JavaScript

Table of Contents

Introduction to JavaScript

Dynamic Behavior at the **Client Side**
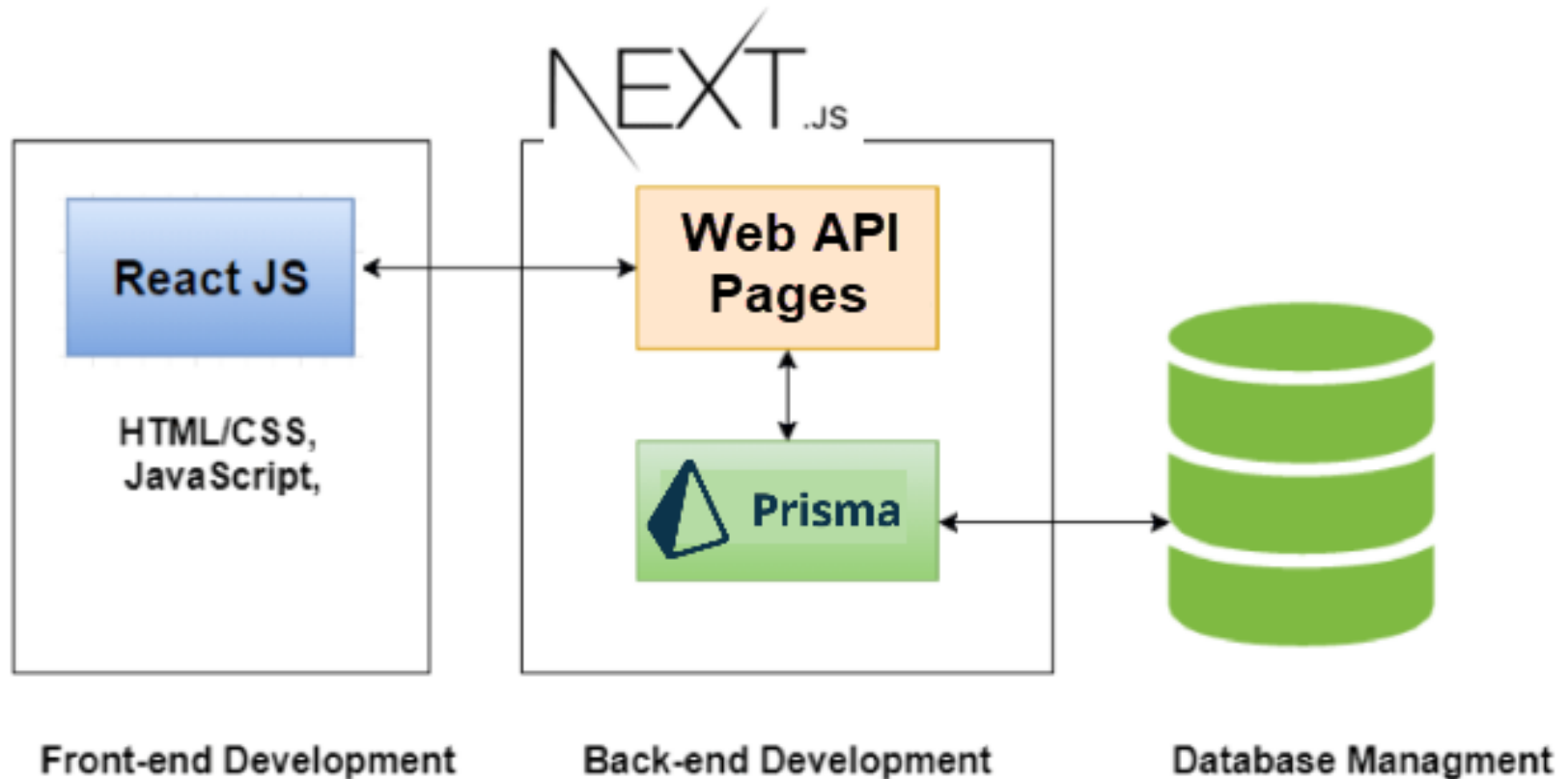
Or Server-Side Web applications
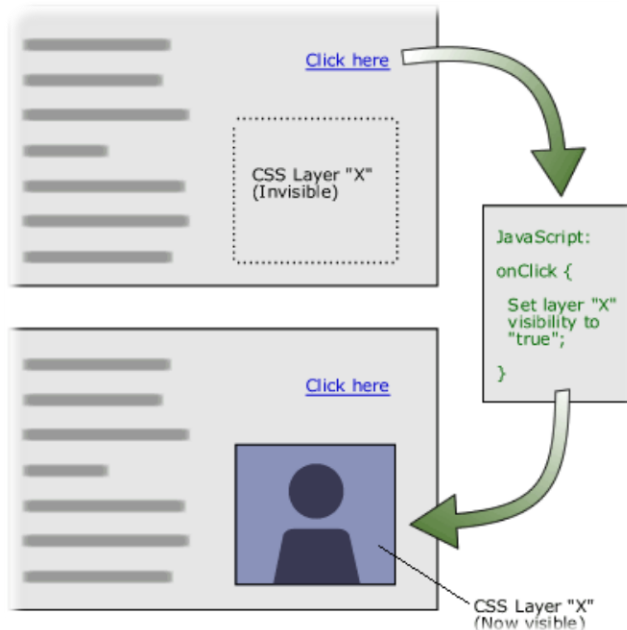
JavaScript

