

Introduction to JavaScript

Basic Syntax, Conditions and Loops



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

Table of Contents

1. Introduction
2. JavaScript Syntax
3. Conditional Statements
4. Logical Operators
5. Loops
6. Debugging and Troubleshooting





JavaScript Overview

Definition, Execution, IDE Setup

What is JavaScript?



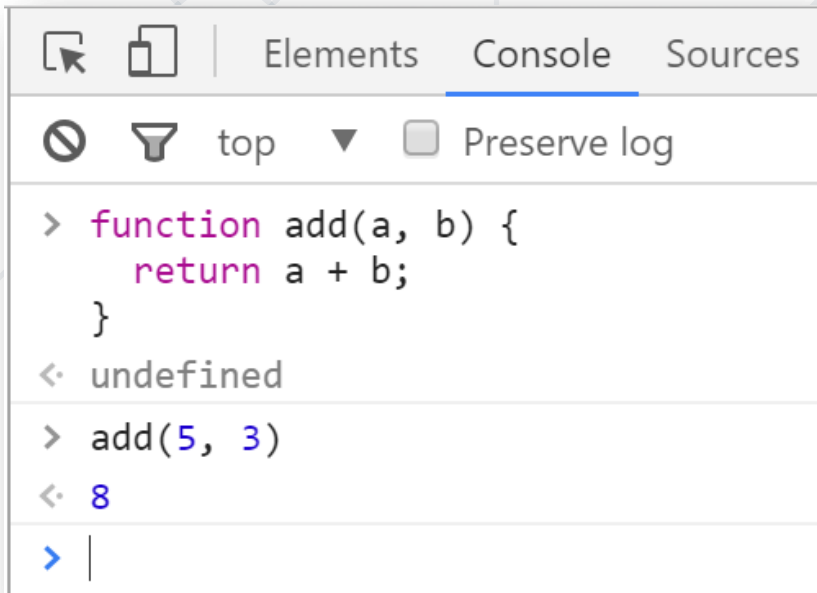
- JavaScript (**JS**) is a **high-level** programming language
 - One of the **core technologies** of the World Wide Web
 - Enables **interactive** web pages and applications
 - Can be **executed** on the **server** and on the **client**
- Features:
 - C-like **syntax** (curly-brackets, identifiers, operator)
 - **Multi-paradigm** (imperative, functional, OOP)
 - Dynamic **typing**

Web Browser Support

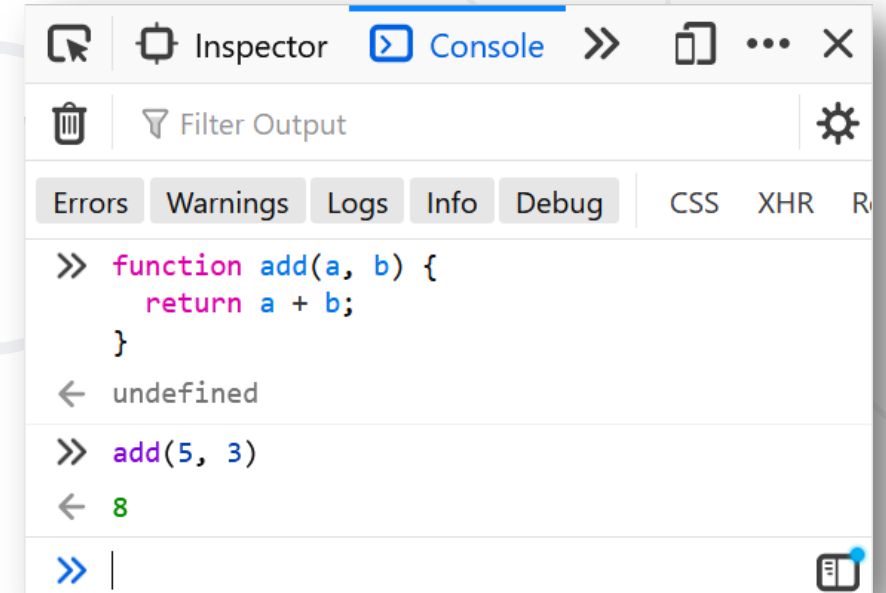
Developer Console: [F12]



