Slide Note

Customers who placed a bigger order amount are premium customers. These types of customers are offered incentives due to their special status.

The customer's order amount for a month are stored in an array.

Premium Customers

Imagine a scenario where an online shopping website classifies customers who have placed a higher number of orders as its "premium customers". They offer them certain incentives due to their special status.

The customer's order amount for a month is stored in an array. Now, the order amount must be sorted in descending order to identify the top premium customers.

How can you sort the array values to achieve this?

Slide Note

Display Products

Imagine another scenario where the same online website wants to display their products by price range so that the customer can purchase right away by just clicking.

All the products are stored in an array that needs to be filtered based on different prices.

How can you manipulate the array of products?

Menu

Slide Note

Employee Salary

Imagine a scenario where a training organization has raised the salaries of its employees by 10%. The organization wants to calculate the new incremented salaries of all employees.

The employee salaries are stored in an array. You need to update each salary value in the array by 10%.

How can you manipulate each array value?

Menu

00:03:00:00:42 03/ 42

Slide Note

Employee Service Excellence Award

Long-serving employees are considered highly trustworthy and still form a critical part of any organizational workforce. So, the same organization wanted to help the employees who had worked for more than 20 years.

You need to find the long-serving employees based on their joining dates stored in an array.

How can you search for employees from the array?

Does JavaScript provide any built-in methods to perform these array operations?

Slide Note

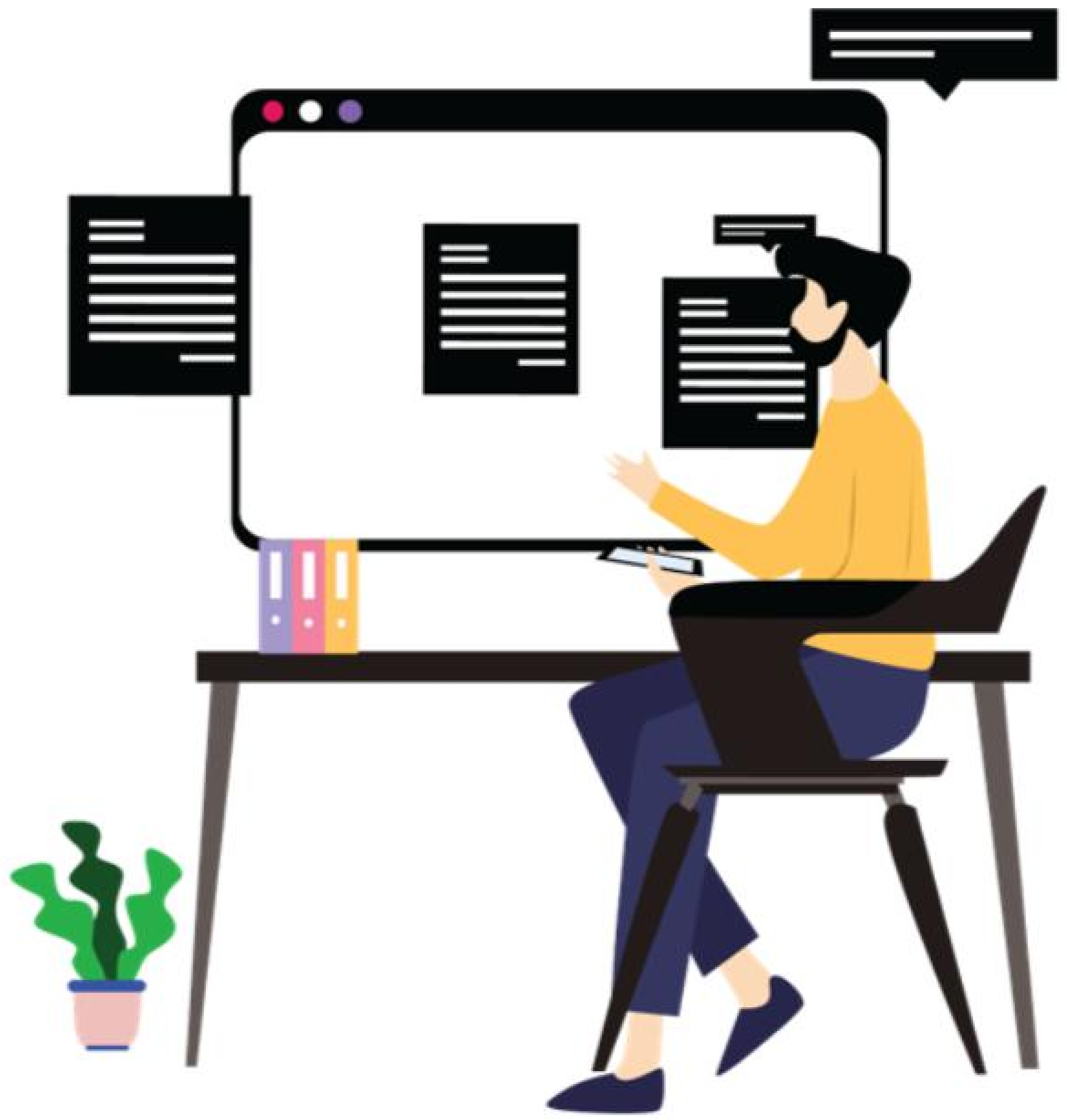
Can we have functions perform these array operations to reuse across different applications?

