Javascript In Less Than 50 Pages

Gabriel Torkel

This book was done for beginners who wants to learn or who are struggling to understand basic concepts of Javascript.

Unlike boring tutorials and 8+ hours long videos, our sessions are quick and simple to read, practice and understand.

With focus on the understanding of the most important subjects, avoiding at this stage exploring concepts that are too complex and confusing for the beginner.

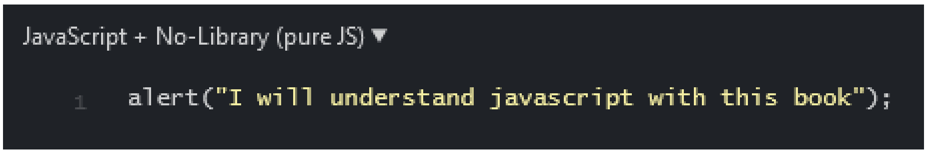
To learn with this book, type and run the codes as described in each session. Typing and running the codes, even if repetitive, will make the new knowledge stick like almost no other method will.

How to Tell Your Computer To Do Things

Go to jsfiddle.net and type the *exact* green text below in the Javascript field:

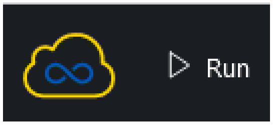
alert("I will understand javascript with this book");

Should look like this:

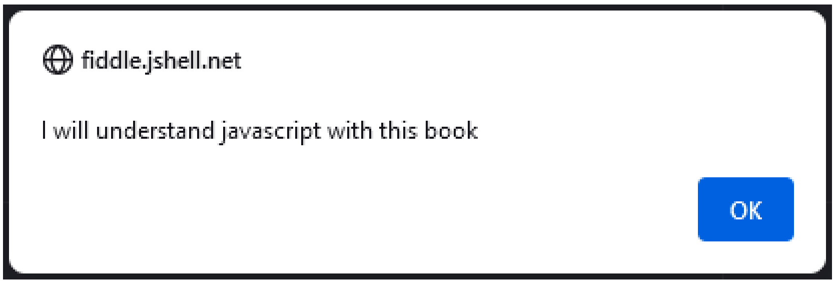


```
alert("I will understand javascript with this book");
```

Then click Run at the top left of the page:



It displays a window like this after you click on **Run**:



fiddle.jshell.net

I will understand javascript with this book

Congratulations on writing your first Javascript code!

This is an alert window.

It was shown because you “told” the computer to display it. How?

To communicate with people, we need a language they understand. In this book, we are using **English**.

In the same way, to “communicate” with the computer we use a language the computer can understand, in this situation, **Javascript**.

The code tells this to the computer:

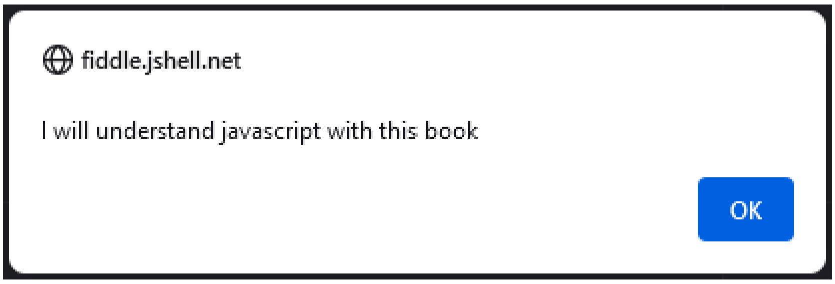
alert("I will understand javascript with this book");

Yellow part: “I want you to display an alert window…”

alert("I will understand javascript with this book");

Blue part: “with the message: I will understand javascript with this book”

And we get the result:



fiddle.jshell.net

I will understand javascript with this book

Javascript, as a programming language, performs acts we tell them to perform.

To communicate with people we use **phrases**.

To communicate with computers, to tell them to do things, we use **lines of command**.

*Every game you play, every app you use, every website you visit is made through a series of **lines of commands**.*

Just like the alert window we saw.

The **alert("message");** line is a **line of command** to perform a specific action of displaying an alert window with a specific message.

Now go to jsfiddle.net, delete any code in the Javascript field, type this and click to run:

alert("javascript can be easy and fun to learn");

As you see, the message in the screen has changed. It now displays a new message.

The part of the code between the double quotes **" "** can be modified to display a message we want.

You can go to jsfiddle.net and type your name in it and see it on the screen, or the name of your pet, or your town.

Our Communication Needs to be "Perfect"

After that, delete any code and type this:

alert("I hope this alert window works);

What happens after you click to run the code?

Exactly, nothing happens.

That's because we wrote a code that is incorrect.

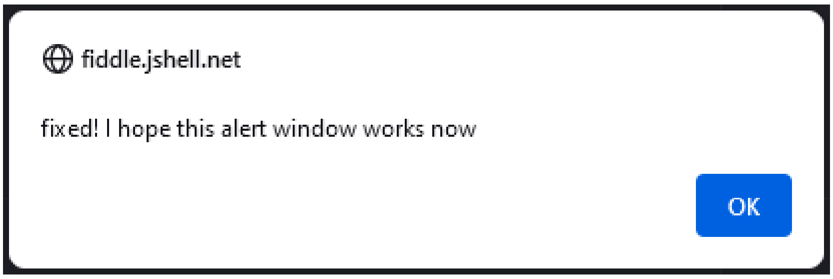
That the computer cannot understand.

Want to guess what is wrong?

If you said there should be a double quote right before the parentheses, you are correct! Let's run again:

alert("fixed! I hope this alert window works now");

Result:



fiddle.jshell.net

fixed! I hope this alert window works now

And there we have our alert window again!

Computers can do what we ask them to do, but we have to be very precise with our coding. If there is a mistake (like we saw above), the code doesn't work.

And the formula for telling the computer to display an alert message through Javascript, using double quotes is:

alert("I'm learning already!");

Single Quotes and Double Quotes Can Work

Try to guess what you think will happen to this line (go to jsfiddle.net and type it):

alert(typing and hoping it works);

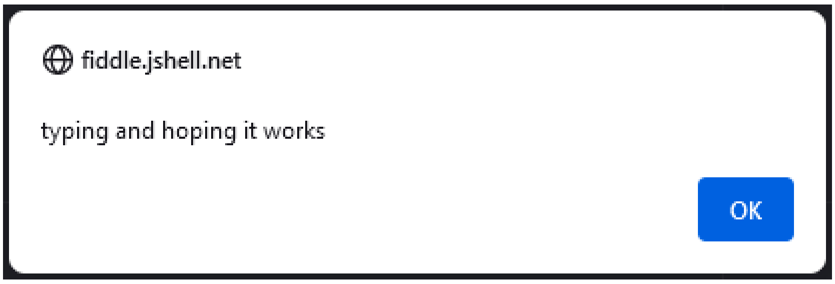
It doesn't work. As you learned, we need to add quotes around the messages for our code to be correct.

And we have to type the code correctly, otherwise it won't work.

Let's check if this line of command works on jsfiddle.net:

alert('typing and hoping it works');

Result:



fiddle.jshell.net

typing and hoping it works

It worked! As you see, if we add single quotes **' '** instead of double quotes **" "**, the code still works.

This happens because Javascript allows us to use both double quotes OR single quotes whenever we are typing a text.

We Have To Be Really Precise

Type and run it:

Alert('this A is upper case');

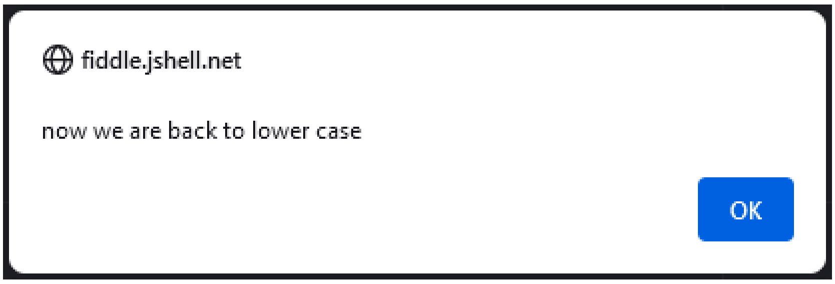
Now delete all code, type and run this:

ALERT('now all letters are upper case');

Again, delete, type, and run this:

alert('now we are back to lower case');

Result:



fiddle.jshell.net

now we are back to lower case

Only the third line works.

This is because in the first and in the second lines of command, we are not using the correct word **alert** in lower case.

In other words, the computer will only display an alert window if the code is correct:

Alert('any message');
doesn't work

ALERT('any message');
doesn't work

alert('any message');
works

This one is easy, try to guess what will happen. Run this code:

alert['I probably know what will happen'];

As you learned, the code doesn't work because it is not following the formula to the alert window:

alert("I'm learning already!");

or, with single quotes:

alert('I'm learning already!');

If A Condition Is Not Met

Type and run this code:

```
if (80 > 100) {
alert('Egyptian Pyramids');
}
```

Our code is correct, but nothing happens. Why we don't see the message **Egyptian Pyramids**?

Because in the first line, which is this one:

```
if (80 > 100) {
```

We add a condition for the computer to execute the second next line of command, where our alert window is.

Our code is telling the computer this:

if (80 > 100) {
***if** 80 is bigger than 100, then…*
alert('Egyptian Pyramids');
}
display an alert window with the message “Egyptian Pyramids”.

Obviously, **80 is not bigger than 100**.

The **condition** of the line of command **if (80 > 100) { was not met**.

It will never be met because 80 will never be bigger than 100. Ever.

How can we see **'Egyptian Pyramids'** in the alert window?

If A Condition Is Met

Type and run it:

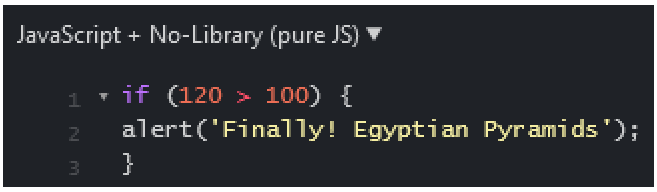
```
if (100 > 100) {
alert('Egyptian Pyramids');
}
```

Still doesn’t work, this is because our condition has not been met yet: **100 is not bigger than 100, it’s equal to it**.