The **code** shown in the example can be **executed** directly in the **browser**



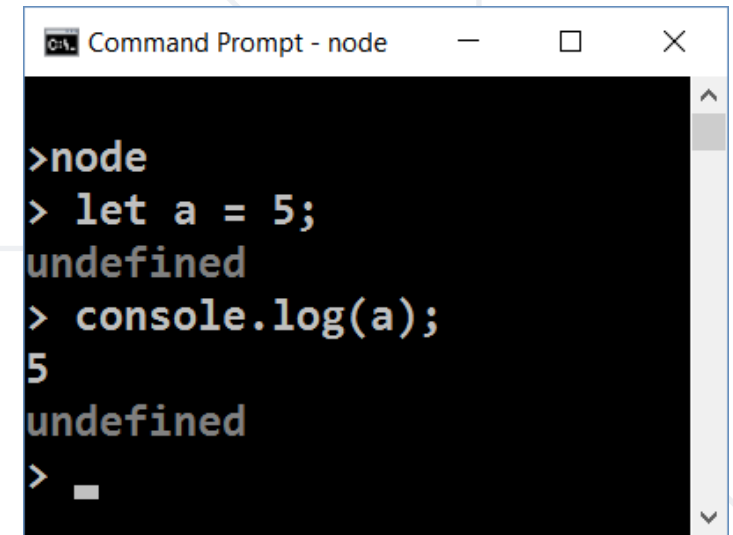
```
> function add(a, b) {  
  return a + b;  
}  
< undefined  
> add(5, 3)  
< 8  
> |
```



```
>> function add(a, b) {  
  return a + b;  
}  
< undefined  
>> add(5, 3)  
< 8  
>> |
```

Node.js

- What is **Node.js**?
 - **Server-side** JavaScript runtime
 - Chrome V8 JavaScript engine
 - NPM **package manager**
 - Install node packages



```
>node
> let a = 5;
undefined
> console.log(a);
5
undefined
>
```

Install the Latest Node.js


Downloads


Latest LTS Version: 16.13.1 (includes npm 8.1.2)


Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS
Recommended For Most Users

