Functions as parameters

Being able to store a function in a variable makes it really simple to pass the function into another function. A function that is passed into another function is called a **callback**. Let's say you had a `helloCat()` function, and you wanted it to return "Hello" followed by a string of "meows" like you had with `catSays`. Well, rather than redoing all of your hard work, you can make `helloCat()` accept a callback function, and pass in `catSays`.

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

// function declaration helloCat accepting a callback
function helloCat(callbackFunc) {
  return "Hello " + callbackFunc(3);
}

// pass in catSays as a callback function
helloCat(catSays);
```

Named function expressions

Inline function expressions

A function expression is when a function is assigned to a variable. And, in JavaScript, this can also happen when you pass a function *inline* as an argument to another function. Take the `favoriteMovie` example for instance:

```
// Function expression that assigns the function displayFavorite
// to the variable favoriteMovie
var favoriteMovie = function displayFavorite(movieName) {
  console.log("My favorite movie is " + movieName);
};

// Function declaration that has two parameters: a function for displaying
// a message, along with a name of a movie
function movies(messageFunction, name) {
  messageFunction(name);
}

// Call the movies function, pass in the favoriteMovie function and name of movie
movies(favoriteMovie, "Finding Nemo");
```

> **Returns:** My favorite movie is Finding Nemo

```
// Function declaration that takes in two arguments: a function for displaying
// a message, along with a name of a movie
function movies(messageFunction, name) {
  messageFunction(name);
}

// Call the movies function, pass in the function and name of movie
movies(function displayFavorite(movieName) {
  console.log("My favorite movie is " + movieName);
}, "Finding Nemo");
```

> **Returns:** My favorite movie is Finding Nemo

This type of syntax, writing function expressions that pass a function into another function inline, is really common in JavaScript. It can be a little tricky at first, but be patient, keep practicing, and you'll start to get the hang of it!

Why use anonymous inline function expressions?

Using an anonymous inline function expression might seem like a very not-useful thing at first. Why define a function that can only be used once and you can't even call it by name?

Anonymous inline function expressions are often used with function callbacks that are probably not going to be reused elsewhere. Yes, you could store the function in a variable, give it a name, and pass it in like you saw in the examples above. However, when you know the function is not going to be reused, it could save you many lines of code to just define it inline.

NEXT