



## JavaScript Knowledge Transformation from JAVA

# Today's Schedule

- JavaScript Intro
- Variables
- Operators

# What is JavaScript?

- *JavaScript* was initially created to “make web pages alive”.
- The programs in this language are called *scripts*. They can be written right in a web page’s HTML and run automatically as the page loads.
- Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.
- In this aspect, JavaScript is very different from another language called [Java](#).



# JavaScript Run Environments

- Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the **JavaScript engine**.
- The browser has an embedded engine sometimes called a “**JavaScript virtual machine**”.
- Different engines have different “codenames”. For example:
  - V8 – in Chrome, Opera and Edge.
  - SpiderMonkey – in Firefox , etc.
- We have installed Node.js so that we can run our JS codes directly in the code editor – Visual Studio Code VSC

# The Role of JavaScript



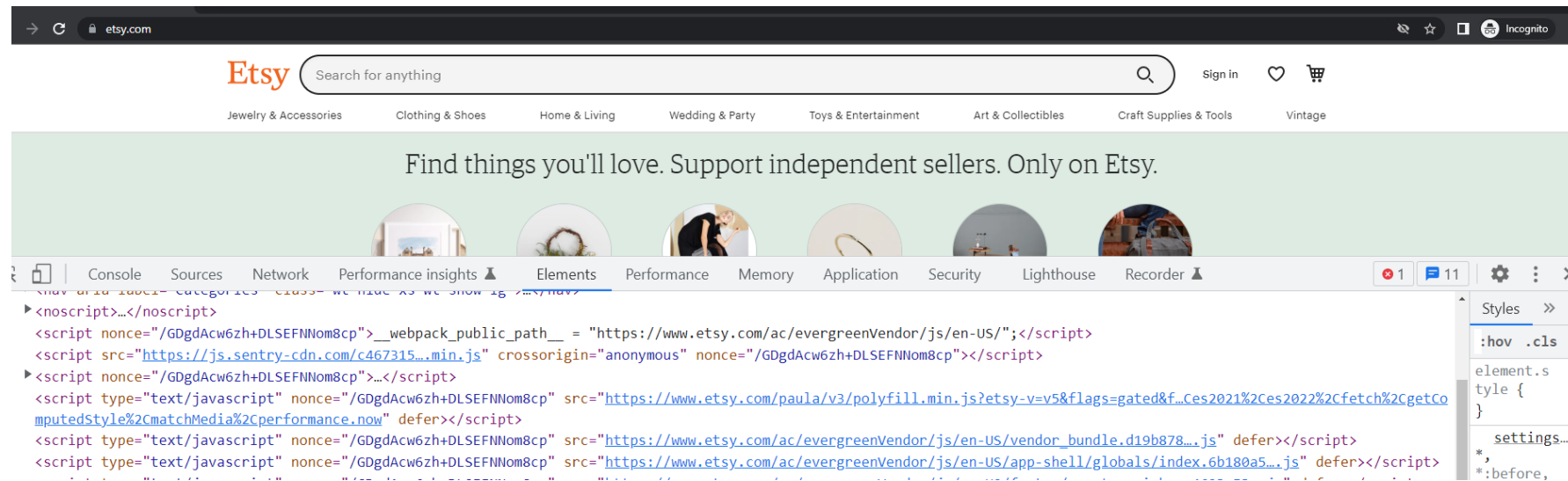
- JavaScript is one of the three core technologies web development.
- For instance, in-browser JavaScript is able to:
  - React to user actions, run on mouse clicks, pointer movements, key presses.
  - Send requests over the network to remote servers, download and upload files (so-called [AJAX](#) technologies).
  - Get and set cookies, ask questions to the visitor, show messages.

# Where do we put JS code?

- The `<script>` tag contains JavaScript code which is automatically executed when the browser processes the tag.
- If we have a lot of JavaScript code, we can put it into a separate file.
- Script files are attached to HTML with the `src` attribute:
  - `<script src="/path/to/script.js"></script>`

