# JavaScript Interview Q&A: Key Concepts & Handwritten Notes (JS 101)

IT Workshop (Maulana Abul Kalam Azad University of Technology)

# JavaScript



---

1. What is the difference between null and undefined in JavaScript?

Answer:
- null: Represents an intentionally empty or non-existent value. It is an object type in JavaScript.
- undefined: Represents a variable that has been declared but not assigned any value.

Example:
```javascript
let a; // declared but not assigned
console.log(a); // undefined

let b = null; // explicitly assigned as null
console.log(b); // null
```

---

2. What is a closure in JavaScript?

Answer:
A closure is a function that retains access to its lexical scope (the environment in which it was created) even after the outer function

has finished executing. Closures allow a function to "remember" the variables of its outer function.

Example:

```javascript
function outer() {
let counter = 0;

return function inner() {
counter++;
console.log(counter);
};
}

const increment = outer();
increment(); // 1
increment(); // 2
```

---

3. What are JavaScript data types?

Answer:

JavaScript has primitive and non-primitive data types.

- Primitive types:
- Number: Represents numeric values (123, 5.67).
- String: Represents textual data ("hello", 'world').
- Boolean: Represents true or false.

- Null: Represents the intentional absence of any value.
- Undefined: A variable that has been declared but not initialized.
- Symbol: A unique, immutable value often used as object property keys.
- BigInt: A large integer type.

- Non-primitive types:
- Object: A collection of key-value pairs.

Example:
```javascript
let num = 42; // Number
let name = "Alice"; // String
let isActive = true; // Boolean
let user = null; // Null
let obj = { age: 25 }; // Object
```

---

4. What is event delegation in JavaScript?

Answer:
Event delegation is a technique in which a single event listener is added to a parent element rather than attaching event listeners to individual child elements. The event listener on the parent element listens for events that bubble up from child elements. This improves performance and allows dynamically added elements to inherit the event handler.

Example:
```javascript
document.querySelector('#parent').addEventListener('click',
function(event) {
if (event.target && event.target.matches('button.classname')) {
console.log('Button clicked!');
}
});
```

---

5. What is the difference between == and === in JavaScript?

Answer:
- == (Equality Operator): Compares values after type coercion (conversion of one type to another).
- === (Strict Equality Operator): Compares both values and types without performing any type coercion.

Example:
```javascript
console.log(5 == '5'); // true, because '5' is coerced to 5
console.log(5 === '5'); // false, because types are different (number vs string)
```

---

6. What are promises in JavaScript?

Answer:
A promise is an object that represents the eventual completion or failure of an asynchronous operation. It can be in three states: pending, resolved (fulfilled), or rejected.

Example:
```javascript
let promise = new Promise(function(resolve, reject) {
let success = true;
if (success) {
resolve("Operation was successful!");
} else {
reject("Operation failed.");
}
});

promise.then(function(result) {
console.log(result); // "Operation was successful!"
}).catch(function(error) {
console.log(error); // "Operation failed."
});
```

---

7. What are arrow functions in JavaScript?

Answer:
Arrow functions are a shorter syntax for writing functions in JavaScript. They do not have their own `this` context and inherit

the `this` value from the enclosing execution context.

Example:

```javascript
// Regular function
const add = function(a, b) {
return a + b;
};

// Arrow function
const addArrow = (a, b) => a + b;

console.log(add(2, 3)); // 5
console.log(addArrow(2, 3)); // 5
```

---

8. What is the this keyword in JavaScript?

Answer:

The `this` keyword refers to the context in which a function is called. Its value can vary depending on how a function is called:

- In a regular function, `this` refers to the global object (or undefined in strict mode).
- In an object method, `this` refers to the object the method is called on.
- In an arrow function, `this` refers to the enclosing lexical context.

Example:

```javascript
function regularFunction() {
console.log(this);
}

const person = {
name: 'John',
greet: function() {
console.log(this.name);
}
};

regularFunction(); // In strict mode, `this` is undefined
person.greet(); // 'John'
```

---

9. What is a JavaScript closure?

Answer:

A closure is a function that retains access to the variables from its lexical scope even after the outer function has returned. Closures are useful for data privacy and creating functions with persistent state.

Example:

```javascript
function outer() {
```

```
let count = 0;
return function inner() {
count++;
console.log(count);
};
}

const counter = outer();
counter(); // 1
counter(); // 2
```

---

10. What is hoisting in JavaScript?

Answer:
Hoisting refers to the behavior of JavaScript where variable and function declarations are moved to the top of their containing scope during the compilation phase. However, only the declarations are hoisted, not the assignments.

Example:
```javascript
console.log(x); // undefined, because only declaration is hoisted
var x = 5;

foo(); // "Hello, world!", function declarations are hoisted
function foo() {
console.log("Hello, world!");
```

```
}

bar(); // Error: bar is not a function
var bar = function() {
console.log("This won't run.");
};
```

---

11. What is the difference between var, let, and const in JavaScript?

Answer:
- var: Function-scoped or globally-scoped, can be re-declared and updated.
- let: Block-scoped, can be updated but not re-declared within the same scope.
- const: Block-scoped, cannot be updated or re-declared. Used for constant values.