It’s an easy guess now, run and try it:

```
if (120 > 100) {
alert('Finally! Egyptian Pyramids');
}
```

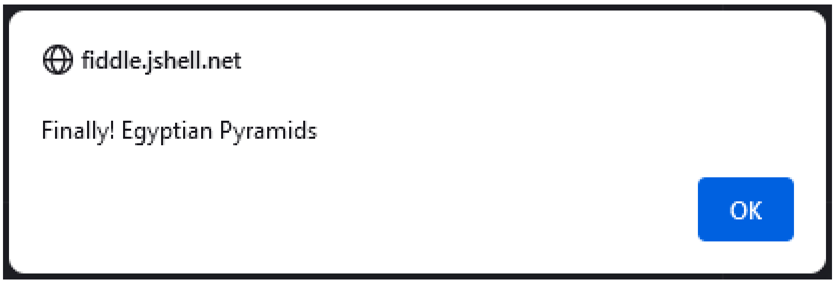
Should look like this on jsfiddle.net:



JavaScript + No-Library (pure JS) ▼

```
if (120 > 100) {
alert('Finally! Egyptian Pyramids');
}
```

Result:



fiddle.jshell.net

Finally! Egyptian Pyramids

Easy, right? We can now see the message in the alert window because our condition was met: **120 is bigger than 100**.

The structure for the **if** command is:

```
if (condition is met) {
   execute the lines of command
   inside these curly brackets
}
```

Notice in the first line we start with **if**.

Then we add the condition between parentheses **( )**.

Next, we open a Curly Bracket **{**.

Inside the Curly Brackets, we add all the lines of command we want to execute ONLY if the condition between the parentheses is met.

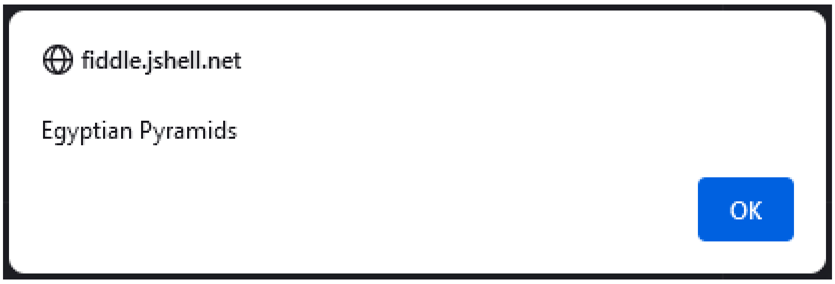
Then we close the Curly Bracket **}**.

Making it Easier for the Human To Read

Code to run:

```
if (120 > 100) { alert('Egyptian Pyramids'); }
```

Result:



fiddle.jshell.net

Egyptian Pyramids

Maybe you thought the code would not work. But it **did**.

This is because we can add the if condition in one line. And it is ok in the example above because we have a small command inside the curly brackets.

But in most of the cases we don’t really want to use one line to code an **if** command.

Check this code below (we don’t need to run it on jsfiddle.net this time):

```
if (120 > 100) { alert('Egyptian Pyramids'); alert('Niagara Falls'); alert('French Quarter'); alert('Mount Rushmore'); alert('Yellowstone'); }
```

This Javascript code is not incorrect, in other words, **the computer can understand it**.

However, it’s NOT so good visually for the person who needs to eventually edit the code. It **makes it more challenging for the human to understand it because is poorly organized**.

This is why we adopt good practices in coding javascript.

For the **if** command, this is what we do.

In our example in blue above we start by isolating the first **if + (command)** line, until the **opening curly bracket**:

```
if (120 > 100) {
alert('Egyptian Pyramids'); alert('Niagara Falls'); alert('French Quarter');
alert('Mount Rushmore'); alert('Yellowstone'); }
```

Next, we add 1 **alert();** command per line:

```
if (120 > 100) {
alert('Egyptian Pyramids');
alert('Niagara Falls');
alert('French Quarter');
alert('Mount Rushmore');
alert('Yellowstone'); }
```

Then, we isolate the **closing curly bracket** after the last **alert();** command:

```
if (120 > 100) {
alert('Egyptian Pyramids');
alert('Niagara Falls');
alert('French Quarter');
alert('Mount Rushmore');
alert('Yellowstone');
}
```

And finally, we press 4 times the space bar on the keyboard before each line between the curly brackets:

```
if (120 > 100) {
    alert('Egyptian Pyramids');
    alert('Niagara Falls');
    alert('French Quarter');
    alert('Mount Rushmore');
    alert('Yellowstone');
}
```

Note: pressing 1 time the TAB key instead of 4 times the space bar works in many coding editors (including on jsfiddle.net's). Whenever possible, pressing the TAB Key is preferred to organize lines of code instead of the space bar.

Which of these codes are easier to read?

This one:

```
if (120 > 100) { alert('Egyptian Pyramids'); alert('Niagara Falls'); alert('French
Quarter'); alert('Mount Rushmore'); alert('Yellowstone'); }
```

Or this one:

```
if (120 > 100) {
    alert('Egyptian Pyramids');
    alert('Niagara Falls');
    alert('French Quarter');
    alert('Mount Rushmore');
    alert('Yellowstone');
}
```

Our second code is better organized and easier to read and edit. And the computer can execute it the same way.

This is how our coding should be:

I - Correct, so the computer understands it and performs what we want.
II - Easy for the human to read and edit it.

Having More Than One Condition

Run this code:

```
if (50 > 100) {
    alert('Niagara Falls');
}
else if (30 > 70) {
    alert('Yellowstone');
}
else if (20 > 30) {
    alert('Mount Rushmore');
}
```

We don't see any alert window because none of our conditions are met:

- 50 is not bigger than 100, therefore, we don't see **Niagara Falls**.

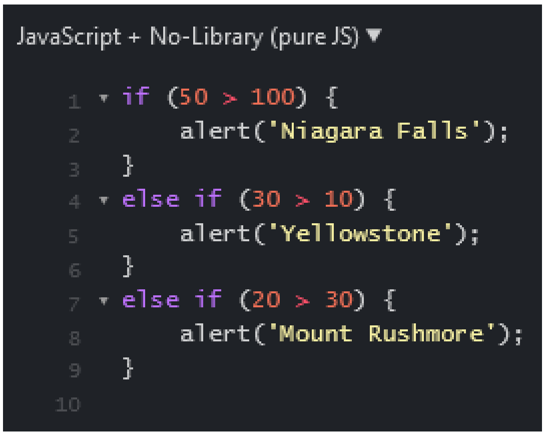
- 30 is not bigger than 70, therefore, we don’t see **Yellowstone**.
- 20 is not bigger than 30, therefore, we don’t see **Mount Rushmore**.

If the second condition is met, we can see the alert window.

Run this code:

```
if (50 > 100) {
   alert('Niagara Falls');
}
else if (30 > 10) {
   alert('Yellowstone');
}
else if (20 > 30) {
   alert('Mount Rushmore');
}
```

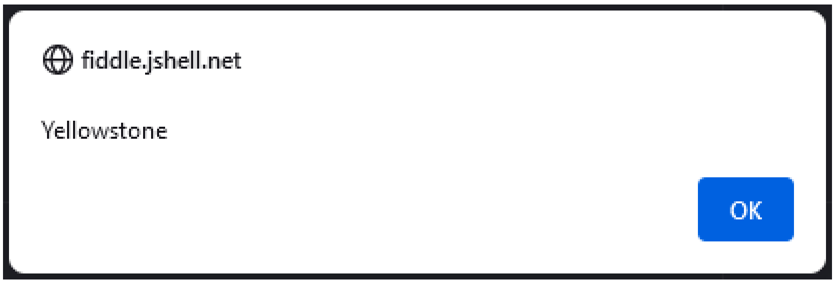
Should look like this:



JavaScript + No-Library (pure JS) ▼

```
if (50 > 100) {
     alert('Niagara Falls');
}
else if (30 > 10) {
     alert('Yellowstone');
}
else if (20 > 30) {
     alert('Mount Rushmore');
}
```

Result:



fiddle.jshell.net

Yellowstone

The **if** + **else if** lines of command allow us to add more than one condition.

The first conditional command one should always be **if (condition 1)**.

The others should be **else if (condition 2)**, **else if (condition 3)**, and so on.

Always using curly brackets to execute the commands related to a condition:

```
if (50 > 100) {
   alert('Niagara Falls');
}
else if (30 > 10) {
```

We use the curly bracket for each ***else if***

```
   alert('Yellowstone');
```

Then we add the commands to execute if the condition is met

```
}
```

We close it with a curly bracket

```
else if (20 > 30) {
   alert('Mount Rushmore');
}
```

If No Conditions are Met

Run this code:

```
if (50 > 100) {
   alert('Niagara Falls');
}
else if (30 > 70) {
   alert('Yellowstone');
}
else if (20 > 30) {
   alert('Mount Rushmore');
}
else {
   alert('No place to display');
}
```

Because none of the conditions has been met we gave the computer no option but to show the alert window with the message: **No place to display**.

This happens because we use the **else** command.

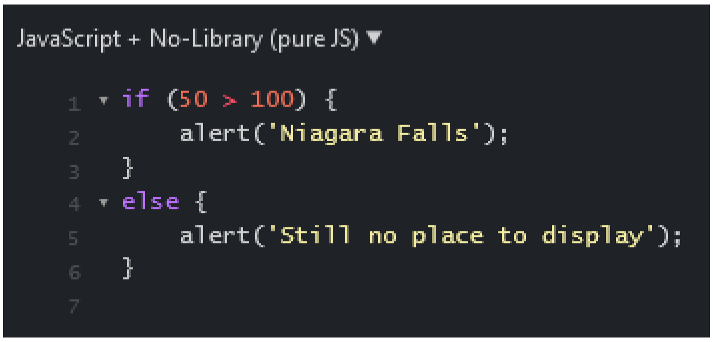
It is the final option for the code.

Notice it doesn’t have parentheses with any condition.

Now let’s simplify it:

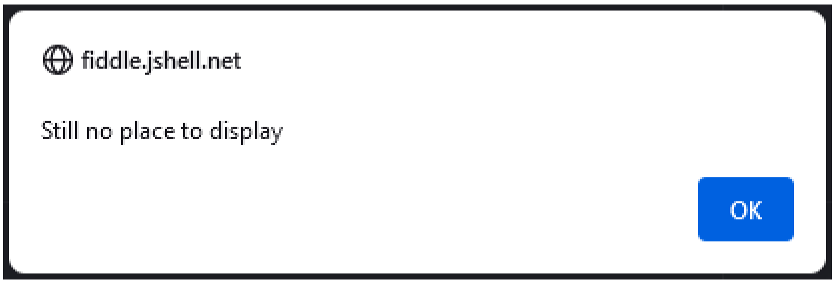
```
if (50 > 100) {
   alert('Niagara Falls');
}
else {
   alert('Still no place to display');
}
```

Should look like this:



```
if (50 > 100) {
     alert('Niagara Falls');
}
else {
     alert('Still no place to display');
}
```

Result:



Still no place to display

Again, we don’t see **Niagara Falls** in the alert window because our condition was not met.

