JavaScript Part 1: First Steps - Variables and Data Types

JavaScript is the programming language that web browsers run, so it is what makes a page respond to a visitor. Every program starts with the same need: keeping a value under a name so that it can be used again. Four steps make this safe and clear: declare, name, type and check. Each step answers one simple question, and the free developer console that every browser includes is enough to practise them. In this part, Rana, an assistant in the online ordering team of a bakery chain, uses them to fix the item counter of a customer's basket.

1. Declare

A variable is a named place that holds a value. We create one with a keyword, which is called declaring it. The question is: will this value change later? When the answer is no, we use the keyword const, and when it will change, we use the keyword let. If the code tries to give a new value to a const, JavaScript stops with an error, so the mistake shows up at once instead of hiding inside a wrong result. A good habit is to start with const and switch to let only when the value must change.

Example: A shop prints its opening hours on the door, but rewrites its stock list on a board every evening. Ask the Declare question about each one. Will the opening hours change later? No, so they behave like a const, and nobody repaints the door every day. Will the stock list change later? Yes, it changes constantly, so it behaves like a let. Asking the question about each value tells us which keyword to write.

At work: Rana asks the Declare question about each value in the basket. The delivery fee is the same for every order, so she declares it with const, while the number of cakes grows with every click, so she declares it with let. Later, a change elsewhere in the code tries to overwrite the fee, and the page stops with an error that names the problem. The wrong total never reaches a customer, and colleagues reviewing the code can see at a glance which values are fixed and which can change.

2. Name

Every variable needs a name, and the name must follow a few rules. It can contain letters, digits, the underscore and the dollar sign, but it cannot start with a digit and it cannot contain spaces. Words that already have a meaning in JavaScript, such as let and const, cannot be used as names. Names are case sensitive, so deliveryFee and deliveryfee are two different variables. The common style is called camelCase: the first word is lowercase, and every later word starts with a capital letter. The question is: would a colleague understand this name without asking us? A name such as itemCount explains itself, while a name such as x forces everyone to guess.

Example: In a spreadsheet, everyone understands a column titled Delivery fee, but nobody understands a column titled Column B. The reader must open the cells and guess. Variables behave the same way: with a name such as f, a colleague must read the whole program to learn what it holds. With a name such as deliveryFee, the answer is already on the page. So a clear name is documentation that travels with the value.

At work: Rana first wrote the basket total as t and the delivery fee as f. When the total looked wrong, a teammate could not tell which variable held what. She applies the Name question and renames them totalPrice and deliveryFee. The next error message says that deliveryfee is not defined, and the teammate spots the missing capital letter within minutes.

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

3. Type

Every value in JavaScript has a type, and the type decides what the code can do with it. A string is text inside quotation marks, and a number covers both whole numbers and decimals. A boolean is either true or false. Undefined means that a variable exists but has no value yet, while null is a value we set on purpose to say that nothing is here. The question is: what kind of value is this? The plus sign adds two numbers but joins two strings, so the same symbol behaves differently depending on the type. Decimals are stored approximately, so tiny rounding differences can appear, which is why many shops calculate money in the smallest unit, such as cents, and convert only at the end. Text that comes from a form field is always a string, even when the visitor types a number.

Example: A phone number that starts with a zero looks like a number, yet nobody adds phone numbers together. A number would also drop the leading zero, so the phone number is stored as a string. The quantity of cakes, however, is used in arithmetic, so it should be a number.

At work: Rana reads the quantity from the order form, and it arrives as a string. She asks the Type question, and the answer explains the strange result on her test page. The basket already holds two cakes, the customer types three more, and the plus sign joins the two side by side, so the counter shows twenty-three instead of five. A counter like that would confuse any customer.

4. Check

Instead of guessing the type, we ask for it. The typeof operator returns a string that names the type of a value, such as string, number or boolean. We print results with console.log and read them in the browser console. To turn text into a number we use the Number function, and when the text is not a valid number the result is NaN, which stands for not a number. One historical quirk is worth remembering: typeof applied to null returns object. The question is: does the type match what the next line of code expects? For example, `typeof quantity` returns the word string when the quantity came from a form.

Example: A cashier looks at a banknote before accepting it, instead of assuming it is genuine. The look takes a second and prevents an expensive mistake. Without it, a fake note goes into the drawer, and the loss appears only at the end of the day. Checking the type of a value before using it works the same way: one quick question stops a wrong result from travelling through the whole program.

At work: Rana prints typeof quantity and sees the word string. She converts the value with the Number function, checks that the result is not NaN, and only then adds it to the cakes already in the basket. The counter now shows five, and a visitor who types letters receives a clear message instead of a broken page.

Conclusion

Rana's counter now shows the right number and the code reads clearly, thanks to four steps. Declare asks whether the value will change later, and name asks whether a colleague would understand the name. Type asks what kind of value this is, and check asks whether the type matches what the next line expects. These habits matter in every later part of the series, because functions, arrays and forms all depend on values that have clear names and known types.

Sources

- MDN Web Docs: JavaScript data types and data structures – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures
- MDN Web Docs: const statement – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const
- MDN Web Docs: typeof operator – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof
- MDN Web Docs: Grammar and types (variable names, declarations) – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types

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