Menu

00:05 00:00:42 05/ 42

Slide Note

Perform Common Array Operations Using JavaScript Array Methods



Menu

00:06 00:00:42

06/ 42

Slide Note

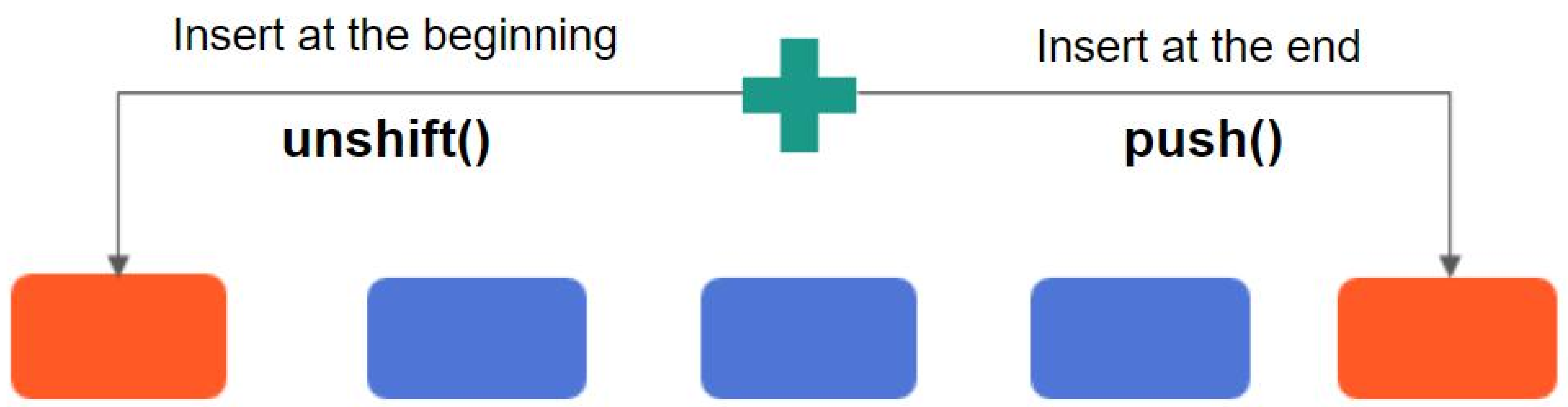
Learning Objectives

- Demonstrate common array operations using array methods
- Explain function expressions and arrow functions
- Filter, transform, and aggregate data using array methods
- Perform operations on complex data structures

Menu

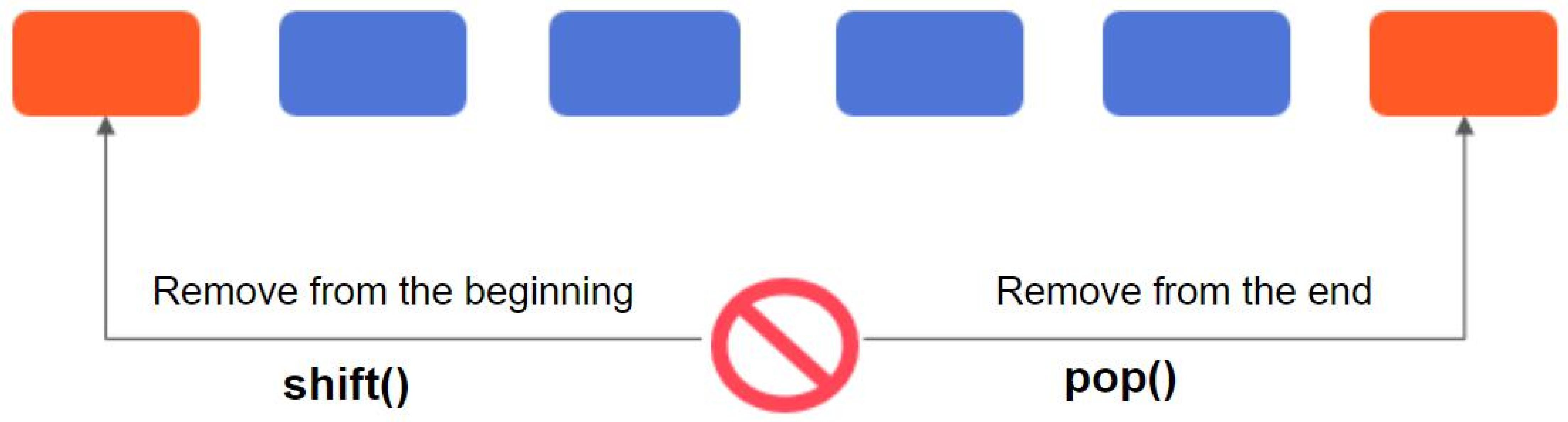
00:07 00:00:42 07/ 42

Array Operations - Insertion



Slide Note

Array Operations – Removal



Menu

Slide Note

Add/Remove Elements Using for Loop

Write a JavaScript program to add/remove the array elements using `for` loop.