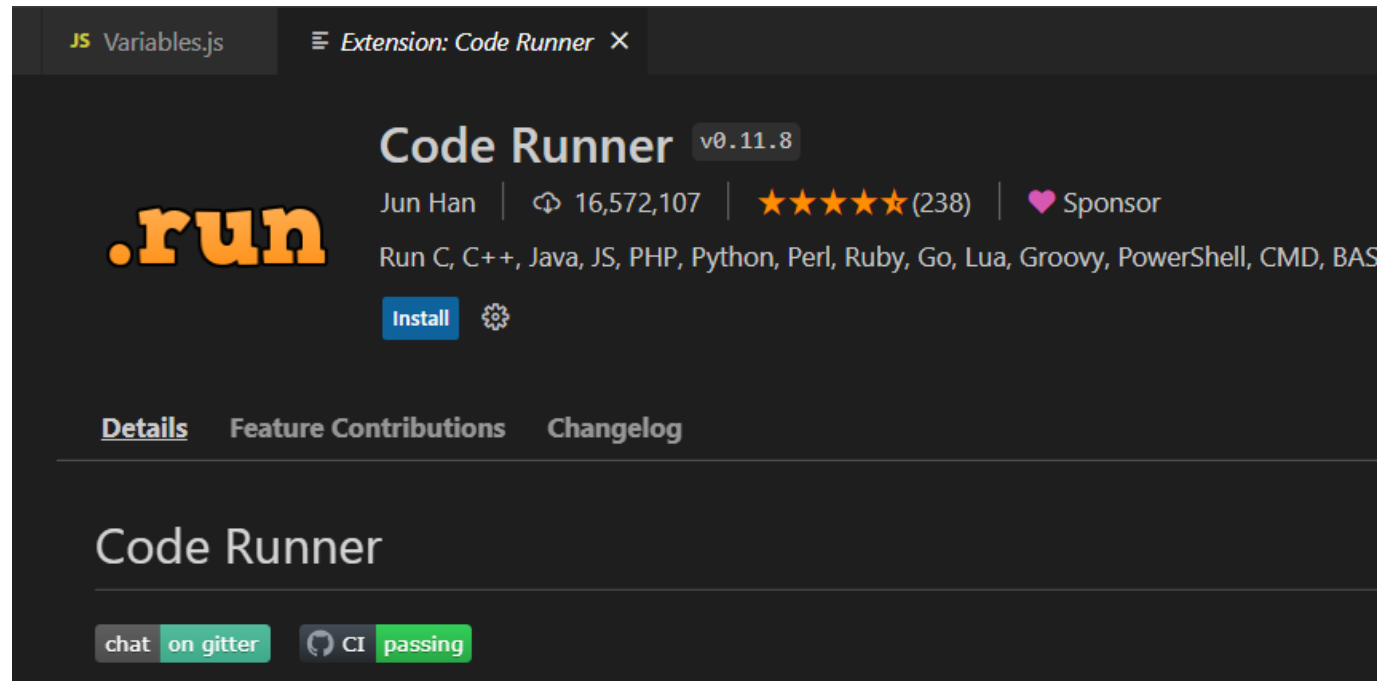
# How do we see JS code on a Browser Page?

- Let's learn how to open Developer console of Google Chrome
- Press F12 or, if you're on Mac, then Cmd+Opt+J.



# VSC Extensions Required

- Code Runner



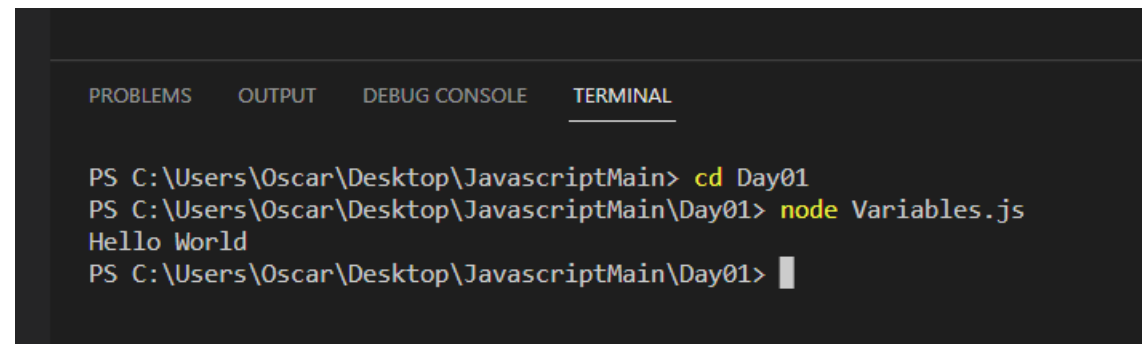
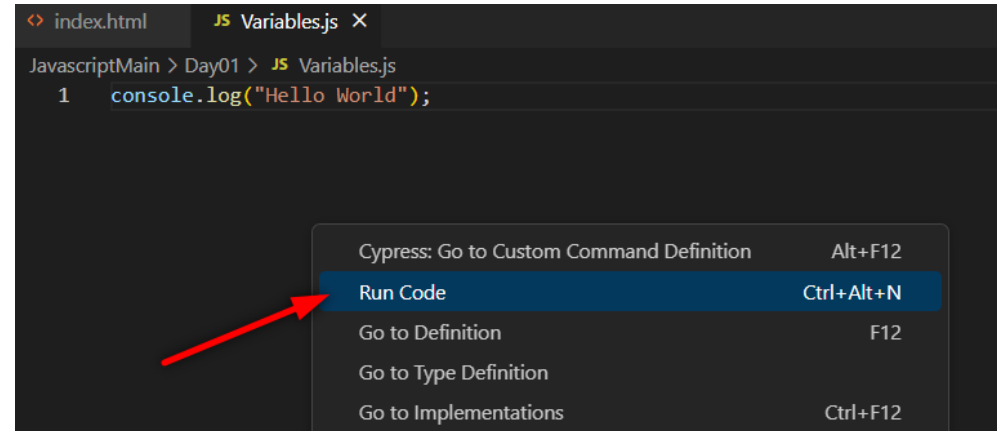