Current
Latest Features


Windows Installer
node-v16.13.1-x64.msi


macOS Installer
node-v16.13.1.pkg

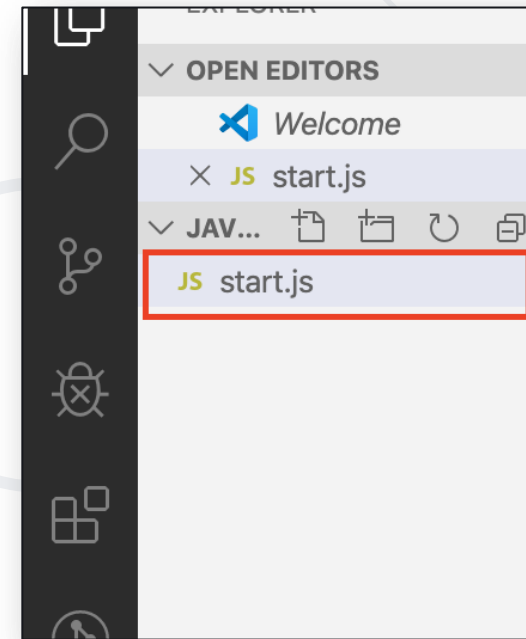
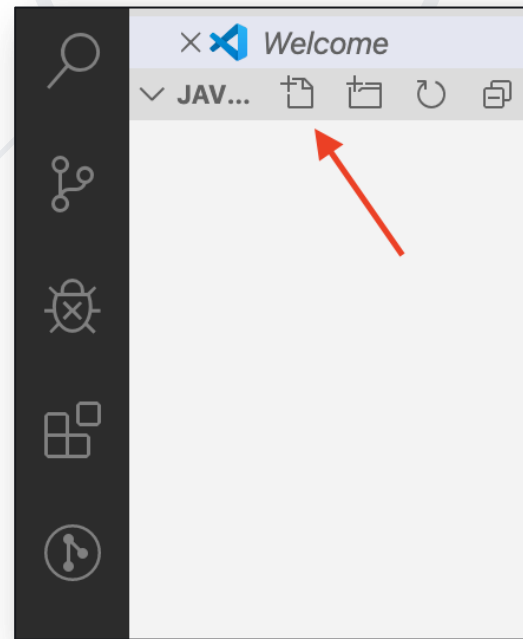
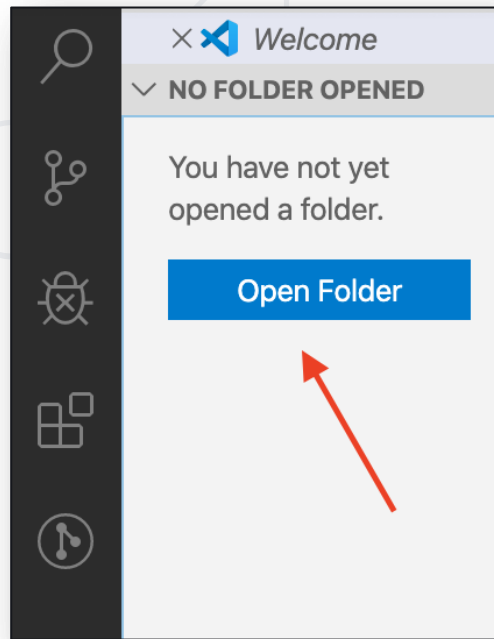

Source Code
node-v16.13.1.tar.gz

- Windows Installer (.msi)
- Windows Binary (.zip)
- macOS Installer (.pkg)
- macOS Binary (.tar.gz)
- Linux Binaries (x64)
- Linux Binaries (ARM)
- Source Code

32-bit	64-bit
32-bit	64-bit
64-bit / ARM64	
64-bit	ARM64
64-bit	
ARMv7	ARMv8
node-v16.13.1.tar.gz	

Using Visual Studio Code

- **Visual Studio Code** is powerful text editor for JavaScript and other projects
- In order to create your **first project**:






JavaScript Syntax

Functions, Operators, Input and Output

JavaScript Syntax

- C-like **syntax** (curly-brackets, identifiers, operator)
- Defining and Initializing variables:



Declare a variable with let

```
let a = 5;  
let b = 10;
```

Variable name

Variable value

- Conditional statement:

```
if (b > a) {  
    console.log(b);  
}
```

Body of the conditional statement

Functions and Input Parameters

- In order to solve different problems, we are going to use **functions** and the **input** will come as **parameters**
- A function is similar to a **procedure**, that executes when called

declaration

parameters

```
function solve (num1, num2) {  
    //some logic  
}
```

```
solve(2, 3);
```

calling the function

- We use the **console.log()** method to print to console:

```
function solve (name, grade) {  
  console.log('The name is: ' + name + ', grade: ' + grade);  
}  
solve('Peter', 3.555);  
//The name is: Peter, grade: 3.555
```

- Text can be composed easier using interpolated strings:

```
console.log(`The name is: ${name}, grade: ${grade}`);
```

- To format a number, use the **toFixed()** method (converts to **string**):

```
grade.toFixed(2); //The name is: Petar, grade: 3.56
```

