**What are Conditional Statements?**

In life, we make decisions based on circumstances. Think of an everyday decision as mundane as falling asleep — if we are tired, we go to bed, otherwise, we wake up and start our day.

These if-else decisions can be modeled in code by creating *conditional statements*. A conditional statement checks a specific condition(s) and performs a task based on the condition(s).

In this lesson, we will explore how programs make decisions by evaluating conditions and introduce logic into our code!

We’ll be covering the following concepts:

* if, else if, and else statements
* comparison operators
* logical operators
* truthy vs falsy values
* ternary operators
* switch statement

So *if* you’re ready to learn these concepts go to the next lesson— *else*, read over the concepts, observe the diagram, and prepare yourself for this lesson!

# If Statement

We often perform a task based on a condition. For example, if the weather is nice today, then we will go outside. If the alarm clock rings, then we’ll shut it off. If we’re tired, then we’ll go to sleep.

In programming, we can also perform a task based on a condition using an if statement:

if (true) {

console.log('This message will print!');

}

// Prints: This message will print!

Notice in the example above, we have an if statement. The if statement is composed of:

* The if keyword followed by a set of parentheses () which is followed by a code block, or block statement, indicated by a set of curly braces {}.
* Inside the parentheses (), a condition is provided that evaluates to true or false.
* If the condition evaluates to true, the code inside the curly braces {} runs, or executes.
* If the condition evaluates to false, the block won’t execute.

Let’s make an if statement.

**Instructions**

**1.**

Using the let keyword, declare a variable named sale. Assign the value true to it.

Hint

To create a let variable, follow the syntax below:

let sampleVar = true;

true is not a string, and therefore does NOT need quotes around it.

**2.**

Now create an if statement. Provide the if statement a condition of sale.

Inside the code block of the if statement, console.log() the string 'Time to buy!'.

Hint

An if statement has 3 parts: the if keyword, a condition wrapped in parentheses (), and a code block wrapped in curly braces {}.

if (conditionGoesHere) {

// Code to execute if the provided condition evaluates to true

}

**3.**

Notice that the code inside the if statement ran, since 'Time to buy!' was logged to the console.

Below the sale variable declaration, but before the if statement, reassign sale to false. Run your code and observe what happens, we’ll be changing this behavior in the next exercise.

Hint

There shouldn’t be anything that is printed to console after you reassign sale to false.

let sale = true;

sale = false;

if(sale){

  console.log('Time to buy!')

}

# If...Else Statements

In the previous exercise, we used an if statement that checked a condition to decide whether or not to run a block of code. In many cases, we’ll have code we want to run if our condition evaluates to false.

If we wanted to add some default behavior to the if statement, we can add an else statement to run a block of code when the condition evaluates to false. Take a look at the inclusion of an else statement:

if (false) {

console.log('The code in this block will not run.');

} else {

console.log('But the code in this block will!');

}

// Prints: But the code in this block will!

An else statement must be paired with an if statement, and together they are referred to as an if...else statement.

In the example above, the else statement:

* Uses the else keyword following the code block of an if statement.
* Has a code block that is wrapped by a set of curly braces {}.
* The code inside the else statement code block will execute when the if statement’s condition evaluates to false.

if...else statements allow us to automate solutions to yes-or-no questions, also known as binary decisions.

**Instructions**

**1.**

Add an else statement to the existing if statement. Inside the code block of the else statement, console.log() the string 'Time to wait for a sale.'

Hint

An else statement does not take another condition. Therefore, it will NOT have a set of parentheses. The syntax for adding an if...else statement looks like:

if (conditionGoesHere) {

// Code for if statement

} else {

// Code for else statement

}

let sale = true;

sale = false;

if(sale) {

  console.log('Time to buy!');

}

else{

    console.log('Time to wait for a sale.');

}

# Comparison Operators

When writing conditional statements, sometimes we need to use different types of operators to compare values. These operators are called comparison operators.

