CALLING FUNCTIONS

The Python Bakery

What Are Functions?

- Functions: Reusable chunks of code

Functions are reusable chunks of code.
Once code is wrapped in a function, we can use it in other places.
First, we'll learn how to use functions.
Later, we'll learn how to make them for ourselves.

Calling Functions

- "Use" a function
- "Call" a function
- "Invoke" a function
- "Execute" a function
- "Run" a function

When you want to "use" a Function, we say that you "call" it.
Another term is "invoking" a function.
You can also use the terms "execute" and "run", although those will usually be for full programs.

Syntax

```
# Function name followed by parentheses
round(3.75)
```

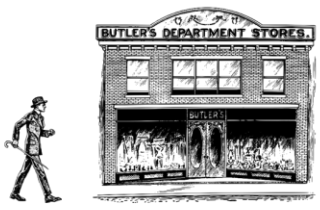
There are two essential parts to calling a function:
The name of the function, and then parentheses following the name.

Calling or Not

`go_to_store`

Talking about going to the store
(without parentheses)

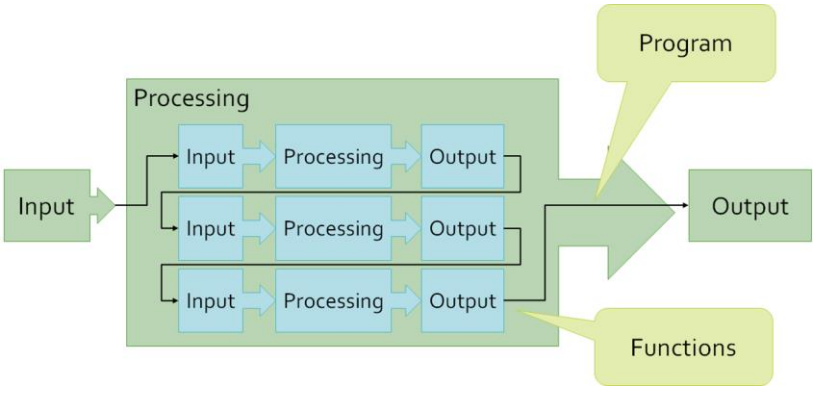
`go_to_store()`



Actually going to the store
(with parentheses)

To call a function, you MUST add parentheses.
Otherwise, you are simply referring to the name of the function rather than executing it.
Think of it as the difference between "thinking about going to the store" as opposed to "actually physically going to the store".
The parentheses are what actually make this a function call, and not just the function name being referenced for no reason.

Functions vs. Programs



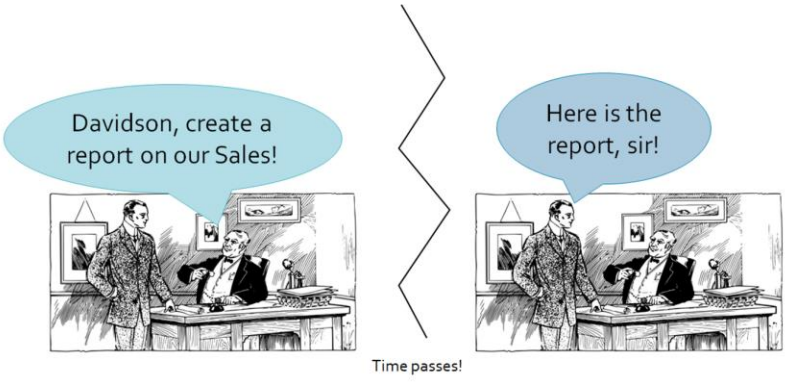
A program and a function are similar, but not quite the same.
They both have the same "Input-Process-Output" pattern.
However, functions are typically much smaller, and are meant to accomplish a single specific task.

Functions vs. Methods

```
message = "MAKE THIS LOWERCASE"
# Call the `lower` method
message = message.lower()
print(message)
```

Methods are a special kind of function.
They are attached strongly to a value.
The way we write a method call is slightly different too, since it needs to incorporate that value.
When you want to use a Method, you need the name, parentheses, AND two more pieces:
The calling value or variable and a period.
In later lessons, we'll learn more about this strange syntax.
For now, get comfortable with the difference between calling functions and methods.