# Hello World!

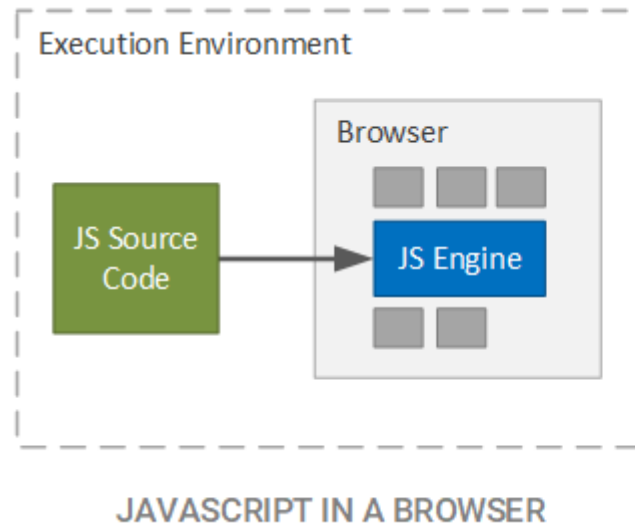
- Let's write our first JavaScript code:
  - Assuming we have installed VSC, Node.js, and extensions (liveServer, CodeRunner)
  - Create a folder named JavaScriptProgramming on Desktop
  - Open folder in VSC
  - Create folder named Day01
  - Create Variables.js file

# Hello World!

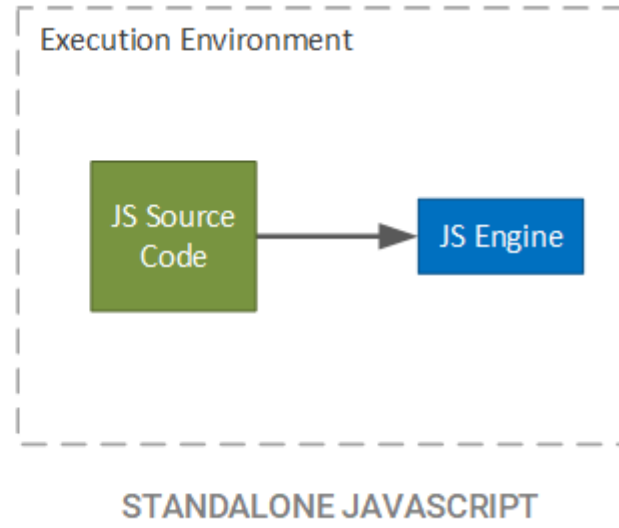
- Run your codes with Node.js engine in your computer with Code Runner extension
- Or running your code from Terminal