- JavaScript is a platform independent **scripting** language

  o Lightweight but a powerful **interpreted** language

  o Supports both **functional** and **object-oriented** programming style

  o Current Version ES 2023 (ECMAScript 2023)

  o Can be used for:

    - **Client-side scripting**: embedded in HTML pages and interpreted by the Web browser

    - **Server-side programming** using **Node.js**

    - **Desktop app development** (e.g., https://electronjs.org)

    - **Mobile app development** (e.g., https://reactnative.dev/)

What Can JavaScript Do?

- **Web Client-side Dynamic Behavior**

  - Handle client-side events such as button clicked event
    - e.g., Changing an image on moving mouse over it

  - Manipulate the Document Object Model (DOM) of the page: read, modify, add, delete HTML elements

  - Validate form input values before being submitted to the server

  - Perform computations, sorting and animation

  - Perform asynchronous server calls (AJAX) to load new page content or submit data to the server without reloading the page

- Server-side Web applications development using Node.js

- Other usage such as desktop apps, mobile apps and game development

Full stack web development using React, Next.js and Prisma

NEXT.js

React JS	Web API Pages

HTML/CSS,
JavaScript,

Prisma

Front-end Development　　　　**Back-end Development**　　　　**Database Managment**

JavaScript is the common language throughout the full stack, and
JSON is the common data format

JS IS EVERYWHERE

JavaScript Syntax

- ## JavaScript is syntactically a C family language

  - o It differs from C mainly in its type system

- ## The JavaScript syntax is like Java and C#

  - o Variables (by dynamically typed in JavaScript)

  - o Operators (+, *, =, !=, &&, ++, …)

  - o Conditional statements (`if`, `else`, `switch`)

  - o Loops (`for`, `while`)

  - o Arrays (`myArray[]`) and associative arrays (`myArray['abc']`)

  - o Functions

  - o Classes

- Although there are **strong outward similarities** between JavaScript and Java, the two are **distinct languages and differ greatly in their design**

Data Types in JavaScript

Declaring Variables

- Declare variables using **const**. If you intend to change the variable value, then use **let**.

  o Variable names in JavaScript is case-sensitive

- The syntax is the following:

```
const <identifier> [= <initialization>];
```

- Example:
```
const height = 110;
```

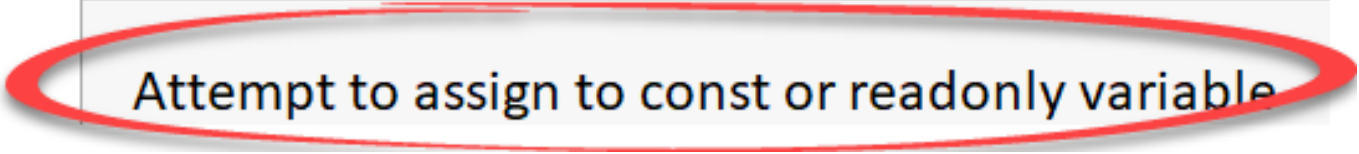- **const** - creates a block scope variable (accessible only in its scope)

```
for(const n of [1, 2, 3, 4]){
    console.log(n);
}
//accessing n here throws exception
```

Declaring Mutable Variable using let

- **const** - creates a constant variable. Its value is read-only and cannot be changed

```
const height = 110;

height = 120;
            Attempt to assign to const or readonly variable
```

- **let** - to declare a Mutable Variable use let

```
let height = 110;
height = 120;
```

JavaScript Data Types

- JavaScript is a **Loosely Typed** and **Dynamic** language
  - The variable datatype is derived from the assigned value

```
const count = 5; // variable holds a number
const name = 'Ali Dahak'; // variable holds a string
const grade = 5.25 // grade holds a number
```

