

Programming Languages

Very Brief Introduction to the World of Programming



SoftUni Team

Technical Trainers



SoftUni



Software University

<https://about.softuni.bg>

Have a Question?



sli.do

#qa-fund

1. Intro to **Coding**: Commands and Programs
2. **Programming Languages**
 - **Low-Level** and **High-Level** Languages
 - **Scripting** vs. **Compiled** Languages
 - **Most Popular Programming Languages**: JavaScript, Python, Java, C#
3. **IDE** (Integrated Development Environments)
4. Getting Familiar with **HTML**, **CSS** and **JavaScript**
5. Explore and Run a **Real-World Software Project**





What is "Coding"?

Programming Code, Commands, Programs

What is "Programming" (Coding)?

- Give **commands** to the computer

```
console.log(3+5)
```

```
console.log("Hello")
```

```
x = 5
```

```
console.log(x * x)
```

- Commands are arranged one after another into a **computer program / source code**

```
leva = prompt("Enter amount in BGN:")  
euro = leva / 1.95583  
console.log("Euro: ", euro)
```





Programming Languages

Machine, Assembler, C, C++, Java, Python, JS, ...

Definition of Programming Language

- **Programming language**: a formal language (syntax)
 - Used to write **instructions** (commands or programs) that can be **executed by a computer**
 - A **set of rules** (syntax and command format) used to construct computer programs (programming code)
- Different **types** of programming languages:
 - **Low-level / high-level, scripting / compiled, statically-typed / dynamic, procedural / object-oriented / functional, etc.**



History: Machine & Assembly Language

- **Machine language** (1st generation, 1940s)
 - The first programming language, used in earliest computers
 - **Binary code**, directly executed by the CPU (central processing unit)
- **Assembly language** (2nd generation, 1950s)
 - Simplify coding in machine language
 - Uses **mnemonics** to represent **machine instructions**, easier to understand and learn

machine code

```
10110100 10110111 00101011
00011010 00010100 10111011
10001000 11110111 00101000
10101010 00101101 00010001
01010010 11101100 11010001
10010100 10010100 00100000
00000000 10100001 00110001
10101001 00010101 00101010
00100100 10010100 01110001
11110101 11101011 00101111
01010010 10000101 11111110
```

```
.MODEL SMALL
.STACK 100H
.CODE

MOV AX, 0x3C
MOV BX, 00000000000001010B
ADD AX, BX
MOV BX, 14
SUB AX, BX

MOV AH, 04CH
INT 21H
```


- **Fortran** (1957) – the first high-level language
 - For scientific and engineering computations
 - Easier to read and write than assembler
- **COBOL** (1959) – developed for business applications (and is still in use today)
 - Easily to understand by analysts and managers
- **BASIC** (1964) – developed for educational purposes, easy to learn
 - Still used today for simple coding tasks

Modern Languages: C, C++, Python, Java, JS

- **C** (1972) – powerful and efficient mid-level language for system programming (e. g. the Linux kernel is written in C)
- **C++** (1983) – complex, object-oriented, highly-efficient language for system and high-performance apps and games
- **Python** (1991) – high-level, simple scripting language for scientists, easy to learn, for Web apps, data science and AI
- **Java** (1995) – popular high-level, cross-platform, object-oriented language for Web, mobile and business apps
- **JavaScript** (1995) – simple, dynamic scripting language for Web, runs in the Web browsers, also server-side

