

# **Web Application Development for Industrial Engineers**

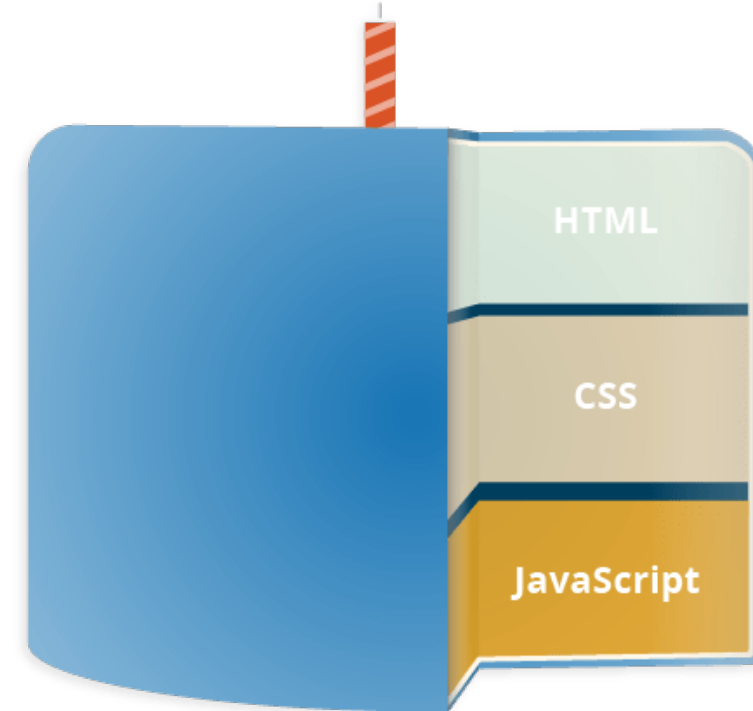
**การพัฒนาแอปพลิเคชันสำหรับวิศวกรอุตสาหกรรม**

# JavaScript

- JavaScript is a scripting or programming language
- Allows implementation of complex features on web pages.
  - Content updates
  - Animation
  - Interactive maps
  - Audio/video contents

# 3 Layers in Web Technology

- **HTML** : markup language
  - Defining structure
- **CSS** : stylesheet language
  - Apply styling to HTML content
- **JavaScript** : scripting language
  - Add dynamics to content



# HTML

```
<p>Player 1: Chris</p>
```

# CSS

```
p {  
  font-family: 'helvetica neue', helvetica, sans-serif;  
  letter-spacing: 1px;  
  text-transform: uppercase;  
  text-align: center;  
  border: 2px solid rgba(0, 0, 200, 0.6);  
  background: rgba(0, 0, 200, 0.3);  
  color: rgba(0, 0, 200, 0.6);  
  box-shadow: 1px 1px 2px rgba(0, 0, 200, 0.4);  
  border-radius: 10px;  
  padding: 3px 10px;  
  display: inline-block;  
  cursor: pointer;  
}
```

# JavaScript

```
const para = document.querySelector('p');  
  
para.addEventListener('click', updateName);  
  
function updateName() {  
    let name = prompt('Enter a new name');  
    para.textContent = 'Player 1: ' + name;  
}
```

- <https://codepen.io/nnnpoooh/pen/poWopXd>

# What just happened?

JavaScript allows

- Storing value inside a variable (user input).
- Performing operations on variables (joining text).
- Running code in response to certain `events` occurring on a web page (click event).
- Updating content shown in the page.

# Application Programming Interfaces (APIs)

- Extra functionality on top of client-side JavaScript language.
- APIs are ready-made sets of code building blocks for developers.



# Types of APIs

- Browser APIs
  - Functionalities built into web browsers
- Third Parties APIs
  - Functionalities built by vendors

# Browser APIs

- `DOM` (Document Object Model) API
  - Allows manipulation of HTML and CSS.
- `Geolocation` API
  - Retrieves geographical information.
- `Canvas` and `WebGL`
  - Allows creation of animated 2D and 3D graphics.
- `Audio` and `Video` APIs
  - Enables multimedia.

