Topic 1: Variables in JavaScript

المتغيرات هي الأساس في البرمجة - فهي تخزن البيانات وتسمح لنا بالتعامل معها بطريقة مرنة

Variable Declaration Types

[أنواع إعلان المتغيرات في JavaScript]

```
// 1. var - يمكن إعادة الإعلان وإعادة التعيين
var name = "Ahmed";
var name = "Mohammed"; // ☑ لا يوجد خطأ
name = "Ali"; // ☑ يمكن إعادة التعيين

// 2. let - يمكن إعادة التعيين فقط
let age = 25;
age = 26; // ☑ يمكن إعادة التعيين
// let age = 27; // ❌ خطأ - لا يمكن إعادة الإعلان

// 3. const - ثابت لا يمكن تغييره
const PI = 3.14;
// PI = 3.15; // ❌ خطأ - لا يمكن إعادة التعيين
// const PI = 3.16; // ❌ خطأ - لا يمكن إعادة الإعلان
```

- var: قديم وغير آمن، يمكن إعادة الإعلان والتعيين في أي مكان - let: حديث وآمن، يمكن إعادة التعيين فقط - const: ثابت، لا يمكن تغييره بعد التعيين الأولي

Variable Scope

[نطاق المتغيرات - أين يمكن استخدامها]

```
// Global scope - متغير عام
var globalVar = "I'm global";

function testScope() {
  // Function scope - متغير محلي في الدالة
  let localVar = "I'm local";
  console.log(globalVar); // ☑ يمكن الوصول للمتغير العام
  console.log(localVar);  // ☑ يمكن الوصول للمتغير المحلي
}

// console.log(localVar); // ❌ خطأ - لا يمكن الوصول للمتغير المحلي خارج الدالة
```

نطاق المتغير يحدد أين يمكن استخدامه. المتغيرات العامة متاحة في كل مكان، والمحلية متاحة فقط داخل الدالة أو الكتلة التي تم إعلانها فيها

Topic 2: Conditional Statements (if-else)

العبارات الشرطية تسمح لنا بتنفيذ كود مختلف بناءً على شروط معينة

Basic if-else Structure

[البنية الأساسية للعبارات الشرطية]

```
// Simple if statement
let score = 85;

if (score >= 90) {
  console.log("Excellent!");
} else if (score >= 80) {
  console.log("Very Good!");
} else if (score >= 70) {
  console.log("Good!");
} else {
  console.log("Needs Improvement");
}
// Output: Very Good!
```

يتحقق من الشروط بالترتيب. إذا كان الشرط الأول صحيح، يتم تنفيذ الكود المطلوب. وإلا ينتقل للشرط التالي if-else

Comparison Operators

[مشغلات المقارنة المستخدمة في الشروط]

```
let a = 10;
let b = "10";

// Equality operators
console.log(a == b);   // true (compares values only)
console.log(a === b);  // false (compares values AND types)

// Inequality operators
console.log(a != b);   // false
console.log(a !== b);  // true

// Comparison operators
console.log(a > 5);    // true
console.log(a >= 10);  // true
console.log(a < 15);   // true
console.log(a <= 10);  // true
```

تقارن القيم فقط، بينما === تقارن القيم والأنواع معاً. يُنصح دائماً باستخدام === للدقة ==

Logical Operators

[المشغلات المنطقية لدمج الشروط]

```
let age = 25;
let hasLicense = true;

// AND operator (&&)
if (age >= 18 && hasLicense) {
  console.log("Can drive!");
}

// OR operator (||)
if (age < 18 || !hasLicense) {
  console.log("Cannot drive!");
}

// NOT operator (!)
if (!hasLicense) {
  console.log("Need to get license first!");
}
```

تتطلب أن يكون كلا الشرطين صحيحين، || تتطلب أن يكون أحد الشرطين صحيحاً على الأقل، ! تعكس قيمة الشرط &&