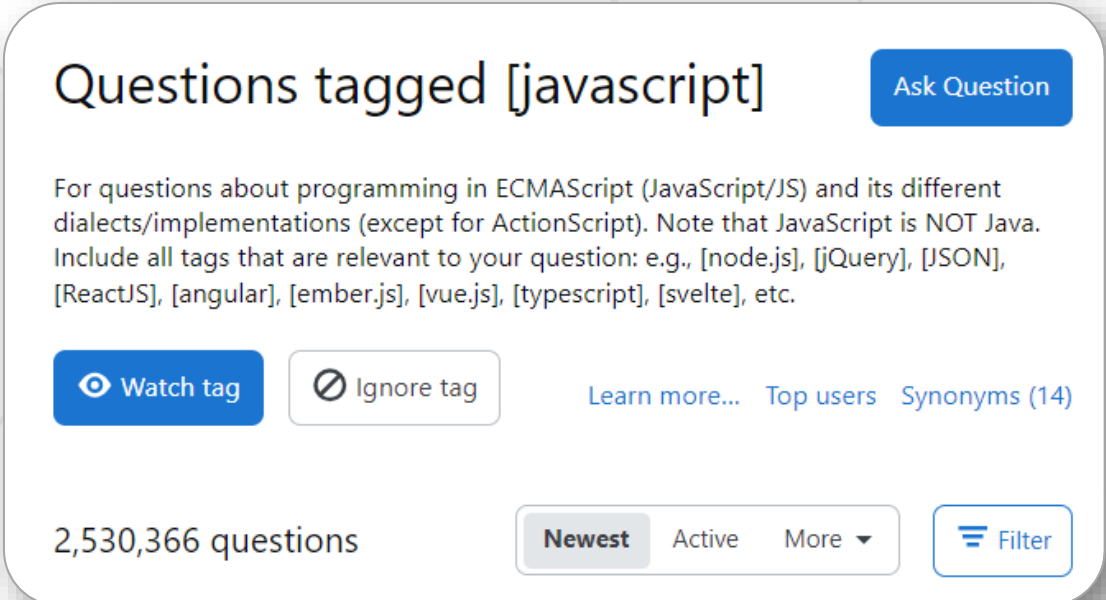
- **PHP** (1995) – scripting language for Web sites and server-side Web development
- **C#** (2000) – modern object-oriented language for universal use: business apps, Web apps, mobile apps
- **Go** (2009) – simple, efficient lang for high-performance apps
- **TypeScript** (2012) – strongly-typed JS, less prone to errors
- **Swift** (2014) – general purpose lang, for the Apple ecosystem
- Other modern mainstream languages:
 - **Kotlin, Scala, Rust, Ruby, Dart, Perl**

Most Popular Programming Languages

- **Stack Overflow** all time questions asked **by tag** (May 2024)

<https://stackoverflow.com/questions/tagged/java>

- **JavaScript** → 2.53M questions
- **Java** → 1.91M questions
- **Python** → 2.20M questions
- **C#** → 1.62M questions
- **PHP** → 1.47M questions
- **C++** → 0.80M questions



Questions tagged [javascript] [Ask Question](#)

For questions about programming in ECMAScript (JavaScript/JS) and its different dialects/implementations (except for ActionScript). Note that JavaScript is NOT Java. Include all tags that are relevant to your question: e.g., [node.js], [jQuery], [JSON], [ReactJS], [angular], [ember.js], [vue.js], [typescript], [svelte], etc.

[Learn more...](#) [Top users](#) [Synonyms \(14\)](#)

2,530,366 questions Newest Active More ▾ [Filter](#)

- Many tech languages are **not real programming languages!**
- **SQL**: database query and manipulation language
 - **PL/SQL, Transact SQL**, etc. – more powerful, still DB specialized
- **HTML** and **CSS**: visualize Web content (text + images + links)
- **XML, JSON** and **YAML**: represent, store and transport data
- **Bash / PowerShell**: system administration scripting tools
- **HCL**: describe and configure virtual infrastructure (IaC)



