

Intern and Fresher Developer Interview Questions and Answers

Intern / Fresher | All technologies

Common interview questions and answers for software engineering interns and freshers: JavaScript, React, SQL, Java, C#, Python and more, with simple explanations and examples.

27 questions Read online: <https://www.fredevtemplates.com/interview-questions/level/intern>

JavaScript

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

Level: intern JavaScript

var is function-scoped and can be redeclared; let and const are block-scoped, and const cannot be reassigned after it is set.

var is function-scoped, is hoisted and initialised to undefined, and can be redeclared in the same scope. let and const are block-scoped: they only exist inside the nearest pair of curly braces.

let and const are also hoisted, but they stay in the temporal dead zone until the line where they are declared, so using them earlier throws a ReferenceError instead of silently giving undefined.

const prevents reassigning the variable, not mutating the value. A const array or object can still be changed by pushing to it or setting a property. Use const by default, let when you need to reassign, and avoid var in new code.

Key points

var: function scope, hoisted as undefined, can be redeclared

let: block scope, temporal dead zone, cannot be redeclared in the same scope

const: like let, but the binding cannot be reassigned

```
if (true) {  
  var a = 1;  
  let b = 2;  
}  
console.log(a); // 1 (var ignores the block)  
console.log(b); // ReferenceError: b is not defined  
  
const list = [1, 2];  
list.push(3); // allowed, the array is mutated  
list = []; // TypeError: Assignment to constant variable
```

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

Level: intern JavaScript

== compares values after type coercion, while === compares both value and type with no conversion, so it is the safer default.

The loose equality operator == converts the operands to a common type before comparing, which leads to surprising results such as 0 == "" and null == undefined both being true. The strict equality operator === returns false immediately if the types differ.

Most teams enforce === with a lint rule. The one commonly accepted use of == is value == null, which checks for both null and undefined in one expression.

Neither operator compares objects by content. Two separate objects or arrays with identical contents are never equal, because both operators compare references.

Key points

`==` checks type and value, `===` coerces first

Prefer `===` everywhere except deliberate null checks

Objects are compared by reference, not by content

```
0 == ""; // true (both coerced to 0)
0 === ""; // false (number vs string)
null == undefined; // true
null === undefined; // false
NaN === NaN; // false (use Number.isNaN)
[1] === [1]; // false (different references)
```

3. What is hoisting in JavaScript?

Level: intern JavaScript

Hoisting is JavaScript moving declarations to the top of their scope during compilation, so functions and var variables can be referenced before the line where they are written.

Before running code, the engine processes declarations in each scope. Function declarations are hoisted with their full body, so you can call them before they appear. var variables are hoisted and set to undefined, but their assignment stays where you wrote it.

let, const and class declarations are hoisted too, but not initialised. Accessing them before the declaration throws a ReferenceError (the temporal dead zone). Function expressions assigned to a variable follow the variable's rules, not the function declaration rules.

Key points

Declarations are hoisted, assignments are not

Function declarations are available before their definition

let and const exist but cannot be used until declared

```
sayHi(); // "hi" - function declaration is fully hoisted
console.log(x); // undefined - var is hoisted, value is not
console.log(y); // ReferenceError - let is in the temporal dead zone

function sayHi() { console.log("hi"); }
var x = 5;
let y = 10;

notYet(); // TypeError: notYet is not a function
var notYet = function () {};
```

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

Level: intern JavaScript

undefined means a variable has been declared but no value was assigned; null is an explicit value meaning 'no value' that you assign on purpose.

JavaScript uses undefined automatically: for uninitialised variables, missing function arguments, missing object properties and functions with no return statement. null is something a developer sets deliberately to say that a value is intentionally empty.

typeof undefined is 'undefined', while typeof null is 'object', a long-standing bug in the language that cannot be fixed without breaking the web. Use value == null to test for both, or optional chaining and nullish coalescing (user?.name ?? 'Guest') to handle them safely.

```
let a; // undefined
const b = null; // null
typeof a; // "undefined"
typeof b; // "object" (historical bug)
a == b; // true
a === b; // false
const name = user?.name ?? "Guest";
```

5. What is the difference between map, filter and reduce in JavaScript?

Level: intern JavaScript

map transforms every item into a new array of the same length, filter keeps only items that pass a test, and reduce folds the whole array into a single value.

