Expressions and Evaluations

Expressions and Evaluations

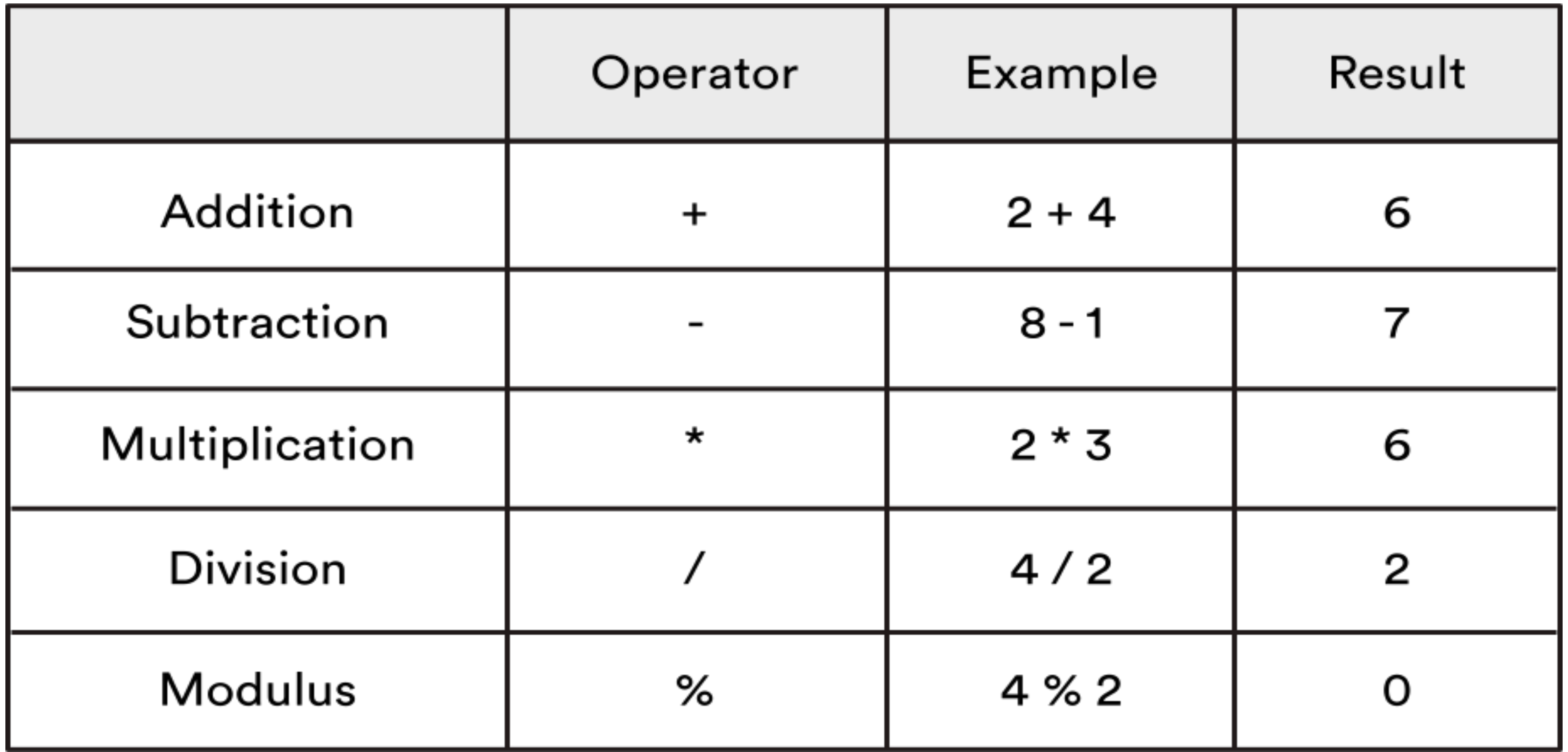
- Chances are, you've played around with a scientific calculator at some point. Kind of fun to punch in a big number (for example, 9876435), then x, and then another big number (say, 373848221), and hit the = button, right?
- Then, the calculator spits back a result (in this case, 3692287654572135) and you ooh and aah — what a mind-boggling large number!
- Well, that information you typed into the calculator is called an **expression**: a collection of values (*12345*) and operators (like + or x).
- The process of reducing this expression down to a single value is called **evaluation**. The JS Bin console is similar to a scientific calculator. It accepts an expression (in JavaScript) from its user and attempts to evaluate that expression, yielding a single value.
- The video that follows defines and explores expressions and evaluations.