Low-Level & High-Level Languages

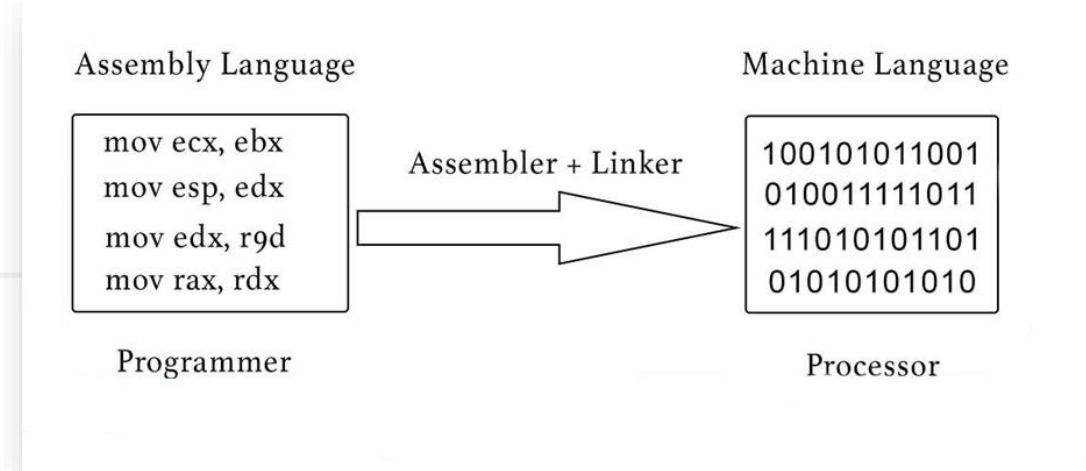
Assembler vs. Modern Languages

Low-level and high-level languages

- Programming languages may differ in terms of **level of abstraction** and their **relationship to the hardware**
- **Low-level languages** runs **closer to the hardware** and computer's architecture
- **High-level languages** are more **abstract** and easier
 - **Easy to use**, read, write and understand than low-level languages
 - Developers **write less code** in higher-level languages (e. g. Python) than in lower-level languages (e. g. C)



- Low-level languages == **machine code** and **assembly languages**
 - Write code for **direct execution the computer's CPU**
- Difficult to read and write
- Rarely used today
 - **Modern compilers** are better than humans in machine code
- Example: Assembler



Machine Language

ADD contents of 2 registers, store result in third.	1010000100 RR RR RR ex: R0 = R1 + R2 1010000100 00 01 10
SUBTRACT contents of 2 registers, store result into third	1010001000 RR RR RR ex: R0 = R1 - R2 1010001000 00 01 10
Halt the program	1111111111111111

Characteristics of Low-Level Languages

- Specific to a **particular CPU architecture** and machine
- Direct control over the **hardware** and **memory**
- **Difficult** to read, write and maintain
- Used in developing operating systems, device drivers, and firmware
- Examples of low-level languages include Assembly language and machine code



- High-level languages – **closer to the natural language**
- Often use English-like keywords and syntax to make programming **more intuitive** and easier to learn
- High-level languages may be **compiled** into low-level code
- Or can be execute line by line by a language **interpreter**

```
! function (t, e) {  
  "object" = typeof exports && "undefined" !== typeof module ? module.exports :  
  "function" = typeof define && define.amd ? define(e) : (t = "undefined"  
  globalThis ? globalThis : t || self).bootstrap = e()  
}(this, (function () {  
  "use strict";  
  const t = {  
    find: (t, e = document.documentElement) => [].concat(...Element.prototype.  
    querySelectorAll.call(e, t)),  
    findOne: (t, e = document.documentElement) => Element.prototype.querySe  
    (e, t),  
    children: (t, e) => [].concat(...t.children).filter(t => t.matches(e)),  
    parents(t, e) {  
      const i = [];  
      let n = t.parentNode;  
      for (; n && n.nodeType === Node.ELEMENT_NODE && 3 !== n.nodeType;  
        (e) && i.push(n), n = n.parentNode;
```

Characteristics of High-Level Languages

- More **abstract** and use natural **language-like syntax**
 - Require **less code** to be written
 - **Easier to learn** and use than low-level languages
 - **Independent** of the CPU architecture and machine
- Used in developing Web and Mobile apps, scientific simulations, and enterprise software
- Examples of high-level languages:
 - Java, Python, C#, and JavaScript, PHP