Here is a list of some handy comparison operators and their syntax:

* Less than: <
* Greater than: >
* Less than or equal to: <=
* Greater than or equal to: >=
* Is equal to: ===
* Is not equal to: !==

Comparison operators compare the value on the left with the value on the right. For instance:

10 < 12 // Evaluates to true

It can be helpful to think of comparison statements as questions. When the answer is “yes”, the statement evaluates to true, and when the answer is “no”, the statement evaluates to false. The code above would be asking: is 10 less than 12? Yes! So 10 < 12 evaluates to true.

We can also use comparison operators on different data types like strings:

'apples' === 'oranges' // false

In the example above, we’re using the identity operator (===) to check if the string 'apples' is the same as the string 'oranges'. Since the two strings are not the same, the comparison statement evaluates to false.

All comparison statements evaluate to either true or false and are made up of:

* Two values that will be compared.
* An operator that separates the values and compares them accordingly (>, <, <=,>=,===,!==).

Let’s practice using these comparison operators!

**Instructions**

**1.**

Using let, create a variable named hungerLevel and set it equal to 7.

**2.**

Write an if...else statement using a comparison operator. The condition should check if hungerLevel is greater than 7. If so, the conditional statement should log, 'Time to eat!'. Otherwise, it should log 'We can eat later!'.

After you press run, play around with the condition by tweaking the comparison of hungerLevel by using different operators such as <=,>=,>, and <.

Hint

The syntax for an if...else statement is:

if (condition) {

// Execute this code

} else {

// Execute this code

}

let sale = true;

sale = false;

if(sale) {

  console.log('Time to buy!');

}

else{

    console.log('Time to wait for a sale.');

}

# Comparison Operators

When writing conditional statements, sometimes we need to use different types of operators to compare values. These operators are called comparison operators.

Here is a list of some handy comparison operators and their syntax:

* Less than: <
* Greater than: >
* Less than or equal to: <=
* Greater than or equal to: >=
* Is equal to: ===
* Is not equal to: !==

Comparison operators compare the value on the left with the value on the right. For instance:

10 < 12 // Evaluates to true

It can be helpful to think of comparison statements as questions. When the answer is “yes”, the statement evaluates to true, and when the answer is “no”, the statement evaluates to false. The code above would be asking: is 10 less than 12? Yes! So 10 < 12 evaluates to true.

We can also use comparison operators on different data types like strings:

'apples' === 'oranges' // false

In the example above, we’re using the identity operator (===) to check if the string 'apples' is the same as the string 'oranges'. Since the two strings are not the same, the comparison statement evaluates to false.

All comparison statements evaluate to either true or false and are made up of:

* Two values that will be compared.
* An operator that separates the values and compares them accordingly (>, <, <=,>=,===,!==).

Let’s practice using these comparison operators!

**Instructions**

**1.**

Using let, create a variable named hungerLevel and set it equal to 7.

**2.**

Write an if...else statement using a comparison operator. The condition should check if hungerLevel is greater than 7. If so, the conditional statement should log, 'Time to eat!'. Otherwise, it should log 'We can eat later!'.

After you press run, play around with the condition by tweaking the comparison of hungerLevel by using different operators such as <=,>=,>, and <.

Hint

The syntax for an if...else statement is:

if (condition) {

// Execute this code

} else {

// Execute this code

}

let hungerLevel = 7;

if(hungerLevel>7){

  console.log('Time to eat!');

}

else{

  console.log('We can eat later!');

}

# Logical Operators

Working with conditionals means that we will be using booleans, true or false values. In JavaScript, there are operators that work with boolean values known as logical operators. We can use logical operators to add more sophisticated logic to our conditionals. There are three logical operators:

* the and operator (&&)
* the or operator (||)
* the not operator, otherwise known as the bang operator (!)