Click here for the demo solution.

DEMO

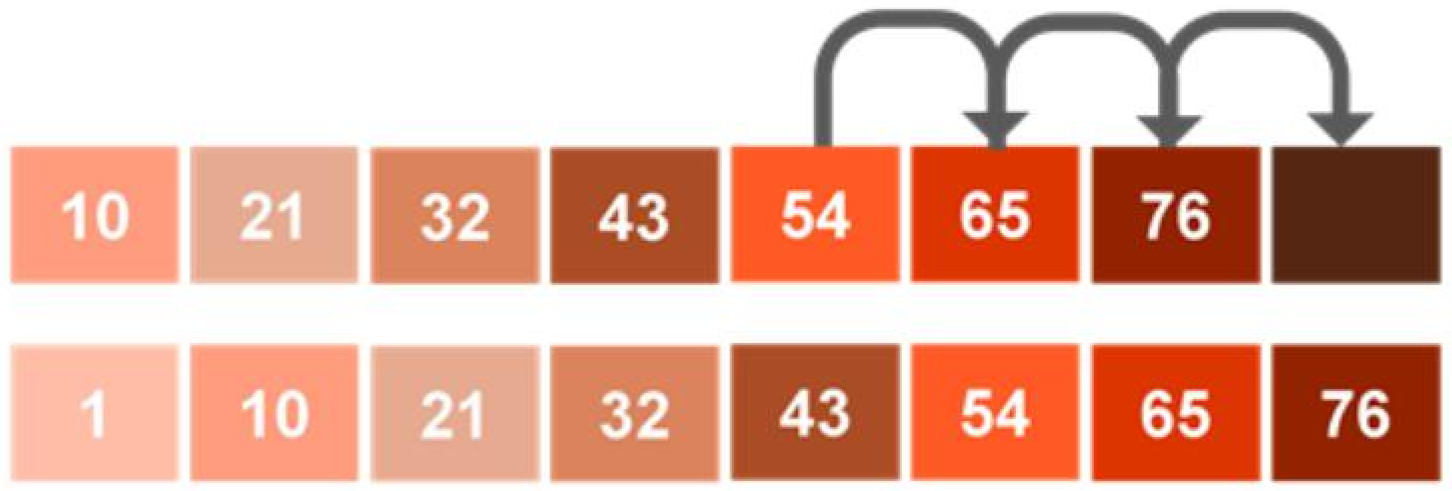
Menu

00:10 00:00:42 10/ 42

Slide Note

Add Elements at the Beginning

- The code adds an element at the beginning of the array by pushing the elements towards right.
    1. Increase the length of the array by 1.
    2. Shift the last element to the right, i.e., to the newly added memory location.
    3. Repeat the previous step till you reach the first element.
    4. Insert the new element at the first index position.



```
let arr = [10, 21, 32, 43, 54, 65, 76];
function addElementAtBeginning(arr,
element){
  arr.length = arr.length + 1;
  for (let i = arr.length - 1; i > 0; i--) {
     arr[i] = arr[i - 1];
   }
   arr[0] = element;
}
addElementAtBeginning(arr, 1);
//Prints [1, 10, 21, 32, 43, 54, 65, 76]
console.log(arr);
```

Menu

00:11 00:00:42 11/ 42

Slide Note

Add and Remove Elements

- The first code adds an element to the end of the array. The steps to achieve this are:
  1. Increase the length of the array by 1
  2. Add the new element at the last index.(length-1)
- The second code removes an element at the end of the array. The steps to achieve this are:
  1. Store the last element in a variable.
  2. Reduce the length of the array by 1 and return the last element.
- Do you need to write a code every time for these kind of operations?
- Does JavaScript provide any built-in methods?

```
let arr = [10, 21, 32, 43, 54, 65, 76];
function addElement(arr, element) {
arr.length = arr.length + 1;
arr[arr.length - 1] = element;
}
addElement(arr, 87);
//Prints [10, 21, 32,43, 54, 65, 76, 87]
console.log(arr);
```

```
function removeElement(arr) {
let lastElement = arr[arr.length-1];
arr.length = arr.length-1;
return lastElement;
}
//prints 87
console.log(removeElement(arr));
//Prints [10, 21, 32,43, 54, 65, 76]
console.log(arr);
```

Menu

Slide Note

Array Operations: Built-in Methods

Insertion	• push(), unshift()
Removal	• pop(), shift(), splice()
Traversal	• forEach()
Transformation	• map(), filter(), slice()
Search	• find(), indexOf(), lastIndexOf()
Aggregation	• reduce()

Menu

00:13:00:00:42

Slide Note

Array Methods to Add/Remove Elements

Write a JavaScript program that changes the array elements while performing array operations using methods like push, shift, pop, unshift and sort, etc.