All three are array methods that do not mutate the original array (when your callback is pure). map returns an array with the same number of items. filter returns a shorter or equal array. reduce takes an accumulator and returns anything: a number, an object, even another array.

They chain well: filter first to reduce the work, then map, then reduce. Forgetting the initial value in reduce is a common bug: on an empty array it throws a TypeError.

Key points

map: one output per input

filter: keeps items where the callback returns truthy

reduce: always pass an initial value

```
const orders = [{ price: 20, paid: true }, { price: 35, paid: false }, { price: 15, paid: true }];

const prices = orders.map(o => o.price); // [20, 35, 15]
const paid = orders.filter(o => o.paid); // 2 orders
const total = orders
  .filter(o => o.paid)
  .reduce((sum, o) => sum + o.price, 0); // 35
```

React

6. What is the virtual DOM in React and how does it work?

Level: intern React

The virtual DOM is a lightweight JavaScript copy of the UI that React compares with the previous version after each update, so it only changes the parts of the real DOM that actually differ.

When state or props change, React re-runs your component to produce a new tree of React elements (the virtual DOM). It then compares that tree with the previous one in a process called reconciliation, works out the minimal set of changes, and applies only those to the real DOM.

Touching the real DOM is comparatively expensive, and batching updates lets React do it once per update instead of once per change. The virtual DOM is not automatically faster than hand-written DOM code, but it gives you a declarative model: you describe what the UI should look like for the current state, and React figures out how to get there.

Key points

Virtual DOM = tree of plain objects describing the UI

Reconciliation compares old and new trees (diffing)

Only the differences are applied to the real DOM

7. What is the difference between props and state in React?

Level: intern React

Props are read-only inputs passed from a parent component, while state is data a component owns and can change over time, which triggers a re-render when updated.

Props make a component reusable: the parent decides the values, and the child must not modify them. State is private memory for a component, created with `useState` or `useReducer`. Updating state tells React to re-render that component with the new value.

A good rule is to keep state as low as possible in the tree, and lift it up to the closest common parent only when two components need to share it. Data that can be calculated from props or state should be computed during render and not stored in extra state.

Key points

Props flow down, are read-only and come from the parent

State is owned by the component and changes cause re-renders

Derived values should be calculated, not stored in state

```
function Counter({ step }) { // step is a prop (read-only)
  const [count, setCount] = useState(0); // count is state (owned here)
  return <button onClick={() => setCount(count + step)}>{count}</button>;
}

<Counter step={5} />
```

HTML & CSS

8. What are semantic HTML elements and why do they matter?

Level: intern HTML & CSS

Semantic elements describe their meaning in the name, such as header, nav, main, article and footer, so browsers, assistive technology and search engines can understand the page structure.

A `<div>` or `<span>` says nothing about the content. `<header>`, `<nav>`, `<main>`, `<article>`, `<section>`, `<aside>` and `<footer>` tell the browser what that region is for. Screen readers use them as landmarks, and search engines use them as a hint about what is primary content.

Use one `<main>` per page, put the site chrome in `<header>` and `<footer>`, and wrap a self-contained story or card in `<article>`. Keep using `<div>` when you only need a styling hook. Pair landmarks with a clear heading order (h1 then h2) rather than skipping levels.

Key points

Semantic tags describe meaning, not just layout

They help accessibility, SEO and maintainability

`div` and `span` are still correct when there is no meaning to express

9. What is the CSS box model?

Level: intern HTML & CSS

Every element is a box made of content, padding, border and margin. The default content-box size counts only the content; border-box includes padding and border in the width you set.

From the inside out: content (the text or image), padding (space inside the border), border, then margin (space outside the border). Adjacent vertical margins collapse into the larger of the two, which surprises people building stacked cards.