Primitive types

- There are 7 data types in JavaScript:
  - number
  - bigint
  - string
  - boolean
  - undefined
  - function
  - object (everything else is an object)
- Use **typeof** to find out the variable type
- A string is a sequence of characters enclosed in single (' ') or double quotes (" ")

```
const str1 = "Some text saved in a string variable";
const str2 = 'text enclosed in single quotes';
```

String Methods

- `str.length` returns the number of characters

- Indexer (`str[index]`) or `str.charAt(index)`
  - Gets a single-character string at location `index`
  - If index is outside the range of string characters, the indexer returns `undefined`
    - e.g., `string[-1]` or `string[string.length]`

- str3 = str1.concat(str2) or str3 = str1 + str2;
  - Returns a new string containing the concatenation of the two strings

- Other String methods

http://www.w3schools.com/jsref/jsref_obj_string.asp

Convert a number to a string

- Use number's method (**toString**)

  str = num.toString()

- Use **String** function

  str = String(num)

Convert a string to a number

- Use the **parseInt** function

  num = parseInt(str)

- Use the Number function

  num = Number(str)

- Use the + prefix operator

  num = +str

Template Literals

- Template Literals allow creating dynamic templated string with placeholders
  - Replaces long string concatenation!

```
const person = {fname: 'Samir',  lname:'Mujtahid'};

console.log(`Full name: ${person.fname} ${person.lname}`);
```

undefined vs. null Values

- In JavaScript, undefined means a variable has been declared but **has not been assigned a value**, e.g.,:

```
let testVar; console.log(testVar); //shows undefined
console.log(typeof testVar); //shows undefined
```

- null is an assignment value. It can be assigned to a variable as a representation of no value:

```
let testVar = null;
console.log(testVar); //shows null
console.log(typeof testVar); //shows object
```

=> undefined and null are two distinct types: **undefined** is a value of type "undefined" while null is an **object**

NaN

- NaN (Not a Number) is an illegal number

- Result of undefined or erroneous operations such **'A' * 2** will return a **NaN**

- Toxic: any arithmetic operation with **NaN** as an input will have **NaN** as a result

- Use **isNaN()** function determines whether a value is an illegal number (Not-a-Number).

  o **NaN** is not equal to anything, including **NaN**

  ```
  NaN === NaN is false

  NaN !== NaN is true
  ```

Comments

```
// slash slash line comment

/*
    slash star
    block
    comment
*/
```

Operators in JavaScript

Arithmetic, Logical, Comparison, Assignment, Etc.

Categories of Operators in JS

Category	Operators
Arithmetic	+ - * / % ++ --
Logical	&& || ^ !
Binary	& | ^ ~ << >>
Comparison	== != < > <= >= === !==
Assignment	= += -= *= /= %= &= |= ^= <<= >>=
String concatenation	+
Other	. [] () ?: new

http://www.w3schools.com/js/js_operators.asp

Comparison Operators

- Comparison operators are used to compare variables

  ○ ==, <, >, >=, <=, !=, ===, !==

- Comparison operators example:

```javascript
const a = 5;
const b = 4;
console.log(a >= b); // True
console.log(a != b); // True
console.log(a == b); // False

console.log(0 == ""); // True
console.log(0 === ""); //False
```
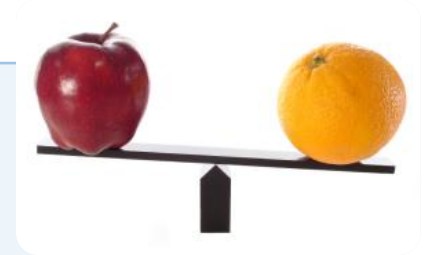
== vs. ===

!=

Non-equality comparison:
Returns true if the operands are
not equal to each other.

==

Equality comparison:
Returns true when both operands are
equal. The operands are converted to
the same type before being compared.

!==

Non-equality comparison without type
conversion:
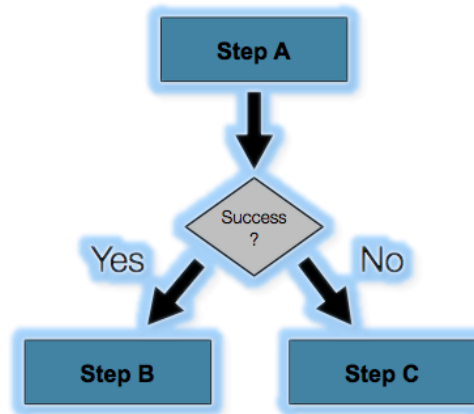Returns true if the operands are not
equal OR they are different types.

