

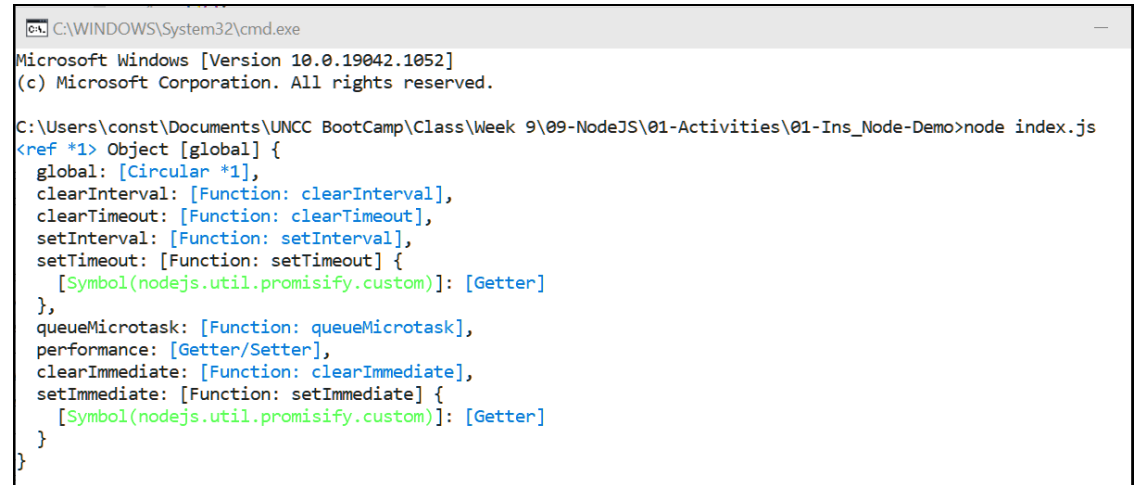
Node

[Install Node.js](#)

[Run in external terminal](#)

// Run this from the command line using 'node index.js'

```
(function () {  
  console.log(this);  
})();
```



```
C:\WINDOWS\System32\cmd.exe  
Microsoft Windows [Version 10.0.19042.1052]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\01-Ins_Node-Demo>node index.js  
<ref *1> Object [global] {  
  global: [Circular *1],  
  clearInterval: [Function: clearInterval],  
  clearTimeout: [Function: clearTimeout],  
  setInterval: [Function: setInterval],  
  setTimeout: [Function: setTimeout] {  
    [Symbol(nodejs.util.promisify.custom)]: [Getter]  
  },  
  queueMicrotask: [Function: queueMicrotask],  
  performance: [Getter/Setter],  
  clearImmediate: [Function: clearImmediate],  
  setImmediate: [Function: setImmediate] {  
    [Symbol(nodejs.util.promisify.custom)]: [Getter]  
  }  
}
```

Arrow Function

```
// All of the `createGreeting` functions are equivalent
var createGreeting = function(message, name) {
  return message + ", " + name + "!";
};

// We can safely swap out function expressions with arrow functions
// most of the time
var createGreeting = (message, name) => {
  return message + ", " + name + "!";
};

// If the arrow function body contains only one expression, we can
// omit the curly braces and auto return it
var createGreeting = (message, name) => message + ", " + name + "!";

// If an arrow function only has one parameter, we can omit the parens ()
// around the single parameter
var greet = greeting => console.log(greeting);

// We call arrow functions the same way as we call regular functions
var greeting = createGreeting("Hello", "Angie");

// Logs "Hello, Angie!";
greet(greeting);
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\03-Ins_Arrow-Function>node 01-syntax.js
Hello, Angie!

C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\03-Ins_Arrow-Function>node 02-context.js
Hodor is thinking...
Hodor is thinking...
undefined!
Hodor!

C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\03-Ins_Arrow-Function>node 03-property-methods.js
undefined
The dog's tag reads: undefined.
```

```
// Depending on the environment `setTimeout` is called in, it may refer to one
// of two objects
// In the browser, `setTimeout` is a property of the `window` object
// In node, it belongs to a special "Timeout" object
var person = {
  name: "Hodor",
  saySomething: function() {
    console.log(this.name + " is thinking...");
    setTimeout(function() {
      console.log(this.name + "!");
    }, 100);
  }
};
person.saySomething(); // prints "Hodor is thinking..."
// prints "undefined!" 100ms later

// Arrow functions bind the `this` keyword to the object it's created inside of
// i.e. whatever `this` is where it's created
var person = {
  name: "Hodor",
  saySomething: function() {
    console.log(this.name + " is thinking...");
    setTimeout(() => console.log(this.name + "!"), 100);
  }
};
person.saySomething(); // "Prints Hodor is thinking..."
// prints `Hodor!` 100ms later
```