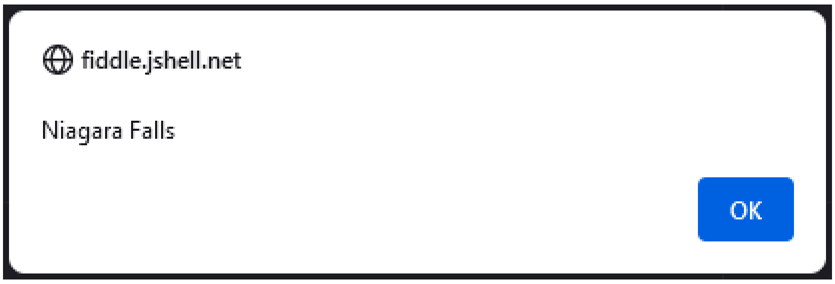
And because no previous conditions have been met, we saw the message: **No place to display still**.

If A Condition Is Met And We Have An Else

Let’s run this code:

```
if (120 > 100) {
   alert('Niagara Falls');
}
else {
   alert('No place to display');
}
```

Result:



Niagara Falls

Now, as you noticed, because the **if** condition 120 > 100 was met, we saw the message **Niagara Falls**.

We did not see the message in the **else** brackets.

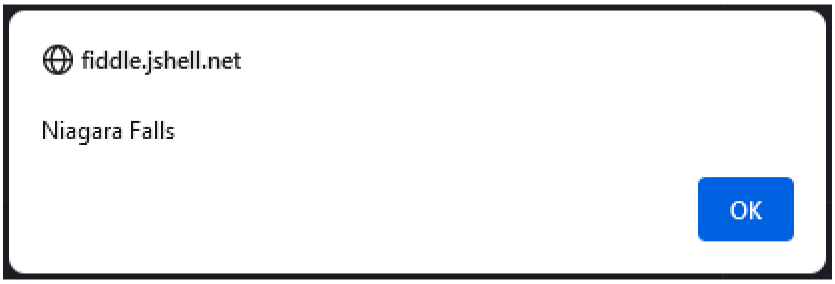
This is because if our previous condition is met, we don't get to execute the lines of command inside the **else** brackets.

If Any Condition Is Met

In this code, the three conditions are met. What do you think will happen? Run it and check:

```
if (120 > 100) {
   alert('Niagara Falls');
}
else if (80 > 70) {
   alert('Yellowstone');
}
else if (50 > 30) {
   alert('Mount Rushmore');
}
else {
   alert('No place to display');
}
```

Result:



fiddle.jshell.net

Niagara Falls

As you see, only the alert message of the first condition is shown: **Niagara Falls**.

- **120 is bigger than 100**, therefore, we see a message with **Niagara Falls**.
- **80 is bigger than 70**, however, we don't see a message with **Yellowstone** because there was already one previous condition met.
- **50 is bigger than 30**, however, we don't see a message with **Mount Rushmore** because there was already one previous condition met.

This is because if we have any condition met, the code "stops" looking for newer answers.

The goal of an **if** + **if else** + **else** commands set is to find ONE condition that is met.

If the first condition is met, it "skips" the **second**, **third**, and the **else** commands.

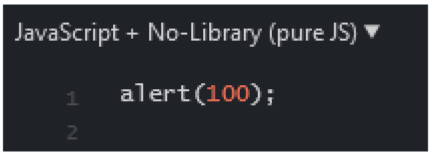
If the first condition is not met, the computer checks the second. If the second is met, it skips the third and the **else** commands.

Numbers and Texts

Think what can happen if we run this code and try it:

alert(100);

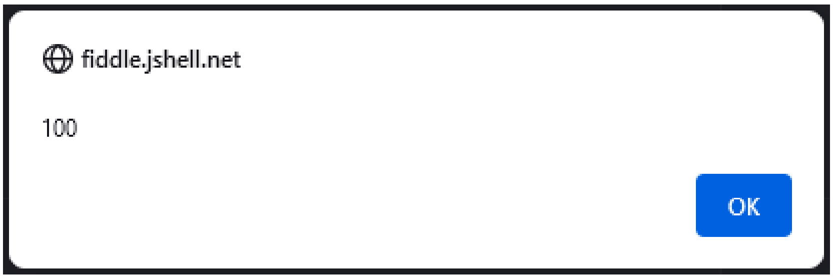
Should look like this:



JavaScript + No-Library (pure JS) ▼

```
alert(100);
```

Result:



fiddle.jshell.net

100

Maybe you thought that it would not work, because we did not have single quotes **' '** nor double quotes **" "**.

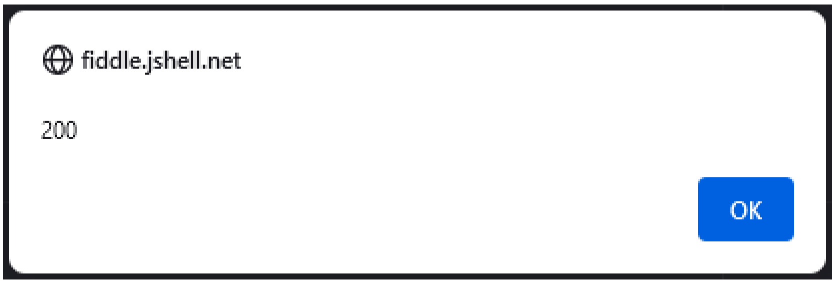
But it worked, and we got **100** as the result.

If we are writing numbers only on the alert window we do not use quotes.

Now try this code:

alert(100 + 100);

Result:



fiddle.jshell.net

200

Maybe you thought we would see 100 + 100 in the message of the alert window. But instead, we saw **200**, the result of **100 + 100**.

This happens because when we write numbers with operation symbols like addition **+** and subtraction **-**, it will give us the result of that operation. Like you see here:

alert(70 + 30);
Addition

alert(100 - 20);
Subtraction

alert(100 * 2);
Multiplication

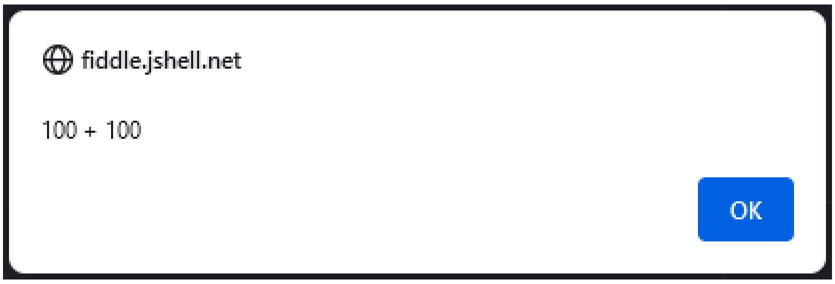
alert(20 / 2);
Division

Text Strings

Now read this line, and try to guess what will happen once we run it:

alert('100 + 100');

Result:



fiddle.jshell.net

100 + 100

Interestingly, we do not get the result of a sum of **100 + 100**, which would be **200**.

We get **100 + 100** in the alert window. Why?

What was the difference between the **alert('100 + 100');** line and the first code of the last session, **alert(100 + 100);** ?

Now we are using single quotes. In other words, we transformed the **100 + 100** in a **text**, not a **number**.

In Javascript, we call a portion of text in the same line a **text string,** or just **string**.

A few examples with the alert box:

alert(50 + 50);
*Message is a **number** (we have a sum and no quotes)*

alert('50 + 50');
*Message is a **string** (we have a text around quotes)*

alert(30 * 20);
*Message is a **number** (we have a multiplication and no quotes)*

alert('30 + 30 equals 60');
*Message is a **string** (we have a text around quotes)*

Variables

Let’s run this line:

alert("Niagara Falls");

Result:

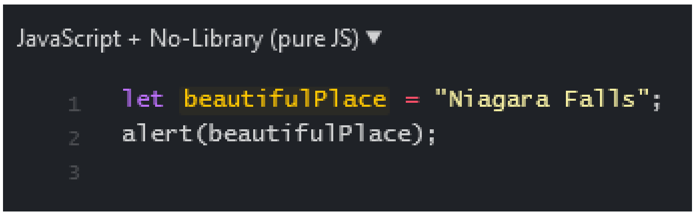
fiddle.jshell.net

Niagara Falls

And now we delete the codes and try these lines:

let beautifulPlace = "Niagara Falls";
alert(beautifulPlace);

Should look like this:



```
let beautifulPlace = "Niagara Falls";
alert(beautifulPlace);
```

Result:

fiddle.jshell.net

Niagara Falls

Both results are the same.

In the second code, we did not type **“Niagara Falls”** inside the alert message parentheses, but we go it as a result. What happened?

In the first line, we used a **Variable**, starting with the word **let**.

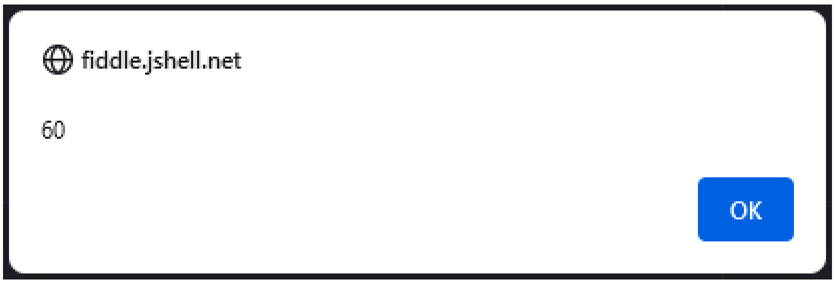
We named this variable **beautifulPlace**.

And we attributed a value to this variable. This value is the text string **"Niagara Falls"**.

Let’s try another code:

```
alert(30 + 30);
```

Result:



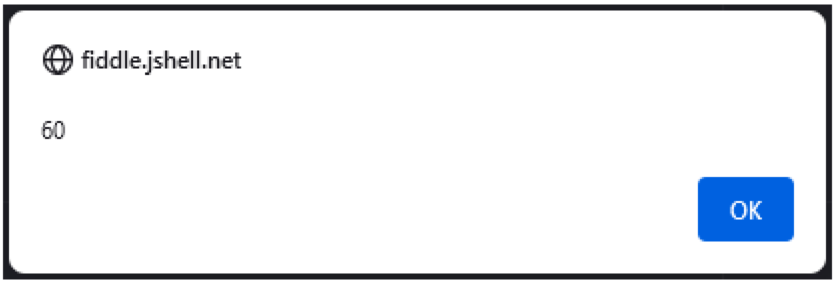
fiddle.jshell.net

```
60
```

Then delete the code and try this one:

```
let randomSum = 30 + 30;
alert(randomSum);
```

Result:



fiddle.jshell.net

```
60
```

You probably notice how the variables work. In simple terms, all they do is hold a value that we want.