Number of decimal places

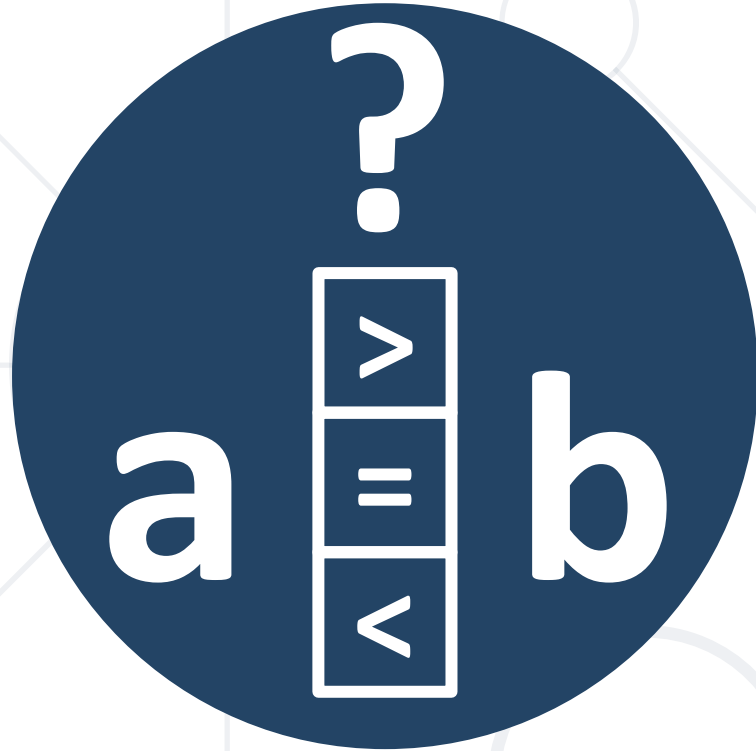
Problem: Multiply Number by Two

- Write a function that receives a **number** and prints as a result that number **multiplied by two**

Input	Output
2	4

```
function solve (num) {  
  console.log(num * 2);  
}  
solve(2);
```





Comparison Operators

Comparison Operators

Operator	Notation in JS
Equal value	<code>==</code>
Equal value and type	<code>===</code>
Not equal value	<code>!=</code>
Not equal value/type	<code>!==</code>
Greater than	<code>></code>
Greater than or Equal	<code>>=</code>
Less than	<code><</code>
Less than or Equal	<code><=</code>

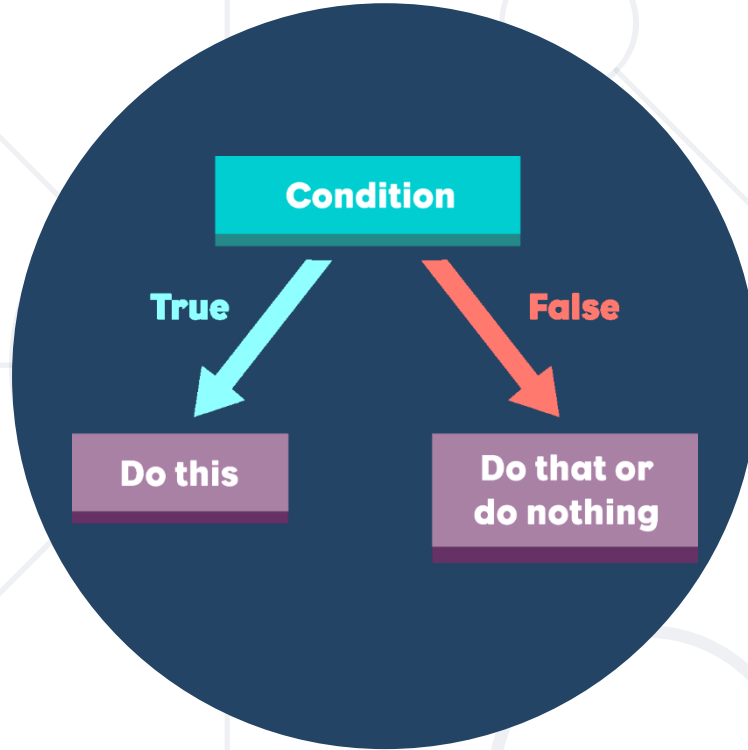


Comparison Operators

- Values can be compared:

```
let a = 5;  
let b = 10;  
console.log(a < b);           // true  
console.log(a > 0);           // true  
console.log(a > 100);         // false  
console.log(a < a);           // false  
console.log(a <= 5);          // true  
console.log(b == 2 * a);      // true  
console.log("2" === 2);      // false
```





Conditional Statements

Implementing Control-Flow Logic

What is a Conditional Statement?