===

Equality and type comparison:
Returns true if both operands are
equal and of the same type.

- See Examples

http://www.w3schools.com/js/js_comparisons.asp

Conditional Statements

if-else Statement – Example

- Checking a number if it is odd or even

```
const number = 10;

if (number % 2 === 0)
{
    console.log('This number is even');
}
else
{
    console.log('This number is odd');
}
```

switch-case Statement

- Selects for execution a statement from a list depending on the value of the **switch** expression

```
switch (day)
{
    case 1: console.log('Monday'); break;
    case 2: console.log('Tuesday'); break;
    case 3: console.log('Wednesday'); break;
    case 4: console.log('Thursday'); break;
    case 5: console.log('Friday'); break;
    case 6: console.log('Saturday'); break;
    case 7: console.log('Sunday'); break;
    default: console.log('Error!'); break;
}
```

False-like conditions

- These values are always false (when used in a condition)
  - false
  - 0 (zero)
  - "" (empty string)
  - null
  - Undefined
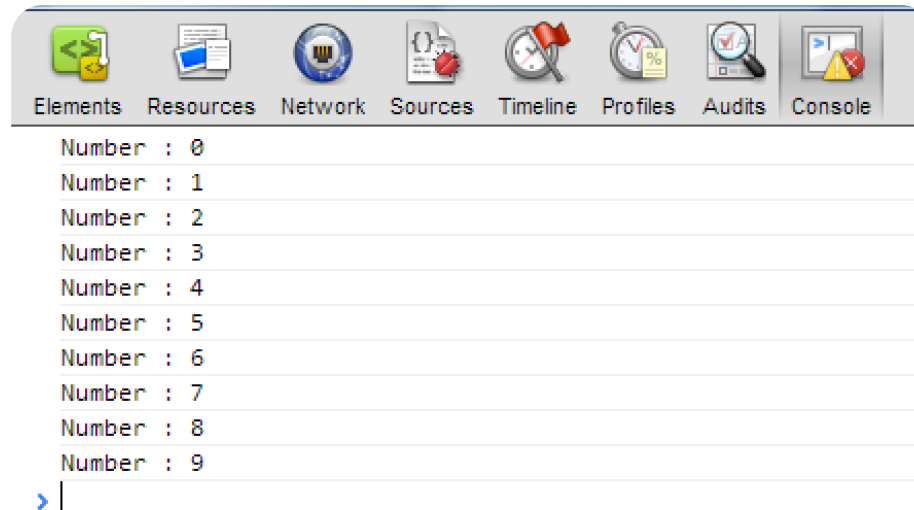 - NaN

- All other values are true
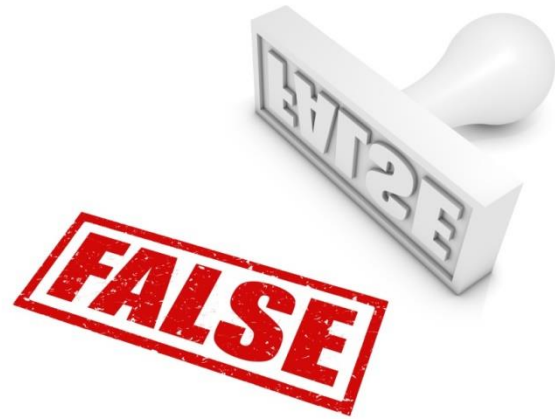
while (...)
do { ... }
for { ... }
Loops

Execute Blocks of Code Multiple Times

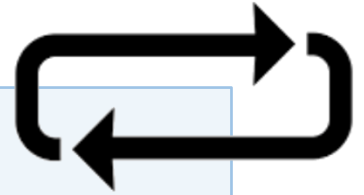# While Loop – Example

```
let counter = 0;
while (counter < 10){
    console.log(`Number : ${counter}`);
    counter++;
}
```

