Introduction to Javascript

Why isn't HTML/CSS enough?

- HTML/CSS is static.
- HTML/CSS just describes content and styles.
- It doesn't process or manipulate any of that content. Or allow us to respond to user interactions.

What Javascript do?

- Handle events (clicks, hovers, selection, etc)
  e.g. clicking on a button reveals something

- Can process and manipulate content
  e.g. changing the existing HTML elements &
  update existing CSS styles

- Retrieve/send data from/to servers

Where does JS go?

Like CSS, it has many places it can be.

Option 1: In the HTML file itself

```
<body>
  <h1>Hello world</h1>
  <script>
    document.write('Hello world!') ;
  </script>
</body>
```

Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.

Where does JS go?

Option 2: In a separate file

```
<body>
  <script src="script.js"></script>
</body>
```

Placing scripts in external files has some advantages:

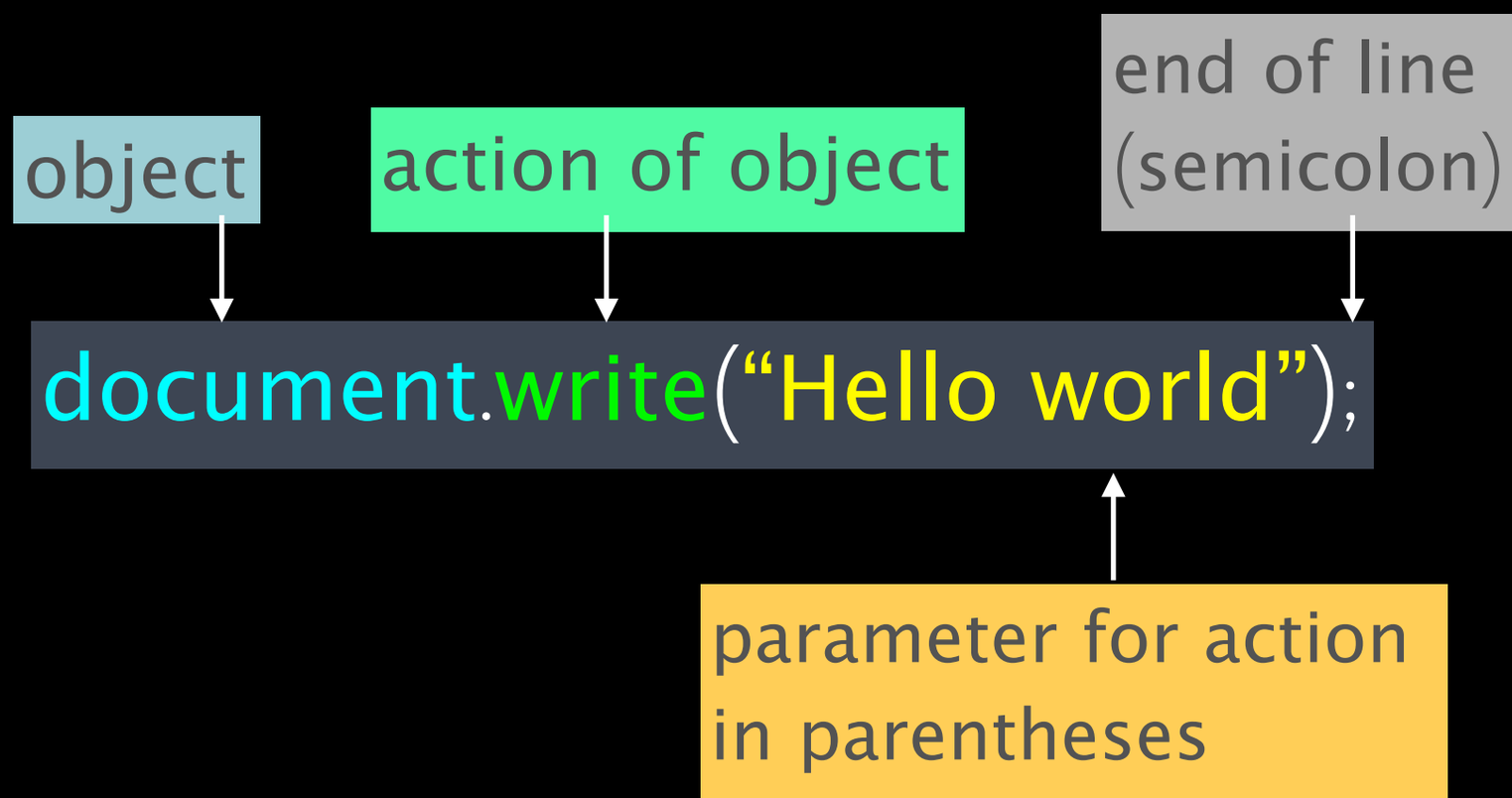
- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Browser will store JavaScript files in its cache can speed up page loads