With box-sizing: content-box (the default), width: 200px is the content only, so padding and border make the element wider. box-sizing: border-box makes width include padding and border, which is why most teams set *,

`*::before, *::after { box-sizing: border-box; }.`

```
*, *::before, *::after { box-sizing: border-box; }

.card {
width: 320px; /* includes padding and border */
padding: 16px;
border: 1px solid #ddd;
margin-bottom: 16px;
}
```

10. What is the difference between display none and visibility hidden in CSS?

Level: intern HTML & CSS

display: none removes the element from the layout so it takes no space; **visibility: hidden** hides it but leaves the empty space, and the element can still be read in the accessibility tree in some cases.

display: none is as if the element was not in the document for layout: later siblings move up. visibility: hidden keeps the box, so a 200px-tall banner still reserves 200px. opacity: 0 is similar to hidden for layout, but the element can still receive pointer events unless you also set pointer-events: none.

For content that should not be announced to screen readers, prefer display: none or the HTML hidden attribute. To hide visually but keep it available to assistive technology (a skip link target, for example), use a visually-hidden class rather than display: none.

11. What is the difference between inline, block and inline-block in CSS?

Level: intern HTML & CSS

Block elements start on a new line and take the full available width; **inline elements** sit in the text flow and ignore width and height; **inline-block** sits in the flow but accepts width, height, padding and margin on all sides.

Headings, paragraphs and div are block by default. span, a and strong are inline: width, height and vertical margin do not apply the way you expect. inline-block is the middle ground used for buttons or icon-and-label pairs that should sit in a row but have a fixed size.

Today Flexbox or Grid usually replaces floats and long rows of inline-block, because inline-block still leaves gaps from the whitespace between tags. Use the display value that matches the layout job, not a habit.

TypeScript

12. What is TypeScript and why do we use it?

Level: intern TypeScript

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It catches type errors in the editor and at build time, and it documents the shape of data for the next person who reads the code.

Every valid JavaScript file is already TypeScript, which is why teams can adopt it gradually. The compiler (tsc) erases types and emits JS that runs anywhere JavaScript runs. Types exist only at compile time: they are not available for runtime checks unless you add a library or generate them.

The usual wins are safer refactors, better editor autocomplete, and APIs that are harder to call wrongly. The cost is a build step and time spent modelling awkward JavaScript patterns. Interviews often follow this with "when would you still use JavaScript?" - small scripts, prototyping, or a codebase that is not yet worth converting.

13. What is the difference between interface and type in TypeScript?

Level: intern TypeScript

Both describe object shapes. interface can be merged and is preferred for public object contracts; type can express unions, intersections and primitives, and cannot be reopened later.

For a simple object, they look the same and teams pick one style. interface supports declaration merging: two interface User blocks become one, which is how many .d.ts files extend library types. type aliases are fixed once written.

Use type when you need a union (string | number), a mapped type, or a tuple. Use interface when you are describing an object that others might extend. You can implement either with a class. Do not waste interview time claiming one is "faster"; they compile to the same nothing.

Key points

interface: extendable, mergeable, object-shaped

type: unions, intersections, primitives, tuples

Pick one style for objects and stay consistent

```
interface User { id: string; name: string }
interface User { email?: string } // merged

type Id = string | number; // union: type only
type Admin = User & { role: "admin" }; // intersection
```

SQL

14. What is the difference between INNER JOIN and LEFT JOIN in SQL?

Level: intern SQL

INNER JOIN returns only rows that have a match in both tables, while LEFT JOIN returns every row from the left table and fills the right table's columns with NULL when there is no match.

Use INNER JOIN when you only want records that exist on both sides, such as orders that have a customer. Use LEFT JOIN when the left table is the main list and the related data is optional, such as all customers including those who have never placed an order.

A common trick is to find rows with no match by left joining and filtering on WHERE right.id IS NULL. Be careful: putting a condition on the right table in the WHERE clause turns a left join back into an inner join, so put it in the ON clause instead. RIGHT JOIN is the mirror image of LEFT JOIN, and FULL OUTER JOIN keeps unmatched rows from both sides.

