perplexity

# stack vs heap for datatypes in js, where addresses get stored

In JavaScript, **primitive types** and **reference types** are stored differently in memory, using the stack and the heap, respectively:

- **Stack Memory:**
  - Used for **primitive data types** (Number, String, Boolean, null, undefined, Symbol, BigInt) [1] [2] [3] [4].
  - The actual values are stored directly on the stack.
  - Each assignment or function call pushes values on the stack in a last-in, first-out (LIFO) manner, making access fast and memory management automatic [1] [2] [3].
  - Example:

```
let a = 10; // a is stored directly in the stack
let b = a;  // b gets a copy of a's value (independent in memory)
```

- **Heap Memory:**
  - Used for **reference types** — objects, arrays, and functions [1] [2] [3] [4].
  - The **actual data** is stored in the heap, which is a larger area designed for dynamic and complex data structures.
  - The **stack holds only a reference (address-like pointer) to the object/array** in the heap, not the object itself [1] [2] [4].
  - Example:

```
let obj1 = {name: "Alex"}; // obj1 holds a reference (on the stack) to data in th
let obj2 = obj1;  // obj2 is another variable on the stack pointing to the same h
// Mutations via obj2 are seen in obj1, since the reference is the same
```

**Summary Table:**

| Memory Type | Data Types | What is Stored in Stack | What is Stored in Heap |
|---|---|---|---|
| Stack | Primitive | Actual value | (N/A) |
| Heap | Reference (obj, array, func) | Reference (pointer) | Actual object/array/function values |

**Where addresses get stored:**

- For objects/arrays (reference types), the **reference** (essentially a "heap address") is stored on the stack, but you cannot access or print this address directly as JavaScript does not expose real memory addresses [1] [2] [3] [4].

- All primitives are stored as direct values on the stack, so no "address" is needed to get the value[1] [2].

**Caveat:**
While the technical details of stack vs heap are relevant for performance and behavior, the actual memory addresses are abstracted away by JavaScript engines—you work only with values and references, not with physical addresses[1] [2] [4].In JavaScript:

- **Primitive data types** (such as numbers, strings, booleans, null, undefined, symbol, and bigint) are stored **directly on the stack**. Each variable of a primitive type holds its own value independently in the stack memory[1] [2] [3] [4].
- **Reference types** (such as objects, arrays, and functions) are stored **in the heap**. However, the **stack holds a reference (a kind of internal address or pointer) to the actual data** located in the heap[1] [2] [4]. This reference allows multiple variables to point to the same object or data structure in the heap.

You cannot view or print true memory addresses in JavaScript—these references are managed behind the scenes by the JavaScript engine for safety and portability. Only the behavior (copy-by-value for primitives, copy-by-reference for objects/arrays) is exposed to developer code[1] [4].

⁂


1. https://dev.to/alwaysaman/day-6understanding-stack-and-heap-in-javascript-3766
2. https://www.geeksforgeeks.org/javascript/memory-management-in-javascript/
3. https://dev.to/chintanonweb/mastering-javascript-memory-a-beginners-guide-to-stack-and-heap-57nj
4. https://codedamn.com/news/javascript/memory-management-complete-guide