// Avoid using arrow functions for object methods

```
var dog = {
  name: "Lassie",
  sound: "Woof!",
  makeSound: () => console.log(this.sound),
  readTag: () => console.log("The dog's tag reads: " + this.name + ".")
};
```

// Prints `undefined`
dog.makeSound();

// Prints `The dog's tag reads: undefined.`
dog.readTag();

// In the makeSound and readTag methods, `this` doesn't refer to `dog`
// If this code run in node, `this` refers to `module.exports` (the object containing
// all the exports in this file)
// If this code was run in the browser, `this` would refer to the window

Const

&

Let

```
// // 1. const can be used for values which we will not reassign
const age = 25;
age++; // TypeError: Assignment to constant variable.
// =====
const name = "";
name = "Cherie"; // TypeError: Assignment to constant variable.
// =====
// 2. const doesn't mean `constant value`, instead means
//     `constant reference`
// Unlike primitive data types, objects and arrays are passed by
//     reference, rather than passed by value
const beatles = ["John", "Paul", "Ringo"];
beatles.push("George");
// This works because by updating an array's contents,
//     we aren't changing the reference to the underlying array
console.log(beatles); // Prints `["John", "Paul", "Ringo", "George"]`
console.log(beatles);
const person = { name: "Brianna", age: 11 };
person.age++;
person.favoriteMovie = "Spider-Man";
person.name = "Carla";
console.log(person); // Prints `
{ name: 'Carla', age: 12, favoriteMovie: 'Spider-Man' }`
// =====
// 3. While we can MODIFY arrays and objects that are using `const`,
//     we can't reassign them
const item = {
  id: 23,
  title: "Underwater Basket-Weaving DVD",
  price: "$17.99",
};
item.price = "$1.99";
console.log(item);
item = {
  id: 11,
  title: "Underwater Basket-Weaving Shoes",
  price: "$101.43",
}; // TypeError: Assignment to constant variable.
const ninjaTurtles = [];
// The same rules apply to arrays, we can MODIFY them,
//     but not completely reassign them
const ninjaTurtles = ["Michaelangelo", "Leonardo", "Raphael",
  "Donatello"]; // TypeError: Assignment to constant variable.
ninjaTurtles.enemies = ["shredder", "krang"];
console.log(ninjaTurtles);
ninjaTurtles.enemies = ["footsoldiers", "negazone"];
console.log(ninjaTurtles);
```

```
// 1. When using var, our counter exists after a for-loop is done
for (var i = 0; i < 5; i++) {
  console.log(i);
}
console.log(i); // Prints 5
// When using let, our counter is not defined outside of the for-loop block
let x = 42;
for (let j = 0; j < 5; j++) {
  console.log(j);
  console.log(x);
}
console.log(j); // ReferenceError: j is not defined
let j = 42;
console.log(j); // prints 42
// =====
// 2. When using while loops, any values we create inside
//     exist outside of the while-loop block
var count = 0;
while (count < 5) {
  var tripled = count * 3;
  count++;
}
console.log(tripled); // Prints 12
// =====
// 3. When using let, values defined inside of the
//     while-loop block don't exist outside of it
let c = 0;
while (c < 5) {
  let quadrupled = c * 3;
  c++;
}
// console.log(quadrupled); // ReferenceError: quadrupled is not defined
// =====
// 4. When writing conditionals, values defined inside the conditional block
//     exist outside of it
if (true) {
  var favoriteColor = "red";
}
console.log(favoriteColor); // Prints `red`
// When using let, values defined inside of a conditional block don't exist outside
let favoriteFood;
if (true) {
  favoriteFood = "pizza";
}
// This works since favoriteColor is not defined inside of a block
console.log(favoriteFood);
// Prints `pizza`
```

