Computational Thinking

Programming Fundamentals with JavaScript

- Variables 9 min
- Methods 7 min
- Functions 5 min
- Types 4 min
- DukeLearnToProgram Environment 10 min
- Try It! Using Variables, Methods and Functions 30 min
- For Loops 6 min
- Try It! Using For Loops 30 min
- Conditional Execution 7 min
- Programming Exercise: Modifying Images 1h 30m
- **Practice Quiz:** Modifying Images with JavaScript 8 questions

Implementing the Green Screen Algorithm

Review

Consider the following code.

```
1  function square (x){
2    var ans = x * x;
3    return ans;
4  }
5
6  var num = square(3) + 2;
```

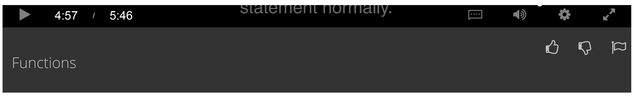
What will be the value of **num** after this code runs?

- 2
- 9
- 11

Correct
Correct! 3x3 + 2 = 11

- 5

Continue



Functions

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:03
Methods combine together multiple, potentially complex steps that operate on a specific object. There's a similar concept which does not work on an object called a function. For example, print(x), is a call to the print function passing in x. Print(x) will print out whatever x is. This is a function, as it does not belong to any particular object.

0:27
Notice there is no object dot before the word print, but the syntax is otherwise the same. You can write your own functions, which can be quite useful to avoid duplicating code. And to make testing and debugging easier. You could write your own methods too, but we are not going to talk about writing classes and methods in this course. We will, however, discuss them in Java in the later courses.

0:52
To see an example of writing your own function, let us take a look at this relatively simple example, a function that takes a number and squares it.

1:02
We have defined the function first and then called it later.

1:07
Now let's break down the syntax, then see the semantics.

1:12
First we have the keyword function, which says that we are about to define a new function.

1:18
Then we have the name of the function. As with variables, you can pick most any name you want. But it should be descriptive. In this case, square is a good name since it describes what the function does. Square's a number.

1:34
Next, we have the parameter list in parentheses. Here we are declaring the names of the parameters. These will act like variables which are initialized by the values passed in when the function is called. They tell the function what specifically it should do.

1:51
In this case, the parameter says, what particular number to square. We decided to call it x. If there were multiple parameters, we could separate their names by commas.

2:03
Next is the body of the function. This is the code that specifies what the function should do.

2:09
The body of the function is always inside of curly braces.

2:14
There's a new type of statement in the body of this function, return ans;. This is a return statement, which is how a function says what it's answer is. The syntax of a return statement is the keyword return followed by an expression whose value is what the function should give back as its answer, followed by a semicolon. Later in the code we have a call to this function, which is how we make use of it. Here we are squaring 4 and using the result of that computation to initialize a variable y. Now that you have seen the syntax of declaring and calling a function, let's step through the behavior. The declaration itself is not executed, it just tells the computer what the function means for later use. So we start here, right before the variable declaration. The first thing that we need to do when we call a function, is create a frame for it.

3:13
This is a space for the function to have its own parameters and variables. As you will see soon, they go away when the function is done.

3:22
Next, we need to make a box for each parameter that square expects. In this case, square expects one parameter called X. We are going to create one box named X. We will initialize the X parameter with the value 4. Since that is what was passed into the function call.

3:42
Next, we're going to make a note of where we called square from.

3:46
We know that with a 1 in the top corner of the frame and then the code on top of the call.

3:53
This may seem trivial in this example, but if the square function were more complicated and called in many places, we might forget where to come back to. Now that the frame is set up, we jump into the code for square.

4:07
Our execution arrow moves into the body right before the first line of code. Now we begin executing the code here according to all the normal rules. We make a box for ans and initialize it to 4 times 4 which is 16. Now we are right before a return statement. Return statements are a new concept, but they tell the computer to give an answer back from this function.

4:36
In this case, the answer to give back is 16, since we are returning ans, and the value of ans is 16.

4:45
This means that the call to square that we are executing will evaluate to 16.

4:51
Now our execution returns to where we made the call. We finish the assignment statement normally.

4:58
Okay, great.

5:00
Now you know how to call methods and functions. As well as the basics of how to write your own function. But why are these so important? Well, they are a great example of abstraction. They package up a computation which may be quite complicated, or which may require knowing details that you should not need to be concerned with, and give that computation a simple interface.

5:25
For example, the interface of this simple image method getWidth is that you call it on an image and it gives you the image's width.

5:35
The implementation is hidden in the dukelearntoprogram.com libraries and you've not seen it all nor do you need to see it to use it. Thank you.

Downloads

- Lecture Video mp4
- Subtitles (English) WebVTT
- Transcript (English) txt
- Lecture Slides pdf

Would you like to help us translate the transcript and subtitles into additional languages?