Other loop structures

- Do-While Loop:

```
do {
    statements;
}
while (condition);
```

- **For** loop:

```
for (initialization; test; update) {
    statements;
}
```

```
// Compute n!:
let factorial = 1;
for (let i = 1; i <= n; i++){
    factorial *= i;
}
```
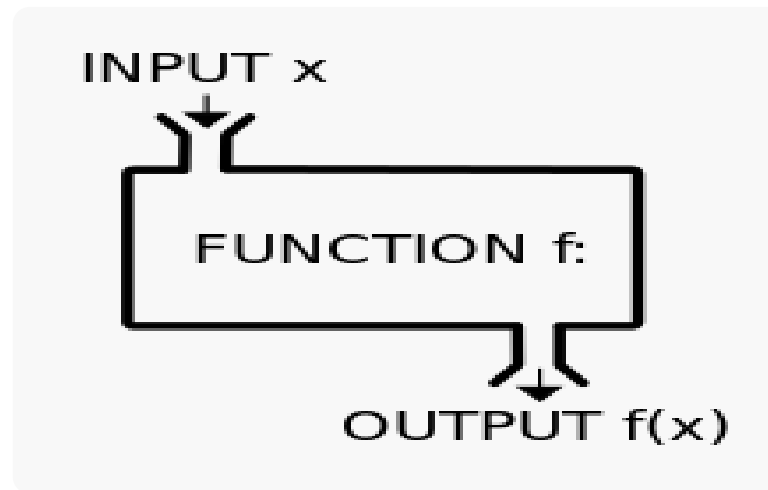
For-of loop

- For-of loop iterates over a list of values

```javascript
const nums = [1, 2, 3, 4, 5, 6, 7, 8, 9];
let sum = 0;

for (const n of nums) {

    sum += n;

}
```

- For-of loop iterates over the properties of an object

```javascript
const student = { firstName: "Ali", lastName: "Mujtahid" };

for (const [key, value] of Object.entries(student)) {
    console.log(`${key} = ${value}`);
}
```

Functions

INPUT x

FUNCTION f:

OUTPUT f(x)

```
function (parameter) {
    return expression;
}
```

```
function double (number) { return number * 2; }
double(212); // call function
```

```
// Function expression
const average = function (a, b) {
    return (a + b) / 2;
}
average(10, 20); // call function
```

OR

```
const average = (a, b) => (a + b) / 2;
average(10, 20);  // call function
```

Arrow Function
Also called LAMBDA expressions

Function Scope

- Every variable has its scope of usage
  - A scope defines where the variable is accessible
  - Generally there are local and global scope

```
const arr = [1, 2, 3, 4, 5, 6, 7];
function countOccurences (value){
  let count = 0;
  for (const num of arr){
    if (num == value){
      count++;
    }
  }
  return count;
}
```

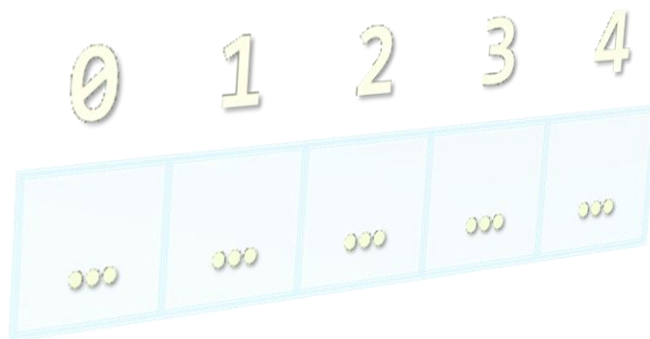**arr** is in the global scope (it is accessible from anywhere)

count is declared inside countOccurences and it can be used only inside it

num is declared inside the for loop and it can be used only inside it

Arrays

Processing Sequences of Elements

https://sdras.github.io/array-explorer/
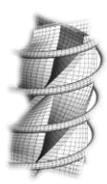
Declaring Arrays

- Declaring an array in JavaScript

```javascript
// Array holding integers
const numbers = [1, 2, 3, 4, 5];

// Array holding strings
const weekDays = ["Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday", "Sunday"]

// Array of different types
const mixedArr = [1, new Date(), "hello"];

// Array of arrays (matrix)
const matrix = [
            [1,2],
            [3,4],
            [5,6]
        ];
```

Processing Arrays Using **for** Loop

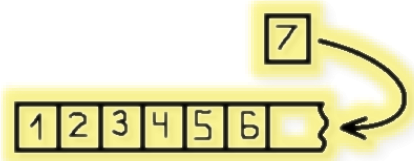☀ The for-of loop iterates over a list of values

```
let sum = 0;
for(const number of [1, 2, 3])
  sum+= number;
```

- Printing array of integers in reversed order:

```
const array = [1, 2, 3, 4, 5];
for (const i = array.length-1; i >= 0; i--) {
    console.log(array[i]);
} // Result: 5 4 3 2 1
```

- Initialize an array:

```
for (const index = 0; index < array.length; index++)
{
    array[index] = index;
}
```

