

{ POWER.CODERS }

Conditions and loops in JavaScript

AGENDA

Today we will do

- > Short Recap
- > Conditions
- > Loops

HOW DO YOU DEFINE A VARIABLE?



You need

HOW DO YOU DEFINE A VARIABLE?

You need

- > a keyword `let`, `var` or `const`
- > a name for the variable (remember the list of forbidden words)
- > a value for the variable (optional)

KEYWORDS FOR VARIABLES




KEYWORDS FOR VARIABLES

- > `var` variables can be re-declared and updated. Scope is global or function.
- > `let` variables can be updated, but not re-declared within the same scope. Scope is block. New in ES6.
- > `const` variables maintain constant values. Their scope is block. Their values cannot be updated. They cannot be re-declared. New in ES6.

WHICH NAMES FOR JS VARIABLES ARE ACCEPTABLE?

- > total%sum
- > 1number
- > firstNumber
- > _sum

HOW CAN YOU CHECK WHAT YOUR CODE IS DOING?



HOW CAN YOU CHECK WHAT YOUR CODE IS DOING?

- > `console.log()`
- > `window.alert()`
- > `document.write()`

CAN YOU TELL ME THE DATA TYPES PER LINE?

- > 42
- > "my text is great"
- > false

EXERCISE: AGE CALCULATOR

Want to find out how old you'll be? Calculate it!

- Store your birth year in a variable.
- Store a future year in a variable.
- Calculate your 2 possible ages for that year based on the stored values.

For example, if you were born in 1988, then in 2026 you'll be either 37 or 38, depending on what month it is in 2026.

- Output them to the screen/console like so: "I will be either NN or NN in YYYY", substituting the values.

SOLUTION: AGE CALCULATOR

```
let birthYear = 1981;
let futureYear = 2067;
let ageBeforeBirthday = futureYear - birthYear;
let ageAfterBirthday = ageBeforeBirthday - 1;
document.write(`I will be either ${ageAfterBirthday} or ${ageBeforeBirthday}
  in ${futureYear}`);
```

SOLUTION: AGE CALCULATOR

```
let birthYear = 1981;
let futureYear = 2067;
let ageBeforeBirthday = futureYear - birthYear;
let ageAfterBirthday = ageBeforeBirthday - 1;
document.write(`I will be either ${ageAfterBirthday} or ${ageBeforeBirthday}
  in ${futureYear}`);
```

I will be either **85** or **86** in **2067**.

BUILDING BLOCKS OF PROGRAMMING:

Remember those buildings blocks?

- > Data Types
- > Variables
- > Conditions
- > Loops
- > Functions

BUILDING BLOCKS OF PROGRAMMING:

Remember those buildings blocks?

- > Data Types
- > Variables
- > Conditions
- > Loops
- > Functions

Today, we will learn about **conditions** and **loops**.

CONDITIONS



CONDITIONS

With **conditions** you can influence the code the browser will execute.

CONDITIONS

With **conditions** you can influence the code the browser will execute.

- If a condition is **true** we will do something.
- If a condition is **false** we won't.

CONDITIONS

With **conditions** you can influence the code the browser will execute.

- If a condition is **true** we will do something.
- If a condition is **false** we won't.

**If Susanne is sitting in front of her laptop
she will go to the next slide now.**

CONDITIONS

In JavaScript a condition looks like this:

```
if(statement) {  
    doSomething();  
}
```

CONDITIONS

In JavaScript a condition looks like this:

```
if(statement) {  
    doSomething();  
}
```

Only if the statement is true `doSomething` will be executed.

AGE CALCULATOR 2.0



Let's see conditions in action by optimizing our age calculator

BOOLEAN LOGIC

Booleans are used by JavaScript to **check if a statement is true**.

Logical operators allow you to connect as many expressions as you wish.

Operator	Description	Example
&&	logical AND	true && true = true
	logical OR	true false = true
!	logical NOT	!false = true

LOGICAL OPERATORS

True and **False** are often represented by 1 and 0.

A	B	A && B	A B	!A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

COMPARISON OPERATORS

Are used in logical statements to determine equality or difference between variables or values. They return **true** or **false**.

Operator	Description	let x=5;	Result
==	equal to	x == '5'	True
===	equal value and equal type	x === '5'	False
!=	not equal	x != 3	True
>	greater than	x > 4	True
<	less than	x < 4	False
>=	greater than or equal to	x >= 5	True
<=	less than or equal to	x <= 4	False

if else

The `else` statement tells JavaScript to execute something if the condition is **false**.

```
if (statement) {  
    doSomething();  
} else {  
    doSomethingElse();  
}
```

If the condition is **false** `doSomethingElse` will be executed.

TERNARY CONDITION

```
condition ? expr1 : expr2
```

If the condition is true, provide expr1 else provide expr2.

It is like shorthand for an `if` `else` statement.