Functional Loops

```
const moviePatrons = [
  { name: "Tom", age: 16 },
  { name: "Ashley", age: 31 },
  { name: "Sarah", age: 18 },
  { name: "Alvin", age: 22 },
  { name: "Cherie", age: 14 },
  { name: "Malcolm", age: 15 }
];

// 1.
// forEach is a functional way of iterating through an array
// without a for-loop
moviePatrons.forEach(patron => console.log(patron.age));

// 2.
// Filter returns a new array containing only elements
// whose callback returns a truthy value
const canWatchRatedR = moviePatrons.filter(function(patron) {
  return patron.age > 17;
});
console.log(canWatchRatedR);

// 3.
// Map returns a brand new array the same length as the first.
// Each element is passed into the callback.
// Whatever is returned from the callback at each iteration is
// what goes into that index of the new array
const cardedMoviePatrons = moviePatrons.map(patron => {
  // Copy the object being iterated over
  const pObj = { ...patron };
  // Do everything else the same
  if (pObj.age >= 17) {
    pObj.canWatchRatedR = true;
  } else {
    pObj.canWatchRatedR = false;
  }
  // Be sure to return the new obj, not the parameter
  return pObj;
});

console.log("Movie Patrons: ")
console.log(moviePatrons);
console.log("\nCarded Movie Patrons: ");
console.log(cardedMoviePatrons);
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\07-Ins_Functional-Loops>node index.js
16
31
18
22
14
15
[
  { name: 'Ashley', age: 31 },
  { name: 'Sarah', age: 18 },
  { name: 'Alvin', age: 22 }
]
Movie Patrons:
[
  { name: 'Tom', age: 16 },
  { name: 'Ashley', age: 31 },
  { name: 'Sarah', age: 18 },
  { name: 'Alvin', age: 22 },
  { name: 'Cherie', age: 14 },
  { name: 'Malcolm', age: 15 }
]
Carded Movie Patrons:
[
  { name: 'Tom', age: 16, canWatchRatedR: false },
  { name: 'Ashley', age: 31, canWatchRatedR: true },
  { name: 'Sarah', age: 18, canWatchRatedR: true },
  { name: 'Alvin', age: 22, canWatchRatedR: true },
  { name: 'Cherie', age: 14, canWatchRatedR: false },
  { name: 'Malcolm', age: 15, canWatchRatedR: false }
]
```

Template-Literals

```
const arya = {  
  first: "Arya",  
  last: "Stark",  
  origin: "Winterfell",  
  allegiance: "House Stark"  
};
```

```
const greeting = `My name is ${arya.first}!  
I am loyal to ${arya.allegiance}.`;
```

```
console.log(greeting); // prints  
// My name is Arya!  
// I am loyal to House Stark.
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS  
\01-Activities\09-Ins_Template-Literals>node index.js  
My name is Arya!  
I am loyal to House Stark.
```

// music should be an object with title, artist, and album properties

```
const music = {  
  // code here  
  title: "here to stay",  
  artist: "Korn",  
  album: "untouchables",  
};
```

// Write code between the backticks tags to output the data from the music object above.

```
const songSnippet = `Dude, you gotta listen to ${music.title} by ${music.artist} from their fourth release ${music.album}.`;`;  
console.log(songSnippet);
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\10-Stu_Template-Literals  
\Unsolved>node index.js  
Dude, you gotta listen to here to stay by Korn from their fourth release untouchables.
```

Process Argv

// returns an array of command line arguments

```
console.log(process.argv);
```

// arguments passed from the command line are accessed by index

```
console.log(process.argv[2]);
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\11-Ins_process-argv>node index.js 3 4
[
  'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\const\\Documents\\UNCC BootCamp\\Class\\Week 9\\09-NodeJS\\01-Activities\\11-Ins_process-argv\\index.js',
  '3',
  '4'
]
3
```

```
console.log(
  process.argv[3] === process.argv[2] ? "clear as mud" : "do it again Mahesh"
);
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\12-Stu_Parameter-Check>node index.js 3 4
do it again Mahesh
```