Topic 3: Functions

الدوال هي كتل من الكود قابلة لإعادة الاستخدام، تنفذ مهمة محددة

Function Declaration

[إعلان الدوال - الطريقة التقليدية]

```
// Function declaration
function greet(name) {
  return "Hello, " + name + "!";
}

// Function expression
const greet2 = function(name) {
  return "Hello, " + name + "!";
};

// Arrow function (ES6)
const greet3 = (name) => {
  return "Hello, " + name + "!";
};

// Arrow function with implicit return
const greet4 = name => "Hello, " + name + "!";

console.log(greet("Marwan")); // Output: Hello, Marwan!
```

هناك ثلاث طرق لإنشاء الدوال: الإعلان التقليدي، التعبير عن الدالة، والدالة السهمية. جميعها تؤدي نفس النتيجة

Function Parameters and Return Values

[معاملات الدالة وقيم الإرجاع]

```
// Function with multiple parameters
function calculateArea(width, height) {
  return width * height;
}

// Function with default parameters
function greetUser(name = "Guest", age = 18) {
  return `Hello ${name}, you are ${age} years old`;
}

// Function returning multiple values (as object)
function getUserInfo() {
  return {
    name: "Ahmed",
    age: 25,
    city: "Cairo"
  };
}

console.log(calculateArea(5, 3)); // Output: 15
console.log(greetUser()); // Output: Hello Guest, you are 18 years old
console.log(greetUser("Marwan", 20)); // Output: Hello Marwan, you are 20 years
old

const user = getUserInfo();
console.log(user.name); // Output: Ahmed
```

الدوال يمكن أن تأخذ معاملات متعددة، ويمكن تعيين قيم افتراضية. يمكن أن ترجع قيمة واحدة أو كائن يحتوي على قيم متعددة

Topic 4: DOM Selectors (Detailed Guide)

JavaScript باستخدام HTML هي أدوات قوية للتعامل مع عناصر DOM محددات

1. getElementById() Selector

[اختيار عنصر باستخدام المعرف الفريد]

```
// Basic usage
const header = document.getElementById('main-header');

// Safe selection with error handling
function safeGetElementById(id) {
  const element = document.getElementById(id);
  if (!element) {
```

```
    console.warn(`No element found with id: ${id}`);
    return null;
  }
  return element;
}
```

- أسرع وأكثر كفاءة من محددات أخرى - يفترض وجود معرف فريد في المستند - يرجع العنصر الأول المطابق أو null

2. querySelector() Selector

[محدد متعدد الاستخدامات يدعم محددات CSS]

```
// Select first matching element
const firstButton = document.querySelector('button');
const specificButton = document.querySelector('.primary-btn');
const inputByAttribute = document.querySelector('input[name="username"]');

// Nested selections
const firstParagraphInDiv = document.querySelector('div p');
```

- يرجع أول عنصر مطابق فقط - مرن ودقيق في الاختيار - CSS يدعم جميع محددات

3. querySelectorAll() Selector

[اختيار جميع العناصر المطابقة]

```
// Select all paragraphs
const allParagraphs = document.querySelectorAll('p');

// Select elements with specific class
const highlightedElements = document.querySelectorAll('.highlight');

// Iterate through selected elements
allParagraphs.forEach((paragraph, index) => {
  paragraph.textContent = `Paragraph ${index + 1}`;
});
```

- يرجع NodeList للتعامل مع العناصر - مفيد للعمليات الجماعية forEach بجميع العناصر المطابقة - يمكن استخدام

4. getElementsByClassName() Selector

[اختيار العناصر بواسطة اسم الفئة]

```
// Select elements by class name
const errorMessages = document.getElementsByClassName('error');

// Convert to array for more operations
```

```
const errorArray = Array.from(errorMessages);
errorArray.forEach(msg => {
  msg.style.color = 'red';
});
```

- يرجع HTMLCollection حية - التحديث التلقائي عند تغيير المستند - يحتاج التحويل للمصفوفة لبعض العمليات

5. getElementsByTagName() Selector

[اختيار العناصر بواسطة اسم العلامة]

```
// Select all div elements
const divElements = document.getElementsByTagName('div');

// Select first div
const firstDiv = divElements[0];

// Count elements
console.log(`Number of divs: ${divElements.length}`);
```

