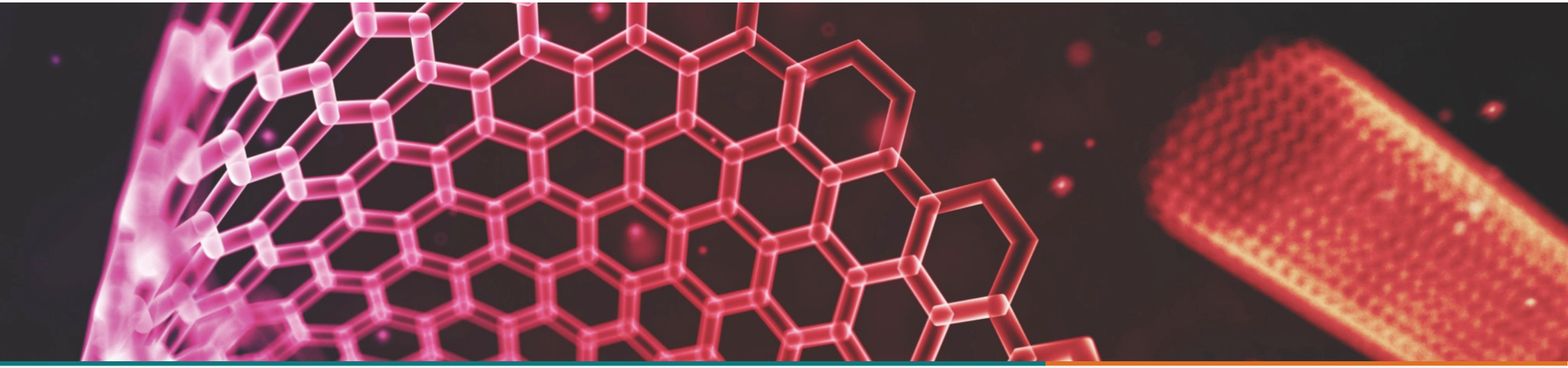
STEVENS
INSTITUTE of TECHNOLOGY
Schaefer School of Engineering & Science

CS 546 – Web Programming I
Introduction to JavaScript



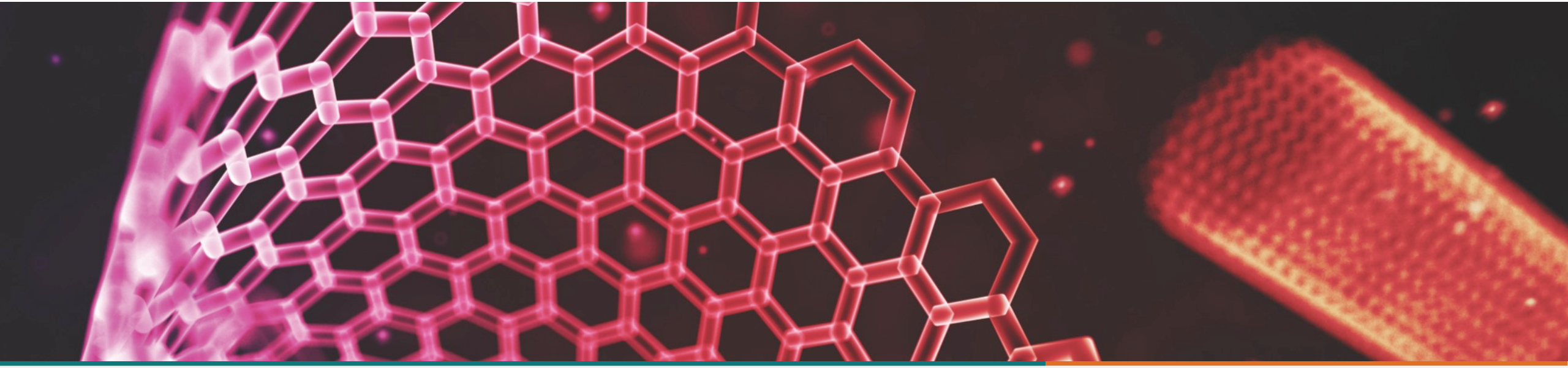
STEVENS
INSTITUTE of TECHNOLOGY

Schaefer School of
Engineering & Science

stevens.edu

Patrick Hill
Adjunct Professor
Computer Science Department
Patrick.Hill@stevens.edu

Introduction to Node.js



What is Node.js?