# Javascript Virtual Machine Browser vs Node.js



**HTML file needed: Runs on browser JS Virtual Machine**



**No HTML file: Runs on Node.js Machine**

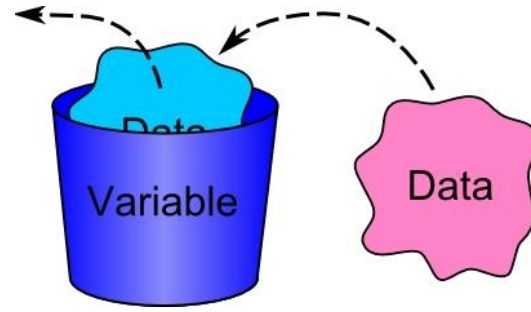
# Code structure

- Statements : are syntax constructs and commands that perform actions.
  - `console.log('Hello World');`
- A semicolon may be omitted in most cases when a line break exists.
- **In most cases, a newline implies a semicolon. But “in most cases” does not mean “always”!**
- But there are situations where JavaScript “fails” to assume a semicolon where it is really needed.

# Today's Schedule

- JavaScript Intro
- Variables
- Operators

# Variables



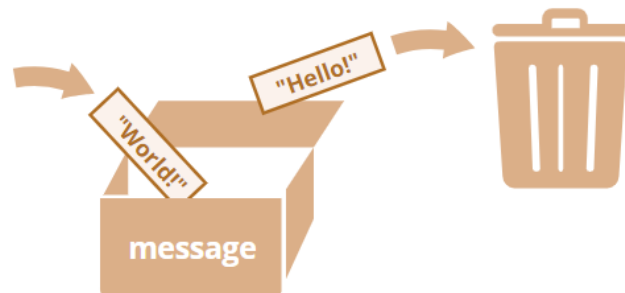
- A variable is a “named storage” for data. We can use variables to store data.
- To create a variable in JavaScript, use the `let` keyword.
- The statement below creates (in other words: declares) a variable with the name “message”:
  - `let message;`
  - Now, we can put some data into it by using the assignment operator `=`
  - `message = 'Hello'; // store the string 'Hello' in the variable named message`

# Variables

- We can easily grasp the concept of a “variable” if we imagine it as a “box” for data, with a uniquely-named sticker on it.
- For instance, the variable message can be imagined as a box labeled "message" with the value "Hello!" in it:



- We can put any value in the box. We can also change it as many times as we want. When the value is changed, the old data is removed from the variable:



# Variable naming

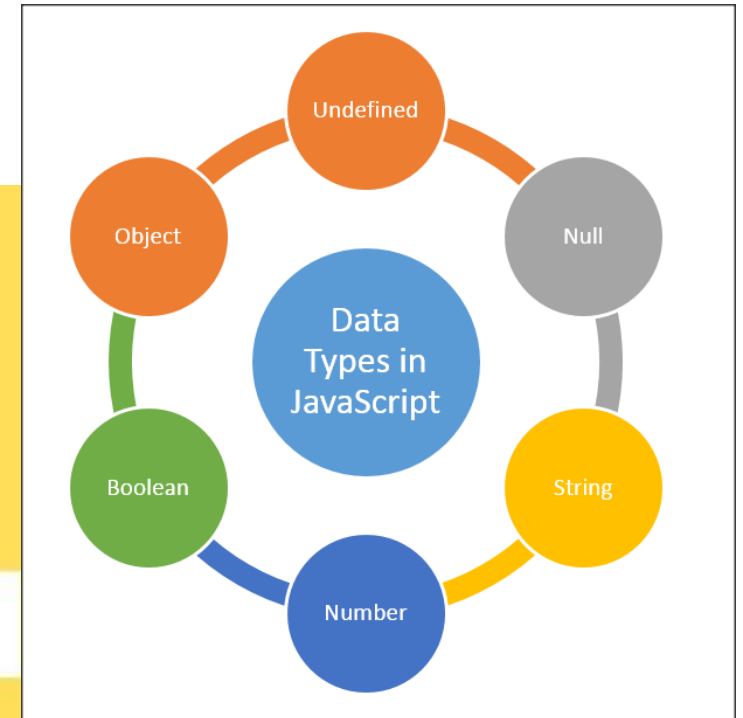
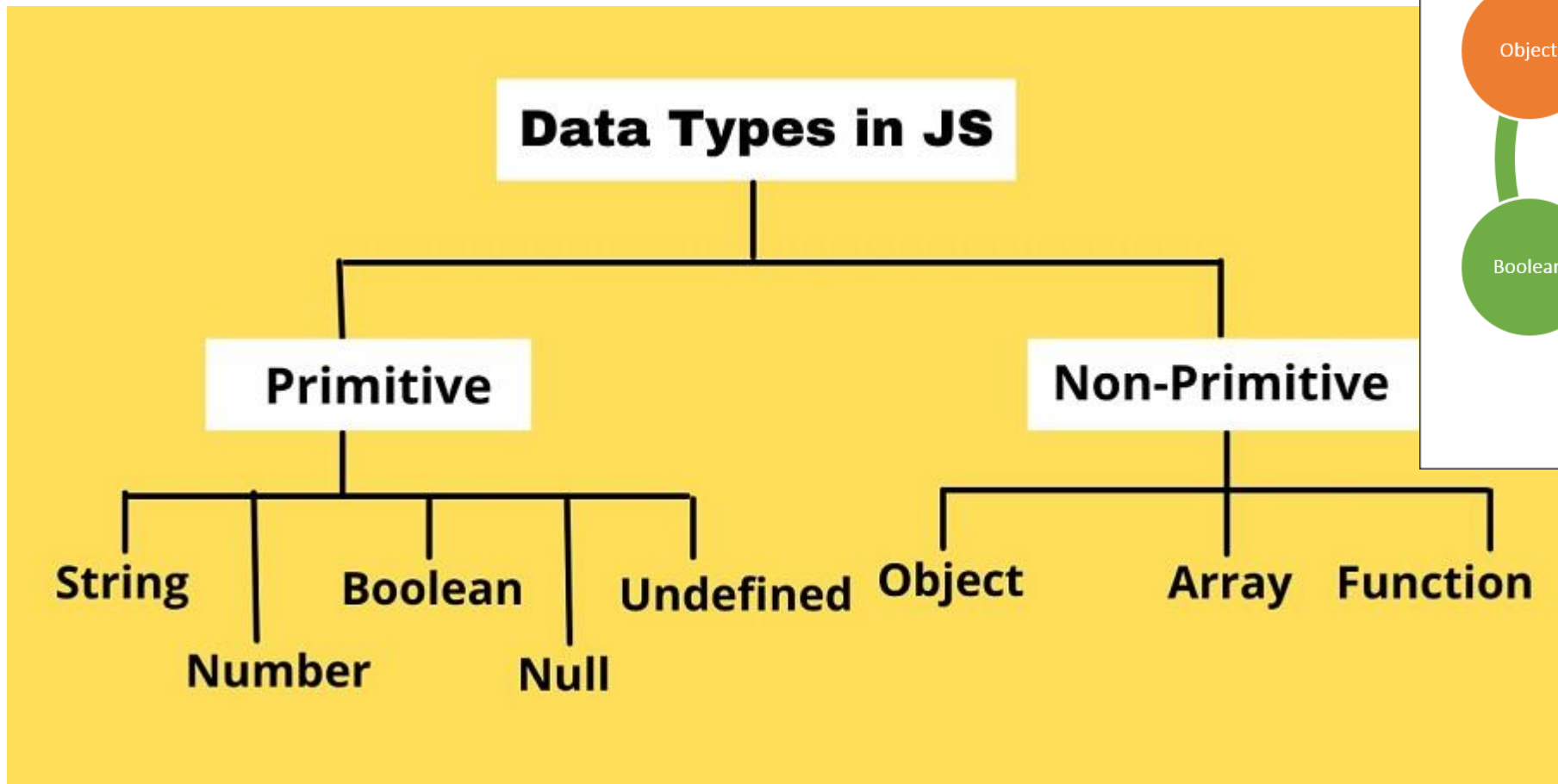
- There are two limitations on variable names in JavaScript:
  - The name must contain only letters, digits, or the symbols \$ and \_.
  - The first character must not be a digit.
- Examples of valid names:
  - `let userName;`
  - `let test123;`
- When the name contains multiple words, **camelCase** is commonly used. That is: words go one after another, each word except first starting with a capital letter:
  - `myVeryLongName.`



# Variable naming

- Case matters
  - Variables named apple and APPLE are two different variables.
- Reserved names
  - There is a list of reserved words, which cannot be used as variable names because they are used by the language itself.
  - For example: let, class, return, and function are reserved.
- Name things right
  - A variable name should have a clean, obvious meaning, describing the data that it stores.