- يرجع جميع العناصر بعلامة HTML محددة - مفيد للعمليات على نوع معين من العناصر

Topic 5: appendChild Method

appendChild تضيف عنصر جديد كطفل للعنصر المحدد في نهاية قائمة الأطفال

Basic appendChild Usage

[الاستخدام الأساسي لـ appendChild]

```
// Create a new element
const newParagraph = document.createElement('p');
newParagraph.textContent = 'This is a new paragraph';

// Add it to the body
document.body.appendChild(newParagraph);

// Create and add multiple elements
const container = document.getElementById('container');

const title = document.createElement('h2');
title.textContent = 'New Section';

const description = document.createElement('p');
description.textContent = 'This is the description';

container.appendChild(title);
container.appendChild(description);
```

تضيف العنصر في نهاية العنصر الأب. إذا كان العنصر موجود بالفعل في مكان آخر، سيتم نقله للموقع appendChild الجديد

Creating Complex Elements

[appendChild إنشاء عناصر معقدة باستخدام]

```
// Create a complete card element
function createCard(title, content, imageUrl) {
  const card = document.createElement('div');
  card.className = 'card';

  // Create card image
  const img = document.createElement('img');
  img.src = imageUrl;
  img.alt = title;
  img.className = 'card-image';

  // Create card content
  const cardContent = document.createElement('div');
  cardContent.className = 'card-content';

  const cardTitle = document.createElement('h3');
  cardTitle.textContent = title;

  const cardText = document.createElement('p');
  cardText.textContent = content;

  // Assemble the card
  cardContent.appendChild(cardTitle);
  cardContent.appendChild(cardText);

  card.appendChild(img);
  card.appendChild(cardContent);

  return card;
}

// Use the function
const cardContainer = document.getElementById('cards');
const newCard = createCard('Product Title', 'Product description here',
'product.jpg');
cardContainer.appendChild(newCard);
```

appendChild يمكن إنشاء عناصر معقدة ببناءها خطوة بخطوة وإضافة كل جزء باستخدام

Topic 6: insertBefore Method

insertBefore تضيف عنصر جديد قبل عنصر محدد في قائمة الأطفال

Basic insertBefore Usage

[الاستخدام الأساسي لـ insertBefore]

```
// Create a new element
const newItem = document.createElement('li');
newItem.textContent = 'New List Item';

// Get the reference element (where to insert before)
const referenceElement = document.getElementById('second-item');

// Get the parent element
const parentList = document.getElementById('my-list');

// Insert the new element before the reference element
parentList.insertBefore(newItem, referenceElement);
```

يحتاج لعنصرين: العنصر المراد إضافته، والعنصر المرجعي الذي سيتم الإدراج قبله insertBefore

Inserting at Specific Positions

[الإدراج في مواقع محددة]

```
// Insert at the beginning (before first child)
const firstItem = document.createElement('li');
firstItem.textContent = 'First Item';
const list = document.getElementById('my-list');

if (list.firstChild) {
  list.insertBefore(firstItem, list.firstChild);
} else {
  list.appendChild(firstItem); // If list is empty
}

// Insert at specific position
function insertAtPosition(parent, newElement, position) {
  const children = parent.children;

  if (position >= children.length) {
    // Insert at the end
    parent.appendChild(newElement);
  } else {
    // Insert at specific position
    parent.insertBefore(newElement, children[position]);
  }
}

// Usage
const newDiv = document.createElement('div');
```

```
newDiv.textContent = 'Inserted at position 2';
insertAtPosition(container, newDiv, 2);
```

لإدراج العناصر في بداية القائمة أو في موقع محدد بناءً على الفهرس insertBefore يمكن استخدام

Dynamic Insertion Examples

[أمثلة على الإدراج الديناميكي]

```
// Insert new row in table
function addTableRow(tableId, rowData) {
  const table = document.getElementById(tableId);
  const tbody = table.querySelector('tbody');

  const newRow = document.createElement('tr');

  rowData.forEach(cellData => {
    const cell = document.createElement('td');
    cell.textContent = cellData;
    newRow.appendChild(cell);
  });

  // Insert after header row
  const headerRow = tbody.querySelector('tr');
  tbody.insertBefore(newRow, headerRow.nextSibling);
}

// Insert navigation item
function addNavItem(text, href) {
  const nav = document.getElementById('navigation');
  const newItem = document.createElement('a');
  newItem.href = href;
  newItem.textContent = text;
  newItem.className = 'nav-item';

  // Insert before the last item (if it's a "Contact" link)
  const lastItem = nav.lastElementChild;
  if (lastItem && lastItem.textContent === 'Contact') {
    nav.insertBefore(newItem, lastItem);
  } else {
    nav.appendChild(newItem);
  }
}
```