When we use the && operator, we are checking that two things are true:

if (stopLight === 'green' && pedestrians === 0) {

console.log('Go!');

} else {

console.log('Stop');

}

When using the && operator, both conditions must evaluate to true for the entire condition to evaluate to true and execute. Otherwise, if either condition is false, the && condition will evaluate to false and the else block will execute.

If we only care about either condition being true, we can use the || operator:

if (day === 'Saturday' || day === 'Sunday') {

console.log('Enjoy the weekend!');

} else {

console.log('Do some work.');

}

When using the || operator, only one of the conditions must evaluate to true for the overall statement to evaluate to true. In the code example above, if either day === 'Saturday' or day === 'Sunday' evaluates to true the if‘s condition will evaluate to true and its code block will execute. If the first condition in an || statement evaluates to true, the second condition won’t even be checked. Only if day === 'Saturday' evaluates to false will day === 'Sunday' be evaluated. The code in the else statement above will execute only if both comparisons evaluate to false.

The ! not operator reverses, or negates, the value of a boolean:

let excited = true;

console.log(!excited); // Prints false

let sleepy = false;

console.log(!sleepy); // Prints true

Essentially, the ! operator will either take a true value and pass back false, or it will take a false value and pass back true.

Logical operators are often used in conditional statements to add another layer of logic to our code.

**Instructions**

**1.**

In **main.js** there are two variables mood and tirednessLevel.

Let’s create an if...else statement that checks if mood is 'sleepy' and tirednessLevel is greater than 8.

If both conditions evaluate to true, then console.log() the string 'time to sleep'. Otherwise, we should console.log() 'not bed time yet'.

After you press “Run”, play around with the || operator and the ! operator! What happens if you negate the value of the entire statement with ! and switch to || instead of &&?

Hint

To check if a value is the same as another value we use the === operator. For example:

'apples' === 'oranges' // false

To check if a a value is greater than another value we use the > operator. For example:

10 > 5 // true

To use a logical operator like && in an if statement:

if('apples' === 'oranges' && 10 > 5) {

console.log('Both statements are true!')

} else {

console.log('At least one statement is false!')

}

You can even add optional parentheses to make it clearer for other developers:

if(('apples' === 'oranges') && (10 > 5)) {

console.log('Both statements are true!')

} else {

console.log('At least one statement is false!')

}

let mood = 'sleepy';

let tirednessLevel = 6;

if(mood==='sleepy' && tirednessLevel > 8) {

  console.log('time to sleep');

}

else{

  console.log('not bed time yet');

}

# Truthy and Falsy

Let’s consider how non-boolean data types, like strings or numbers, are evaluated when checked inside a condition.

Sometimes, you’ll want to check if a variable exists and you won’t necessarily want it to equal a specific value — you’ll only check to see if the variable has been assigned a value.

Here’s an example:

let myVariable = 'I Exist!';

if (myVariable) {

console.log(myVariable)

} else {

console.log('The variable does not exist.')

}

The code block in the if statement will run because myVariable has a truthy value; even though the value of myVariable is not explicitly the value true, when used in a boolean or conditional context, it evaluates to true because it has been assigned a non-falsy value.

So which values are falsy— or evaluate to false when checked as a condition? The list of falsy values includes:

* 0
* Empty strings like "" or ''
* null which represent when there is no value at all
* undefined which represent when a declared variable lacks a value
* NaN, or Not a Number

Here’s an example with numbers:

let numberOfApples = 0;

if (numberOfApples){

console.log('Let us eat apples!');

} else {

console.log('No apples left!');

}

// Prints 'No apples left!'

The condition evaluates to false because the value of the numberOfApples is 0. Since 0 is a falsy value, the code block in the else statement will run.

**Instructions**

**1.**

Change the value of wordCount so that it is truthy. This value should still be a number.

After you make this change and run your code, 'Great! You've started your work!' should log to the console.