This helps in making our code dynamic.

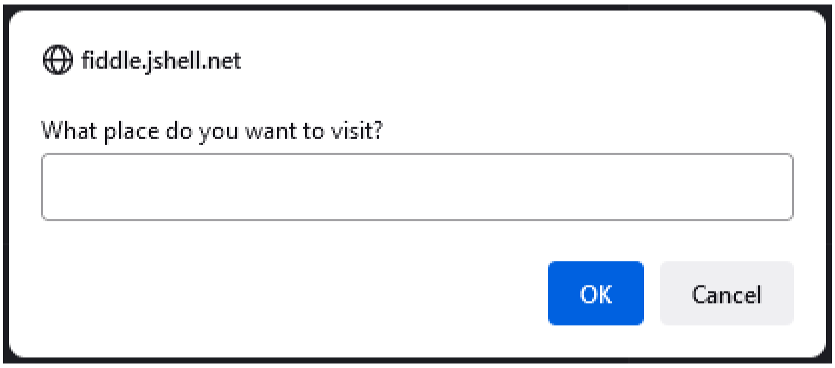
Let’s understand better how variables help by learning another command.

Prompt Window

Let’s run this line:

```
prompt("What place do you want to visit?");
```

Result:



fiddle.jshell.net

What place do you want to visit?

What we see here is a prompt window. It asks us to enter information. In this case, it asks us **"What place you want to visit?"**.

If you did type any answer in the box, nothing happened, because we need more lines of code for the computer to perform actions.

Now let's run these lines from the last session:

```
let beautifulPlace = "Niagara Falls";
alert(beautifulPlace);
```

Result:

fiddle.jshell.net

Niagara Falls

Again, we see **"Niagara Falls"** because we attributed it as value to the variable **beautifulPlace**.

And now let's delete and run these lines. Type any place you want as an answer:

```
let beautifulPlace = prompt("What place you want to visit?");
alert(beautifulPlace);
```

First you saw a prompt window with the message "What place you want to visit?".

You then saw the place you typed in an alert message.

By now you probably see how a **variable** makes the page more dynamic and how we can interact with it more.

Let's try a few more examples:

```
let myFirstPet = prompt("What is the name of your first pet?");
alert(myFirstPet);
```

```
let myBestTrip = prompt("What is the best trip you've done?");
alert(myBestTrip);
```

```
let myFavoriteSinger = prompt("What is the name of your favorite singer?");
alert(myBestTrip);
```

Building a Function

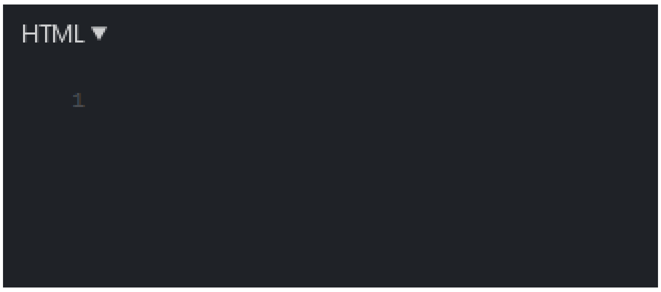
Let's run this line:

```
function askMeAboutMyTrip() {
   let myBestTrip = prompt("What is the best trip you've done?");
   alert(myBestTrip);
}
```

This code is correct. But nothing we cannot see a prompt window or an alert window.

This is because we have only **declared the function**. We did not call it yet.

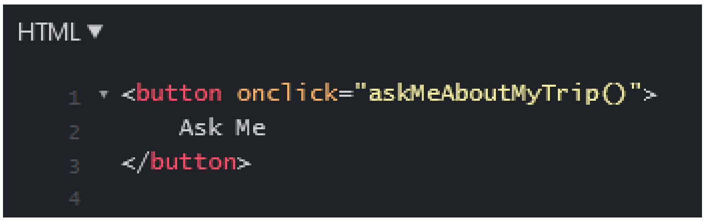
Now, do not delete the last code from jsfiddle.net. Just go to the HTML field:



Type this code in this field:

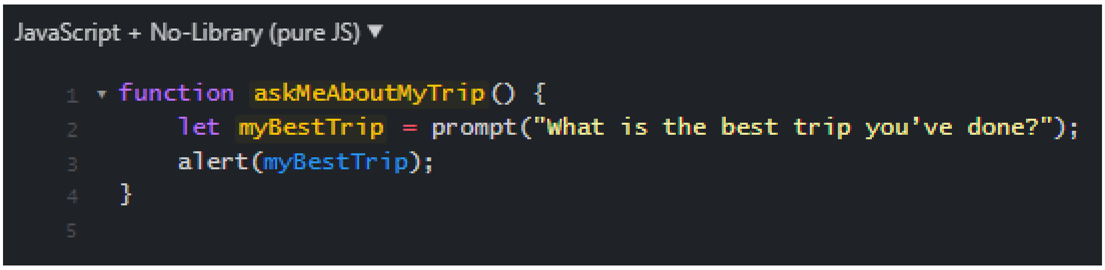
```
<button onclick="askMeAboutMyTrip()">
   Ask Me
</button>
```

Should look like this:



```
<button onclick="askMeAboutMyTrip()">
    Ask Me
</button>
```

The Javascript field should have this code:



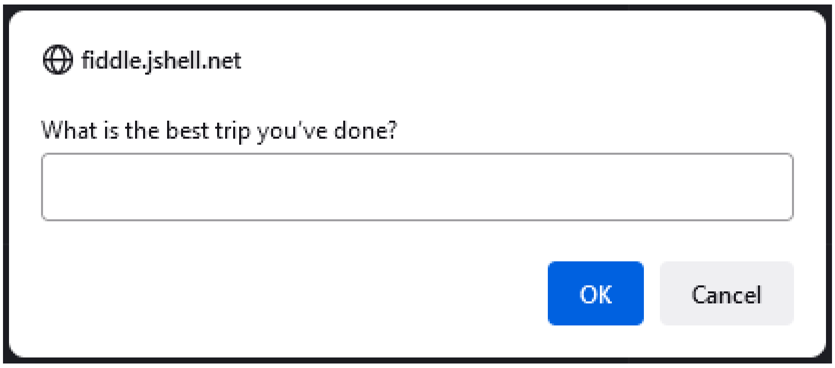
```
function askMeAboutMyTrip() {
    let myBestTrip = prompt("What is the best trip you've done?");
    alert(myBestTrip);
}
```

Now click on Run.

We still did not see a prompt window, but in **Results** on jsfiddle.net we see a button that says “Ask Me”:

Click on this button.

Now we have our prompt window:


fiddle.jshell.net

What is the best trip you've done?

Type an answer and click on **OK**. You will see an alert window with your answer.

You just did your first **Javascript Function**. And that’s a big deal.

Functions are extremely important in programming languages.

It makes the project much more powerful.

How did we do it? And how was it triggered?

Declaring the Existence of a Function

First, we created a function and declared it. We started by using the keyword **function** in the first line:

function

Next, we named the function. We used a very intuitive name for what the function was going to do: **askMeAboutMyTrip.**

```
function askMeAboutMyTrip
```

And after naming it we added two parentheses **()**. Because in Javascript we use those to declare a function.

```
function askMeAboutMyTrip()
```

Still on the first line, we use a curly bracket **{**.

```
function askMeAboutMyTrip() {
```

Then we add the lines of command between the curly brackets **{** and **}** with the actions we want to be performed if the function is executed.

```
function askMeAboutMyTrip() {
   let myBestTrip = prompt("What is the best trip you’ve done?");
   alert(myBestTrip);
```

And finally we add the curly bracket to close the function:

```
function askMeAboutMyTrip() {
   let myBestTrip = prompt("What is the best trip you’ve done?");
   alert(myBestTrip);
}
```

Executing the Function We Declared

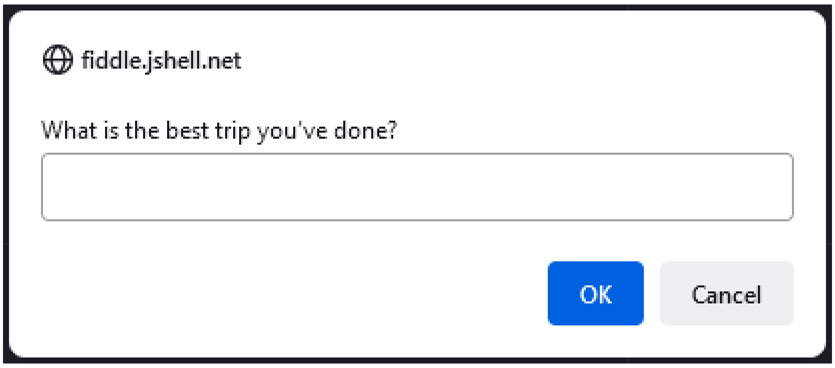
All we have to do is to type the name of the function we created (with the parentheses):

```
askMeAboutMyTrip();
```

Run it on jsfiddle.net:

```
function askMeAboutMyTrip() {
   let myBestTrip = prompt("What is the best trip you’ve done?");
   alert(myBestTrip);
}

askMeAboutMyTrip();
```

Result:



fiddle.jshell.net

What is the best trip you've done?

With such lines of code, as soon as a visitor arrives on our website, they would see the prompt window.

You don’t see many prompt windows out there when you’re browsing websites.

Because the prompt window is rarely used this way. This is why we add a button with the option to execute the function if we want.

Keep the code you typed on jsfiddle.net in the JS field.

And add this code we used previously in the HTML field:

```
<button onclick="askMeAboutMyTrip()">
   Ask Me
</button>
```

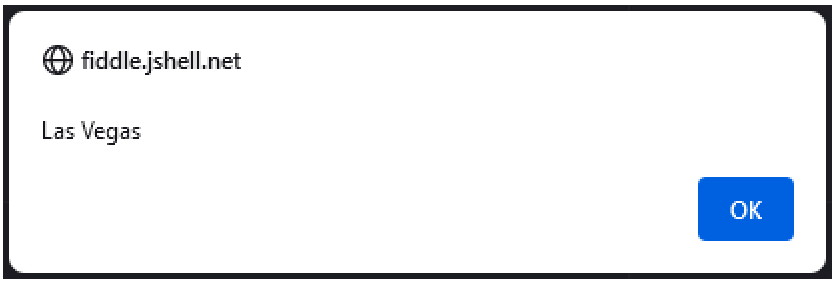
Like we did in the previous session, once we click the button we execute the function **askMeAboutMyTrip()** by interacting with the website.

