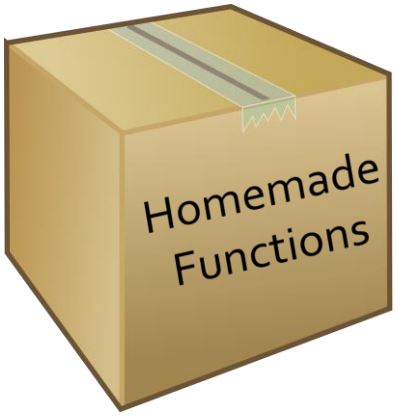
DEFINING FUNCTIONS

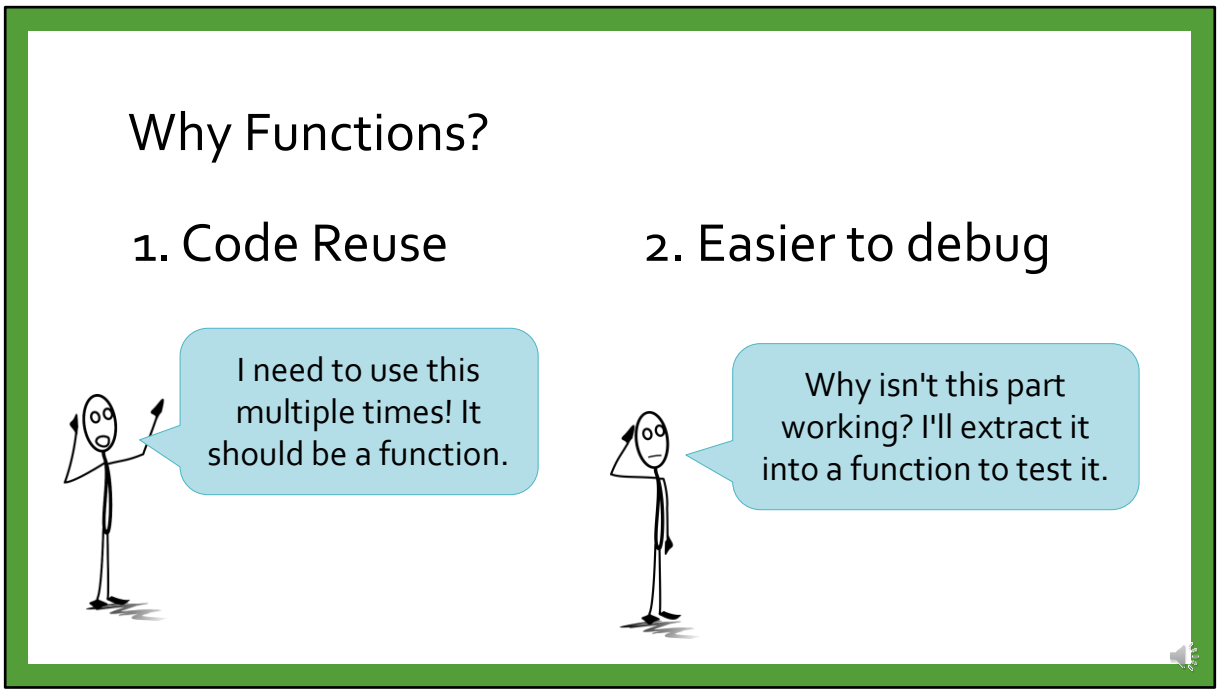
An Introduction to Computer Science

Let's learn about Defining Functions.