Let the computer talk to you

Ways to output a message:

1. alert(“Hello world!”);
   -Writing into an alert box
2. document.write(“Hello world”);
   -Writing into the HTML page
3. console.log(“Hello world!”);
   -Writing into the browser console (useful for testing purposes)
4. using innerHTML
   -Writing into an HTML element (DOM manipulation)

Anatomy of an output statement



Statements

- Statements define what the script will do and how it will be done. For example,

```
alert(“Hello, world!”);
```

which shows the message “Hello, world!”

- Statements can be separated with a semicolon.

```
alert(“Hello”); alert(“World”);
```

- Statements are written on separate lines to make the code more readable

```
alert(‘Hello’);
alert(‘World’);
```

Semicolons

- The Semicolon acts as an indication of where the line of instructions ends.

```
alert(“Hello”); // one line of instruction
alert(“World”); // another line of instruction
```

- A semicolon may be omitted in most cases when a line break exists.

- A new line can also indicate where the line of instruction ends.

```
alert(“Hello”)  // one line of instruction
alert(“World”) // another line of instruction
```

Comments

As programs become more and more complex, it become necessary to add *comments* to describe what the code does and why.

Comments can be put into any place of a script. They don't affect its execution because the engine simply ignores them.

One-line comments

```
// this is a comment
// this is another comment
// alert("Hello world")
```

Multiline comments

```
/*  this is a
    multiple lines
    comment */
```

Variables

Variables

Variables are containers used to store any data values. It lets you name your values.