There are many languages and environments that you can develop web applications in, each with great strengths and great weaknesses.

Node.js is a JavaScript runtime environment that runs on a computer, rather than in a browser.

- In simple terms, it's JavaScript being run as a script on a computer.

You can run these scripts from the command line.

Node.js often comes bundled with npm, the Node.js Package Manager, which allows you to add dependencies to your application.

- In Node, you use the `require` function to require other modules (from packages, or from other files).

Why Are We Using Node.js?

Node.js has been chosen for this course for several reasons:

- It is particularly easy to setup
- For the sake of learning, it will be much easier on you to learn one programming language for both the frontend and backend rather than to learn one programming language for the frontend and another on the backend.
- There are many node packages available for you to use.
- Node.js promotes extremely modular code, making it easy to organize your code.
- Node.js scripts can be run via the command line, making it very easy to test your code without building out your actual web applications, but rather making and running other scripts. You can then phase your code into a web application in a more natural way.

What Is a Module?

Generally, a module is an individual unit that can be plugged into another system or codebase with relative ease. Modules do not have to be related, allowing you to write a system that allows many different things to interact with each other by writing code that glues them all together.

They are very flexible and allow you to organize your code very well!

In Node.js, you will be using modules everywhere. In our case, a module will be a specific object (think, an instance of a class) that has certain methods and data that you can access from other scripts. You will create your first module in lecture 2.

What Are Packages and NPM?

Node.js has a *massive* repository of published code that you can very easily pull into your assignments (where applicable) through the *node package manager* (npm).

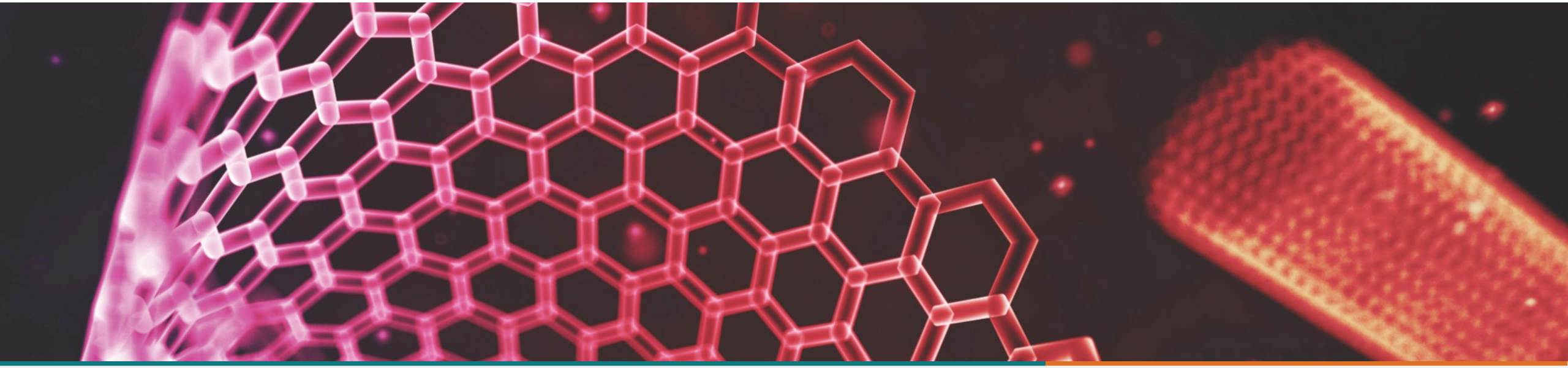
You will require the modules that your packages export, and use code that other people have created, tested, and tried. You will then use these packages to expand on your own applications and build out fully functional applications.