Function Parameters

Run it on jsfiddle.net:

```
function myFavoriteTown(city) {
   alert(city);
}

myFavoriteTown("Las Vegas");
```

Result:



fiddle.jshell.net

Las Vegas

As you see, we created the function **myFavoriteTown()**;

But unlike the example of the last session, the parentheses of the first line of the function are not empty:

```
function myFavoriteTown(city) {
```

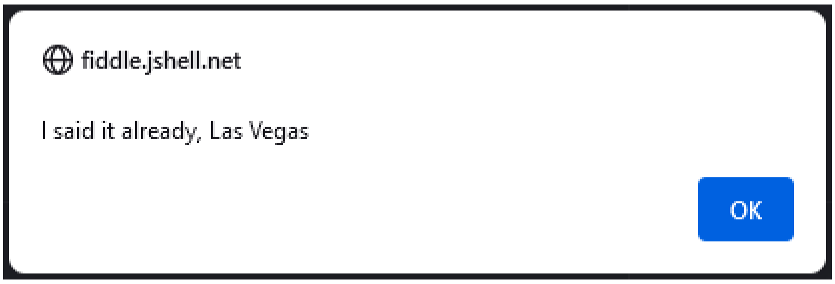
It has what we call a **parameter**.

Parameters allow us to add information when executing a function. This makes our coding more flexible.

Run this code now:

```
function myFavoriteTown(city) {
   alert(city);
}

myFavoriteTown("I said it already, Las Vegas");
```

Result:



fiddle.jshell.net

I said it already, Las Vegas

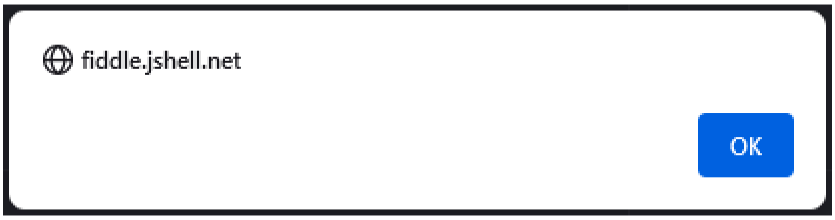
As you probably noticed, for us to see the parameter **"I said it already, Las Vegas"** in an alert window, the **alert();** must have the parameter between the parentheses:

```
   alert(city);
```

Let’s test this one:

```
function myFavoriteTown(city) {
   alert();
}

myFavoriteTown("I said it already, Las Vegas");
```

Result:



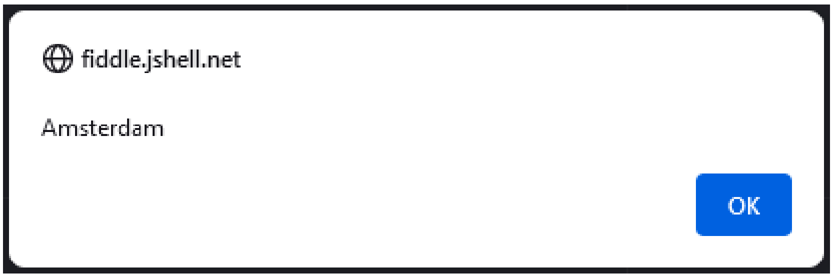
fiddle.jshell.net

As you can see, we get an empty alert window, because there is neither a message nor parameter nor variable in the **alert()** code inside the function **myFavoriteTown()**.

We add again the same parameter **city** from the first line to the **alert()** command inside the function:

```
function myFavoriteTown(city) {
   alert(city);
}

myFavoriteTown("Amsterdam");
```

Result:



fiddle.jshell.net

Amsterdam

The Alert Command is a Function With A Parameter

By now you probably noticed the similarity between the function we created:

```
myFavoriteTown(city);
```

And the alert command:

```
alert("I will learn Javascript with this book");
```

Both are executed by writing them in a separate line of command with this structure:

```
FUNCTION_NAME(PARAMETER);
```

An important difference is that the **alert()** function is a built-in function, that already exists in Javascript.

In other words, we do not need to declare it and build its functionality.

The **myFavoriteTown()** function is created by us to perform a specific action in our code.

You may be familiar with the **prompt()** function as well. It's another built-in function.

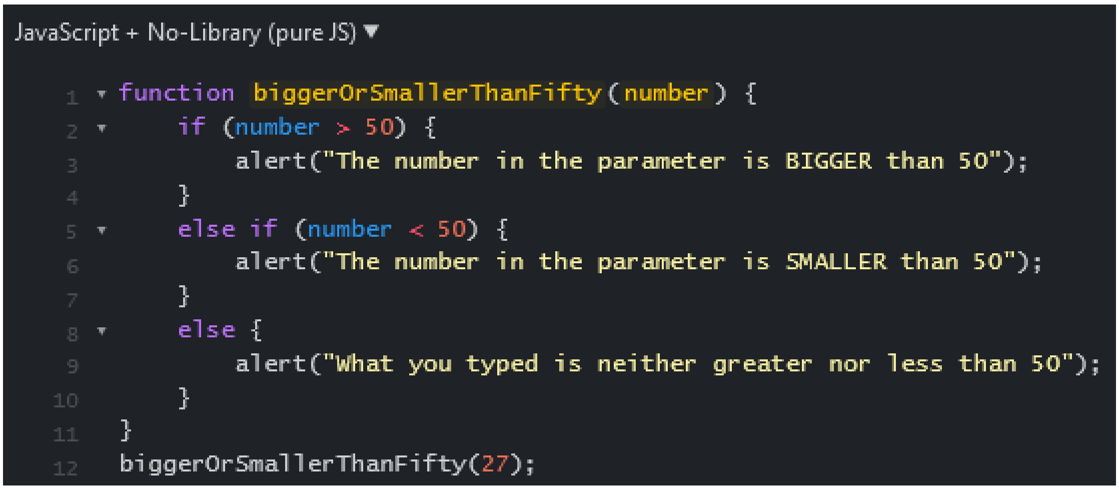
Function With Conditions…

Let’s run the code below:

```
function biggerOrSmallerThanFifty(number) {
   if (number > 50) {
      alert("The number in the parameter is BIGGER than 50");
   }
   else if (number < 50) {
      alert("The number in the parameter is SMALLER than 50");
   }
   else {
      alert("What you typed is neither greater nor smaller than 50");
   }
}
```

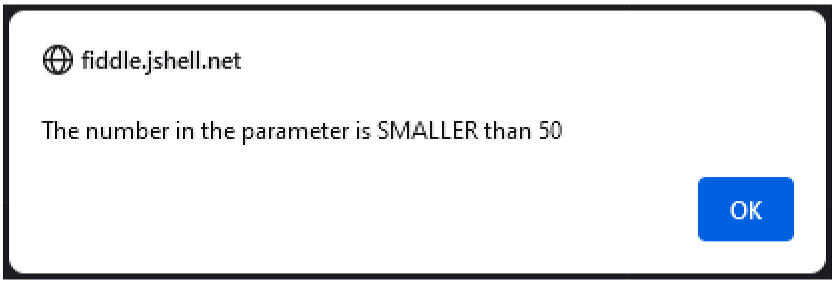
```
biggerOrSmallerThanFifty(27);
```

Should look like this on jsfiddle.net:



```
function biggerOrSmallerThanFifty(number) {
     if (number > 50) {
          alert("The number in the parameter is BIGGER than 50");
     }
     else if (number < 50) {
          alert("The number in the parameter is SMALLER than 50");
     }
     else {
          alert("What you typed is neither greater nor less than 50");
     }
}
biggerOrSmallerThanFifty(27);
```

Result:

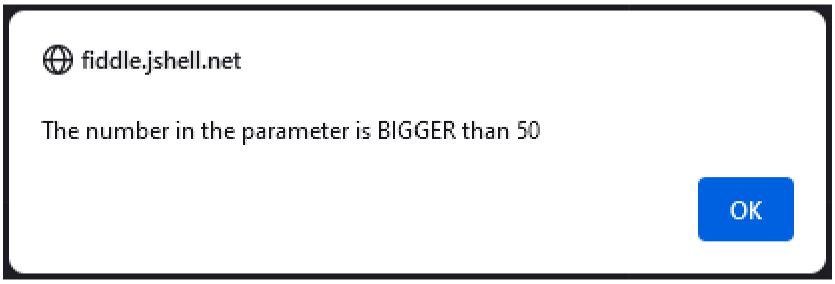


The number in the parameter is SMALLER than 50

After you run, delete the last line and write this one instead:

```
biggerOrSmallerThanFifty(70);
```

Result:



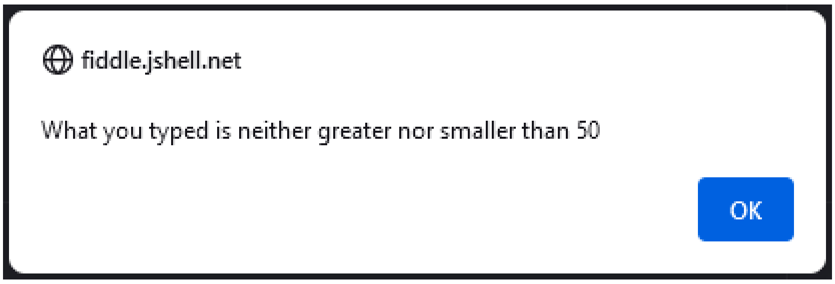
fiddle.jshell.net

The number in the parameter is BIGGER than 50

After you run again, delete the last line and write this one instead:

biggerOrSmallerThanFifty(50);

Result:



fiddle.jshell.net

What you typed is neither greater nor smaller than 50

We just built a function slightly more complex than the one we saw in previous sessions.

We did add conditions through the keywords **if**, **else if**, and **else**.

function biggerOrSmallerThanFifty (number) {
*Declares a function with the name **biggerOrSmallerThanFifty**, with the parameter **(number)***

if (number > 50) {
Starts with a condition, if number > 50, then…

alert("The number in the parameter is BIGGER than 50");
This alert window is shown

}

else if (number < 50) {
The second condition, if number < 50, then…

alert("The number in the parameter is SMALLER than 50");
This alert window is shown

}

else {
In case none of the previous conditions are met

alert("What you did type is not bigger nor smaller than 50");
This alert window is shown

}

}

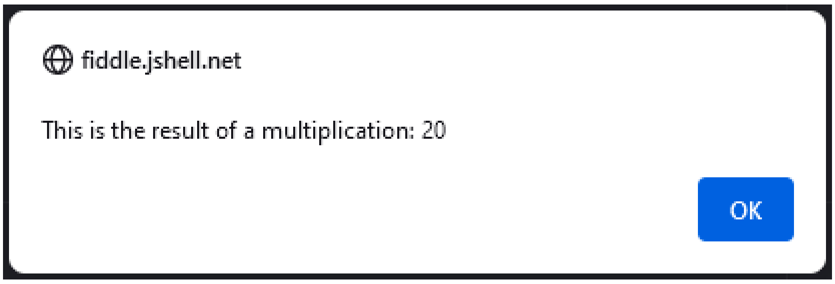
biggerOrSmallerThanFifty(27);
*We execute the function **biggerOrSmallerThanFifty** with the parameter **(27)***

Concatenation

Type and run this code:

alert("This is the result of a multiplication: " + 10 * 2);

Result:



fiddle.jshell.net

This is the result of a multiplication: 20

We get a phrase + the result of the operation **10 * 2** on the same alert message.