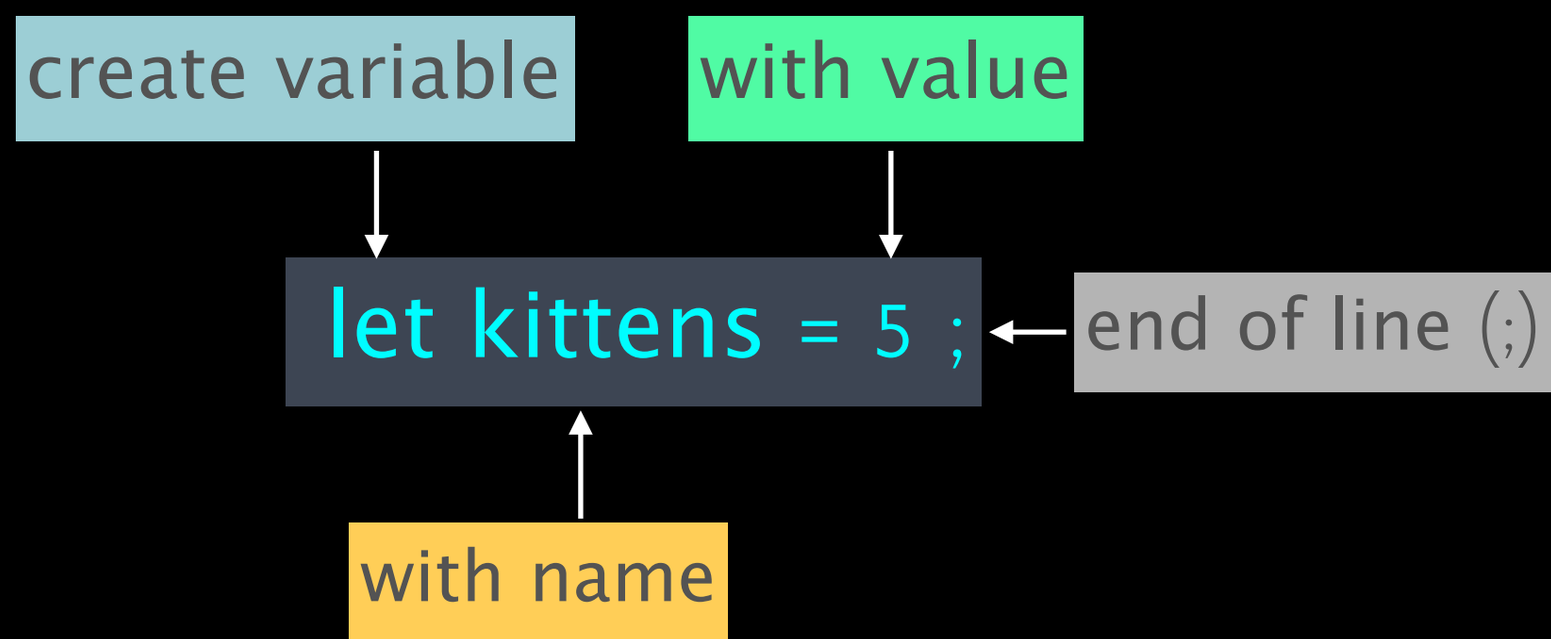
eg. **let x = 20;**

x is a variable that stores the value **20**

- **let** – is a modern variable declaration.
- **const** – is like let, but the value of the variable can't be changed.
- **var** – is an old-school variable declaration.

For now, just stick to the new const and let.
Read more…

Anatomy of a variable



Variable Naming

There are two limitations on variable names in JavaScript:

1. The name must contain only letters, digits, or the symbols $ and _.
2. The first character must not be a digit.

Eg of **valid** names:

- let userName;
- let test123;
- let x;

Eg of **invalid** names:

- let #firstname;
- let 1stButton;
- let my-name;

Variable Naming

1. Case matters
   - Variables named apple and AppLE are two different variables.
2. Reserved names
   - There is a list of reserved words, which **cannot** be used as variable names because they are used by the language itself.
   eg. let, const, if, else, function, etc
3. Name things right
   - Use camelCase
   - A variable name should have a clean, obvious meaning, describing the data that it stores.
   eg. newUserName, startButton, signUpDate

Declare & Assign value to variable

```
let userName = “Sandra”;
console.log(userName)   // “Sandra”
```

Reassign value to the same variable

```
userName = “mei”;
console.log(userName)   // “mei”
```

Note: You only need to specify ‘let’ the first time you declare the variable.

Constant variable

To declare a constant (unchanging) variable, use **const** instead of **let**.

This means that the value of a variable declared with **const** remains the same within its scope. It cannot be reassigned or re-declared.

An attempt to do so would cause an error.

undefined - variable default value

If value is not assigned to a variable, it has a default value of undefined.

```
let userName;
console.log(userName)   // “undefined”
```

Data Types

Data types we are going to cover are:

1. String
2. Number
3. Boolean
4. Array
5. Object

Data Type: String

A string is a line of text.

A string can be any text inside “double-quotes” or ‘single-quotes’.

```
//Displays John Smith (without quotes)

const name = “John Smith”;
document.write(name);
```

You can do things to Strings

Combine strings:

```
console.log("Hi " + name + "!") //using "quotes" & +
console.log(`Hi ${name}!`) //using `back-tick` & ${ }
//both equal to "Hi John Smith!"
```

Get a length of a string:

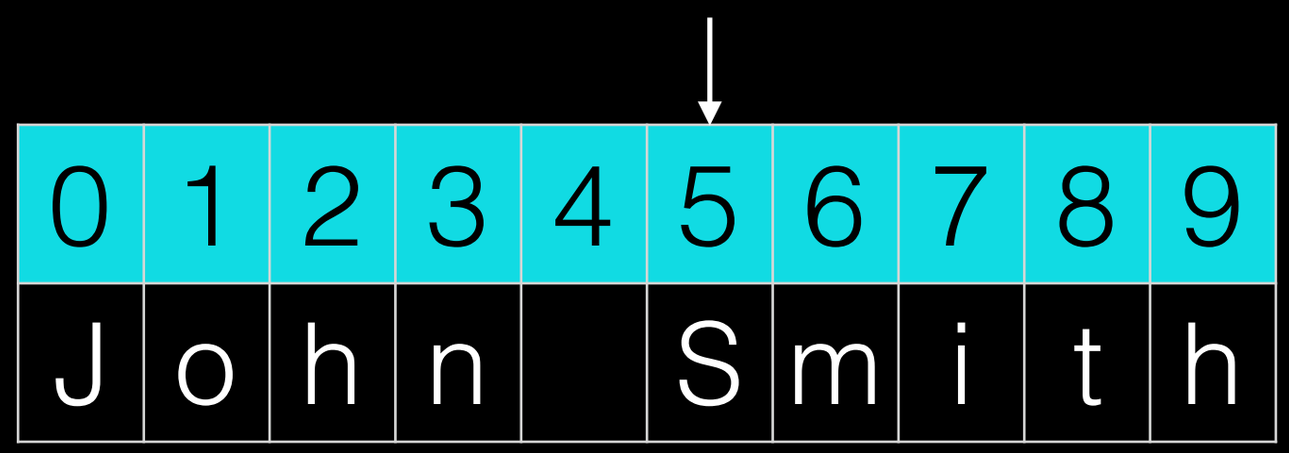
```
name.length //is equal to 10
```

You can do things to Strings

<u>Get a character at a position in a string:</u>

String indexes are zero-based:
The first character is in position 0, the second in 1, and so on.

eg: name[5] //is equal to "S".



Data Type: Number

Numbers in JS can be written in 2 ways:

As integers (no decimal)

2, 42, 900001

As floats (when you need decimal)

42.0, 0.0001, 10.2222

Arithmetic Operators

things you can do with numbers

Addition	42 + 24	
Subtraction	100 - 12	
Multiplication	20 * 4	
Division	100 / 3	
Remainder	42 % 6	Gives you remainder of a division (also called modulo)
Negation	-53	Negative value of number
Increment and Decrement	num++ num - -	increment or decrement by 1

Operator Precedence

Operators have orders of precedence. Ones with highest precedence will be calculated first.

```
5 + 6 * 3 //is equal to 23

(5 + 6) * 3 //is equal to 33
```

Data Type: Boolean

We can use booleans to determine conditions.

A boolean is a value that has 2 states:

true or false

Concept: Boolean expressions

A combination of values and operators that evaluate to either true or false

Examples in English:

- today is Monday✅ → true
- (today is Monday)✅ or (yesterday was Saturday)❌ → true
- (today is Monday)✅ and (we are in Malaysia)✅→ true
- (today is Saturday)❌ or (we are not in Malaysia)❌ → false

Boolean Operators in JS

Instead of using ‘and’, ‘or’, ‘not’, we use these symbols:

And, &&	cold && rainy	is true if both true
Or, ||	warm || dry	is true if at least one is true
Not, !	!green	is opposite of boolean

Truth Table

A	B	A && B	A || B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

!A Truth Table

A	!A
true	false
false	true

Boolean Comparators in JS

We can also compare values. More comparators here.

Less than, <	100 < 3	true if left value is less than right value.
Less than or equal, <=	100 <= 3	true if left value is less than or equal to right value.
Greater than, >	100 > 3	true if left value is more than right value.
Greater than or equal to, >=	100 >= 3	true if left value is more than or equal to right value.
Equal to, ==	mei == mei	true if values are identical in value.
Not equal to, !=	sandra != mei	true if values are not identical in value.

Try it out!

Determine if the following statements are true or false.

```
let x = 0 ; let y = 20 ;
```

No.	Statements	Answer
1	(x == 0) || (x == 1)	true
2	(x < y) && (x > 0)	false
3	(y == 0) || (y > 10)	true
4	!(x == 0) || !(y == x)	true
5	(x < y) && (x > 0) || (y > 8)	true