Example:
```javascript
var a = 10;
let b = 20;
const c = 30;

a = 15; // allowed
b = 25; // allowed
c = 35; // Error: Assignment to constant variable
```

```

---

12. What is the event loop in JavaScript?

Answer:
The event loop is a mechanism that allows JavaScript to handle asynchronous operations (like setTimeout, fetch, etc.). It operates by constantly checking the call stack (where functions are executed) and the message queue (where asynchronous tasks are placed). If the call stack is empty, the event loop picks up tasks from the queue and executes them.

Example:
```javascript
console.log("Start");

setTimeout(function() {
console.log("This is async");
}, 0);

console.log("End");
```

Output:
```
Start
End
This is async
```
```

```

---

13. What is the spread operator in JavaScript?

Answer:
The spread operator (...) is used to unpack or spread elements from an array or object into individual elements.

Example:
```javascript
// For arrays
let arr1 = [1, 2, 3];
let arr2 = [...arr1, 4, 5];
console.log(arr2); // [1, 2, 3, 4, 5]

// For objects
let obj1 = { a: 1, b: 2 };
let obj2 = { ...obj1, c: 3 };
console.log(obj2); // { a: 1, b: 2, c: 3 }
```

---

14. What is the setTimeout function in JavaScript?

Answer:
The setTimeout() function allows you to delay the execution of a function or a piece of code for a specified amount of time (in
```

milliseconds).

Example:

```javascript
console.log("Start");

setTimeout(() => {
console.log("Executed after 2 seconds");
}, 2000);

console.log("End");
```

---

15. What are template literals in JavaScript?

Answer:

Template literals are string literals allowing embedded expressions and multi-line strings. They are enclosed in backticks (\` \`) instead of quotes.

Example:

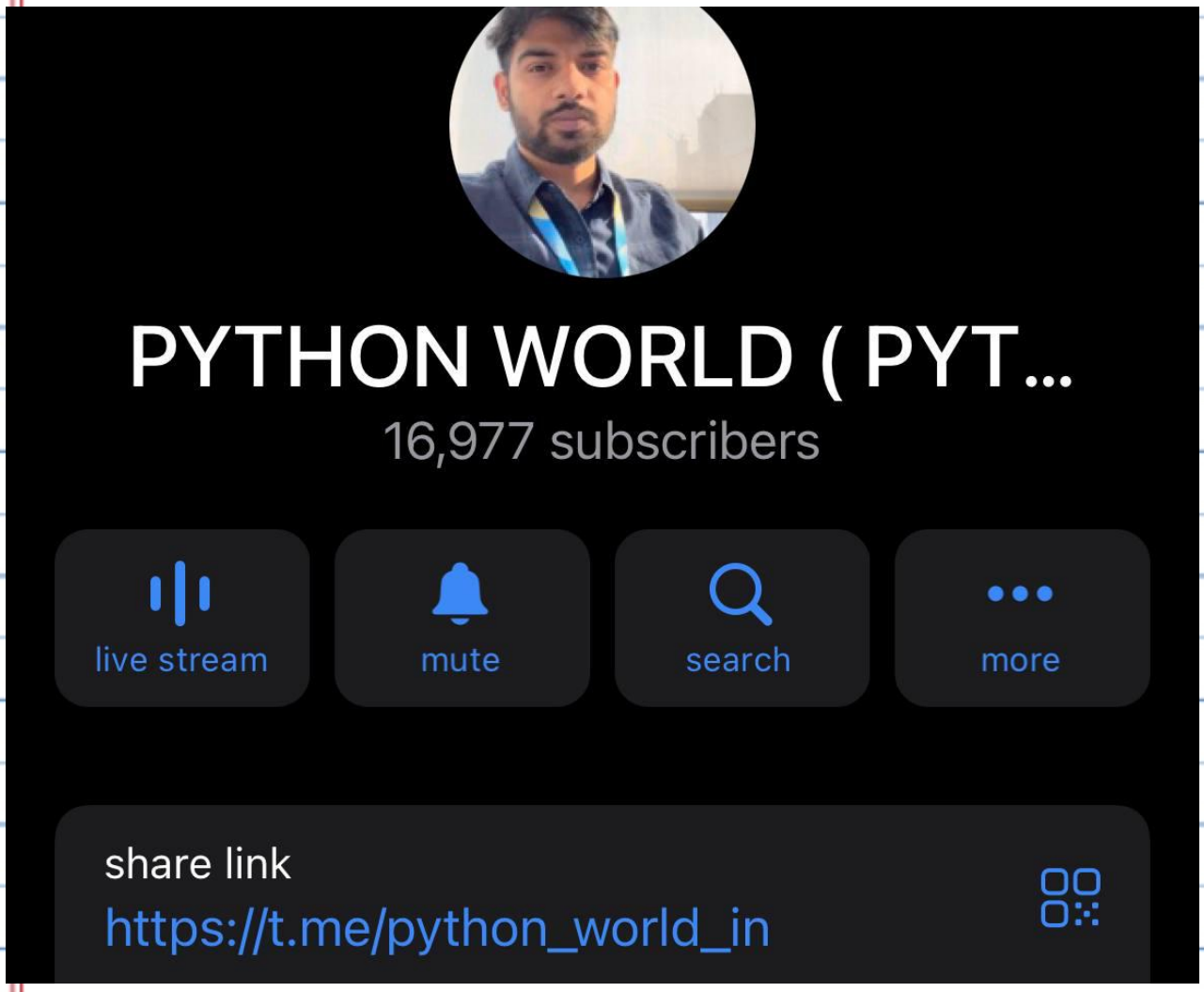
```javascript
let name = "John";
let message = `Hello, ${name}! Welcome.`;
console.log(message); // "Hello, John! Welcome."

// Multi-line string
let multiline = `This is a
```

multi-line string.`;
console.log(multiline);
```

---

Want Handwritten Notes, Project , video Lecture whatsapp on 9771025313, 6377362399


```