The **if-else** statement:

- Do action depending on condition

```
let a = 5;  
if (a >= 5) {  
    console.log(a);  
}
```

If the condition **is met**,
the code will execute

- You can chain conditions

```
else {  
    console.log('no');  
}
```

Continue on the **next condition**, if the first is **not met**



Problem: Excellent Grade

- Write a function that receives a **single number** and checks if the grade is excellent or not.
- If it is, print "**Excellent**", otherwise print "**Not excellent**"

Input	Output
5.50	Excellent
4.35	Not excellent

```
function solve(grade){  
  if (grade >= 5.50) {  
    //TODO  
  } else {  
    //TODO  
  }  
}
```

Chained Conditional Statements

- The **if / else - if / else...** construct is a series of checks

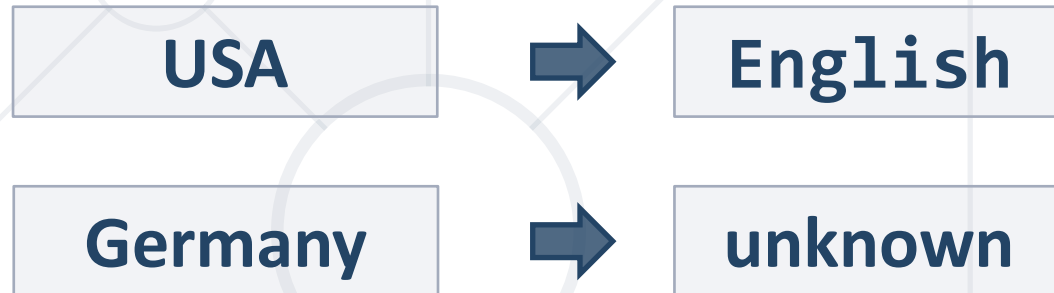
```
let a = 5;  
if (a > 10)  
    console.log("Bigger than 10");  
else if (a < 10)  
    console.log("Less than 10");  
else  
    console.log("Equal to 10");
```

Only "**Less than 10**"
will be printed

- If one condition is true, it does not proceed to verify the following conditions

Problem: Foreign Languages

- By given country print the typical spoken language:
 - English -> England, USA
 - Spanish -> Spain, Argentina, Mexico
 - other -> unknown



Solution: Foreign Languages

```
function solve(country){  
  if (country == 'England') {  
    console.log('English');  
  } else if (country == 'USA') {  
    console.log('English');  
  } else if (country == 'Spain') {  
    console.log('Spanish');  
  } else if (country == 'Argentina') {  
    console.log('Spanish');  
  } else if (country == 'Mexico') {  
    console.log('Spanish');  
  } else {  
    console.log('unknown')  
  }  
}
```

The Switch-case Statement

- Works as a series of **if / else if / else if...**

```
switch (...){
```

```
case ...:
```

```
// code
```

```
break;
```

```
case ...:
```

```
// code
```

```
break;
```

```
default:
```

```
// code
```

```
break;
```

```
}
```

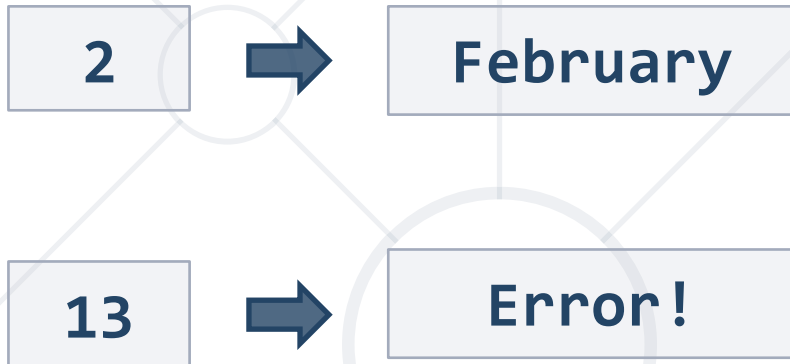
The condition in the **switch case** is a value