Dynamic Arrays

- All arrays in JavaScript are dynamic
  - Their size can be changed at runtime
  - New elements can be inserted to the array
  - Elements can be removed from the array
- Methods for array manipulation:
  - `array.push(element)`
    - Inserts a new element at the tail of the array
  - `array.pop()`
    - Removes the element at the tail
    - Returns the removed element

Insert/Remove at the head of the array

- `array.unshift(element)`
  - Inserts a new element at the head of the array
- `array.shift()`
  - Removes and returns the element at the head

```
const numbers = [1, 2, 3, 4, 5];
console.log(numbers.join("|")); // result: 1|2|3|4|5

const tail = numbers.pop();        // tail = 5;
console.log(numbers.join("|")); // result: 1|2|3|4

numbers.unshift(0);
console.log(numbers.join("|")); // result: 0|1|2|3|4

const head = numbers.shift();      // head = 0;
console.log(numbers.join("|")); // result: 1|2|3|4
```

Deleting Elements

- Splice removes item(s) from an array and returns the removed item(s)

- This method changes the original array

- Syntax:

`array.splice(index,howmany)`

```
const myArray = ['a', 'b', 'c', 'd'];
const removed = myArray.splice(1, 1);
// myArray after splice ['a', 'c', 'd']
```

Destructuring assignment

- The destructuring assignment makes it easier to **extract** data from arrays or objects into distinct variables

```
const colors = ["red", "green", "blue", "yellow"];

//Extracting array elements and assigning them to variables
const [primaryColor, secondaryColor, ...otherColors] = colors;
primaryColor  = 'red' , secondaryColor = 'green' and
otherColors = [ 'blue', 'yellow' ]
```

3 dots ... is called the rest operator

Spread Operator

- **Spread Operator (3 dots ...)** allows converting an array into consecutive arguments in a function call

```
const nums = [5, 4, 23, 2];
//Spead could be used to convert the array
//into multiple arguments
const max = Math.max(...nums);
console.log("max:", max);
```

- **Spread Operator can also be used to concatenate arrays**

```
const cold = ['autumn', 'winter'];
const warm = ['spring', 'summer'];
// construct an array
const seasons = [...cold, ...warm];
// => ['autumn', 'winter', 'spring', 'summer']
```

Sets

- A collection of values **without duplicates**
  - Sets do not allow duplicate values to be added

```
const names = new Set();
names.add('Samir');
names.add('Fatima');
names.add('Mariam');
names.add('Ahmed');
names.add('Samir'); // won't be added

for (const name of names) {
    console.log(name);
}
```

Maps

- Map is a **collection** of key-value pairs

```javascript
const map = new Map();
map.set(1, 'One');
map.set(2, 'Two');

for(const pair of map) {
    console.log(pair);
}

for(const key of map.keys()) {
    console.log( key, numbersMap.get(key) );
}

for(const value of map.values()) {
    console.log(value);
}
```

Arrow Function (aka Lambda)

λ

# Imperative vs. Declarative

Imperative Programming

- You tell the computer **how** to perform a task.

Declarative Programming

- You tell the computer **what you want**, and you let the compiler (or runtime) figure out the best way to do it. This makes the code simpler and more concise
  - The declarative programming paradigm expresses the logic of computation **without** describing its control flow

What is a Lambda?
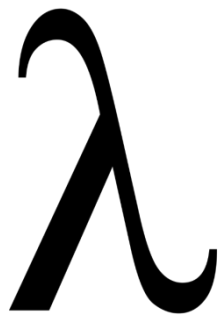
- Lambda is *anonymous function.* It has:
  - Parameters
  - A body
  - A return type

- Also known as **Arrow Function**

- It **don't have a name** (anonymous method)

- It can be assigned to a variable

- It **can be passed as a parameter** to other function:
  - As *code* to be executed by the receiving function

- Concise syntax:

```
Parameters => Body
```

Passing Lambda as a Parameter

- Lambda expression can be passed as a parameter to methods such as *forEach*, *filter* and *map* methods :

const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9];
numbers.forEach (**e => console.log(e)**);

forEach **- Calls a Lambda on Each Element of the list**

- Left side of **=>** operator is a parameter variable

- Right side is the code to operate on the parameter and compute a result

- Allows working with arrays in a **functional style**

Common operations on arrays

.map

o Applies a function to each array element

.filter(condition)

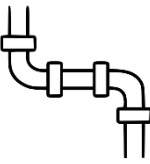o Returns a new array with the elements that satisfy the condition