Click here for the demo solution.

DEMO

Menu

00:14 00:00:42

14/ 42

Slide Note

```
//Add elements from the start of an array
let fruits = [ 'Banana', 'Apple', 'Grapes'];

fruits.splice(2, 0, 'Lemon', 'Kiwi');

console.log(fruits);
//['Banana', 'Apple', 'Lemon', 'kiwi',
'Grapes']
```

```
//Remove elements from the end of the array
let fruits = [ 'Banana', 'Apple',
'Grapes', 'Mango', 'Cherry'];
let removedItems = fruits.splice(-3,3);

console.log(fruits);
//['Banana', 'Apple']

console.log(removedItems);
//['Grapes', 'Mango', 'Cherry']
```

Add/Remove Multiple Elements in an Array

- splice(): Adds/removes elements from an array.
- In the first code, the splice method adds 2 elements at the second position.
- In this method, the first parameter(2) defines the position where new elements need to be added.
- The second parameter(0) defines how many elements need to be removed.
- In the second code, the splice method removes the last 3 elements from the end of the array.
- In this method, the first parameter (-3) defines the position where new elements need to be spliced in.
- The second parameter (3) defines how many elements are to be removed.

Menu

Slide Note

slice(): Copy an Array

- slice(): selects a part of an array from start to end, where start and end represent the index of items in that array and returns the new array.
- The original array **will not be modified.**
- In the first code, the slice method slices out a part of an array starting from the second element.
- In the second code, this method takes 2 arguments.
    - The first argument defines the index start position.
    - The second argument defines the index end position (excluding it).

```
let fruits = [ 'Banana', 'Apple',
'Grapes', 'Mango', 'Cherry'];
const fruitList = fruits.slice(2);
console.log(fruitList);
//['Grapes', 'Mango', 'Cherry']
```

```
let fruits =  [ 'Banana', 'Apple',
'Grapes', 'Mango', 'Cherry'];
const fruitList = fruits.slice(1,3);
console.log(fruitList);
//['Apple', 'Grapes']
```

Slide Note

Array Spread Operator

```
const colors1 = ["Violet", "Indigo",
"Blue "];
const colors2 = ["Green", "Yellow",
"Orange", "Red"];
let colors = [...colors1,
...colors2];
//Prints all the rainbow colors
console.log(colors);
//Assign the first and second item
from num array to 'one' and 'two'
variables and the rest in an array
const num = [1, 2, 3, 4, 5, 6];
const [one, two, ...rest] = num;
```

- The JavaScript spread operator is denoted by three dots (...).
- It allows us to quickly copy all or part of an existing array or object into another array or object.
- The spread operator is used often in combination with the array destructuring.

Menu

Slide Note

Quick Check

The code snippet to store elements in queue is shown below.
The queue follows the FIFO pattern for adding and removing elements from the queue.
What should be the code after line no. 4 to remove the first element pushed in?

```
let words = [];
words.push('a');
words.push('b');
words.push('c');
//code to remove the first element pushed in
```

Menu

Slide Note

Quick Check: Solution

The code snippet to store elements in queue is shown below.
The queue follows the FIFO pattern for adding and removing elements from the queue.
What should be the code after line no. 4 to remove the first element pushed in?

```
let words = [];
words.push('a');
words.push('b');
words.push('c');
//code to remove the first element pushed in
```

If pop( ) is used, the elements are added and removed as per the LIFO method.

shift( ) removes the element at the first position, so the FIFO method is followed.

Slide Note

Shallow Copy of an Array

Write a JavaScript program to copy a portion of an array without mutating the array using the slice method.

Click here for the demo solution.