List of conditions (values) for the inspection

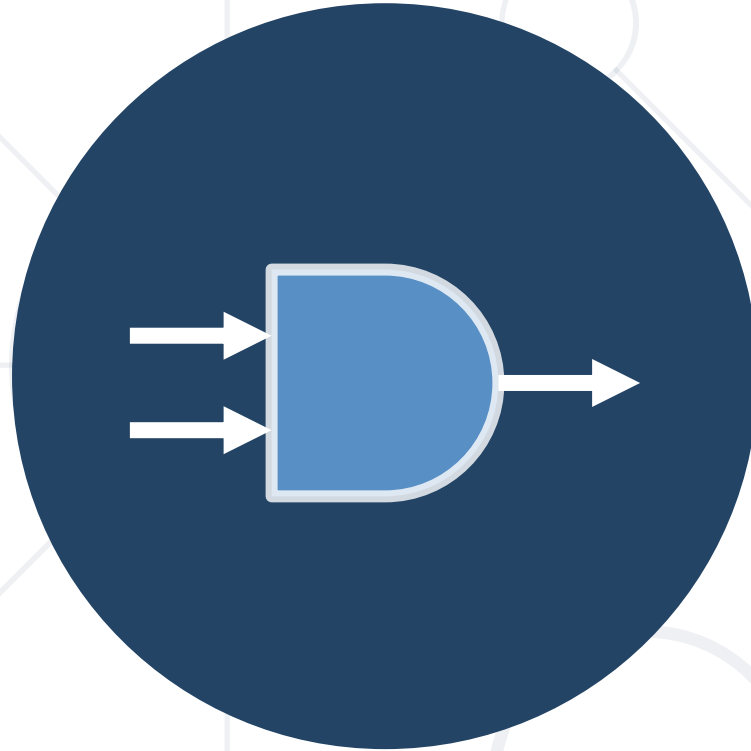
Code to be executed if there is no match with any case

Problem: Month Printer

- Write a program that takes an integer as a parameter and prints the corresponding month
- If the number **is more than 12** or **less than 1** print **"Error!"**



```
function solve(month) {  
    switch (month) {  
        case 1: console.log("January"); break;  
        case 2: console.log("February"); break;  
        // TODO: Add the other cases  
        default: console.log("Error!"); break;  
    }  
}
```

Logical Operators

Writing More Complex Conditions

- **Logical operators** give us the ability to write multiple conditions in one **if** statement
- They return a boolean result (**true** or **false**)

Operator	Description	Example
!	NOT	!false -> true
&&	AND	true && false -> false
	OR	true false -> true

- Logical **"AND"**

- Checks the fulfillment of several conditions simultaneously

```
let a = 3;  
let b = -2;  
console.log(a > 0 && b > 0); // expected output: false
```

- Logical **"OR"**

- Checks that at least one of several conditions is met

```
let a = 3;  
let b = -2;  
console.log(a > 0 || b > 0); // expected output: true
```

- Logical "NOT"
 - Checks if a condition is **not** met

```
let a = 3;  
let b = -2;  
console.log(!(a > 0 || b > 0));  
// expected output: false
```

Problem: Theatre Promotions

- A theatre has the following **ticket prices** according to the age of the visitor and the type of day
 - If the given **age** does not fit one of the categories, print: "**Error!**"

Day / Age	$0 \leq \text{age} \leq 18$	$18 < \text{age} \leq 64$	$64 < \text{age} \leq 122$
Weekday	12\$	18\$	12\$
Weekend	15\$	20\$	15\$
Holiday	5\$	12\$	10\$

Weekday
42



18\$

Holiday
-12



Error!

