# Higher-order Functions I



#### Overview

```
- Why functions are special
 - Passing functions into other functions (callbacks)
 - .forEach
*/
```



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



## example: amazingArray

```
/* we know we can push strings, or any value into arrays */
let amazingArray = [];
let happyString = 'happy';
amazingArray.push(happyString);
amazingArray.push(happyString);
amazingArray.push(happyString);
console.log(amazingArray);
```



## example: amazingArray

```
/* functions aren't special. we can push them into an array, too! */
let amazingArray = [];
function happyFunction() {
 console.log('I am happy!');
amazingArray.push(happyFunction);
amazingArray.push(happyFunction);
amazingArray.push(happyFunction);
console.log(amazingArray);
```



## example: amazingArray

```
/* how do we call all the functions in the array? how have we always
  looped through an array of values? */
function happyFunction() {
 console.log('I am happy!');
let amazingArray = [happyFunction, happyFunction, happyFunction];
for (let i = 0; i < amazingArray.length; i++) {
 let element = amazingArray[i]; // each element is a function!
 element();
```



## Passing values into functions

```
/* we know we can pass strings, or any value, into a function */
function logsAType(value) {
 console.log(typeof value);
logsAType('happy string');
```



## Passing functions into functions

8

```
/* if functions are like any other value, we can pass functions into other
  functions, too */
/* functions that take a function or return a function are called
  "higher-order functions" */
function logsAType(value) {
 console.log(typeof value);
function happyFunction() {
 console.log('I am happy!');
logsAType(happyFunction);
```



# Passing functions into functions

```
/* if we want happyFunction to run, we have to call it */
function callsAFunction(anotherFunction) {
 anotherFunction(); // invoking this time
function happyFunction() {
 console.log('I am happy!');
callsAFunction(happyFunction);
```



## Passing functions into functions

```
/* a function passed into another function is often called a callback */
function callsACallback(callback) {
 callback();
function happyFunction() {
 console.log('I am happy!');
function greatFunction() {
 console.log('I am great');
callsACallback(happyFunction);
callsACallback(greatFunction);
```



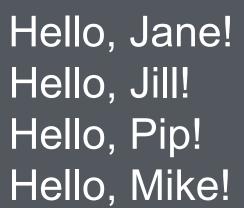
## example: callsWithName

```
function saysHi(name) {
 console.log('Hi', name);
function saysBye(name) {
 console.log('Bye', name);
function callsWithName(name, callback) {
 sayHiOrBye(name);
callsWithName('Sadie', saysHi);
callsWithName('Sadie', saysBye);
```



## example: callsWithHello

```
function addWorld(string) {
 return string + 'world';
function callsWithHello(func) {
 return func('hello');
let result = callsWithHello(addWorld);
console.log(result);
```



## example: sayToAll

```
function sayToAll(names, sayWithNameFunc) {
     for(let i = 0; i < names.length; i++) {
      sayWithNameFunc(names[i]);
    let group = ["Jane", "Jill", "Pip", "Mike"];
    function sayHelloWithName(name) {
     console.log("Hello, " + name + "!");
    sayToAll(group, sayHelloWithName);
14
```



## example: sayToAll

```
/* we can pass anonymous functions into another function, too */
function sayToAll(names, sayWithNameFunc) {
 for(let i = 0; i < names.length; i++) {
  sayWithNameFunc(names[i]);
let group = ["Jane", "Jill", "Pip", "Mike"];
sayToAll(group, function (name) {
 console.log("Bye, " + name + "!");
});
```

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

## example: calc

```
function plus(num1, num2) {
 return num1 + num2;
function minus(num1, num2) {
 return num1 - num2;
function calc(num1, operationFunc, num2) {
 return operationFunc(num1, num2);
console.log(calc(10, plus, 20));
console.log(calc(50, minus, 10));
```

15



```
/* a function passed into another function is often called a callback */
/* some built-in JS features use callbacks */
/* .forEach is an array method; it accepts a callback as its only
  argument */
/* .forEach calls the callback for each element in the array */
/* when .forEach calls the callback, it passes the current element
  as the first argument of the callback */
```

#### .forEach

```
let bridges = ['Brooklyn', 'Golden Gate', 'London'];
function logUpperCase(string) {
 console.log(string.toUpperCase());
bridges.forEach(logUpperCase);
```

### .forEach

```
/* the callback passed into forEach also takes an optional second
 argument. for Each passes the current index of the element as the second
 argument. */
let bridges = ['Brooklyn', 'Golden Gate', 'London'];
function logWithIdx(string, idx) {
 console.log(string, 'is at index', idx);
bridges.forEach(logWithIdx);
```



# Recap

```
- Why functions are special
 - Passing functions into other functions (callbacks)
 - .forEach
*/
```