Hint

You can change the assignment of wordCount at the top of the file:

let wordCount = \_\_insert value here\_\_;

if (wordCount) { ... }

Or reassign wordCount after the declaration but before the if...else statement.

let wordCount = 0;

wordCount = \_\_insert value here\_\_;

if (wordCount) { ... }

**2.**

Change the value of favoritePhrase so that it is still a string but falsy.

After you make this change and run your code, 'This string is definitely empty.' should log to the console.

Hint

A string that is falsy has the value of '' or "".

let wordCount = 1;

if (wordCount) {

  console.log("Great! You've started your work!");

} else {

  console.log('Better get to work!');

}

let favoritePhrase = '';

if (favoritePhrase) {

  console.log("This string doesn't seem to be empty.");

} else {

  console.log('This string is definitely empty.');

}

**Truthy and Falsy Assignment**

Truthy and falsy evaluations open a world of short-hand possibilities!

Say you have a website and want to take a user’s username to make a personalized greeting. Sometimes, the user does not have an account, making the username variable falsy. The code below checks if username is defined and assigns a default string if it is not:

let defaultName;

if (username) {

defaultName = username;

} else {

defaultName = 'Stranger';

}

If you combine your knowledge of logical operators you can use a short-hand for the code above. In a boolean condition, JavaScript assigns the truthy value to a variable if you use the || operator in your assignment:

let defaultName = username || 'Stranger';

Because || or statements check the left-hand condition first, the variable defaultName will be assigned the actual value of username if is truthy, and it will be assigned the value of 'Stranger' if username is falsy. This concept is also referred to as *short-circuit evaluation*.

**Instructions**

**1.**

Let’s use short-circuit evaluation to assign a value to writingUtensil. Do not edit tool yet, we’ll return to tool in the next step.

Assign to writingUtensil the value of tool and if tool is falsy, assign a default value of 'pen'.

Hint

Make sure you’re assigning tool before 'pen'. You will be using the || operator to short-circuit evaluate. For example:

let newVar = anotherVar || 'default value'

**2.**

Notice that text 'The pen is mightier than the sword' logged to the console. Which means the value of writingUtensil is 'pen'.

What if we reassign the value of tool to 'marker'. Let’s see what happens to the value of writingUtensil.

Hint

Since we reassign tool to a non-empty string, the value of writingUtensil becomes 'marker' and 'The marker is mightier than the sword' is logged to the console.

let tool = 'marker';

// Use short circuit evaluation to assign  writingUtensil variable below:

let writingUtensil = tool || 'pen';

console.log(`The ${writingUtensil} is mightier than the sword.`);

# Ternary Operator

In the spirit of using short-hand syntax, we can use a ternary operator to simplify an if...else statement.

Take a look at the if...else statement example:

let isNightTime = true;

if (isNightTime) {

console.log('Turn on the lights!');

} else {

console.log('Turn off the lights!');

}

We can use a ternary operator to perform the same functionality:

isNightTime ? console.log('Turn on the lights!') : console.log('Turn off the lights!');

In the example above:

* The condition, isNightTime, is provided before the ?.
* Two expressions follow the ? and are separated by a colon :.
* If the condition evaluates to true, the first expression executes.
* If the condition evaluates to false, the second expression executes.

Like if...else statements, ternary operators can be used for conditions which evaluate to true or false.

**Instructions**

**1.**

Refactor, or edit, the first if...else block to use a ternary operator.

Hint

The structure of a ternary operator looks like:

condition ? firstExpression : secondExpression

The firstExpression will execute if condition is truthy. If condition is falsy then the secondExpression will execute.

**2.**

Refactor the second if...else block to use a ternary operator.

**3.**

Refactor the third if...else block to use a ternary operator.

Hint

If you’re using single quotes in the falsy expression, make sure you use the escape character \. Otherwise, JavaScript will not interpret the string correctly. For example:

'Look we\'re using the escape character!'

You can also avoid using \ altogether by using double quotes:

"Look we're not using the escape character!"

