JavaScript Part 2: Operators, Strings and Template Literals

Every business page eventually calculates something, compares something and writes a sentence for a person. Operators and strings are the tools for all three jobs. Four steps organise them: calculate, compare, shape and template. Each step answers one simple question, and the browser console is enough to try every line. In this part, Omar, an operations coordinator at a courier company, uses them to build the confirmation message of a shipment.

1. Calculate

Operators are symbols that tell JavaScript to do something with values. The arithmetic operators are plus, minus, times, divide and remainder, and there is also an operator for powers. JavaScript follows the order we know from school mathematics: multiplication and division come before addition and subtraction, and anything inside parentheses comes first. The question is: in which order will JavaScript work out this line? When the order is not obvious to a reader, we add parentheses to make it visible, even if they are not strictly needed. The remainder operator, written with a percent sign, gives what is left after a division. Together with rounding down, it answers questions such as how many boxes can be filled completely and how many items are left over.

Example: A warehouse holds forty-seven parcels, and one pallet takes ten. Dividing and rounding down gives four full pallets, and the remainder gives seven parcels waiting for the next pallet. Without the remainder, the supervisor would only know that the parcels fill four pallets and a little more. With both answers, the supervisor can plan the truck load and order one more pallet at once.

At work: Omar calculates the price of a shipment from its weight, the rate per kilogram and a handling fee. He asks the Calculate question about that line. For a parcel of eight kilograms at five per kilogram plus a handling fee of three, the total is forty-three. He puts parentheses around the multiplication, even though it would come first anyway, so that the next reader sees the intention at once.

2. Compare

Comparison operators ask a question about two values and answer with a boolean: greater than, less than, greater than or equal, and equal. Equality has two forms. The strict form, written with three equals signs, is true only when the value and the type are both the same. The loose form, written with two equals signs, converts the types first, so the text five and the number five are considered equal. The question is: do we want these two values to match exactly, including their type? In nearly all cases the answer is yes, so the strict form is the safe choice. A single equals sign is something else again, because it assigns a value instead of comparing.

Example: A guard at the gate checks a visitor's name against the list, and he wants an exact match, not a similar one. If the list holds one spelling and the visitor shows a slightly different one, a guard who accepts anything close will let the wrong person in. A guard who accepts only the exact spelling stops the mistake. Strict equality is that exact check, and it accepts only when everything matches, including the type.

At work: Omar compares the order value with the limit for free handling. The order value arrives from a form as text, so he converts it with the Number function, as in the first part of this series. He asks the Compare question and uses the strict form, so a value that is still text fails in testing instead of passing by accident. The result is a boolean, and it decides whether the handling fee is added.

3. Shape

A string is a sequence of characters, and its length property counts them. Strings come with methods, which are built-in actions that we call by writing a dot after the string. The trim method removes spaces from both

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

ends, the toUpperCase and toLowerCase methods change the letter case, and the includes method answers true or false depending on whether a piece of text is found. The slice method returns a part of the text between two positions, and positions are counted from zero. The question is: what shape must this text have before we use it? Strings never change in place, because every method returns a new string, so we keep the result in a variable. Cleaning text first prevents mismatches, for example when a customer types a name with extra spaces.

Example: A receptionist tidies a handwritten name before typing it into the system, with no extra spaces and the same spelling style as the customer list. The system then finds the customer on the first search. In code, the trim method removes the spaces, and the lowercase method gives every name the same style. Both take a fraction of a second, and the clean name then matches the list.

At work: The customer's name reaches Omar's code as text with extra spaces and mixed letter case. He asks the Shape question, removes the spaces with trim, and sets the case with the lowercase and uppercase methods so that every name has the same form. He also uses includes to check whether the delivery note mentions the word fragile, and he flags the parcel. The search in the customer list now succeeds on the first attempt.

4. Template

Joining text with the plus sign quickly becomes hard to read, with quotation marks and spaces everywhere. Template literals solve this problem. They are written between backticks instead of quotation marks, and an expression placed inside a dollar sign with curly braces is replaced by its value. A template literal can also continue over several lines, and the line breaks are kept. The question is: can a colleague read the finished sentence straight from the code? For example, `Hello ${customerName}` produces a greeting with the real name in place of the placeholder, and the part in the braces can even be a calculation. The result is a string like any other, so the methods from the previous step work on it as well.

Example: A form letter has gaps for the name and the date, and the clerk fills them in without rewriting the letter. The sentences around the gaps stay the same. In a template literal, the placeholder inside the dollar sign and braces is such a gap, and the program fills it with the real name or the real total. The words around the placeholder stay exactly as they were written.

At work: Omar builds the confirmation message in one template literal that holds the customer name, the weight, the handling fee and the total. He asks the Template question and reads the message aloud, and it sounds like a normal sentence. The old version, built with many plus signs, was missing a space and printed the weight stuck to the word kilograms. The new message reaches the customer without that mistake.

Conclusion

Omar's confirmation message now shows the right price in a clean, readable sentence. Calculate asks in which order JavaScript will work out a line, and compare asks whether two values must match exactly, including their type. Shape asks what form the text needs before it is used, and template asks whether a colleague can read the finished sentence straight from the code. In the next part, conditions and loops let the code choose between actions and repeat them.

Sources

- MDN Web Docs: Expressions and operators – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators
- MDN Web Docs: Strict equality (===) – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Strict_equality
- MDN Web Docs: String – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String
- MDN Web Docs: Template literals – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals
- MDN Web Docs: Operator precedence – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_precedence
- MDN Web Docs: Remainder (%) – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Remainder

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