Key points

INNER: matches only. LEFT: all left rows plus matches

Unmatched right-side columns are NULL

Filter the right table in ON, not WHERE, to keep left-join behaviour

```
-- every customer, with order count (0 if none)
SELECT c.id, c.name, COUNT(o.id) AS orders
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.id
GROUP BY c.id, c.name;

-- customers who never ordered
SELECT c.*
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.id
WHERE o.id IS NULL;
```

15. What is the difference between WHERE and HAVING in SQL?

Level: intern SQL

WHERE filters individual rows before grouping, while HAVING filters groups after GROUP BY, so only HAVING can use aggregate functions like COUNT or SUM.

SQL processes a query in a logical order: FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY. Because WHERE runs before grouping, the aggregates do not exist yet. HAVING runs after grouping, so it can test them.

Put every condition that does not need an aggregate in WHERE. It reduces the rows before grouping, which is faster than filtering the groups afterwards.

```
SELECT department, COUNT(*) AS staff, AVG(salary) AS avg_salary
FROM employees
WHERE status = 'active' -- filters rows first
GROUP BY department
HAVING COUNT(*) >= 5; -- filters groups after aggregation
```

16. What is the difference between DELETE, TRUNCATE and DROP in SQL?

Level: intern SQL

DELETE removes selected rows and can be rolled back, TRUNCATE quickly removes all rows and keeps the table structure, and DROP removes the entire table including its structure.

DELETE is a DML command: it can use a WHERE clause, fires row-level triggers, and is logged row by row, so it is slower on big tables but fully transactional. TRUNCATE is a DDL command that empties the whole table, usually resets identity counters and is much faster because it deallocates data pages instead of logging each row. DROP TABLE deletes the table definition, indexes and permissions as well.

Transaction behaviour differs by database: in PostgreSQL and SQL Server TRUNCATE can be rolled back inside a transaction, while in MySQL and Oracle it causes an implicit commit. Always check your database's documentation before relying on it.

```
DELETE FROM orders WHERE created_at < '2024-01-01'; -- some rows
TRUNCATE TABLE temp_imports; -- all rows, keep table
DROP TABLE temp_imports; -- table is gone
```

17. What is the difference between a primary key and a foreign key?

Level: intern SQL

A primary key uniquely identifies each row in its own table and cannot be NULL, while a foreign key is a column that references the primary key of another table to link the two.

A table has one primary key (which may be made of several columns). It guarantees uniqueness and is indexed automatically. A table can have many foreign keys. A foreign key enforces referential integrity: you cannot insert an order for a customer that does not exist, and you cannot delete a customer that still has orders unless you define a cascade rule such as ON DELETE CASCADE.

A foreign key column may contain duplicates and, unless declared NOT NULL, may be NULL. A unique key also enforces uniqueness but, unlike a primary key, allows NULL (the exact number of NULLs allowed depends on the database).

```
CREATE TABLE customers (
  id INT PRIMARY KEY,
  name VARCHAR(100) NOT NULL
);

CREATE TABLE orders (
  id INT PRIMARY KEY,
  customer_id INT NOT NULL,
  FOREIGN KEY (customer_id) REFERENCES customers(id) ON DELETE RESTRICT
);
```

C#

18. What is the difference between a value type and a reference type in C#?

Level: *intern C#*

A value type variable holds its data directly and is copied on assignment, while a reference type variable holds a reference to an object, so assignment copies the reference and both variables see the same object.

Value types include int, double, bool, char, DateTime, enums and any struct. Reference types include class, string, arrays, delegates and object. A value type is stored inline where it is declared: on the stack for a local variable, or inside the containing object on the heap. A reference type's object always lives on the managed heap and is cleaned up by the garbage collector.

This changes behaviour when you pass values to methods. Passing a struct hands the method its own copy, so changes inside do not leak out. Passing a class passes a copy of the reference, so the method can modify the same object. Reference types can be null; value types cannot, unless you use Nullable<T> (int?).

