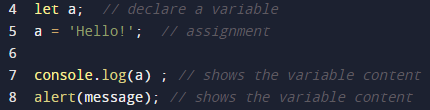
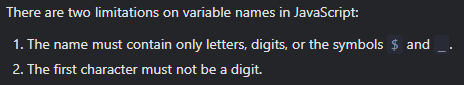
**VARIABLE AND OPERATORS**

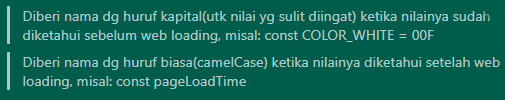
1. **Variable**

Let : we can change the value (of variable)

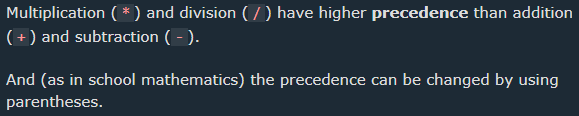
Const : unchanging value (of variable)

Var(the old-school declaring variable) is almost same with let. (avoid var).

1. Variable naming

khusus untuk const:

1. **Numbers**
2. Arithmethic
3. Operator Precedence



1. Numbers

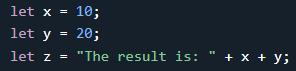
JS has only one type of number. Numbers can be written with or without decimals.

Extra large or extra small numbers can be written with scientific (exponent) notation:

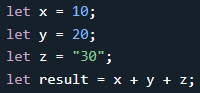
* Integers (numbers without a period or exponent notation) are accurate up to 15 digits.
* The maximum number of decimals is 17.
* Floating point arithmetic isn’t always 100% accurate:

1. Numbers & String

WARNING: JS use ‘+’ for both addition and concatenation. Numbers are added. Strings are concatenated. If you add number and string, the result will be string concatenation.

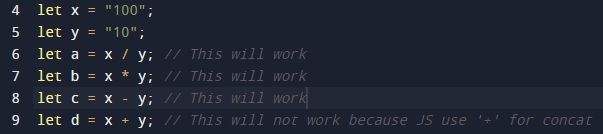
* A common Mistake

The result is: 1020 (not 30) because z is add string with number(concat)



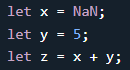
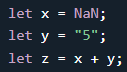
The result is 3030(not 102030) because x + y is

addition and then + z is concatenation

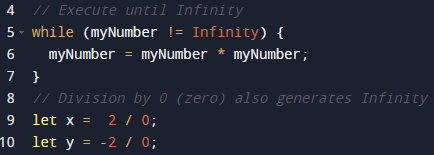
* JS will try to convert strings to numbers in all numeric operations
* NaN – Not a Number

Trying to do arithmetic with a non-numeric string will result in NaN

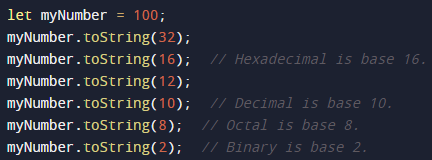
However, if the string is numeric, the result will be a number:

If you use NaN in a mathematical operation, the result will also be NaN:

Or the result might be a concatenation like NaN5:

* Infinity (or -Infinity)
* Hexadecimal

JS interprets hexadecimal if they are preceded by 0x. example

toString() method to output numbers from base 2 to 36

* JS Numbers as Objects

Note: don’t create number objects

The ‘new’ keyword complicates the code and slows down execution speed. Number Objects can produce unexpected results.

1. Useful number methods

- toFixed(2) : untuk membuat angka desimal menjadi 2 angka dibelakang koma(dengan pembulatan.

- Number(“3”) : converts its argument to number data types(if it can).

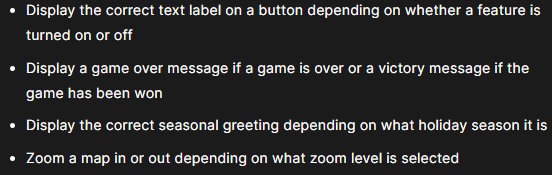
- String(3) : converts its argument to string.

3. Strict & Loose Equality

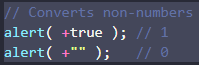
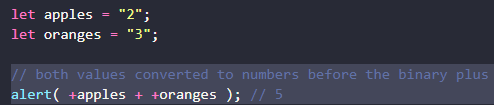
is operator checks whether its two operands are equal, returning a Boolean result.

Strict equality : ‘ === ’ (value and data type must be the same)

Loose equality : ‘ == ‘ (value must be the same, but data type does not have)

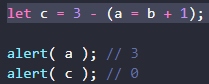
* Common use (booleans) for conditional

1. **OPERATORS**
2. Unary & Binay

* An operator is unary if it has a single operand (i.e -3).
* An operator is binary if it has two operands.
* Numeric conversion, unary +

Note: unary plus is higher precendence than any general operators (like unary negation, exponentioal, multiplication).

1. Assignment



sometimes we see it in JS libraries. Don’t write the code like that, because it's make code hard to read.

* Chaining Assignment

Once again, for the purposes of readability it’s better to split such code into few lines:

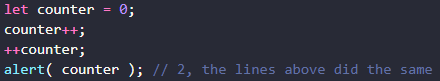
1. Increment & Decrement (++ and --)

Note: Increment/decrement can only be applied to variables. Trying to use it on a value like 5++ will give an error.

1. prefix form (++a) : return new value.
2. postfix form (a++) : return old value.

The difference between them

* If the result of increment/decrement is not used, there is no difference in which form to use:



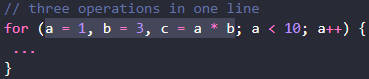
* If we’d like to increase a value and immediately use the result of the operator, we need the prefix form:



* If we’d like to increment a value but use its previous value, we need the postfix form:

1. Comma

The comma operator allows us to evaluate several expressions, dividing them with a comma , Each of them is evaluated but only the result of the last one is returned.

Why do we need an operator that throws away everything except the last expression?