Video: Expressions

- MyGA | General Assembly

Arithmetic Operators

- How do we combine numbers and operators to come up with more complex expressions in JS?
- It's simple — we use *arithmetic operators*.



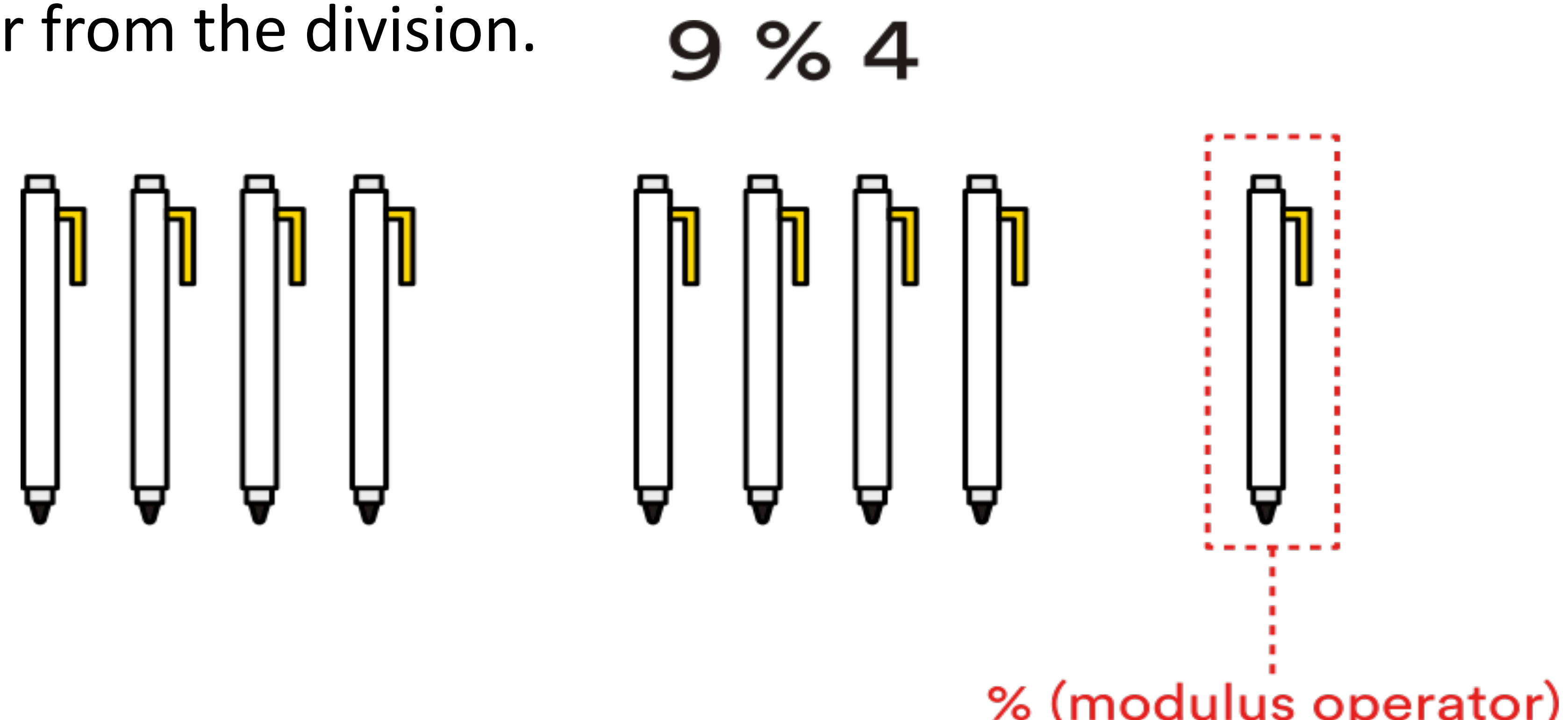
	Operator	Example	Result
Addition	+	2 + 4	6
Subtraction	-	8 - 1	7
Multiplication	*	2 * 3	6
Division	/	4 / 2	2
Modulus	%	4 % 2	0