DEMO

Menu

00:20 00:00:42 20/ 42

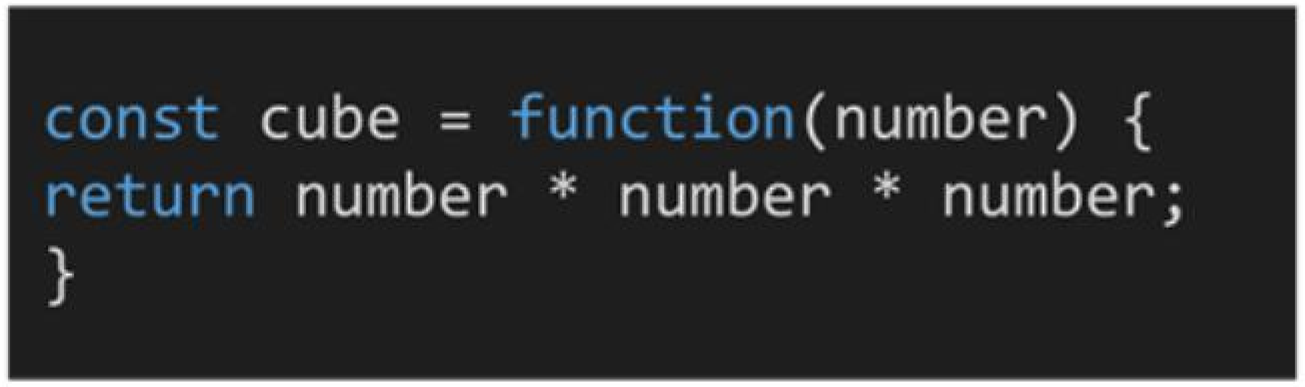
Functions: First Class Citizens of JavaScript

- Functions are the first-class citizens in JavaScript.
- A programming language is said to have first-class functions if functions in that language are treated like other variables. For example:
  - A function can be assigned as a value to a variable
  - They can be passed as an argument to other functions
  - They can be returned by other functions

Slide Note

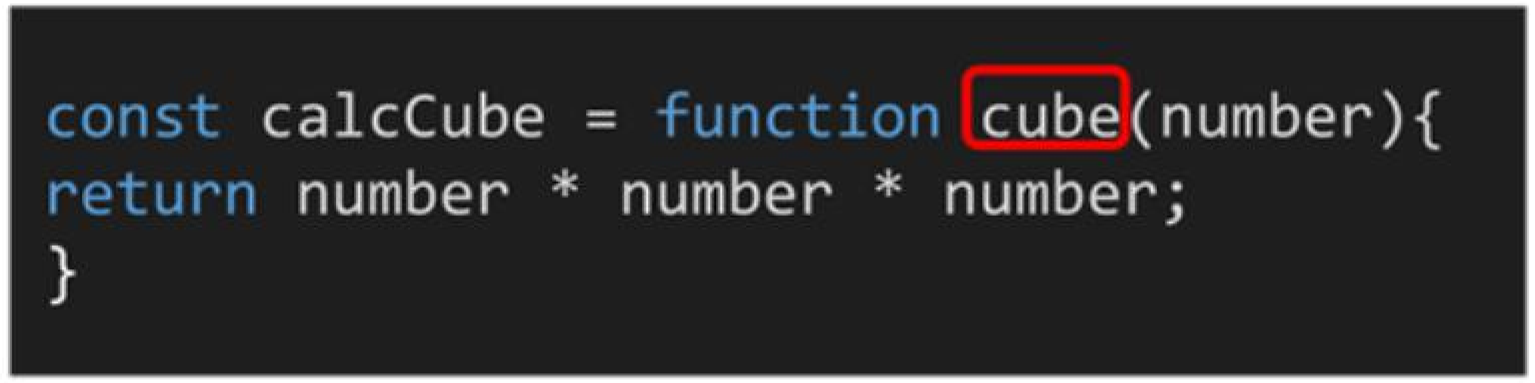
Function Expressions

- Functions can also be created by using expressions. Such functions do not have a name and are called as anonymous functions.
- A function expression can be stored in a variable.
- It is very similar to and has almost the same syntax as a function definition.
- Names can be optionally provided with a function expression, which allows a function to refer to itself, making debugging easier.



```
const cube = function(number) {
return number * number * number;
}
```

Anonymous functions



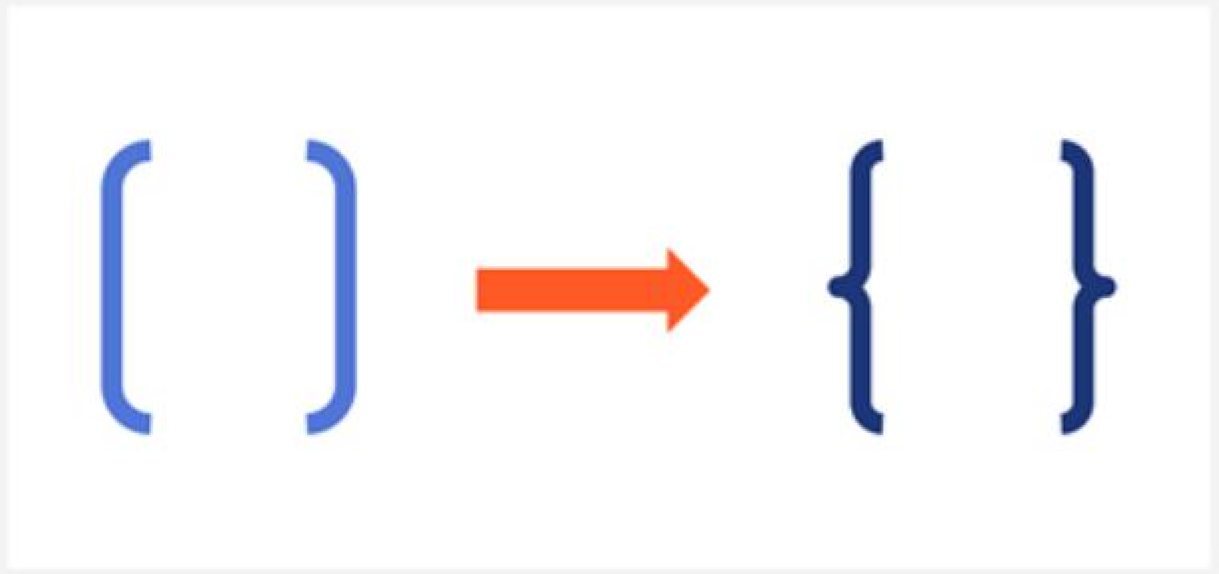
```
const calcCube = function cube(number){
return number * number * number;
}
```