Read-Write File

```
// fs is a Node standard library package for reading and writing files
const fs = require("fs");
```

```
// Return the contents of 'data.csv' as a string in the variable "data"
// "utf8" encodes the raw buffer data in human-readable format
fs.readFile("data.csv", "utf8", (error, data) =>
  error ? console.error(error) : console.log(data)
);
```

```
// Uncomment this next function to write to the file with anything you pass in as process.argv[2]
fs.writeFile("log.txt", process.argv[2], (err) =>
  err ? console.error(err) : console.log("Success!")
);
```

Data.csv:

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\13-Ins_Read-Write-File>node index.js 3
10,123,76,5003,9,0.127,40,-1,2^4
Success!
```

Log.txt:

3

Modularization

```
const pie = 'apple';
const predictable = () => 1;
```

```
// module.exports is an object we use to store variables or methods
module.exports = {
  pie,
  predictable,
};
```

```
const badmath = require('./badmath.js');
console.log(badmath.pie);
console.log(badmath.predictable());
```

```
// TODO: Import `maths.js`
const maths = require("./maths.js");
```

```
// TODO: Capture the values passed from the command line into
       these three variables: `operation`, `numOne` and `numTwo`
let operation = process.argv[2];
let numOne = parseInt(process.argv[3]);
let numTwo = parseInt(process.argv[4]);
```

```
// TODO: Create a `switch` statement that accepts an `operation` parameter
// and each `case` uses the corresponding `maths` method
// to perform each math operation on the two numbers, `numOne` and `numTwo`
switch (operation) {
  case "sum":
    console.log(maths.sum(numOne, numTwo));
    break;
  case "difference":
    console.log(maths.difference(numOne, numTwo));
    break;
  case "product":
    console.log(maths.product(numOne, numTwo));
    break;
  case "quotient":
    console.log(maths.quotient(numOne, numTwo));
    break;
}
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\
01-Activities\15-Ins_Modularization>node index.js
apple
1
```

```
module.exports = {
  sum: (a, b) => a + b,
  difference: (a, b) => a - b,
  product: (a, b) => a * b,
  quotient: (a, b) => a / b,
};
```

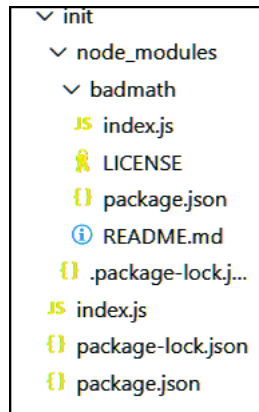
```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\
16-Stu_maths\Unsolved>node index.js sum 3 4
7

C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\
16-Stu_maths\Unsolved>node index.js product 3 4
12

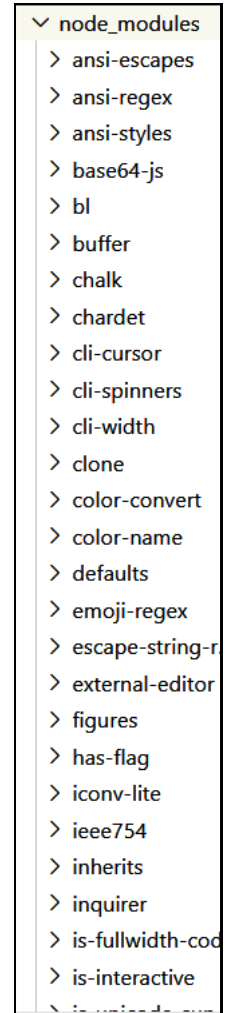
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\
16-Stu_maths\Unsolved>node index.js difference 3 4
-1
```


NPM

```
{
  "name": "12-ins_npm-init",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "dependencies": {
    "badmath": "^1.0.1"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```



```
{
  "name": "package",
  "version": "1.0.0",
  "description": "for unsolved",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "inquirer"
  ],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "inquirer": "^8.1.1"
  }
}
```