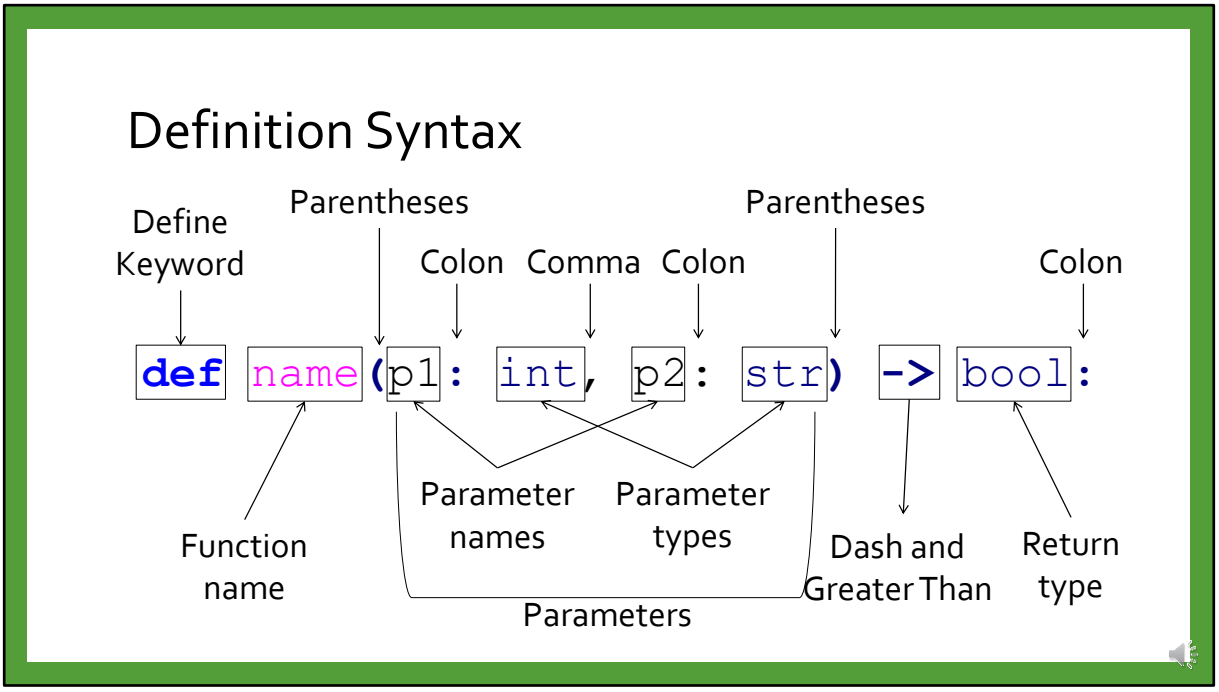
Defining Functions



You can create your own functions in Python.
In fact, this is one of the most powerful features of programming, the ability to create your own functions.



There are two major pragmatic advantages of functions:
First, they allow us to reuse a chunk of code in multiple places.
Second, they allow us to debug a chunk of code in isolation from the rest of the program.

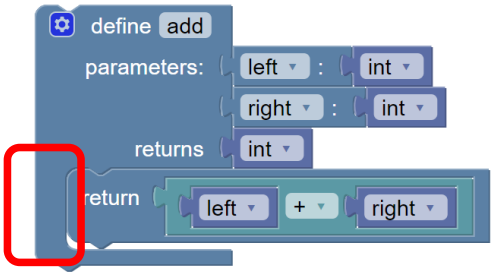


To create a new function, you use the `def` keyword, which stands for "define".
You write `def`, the name of the function, an open parentheses, each of the parameters separated by commas, a closed parentheses, a dash, a greater than, the return type of the function, and a colon.
The parameters need their variable names, a colon, and their parameter type.
This entire line is called the header.

Function Body

```
def add(left: int, right: int) -> int:
    return left + right
```

Indent with 4 spaces



When you call a function, you are executing the code stored in the "Body" of the function.
Everything "inside" the body should be indented 4 spaces.
In the block version, this is shown visually with the bar on the left.
The body must be there - in other words, it cannot be empty.

Naming a function

1. Use verbs
1. Function names can only have
   - Letters (abcABC)
   - Numbers (123)
   - Underscores (_)
2. Function Names must not begin with
   - Numbers

Usually, you should use a verb as the name of the function.
The name helps other programmers understand what the function does.
Naming a function is just like naming a variable: you may only use letters, numbers, and underscores, and it cannot start with a number.

Calling Your Functions

```
def add5(a_number: int):
    return a_number + 5

add5(10)
add5(3)
```