This was possible because of what we call **concatenation**.

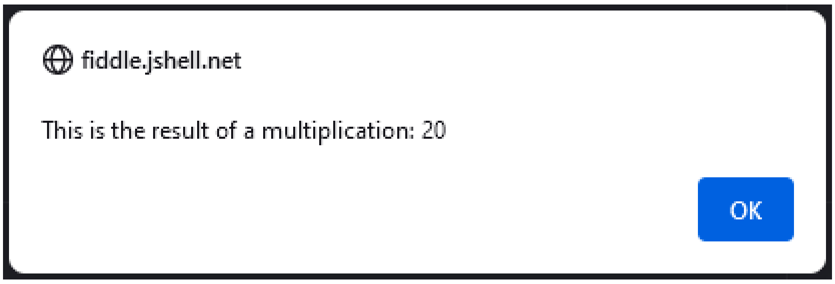
A concatenation happens once we assemble two or more data to form a single **text string**.

We do it by using the plus sign **+**. We simply add it between two or more values.

Let's run this one now:

let multiplication = 10 * 2;
alert("This is the result of a multiplication: " + multiplication);

Result:



fiddle.jshell.net

This is the result of a multiplication: 20

We get the same value as last example because we are using a variable.

This time we did a concatenation between:

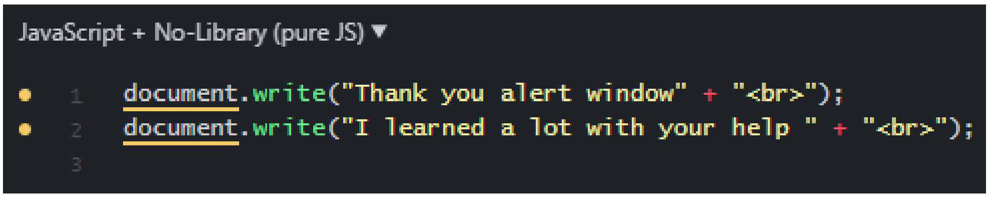
A string (**"This is the result of a multiplication: "**) and a variable (**multiplication**).

Instead Of the Alert Window

Type and run this code:

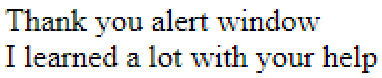
document.write("Thank you alert window" + "<br>");
document.write("I learned a lot with your help " + "<br>");

Should look like this:



```
document.write("Thank you alert window" + "<br>");
document.write("I learned a lot with your help " + "<br>");
```

Result (in the 4th square of jsfiddle.net):



```
Thank you alert window
I learned a lot with your help
```

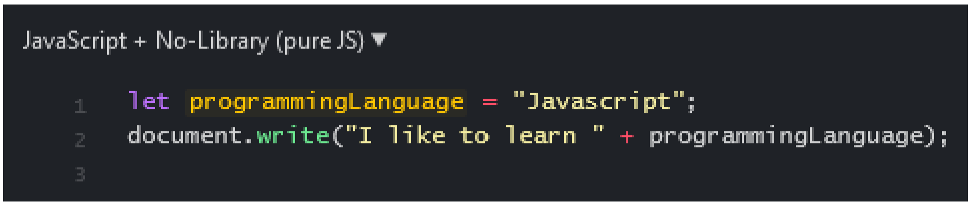
The **alert()** command has helped us so far. But we need a command where we can better visualize the results of our lines of code.

Like we just saw in the code we run above.

Run this code now:

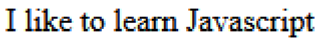
let programmingLanguage = "Javascript";
document.write("I like to learn " + programmingLanguage);

Should look like this:



```
let programmingLanguage = "Javascript";
document.write("I like to learn " + programmingLanguage);
```

Result:



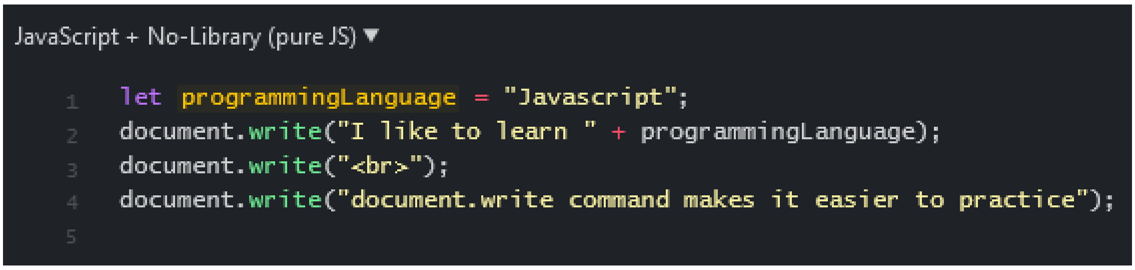
```
I like to learn Javascript
```

As you can see, the command **document.write()** displays the data we add between the parenthesis in the “Result” part of jsfiddle.net

Try this code now:

let programmingLanguage = "Javascript";
document.write("I like to learn " + programmingLanguage);
document.write("<br>");
document.write("document.write command makes it easier to practice");

Should look like this:



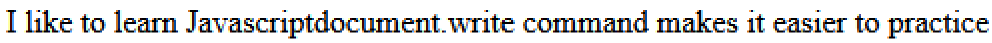
```
let programmingLanguage = "Javascript";
document.write("I like to learn " + programmingLanguage);
document.write("<br>");
document.write("document.write command makes it easier to practice");
```

The **document.write("<br>");** line allows us to add the HTML tag **<br>**, which means line break.

Let’s try the code above without this line and see the result:

```
let programmingLanguage = "Javascript";
document.write("I like to learn " + programmingLanguage);
document.write("document.write command makes it easier to practice");
```

Result:



```
I like to learn Javascriptdocument.write command makes it easier to practice
```

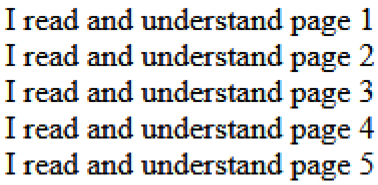
Doesn’t look so good. This is why we add the line break.

Loops

Type and run this code:

```
document.write("I read and understand page " + 1);
document.write("<br>");
document.write("I read and understand page " + 2);
document.write("<br>");
document.write("I read and understand page " + 3);
document.write("<br>");
document.write("I read and understand page " + 4);
document.write("<br>");
document.write("I read and understand page " + 5);
```

Result:



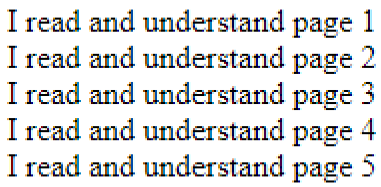
```
I read and understand page 1
I read and understand page 2
I read and understand page 3
I read and understand page 4
I read and understand page 5
```

Our code works but is not the best option to display our message. By far.

As you noticed, we had to make the code very repetitive.

So, how can we get the same result with fewer codes? Through what we call **loops**.

In this example, our **loop** should repeat the string " seconds remaining" 6 times. While at the same time counting from 1 to 5. It gives us this result:

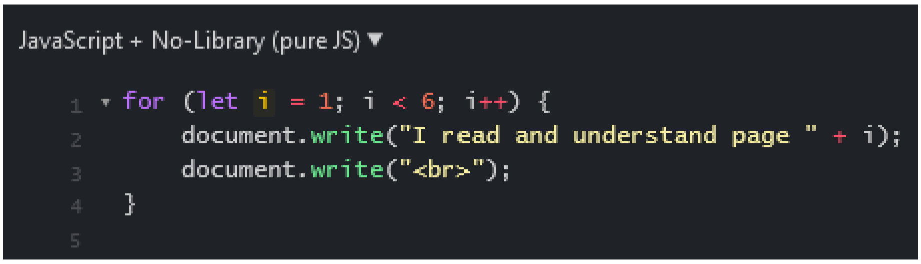


```
I read and understand page 1
I read and understand page 2
I read and understand page 3
I read and understand page 4
I read and understand page 5
```

Write and run this code:

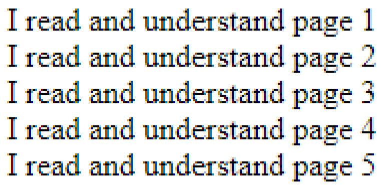
```
for (let i = 1; i < 6; i++) {
   document.write("I read and understand page " + i);
   document.write("<br>");
}
```

Should look like this:



```
for (let i = 1; i < 6; i++) {
    document.write("I read and understand page " + i);
    document.write("<br>");
}
```

Result:



```
I read and understand page 1
I read and understand page 2
I read and understand page 3
I read and understand page 4
I read and understand page 5
```

As you see, we got the same results as before with fewer lines of code:

Both codes give the same result. It was possible because we used a loop. In this case the **for** loop.

Without a Loop:

```
document.write("I read and understand page " + 1);
document.write("<br>");
document.write("I read and understand page " + 2);
document.write("<br>");
document.write("I read and understand page " + 3);
document.write("<br>");
document.write("I read and understand page " + 4);
document.write("<br>");
document.write("I read and understand page " + 5);
```

Or using the **for** loop:

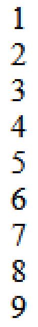
```
for (let i = 1; i < 6; i++) {
   document.write("I read and understand page " + i);
   document.write("<br>");
}
```

The First Line of The “For” Loop

Let’s run this for loop:

```
for (let i = 1; i < 10; i++) {
   document.write(i);
   document.write("<br>");
}
```

This is the result:

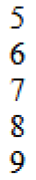


```
1
2
3
4
5
6
7
8
9
```

Now let’s try this one:

```
for (let i = 5; i < 10; i++) {
   document.write(i);
   document.write("<br>");
}
```

We replaced the i = 1 with i = 5 in the first line, and now we see numbers being counted from 5 to 9 as the result of the loop:



```
5
6
7
8
9
```

How does this happens? The **for loop** requires us to add a **starting point**. We call it **index** (which we did in the **let i = 5** part).