You still have to check if favoritePhrase is equivalent to 'Love That!' in your ternary operator.

let isLocked = false;

isLocked ? console.log('You will need a key to open the door.') : console.log('You will not need a key to open the door.');

let isCorrect = true;

isCorrect ? console.log('Correct!') : console.log('Incorrect!');

let favoritePhrase = 'Love That!';

(favoritePhrase === 'Love That!') ? console.log('I love that!') : console.log("I don't love that!");

**Else If Statements**

We can add more conditions to our if...else with an else if statement. The else if statement allows for more than two possible outcomes. You can add as many else if statements as you’d like, to make more complex conditionals!

The else if statement always comes after the if statement and before the else statement. The else if statement also takes a condition. Let’s take a look at the syntax:

let stopLight = 'yellow';

if (stopLight === 'red') {

console.log('Stop!');

} else if (stopLight === 'yellow') {

console.log('Slow down.');

} else if (stopLight === 'green') {

console.log('Go!');

} else {

console.log('Caution, unknown!');

}

The else if statements allow you to have multiple possible outcomes. if/else if/else statements are read from top to bottom, so the first condition that evaluates to true from the top to bottom is the block that gets executed.

In the example above, since stopLight === 'red' evaluates to false and stopLight === 'yellow' evaluates to true, the code inside the first else if statement is executed. The rest of the conditions are not evaluated. If none of the conditions evaluated to true, then the code in the else statement would have executed.

**Instructions**

**1.**

Let’s create a program that keeps track of the way the environment changes with the seasons. Write a conditional statement to make this happen!

In **main.js** there is already an if...else statement in place. Let’s add an else if statement that checks if season is equal to 'winter'.

Inside the code block of the else if statement, add a console.log() that prints the string 'It\'s winter! Everything is covered in snow.'.

Hint

Add the else if statement after the if statement but before the else statement. The else if statement takes a condition that is wrapped in parentheses (). For example:

if (someCondition) {

// Code for if statement

} else if (anotherCondition) {

// Code for else if statement

} else {

// Code for else statement

}

**2.**

Add another else if statement that checks if season is equal to 'fall'.

Inside the code block of the else if statement you just created, add a console.log() that prints the string 'It\'s fall! Leaves are falling!'.

Hint

Add the second else if statement after the first else if statement but before the else statement. The else if statement takes a condition that is wrapped in parentheses (). For example:

if (someCondition) {

// Code for if statement

} else if (anotherCondition) {

// Code for else if statement

} else if (yetAnotherCondition) {

// Code for second else if statement

} else {

// Code for else statement

}

**3.**

Add a final else if statement that checks if season is equal to 'summer'.

Inside the code block of the else if statement you just created, add a console.log() that prints the string 'It\'s sunny and warm because it\'s summer!'.

Hint

Add the third else if statement after the second else if statement but before the else statement. The else if statement takes a condition that is wrapped in parentheses (). For example:

if (someCondition) {

// Code for if statement

} else if (anotherCondition) {

// Code for else if statement

} else if (yetAnotherCondition) {

// Code for second else if statement

} else if (finalCondition) {

// Code for third else if statement

} else {

// Code for else statement

}

let season = 'summer';

if (season === 'spring') {

  console.log('It\'s spring! The trees are budding!');

} else if (season === 'winter') {

  console.log('It\'s winter! Everything is covered in snow.');

} else if(season === 'fall'){

  console.log('It\'s fall! Leaves are falling!');

}else if(season === 'summer'){

  console.log('It\'s sunny and warm because it\'s summer!');

}

else {

  console.log('Invalid season.');

}

# The switch keyword

else if statements are a great tool if we need to check multiple conditions. In programming, we often find ourselves needing to check multiple values and handling each of them differently. For example:

let groceryItem = 'papaya';

if (groceryItem === 'tomato') {

console.log('Tomatoes are $0.49');

} else if (groceryItem === 'papaya'){

console.log('Papayas are $1.29');

} else {

console.log('Invalid item');

}

In the code above, we have a series of conditions checking for a value that matches a groceryItem variable. Our code works fine, but imagine if we needed to check 100 different values! Having to write that many else if statements sounds like a pain!

