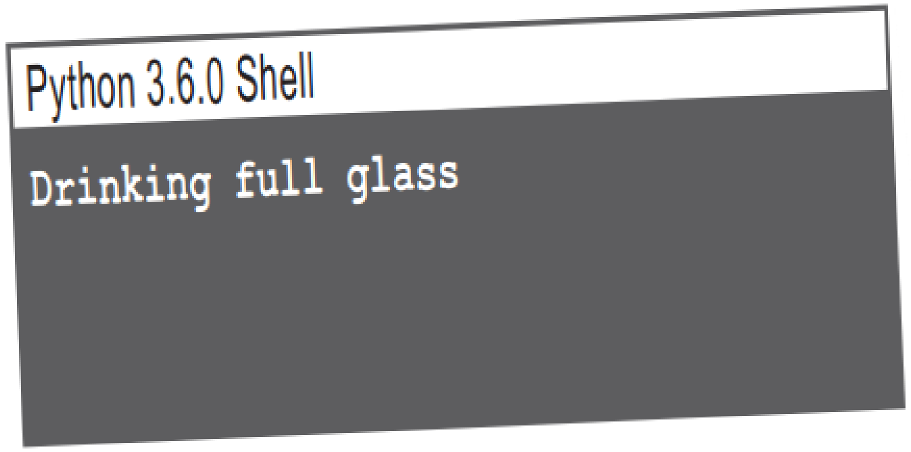


Making the drink me function call





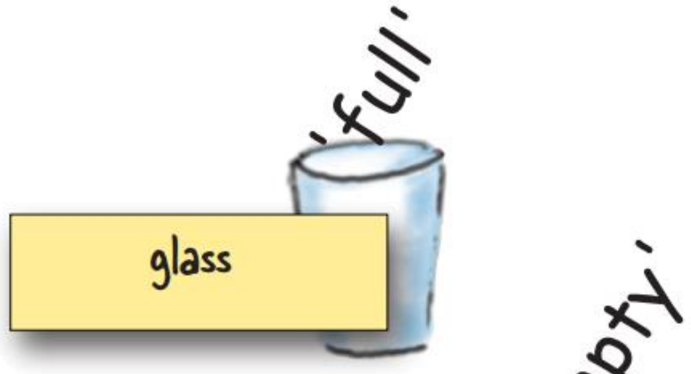
```
def drink_me(param):
     msg = 'Drinking ' + param + ' glass'
     print(msg)
     param = 'empty'

glass = 'full'
drink_me(glass)
print('The glass is', glass)
```