Arithmetic Operators (Continued.)

- All of the standard arithmetic operators learned in grade school (addition, subtraction, division, and multiplication) are supported in JS. These should look familiar.
- But if you don't have a background in programming, that last operator — the modulus operator — might be new.

Modulus

- The modulus operator shows the remainder of a division problem.
- For example, 9 divided by 4 equals 2 with a remainder of 1. The modulus operator takes two numbers as inputs and returns what's leftover from the division.



Modulus (Continued)

- The modulus operator % is particularly useful in programming if we want to find out if a number is even or odd.
- If we divide by 2 and have a remainder of 1, we know the number is odd. If we have a remainder of 0, then we know that the number is even. Let's look at some examples.
- Odd numbers:
- `5 % 2;`
- `=> 1`
-
- `7 % 2;`
- `=> 1`
-
- Even numbers:
- `4 % 2;`
- `=> 0`
-
- `2 % 2;`
- `=> 0`
-
- Make sense? Good. This info will come in handy later on.
-

Try It!

- Look at the following five problems. Type each line of code into the JS Bin Console and see what is returned.

1. `45 % 6;`
2. `10 - 20;`
3. `7 / 2;`
4. `3 * 2;`
5. `10 % 4;`

String Concatentation and Coercion

- Now, let's see how you can use string values (textual information) in JS.
- When given string values, the + operator actually behaves differently — it concatenates, or combines, two strings together to make one big string.

Video: Hello Operators

- MyGA | General Assembly

String Concatentation and Coercion

- As you can see, putting single or double quotation marks around a value turns it into a string.
- So, even though both "6" and "8" look like numbers to us humans, JS sees that they're in quotation marks and therefore treats them as strings.
- `var number1 = "6";`
- `var number2 = "8";`
-
- `number1 + number2;`
- `// => "68"`
-
- Using the + operator to put the two strings together literally puts them next to each other, instead of evaluating their total.
- This is called concatenation (when strings are glued together).

String Concatentation and Coercion (Continued)