# Third Parties APIs

- Line APIs
- Facebook APIs

# Add JavaScript to the page

- Inline
- External file

# Inline

## HTML

```
<head>
  <script>
    //   JavaScript goes here
  </script>
</head>
<body>
  <button>Click me</button>
</body>
```

# Inline

## JavaScript

```
document.addEventListener('DOMContentLoaded', function () {  
  function createParagraph() {  
    let para = document.createElement('p');  
    para.textContent = 'You clicked the button!';  
    document.body.appendChild(para);  
  }  
  
  const buttons = document.querySelectorAll('button');  
  
  for (let i = 0; i < buttons.length; i++) {  
    buttons[i].addEventListener('click', createParagraph);  
  }  
});
```

## External file

- Create an `index.html` file with `<button>Click me</button>`
- Add `<script src="script.js" defer></script>` in `header` tag.
- Create `script.js`

## External file (cont.)

- Add

```
function createParagraph() {  
  let para = document.createElement('p');  
  para.textContent = 'You clicked the button!';  
  document.body.appendChild(para);  
}  
  
const buttons = document.querySelectorAll('button');  
  
for (let i = 0; i < buttons.length; i++) {  
  buttons[i].addEventListener('click', createParagraph);  
}
```



# Script loading strategies

- All the HTML on a page is loaded in the order in which it appears.
- Sometimes, your code won't work if the JavaScript is loaded and parsed *before* the HTML you are trying to do something to.

# Inline Script

- For the inline example, the loading is done through

```
document.addEventListener("DOMContentLoaded", function() {  
    ...  
});
```

- This is an event listener, which listens for the browser's DOMContentLoaded event.
- The JavaScript inside this block will not run until all html content is loaded.

# External file

- In the external example, the `defer` attribute tells the browser to continue downloading the HTML content once the `<script>` tag element has been reached.

```
<script src='script.js' defer></script>
```

- The code in `script.js` will not run until all html content is loaded.

# Comment

- Single line

```
// I am a comment
```

- Multi-line

```
/*  
  I am also  
  a comment  
*/
```

# Prototyping (ลองอะไรง่าย ๆ)

- Developer tools console
- Quokka (no browser API)

# Guess Game

[https://ie-software-dev.netlify.app/codes/t06\\_js/t03\\_guess\\_game/](https://ie-software-dev.netlify.app/codes/t06_js/t03_guess_game/)

# Functionality

- Let player enter a number.
- Check whether the number is correct.
- Inform the play if the guess is too high or too low.
- Stop the game after 10 trials.
- Player can reset the game after winning or gameover.

# HTML

```
<h1>Number guessing game</h1>

<p>Guess the number between 1 and 100.</p>

<div class="form">
  <label for="guessField">Enter a guess:</label>
  <input type="number" id="guessField" class="guessField" />
  <input type="submit" value="Submit guess" class="guessSubmit" />
</div>

<div class="resultParas">
  <p class="guesses"></p>
  <p class="lastResult"></p>
  <p class="lowOrHi"></p>
</div>
```



# Link with JavaScript

- Create `script.js`
- In HTML `<header>` add

```
<script src='script.js' defer></script>
```

# Add variables

```
let randomNumber = Math.floor(Math.random() * 100) + 1;
const guesses = document.querySelector('.guesses');
const lastResult = document.querySelector('.lastResult');
const lowOrHi = document.querySelector('.lowOrHi');
const guessSubmit = document.querySelector('.guessSubmit');
const guessField = document.querySelector('.guessField');
let guessCount = 1;
let resetButton;
```

# Variable

- Store data and reference
- Declare variable with `let`

## Variable (cont)

- From the script

```
let randomNumber = Math.floor(Math.random() * 100) + 1;
```

- The variable `randomNumber` is assigned a random number between 1 and 100, calculated using a mathematical algorithm.

# Constant

- Constants are also used to name values.
- Unlike variables, you can't change the value once set.
- Declare constant with `const`

## Constant (cont)

```
const guesses = document.querySelector('.guesses');  
const lastResult = document.querySelector('.lastResult');  
const lowOrHi = document.querySelector('.lowOrHi');
```

- These constants are each made to store a reference to the results paragraphs in our HTML.
- Constants store **references** to parts of our user interface.
  - The text inside some of these elements might change, but the reference does not change.

## Constant (cont)

```
const guessSubmit = document.querySelector('.guessSubmit');  
const guessField = document.querySelector('.guessField');
```

- The two constants store references to the form text input and submit button.
- They are used to control submitting the guess later on.

## Variable (cont)

```
let guessCount = 1;  
let resetButton;
```

- `guessCount` stores a guess count of `1`
  - Used to keep track of how many guesses the player has had
- `resetButton` stores a reference to a reset button
  - doesn't exist yet (but will later).



# Browser objects

- In JavaScript, most of the items you will manipulate in your code are `objects`.
- An `object` is a collection of related functionality stored in a single grouping.
- Let's focus on the built-in `objects` that your browser contains.
  - They allows you to do lots of *cool* things.