Testing Your Install

No programming environment is complete without our next step! Let's check that you configured and installed Node.js properly by running our first file, `hello.js`.
The file is available on the course's GitHub.

- You can do this with the command `node hello.js`
- If you look at this file, you will notice that it is only one line. You will be able to log messages to your terminal using the `console.log()` function.

Introduction to JavaScript and Syntax



Some Basic Facts About JavaScript

JavaScript is a loosely typed language, a concept that you may have seen before.

- Loose typing means that you don't strictly declare types for variables, and you can change the type of data that you store in each variable.
- Due to this loosely typed nature, you will need to check all inputs for
    1. If there is any input at all.
    2. The input is the correct data type.
    3. The input is in the correct range.

Some Basic Facts About JavaScript

If you write a function in JavaScript that is expecting two number inputs, JavaScript will let you pass non-numbers into the function. It will not produce an error but will give you undesired and unintended results.

For example, If you write a function that takes in two numbers and divides them, and one of the input parameters is a string or the denominator is 0, the function will not produce an error, but your results will not be the intended results.

JavaScript will also allow you to call a function that is expecting input parameters with no parameters at all. Therefore, type checking, checking if input is valid, checking that input is there etc.. If VERY crucial in JavaScript. You need to make sure you think of and cover any condition that will cause unintended results.

Some Basic Facts About JavaScript

Say you have the following code:

```
function divideTwoNumbers(num, den) {
  return num / den;
}
```

JavaScript will let you call the function with the following input parameters without error!

```
console.log(divideTwoNumbers(10, 0)); //returns Infinity
console.log(divideTwoNumbers('Patrick', 'Aiden')); //returns NaN
console.log(divideTwoNumbers()); //returns NaN
console.log(divideTwoNumbers(10)); //returns NaN
```

We will be covering error handing and input checking in lecture 2.

Some Basic Facts About JavaScript

There are five primitives currently, with a sixth (Symbol) on the way

- Boolean
- Number
- String
- Null
- Undefined

JavaScript also has Objects, which all non-primitives fall under

- Functions in JavaScript are types of Objects
- Objects are prototypical

Defining Variables

There are three keywords used to define variables in JavaScript:

Keyword	Scope	Explanation	Example
const	Block	Initializes a block scoped variable that cannot get overwritten. Most common used in this course.	const twelve = 11 + 1;
let	Block	Initializes a block scoped variable that can get overwritten.	let firstName = 'Patrick';
var	Functional	Initializes a variable that can get overwritten; **not used in course.**	var lastName = 'Hill'

Strings and Basic Syntax

Shortly, we will open up ***strings.js*** to take a look at basic syntax and some string operations. Variables are assigned with the ***var***, ***let*** and ***const*** keywords, and each line ends with a semicolon (the semicolon is not required in JavaScript, but it is good coding practice).

We will run ***strings.js*** the same way as you ran ***hello.js***; you'll see how to store variables, make comments, and do some basic string operations.

To see the number of functions that you can perform on strings which are displayed on the next slide.

Method	Description	Method	Description
charAt()	Returns the character at the specified index (position)	slice()	Extracts a part of a string and returns a new string
charCodeAt()	Returns the Unicode of the character at the specified index	split()	Splits a string into an array of substrings
concat()	Joins two or more strings, and returns a new joined strings	startsWith()	Checks whether a string begins with specified characters
endsWith()	Checks whether a string ends with specified string/characters	substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
fromCharCode()	Converts Unicode values to characters	substring()	Extracts the characters from a string, between two specified indices
includes()	Checks whether a string contains the specified string/characters	toLocaleLowerCase()	Converts a string to lowercase letters, according to the host's locale
indexOf()	Returns the position of the first found occurrence of a specified value in a string	toLocaleUpperCase()	Converts a string to uppercase letters, according to the host's locale
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string	toLowerCase()	Converts a string to lowercase letters
localeCompare()	Compares two strings in the current locale	toString()	Returns the value of a String object
match()	Searches a string for a match against a regular expression, and returns the matches	toUpperCase()	Converts a string to uppercase letters
repeat()	Returns a new string with a specified number of copies of an existing string	trim()	Removes whitespace from both ends of a string
replace()	Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced	valueOf()	Returns the primitive value of a String object
search()	Searches a string for a specified value, or regular expression, and returns the position of the match		

String Interpolation

String interpolation is a really useful programming language feature that allows you to inject variables directly into a string.