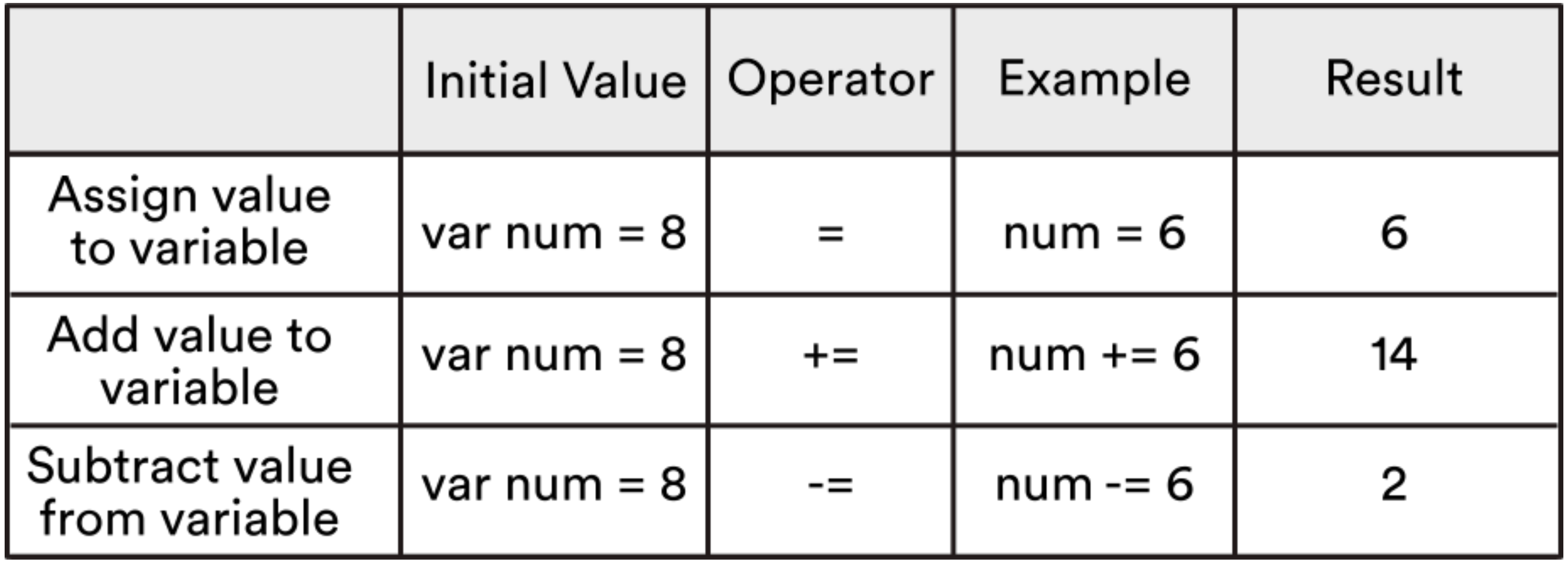
- Here's another example of concatenation.
- JS glued the two strings together, but do you notice anything wrong?
- `var firstName = "Han";`
- `var lastName = "Solo";`
- `firstName + lastName;`
- `// => "HanSolo"`

String Concatentation and Coercion (Continued)

- There's no space between the two words!
- This is because we didn't add the spaces in ourselves. It's just one of many reasons why we have to carefully watch our spacing and grammar.
- To fix this, we'll have to add in the space ourselves.
- `var firstName = "Han";`
- `var lastName = "Solo";`
- `firstName + " " + lastName;`
- `// => "Han Solo"`

Assignment Operators

- Now, let's get back to some math and look at assignment operators.
- You're already familiar with the = assignment operator, but there are also ones we can use to add or subtract value from a variable. Take a look:



	Initial Value	Operator	Example	Result
Assign value to variable	var num = 8	=	num = 6	6
Add value to variable	var num = 8	+=	num += 6	14
Subtract value from variable	var num = 8	-=	num -= 6	2

Assignment Operators (Continued.)

- The += operator adds a value to an existing variable.
- The -= operator subtracts a value from an existing variable.
- Note: Keep in mind that we'll always need an = somewhere in the line of code when we want to either assign or update the value of a variable, as in the above chart.
-

Try It!

- Type each of the following lines of code in the JS Bin Console and hit return to run each line of code.

1. `var myNumber = 8;`
2. `myNumber += 3;`
3. `myNumber -= 5;`
4. What is the final value of `myNumber`?
5. Type `myNumber;` into the console and hit return to check!

Try It! (Continued)

- Answer: The final value of `myNumber` should be 6.
- While we've covered what seems like a lot of math in this section, don't worry — you're not going to be doing calculus in this course. It's important that we review these concepts, because there will be many times when you'll solve a problem by using one of their basic principles.
- When it comes down to it, computers operate with a simple, straightforward logic.