Metaphor



You don't have to know how a function accomplishes its goal, in order to use it.
Think of it like a well-run office.
When you ask a co-worker to complete some work, you don't care how it happens, just the inputs and outputs to their process.
This separation of concerns allows you to focus on pieces of a program without worrying about the whole thing.

Returning Values

```
# call function, result is returned and assigned
new_value = round(3.9)

# Can use print it, update it, etc.
print(new_value)
new_value += 1
print(new_value)
```

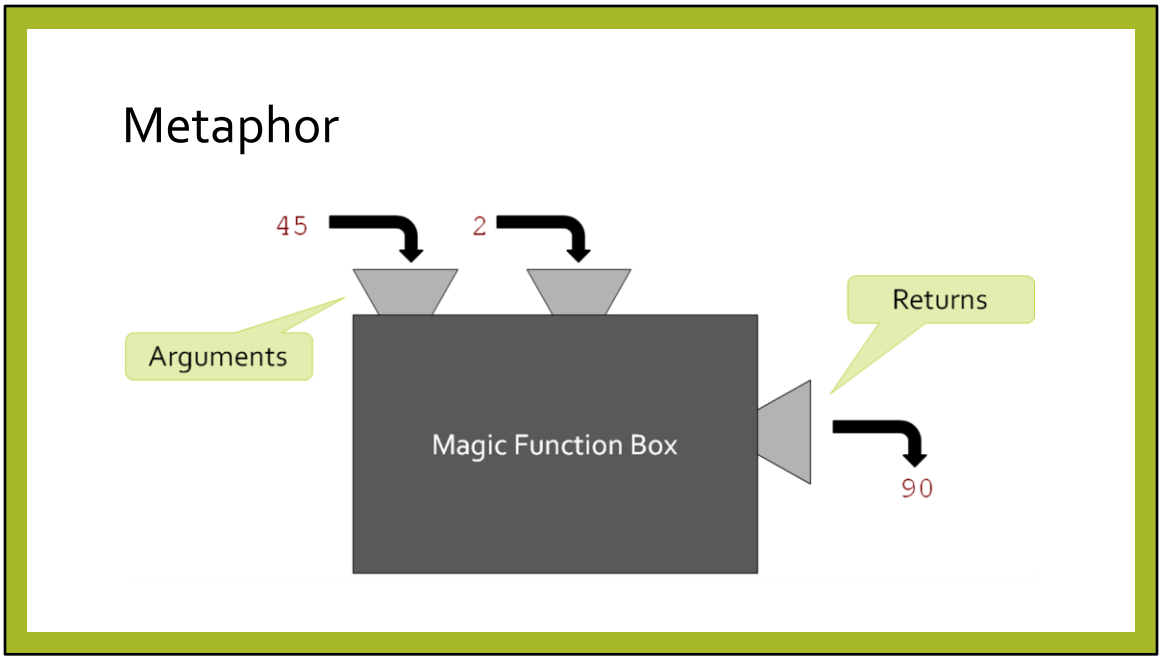
Functions are useful because they can return values.
Functions can return any type of value: string, boolean, integer, whatever.
Mentally, when a function is called, you can imagine the result replacing the entire function call.
In a way, this is similar to what happens when you do addition or subtraction.

Arguments

```
# 1 argument
round(4.3)

# 3 arguments
print("First", "Second", "Third")
```

One of the most useful features of functions are arguments.
Arguments are values that are given to functions that affect the behavior of the function.
Arguments for a function are placed inside the parentheses, and each one is separated by a comma.
We say that these arguments are being "passed" to the function.



Think of a function as a magic black box.
You cannot see inside the box to know how it works, but you do not need to.
Arguments are the knobs and dials on the outside of the box that let you control its workings.
Returned values are like a slot that dispenses the results of the function.