Until the release of ES6, string interpolation was not available in JavaScript. The lack of this feature has led to horrifying concatenated code that looks like this:

```
function crazyString(something, someone, somewhere) {
  return someone + ' was looking for ' + something + ' in the general vicinity of ' +   somewhere;
}
```

What's the wrong with this?

It is difficult to read and understand, especially when the function grow more complex. You also have to keep track of all the white space between each string, which is easy to overlook. Many developers have been forced to write JavaScript like this because JavaScript simply lacked string interpolation until now, but now that it is supported, we could instead do this for our return statement:

```
return `${someone} was looking for ${something} in the general vicinity of ${somewhere}`;
```

Looks much cleaner right? You would wrap your string in back-ticks, like this `` `string` `` instead of normal quotes and embed your variables with `${}`, like `` `string ${variable}` ``. This also maintains the format of the string maintaining line breaks and spaces. Think of `printf()` in Java

Booleans and Equality

JavaScript is a truthy language; that means that many things can evaluate as true or false. We will open ***boolean.js*** and take a look at some if statements.

Running ***boolean.js*** will show you all the things that are true and false in JavaScript.

We define a Boolean like so:

```
let myTrueBool = true;
let myFalseBool = false;
```

Numbers

Numbers in JavaScript follow the same pattern. You'll have access to several basic mathematical operators (*-+/) out of the box, and several more via the Math global variable.

You can read about the Math variable on the MDN

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

We will open ***numbers.js*** and look at how to handle numbers.

We define a numbers like so:

```
let num1 = 55;
let num2 = 101.5;
```

Functions

Functions are one of the most fundamental building blocks of JavaScript.

In JavaScript, variables can store functions; this makes it simple to do things like pass callbacks to functions. A callback is when a function takes a function as a parameter, to call it later. Functions can even return other functions!

We will take a look at ***`functions.js`*** to see how we use functions.

There are a few different ways we can write functions in JavaScript.

Here are the kind you will be using the most:

- Basic named functions
- Function expressions
- Arrow functions

Functions: Basic Named Functions

This is the basic way we define functions in JavaScript.

We use the `function` keyword followed by the name of the function with any input parameters it takes inside the parentheses.

```
function sayHello() {
  return 'Hello!';
}
```

```
function addTwoNumbers(num1, num2) {
  return num1 + num2;
}
```

We then call the function by name:

```
console.log(sayHello());
console.log(addTwoNumbers(1, 2));
```

Functions: Function Expressions

A function expression defines a named or anonymous function.

Although it is uncommon to give a function a name when using function expressions

An anonymous function is a function that has no name.

```
let addTwoNumbers = function uncommon(num1, num2) {
  return num1 + num2;
};

let sayHi = function() {
  return 'Hi there!';
};

let multiplyTwoNumbers = function(num1, num2) {
  return num1 * num2;
};
```

We then call the function by the assigned variable name:

```
console.log(sayHi());
console.log(multiplyTwoNumbers(5, 2));
console.log(addTwoNumbers(5, 2));
```

Functions: Arrow Functions

An Arrow Function Expression is a shorter syntax for writing function expressions.

There are a couple of ways to write arrow functions.

