What does the **map()** method return? **B) A new array with transformed elements** What is the output of this code? **const nums = [1, 2, 3]; const doubled = nums.map(n => n * 2); console.log(doubled); B) [2, 4, 6]** Which of the following correctly uses **map()** to create an array of squares from **nums**? **const nums = [1, 2, 3]; A) const squares = nums.map(n => n ** 2);** What does the callback function in **map()** receive as arguments? **C) Current element, index, and the original array** What does the **filter()** method return? **A) A new array with elements that pass the condition** Which code returns only even numbers from **nums**? **const nums = [1, 2, 3, 4]; B) nums.filter(n => n % 2 === 0)** What happens if no elements match the filter condition? **C) Returns an empty array** Which of the following creates an array of strings longer than 3 characters? **const words = ["cat", "horse", "dog", "elephant"]; B) words.filter(word => word.length > 3)** What does **forEach()** return? **C) Undefined** Which statement correctly uses **forEach** to log each number in **nums**? **const nums = [1, 2, 3]; A) nums.forEach(n => console.log(n));** Can **forEach** be used to modify the original array? **A) Yes** What does **fetch()** return? **A) A promise** What keyword is used to wait for a promise to resolve?**C) await** What is wrong with this code? **async function getData() { let data = fetch("https://api.example.com/data"); console.log(data);} B) fetch() is missing await** What symbol is used to wrap template literals? **C)** `` What is the output of this code? **const name = "Alice"; console.log(`Hello, ${name}!`); A) Hello, Alice!** Which method selects a single element by its ID? Circle all that apply. **A) document.querySelector("#id") B) document.getElementById("id")** How do you add a paragraph to an element with an ID of **container**?**const container = document.getElementById("container"); B) container.innerHTML = " Hello ";** Given this array of objects, how would you iterate over it and log each item's title? **const products = [{ id: 1, title: "Laptop", price: 1200 }, { id: 2, title: "Phone", price: 800 }, { id: 3, title: "Tablet", price: 500 }]; C) products.forEach(product => console.log(product.title** What will the following code log? **const person = { name: "Alice", age: 25, city: "New York" }; const { name, age } = person; console.log(name, age); A) Alice 2** How can you extract the price from this object using destructuring? **const product = { id: 1, name: "Laptop", price: 1200 }; B) const { price } = product;** What will the following code output? **const nums = [1, 2, 3, 4]; const [first, , third] = nums; console.log(first, third); A) 1 2** What does the spread operator (...) do in the following code? **const nums = [1, 2, 3]; const moreNums = [...nums, 4, 5]; C) Copies the contents of nums into moreNums** What will this code log to the console? const obj1 = { a: 1, b: 2 }; const obj2 = { b: 3, c: 4 }; const merged = { ...obj1, ...obj2 }; console.log(merged); **B) { a: 1, b: 3, c: 4 }** What will this code log? const person = { name: "Alice", age: 25 }; const clone = { ...person, age: 30 }; console.log(clone); **B) { name: "Alice", age: 30 }**How can you extract **name** and collect the rest of the properties into rest? const person = { name: "Alice", age: 25, city: "New York" }; **A) const { name, ...rest } = person;** What will the following code output? const nums = [1, 2, 3]; const [first, ...rest] = nums; console.log(rest); **A) [2, 3]** What does this code do? const arr1 = [1, 2, 3]; const arr2 = [4, 5]; const combined = [...arr1, ...arr2]; console.log(combined);
C)Concatenates arr1 and arr2 into a new array

Higher Order Methods: Actually takes 3 arguments: the value (required), index of that value (optional), original array (optional)

.forEach: Visits each item and performs the function on it. Returns undefined. Undefined can modify the og array

.map: Visits each item, performs the transformation function. Returns an array of the transformed values

.filter: Visits each node and performs the T/F evaluation with it. If true it is added to the array that is returned. Returns an array of only the values that pass the test.

.reduce: Combines all the elements into one value. You pass in the accumulator (which will also be the return value) and the current value, and apply whatever function somehow reduces them (ex. comparison or addition)

.find: Returns the first element that meets the T/F condition.

Pure function- Don't modify the data coming in. If given identical input will always produce the same output

---

Weird JavaScript Quirks: In JS if you give the function too many/too few arguments, it'll still run, either ignoring the extras or setting the missing values to null.

Spread (…) : Destructures arrays/objects and places them as individual elements in the new object/array.

WITH SPREAD: nums=[1,2,3]; moreNums=[…nums,4,5]; is the same as [1,2,3,4,5];	WITHOUT SPREAD: nums=[1,2,3]; moreNums=[nums,4,5]; is the same as [[1,2,3],4,5];
obj1={a:1, b:2}; obj2={c:3}; merged={…obj1, …obj2}; is the same as {a:1, b:2, c:3}	*Conflicts in names (ex. both had a b data field) will result in the value being overwritten.*

Also useful to do for function parameters Ex. sum(…nums) will pass all elements in nums as individual parameters. Very Useful for copying objects/arrays. If you just did arr1=arr2 it would make a shallow copy, but if you do arr1=[…arr2] it'll make a deep copy.

Object Destructuring: Grab specific data fields from an object using their names. Ex. let { name, age } = person (person is an object with name and age fields) will get the values from person and assign them to variables named name and age. Ex. let { n: name, howOld: age } = person; will get the info but assign them to variables called n and howOld. It's tricky but this works with nested objects too (another set of curly braces).

Array Destructuring: Ex. let [a,b] = nums; will get the first two values in nums and assign them to variables named a and b. ***Rest Operator:*** Used in object destructuring to save all other values into one object?Array? Ex. let { name, …rest} = person; Will save the name value into the name variable, and save the all other data fields that weren't previously extracted into the rest variable.

URLs: Ex. http: //www.example.com /home/search? /item=vw+beetle (protocol domain path parameters)

API Authentication: Each user is assigned a token that is passed back and forth when doing fetch requests. Async functions run on the side, the program won't wait until the command finished to continue. Use the key word "await" if you want a command to wait until the async promise is fulfilled.

Client Request: Contains a header with metadata, URL, method (ex. POST), and authorization, and a body with the payload	Server Response: Contains a header with the success status and type, and a body with the desired info. Code 1xx is waiting, 2xx is a success, 3xx-5xx failure

Fetch: Returns a promise. Fetch Methods: GET, PUT, POST, DELETE (read, update, create, delete)

Promise: Gives you a "ticket" that the info you requested will come, when the promise resolves, you'll get the data.

Ex. async function getUserData() { //Async means that it will be an HTTP request

```
 const response = await fetch("https://photo-app-secured.herokuapp.com/api/profile/", {   //URL
   method: "GET",            //Header Metadata, specify the method, the content type and
   headers: {                                                  //authorization if needed
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${token}`
   }
 });       //Will return a promise, (stored in response)
 const data = await response.json(); //Then we turn that response into a JSON object we can use
}                                                                  //Only necessary when we are receiving data
async function functionName(bodyInfo1, bodyInfo2) {
 const bodyData = {              //Needed for adding new info (ex. POST request)
   "post_id": `${bodyInfo1}`,
   "text": `${bodyInfo2}`
 }; const response = await fetch("https://ex.url", {
   method: "POST",               //or DELETE, GET, PUT   headers: {
     'Content-Type': 'application/json',  //when we want/are sending JSON objects(?)
     'Authorization': `Bearer ${token}`},
   body: JSON.stringify(postData)        //This converts the info (object) into a JSON info usable by the API //Only necessary
when sending in data
```