This index is the starting point of our loop.

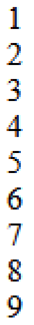
As you see, in the first example, our index was **1**. This is why we got numbers starting counting on **1**.

In the second example, our index was 5, and the result started counting on **5**.

Now let’s run the first example again:

```
for (let i = 1; i < 10; i++) {
   document.write(i);
   document.write("<br>");
}
```

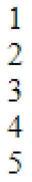
Result:



```
1
2
3
4
5
6
7
8
9
```

And then we run this code:

```
for (let i = 1; i < 6; i++) {
   document.write(i);
   document.write("<br>");
}
```



```
1
2
3
4
5
```

The second information the for loop requires us to give is the limit.

We set this in the **i < 6** part.

The **let i = 1; i < 6;** part of the for loop tells this to the computer:

> *Start a for loop that begins with index **1** and run the loop as long as index **< 6**.*

As you can see, the first line of the for loop also has the **i++** part.

This part tells the **for** loop about the incrementing of the index.

The pace described in **i++** means that index should increase its number **+ 1** in every repetition.

The **let i = 1;** part tells the for loop should start with index 1.

The **i < 6;** part tells the for loop should run for as long as **i < 6**.

If value of **i** is **1**, the loop keeps running. If it's **2**, **3**, **4**, and **5** it will keep running. But as soon as the number is **6**, the for loop stops. Because **6** is not **< 6**.

And finally, the **i++** part tells the loop to increment the index with + 1 in every repetition.

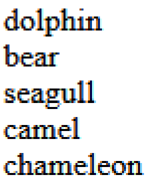
So, on the first time running value of **i** is **1**. On the second time, it's **2**. On the third time, is **3**. On the fourth time is four, and on the fifth time it's **5**.

Calling Elements of an Array

Let's run this code:

```
let animal_1 = 'dolphin';
let animal_2 = 'bear';
let animal_3 = 'seagull';
let animal_4 = 'camel';
let animal_5 = 'chameleon';

document.write(animal_1 + '<br>');
document.write(animal_2 + '<br>');
document.write(animal_3 + '<br>');
document.write(animal_4 + '<br>');
document.write(animal_5 + '<br>');
```

Result:



```
dolphin
bear
seagull
camel
chameleon
```

We have 5 different variables that are of similar context. They all have the name of an animal as a value.

To make our coding more flexible, instead of making 5 different variables, we could use an **array**.

Which one is simpler?

Without an array:

```
let animal_1 = 'dolphin';
let animal_2 = 'bear';
let animal_3 = 'seagull';
let animal_4 = 'camel';
let animal_5 = 'chameleon';
```

Or with an array:

```
let animal = ['dolphin', 'bear', 'seagull', 'camel', 'chameleon'];
```

This is how an array is built:

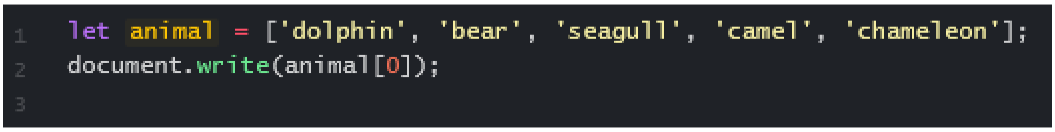
```
let arrayName = ['element 1', 'element 2', 'element 3'];
```

And we can keep adding elements.

Let’s call one of the values inside our previous array. Run this code:

```
let animal = ['dolphin', 'bear', 'seagull', 'camel', 'chameleon'];

document.write(animal[0]);
```

Should look like this:



```
let animal = ['dolphin', 'bear', 'seagull', 'camel', 'chameleon'];
document.write(animal[0]);
```

Result:

```
dolphin
```

As you can see, it works! But how does it works?

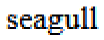
When we create an array, it assigns an **index** to each of its values. The index starts with **0** and gets incremented 1 by 1.

The first element will have an **index** of **0**. The second element will have an **index** of **1**. The third element, will have **index** of **2** and so on.

Let’s run this code:

```
let animal = ['dolphin', 'bear', 'seagull', 'camel', 'chameleon'];

document.write(animal[2]);
```

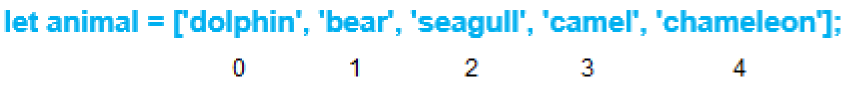
Result:



```
seagull
```

We see “seagull” as the result. Why? Because its index is 2.

Here are the indexes of each element:



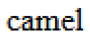
```
let animal = ['dolphin', 'bear', 'seagull', 'camel', 'chameleon'];
                 0          1        2         3          4
```

So, what will be the result of this code?

let animal = ['dolphin', 'bear', 'seagull', 'camel', 'chameleon'];

document.write(animal[3]);

Result:



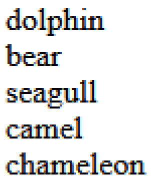
```
camel
```

Let’s see now all the results, just like in our first code of this session:

let animal = ['dolphin', 'bear', 'seagull', 'camel', 'chameleon'];

```
document.write(animal[0] + '<br>');
document.write(animal[1] + '<br>');
document.write(animal[2] + '<br>');
document.write(animal[3] + '<br>');
document.write(animal[4] + '<br>');
```

Result:



```
dolphin
bear
seagull
camel
chameleon
```

Calling Properties of an Object

Let’s see this code:

```
let petName = "Nebuchadnezzar III";
let petSpecies = "cat";
let eats = "cat food";
let guardianName = "William";
let hoursOfSleep = "18 hours per day";
```

We notice these variables **petName**, **petSpecies**, **eats**, **guardianName**, **hoursOfSleep** are related. They have information about a pet.

When we have different variables that are related, it’s much better to use an **object**.

Let’s see how it would work in code lines:

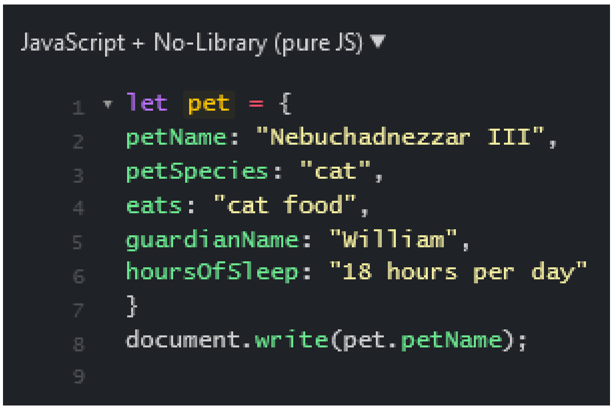
```
let pet = {
petName: "Nebuchadnezzar III";
petSpecies: "cat";
eats: "cat food";
guardianName: "William";
hoursOfSleep: "18 hours per day";
}
```

In the example above, **pet** is called a Javascript **Object**. The codes **petName**, **petSpecies**, **eats**, **guardianName**, **hoursOfSleep** are called **Properties** of the object.

Now let’s run:

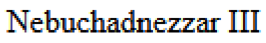
```
let pet = {
petName: "Nebuchadnezzar III",
petSpecies: "cat",
eats: "cat food",
guardianName: "William",
hoursOfSleep: "18 hours per day"
}

document.write(pet.petName);
```

Should look like this:



```
let pet = {
petName: "Nebuchadnezzar III",
petSpecies: "cat",
eats: "cat food",
guardianName: "William",
hoursOfSleep: "18 hours per day"
}
document.write(pet.petName);
```

We see the name of the pet on the screen:

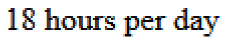


```
Nebuchadnezzar III
```

Now let’s see the property **hoursOfSleep** of **pet**. Keep most of the code above on jsfiddle.net. Replace the last line with this one:

```
document.write(pet.hoursOfSleep);
```

Result:

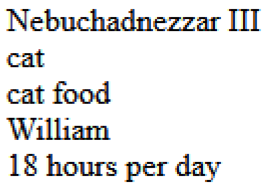


```
18 hours per day
```

Now let’s see all properties of **pet**. Let’s type and run this code:

```
let pet = {
petName: "Nebuchadnezzar III",
petSpecies: "cat",
eats: "cat food",
guardianName: "William",
hoursOfSleep: "18 hours per day"
}

document.write(pet.petName + "<br>");
document.write(pet.petSpecies + "<br>");
document.write(pet.eats + "<br>");
document.write(pet.guardianName + "<br>");
document.write(pet.hoursOfSleep + "<br>");
```

Result:



```
Nebuchadnezzar III
cat
cat food
William
18 hours per day
```

Tiny Project

To put many of the concepts we studied together, now we are going to do a small project.

We want to make a page where we can change the background color with the click of a button.

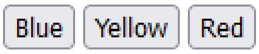
I) Make the buttons in HTML

Go to the HTML field of jsfiddle.net and add 3 buttons, with a color name on each.

The code should look like this:

```
<button>Blue</button>
<button>Yellow</button>
<button>Red</button>
```

Result:



If we click on it nothing happens. Why? Because we need to “tell” the computer to change the colors once we click the button. We do this through lines of code in Javascript.

II) Make the button trigger a Javascript code once we click them

We do this by adding **onclick="JavascriptCodes"** to each button in the HTML field in the place we see below. Let’s type this code in the HTML field and run it:

```
<button onclick="#">Blue</button>
<button onclick="#">Yellow</button>
<button onclick="#">Red</button>
```

Now let’s add a Javascript code to test if the buttons are working.

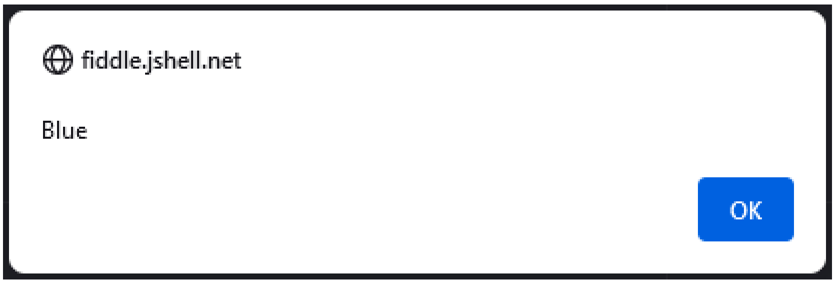
In this quick test, we are going to use an alert window.

On the HTML field, type this code and run it:

```
<button onclick="alert('Blue');">Blue</button>
<button onclick="alert('Yellow');">Yellow</button>
<button onclick="alert('Red');">Red</button>
```

By clicking on each of the buttons we see an alert window.