Key points

Value types are copied, reference types share one object

Reference types can be null, value types need int? syntax

Large mutable structs are usually a bad idea

```
struct PointS { public int X; }
class PointC { public int X; }

var s1 = new PointS { X = 1 }; var s2 = s1; s2.X = 99;
Console.WriteLine(s1.X); // 1 (copy)

var c1 = new PointC { X = 1 }; var c2 = c1; c2.X = 99;
Console.WriteLine(c1.X); // 99 (same object)
```

19. What is the difference between string and StringBuilder in C#?

Level: *intern C#*

A string is immutable, so every change creates a new string object, while StringBuilder is a mutable buffer that modifies its contents in place, making it much faster for repeated concatenation.

Each time you do `s += "x"` in a loop, the runtime allocates a new string and copies the old content, which is $O(n^2)$ work overall and creates a lot of garbage. `StringBuilder` keeps an internal character buffer that grows as needed, so appending is cheap and you call `ToString()` once at the end.

For a handful of concatenations, a plain string or string interpolation is clearer and just as fast. Use `StringBuilder` when building text in a loop or from many pieces.

```
var sb = new StringBuilder();
for (int i = 0; i < 10_000; i++)
{
    sb.Append(i).Append(', ');
}
string result = sb.ToString();
```

Java

20. What is the difference between JDK, JRE and JVM in Java?

Level: intern Java

The JVM runs Java bytecode, the JRE is the JVM plus the standard libraries needed to run Java programs, and the JDK is the JRE plus the development tools such as the compiler (javac).

You write source code in .java files, the compiler javac (part of the JDK) turns it into platform-independent bytecode in .class files, and the JVM on any operating system loads and executes that bytecode. That is how Java achieves "write once, run anywhere".

Since Java 11 Oracle and most vendors no longer ship a separate JRE download. You install a JDK, or build a trimmed runtime with jlink. The JVM itself also contains the class loader, the bytecode verifier, the JIT compiler that turns hot bytecode into native code, and the garbage collector.

Key points

JDK = JRE + development tools

JRE = JVM + class libraries

JVM executes bytecode and is platform specific

21. What is the difference between == and equals() in Java?

Level: intern Java

== compares primitives by value and objects by reference (whether they are the exact same object), while equals() compares the logical content of two objects when the class overrides it.

For String this surprises many people: two strings with the same text created with new are different objects, so == is false even though equals() is true. String literals may be equal under == only because of the string pool, which you should never rely on.

The default equals() inherited from Object behaves like ==. If you override equals() in your own class, you must also override hashCode() so that equal objects have equal hash codes, otherwise they will be lost in a HashMap or HashSet.

Key points

== compares references for objects

equals() compares content if overridden

Override hashCode() whenever you override equals()

```
String a = new String("java");
String b = new String("java");
System.out.println(a == b); // false - different objects
System.out.println(a.equals(b)); // true - same content

Integer x = 127, y = 127;
System.out.println(x == y); // true (cached small values) - do not rely on it
```

22. What is the difference between String, StringBuilder and StringBuffer in Java?

Level: intern Java

String is immutable, StringBuilder is a mutable and fast but not thread-safe buffer, and StringBuffer is a mutable buffer with synchronized methods that is thread-safe but slower.

Because String is immutable, every concatenation creates a new object, so building text in a loop with += is wasteful. Immutability also makes strings safe to share between threads, cache their hash code and use as HashMap keys, and it is what allows the string pool.