* Precedence of AND && is higher than OR ||

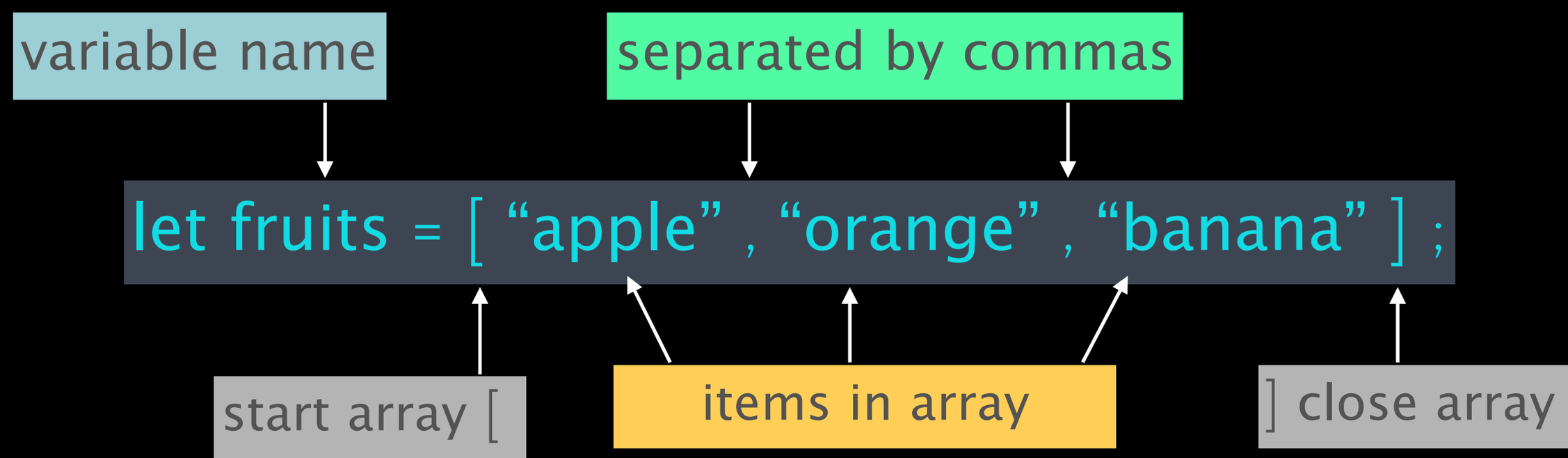
Data Type: Arrays

Arrays are used to store multiple values in a single variable.

What can I put inside an array?
Answer: Anything

```
let fruits = ["apple", "orange", "banana"] ;
let numbers = [1, 2, 3, 4, 5] ;
let things = [42, 'hello', -1, true] ;
let emptyArray = [ ] ;
```

Anatomy of an array



item	“apple”	“orange”	“banana”
index	0	1	2

Getting an item

```
let fruits = [“apple”, “orange”, “banana”] ;

console.log( fruits[0] ); // apple
console.log( fruits[1] ); // orange
console.log( fruits[2] ); // banana
console.log( fruits[3] ); // undefined
```

item	“apple”	“orange”	“banana”
index	0	1	2

Setting an item

```
let fruits = [“apple”, “orange”, “banana”] ;

console.log( fruits[1] ); // orange
fruits[1] = “kiwi” ;
console.log( fruits[1] ); // kiwi
```

item	“apple”	“kiwi”	“banana”
index	0	1	2

Array length

Number of items in array

```
let fruits = [“apple”, “orange”, “banana”] ;
console.log(fruits.length) //3
```

item	“apple”	“orange”	“banana”
index	0	1	2

Zero-indexing quiz!

let names = [“John”, “Edwin”, “Nick”, “Sally”, “Billy”, “Matt”, “Edmond”, “Kenny”] ;

What is the index of:

- Matt
- John
- Kenny

Data Type: Object

Objects are similar to arrays, they allow us to store collections of data in a neat and organized way.

An object can be created with curly braces {...} with an optional list of properties.

A property is a "key: value" pair, where key is a string (also called a "property name"), and value can be anything.

Anatomy of an Object

object name

start object {

```
const user = {
        name: "John" ,
        age : 20 ,
        isAdmin : true
} ;
```

key:value pairs

separated by commas

} end object

```
// "user" object with
// key "name" store value "John"
// key "age" store value 20
// key "isAdmin" store value true
```

Getting value of key

```
let person = { name: “John” ,
               age: 20 ,
               food: [ “burger”, “pizza” ]
             } ;

console.log(person.name) ; // John
console.log(person[“name”]) ; // John

console.log(person.food) ; // [“burger”, “pizza”]
console.log(person[“food”]) ; // [“burger”, “pizza”]

console.log(person.food[1]) ; // pizza
console.log(person[“food”][1]) ; // pizza
```

Setting value of key

```
let person = { name: “John” ,
               age: 20 ,
               food: [ “burger”, “pizza” ]
             } ;
```

Change John’s favourite food from “pizza” to “durian” :

```
console.log(person.food[1]) ; // pizza
person.food[1] = “durian”
console.log(person.food[1]) ; // durian
```