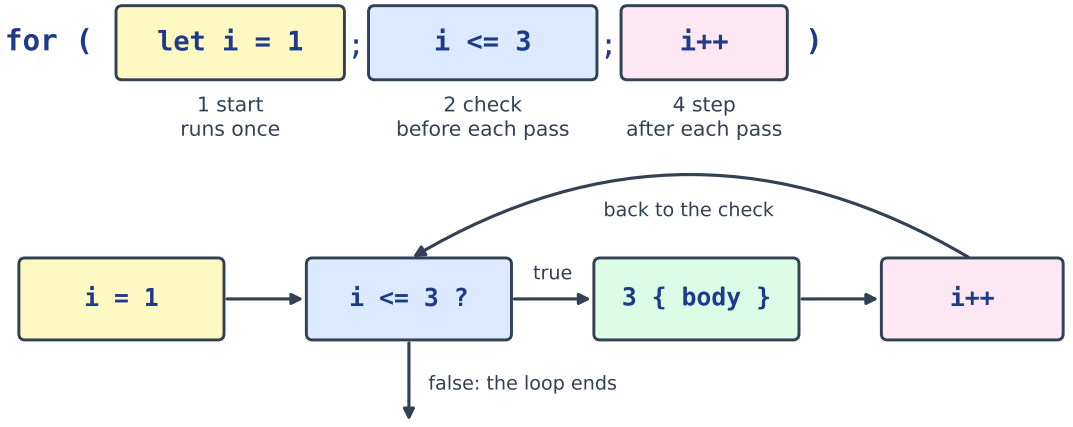
# Loops

Ref Javascript

## for loops




A loop repeats a block of code. A `for` loop is best when you know how many times. Its brackets hold three parts, separated by semicolons:

```
for (let i = 1; i <= 3; i++) {
  console.log(i);
}
// 1
// 2
// 3
```

1. **start**: `let i = 1` runs once, before the loop.
2. **condition**: `i <= 3` is checked before every pass; the loop stops when it is false.
3. **step**: `i++` (add 1 to `i`) runs after every pass.

Each pass through the loop is one iteration . Change the step to count in other ways:

```
let line = "";
for (let i = 10; i > 0; i -= 3) {
  line += i + " ";
}
console.log(line);   // 10 7 4 1
```

The accumulator pattern makes a variable **before** the loop and updates it in every pass:

```
let total = 0;
for (let n = 1; n <= 5; n++) {
  total += n;          // the same as total = total + n
}
console.log(total);    // 15
```

## while loops

A `while` loop repeats while its condition is true. Use it when you do not know in advance how many passes you need:

```
let money = 100;
let years = 0;
while (money < 200) {
  money = money * 1.1;   // 10% interest each year
  years++;
}
console.log(years);      // 8
```

- The condition is checked **before** each pass, so the body may run zero times.
- If the condition stays true and no `break` or other exit is reached, you get an infinite loop . (Press **Stop** if that happens.)

## for...of, break & continue

`for...of` visits every item of an array (or every character of a string) in turn, with no index to manage:

```
const fruits = ["apple", "kiwi", "plum"];
for (const fruit of fruits) {
  console.log(fruit.toUpperCase());
}
// APPLE
// KIWI
// PLUM
```

`break` leaves a loop at once; `continue` skips the rest of this pass and goes on to the next one:

```
const marks = [55, 80, -1, 90];
let sum = 0;
for (const m of marks) {
  if (m < 0) {
    break;            // -1 means "end of the data"
  }
  if (m < 60) {
    continue;         // skip a fail, keep looping
  }
  sum += m;
}
console.log(sum);     // 80
```

### Common mistakes

- An off-by-one error: `i < 3` runs for 0, 1, 2; `i <= 3` runs one more time. Count the passes.
- Starting the total inside the loop: `let total = 0;` must come **before** the loop, or it is reset every pass.
- A `while` loop runs for ever if its condition stays true and no exit is reached.