Use StringBuilder for building text in a single thread, which is almost always. StringBuffer is a legacy class from Java 1.0 and is rarely needed, since sharing a builder across threads is uncommon and you usually need your own higher-level synchronisation anyway.

```
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 5; i++) {
    sb.append(i).append(', ');
}
String result = sb.toString(); // "0,1,2,3,4,"
```

23. What is the difference between final, finally and finalize in Java?

Level: intern Java

final is a keyword that prevents reassigning a variable, overriding a method or extending a class; finally is a block that always runs after try/catch for cleanup; finalize() was a method called by the garbage collector and is now deprecated.

final on a variable means it can be assigned once (the object it refers to can still change). On a method it cannot be overridden, and on a class it cannot be subclassed (String is final). finally runs whether or not an exception was thrown, which is why it was used to close resources, but try-with-resources is now the preferred way.

finalize() was unreliable because nobody knows when, or whether, the garbage collector will call it. It has been deprecated for removal since Java 18. Use try-with-resources and AutoCloseable, or Cleaner, instead.

```
try (BufferedReader reader = new BufferedReader(new FileReader("data.txt"))) {
    System.out.println(reader.readLine());
} catch (IOException e) {
    log.error("read failed", e);
} // reader is closed automatically
```

Python

24. What is the difference between a list and a tuple in Python?

Level: intern Python

A list is mutable and can be changed after creation, while a tuple is immutable; tuples are slightly faster, use less memory and can be used as dictionary keys when their items are hashable.

Use a list for a collection that grows, shrinks or is reordered, such as the items in a shopping cart. Use a tuple for a fixed group of values that belong together, such as a coordinate (x, y) or a database row. Tuples also signal to the reader that the data should not change.

Immutability applies to the tuple itself: if a tuple contains a list, that list can still be modified. A one-item tuple needs a trailing comma, (5,), because (5) is just the number 5 in parentheses.

```
items = [1, 2, 3]
items.append(4) # fine

point = (10, 20)
point[0] = 5 # TypeError: 'tuple' object does not support item assignment

locations = {(10, 20): "home"} # tuples can be dict keys
```

25. What is the difference between is and == in Python?

Level: intern Python

== checks whether two objects have equal values, while is checks whether they are the exact same object in memory.

== calls the object's `__eq__` method. is compares identity (the same as comparing `id(a) == id(b)`). Two separate lists with the same content are == but not is.

Use is for singletons, mainly None (if x is None), True and False. Do not use is to compare numbers or strings: CPython caches small integers and some strings, so the result can look right in a quick test and fail in real code.

```
a = [1, 2]
b = [1, 2]
a == b # True (same value)
a is b # False (different objects)
c = a
c is a # True

if result is None:
    ...
```

Node.js

26. What is Node.js and why is it called single-threaded?

Level: intern Node.js

Node.js is a JavaScript runtime built on Chrome's V8 engine that runs your JavaScript on a single main thread and uses non-blocking I/O, so one thread can handle many concurrent connections.

Your JavaScript code runs on one thread, so there is no shared-memory race condition between your own functions. But Node.js is not single-threaded overall: the libuv library handles network I/O through the operating system's asynchronous mechanisms and uses a thread pool (four threads by default) for file system access, DNS lookups, compression and some crypto functions.

When a request arrives and needs a database result, Node starts the operation, registers a callback and moves on to other work instead of waiting. This model is excellent for I/O-heavy applications such as APIs, chat and streaming, and poor for long CPU-bound computation on the main thread, which blocks every other request.

Key points

Your JavaScript runs on one thread

libuv uses the OS and a thread pool for I/O

Great for I/O-bound work, risky for CPU-bound work

27. What is the difference between package.json and package-lock.json?

Level: intern Node.js

package.json lists your project's dependencies with allowed version ranges, while package-lock.json records the exact versions that were installed so every machine gets an identical dependency tree.

In package.json, a range such as ^4.18.0 allows any compatible 4.x update. The lock file pins the exact resolved version and the integrity hash of every package, including nested dependencies. Commit both files, and use npm ci in CI and production builds: it installs exactly what the lock file says and fails if the two files disagree, while npm install may update the lock file.

Also know the difference between dependencies (needed at runtime) and devDependencies (build and test tools), and run npm audit regularly to check for known vulnerabilities.

These answers are written for real interview conversations. Prefer explaining a trade-off out loud over reciting a definition. Full explanations and related questions: <https://www.fredevtemplates.com/interview-questions/level/intern>