



JavaScript Conditionals

Cheat

sheet: <https://www.codecademy.com/learn/introduction-to-javascript/modules/learn-javascript-control-flow/reference>

What are Conditional Statements?

A conditional statement checks specific condition(s) and performs a task based on the condition(s).

if, else if, and else statements.

comparison operators.

logical operators.

truthy vs falsy values.

ternary operators.

the switch statement.

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

Comparison Operators

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`.

comparison statements

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 (`>`, `<`, `<=`, `>=`, `===`, `!==`).

Scope

Scope

Scope defines where variables and functions are accessible inside of your program. In JavaScript, there are two kinds of scope - **global scope**, and **function scope**.

Var vs let

The main difference between `var` and `let` is that instead of being function scoped, `let` is block scoped. What that means is that a variable created with the `let` keyword is available inside the “block” that it was created in as well as any nested blocks. When I say “block”, I mean anything surrounded by a curly brace `{ }` like in a `for` loop or an `if` statement.

var, let and const

var declarations are globally scoped or function scoped while let and const are block scoped.

var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.

They are all hoisted to the top of their scope but while var variables are initialized with undefined, let and const variables are not initialized.

While var and let can be declared without being initialized, const must be initialized during declaration.

Lets compare

```
function discountPrices (prices, discount) {
  var discounted = []

  for (var i = 0; i < prices.length; i++) {
    var discountedPrice = prices[i] * (1 - discount)
    var finalPrice = Math.round(discountedPrice * 100) / 100
    discounted.push(finalPrice)
  }

  console.log(i) // 3
  console.log(discountedPrice) // 150
  console.log(finalPrice) // 150

  return discounted
}
```

Remember that we were able to log `i`, `discountedPrice`, and `finalPrice` outside of the for loop since they were declared with `var` and `var` is function scoped.

```
function discountPrices (prices, discount) {
  let discounted = []

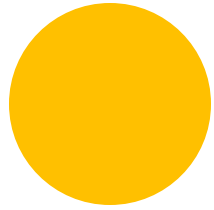
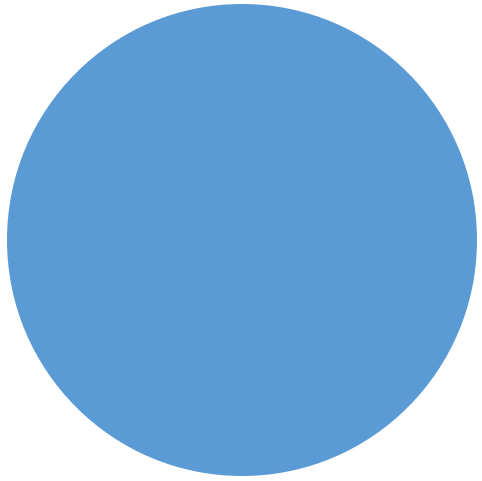
  for (let i = 0; i < prices.length; i++) {
    let discountedPrice = prices[i] * (1 - discount)
    let finalPrice = Math.round(discountedPrice * 100) / 100
    discounted.push(finalPrice)
  }

  console.log(i) // 3
  console.log(discountedPrice) // 150
  console.log(finalPrice) // 150

  return discounted
}

discountPrices([100, 200, 300], .5) // ✗ ReferenceError: i is not
defined
```

We get `ReferenceError: i is not defined`. What this tells us is that variables declared with `let` are block scoped, not function scoped. So trying to access `i` (or `discountedPrice` or `finalPrice`) outside of the “block” they were declared in is going to give us a reference error as we just barely saw.



Simply put

var VS let

var: function scoped

let: block scoped

Hoisting

- In JavaScript, variables are initialized with the value of `undefined` when they are created. That's "Hoisting" is. The JavaScript interpreter will assign variable declarations a default value of `undefined` during what's called the "Creation" phase.

Variable Declaration vs Initialization

- A variable declaration introduces a new identifier.

Like this ---- var age

We declared it but we havnt given it a value to initialize it.

Like this ---- var age =16

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 (!)

&&

and

|| or

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

! NOT

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

See if you get the humor:

`!false`

It's funny because its true!

Truthy and Falsy

anything that returns true in a conditional is called **truthy**. Anything that returns false in a conditional is called **falsy**. All object types can be described as either **truthy** or **falsy**.

```
let myVariable = 'I Exist!';  
if (myVariable) {  
  console.log(myVariable)  
} else {  
  console.log('The variable does not  
exist.')}
```


Truthy falsy

truthy



```
let myVariable = 'I Exist!';
if (myVariable) {
  console.log(myVariable)
} else {
  console.log('The variable does
not exist.')
}
```

The list of falsy values includes:

- \emptyset
- 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

Ternary Operator

The conditional operator assigns a value to a variable based on a condition.

Syntax	Example	See the condition?
<code>variablename = (condition) ? value1:value2</code>	<code>voteable = (age < 18) ? "Too young":"Old enough"</code> 	

The switch keyword

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.

Switch sample

```
<script>
let athleteFinalPosition = prompt("did you get first place, second
place or third place?");

switch(athleteFinalPosition){
  case 'first place':
    document.write('You get the gold medal!');
    break;
  case 'second place':
    document.write('You get the silver medal!');
    break;
  case 'third place':
    document.write('You get the bronze medal!');
    break;
  default:
    document.write('No medal awarded. ');
    break;
}

alert("Thanks for playing");

</script>
```