Inquirer

```
const inquirer = require('inquirer');
inquirer
  .prompt([
    {
      type: 'input',
      message: 'What is your user name?',
      name: 'username',
    },
    {
      type: 'password',
      message: 'What is your password?',
      name: 'password',
    },
    {
      type: 'password',
      message: 'Re-enter password to confirm:',
      name: 'confirm',
    },
  ])
  .then((response) =>
    response.confirm === response.password
      ? console.log('Success!')
      : console.log('You forgot your password?!')
  );
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\
01-Activities\20-Stu_Inquirer-Users>node index.js
? What is your name? ryan
? What languages do you know? CSS, JavaScript
? What is your preferred method of communication? telekinesis
Success!
```

```
Class > Week 9 > 09-NodeJS > 01-Activities > 20-Stu_Inquirer-Users > {} ryan.json > ...
1  {}
2    "name": "ryan",
3    "stack": [
4      "CSS",
5      "JavaScript"
6    ],
7    "contact": "telekinesis"
8  }
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\
01-Activities\19-Ins_Inquirer-Demo>node index.js
? What is your user name? ryan
? What is your password? [hidden]
? Re-enter password to confirm: [hidden]
Success!
```

```
const inquirer = require('inquirer');
const fs = require('fs');
inquirer
  .prompt([
    {
      type: 'input',
      name: 'name',
      message: 'What is your name?',
    },
    {
      type: 'checkbox',
      message: 'What languages do you know?',
      name: 'stack',
      choices: ['HTML', 'CSS', 'JavaScript', 'MySQL'],
    },
    {
      type: 'list',
      message: 'What is your preferred method of communication?',
      name: 'contact',
      choices: ['email', 'phone', 'telekinesis'],
    },
  ])
  .then((data) => {
    const filename = `${data.name.toLowerCase().split(' ').join('')}.json`;
    fs.writeFile(filename, JSON.stringify(data, null, 't'), (err) =>
      err ? console.log(err) : console.log('Success!')
    );
  });
```

For - Of

// 1. Using arrays

```
const songs = ['Bad Guy', 'The Wheels on the Bus', 'Friday'];
for (const value of songs) console.log(value);
```

```
console.log("\n===== \n");
```

// 2. Using multidimensional arrays

```
const moreSongs = [
  ['Bad Guy', 1],
  ['The Wheels on the Bus', 2],
  ['Friday', 3],
];
for (const [key, value] of moreSongs) {
  console.log(`${key}'s chart position is ${value}`);
}
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-
width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <link href="style.css" rel="stylesheet" type="text/css" />
    <title>For...of</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h2>Spotify Global Top 50</h2>
    <ul id="songs">
      <li>1. Fast by Sueco the Child</li>
      <li>2. Old Town Road - Remix by Lil Nas X</li>
      <li>3. Wrong by Luh Kel</li>
      <li>4. Piece Of Your Heart by MEDUZA</li>
      <li>5. Kill This Love by BLACKPINK</li>
      <li>6 bad guy by Billie Eilish</li>
      <li>7. Verte Ir by DJ Luian</li>
      <li>8. Beef FloMix by Flo Milli</li>
      <li>9. Avengers: Endgame by Jared Moreno Luna</li>
      <li>10. Old Town Road by Lil Nas X</li>
    </ul>
    <script type="text/javascript" src="index.js"></script>
  </body>
</html>
```

```
const elements = document.querySelectorAll("li");
```

```
// for (const song of songs) {
//   song.classList.add("red");
// }
```

```
for (const song of elements) {
  const [hit, artist] = song.textContent.split(" by ");
```

```
  song.textContent = "";
```

```
  const liElem = document.createElement("li");
```

```
  const hitElem = document.createElement("span");
  hitElem.textContent = hit;
```

```
  const artistElem = document.createElement("span");
  artistElem.textContent = artist;
```

```
  const byElem = document.createElement("span");
  byElem.textContent = " by ";
```

```
  hitElem.classList.add("red");
  artistElem.classList.add("green");
```

```
  song.appendChild(hitElem);
  song.appendChild(byElem);
  song.appendChild(artistElem);
}
```

```
.red {
  color: red;
}
.green {
  color: green;
}
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\
09-NodeJS\01-Activities\21-Ins_for-of>node index.js
```

```
Bad Guy
```

```
The Wheels on the Bus
```

```
Friday
```

```
=====
```

```
Bad Guy's chart position is 1
```

```
The Wheels on the Bus's chart position is 2
```

```
Friday's chart position is 3
```