## Browser objects (cont)

- Get a reference to a browser object

```
const guessField = document.querySelector('.guessField');
```

- Do something with it (run in `script.js`, not console.)

```
guessField.focus();
```

- Insert value (has to be number due to `<input type="number" .../>` )

```
guessField.value = 1;
```

# Add content to a field

- Get a reference to `<p>` object

```
const guesses = document.querySelector('.guesses');
```

- Add content

```
guesses.textContent = 'Where is my paragraph?';
```

- Styling

```
guesses.style.backgroundColor = 'yellow';  
guesses.style.fontSize = '200%';  
guesses.style.padding = '10px';  
guesses.style.boxShadow = '3px 3px 6px black';
```

# Function

- Functions are reusable blocks of code that you can write once and run again and again
  - saving the need to keep repeating code all the time.
- For example, let put this code into `script.js`

```
function checkGuess() {  
    alert('I am a placeholder');  
}
```

- You can run `checkGuess()` from the console.

# Operators

- Try from the console.

Operator	Name	Example
+	Addition	6 + 9
-	Subtraction	20 - 15
*	Multiplication	3 \* 7
/	Division	10 / 5

# Text Operation

```
let name = 'Bingo';  
name;  
let hello = ' says hello!';  
hello;  
let greeting = name + hello;  
greeting;
```

# Shortcut

```
name += ' says hello!';
```

is the same as

```
name = name + ' says hello!';
```

# Comparison operator

Operator	Name	Example
<code>===</code>	(Strict) equality	<code>2 === 2</code>
<code>!==</code>	Non-equality	<code>2 !== 3</code>
<code>&lt;</code>	Less than	<code>2 &lt; 3</code>
<code>&gt;</code>	Greater than	<code>3 &gt; 2</code>



```

function checkGuess() {
  let userGuess = Number(guessField.value);
  if (guessCount === 1) {
    guesses.textContent = 'Previous guesses: ';
  }
  guesses.textContent += userGuess + ' ';

  if (userGuess === randomNumber) {
    lastResult.textContent = 'Congratulations! You got it right!';
    lastResult.style.backgroundColor = 'green';
    lowOrHi.textContent = '';
    setGameOver();
  } else if (guessCount === 10) {
    lastResult.textContent = '!!!GAME OVER!!!';
    lowOrHi.textContent = '';
    setGameOver();
  } else {
    lastResult.textContent = 'Wrong!';
    lastResult.style.backgroundColor = 'red';
    if (userGuess < randomNumber) {
      lowOrHi.textContent = 'Last guess was too low!';
    } else if (userGuess > randomNumber) {
      lowOrHi.textContent = 'Last guess was too high!';
    }
  }

  guessCount++;
  guessField.value = '';
  guessField.focus();
}

```

# Conditionals

```
if (guessCount === 1) {  
    guesses.textContent = 'Previous guesses: ';  
}
```

- The simplest form of conditional block starts with the keyword `if`
  - then some parentheses,
  - then some curly braces.
- Inside the parentheses we include a test.
  - If the test returns `true`, we run the code inside the curly braces.
  - If not, we don't

```
if (userGuess === randomNumber) {  
    // ...  
} else if (guessCount === 10) {  
    // ...  
} else {  
    // ...  
}
```

- The first `if(){ }` structure checks whether the user's guess is equal to the `randomNumber`.
- Then `else if(){ }` structure checks whether this turn is the user's last turn.
- Then `else { }` structure only run if neither of the other two tests returns `true`.
  - The player didn't guess right, but they have more guesses left.

```
if (userGuess === randomNumber) {  
    lastResult.textContent = 'Congratulations! You got it right!';  
    lastResult.style.backgroundColor = 'green';  
    lowOrHi.textContent = '';  
    setGameOver();  
}
```

- If the conditional test is `true` (player has won).
- Show the player a congratulations message with a green color
- Clear the contents of the Low/High guess information box
- Run a function called `setGameOver()` (*not yet defined*).

```
else if (guessCount === 10) {  
    lastResult.textContent = '!!!GAME OVER!!!';  
    lowOrHi.textContent = '';  
    setGameOver();  
}
```

- If the conditional test is `true` (user's last turn.)
- Show a game over message.
- Clear the contents of the Low/High guess information box.
- Run a function called `setGameOver()` (*not yet defined*).

```
else {  
    lastResult.textContent = 'Wrong!';  
    lastResult.style.backgroundColor = 'red';  
    if (userGuess < randomNumber) {  
        lowOrHi.textContent = 'Last guess was too low!';  
    } else if (userGuess > randomNumber) {  
        lowOrHi.textContent = 'Last guess was too high!';  
    }  
}
```

- The code will run when the player didn't guess right, but they have more guesses left.
- Perform another conditional test to check whether the guess was higher or lower than the answer
- Display a further message as appropriate to tell them higher or lower.