.find(condition) / findIndex(condition)

o Returns the first array element that satisfy the condition

.reduce

o Applies an accumulator function to each array element to reduce them to a single value

Operations Pipeline

- **A pipeline of operations:** a sequence of operations where the output of each operation becomes the input into the next

  - e.g., .filter -> .map -> .reduce

- Operations are either **Intermediate** or **Terminal**

- **Intermediate operations** produce a new array as output (e.g., map, filter, …)

- **Terminal operations** are the final operation in the pipeline (e.g., reduce, join …)

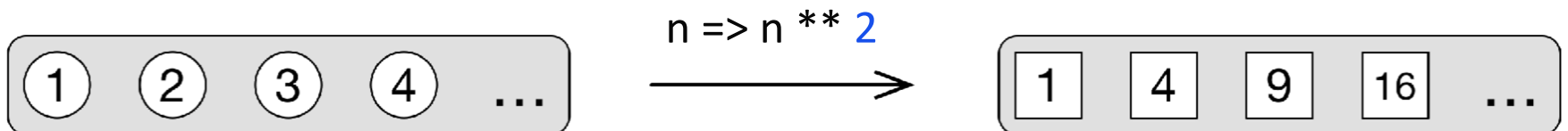 - Once a terminal operation is invoked then no further operations can be performed

Filter

Return elements that satisfy a condition

n => n % 2 == 0

1 2 3 4 ... → 2 4 ...
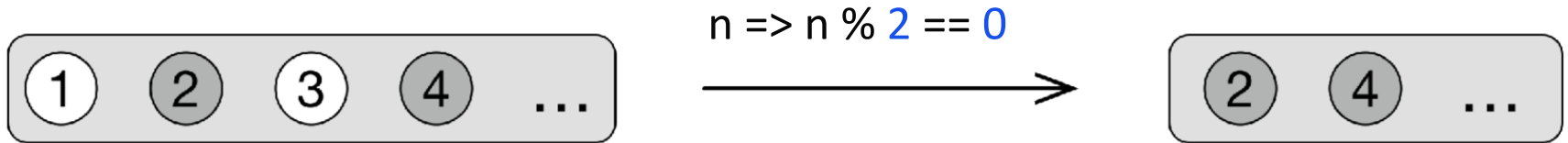
Map

Transform elements by applying a Lambda to each element

n => n ** 2

1 2 3 4 ... → 1 4 9 16 ...

Reduce

Apply an accumulator function to each element of the array to reduce them to a single value

```
// Imperative
let sum = 0
for(const n of numbers)
    sum += n
```

```
//Declarative
const total = numbers.reduce ( (sum, n) => sum + n
)
```
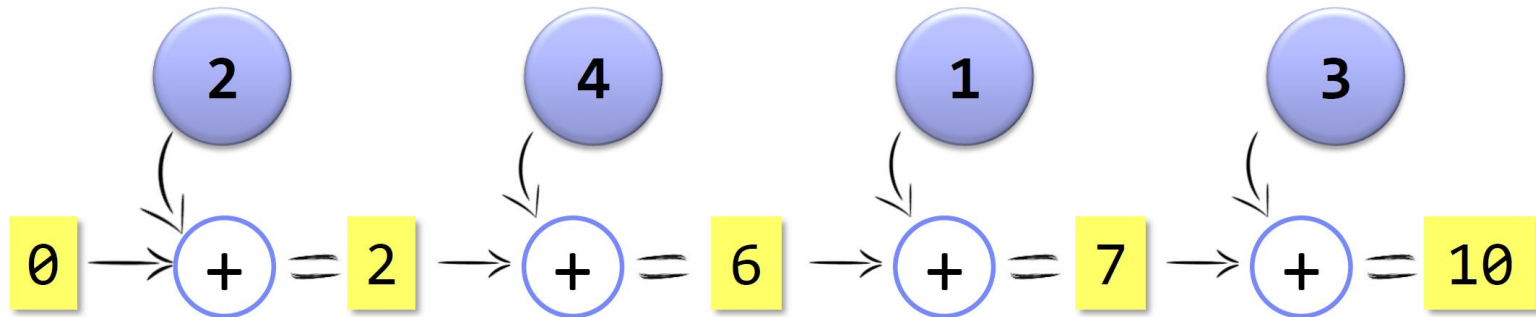
Accumulation Variable