Scripting vs. Compiled Languages

Interpreters vs. Compilers

Scripting Languages

- A **scripting** language is designed to be **interpreted**
 - Executed command-by-command by an interpreter
 - Typically, slower than compiled languages
 - Support the **REPL** model: Read-Evaluate-Print Loop
 - Dynamic, flexible in terms of syntax and error handling
 - Used for automating tasks, such as system administration, web development, and data analysis
 - Examples: **Python, JavaScript, Perl, VB.NET**



Compiled Languages

- A **compiled** language is **compiled** into machine code, which can be directly executed by a computer's CPU
- **More efficient** in terms of **memory usage** and **execution speed** than scripting languages
- Used for developing large-scale applications, such as operating systems, games, and enterprise software
- Less dynamic, strict typing system, strict syntax, more complex than scripting languages
- Examples: **C++, Java, C#, Go**



Language Execution Model

- **Compiled** languages

- Source code is first **compiled** to machine code, then executed
- Syntax errors are found during the **compilation** (at compile time)
- Examples: **C#, Java, C, C++, Swift, Go, Rust**



- **Interpreted** languages

- Each command is read, parsed and executed by an **interpreter**
- Syntax errors are found at **run-time**, during execution
- Examples: **Python, JavaScript, PHP, Perl, Ruby**





Most Popular Languages

C#, Java, JavaScript, Python

C# Programming Language

- Modern, flexible, general-purpose programming language
- **Object-oriented** by nature, statically-typed, compiled
- Runs on .NET Framework / .NET Core



```
static void Main()  
{  
    Console.WriteLine(3+5);  
}
```

Program starting
point

C#: Declaring Variables

- Defining and initializing variables

```
{data type / var} {variable name} = {value};
```

- Example

Variable name

```
int number = 5;
```

Data type


Variable value

```
Console.WriteLine(number);
```

Printing a variable

Java Programming Language

- Modern, flexible, general-purpose programming language
- **Object-oriented** by nature, statically-typed, compiled



```
static void main(String[] args) {  
    int a = 5;  
    System.out.println(a * a);  
}
```

Program
starting
point

- Defining and Initializing variables

```
{data type / var} {variable name} = {value};
```

- Example

Variable name

```
int number = 5;
```

Variable value

Data type

Printing a variable

```
System.out.println(number);
```

Python Programming Language

- Popular, easy to learn, scripting language
- A good start for beginners
- The syntax is close to plain **English language**
- **Interpreted** language
- Dynamic typing
- Supports object-oriented programming
- Cross-platform, tons of libraries



- **Variables** – they are way to **store information** and are characterized by **name**, **type** and **value**

```
age = 25
```

Name

Value

- **Printing data:**

```
print("age = ", age)
```

JavaScript Programming Language

- 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**



- 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** are named pieces of code
 - Can take **input data** (**parameters**) and return **output data** (**result**)
- A function can be **invoked** (called) to execute its body

Declaration

Parameters

```
function calcSum(num1, num2) {  
  return num1 + num2;  
}
```

Call the function

```
let sum = calcSum(2, 3);  
console.log("sum = " + sum);
```

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

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

- Text can be composed easier using **interpolated strings**:

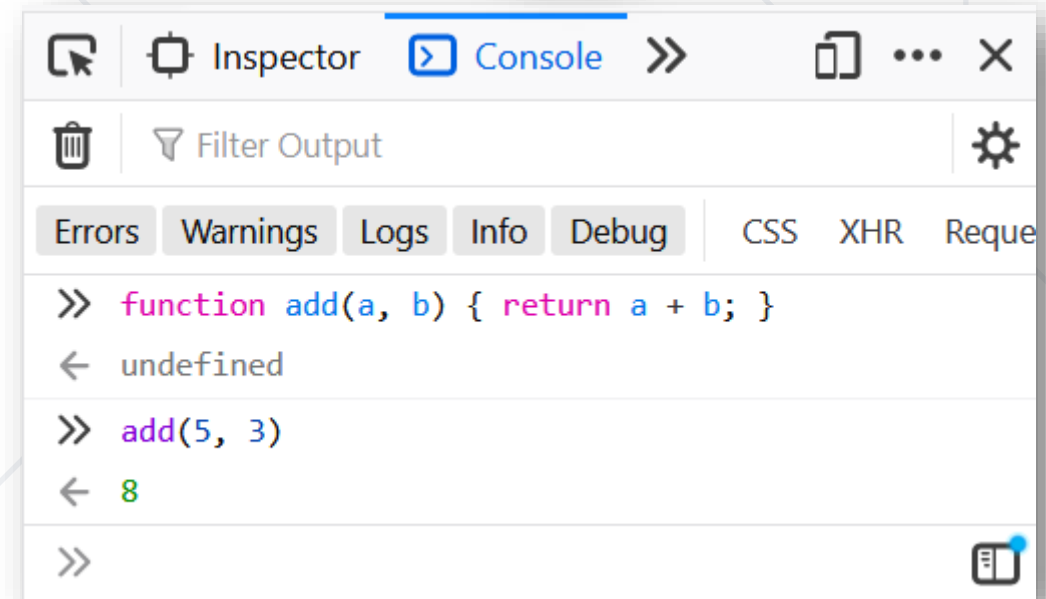
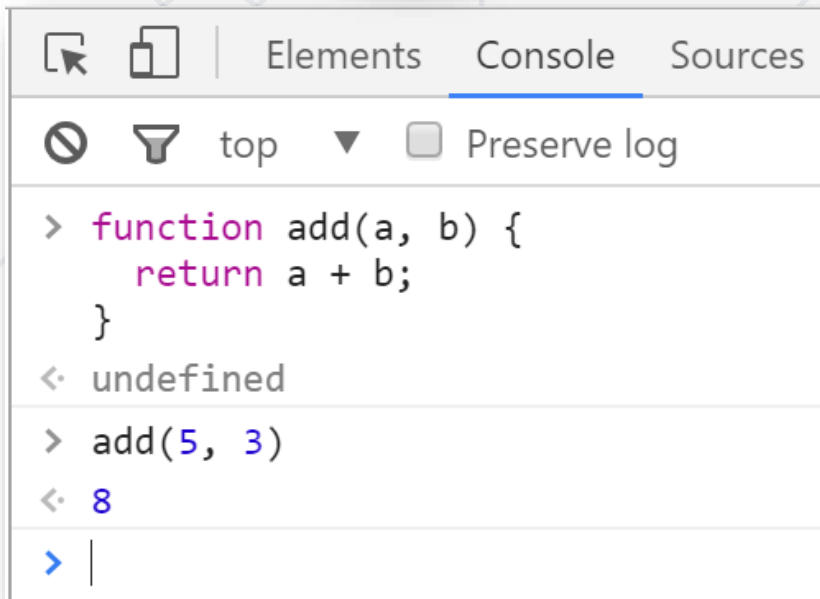
```
console.log(`The name is: ${name}, grade: ${grade}`);
```