Result of what happens after we click the Blue button:


fiddle.jshell.net

Blue

Now we know that our buttons are triggering a Javascript command.

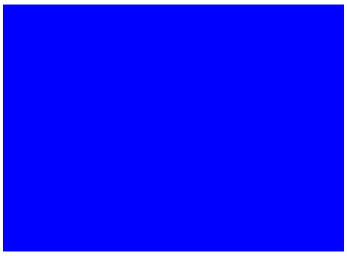
However, we don’t want to see alert windows. We want the color of the background to be changed once we click the button.

III) Add a Javascript code to change the background color

Now in the Javascript field, let’s type and run this code:

```
document.body.style.backgroundColor = 'blue';
```

Result:



Perfect! Now we have a code that can define the background color.

However, we only want the background color to change to blue once we click the “Blue” button.

For that, we need to create a function.

IV) Name a function switchToBlue() and add it to the button

We will call the function **switchToBlue()**.

First, we add it to the “Blue” button.

In the HTML field we write this code (let’s do the Blue button only for now):

```
<button onclick="switchToBlue();">Blue</button>
```

IV) Create the switchToBlue() function

Now let’s go to the Javascript field and type this:

```
function switchToBlue() {
}
```

Remember we already have a code that changes the background color to blue? This one:

```
document.body.style.backgroundColor = 'blue';
```

All we have to do here is to place this code inside the **switchToBlue()** function.

Let’s write it in the Javascript field:

```
function switchToBlue() {
   document.body.style.backgroundColor = 'blue';
}
```

IV) Test the code

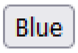
After the last step, the HTML field should have this code:

```
<button onclick=" switchToBlue();">Blue</button>
```

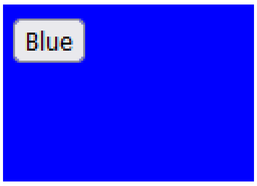
And the Javascript field should have this one:

```
function switchToBlue() {
   document.body.style.backgroundColor = 'blue';
}
```

If we click to Run the code, we can see this:



If we click the Blue button, the background color is changing:



Perfect! Our function works.

Now what we have to do is to add the other two buttons, “Yellow” and “Red”.

VI) Create the switchToYellow() function and the Yellow button

We keep the previous HTML and Javascript codes and add newer lines.

The HTML field should look like this:

```
<button onclick=" switchToBlue();">Blue</button>
<button onclick=" switchToYellow();">Yellow</button>
```

And the Javascript field should look like this:

```
function switchToBlue() {
   document.body.style.backgroundColor = 'blue';
}
function switchToYellow() {
   document.body.style.backgroundColor = 'yellow';
}
```

We click on Run and we can see that the “Blue” and the “Yellow” buttons are working.

Now we add the “Red” button.

VII) Create the switchToRed() function and the Red button

Again keeping the previous HTML and Javascript lines, we add newer for the Red button and switchToRed() function.

The HTML field should look like this:

```
<button onclick="switchToBlue();">Blue</button>
<button onclick=" switchToYellow();">Yellow</button>
<button onclick=" switchToRed();">Red</button>
```

And the Javascript field should look like this:

```
function switchToBlue() {
   document.body.style.backgroundColor = 'blue';
}
function switchToYellow() {
   document.body.style.backgroundColor = 'yellow';
}
function switchToRed() {
   document.body.style.backgroundColor = 'red';
}
```

Click in Run. We test all the buttons and see that everything is working well.

Our tiny project is completed!

Better Javascript Lines For The Tiny Project

We have built 3 functions in the last tiny projects. These 3 functions are very similar.

All they do is switch the background color to a color we defined.

Whenever we have a few functions that perform the same task is a good practice to build only One function instead. And then use parameters to make it perform the way we want.

I) Add a colorSwitchTo() function to the buttons.

In the HTML field, add this:

```
<button onclick="switchTo();">Blue</button>
<button onclick="switchTo();">Yellow</button>
<button onclick="switchTo();">Red</button>
```

Once clicked, the buttons will trigger the **switchTo()** function once we create it.

However, they could not differentiate the actions yet. They all will trigger the same action, and we do not want it to happen.

We want the Blue button to change the background color to blue once we click it.

In the same way, we want the Yellow button to change the background color to yellow once we click it.

And finally, we want the Red button to change the background color to red once we click it.

How to do that if we are using the same **switchTo()** function?

By using parameters.

II) Add a parameter to the switchTo() function of each button.

In the HTML, write this:

```
<button onclick="switchTo('blue');">Blue</button>
<button onclick="switchTo('yellow');">Yellow</button>
<button onclick="switchTo('red');">Red</button>
```

Very important, you notice we did add the **'blue'**, **'yellow'**, and **'red'** parameters as **strings**. We are using the simple quotes **' '** in the parameters.

Now that our buttons have a function with a different parameter for each, we create the function.

III) Create the switchTo() function

Like in the previous session, let’s start with the “Blue” button.

In the HTML field we have this:

```
<button onclick="switchTo('blue');">Blue</button>
```

In the Javascript we declare the **switchTo()** function:

```
function switchTo() {
}
```

Before adding the lines of command inside the function, we need to take a step we did not take in the last session: we need to add a parameter to the **switchTo()** function.

We will call it **backgroundColor**.

Our **switchTo()** function will look like this in the Javascript field:

```
function switchTo(backgroundColor) {
}
```

Why we are adding this parameter? So the **switchTo()** function can receive information on what background color we want it to display. It will make more sense in the next steps.

IV) Add the functionality to the switchTo(backgroundColor) function.

So, we have our **switchTo(backgroundColor)** function ready:

```
function switchTo(backgroundColor) {
}
```

And we have this line of command that changes the background color to **blue** from previous examples:

```
document.body.style.backgroundColor = 'blue';
```

If we add this line of command to the **switchTo(backgroundColor)** function, it will look like this:

```
function switchTo(backgroundColor) {
   document.body.style.backgroundColor = 'blue';
}
```

If you run it, you will see that it works, but it’s wrong.

Why?

Because it’s not dynamic. It will work for switching the background color to **Blue**, but won’t work for **Yellow** nor **Red**.

So, we need to make the **document.body.style.backgroundColor = 'blue';** line dynamic.

We do that by using the parameter of the function.

V) Using the parameter to make the function switchTo(backgroundColor) dynamic.

This was our previous, static code:

```
function switchTo(backgroundColor) {
   document.body.style.backgroundColor = 'blue';
}
```

It will work to change the background color to blue. But it won’t work to change it to other colors if we need it.

We only want the **document.body.style.backgroundColor = 'blue';** line to be executed once we click the **Blue** button.

If we click the **Yellow** button we don’t want it to be executed.

To do so, we can use the keyword **if**.

If we click the blue button: we want the function to switch the background color to blue.

Else if we click the yellow button: we want the function to switch the background color to yellow.

Let’s use the **if** keyword with the **Blue** button now and run it in the Javascript field:

```
function switchTo(backgroundColor) {
   if (backgroundColor === 'blue') {
      document.body.style.backgroundColor = 'blue';
   }
}
```

HTML field:

```
<button onclick="switchTo('blue');">Blue</button>
```

We see that it works.

The next step is adding the **Yellow** button.

VI) We add the Yellow Button and its parameter using else if

In the HTML field:

```
<button onclick="switchTo('blue');">Blue</button>
<button onclick="switchTo('yellow');">Yellow</button>
```

In the Javascript field:

```
function switchTo(backgroundColor) {
   if (backgroundColor === 'blue') {
      document.body.style.backgroundColor = 'blue';
   }
   else if (backgroundColor === 'yellow') {
      document.body.style.backgroundColor = 'yellow';
   }
}
```

And finally, we add the Red button.

VII) We add the Red Button and its parameter using else if again

Our HTML code will be like this:

```
<button onclick="switchTo('blue');">Blue</button>
<button onclick="switchTo('yellow');">Yellow</button>
<button onclick="switchTo('red');">Red</button>
```

The Javascript code will look like this:

```
function switchTo(backgroundColor) {
   if (backgroundColor === 'blue') {
      document.body.style.backgroundColor = 'blue';
   }
   else if (backgroundColor === 'yellow') {
      document.body.style.backgroundColor = 'yellow';
   }
   else if (backgroundColor === 'red') {
      document.body.style.backgroundColor = 'red';
   }
}
```

Where to Go From Now

Congratulations on arriving here. Now you understand Javascript concept better than 95% of the people who try to learn it the painful way through tutorials and long videos, as a beginner.

There is a series of steps to take so you can grow in knowledge:

Master the Fundamentals

Read again the sessions of this book that you may not understand so well yet. Always typing and running the codes we describe.

Ideally, you want to understand the basics of every subject we have seen.

For example, you don't need to build complex functions yet, but understand how a function works, how it is triggered, how function parameters work, etc.

Start Building Your Own (Simple) Projects

At the same time that you re-read the fundamentals, start building small projects. This is by far the best way to grow in knowledge.

You don't need to build complex apps for now. You can build something like a **sum calculator**.

Then after you finish, you can make a calculator that makes **sums and subtractions**.

You can build a page with buttons that change the **font size** and **font family** of a text and so on. Eventually, you can even begin building simple games.

Building projects is challenging and fun.

And with them, you will develop another skill that is very important for any developer: **researching**.

Learn to Research Codes and Concepts

We would like to build a Javascript code that gives us a random number from 50 to 100.

What do we need? We need a built-in function that gives us a random number.

We go to a search engine and type **generate random number with Javascript** and click on search.

We find that we can generate a random number between 50 and 100 by using this line of code:

```
Math.floor(Math.random() * (100 - 50 + 1) ) + 50;
```

Now we attribute its value to a variable and call it with **document.write()**.

Go to jsfiddle.net and run this code in the Javascript field:

```
let randomNumber = Math.floor(Math.random() * (100 - 50 + 1) ) + 50;
document.write(randomNumber);
```

We see that the lines of code work to give us what we wanted: to generate a random number between 50 and 100 once we click Run.

Once building your projects, whenever you need to perform a task you don't know yet, researching will be your best friend.

This is one of the most important skills to develop, both as a beginner and as an experienced developer.

Once you've Built a Few Simple Projects, Time to go Deeper in Concepts

You learned how a loop works and why we use them. You know how to use the **for loop** very well.

Ok, now is time to learn more loops, such as the **while** loop and the **for each** loop.

Similarly, you've learned how to use the **document.write()** function. Now is time to learn more about manipulating web pages using Javascript by learning concepts such as **DOM**.

Good news is now your learning will flow much better because you've learned the fundamentals. It's much easier to study new concepts once you understand Javascript.