Solution: Theatre Promotions

```
function solve(day, age) {  
  let price = 0;  
  if (day == 'Weekday') {  
    if ((age >= 0 && age <= 18) || (age > 64 && age <= 122)) {  
      price = 12;  
    }  
    // TODO: Add else statement for the other group  
  } else if (day == 'Weekend') {  
    if ((age >= 0 && age <= 18) || (age > 64 && age <= 122)) {  
      price = 15;  
    } else if (age > 18 && age <= 64) {  
      price = 20;  
    }  
  }  
}
```

// Continued on next slide

Solution: Theatre Promotions (2)

```
else if (day == 'Holiday') {  
    if (age >= 0 && age <= 18) {  
        price = 5;  
    }  
    // TODO: Add the statements for the other cases  
}  
if (price != 0) {  
    console.log(price + '$');  
} else {  
    console.log('Error!');  
}  
}
```



Loops

Code Block Repetition

What is a Loop?

The **for** loop:

- Repeats until the condition is evaluated

```
for (let i = 1; i <= 5; i++){  
  console.log(i)  
}
```

Incrementation **in**
the condition

The **while** loop:

- Does the same, but has different structure

```
let i = 1  
while (i <= 5) {  
  console.log(i)  
  i++  
}
```

Incrementation
outside the
condition



Problem: Divisible by 3

- Print the numbers from 1 to 100, which are divisible by 3
- The program should not receive input.

```
function solve() {  
    for (let i = 3; i <= 100; i += 3){  
        console.log(i);  
    }  
}
```



Problem: Numbers from N to 1

- Write a function that receives a number **N** and prints the all numbers from **N to 1**. Try using a **while loop**.

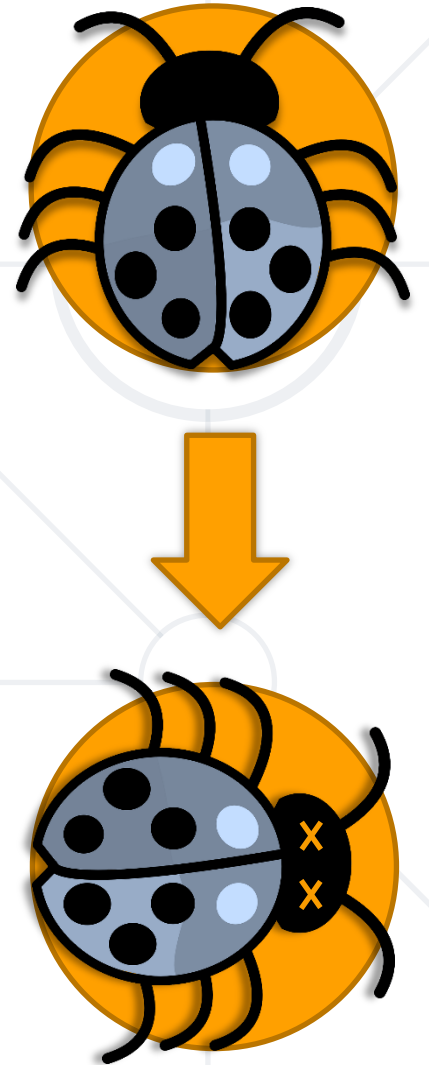
Input	Output
5	5 4 3 2 1

```
function solve(n) {  
  while(/*TODO*/) {  
    console.log(n);  
    n--;  
  }  
}  
solve(5);
```