TERNARY CONDITION

```
a > b ? alert(a) : alert(b)
```

TERNARY CONDITION

```
a > b ? alert(a) : alert(b)
```

```
if(a > b) {  
    alert(a);  
} else {  
    alert(b);  
}
```

if else if

The `else if` statement specifies a new condition if the first condition is false.

```
if(a > b) {  
    alert("a is greater than b");  
} else if(a < b) {  
    alert("b is greater than a");  
} else {  
    alert("a is the same as b");  
}
```

switch

In case of many different conditions resulting in different actions use `switch` instead of multiple `if/else if` statements.

```
switch(expression) {  
  case x:  
    // code block  
    break;  
  case y:  
    // code block  
    break;  
  default:  
    // code block  
}
```

switch

In case of many different conditions resulting in different actions use `switch` instead of multiple `if/else if` statements.

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

- > `default` gets executed if no case matches. **Always** have it.
- > `break` will stop the execution of more code inside the `switch` block.

`switch` vs `if else if`

➤ **Testing expression:** `if else if` can test expressions based on conditions, `switch` only on a single integer or string

switch vs if else if

- **Testing expression:** if else if can test expressions based on conditions, switch only on a single integer or string
- **Multi-way branching:** switch is faster if there is a large group of values to select among

switch vs if else if

- **Testing expression:** if else if can test expressions based on conditions, switch only on a single integer or string
- **Multi-way branching:** switch is faster if there is a large group of values to select among
- **Boolean values:** if else if is better for conditions resulting in true or false

LOOPS



LOOPS



So far our code has been linear, read line after line. **Loops** allow you to run the same code **repeatedly**, adding a different value each time.

LOOPS

So far our code has been linear, read line after line. **Loops** allow you to run the same code **repeatedly**, adding a different value each time.

There are several types of loops:

LOOPS

So far our code has been linear, read line after line. **Loops** allow you to run the same code **repeatedly**, adding a different value each time.

There are several types of loops:

- > while
- > for
- > forEach
- > do while

LOOPS

So far our code has been linear, read line after line. **Loops** allow you to run the same code **repeatedly**, adding a different value each time.

There are several types of loops:

- > while
- > for
- > forEach
- > do while

There are a few more, we will have a look later this week.

while

The `while` loop is the easiest. It repeats through a block of code, as long as a specified condition is **true**.

```
while(statement) {  
    //codeblock this gets looped until the statment is false  
}
```

while

The `while` loop is the easiest. It repeats through a block of code, as long as a specified condition is **true**.

```
while(statement) {  
    //codeblock this gets looped until the statment is false  
}
```

If a condition is always **true**, the loop will run **forever**.
Make sure that the condition eventually becomes **false**.

for

The `for` loop is most commonly used. It has 3 statements, separated by **semi-colons**:

- > **initialization**: declare a new variable used for counting the loops, e.g. `x=0`
- > **condition**: as long as the condition is true the loop will run, e.g. `x<=5`
- > **iterator**: is executed after every iteration of the loop, e.g. `x++`

```
for(initialization; condition; iterator){  
    //codeblock this gets looped until the condition is false  
}
```

forEach

The `forEach` loop came with ES5. It is easier to read than the `for` loop, but slower. It comes down to personal preference which one to use.

```
list.forEach(function(value, index) {  
    //code block here  
});
```

forEach

The `forEach` loop came with ES5. It is easier to read than the `for` loop, but slower. It comes down to personal preference which one to use.

```
list.forEach(function(value, index) {  
    //code block here  
});
```

But wait, if you need to `return` a value and break the loop doing it, `forEach` is no help.

do while

The `do while` loop is rarely used. The loop will always be executed **at least once**, even if the condition is false.

```
do {  
    //code block here  
} while(statement);
```

CONTROLLING LOOP EXECUTIONS

The `break` statement jumps out of a loop and continues executing the code **after** the loop.

The `return` keyword will also break the loop as it immediately returns some value from a loop inside a function.

The `continue` statement breaks **only one iteration** in the loop and continues with the **next** iteration.

DEBUGGING

To detect and remove existing and potential errors (aka bugs) in your code.

It is a very important step in the developer's work. There is no program or application without any bug.

```
console.log("Is often used for debugging");  
debugger; // stops the execution of JavaScript, like setting a breakpoint
```


RECOMMEND TUTORIALS ON DEBUGGING

- Use VS Code: [How to set up debugging](#)
- Use `console.log`: [Boost your debugging skills](#)
- Use dev tools: [Tutorial on YouTube](#)
- Chrome Dev Tools: [JavaScript Debugging Reference](#)

EXERCISES



TRANSLATE DAYS OF THE WEEK

- For each single day of the week log the translated output
- e.g. if it is Monday, log "Montag" and so on
- Choose any language you like
- Try different ways: is it possible with `if` or `switch`?

ODD / EVEN REPORTER

- Prompt for a number between 0 and 20.
- Check if the given number is even or odd.
- Report the result to the screen (e.g. "2 is even").

ODD / EVEN REPORTER 2.0

- > Write a for loop that will iterate from 0 to 20.
- > For each iteration, it will check if the current number is even or odd.
- > Report the result to the screen (e.g. "2 is even").

Let's try all different types of loop

MULTIPLICATION TABLES

- Prompt for a number between 0 and 10.
- Multiply the given number by 9.
- Log the result (e.g. "2 * 9 = 18").

MULTIPLICATION TABLES 2.0

- Write a for loop that will iterate from 0 to 10.
- For each iteration of the for loop, it will multiply the number by 9.
- Log the result (e.g. "2 * 9 = 18").

WORK ON YOUR PROJECT