# Data Types



# Strings

- In JavaScript, there are 3 types of quotes.
  - Double quotes: "Hello".
  - Single quotes: 'Hello'.
  - Backticks: `Hello`.
- Double and single quotes are “simple” quotes. There’s practically no difference between them in JavaScript.
- Backticks are “extended functionality” quotes. They allow us to embed variables and expressions into a string by wrapping them in `${...}`, for example:

# Basic operators, maths

- The following math operations are supported:
  - Addition +,
  - Subtraction -,
  - Multiplication \*,
  - Division /,
  - Remainder %,
  - Exponentiation \*\*.

# Increment/decrement

- Increasing or decreasing a number by one is among the most common numerical operations. So, there are special operators for it:
  - Increment ++ increases a variable by 1
  - Decrement -- decreases a variable by 1
- The operators ++ and -- can be placed either before or after a variable.
  - When the operator goes after the variable, it is in “postfix form”: counter++.
  - The “prefix form” is when the operator goes before the variable: ++counter.
- The prefix form returns the new value while the postfix form returns the old value (prior to increment/decrement).

# Comparison Operators

Description	Operator
Equality	==
Inequality	!=
Identity/Strict Equality	===
Non-identity/Strict Inequality	!==
Greater than	>
Greater than or equal	>=
Less than	<
Less than or equal	<=

## Boolean is the result

All comparison operators return a boolean value:

true – means “yes”, “correct” or “the truth”.

false – means “no”, “wrong” or “not the truth”.

# Logical operators

- `||` (OR)
  - The “OR” operator is represented with two vertical line symbols:
- `&&` (AND)
  - The AND operator is represented with two ampersands `&&`
- Precedence of AND `&&` is higher than OR `||`
  - So the code `a && b || c && d` is essentially the same as if the `&&` expressions were in parentheses: `(a && b) || (c && d)`.
- `!` (NOT)
  - The boolean NOT operator is represented with an exclamation sign `!`.

# Conditional branching: if, if else

- Sometimes, we need to perform different actions based on different conditions.

To do that, we can use the if statement

The if statement evaluates a condition. If the condition evaluates to **true**, any statements in the subsequent code block are executed.

```
if (score >= 50) {  
    congratulate();  
}
```

The diagram illustrates the syntax of an if statement with the following labels:

- KEYWORD**: Points to the word `if`.
- CONDITION**: Points to the expression `(score >= 50)`.
- OPENING CURLY BRACE**: Points to the opening curly brace `{`.
- CODE TO EXECUTE IF VALUE IS TRUE**: Points to the statement `congratulate();`.
- CLOSING CURLY BRACE**: Points to the closing curly brace `}`.



# “question mark” operator

- The so-called “conditional” or “question mark” operator lets us do that in a shorter and simpler way.
- The operator is represented by a question mark ?. Sometimes it’s called “ternary”, because the operator has three operands. It is actually the one and only operator in JavaScript which has that many.
- The syntax is:
- `let result = condition ? value1 : value2;`
  - If condition is true, result is value1, if wrong result is value2

# The "switch" statement

- A switch statement can replace multiple if checks.
- It gives a more descriptive way to compare a value with multiple variants.
- The syntax : has one or more case blocks and an optional default.
- It looks like this:

```
switch(x) {  
  case 'value1': // if (x === 'value1')  
    ...  
    [break]  
  
  case 'value2': // if (x === 'value2')  
    ...  
    [break]  
  
  default:  
    ...  
    [break]  
}
```

# Loops: while and for

- We often need to repeat actions.
- The while loop syntax:

```
do {  
    // loop body  
} while (condition);
```

```
while (condition) {  
    // code  
    // so-called "loop body"  
}
```


- The “do...while” loop: The condition check can be moved below the loop body using

- The for loop is more complex, but it's also the most commonly used loop. It looks like this:

```
for (begin; condition; step) {  
    // ... loop body ...  
}
```

# “callback hell” or “pyramid of doom”

```
loadScript('1.js', function(error, script) {  
  if (error) {  
    handleError(error);  
  } else {  
    // ...  
    loadScript('2.js', function(error, script) {  
      if (error) {  
        handleError(error);  
      } else {  
        // ...  
        loadScript('3.js', function(error, script) {  
          if (error) {  
            handleError(error);  
          } else {  
            // ...  
          }  
        });  
      }  
    });  
  }  
});
```



```
// verify all buying options are present
cy.get("button[data-label*='Rationality']")
  .click()
  .then(() => {
    shops.forEach((shop) => {
      cy.get("ul[id='expanded-buy-books-menu-0']>li>a").contains(shop);
    });
  });
```