1. Like normal expression functions but omitting the `function` keyword and using `=>`
2. Omitting the `return` keyword
3. If our function has no parameters

```
let sayHi = () =>{
  return 'Hi there!';
};

let multiplyTwoNumbers = (num1, num2) =>{
  return num1 * num2;
};

const squared = value => value * value;

const noise = () => console.log('Pling')
```

We then call the function by the assigned variable name:

```
console.log(sayHi());
console.log(multiplyTwoNumbers(5, 2));
console.log(squared(5));
noise();
```

Function Default Input Parameter Values

You can set default values for input parameters in case they are not supplied:

```
function addTwoNumbers(num1 = 1, num2 = 1) {
  return num1 + num2;
}
console.log(addTwoNumbers(5, 5)); //5+5 = 10
console.log(addTwoNumbers()); //1+1 = 2
console.log(addTwoNumbers(5)); //5+1 = 6


let multiplyTwoNumbers = (num1 = 5, num2 = 2) => {
  return num1 * num2;
};
console.log(multiplyTwoNumbers(4, 2)); //4*2 = 8
console.log(multiplyTwoNumbers()); //5*2 = 10
console.log(multiplyTwoNumbers(3)); //3*2 = 6


let subtractTwoNumbers = function(num1 = 5, num2 = 2) {
  return num1 - num2;
};
console.log(subtractTwoNumbers(4, 2)); //4-2 = 2
console.log(subtractTwoNumbers()); //5-2 = 3
console.log(subtractTwoNumbers(3)); //3-2 = 1
```

Functional Scope In JavaScript

We often want to isolate our scope in JavaScript, particularly when we write browser-based JavaScript code in order to avoid conflicts between libraries.

While Node.js scripts isolate their variables between files, all top-level variables in a browser-environment become global variables, even across different files.

In JavaScript, scope is not defined by block unless using the keyword ***`let`*** to define a variable; when using ***`var`***, it is defined by the function you are in.

- https://hackernoon.com/understanding-javascript-scope-1d4a74adcdf5

We will take a look at functional scope in the lecture code in ***`functions.js`***.

Objects In JavaScript

Objects in JavaScript are incredibly dynamic and incredibly flexible, and highly readable.

You can create a basic type of Object, an "Object Literal", very simply, using the `{}` syntax when creating a variable. You can add properties to an object at any time.

In `objects.js` we will see very simple objects and some ways on dealing with objects.

```
let myobj = {
  name: 'Patrick Hill',
  education: [
    { Level: 'High School', Name: 'Bayside High School' },
    { Level: 'Undergraduate', Name: 'Baruch College' },
    { Level: 'Graduate', Name: 'Stevens Institute of Technology' }
  ],
  hobbies: [ 'Playing music', 'Family' ]
};
```

Object Destructuring

Say we have the following object that has a nested array of objects and an array within it:

```
let myobj = {
  name: 'Patrick Hill',
  education: [
    { Level: 'High School', Name: 'Bayside High School' },
    { Level: 'Undergraduate', Name: 'Baruch College' },
    { Level: 'Graduate', Name: 'Stevens Institute of Technology' }
  ],
  hobbies: [ 'Playing music', 'Family' ]
};
```

If we want to store the array of objects in education into a variable, and the hobbies into another variable. We can use the following syntax:

```
let {education, hobbies} = myobj
```

Then we can access JUST the education or hobbies from the object using the variable:

```
console.log(education);//Will print out the education from the object

console.log(hobbies);//Will print out the hobbies from the object
```

```
[
  { Level: 'High School', Name: 'Bayside High School' },
  { Level: 'Undergraduate', Name: 'Baruch College' },
  { Level: 'Graduate', Name: 'Stevens Institute of Technology' }
]
[ 'Playing music', 'Family' ]
```

More info on object destructuring: https://wesbos.com/destructuring-objects/

Arrays In JavaScript

Arrays in JavaScript also inherit from Objects and can store elements of any type.

- You can create an array using the very simple syntax of
  ```
  let myArray = [ 1, 2, 3 ];
  ```

Like an object, you do not have to populate anything in the initial array; you can manipulate it freely after creation. You can create an empty array like so:

```
let myArray = [];
```

Arrays have several built-in methods, which you can see by running through *arrays.js* and are also listed on the next slide.