Named function expression

Menu

Slide Note

Array Methods and Arrow Functions



- Array built-in methods like filter(), find(), map(), etc. take functions as arguments.
- These functions are very simple and concise to create and are often better than function expressions.
- These are called **arrow functions,** which were introduced in the JavaScript ES6 version.
- We can write shorter function syntax using arrow function expressions.

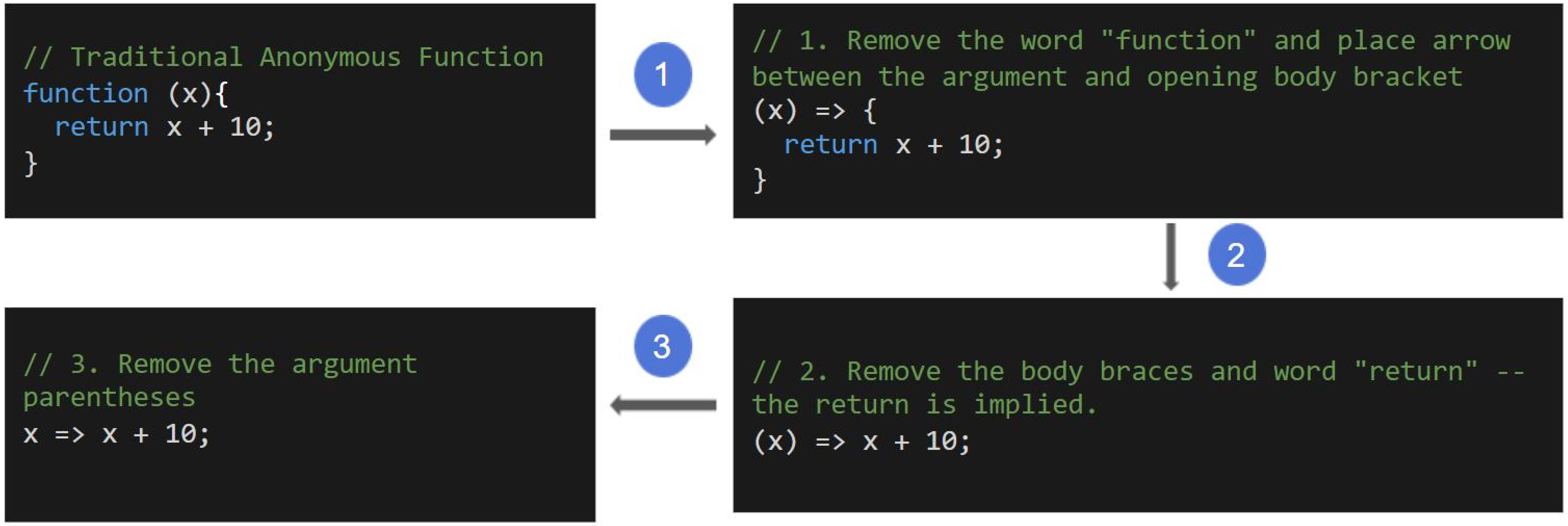
Menu

Slide Note

In Arrow functions, the { braces } and (parentheses) and "return" are required in some cases.

Arrow Functions

- An arrow function expression is a compact alternative to a traditional function expression that has a shorter function syntax.



```
// Traditional Anonymous Function
function (x){
  return x + 10;
}
```

```
// 1. Remove the word "function" and place arrow
between the argument and opening body bracket
(x) => {
  return x + 10;
}
```

```
// 2. Remove the body braces and word "return" --
the return is implied.
(x) => x + 10;
```

```
// 3. Remove the argument
parentheses
x => x + 10;
```

Step-by-step decomposition of anonymous function to arrow function

Menu

00:24 00:00:42 24/ 42

Slide Note

And for **named functions,** arrow expressions are treated like variables:

// Traditional Functionfunction add (a){ return a + 100;}// Arrow Functionlet add = a => a + 100;

