# JavaScript Short Notes

## 1) Document and Window Object

### Window Object:

- The global object in browser environment
- Represents the browser window/tab
- Contains methods like `alert()`, `setTimeout()`, `location`, `navigator`
- All global variables and functions are properties of window

### Document Object:

- Represents the HTML document loaded in the window
- Property of the window object ( `window.document` )
- Provides methods to access and manipulate HTML elements
- Entry point to the DOM

## 2) DOM (Document Object Model)

- Tree-like representation of HTML document structure
- Each HTML element is a node in the tree
- Allows JavaScript to access, modify, add, or delete HTML elements
- Common methods: `getElementById()`, `querySelector()`, `createElement()`, `appendChild()`

## 3) Events

### Common Event Types:

- `onclick` - Fires when element is clicked
- `onload` - Fires when page/image finishes loading
- `onmouseover` - Fires when mouse pointer moves over element
- `onkeyup` - Fires when key is released
- Other events: `onchange`, `onsubmit`, `onfocus`, `onblur`

### Event Handling:

```
element.onclick = function() { /* code */ };
element.addEventListener('click', function() { /* code */ });
```

## 4) Conditional Statements

### if:

```
if (condition) { /* code */ }
```

### if-else:

```
if (condition) { /* code */ }
else { /* code */ }
```

### multiple if:

```
if (condition1) { /* code */ }
if (condition2) { /* code */ }
if (condition3) { /* code */ }
```

### nested if:

```
if (condition1) {
  if (condition2) { /* code */ }
}
```

### switch case:

```
switch(expression) {
  case value1: /* code */ break;
  case value2: /* code */ break;
  default: /* code */
}
```

### Ternary operator:

```
condition ? valueIfTrue : valueIfFalse;
```

## 5) Loops

### for loop:

```
for (let i = 0; i < 10; i++) { /* code */ }
```

### while loop:

```
while (condition) { /* code */ }
```

### do-while loop:

```
do { /* code */ } while (condition);
```

### break & continue:

- `break` - Exits the loop entirely
- `continue` - Skips current iteration, continues with next

## 6) Functions

### Named function:

```
function myFunction(params) { return value; }
```

### Anonymous function:

```
const myFunc = function(params) { return value; };
```

### Arrow function:

```
const myFunc = (params) => { return value; };
// Shorter: const myFunc = params => value;
```

## 7) Function Scope and Block Scope

### Function Scope:

- Variables declared with `var` are function-scoped
- Accessible anywhere within the function

### Block Scope:

- Variables declared with `let` and `const` are block-scoped
- Only accessible within the block `{}` where they're defined

## 8) Hoisting

- JavaScript's behavior of moving declarations to the top
- `var` declarations are hoisted (initialized as undefined)
- Function declarations are fully hoisted
- `let` and `const` are hoisted but in "temporal dead zone" (cannot access before declaration)

## 9) Event Bubbling

- Events propagate from child element up to parent elements
- When event occurs on child, it triggers on parent elements too
- Can be stopped using `event.stopPropagation()`

- Event capturing is opposite direction (parent to child)

## Array

### for-of loop:

```
for (const item of array) { /* code */ }
```

- Iterates over array values
- Cannot access index directly

## Object

### for-in loop:

```
for (const key in object) { /* code */ }
```

- Iterates over object keys/properties
- Access value using `object[key]`

## Asynchronous Programming

### Callback:

```
function doSomething(callback) {
  // async operation
  callback(result);
}
```

### Promise:

```
fetch(url)
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

### Async/Await:

```
async function fetchData() {
  try {
    const response = await fetch(url);
    const data = await response.json();
  } catch(error) { /* handle */ }
}
```

## Import & Export

### Export:

```
export const myVar = 10;
export function myFunc() {}
export default MyComponent;
```

### Import:

```
import { myVar, myFunc } from './module.js';
import MyComponent from './component.js';
```

---

## Advanced Concepts

### Higher Order Function:

- Function that takes another function as argument OR returns a function
- Examples: `map()` , `filter()` , `reduce()`

### Closure:

- Function that has access to variables from outer function scope
- Inner function "remembers" outer function's variables even after outer function has returned

### Curry:

- Technique of transforming function with multiple arguments into sequence of functions each taking single argument
- Example: `f(a, b, c)` becomes `f(a)(b)(c)`