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

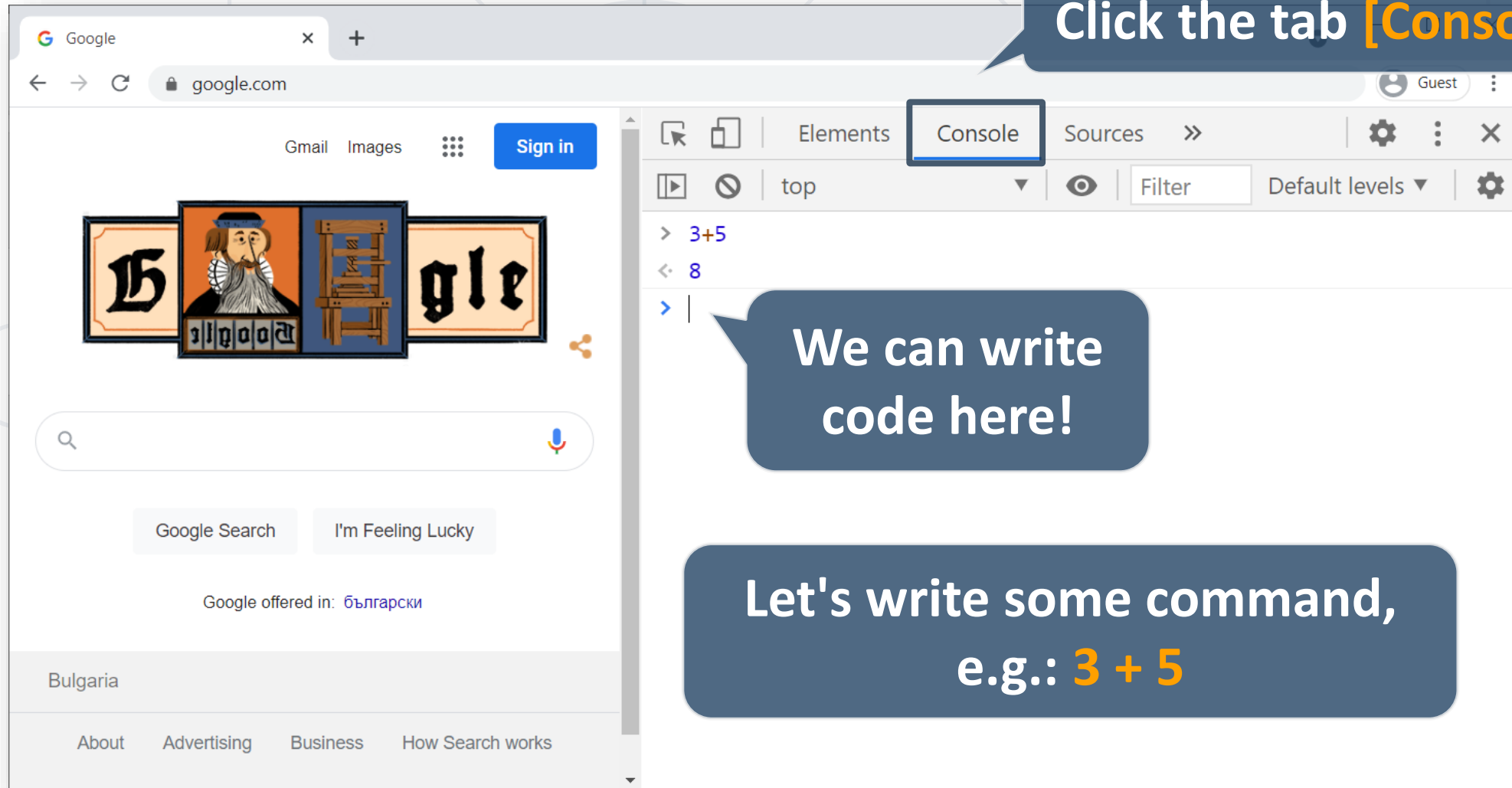
```
grade.toFixed(2); //The name is: Petar, grade: 3.56
```

