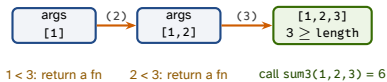


In one line: Functions are values that close over their scope, so these idioms are closures with a shape: **currying** turns $f(a,b,c)$ into $f(a)(b)(c)$; **partial application** fixes some arguments now (bind); an **IIFE** makes a private scope and runs it once; a **tagged template** is a call on a template's pieces; a **generator** is a pausable function that yields values *on demand*.

1 `c = curry(sum3); c(1)(2)(3)`



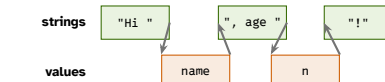
each step is a closure over the args so far; arity comes from `fn.length`:

(a, b = 1, c) → 1 · (... xs) → 0 — curry fires too early

partial: `sum3.bind(null, 10)` fixes a once

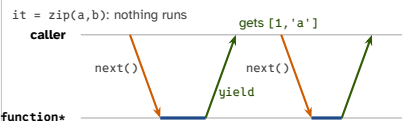
2 `tag`Hi ${name}, age ${n}!``

is the call `tag(strings, name, n)`:



`strings.length = values + 1` (may hold "") · `strings.raw = unescaped · both frozen` · the same array every time this call site runs → a tag can cache per template (lit-html)

3 a generator runs on demand



runs only between `next()` and `yield`; state lives in the paused frame → lazy, infinite sequences OK; break in `for..of` calls `it.return()`

coroutine detail: js-prototypes-objects-classes

How it works

- Currying** = calls of one argument (or “until arity”); **partial application** = fix some args now. The note’s `currying(fn, ... args)` is partial application, like `f.bind(null, 10)` (which also fixes this). `fn.length` counts only parameters before the first default, no rest.
- IIFE** (`(function () { ... })()` or `((() => {...})())`: a private scope, run once — the pre-module way to avoid globals. The parentheses make it an *expression*. `(async () => { await x })()` is only needed in CommonJS or a classic script — ESM (`.mjs`, “type”: “module”) has top-level await.
- arguments**: array-like (no `map`; `Array.from` it), absent in arrows (they see the enclosing function’s), and in sloppy mode with simple params it *aliases* them. Use `... rest`: a real array, and explicit.
- Tagged template**: `tag(strings, ... values)`. Invalid escapes are allowed: cooked undefined, raw kept. `String.raw`C:\new`` keeps the backslash. Uses: `sql` `` placeholders, `html` ``, `css` ``, `gql` ``.
- Running strings as code**: *direct* `eval(s)` sees the local scope (XSS sink, defeats optimisers and minifiers); *indirect* `(0, eval)(s) / eval?.(s)` and `new Function('a', 'return a+1')` see only globals. `node:vm` runs code with a contextified object as its global (writes land on that object).

Example

```
const curry = fn => function curried(...args) {
  return args.length >= fn.length
    ? fn.apply(this, args)
    : (...more) => curried.apply(this, [...args, ...more]);
};
const sum3 = (a, b, c) => a + b + c;
curry(sum3)(1)(2)(3); curry(sum3)(1, 2)(3); // 6, 6
const add10 = sum3.bind(null, 10); // add10(1, 2) = 13

const sql = (strs, ...vals) => ({ // no string concat:
  text: strs.reduce((q, s, i) => q + '$' + i + s), values: vals });
sql`SELECT * FROM t WHERE id = ${id}`;
// { text: 'SELECT * FROM t WHERE id = $1', values: [id] }
```

```
function* zip(...arrs) { // stops at the SHORTEST
  const n = arrs.length ? Math.min(...arrs.map(a => a.length)) : 0;
  for (let i = 0; i < n; i++) yield arrs.map(a => a[i]);
}
[...zip([1, 2, 3], ['a', 'b'])]; // [[1, 'a'], [2, 'b']]
function sumAll(...nums) { // not func(1, 2 3)
  return nums.reduce((s, x) => s + x, 0);
}
function inDir(dir, work) { // chdir is PROCESS-wide
  const prev = process.cwd(); process.chdir(dir);
  try { return work(); } finally { process.chdir(prev); } }
```

Arrays — mutate or copy

`a.splice(start, n, ... items)` **mutates**: removes `n` from `start` (negative = from the end), inserts, returns the removed: on `[1, 2, 3, 4, 5, 6, 7]`, `splice(-5, 5)` returns `[3, 4, 5, 6, 7]` and leaves `[1, 2]`. `slice` copies a range; `toSpliced/toSorted/toReversed/with` (ES2023) copy — use them for React/Redux state. `shift()` removes the first element.

Node odds

`#!/usr/bin/env node + chmod +x + package.json "bin"` makes a CLI. SVG from JS needs `createElementNS(svgNS, 'rect')` with the SVG namespace URI — plain `createElement('rect')` makes an HTML element that never renders.

Interview traps

- ! Currying a function with defaults or `... rest`: `fn.length` is too small, so it fires early — pass the arity explicitly.
- ! IIFE after a line with `no ;` — `const a = b then (function () {})(c)` parses as `b(function(){})(c)`. Lead with `;`.
- ! Tag values are raw values, not escaped: a HTML tag must escape them itself — that is the whole point of tagging.
- ! Calling `node:vm` a sandbox — Node’s docs: “not a security mechanism. Do not use it to run untrusted code.”

Remember

Curry: one arg at a time · **partial**: some now · **IIFE**: scope now · **tag**: a call on the pieces · **generator**: on demand.

Likely questions

- Curry vs partial? — one-arg chain until arity vs pre-filled args.
- Why a frozen, reused strings array? — the tag caches per call site.
- Why avoid `eval`? — caller’s privileges, slow, unminifiable.