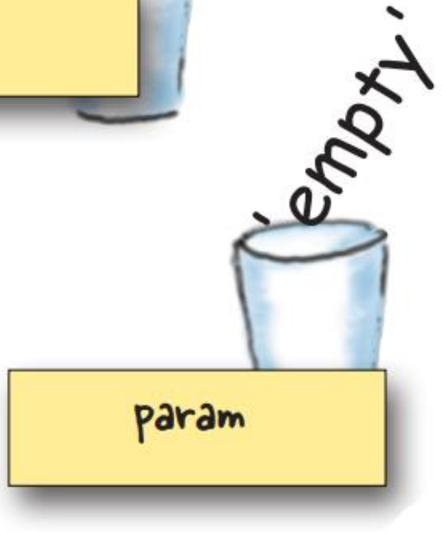


Python 3.6.0 Shell

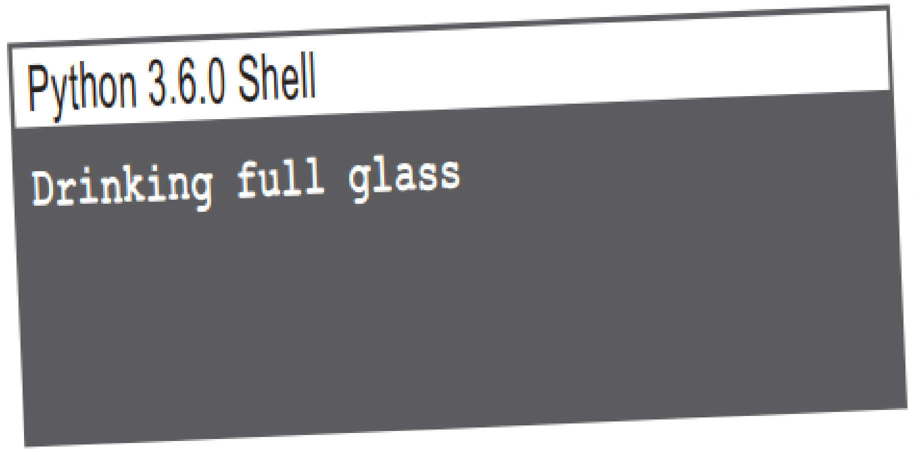
```
Drinking full glass
```

Making the drink me function call





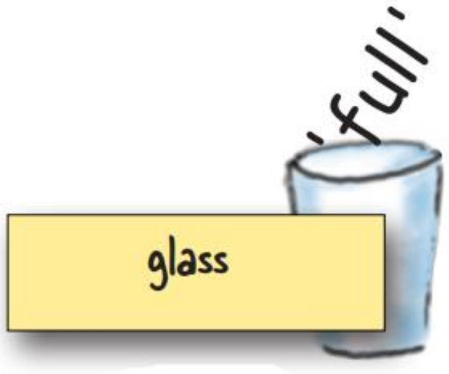
```
def drink_me(param):
      msg = 'Drinking ' + param + ' glass'
      print(msg)
      param = 'empty'

glass = 'full'
drink_me(glass)
print('The glass is', glass)
```



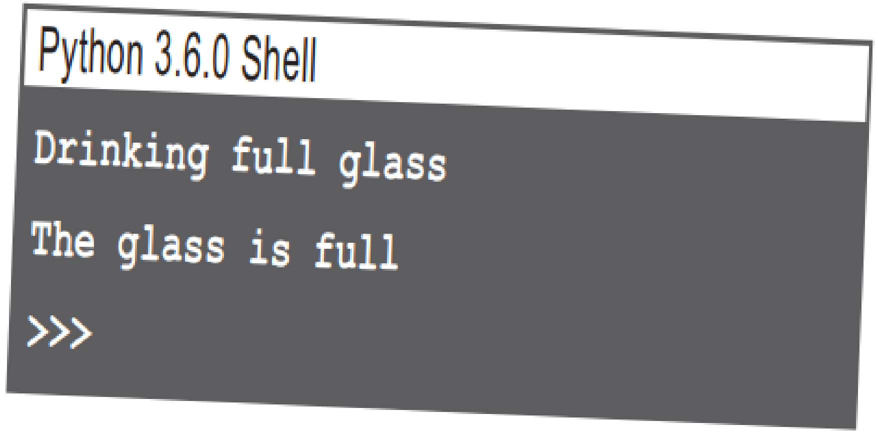
Python 3.6.0 Shell

```
Drinking full glass
```

Making the drink me function call



```
def drink_me(param):
     msg = 'Drinking ' + param + ' glass'
     print(msg)
     param = 'empty'

glass = 'full'
drink_me(glass)
print('The glass is', glass)
```



```
Drinking full glass
The glass is full
>>>
```

So, when we pass a variable to a function, we're actually passing the value of that variable, not the variable itself.

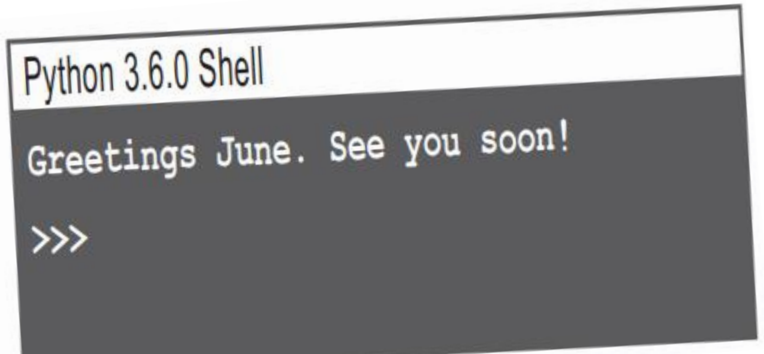
What about using global variables in functions?

Create a global variable greeting.

```
greeting = 'Greetings'


def greet(name, message):
    global greeting
    print(greeting, name + '.', message)


greet('June', 'See you soon!')
```

Tell our function we're going to use a global.

And use it.



Python 3.6.0 Shell

```
Greetings June. See you soon!
>>>
```

```
greeting = 'Greetings'

def greet(name, message):
    global greeting
    greeting = 'Hi'
    print(greeting, name + '.', message)

greet('June', 'See you soon!')
print(greeting)
```

Change the global greeting to 'Hi'.

Print the value of greeting after the call to greet.