مفيد جداً عند الحاجة لإدراج عناصر في مواقع محددة بناءً على شروط معينة insertBefore

---

Topic 7: classList Methods (Comprehensive Guide)

توفر طرق سهلة وقوية للتعامل مع الفئات في العناصر classList

1. add() Method

[إضافة فئة أو فئات جديدة]

```
const element = document.querySelector('.box');

// Add single class
element.classList.add('highlight');

// Add multiple classes
element.classList.add('active', 'visible', 'animated');
```

- يضيف فئة واحدة أو أكثر - لا يؤثر إذا كانت الفئة موجودة بالفعل - مفيد للتغييرات الديناميكية

2. remove() Method

[إزالة فئة أو فئات]

```
// Remove single class
element.classList.remove('inactive');

// Remove multiple classes
element.classList.remove('hidden', 'disabled', 'old-style');
```

- يزيل فئة واحدة أو أكثر - لا يؤثر إذا كانت الفئة غير موجودة - مفيد لإلغاء الحالات

3. toggle() Method

[تبديل وجود الفئة]

```
// Basic toggle
element.classList.toggle('active');

// Force add/remove with second parameter
element.classList.toggle('visible', true);  // Always add
element.classList.toggle('hidden', false); // Always remove
```

- يضيف الفئة إذا كانت غير موجودة - يزيل الفئة إذا كانت موجودة - يدعم التحكم الإجباري

4. contains() Method

[التحقق من وجود فئة]

```
// Check if element has a class
if (element.classList.contains('highlight')) {
  console.log('Element is highlighted');
```

```
}

// Conditional logic
const isActive = element.classList.contains('active');
const isVisible = element.classList.contains('visible');
```

- يرجع true أو false - يستخدم في الشروط المنطقية - مفيد للتحقق قبل العمليات

5. replace() Method

[استبدال فئة بأخرى]

```
// Replace old class with new class
element.classList.replace('old-style', 'new-style');

// Safe replacement with check
if (element.classList.contains('old-style')) {
  element.classList.replace('old-style', 'new-style');
}
```

- يستبدل فئة واحدة بأخرى - يفيد في التحديثات الهيكلية - يتطلب وجود الفئة القديمة

6. length Property

[عدد الفئات الحالية]

```
// Get number of classes
const classCount = element.classList.length;
console.log(`Element has ${classCount} classes`);

// Iterate through classes
for (let i = 0; i < element.classList.length; i++) {
  console.log(element.classList.item(i));
}
```

- يعيد عدد الفئات - مفيد للتحليل والتحقق - يمكن استخدامه للتكرار

Best Practices

1. استخدم classList بدلاً من manipulateClassName مباشرة
2. تحقق من وجود الفئات قبل العمليات
3. استخدم toggle للحالات التبديلية
4. كن حذراً مع العمليات المتكررة
5. استفد من الخصائص للتحليل