Method	Description	Method	Description
concat()	Joins two or more arrays, and returns a copy of the joined arrays	lastIndexOf()	Search the array for an element, starting at the end, and returns its position
copyWithin()	Copies array elements within the array, to and from specified positions	map()	Creates a new array with the result of calling a function for each array element
entries()	Returns a key/value pair Array Iteration Object	pop()	Removes the last element of an array, and returns that element
every()	Checks if every element in an array pass a test	push()	Adds new elements to the end of an array, and returns the new length
fill()	Fill the elements in an array with a static value	reduce()	Reduce the values of an array to a single value (going left-to-right)
filter()	Creates a new array with every element in an array that pass a test	reduceRight()	Reduce the values of an array to a single value (going right-to-left)
find()	Returns the value of the first element in an array that pass a test	reverse()	Reverses the order of the elements in an array
findIndex()	Returns the index of the first element in an array that pass a test	shift()	Removes the first element of an array, and returns that element
forEach()	Calls a function for each array element	slice()	Selects a part of an array, and returns the new array
from()	Creates an array from an object	some()	Checks if any of the elements in an array pass a test
includes()	Check if an array contains the specified element	sort()	Sorts the elements of an array
indexOf()	Search the array for an element and returns its position	splice()	Adds/Removes elements from an array
isArray()	Checks whether an object is an array	toString()	Converts an array to a string, and returns the result
join()	Joins all elements of an array into a string	unshift()	Adds new elements to the beginning of an array, and returns the new length
keys()	Returns a Array Iteration Object, containing the keys of the original array	valueOf()	Returns the primitive value of an array

Conditionals: If...Else If...Else

There are a few ways to do If statement's in JavaScript:

```
let weather = 'Warm and Sunny';
if (weather === 'Warm and Sunny') {
  console.log('Wear something light, perhaps shorts.');
} else if (weather === 'Cold and Rainy') {
  console.log('Dress warm and bring an umbrella');
} else {
  console.log('I have no idea, go outside and check yourself..');
}
```

Single line conditional:

```
function welcome(name, age) {
  if (!name) throw 'Name is undefined';
  if (age < 13) throw 'COPPA laws prohibit use of this platform for anyone under 13, so scram!';
  return `Welcome ${name}, you are ${age} years old, you may use Pied Piper`;
}

console.log(welcome('Patrick', 44));
```

Ternary conditional:

```
let age = 26;
let beverage = age >= 21 ? 'Beer' : 'Juice';
console.log(beverage); // "Beer"
```

Loops

There are 5 different loops in JavaScript

- For loop
- For/In loop
- For/Of loop
- While loop
- Do/While Loop

Arrays also have a `forEach()` method that allows to loop through arrays easily.

For Loop

Loops through a block of code a number of times

```
for (i = 0; i < 10; i++) {
  console.log('The number is ' + i);
}
```

Output:

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

For/In Loop

for/in statement loops through the properties of an object:

```
let person = { fname: 'John', lname: 'Doe', age: 25 };

for (let x in person) {
  console.log(person[x]);
}
```

Output:

```
John
Doe
25
```

For/Of Loop

for/of statement loops through the values of an iterable objects

for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

```
let cars = [ 'BMW', 'Volvo', 'Mini' ];

for (let x of cars) {
  console.log(x);
}
```

Output:

BMW

Volvo

Mini

```
let txt = 'JavaScript';

for (let x of txt) {
  console.log(x)
}
```

Output:

J

a

v

a

S

c

r

i

p

t

While Loop

The while loop loops through a block of code as long as a specified condition is true.

```
let i = 0;
while (i < 10) {
  console.log('The number is ' + i);
  i++;
}
```

Output:

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

This loop may never execute depending on the condition

Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
let i = 0;
do {
  console.log('The number is ' + i);
  i++;
} while (i < 10);
```

Output:

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```

```
let i = 11;
do {
  console.log('The number is ' + i);
  i++;
} while (i < 10);
```

Output:

```
The number is 11
```

This loop will ALWAYS execute at least once

Questions?