Arrow Function Examples

- Braces are required around the arguments when you have multiple or no arguments.

```
// Traditional Anonymous Function
function (x, y){
  return x * y * 100;
}
```

→

```
// Arrow function
(x, y) => x * y * 100;
```

- Braces and return statements are required if there are multiple lines of code inside the body.

```
// Traditional Anonymous Function
function (x, y){
  let z = 10;
  return x * y * z;
}
```

→

```
// Arrow function
(x, y) => {
  let z = 10;
  return x * y * z;
}
```

Slide Note

Various Syntax in Arrow Functions

Types	General Syntax
Function with one parameter	`value => expression`
Function with multiple parameters	`(value1, value2) => expression`

Function with multi-line statements	`value => {`
	`let  result = 0;`
	`return value + result;`
	`}`

Function with multiple parameters with multi-line statements	`(value1, value2) => {`
	`let  result = 0;`
	`return value1 + value2 + result;`
	`}`

Arrow functions stored in variables	`let compute = (a, b) => expression;`

Slide Note

Arrow Functions

Write a JavaScript program to understand the various arrow function notation syntax.

Click here for the demo solution.

DEMO

Menu

00:27:00:00:42 27/ 42

Slide Note

In this code, 'this' keyword refers to the current collection object (or current array object) which invokes the function.

filter(): Filter Array Elements

- The filter() method creates a new array with all elements that pass the test implemented by the provided function.
  - The provided function is a callback function, which is a predicate to test each element of the array. (A callback is a function passed as an argument to another function.)
  - This function returns a value that is either true to keep the element or false otherwise.
- It does not modify the original array.
- `The CheckAge()` function is the callback function.

Note: Second code is Collection object's built-in filter method definition.

```
const ages = [18, 33, 16, 22];
const result = ages.filter(checkAge);
console.log(result);
// [18, 33, 22]
function checkAge(age) {
  return age >= 18;
}
```

```
//Collection object's built-in filter
method
function Collection(...set) {
this.set = set,
  this.filter = function (fn) {
        let filteredSet = [];
        for (let item of set) {
             if(fn(item))  filteredSet
.push(item);
        }
         return filteredSet;
    }
}
```

Menu

Slide Note

•
Syntax: *array*.filter(*function(currentValue, index, arr), thisValue*)

- function: A function to run for each array element. (Required)
- currentValue: The value of the current element. (Required)
- index: The index of the current element. (Optional)
- arr: The array of the current element. (Optional)
- thisValue: A value passed to the function as its **this** value.

Traditional to Anonymous to Arrow Functions

```
const ages = [18, 33, 16, 22];
const result = ages.filter(function checkAge(age){
return age >= 18;
});
console.log(result);
// [18, 33, 22]
```

Traditional Function as Callback Function

```
const ages = [18, 33, 16, 22];
const result = ages.filter(function (age) {
  return age >= 18;
}
console.log(result);
// [18, 33, 22]
```

Anonymous Function

```
const ages = [18, 33, 16, 22];
const result = ages.filter( age =>
age>=18);
console.log(result);
//[18, 33, 22]
console.log(ages);
//[18, 33, 16, 22]
```

Arrow Function

Menu

map(): Transform the Array Elements

- The **map()** method **creates a new array** populated with the results of calling a provided function on every element in the calling array.
- It calls the function once for each element in an array.
- It does not execute the function for empty elements.
- It does not modify the original array.

```
const values = [3, 5, 7, 9];
const result = values.map(square);
Console.log(result);
// [9, 25, 49, 81]
function square(x) {
  return x * x;
}
```

Traditional Function

```
const values = [3, 5 7, 9];
const result = values.map( x => x *
x);
console.log(result);
//[9, 25, 49, 81 ]
console.log(values);
//[3, 5, 7, 9]
```

Arrow Function

Slide Note

The first time that the callback is run, there is no "return value of the previous calculation." If supplied, an initial value may be used in its place. Otherwise, the array element at index 0 is used as the initial value and iteration starts from the next element (index 1 instead of index 0).

reduce(): Aggregate Array Elements

- The reduce() method executes a user-supplied reducer call back function on each element of the array.
- It returns a single value which is the function's accumulated result.
- It does not modify the original array.
- The first code computes the product of all array elements by using the traditional function as a callback function.
- The second code computes the sum of all array elements using the arrow function.
- When the callback is run for the first time, there is no "return value of the previous calculation." It uses initialValue as its value since it is supplied.
- If initialValue is not supplied, the array element at index 0 is used as the initial value, and iteration starts from the next element.

```
//product of all array elements
const values = [13, 5, 7];
const result = values.reduce(prod);
console.log(result);// 455
function prod(result, x) {
  return result * x;
}
```

```
const array1 = [1, 2, 3, 4];
//0+1+2+3+4:Sum of all array elements
const initialValue = 0;
const sumWithInitial = array1.reduce(
  (previousValue, currentValue) =>
previousValue + currentValue,
  initialValue
);
console.log(sumWithInitial); //10
```

Menu

00:31 00:00:42 31/ 42

Slide Note

Transform Student Names

Write JavaScript functions to convert the list of student names into capital letters. Then retrieve the student list which has names starting with 'A.' Also, find the number of student names whose names start with 'A' using the reduce method.

