JAVASCRIPT

CALLBACKS & ITERATORS

CALLBACKS

- Functions can take functions as arguments, and can be returned by other functions.
- Functions that do this are called higher-order functions.
- Any function that is passed as an argument is called a callback function.

CALLBACKS

```
function addTwo(number) {
  return number + 2;
}

function subtractTwo(number) {
  return number - 2;
}

function doToNumber(number, action) {
  return action(number);
}

doToNumber(4, addTwo);

doToNumber(4, subtractTwo);
```

- doToNumber takes another function as an argument (a callback!)
- We invoke that function and return the value
- It is important NOT to use the clappers () when passing a function as an argument

ANONYMOUS FUNCTIONS

```
function double(number) {
  return number * 2;
}

function doToNumber(number, action) {
  return action(number);
}

doToNumber(4, double);

doToNumber(4, function(number) {
  return number * 2;
});
```

- Callbacks can be declared and named (like double is)
- Callbacks can also be anonymous functions (where they are created directly as an argument)

CALLBACKS

```
function addTwo(number) {
  return number + 2;
}
function subtractTwo(number) {
  return number - 2;
}
function doToEach(array, action) {
  for(var i=0; i<array.length; i++){
    array[i] = action(array[i]);
  }
}

var numbers = [4, 7, 1, 8];
doToEach(numbers, addTwo);
doToEach(numbers, subtractTwo);
```

- What is happening here?

CALLBACKS

```
function addTwo(number) {
  return number + 2;
}
function subtractTwo(number) {
  return number - 2;
}
function doToEach(array, action) {
  for(var i=0; i<array.length; i++){
    array[i] = action(array[i]);
  }
}

var numbers = [4, 7, 1, 8];
doToEach(numbers, addTwo);
doToEach(numbers, subtractTwo);
```

- We are passing a function and an array into doToEach
- doToEach loops through the array and changes each item based on what the function returns

CALLBACKS

```
function isEven(number) {
  return number % 2 === 0;
}
function filterBy(array, condition){
  var filteredItems = [];
  for(var i=0; i<array.length; i++){
    if(condition(array[i])){
      filteredItems.push(array[i])
    }
  }
  return filteredItems;
}

var numbers = [4, 7, 1, 8];
filterBy(numbers, isEven)
```

- What is happening here?

CALLBACKS

```
function isEven(number) {
  return number % 2 === 0;
}
function filterBy(array, condition){
  var filteredItems = [];
  for(var i=0; i<array.length; i++){
    if(condition(array[i])){
      filteredItems.push(array[i])
    }
  }
  return filteredItems;
}

var numbers = [4, 7, 1, 8];
filterBy(numbers, isEven)
```

- We are passing a function and an array into filterBy
- filterBy loops through the array and creates a new array where each value returns true from the callback

IMPERATIVE PROGRAMMING

- Until now we have been using an imperative style of programming
- We go through the code step by step

IMPERATIVE PROGRAMMING

```
for (var i=0; i<array.length; i++){
  // do stuff
}
```

For loops are imperative. We are saying:

- Initialize a looping variable
- Use the looping variable to access an element in the array
- Increment the looping variable
- If the looping variable is less than the length of the array, loop again

DECLARATIVE PROGRAMMING

With declarative programming we write code that *describes* what we do

```
array.forEach(function(item){
  // do stuff
}
```

How are we iterating? We don't need to worry about that.

ITERATORS

methods that declaratively iterate
over an array's elements

Method	Purpose	Returns	Callback Should
`forEach(cb)`	General purpose	undefined	Do whatever you want
`map(cb)`	Create new array from source array	new array	Modify each element as desired and return it
`filter(cb)`	Filter source array	new array	Return truthy if elem is to be included
`find(cb)`	Find an element	the first `elem` found	Return truthy if elem is what you're looking for

.forEach( )

- General purpose iterator method.

```
var friends = ["Melissa", "Marc", "Andrew", "Nick"];

friends.forEach(function(friend) {
    console.log(`I have a friend named ` + friend);
});

// logs out "I have a friend named <friend's name>" for each friend
```

- Loops through the array and invokes the callback on each item

.forEach( )

```
var friends = ["Melissa", "Marc", "Andrew", "Nick"];

friends.forEach(function(friend) {
    console.log(‘I have a friend named ’ + friend);
});

// same as

for(var i=0; i<friends.length; i++){
    console.log(‘I have a friend named ‘ + friends[i])
}
```

.forEach( )

```
var friends = ["Melissa", "Marc", "Andrew", "Nick"];

friends.forEach();
```

- We call the iterator method on an array.

.forEach( )

```
var friends = ["Melissa", "Marc", "Andrew", "Nick"];

friends.forEach(function(friend) {

});
```

- The method takes a callback function as an argument
- This function will be called with each item in the array
- The parameter that is in the callback function (friend) will represent the current element in the iteration

.forEach( )

```
var friends = ["Melissa", "Marc", "Andrew", "Nick"];

friends.forEach(function(friend) {
    console.log('I have a friend named ' + friend);
});
```

- We can then use that element to do whatever we want!
- Each time `friend` will be different

PRACTICE

Using forEach log out each of my friends but with their name lower-cased.

.map( )

- Create a new array from a source array, usually "transforming" its values. The returned array is always the same length as the source array.

```
var nums = [1, 2, 3];
var squared = nums.map(function(num) {
    return num * num;
});
```

- Loops through the array and returns a new array with the callback invoked on each item

.map( )

```
var obj = {
    a: "A",
    b: "B",
    c: "C",
    one: 1,
    two: 2,
    three: 3
};

var types = Object.keys(obj).map(function(key) {
    return typeof obj[key];
});
```

PRACTICE

Given an array of instructors, use map to create a new array that adds the string " is awesome" to each element in the array.

.filter( )

- Select certain elements from a source array

```
var nums = [100, 2, 5, 42, 99];

var odds = nums.filter(function(num) {
    return num % 2 !== 0;
});
```

- Loops through the array and returns all elements where the callback returns true
- When using filter, the callback ALWAYS needs to return a boolean

.filter( )

```
var cars = [
    {color: 'red', make: 'BMW', year: 2001},
    {color: 'white', make: 'Toyota', year: 2013},
    {color: 'blue', make: 'Ford', year: 2014},
    {color: 'white', make: 'Tesla', year: 2016}
];

var whiteCars = cars.filter(function(car) {
    return car.color === ‘white’
});
```

PRACTICE

Filter out all "jerks"!

```
var people = ["jerks", "nice people", "jerks", "nice people", "nice people"];
```

.find( )

- Find an element in an array

```
var cars = [
    {color: 'red', make: 'BMW', year: 2001},
    {color: 'white', make: 'Toyota', year: 2013},
    {color: 'blue', make: 'Ford', year: 2014},
    {color: 'white', make: 'Tesla', year: 2016}
];

var firstWhiteCar = cars.find(function(car) {
    return car.color === 'white';
});
```

- Loops through the array and returns the first element where the callback returns true

.find( )

```
var cars = [
   {color: 'red', make: 'BMW', year: 2001},
   {color: 'white', make: 'Toyota', year: 2013},
   {color: 'blue', make: 'Ford', year: 2014},
   {color: 'white', make: 'Tesla', year: 2016}
];

var missingCar = cars.find(function(car) {
    return car.color === 'black';
});

// missingCar = undefined
```

- If there is no element that matches the callback, it will return undefined

PRACTICE

Find the first car whose year is older than 2014 and assign it to a variable named notTooOldCar

REVIEW

- What is a callback?
- Imperative vs Declarative
- What are Iterators?
- forEach()
- map()
- filter()
- find()