Working with Multiple Variables

- Sometimes, we find variables on both sides of the =. Suppose we have two variables, `x`and `y`, like the example below:
- `var x = 5;`
- `var y = 10;`
- `x = y + 10;`
-
- What happens in that third line?
- For starters, everything to the right of the = must be evaluated before any kind of assignment can happen. This is why we like to use the phrase "assignment always happens right to left!"
- `y + 10;` evaluates to 20, so what we're left with is the expression `x = 20;`. This assigns the value 20 to `x`, and the entire expression evaluates to 20.

Working with Multiple Variables (Continued)

- Let's look at one more example using the same two variables, `x` and `y`.
- `var x = 1;`
- `var y = 10;`
- `x = y * 2;`
- `y = x + 1;`
- `x = y + 1;`
- `y = 2 * x;`
-
- Feeling dizzy? Don't worry, we'll step through this one together.

Working with Multiple Variables (Continued)

- `var x = 1;`
- `var y = 10;`
- `x = y * 2;`
- `y = x + 1;`
- `x = y + 1;`
- `y = 2 * x;`
-
- **Line 1**: We declare a new variable `x` and assign it the value `1`.
- **Line 2**: We declare another new variable `y` and assign it the value `10`.
- **Line 3**: As of this point in the code, `y` has a value of `10`. We multiply that by `2`, resulting in `20`. We assign that resulting value to `x`, so `x` now has a value of `20`.
- **Line 4**: `y` then gets assigned a new value of 21 (`20 + 1`).
- **Line 5**: `y` was just changed to 21, so `x` becomes 22 (`21 + 1`).
- **Line 6**: `x` is now 22, so `y` becomes `2 * 22`, or 44.
- One important thing to mention here is that **at no point is any la**

Try It!

- Give the following challenges a try — see if you can predict the final values of x, y, and z.
- Check your answers in JS Bin by copying the entire chunk of code into the editor window, running it, and then checking x,y, and z in the JS Bin console by typing out each variable name and hitting the return key.

Try It: Challenge 1

- **Challenge 1**
- `var x = 1;`
- `var y = 2;`
- `var z = 3;`
- `x = y;`
- `y = z;`
- `z = x;`

Try It: Challenge 2

- **Challenge 2**
- `var x = 1;`
- `var y = 0;`
- `var z = -1;`
- `x = y + z;`
- `y = z * x;`
- `z = x - y;`
- `x = y * y;`
- `y = z * z;`
- `z = z - 1;`
-
- Whoa! That last one's pretty weird — how can `z` be on both sides of the `=`? What do you think happens there?
- The key is remembering how the `=` operator works. Before it assigns anything to the variable on the left, *it first evaluates the expression on the right*.
- This means that, if we have any expression like `x = x + 1;`, what we are doing is taking the old value of `x`, adding `1` to it, and storing this new result back in `x`. In short, we are "incrementing" `x`: increasing its value by `1`, no matter its original value.
-

Exercise

- Write the code to perform the actions listed below in the **JavaScript** panel in the JS Bin editor.
- In the "JavaScript" panel, declare (create) a variable `myNumber`. Assign it the value `30`.
- After declaring the `myNumber` variable in the "JavaScript" panel, be sure to check to make sure you've done things correctly by hitting "Run" in the "Console" panel and then typing `myNumber;` in the "Console" panel and hitting the return/enter key to check its value. You'll want to do this after each step.
- Note: Ensure the *type* of this value is correct — Remember `30` and `'30'` are not the same! Here we want to store a number, so make sure that there are no quotes around the value.
- Reassign (update) the `myNumber` variable to `20`.
- Use the `+=` operator to add 5 to the current value of `myNumber`.
- Now create a second variable `greeting` and assign (give) it the value `"Hello"`.
- Create a third variable `name` and assign it the value `"Margaret"`.
- Create a fourth variable `sayHello`. We want the variable to hold the value `"Hello Margaret"`. Use the variables `greeting` and `name` along with the `+` to create this value (referred to as string concatenation).
- Stuck? Check out the solutions in the Study Guide at the end of this lesson. Not stuck? [slow clap] My you're a quick study, aren't you? See you in the next lesson.
-