

# Higher-order Functions I

---



# Overview

```
1  /*
2    - Why functions are special
3    - Passing functions into other functions (callbacks)
4    - .forEach
5  */
6
7
8
9
10
11
12
13
14
```



# Why are functions special?

```
1  /* Functions are special in JS because...they aren't special */
2
3  /* We think of functions as being different from other values in JS */
4
5  /* Strings, numbers, arrays: we're used to passing them into functions,
6     or returning them from functions */
7
8  /* But functions sometimes seem like they're in a different category,
9     rooted to the line of code where they're defined */
10
11 /* In JS, functions are 'first-class objects', which is another way of
12    saying that functions are like any other value in JS */
13
14
```



[ 'happy', 'happy', 'happy' ]

## example: amazingArray

```
1  /* we know we can push strings, or any value into arrays */
2
3  let amazingArray = [];
4
5  let happyString = 'happy';
6
7  amazingArray.push(happyString);
8  amazingArray.push(happyString);
9  amazingArray.push(happyString);
10
11 console.log(amazingArray);
12
13
14
```



[ [Function], [Function], [Function] ]

## example: amazingArray

```
1  /* functions aren't special. we can push them into an array, too! */
2
3  let amazingArray = [];
4
5  function happyFunction() {
6    console.log('I am happy!');
7  }
8
9  amazingArray.push(happyFunction);
10 amazingArray.push(happyFunction);
11 amazingArray.push(happyFunction);
12
13 console.log(amazingArray);
14
```



I am happy!  
I am happy!  
I am happy!

## example: amazingArray

```
1  /* how do we call all the functions in the array? how have we always
2     looped through an array of values? */
3
4  function happyFunction() {
5     console.log('I am happy!');
6  }
7
8  let amazingArray = [happyFunction, happyFunction, happyFunction];
9
10 for (let i = 0; i < amazingArray.length; i++) {
11     let element = amazingArray[i]; // each element is a function!
12     element(); // we invoke the function the same as always, with parentheses
13 }
14
```



# Passing functions into functions

```
1  /* if functions are like any other value, we can pass functions into other
2     functions, too */
3
4  /* functions that take a function or return a function are called
5     "higher-order functions" */
6  function logsAType(value) {
7      console.log(typeof value);
8  }
9
10 function happyFunction() {
11     console.log('I am happy!');
12 }
13
14 logsAType(happyFunction);
```



# Passing functions into functions

```
1  /* if we want happyFunction to run, we have to call it */
2
3  function callsAFunction(anotherFunction) {
4    anotherFunction(); // invoking this time
5  }
6
7  function happyFunction() {
8    console.log('I am happy!');
9  }
10
11 callsAFunction(happyFunction);
12
13
14
```





I am happy!  
I am great!

# Passing functions into functions

```
1  /* a function passed into another function is often called a callback */
2  function callsACallback(callback) {
3      callback();
4  }
5
6  function happyFunction() {
7      console.log('I am happy!');
8  }
9
10 function greatFunction() {
11     console.log('I am great');
12 }
13 callsACallback(happyFunction);
14 callsACallback(greatFunction);
```



Hi Sadie  
Bye Sadie

## example: callsWithName

```
1  function saysHi(name) {  
2    console.log('Hi', name);  
3  }  
4  
5  function saysBye(name) {  
6    console.log('Bye', name);  
7  }  
8  
9  function callsWithName(name, callback) {  
10   callback(name);  
11 }  
12  
13 callsWithName('Sadie', saysHi);  
14 callsWithName('Sadie', saysBye);
```



## example: callsWithHello

```
1  function addWorld(string) {  
2    // string = 'hello';  
3    // return 'hello world';  
4    return string + ' world';  
5  }  
6  
7  function callsWithHello(func) {  
8    // return addWorld('hello');  
9    return func('hello');  
10 }  
11  
12 let result = callsWithHello(addWorld);  
13 console.log(result);  
14
```



Hello, Jane!  
Hello, Jill!  
Hello, Pip!  
Hello, Mike!

## example: sayToAll

```
1 function sayToAll(names, sayWithNameFunc) {  
2   for(let i = 0; i < names.length; i++) {  
3     sayWithNameFunc(names[i]);  
4   }  
5 }  
6  
7 let group = ["Jane", "Jill", "Pip", "Mike"];  
8  
9 function sayHelloWithName(name) {  
10  console.log("Hello, " + name + "!");  
11 }  
12  
13 sayToAll(group, sayHelloWithName);  
14
```



Bye, Jane!  
Bye, Jill!  
Bye, Pip!  
Bye, Mike!

## example: sayToAll

```
1  /* we can pass anonymous functions into another function, too */
2
3  function sayToAll(names, sayWithNameFunc) {
4    for(let i = 0; i < names.length; i++) {
5      sayWithNameFunc(names[i]);
6    }
7  }
8
9  let group = ["Jane", "Jill", "Pip", "Mike"];
10
11 sayToAll(group, function (name) {
12   console.log("Bye, " + name + "!");
13 });
14
```



## example: calc

```
1  function plus(num1, num2) {  
2    return num1 + num2;  
3  }  
4  
5  function minus(num1, num2) {  
6    return num1 - num2;  
7  }  
8  
9  function calc(num1, operationFunc, num2) {  
10   return operationFunc(num1, num2);  
11 }  
12  
13 console.log(calc(10, plus, 20));  
14 console.log(calc(50, minus, 10));
```



# .forEach

```
1  /* a function passed into another function is often called a callback */
2
3  /* some built-in JS features use callbacks */
4
5  /* .forEach is an array method; it accepts a callback as its only
6     argument */
7
8  /* .forEach calls the callback for each element in the array */
9
10 /* when .forEach calls the callback, it passes the current element
11    as the first argument of the callback */
12
13
14
```



# .forEach

```
1  let bridges = ['Brooklyn', 'Golden Gate', 'London'];
2
3  function logUpperCase(string) {
4    console.log(string.toUpperCase());
5  }
6
7  bridges.forEach(logUpperCase);
8
9
10
11
12
13
14
```





Brooklyn is at index 0  
Golden Gate is at index 1  
London is at index 2

# .forEach

```
1  /* the callback passed into forEach also takes an optional second
2     argument. forEach passes the current index of the element as the second
3     argument. */
4
5  let bridges = ['Brooklyn', 'Golden Gate', 'London'];
6
7  function logWithIdx(string, idx) {
8    console.log(string, 'is at index', idx);
9  }
10
11  bridges.forEach(logWithIdx);
12
13
14
```



## Recap

```
1  /*
2   - Why functions are special
3   - Passing functions into other functions (callbacks)
4   - .forEach
5  */
6
7
8
9
10
11
12
13
14
```





# Romance.js





# Romance.js: Overview

- 100% optional project
- Directions available on LearnDot
- Program overview:
  - Input: a corpus of text (speech, song, poetry, prose)
  - Work: uses a Markov chain to turn input into n lines of new text (see workshop for details)
  - Output: a poem of n lines
- Complete by final night of class to participate in friendly competition; win one of three categories:
  - Most human sounding
  - Funniest
  - Most romantic



# Romance.js: Examples

- Corpus from Trump's victory speech and Clinton's concession speech (note the student added extra logic to turn the poems into haikus!) ([repl](#))
- Corpus from a poem by Pablo Neruda; outputted poems tend to be quite angsty and occasionally romantic ([repl](#))
- Corpus from a scathing review of Guy Fieri's restaurant in NYC ([repl](#))