Function Expressions

Once you know how to declare a function, a whole new set of possibilities will open up to you.

For instance, remember how you can store anything you want in a variable? Well, in JavaScript, you can also store functions in variables. When a function is stored *inside* a variable it's called a **function expression**.

```
var catSays = function(max) {
  var catMessage = "";
  for (var i = 0; i < max; i++) {
    catMessage += "meow ";
  }
  return catMessage;
};
```

Notice how the `function` keyword no longer has a name.

```
var catSays = function(max) {
  // code here
};
```

It's an **anonymous function**, a function with no name, and you've stored it in a variable called `catSays`.

And, if you try accessing the value of the variable `catSays`, you'll even see the function returned back to you.

```
catSays;
```

Returns:

```
function(max) {
  var catMessage = ""
  for (var i = 0; i < max; i++) {
    catMessage += "meow ";
  }
  return catMessage;
}
```
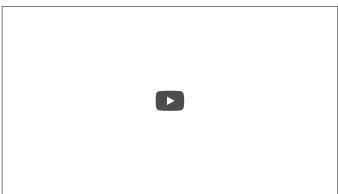
Function expressions and hoisting

Deciding when to use a function expression and when to use a function declaration can depend on a few things, and you will see some ways to use them in the next section. But, one thing you'll want to be careful of, is hoisting.

expression will not be loaded until the interpreter reaches it in the script.

MENTORSHIP

This animation is showing the difference between how hoisting impacts declared functions vs. function expressions. Notice how in the animation the function expression is not hoisted, but the declared function is hoisted.

NEXT

Mentorship
Get support and stay on track

Get support and stay on track