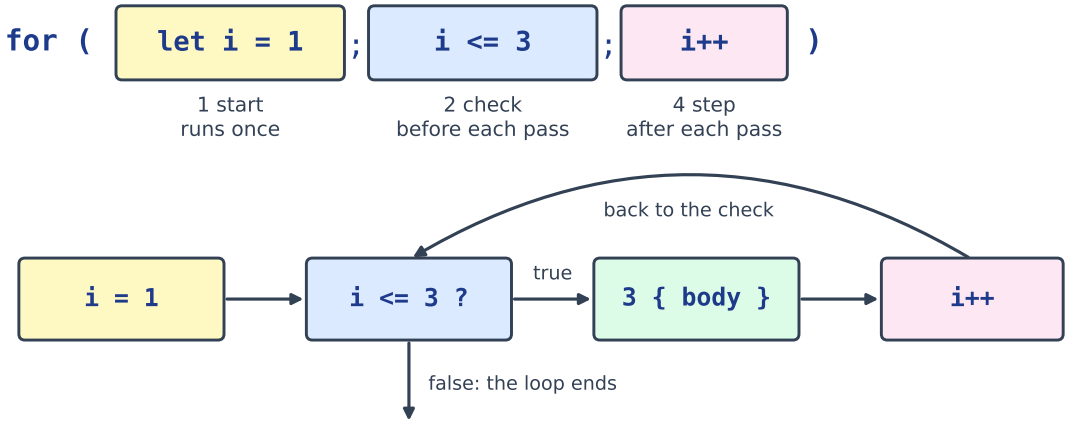
循环

Ref Javascript

for loops



循环循环用于重复执行一段代码块。当你确切知道需要重复多少次时，最适合使用 `for` 循环。它的括号内包含三个部分，用分号分隔：

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

1. **start**: `let i = 1` 在循环开始前仅执行一次。
2. **condition**: `i <= 3` 在每次迭代前都会检查；当其变为假时，循环停止。
3. **step**: `i++`（将 1 加到 `i` 上）在每次迭代后执行。

循环的每一次通过称为一次迭代迭代。可以更改步长以其他方式计数：

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

累加器累加器模式是在循环**之前**声明一个变量，并在每次迭代中更新它：

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

while loops

`while` 循环在其条件为真时重复执行。适用于无法预先确定需要多少次迭代的场景：

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

- 条件在每次迭代**之前**检查，因此循环体可能一次都不执行。
- 如果条件持续为真且未遇到 `break` 或其他退出机制，将形成无限循环无限循环。（若发生此情况请按下**停止**。）

for...of, break & continue

`for...of` 依次访问数组中的每一项（或字符串中的每个字符），无需管理索引：

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

`break` 立即跳出循环；而 `continue` 则跳过本次迭代的剩余部分并进入下一次迭代：

```
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
```

常见错误

- 差一错误：`i < 3` 对 0、1、2 执行操作，而 `i <= 3` 多执行了一次。请仔细计算迭代次数。
- 在循环内部初始化总数：`let total = 0;` 必须在循环**之前**，否则每次迭代都会被重置。
- 如果条件始终为真且未到达退出点，`while` 循环将无限运行。