Finally, use function chaining to get all the above three listed requirements.

Click here for the demo solution.

DEMO

Menu

Slide Note

Function Chaining

- Function chaining is a programming strategy in JavaScript where multiple functions are called on the same object consecutively.
- It uses **dot notation** to group functions on a single line.
- It improves performance and increases code readability by writing concise code..

```
const names = ['james', 'alan', 'sophia', 'lawerence', 'andrew',
'alexander', 'ruby', 'ariana', 'bruce', 'charlie', 'andrea'];
let count = 0;
let countOfNames = names
    .map(name => name.toUpperCase())
    .filter(name => name.startsWith('A'))
    .reduce((count, name) => {
        return ++count;
    }, count);
console.log(`Count of names starting with 'a' :${countOfNames}`);
```

Menu

Slide Note

Quick Check

Here is a code that should filter out colors starting with ‘R.’ However, the code creates an empty array.

What correction is required here?

```
let list = ['red', 'pink', 'rose', 'purple']
let filteredList = list.map(item => item.toUpperCase
()).filter(value => {
  value.startsWith('R')
})
console.log(filteredList)
```

Menu

00:34 00:00:42 34/ 42

Slide Note

Quick Check: Solution

Here is a code that should filter out colors starting with ‘R.’ However, the code creates an empty array.

What correction is required here?

```
let list = ['red', 'pink', 'rose', 'purple']

let filteredList = list.map(item => item.toUpperCase())
.filter(value => {
  return value.startsWith('R')
})

console.log(filteredList)
```

Correction: Return keyword is a must as a filter function needs a predicate that returns Boolean for filtering

Slide Note

Think and Tell

- You have worked with arrays and objects
- Is it possible to have:
  - An array of objects?
  - An array as an object property?
  - An object as object property?
- Would these options lead to the creation of complex data structures?

Menu

00:36:00:00:42 36/ 42

Slide Note

JavaScript provides complex data structures, such as an array of objects where an array is an object property

Menu

00:37:00:00:42 37/ 42

Slide Note

Complex Data Structures

- Working with real-time scenarios requires handling complex data structures.
- The complexity is basically due to one data structure being nested inside another data structure.
- A few examples of complex data structures with examples include:
  - An array of objects
  - An array as object property
  - An object as a property

```
// Array of Objects
const employees = [{
    firstName: "Roger",
    lastName: "Brown",
    email: "roger.b@gmail.com"
},
{
    firstName: "Diana",
    lastName: "Michelle",
    email: "diana.m@gmail.com"
}]
```

```
// Array as Object Property
const employee = {
    employeeName: "Brian Ruth",
    languagesKnown:["english","spanish","french"]
}
```

```
// Object as Property
const employee = {
    employeeName: "Brian Ruth",
    address: {
        city: "",
        state: "California",
        postalCode: "CA"
    }
}
```

Menu

00:38/00:00:42 38/ 42

Slide Note

Manipulate Array of Objects

Write a JavaScript program to find the number of students based on the gender of a given set of student objects.

Click here for the demo solution.

DEMO

Menu

00:39 00:00:42

39/ 42

Slide Note

Comparison – Array vs. Objects

Array

- Arrays are used to create a collection of multiple values that provide similar pieces of information.
- The elements in an array are ordered.
- Items in an array are accessed by a numerical position using the `[]` notation.

```
const cars = ['BMW', 'Merc', 'Ford'];
console.log(cars[0]);   // BMW
console.log(typeof cars); // object
```

- An array is an object in JavaScript, however, the `dot` operator cannot be used to access elements, since the position is a numeric value, and the `dot` operator supports only string.

Object

- Objects are created to model real-time "things" in program.
- Objects have properties that are in key-value format, where each value can be of a different type.
- Properties are accessed using `dot (.)` as well as `[ ]` notation.

```
let car = {
    color: "white",
    make: "BMW",
    type: "sports car"
}
console.log(car.color); // white
Console.log(car["color"]); // white
```

Menu

00:40 / 00:00:42 40/ 42

Slide Note

Using `for...in` loop

- A `for...in` a loop is used to iterate over an object's properties.
- If the object is an array, it retrieves the array's index property.
- If the object is a non-array type, it retrieves the object's property names.
- The code on the slide has employee object created and the details are accessed using `for...in` loop
- At each iteration, the variable prop refers to the object's property and the value is accessed using [ ] notation.

```
const employee = {
    firstName: "Tina",
    lastName: "Keith",
    email: "tina.k@gmail.com",
}

console.log(`Employee Details`);
for (let prop in employee) {
    console.log(`${prop}: ${employee[prop]}`);
}
```

Menu