15

8

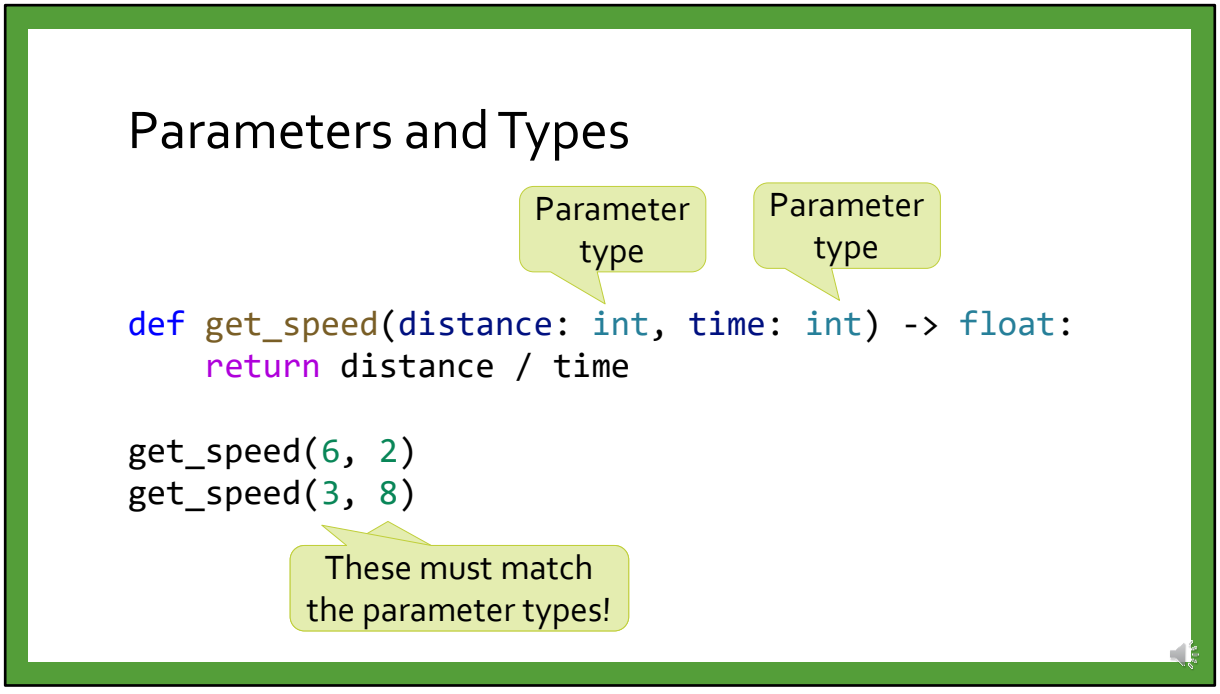
After you've defined a function, you can use it by calling the function.
As we did before, we combine the name of the function with **calling parentheses**.
Note how we still pass in **arguments**.
Here we call the function `add5` twice, first passing in the argument 10
and then the calling it again with the argument 3.

Parameters

```
def subtract(first: int, second: int) -> int:
    return first - second

subtract(3, 8)
subtract(-2, 5)
subtract(10, 10)
```

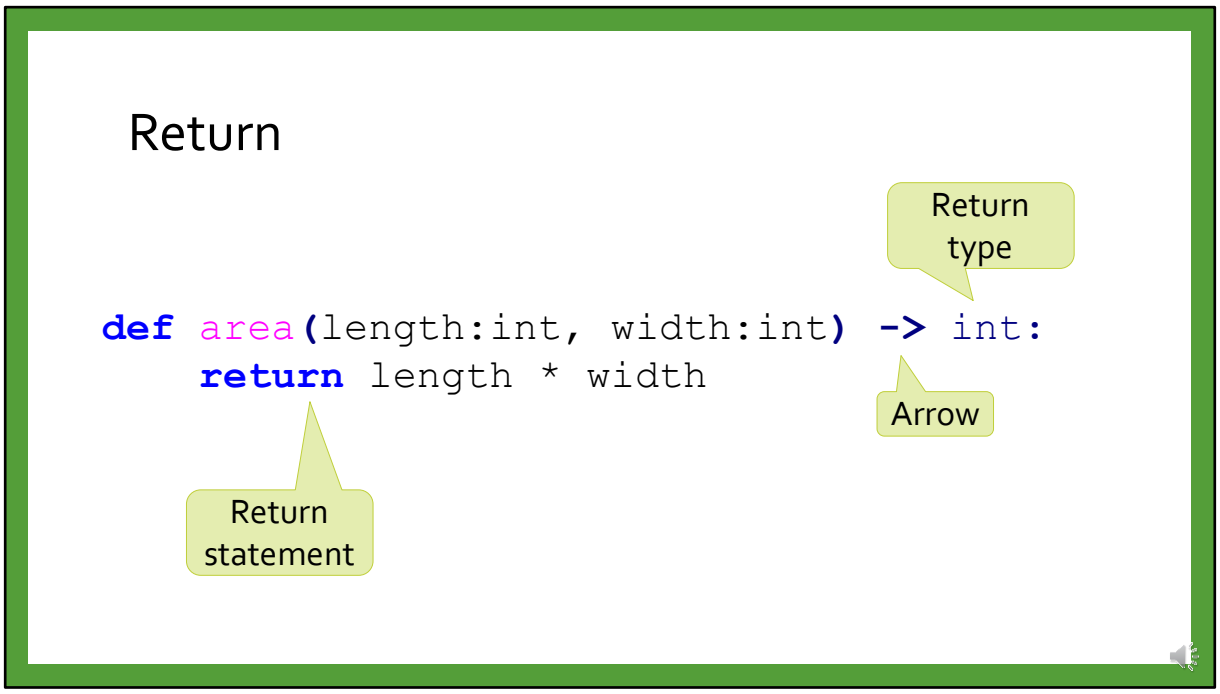
When you define a function, you can choose to add in **parameters**
 to the header.
These parameters will take on the value of the arguments when the f
unction is called.
This can be very tricky to understand.
Each argument exactly matches one parameter.
In the code below, the parameter `first` will match to 3, -2, and -
10.
The parameter `second` will match to 8, 5, and 10.
Remember, each function call happens one after the other.