# Events

- We have a nicely implemented `checkGuess()` function
  - but it won't do anything.
- We want to call it when the `Submit guess`.
- We need an `event`.

# Events (cont)

- Events are things that happen in the browser
  - a button being clicked
  - a page loading
  - a video playing, etc.
- We can then response to an event by which the blocks of code ( `function` ) are run.
- The constructs that listen out for the event happening are called `event listeners`.
- The blocks of code ( `function` ) that run in response to the event firing are called `event handlers`.



# Add event listener

```
guessSubmit.addEventListener('click', checkGuess);
```

- We are adding an `event listener` to the `guessSubmit` button.
- This is a method that takes two input values (called `arguments`)
  - the type of event we are listening out for (in this case `click`) as a string
  - the code we want to run when the event occurs.
- Note that we don't need to specify the parentheses when writing it inside `addEventListener()`.

## Current bug

- If you guess the correct answer or run out of guesses, the game will break.
- This is because we've not yet defined the `setGameOver()` function that is supposed to be run once the game is over.

## setGameOver

```
function setGameOver() {  
    guessField.disabled = true;  
    guessSubmit.disabled = true;  
    resetButton = document.createElement('button');  
    resetButton.textContent = 'Start new game';  
    document.body.append(resetButton);  
    resetButton.addEventListener('click', resetGame);  
}
```

```
guessField.disabled = true;  
guessSubmit.disabled = true;
```

- Disable the form text input and button by setting their `disabled` properties to `true`.
- This is necessary, because if we didn't, the user could submit more guesses after the game is over, which would mess things up.

```
resetButton = document.createElement('button');  
resetButton.textContent = 'Start new game';  
document.body.append(resetButton);
```

- Generate a new <button> element.
- Set its text label to "Start new game".
- Add it to the bottom of our existing HTML.

```
resetButton.addEventListener('click', resetGame);
```

- Sets an event listener on our new button.
- When the button is clicked, a function called `resetGame()` is run.

**resetGame**

```
function resetGame() {  
    guessCount = 1;  
  
    const resetParas = document.querySelectorAll('.resultParas p');  
    for (let i = 0; i < resetParas.length; i++) {  
        resetParas[i].textContent = '';  
    }  
  
    resetButton.parentNode.removeChild(resetButton);  
  
    guessField.disabled = false;  
    guessSubmit.disabled = false;  
    guessField.value = '';  
    guessField.focus();  
  
    lastResult.style.backgroundColor = 'white';  
  
    randomNumber = Math.floor(Math.random() * 100) + 1;  
}
```



```
guessCount = 1;
```

- Puts the `guessCount` back down to 1.

```
const resetParas = document.querySelectorAll('.resultParas p');
for (let i = 0; i < resetParas.length; i++) {
  resetParas[i].textContent = '';
}
```

- Empties all the text out of the information paragraphs.
- We select all paragraphs inside `<div class="resultParas"></div>`.
- We loop through each one, setting their `textContent` to `' '` (an empty string).

## **for** Loops

- Allow you to keep running a piece of code over and over again
- Until a certain condition is met.

## for Loops (cont)

```
for (let i = 1; i < 21; i++) {  
  console.log(i);  
}
```

A `for` loop takes three input values (arguments):

- A starting value `i = 1`
  - You could replace the letter `i` with any name you like.
- A condition `i < 21`
  - The loop will keep going until `i` is no longer less than `21`.
  - When `i` reaches `21`, the loop will no longer run.
- An incrementor `i++`
  - Add `1` to `i` after every iteration.

## Back to the code

```
const resetParas = document.querySelectorAll('.resultParas p');  
for (let i = 0; i < resetParas.length; i++) {  
  resetParas[i].textContent = '';  
}
```

- This code creates a variable containing a list of all the paragraphs inside `<div class="resultParas">` using the `querySelectorAll()` method
- Then it loops through each one, removing the text content of each.

```
resetButton.parentNode.removeChild(resetButton);
```

- Removes the reset button from our code.

```
guessField.disabled = false;  
guessSubmit.disabled = false;  
guessField.value = '';  
guessField.focus();
```

- Enables the form elements.
- Empties and focuses the text field.

```
lastResult.style.backgroundColor = 'white';
```

Removes the background color from the lastResult paragraph.



```
randomNumber = Math.floor(Math.random() * 100) + 1;
```

Generates a new random number so that you are not just guessing the same number again!