Practical Exercise

تمرين عملي يجمع جميع المفاهيم التي تم تعلمها: إنشاء قائمة تفاعلية مع إمكانية الإضافة والحذف والتحرير

```
// Complete interactive list application
class InteractiveList {
  constructor(containerId) {
    this.container = document.getElementById(containerId);
    this.items = [];
    this.init();
  }

  init() {
    // Create list structure
    this.list = document.createElement('ul');
    this.list.className = 'interactive-list';

    // Create input and button
    this.input = document.createElement('input');
    this.input.type = 'text';
    this.input.placeholder = 'Enter item text...';
    this.input.className = 'list-input';

    this.addButton = document.createElement('button');
    this.addButton.textContent = 'Add Item';
    this.addButton.className = 'add-btn';
    this.addButton.addEventListener('click', () => this.addItem());

    // Add elements to container
    this.container.appendChild(this.input);
    this.container.appendChild(this.addButton);
    this.container.appendChild(this.list);
  }

  addItem() {
    const text = this.input.value.trim();
    if (!text) return;

    const item = this.createListItem(text);
    this.list.appendChild(item);
    this.items.push(text);
    this.input.value = '';
  }

  createListItem(text) {
    const li = document.createElement('li');
    li.className = 'list-item';

    const textSpan = document.createElement('span');
    textSpan.textContent = text;
    textSpan.className = 'item-text';

    const editBtn = document.createElement('button');
    editBtn.textContent = 'Edit';
    editBtn.className = 'edit-btn';
```

```
    editBtn.addEventListener('click', () => this.editItem(li, textSpan));

    const deleteBtn = document.createElement('button');
    deleteBtn.textContent = 'Delete';
    deleteBtn.className = 'delete-btn';
    deleteBtn.addEventListener('click', () => this.deleteItem(li));

    li.appendChild(textSpan);
    li.appendChild(editBtn);
    li.appendChild(deleteBtn);

    return li;
  }

  editItem(li, textSpan) {
    const currentText = textSpan.textContent;
    const input = document.createElement('input');
    input.type = 'text';
    input.value = currentText;
    input.className = 'edit-input';

    const saveBtn = document.createElement('button');
    saveBtn.textContent = 'Save';
    saveBtn.className = 'save-btn';
    saveBtn.addEventListener('click', () => {
      const newText = input.value.trim();
      if (newText) {
        textSpan.textContent = newText;
        li.classList.remove('editing');
        li.replaceChild(textSpan, input);
        li.insertBefore(textSpan, li.firstChild);
      }
    });

    li.classList.add('editing');
    li.replaceChild(input, textSpan);
    li.insertBefore(input, li.firstChild);
    li.insertBefore(saveBtn, li.children[1]);

    input.focus();
  }

  deleteItem(li) {
    li.classList.add('deleting');
    setTimeout(() => {
      this.list.removeChild(li);
    }, 300);
  }
}

// Usage
const listApp = new InteractiveList('app-container');
```

Key Concepts Covered

المفاهيم الرئيسية التي تم تغطيتها في هذا الدليل الشامل:

1. **Variables**: `var`, `let`, `const` - أنواع المتغيرات ونطاقاتها
2. **Conditionals**: `if-else`, comparison operators, logical operators - العبارات الشرطية
3. **Functions**: declarations, expressions, arrow functions, parameters - الدوال وطرق إنشائها
4. **DOM Selectors**: `getElementById`, `querySelector`, `querySelectorAll` - الوصول للعناصر
5. **appendChild**: إضافة عناصر جديدة في نهاية العنصر الأب
6. **insertBefore**: إدراج عناصر في مواقع محددة
7. **classList**: إدارة classes للعناصر بطريقة تفاعلية

Advanced: Best Practices

Performance Optimization

نصائح لتحسين الأداء عند التعامل مع DOM

Technique	When to Use	Example
Cache selectors	Multiple operations on same element	`const element = document.getElementById('id');`
Batch DOM changes	Multiple modifications	Use `DocumentFragment`
Event delegation	Many similar elements	Attach event to parent
Avoid inline styles	Frequent style changes	Use `classList` instead

لتحسين الأداء event delegation تخزين العناصر في متغيرات بدلاً من البحث عنها مراراً، تجميع التغييرات، واستخدام

Error Handling

التعامل مع الأخطاء المحتملة في الكود

```
// Safe element selection
function safeGetElement(id) {
  const element = document.getElementById(id);
  if (!element) {
    console.error(`Element with id '${id}' not found`);
    return null;
  }
  return element;
}

// Safe classList operations
function safeToggleClass(element, className) {
  if (element && element.classList) {
    element.classList.toggle(className);
  } else {
    console.error('Invalid element or element has no classList');
```

```
  }
}

// Try-catch for DOM operations
function safeAppendChild(parent, child) {
  try {
    if (parent && child) {
      parent.appendChild(child);
    } else {
      throw new Error('Parent or child element is null');
    }
  } catch (error) {
    console.error('Error appending child:', error.message);
  }
}
```