Higher-order Functions I

Overview

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

Why are functions special?

```
/* 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';
6
   amazingArray.push(happyString);
   amazingArray.push(happyString);
   amazingArray.push(happyString);
10
   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);
13 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; <math>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

```
/* 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!');
14 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!');
10
   callsAFunction(happyFunction);
```

example: callsWithName

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

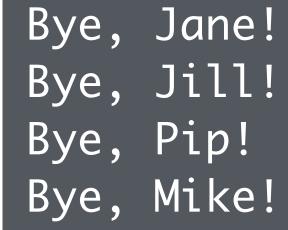
example: callsWithHello

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

example: sayToAll

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

```
function sayToAll(names, sayWithNameFunc) {
     for(let i = 0; i < names.length; <math>i++) {
       sayWithNameFunc(names[i]);
   let group = ["Jane", "Jill", "Pip", "Mike"];
   function sayHelloWithName(name) {
     console.log("Hello, " + name + "!");
10
13 sayToAll(group, sayHelloWithName);
```



example: sayToAll

```
/* we can pass anonymous functions into another function, too */
   function sayToAll(names, sayWithNameFunc) {
     for(let i = 0; i < names.length; <math>i++) {
       sayWithNameFunc(names[i]);
   let group = ["Jane", "Jill", "Pip", "Mike"];
10
   sayToAll(group, function (name) {
     console.log("Bye, " + name + "!");
13 });
```

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);
13 console.log(calc(10, plus, 20));
14 console.log(calc(50, minus, 10));
```

.forEach

```
/* 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. forEach 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);
10
   bridges.forEach(logWithIdx);
```

Recap

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



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 Hilary'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)