A switch statement provides an alternative syntax that is easier to read and write. A switch statement looks like this:

let groceryItem = 'papaya';

switch (groceryItem) {

case 'tomato':

console.log('Tomatoes are $0.49');

break;

case 'lime':

console.log('Limes are $1.49');

break;

case 'papaya':

console.log('Papayas are $1.29');

break;

default:

console.log('Invalid item');

break;

}

// Prints 'Papayas are $1.29'

* The switch keyword initiates the statement and is followed by ( ... ), which contains the value that each case will compare. In the example, the value or expression of the switch statement is groceryItem.
* Inside the block, { ... }, there are multiple cases. The case keyword checks if the expression matches the specified value that comes after it. The value following the first case is 'tomato'. If the value of groceryItem equalled 'tomato', that case‘s console.log() would run.
* The value of groceryItem is 'papaya', so the third case runs— Papayas are $1.29 is logged to the console.
* The break keyword tells the computer to exit the block and not execute any more code or check any other cases inside the code block. Note: Without the break keyword at the end of each case, the program would execute the code for all matching cases and the default code as well. This behavior is different from if/else conditional statements which execute only one block of code.
* At the end of each switch statement, there is a default statement. If none of the cases are true, then the code in the default statement will run.

**Instructions**

**1.**

Let’s write a switch statement to decide what medal to award an athlete.

athleteFinalPosition is already defined at the top of **main.js**. So start by writing a switch statement with athleteFinalPosition as its expression.

Hint

We’ll add cases in the next step. The syntax for your switch statement should look like:

switch(expression) {

}

**2.**

Inside the switch statement, add three cases:

* The first case checks for the value 'first place'
  + If the expression’s value matches the value of the case then console.log() the string 'You get the gold medal!'
* The second case checks for the value 'second place'
  + If the expression’s value matches the value of the case then console.log() the string 'You get the silver medal!'
* The third case checks for the value 'third place'
  + If the expression’s value matches the value of the case then console.log() the string 'You get the bronze medal!'

Remember to add a break after each console.log().

Hint

Your code will look like:

switch(athleteFinalPosition) {

case 'first place':

console.log('You get the gold medal!');

break;

// Add in the other two case statements.

}

**3.**

Now, add a default statement at the end of the switch that uses console.log() to print 'No medal awarded.'.

If athleteFinalPosition does not equal any value of our cases, then the string 'No medal awarded.' is logged to the console.

Remember to add the break keyword at the end of the default case.

Hint

Your code will look like:

switch(athleteFinalPosition) {

...

default:

// This code runs if no case matches

break;

}

let athleteFinalPosition = 'first place';

switch(athleteFinalPosition) {

  case 'first place':

  console.log('You get the gold medal!');

  break;

  case 'second place':

  console.log('You get the silver medal!');

  break;

  case 'third place':

  console.log('You get the bronze medal!');

  break;

  default:

  console.log('No medal awarded.');

  break;

}

# Review

Way to go! Here are some of the major concepts for conditionals:

* An if statement checks a condition and will execute a task if that condition evaluates to true.
* if...else statements make binary decisions and execute different code blocks based on a provided condition.
* We can add more conditions using else if statements.
* Comparison operators, including <, >, <=, >=, ===, and !== can compare two values.
* The logical and operator, &&, or “and”, checks if both provided expressions are truthy.
* The logical operator ||, or “or”, checks if either provided expression is truthy.
* The bang operator, !, switches the truthiness and falsiness of a value.
* The ternary operator is shorthand to simplify concise if...else statements.
* A switch statement can be used to simplify the process of writing multiple else if statements. The break keyword stops the remaining cases from being checked and executed in a switch statement.

**Instructions**

In **main.js**, practice the skills you learned in this lesson.