Parameters and Types



In modern Python, we can specify the type of each parameter.
So far, we know of five types: int, str, float, bool, and None.
Any time you call that function, the arguments must match the type of the parameter.

Return



We describe what type of value the function returns using the arrow and a type in the header.
But note that it is the **return statement** that actually makes a value get returned; the header just describes what should be returned.

Calling and Printing

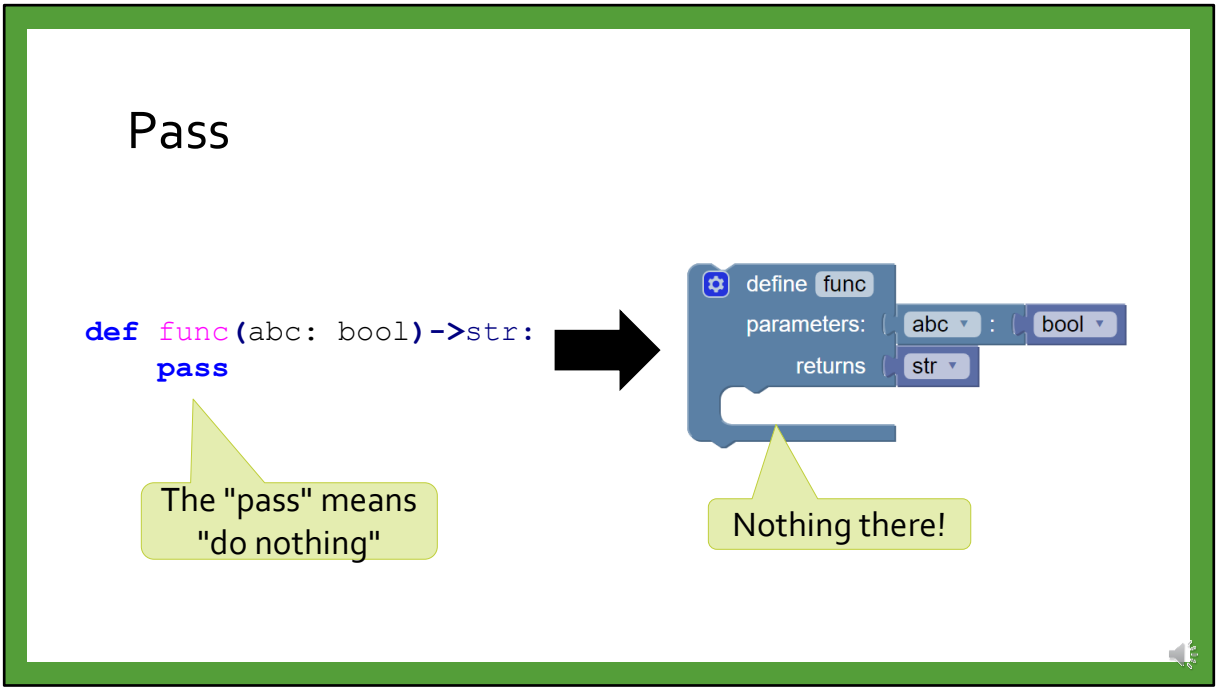
```
def area(length:int, width:int) -> int:
    return length * width

print(area(3, 4))
area(1, 8)
print(area(5, 2))
```

Prints 12

Calculates 8 but does not print!

Prints 10

When you call a function, a value is always returned.
Even if you forget the return statement, the special value `None` will be returned.
If you are writing code in the console, then you will see any non-`None` values appear.
But if you are writing code in a regular editor, the value will not appear in the console unless you use `print`.
We will sometimes print the result of calling a function, but remember
that printing is not necessary to call a function.



Sometimes, we want to define a function without writing its body just yet.
We use a special statement named "pass" to fill in the body until we're ready to write it.
Pass is a very special statement: it does absolutely nothing but take up space, telling the computer to "pass over" this line.
Since we always have to have a body, if we didn't put the word pass there, Python would crash with a syntax error.