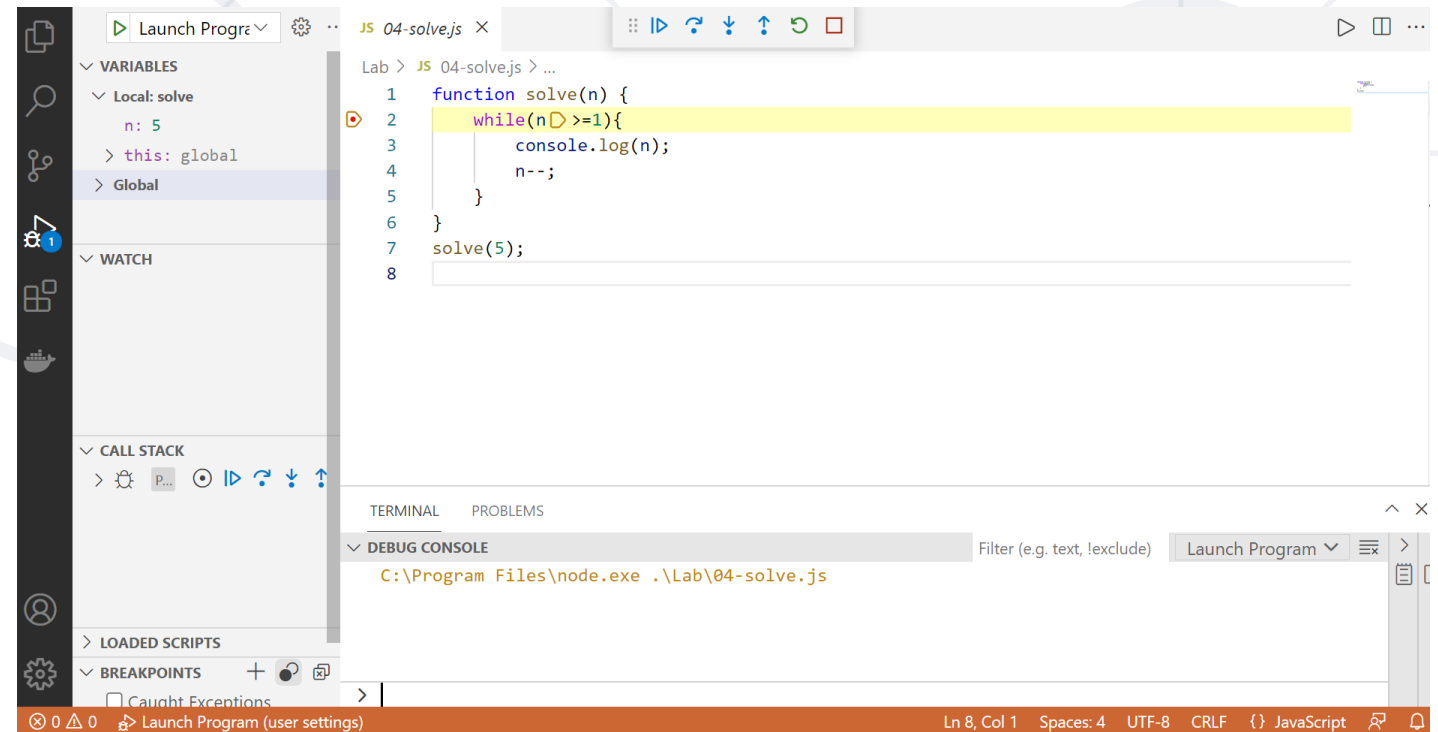
Debugging the Code

- The process of **debugging application** includes:
 - Spotting an error
 - Finding the lines of code that cause the error
 - Fixing the error in the code
 - Testing to check if the error is gone and no new errors are introduced
- Iterative and continuous process



Debugging in Visual Studio Code

- Visual Studio Code has a built-in **debugger**
- It provides:
 - **Breakpoints**
 - Ability to **trace** the code execution
 - Ability to **inspect** variables at runtime



Using the Debugger in Visual Studio Code

- Start without Debugger: **[Ctrl+F5]**
- Start with Debugger: **[F5]**
- Toggle a breakpoint: **[F9]**
- Trace step by step: **[F10]**
- Force step into: **[F11]**

- Declare variables with **'let'**
- Use **if-else** statements to check for conditions
- Use **loops** to avoid repeating code
- Use the **debugger** to check for mistakes in the code