Spotify Global Top 50

- 1. Fast by Sueco the Child
- 2. Old Town Road - Remix by Lil Nas X
- 3. Wrong by Luh Kel
- 4. Piece Of Your Heart by MEDUZA
- 5. Kill This Love by BLACKPINK
- 6 bad guy by Billie Eilish
- 7. Verte Ir by DJ Luian
- 8. Beef FloMix by Flo Milli
- 9. Avengers: Endgame by Jared Moreno Luna
- 10. Old Town Road by Lil Nas X

Rest & Spread

```
// without rest
function add(x, y) {
  return x + y;
}
console.log(add(1, 2, 3, 4, 5)); // => 3
```

```
// 1. rest
function add(...nums) {
  let sum = 0;
  for (let num of nums) sum += num;
  return sum;
}
add(1); // => 1
add(3, 3); // => 6
add(1, 1, 4, 5); // => 11
```

```
// 2.
function howManyArgs(...args) {
  return `You passed ${args.length} arguments.`; // point out the template literal
}
console.log(howManyArgs(0, 1)); // You have passed 2 arguments.
console.log(howManyArgs("argument!", null, ["one", 2, "three"], 4));
// You have passed 4 arguments.
```

```
// 1. spread
const dragons = ["Drogon", "Viserion", "Rhaegal"];
const weapons = ["dragonglass", ...dragons, "wildfire"]; // notice the spread operator ...dragons
console.log(weapons); // prints ["dragonglass", "Drogon", "Viserion", "Rhaegal", "wildfire"]
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\
01-Activities\23-Ins_Rest-and-Spread>node index.js
15
You passed 2 arguments.
You passed 4 arguments.
[ 'dragonglass', 'Drogon', 'Viserion', 'Rhaegal', 'wildfire' ]
```

```
// Exercise 1
const songs = ['Creep', 'Everlong', 'Bulls On Parade', 'Song 2', 'What I Got'];
```

// TODO: Which operator is being used here?

```
const newSongs = [...songs];
```

// TODO: What do you expect to be logged in the console?

```
console.log(newSongs);
```

// Exercise 2

```
const addition = (x, y, z) => {
  const array = [x, y, z];
```

// TODO: What does the reduce() method do?

```
  return array.reduce((a, b) => a + b, 0);
};
```

// TODO: What do you expect to be logged in the console?

```
console.log(addition(1, 2, 3));
```

// TODO: What is this syntax that is being used as the parameter?

```
const additionSpread = (...array) => {
  return array.reduce((a, b) => a + b, 0);
};
```

// TODO: What do you expect to be logged in the console?

```
console.log(additionSpread(1, 2, 3));
```

// TODO: What do you expect to be logged in the console?

```
console.log(additionSpread(1, 2, 3, 4, 100));
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities\
24-Stu_Rest-and-Spread\Unsolved>node index.js
6
6
110
[ 'Creep', 'Everlong', 'Bulls On Parade', 'Song 2', 'What I Got' ]
```