Accumulation Lambda

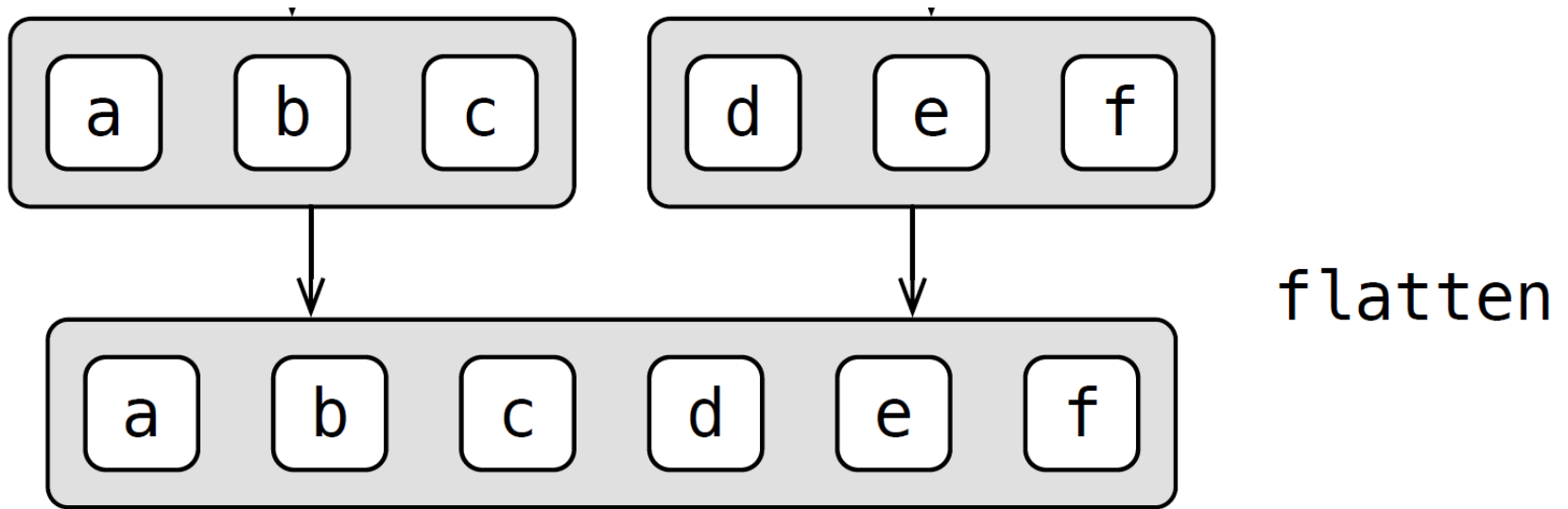**Collapse the multiple elements of an array into a single element**

| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |

reduce → 36

Reduce

.reduce ((sum, n) => sum + n)

Reduce is **terminal** operation that yields a single value

Flat

flatten

```
flattened = [['a', 'b', 'c'], ['d', 'e']].flat()
//flattened array: [ 'a', 'b', 'c', 'd', 'e' ]
console.log("flattened array:", flattened);
```

flatMap

Do a map and flatten the results into 1 list

Each book has a list of authors. **flatMap** combines them to produce a single list of **all** authors

```
const books = [
    {title: "Head First JavaScript", authors: ["Dawn Griffiths", "David Griffiths"]},
    {title: "JavaScript in Action", authors: ["Dmitry Jemerov", "Svetlana Isakova"]}
]

const authors = books.flatMap(b => b.authors);
console.log(authors);
```

Other Array Functions

- `nums.sort((a, b) => a - b)`
  - Sorts the elements of the nums array in ascending order
- `nums.sort((a, b) => b - a)`
  - Sorts the elements of the nums array in descending order
- `array.reverse()`
  - Returns a new array with elements in reversed order
- `array.concat(elements)`
  - Inserts the elements at the end of the array and returns a new array
- `array.join(separator)`
  - Concatenates the elements of the array

Summary

- To start thinking in the functional style ***avoid loops*** and instead use Lambdas

  o Widely used for array processing and UI events handling

- An array can be processed in a pipeline

  o Typical pipeline operations are filter, map and reduce

JavaScript Resources

- Mozilla JavaScript learning links

  o https://developer.mozilla.org/en-US/learn/javascript

- JavaScript features

  o https://github.com/mbeaudru/modern-js-cheatsheet

  o https://exploringjs.com/

- Modern JavaScript Tutorial

  o https://javascript.info/

- JavaScript / Node.js School

  o https://www.classcentral.com/cohorts/js-bootcamp-spring-2022 (YouTube list 1 & list 2)

  o https://nodeschool.io/