JavaScript Part 3: Conditions and Loops

Programs become useful when they can choose and repeat. A condition lets the code choose between actions, and a loop lets it repeat an action without rewriting it. Four steps cover both: decide, combine, repeat and control. Each step answers one simple question, and every example can be tried in the browser console. In this part, Dina, a financial analyst at an insurance company, uses them to sort the daily customer claims.

1. Decide

The if statement runs a block of code only when its condition is true. The condition is written in parentheses and the block in curly braces. We can add else if to test another condition, and else for everything that remains. JavaScript checks the conditions from top to bottom and runs only the first block whose condition is true. The question is: what are the cases, and in which order must they be checked? Putting the most specific case first stops a general rule from catching everything. A final else makes sure that no case is left without an answer.

Example: A receptionist at a clinic follows a simple rule for visitors: staff members go straight in, guests with a booking receive a badge and a seat in the waiting room, and everyone else waits at the desk until someone is free. She checks the cases in exactly that order, so each visitor follows one path only. This is an if, else if and else chain: the first case that fits decides, and nobody is treated twice.

At work: Dina sorts claims by amount. She asks the Decide question: what are the cases, and in which order must they be checked? A claim above the manager limit goes to review, a claim above the small limit goes to a clerk, and everything else is approved automatically. Because the checks run from top to bottom, the largest claims are caught first.

2. Combine

Real rules often depend on more than one condition. The logical and operator, written with two ampersands, is true only when both sides are true. The logical or operator, written with two vertical bars, is true when at least one side is true. The not operator, written with an exclamation mark, reverses a boolean. The question is: must all the conditions hold, or is one enough? JavaScript also treats some values as false inside a condition: zero, empty text, null, undefined, NaN and false itself, while every other value counts as true. This is useful for checks such as whether a name was entered, but it can surprise us when zero is a valid answer.

Example: A bank approves a loan only when the applicant has a steady income and no unpaid debt, so both conditions must hold together, exactly like the and operator. Either missing condition is enough to stop the process. A shop that gives a discount to members or to students works differently, because one true condition is enough for it, and that is the or operator.

At work: Dina's rule for automatic approval needs two things: the amount is below the small limit, and the customer has no earlier warning flag. She asks the Combine question and uses the and operator, because both conditions must hold. For the review queue she uses the or operator, because either a high amount or a warning flag is enough. A claim with an amount of zero, which counts as false in a condition, is checked separately so that it is not mistaken for a missing amount.

This document is a property of Asaseeyat Academy - © Copy Right Protected.

3. Repeat

A loop repeats a block of code without us copying it. The for loop has three parts inside its parentheses: a starting point, a condition to continue and a step after each round. The for of loop is simpler when we need each item of a list in turn, and the while loop repeats as long as its condition stays true. The list itself is called an array, and a later part covers arrays in detail. The question is: what repeats, and what changes in each round? For example, `for (const claim of claims)` reads as: for each claim in the list of claims. Inside the block, the variable claim holds the current item, and a running total can be kept in a variable declared before the loop.

Example: At the end of the day, a cashier counts one kind of banknote again and again until the whole pile is finished. The task is the same in each round, and only the banknote in the cashier's hand changes, which is exactly what changes in each round. A loop describes this once and lets the machine repeat it, whether the pile is small or huge.

At work: Dina has the day's claims in a list. She asks the Repeat question: what repeats, and what changes in each round? The loop visits every claim, applies the rules from the earlier steps and adds the approved amounts to a total. Work that took a clerk an hour is done in a fraction of a second, and the same code handles ten claims or ten thousand.

4. Control

Sometimes a loop must skip an item or stop early. The continue statement ends the current round and moves to the next item, while the break statement ends the loop completely. The question is: when should this loop skip, and when should it stop? A loop also needs a way to end, because if the condition of a while loop never becomes false, the loop runs forever and the page freezes. Changing the variable that the condition depends on, inside the loop, prevents this. Clear exits make loops safe to run on real data.

Example: A security officer checking bags at an airport lets an empty bag pass quickly and moves on to the next one, but stops the whole queue when an alarm sounds. The first is a skip, like continue, and the second is a stop, like break. Both are decisions made inside a repeating task, and the queue stays safe because each exit is clear.

At work: Dina's loop uses continue to skip claims that are already closed. She asks the Control question and uses break as soon as the daily approval budget is used up, so that the remaining claims wait for the next day. She also checks that the while loop which retries a document upload changes its counter in every round. The loop now ends every time, and the finance team trusts the daily report.

Conclusion

Dina's daily sorting now runs on its own and ends every time. Decide asks what the cases are and in which order to check them, and combine asks whether all the conditions must hold or one is enough. Repeat asks what repeats and what changes in each round, and control asks when the loop should skip and when it should stop. In the next part, we pack logic like this into functions that can be reused.

Sources

- MDN Web Docs: Control flow and error handling – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling
- MDN Web Docs: Loops and iteration – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
- MDN Web Docs: if...else statement – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else
- MDN Web Docs: for...of statement – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of

This document is a property of Asaseeyat Academy - © Copy Right Protected.