Object Destructuring

```
const arya = {
  name: 'Arya Stark',
  parents: ['Eddard Stark', 'Catelyn Stark'],
};
const jaime = {
  name: 'Jaime Lannister',
  parents: ['Tywin Lannister', 'Joanna Lannister'],
};
// In the past, if we wanted to pull off an object's property we'd have to do
// something like this:
const aryaName = arya.name;
const aryaParents = arya.parents;
console.log(aryaName); // prints "Arya Stark"
console.log(aryaParents); // prints ["Eddard Stark", "Catelyn Stark"]

// Now with ES6 object destructuring syntax, we can do this:
const { name, parents } = arya;
console.log(name); // prints "Jaime Lannister"
console.log(parents); // prints ["Tywin Lannister", "Joanna Lannister"]

// We can also rename our destructured properties like so:
const { name: jaimeName } = jaime;
console.log(jaimeName); // prints "Jaime Lannister"

// We can also destructure parameters using the same feature.
// e.g. previously we might have done something like this:
const logCharacter = (character) =>
  console.log(
    `${character.name}'s parents are: ${character.parents[0]} and ${character.parents
[1]}`
  );
logCharacter(arya);

// But now we can do this:
const betterLogCharacter = ({ name, parents }) =>
  console.log(`${name}'s parents are: ${parents[0]} and ${parents[1]}`);
betterLogCharacter(jaime);
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\01-Activities
\25-Ins_Obj-Destructuring>node index.js
Arya Stark
[ 'Eddard Stark', 'Catelyn Stark' ]
Arya Stark
[ 'Eddard Stark', 'Catelyn Stark' ]
Jaime Lannister
Arya Stark's parents are: Eddard Stark and Catelyn Stark.
Jaime Lannister's parents are: Tywin Lannister and Joanna Lannister.
```

```
// 1. Object
const nodejs = {
  name: "Node.js",
  type: "JavaScript runtime environment",
};

// TODO: Destructure the object 'nodejs'
const { name, type } = nodejs;

console.log(name); // <= Node.js
console.log(type); // <= JavaScript runtime environment

// 2. Nested Object
const js = {
  name: "JavaScript",
  type: "programming language",
  version: "ES6",
  tools: {
    frameworks: {
      framework1: "AngularJS",
      framework2: "Vue.js",
    },
    libraries: {
      library1: "jQuery",
      library2: "React",
    },
  },
};

// TODO: Destructure the nested object 'js'
const { framework1, framework2 } = js.tools.frameworks;

console.log(framework1); // <= AngularJS
console.log(framework2); // <= Vue.js

// 3. Arrays
const languages = ["HTML", "CSS", "JavaScript"];

// TODO: Destructure the array 'languages'
const [markup, style, scripting] = languages;

console.log(markup, style, scripting); // <= HTML CSS JavaScript
console.log(markup); // <= HTML
```

```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 9\09-NodeJS\
01-Activities\26-Stu_Obj-Destructuring\Unsolved>node index.js
Node.js
JavaScript runtime environment
AngularJS
Vue.js
HTML CSS JavaScript
HTML
```

Git Fork

Now that you have become familiar with Git and Github, you may find yourself wanting to contribute to an open source project. You may also want to use a project as a starting point for your own, in the case of something like a boilerplate application. Luckily, there is a feature in Github and many other version control systems called forking.

Forking a repository makes an exact copy another project that resides on your personal account. This acts as a connection between your own repo and the owner's repo. Additionally, this allows the owner to protect their own code by preventing users from pushing to it directly but still giving developers a way to contribute in a non-intrusive way. Forking is very common in open-source software development.

In this activity we will be using a test repository to help you grasp the fork and pull request workflow:

Fork and Clone

1. To fork this project, click the "fork" button at the top of the repository
2. Choose your personal account when asked 'Where should we fork?'
3. After a few seconds you will be taken to *your* copy of the repository
4. Visit the fork that you just created on GitHub can copy the URL of your fork.
5. Clone your fork of the repository to your local machine and move into that directory:

```
git clone git@github.com:<YOUR_GITHUB_USERNAME>/forking-tutorial.git && cd forking-tutorial
```

6. Run the following command to see that the remote URL for the repository is set to your own personal Github

```
git remote -v
```

Contribute and Pull Request

1. Create a branch of your own

```
git checkout -b <branch name>
```

2. Make some additions or changes to the readme file and stage and commit your work.

```
git add readme.md  
git commit -m "make it better"
```

3. Push your changes to the remote branch

```
git push origin <branch name>
```

4. Back on the forked repository page, you should see a yellow bar at the top with a button to "Compare and Pull Request". Click that button.

- * After clicking that button, take some time to enter in a good description to help the owner of the project decide what it is we are trying to contribute.
- * We will see a list of commit messages and that our branch is "ahead" of the master branch.

5. Click "Create Pull Request". The owner of the project will get an email notification that some changes have been made and that a pull request is ready for their review.

- * The owner can approve or reject the pull request and optionally add some additional comments for your review. While this is a simple edit we are making to the `readme.md` file, you can imagine a situation where your code would require some additional review.