JS in Chrome Web Browser / Firefox

- Developer Console: **[F12]**



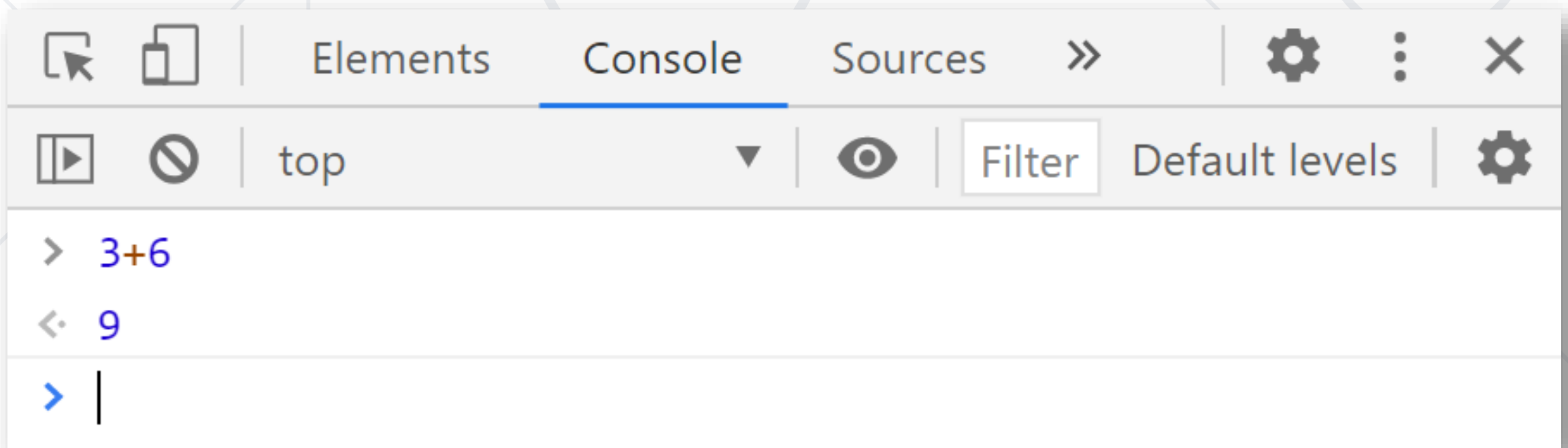
Writing JavaScript Code in the Browser



Problem: Calculate 3 + 6

- Calculate the expression **3+6**

3 + 6

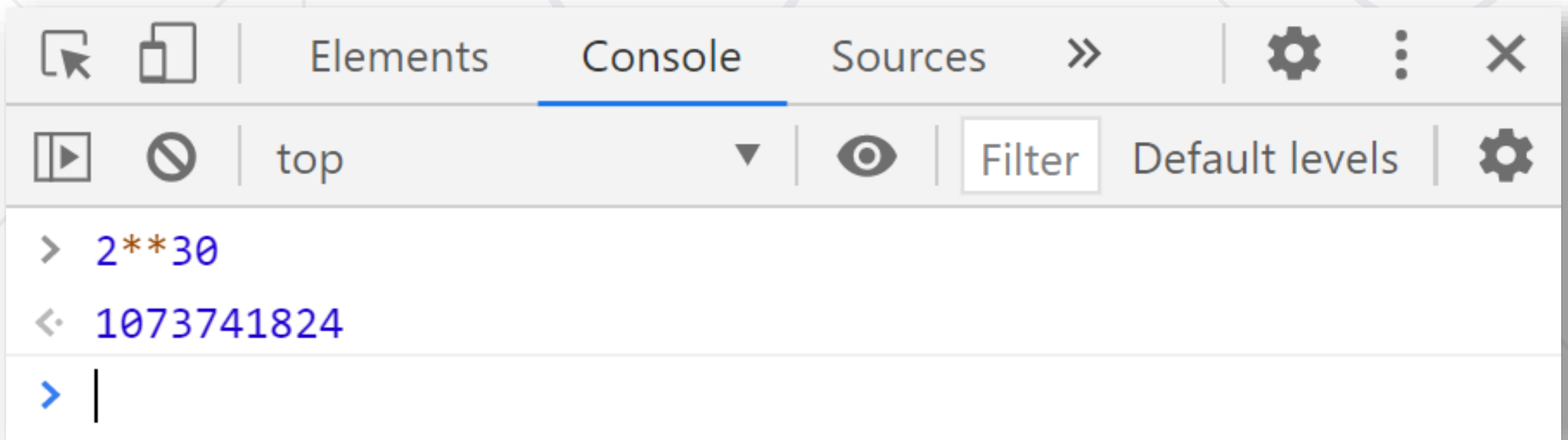


Problem: 2^{30}

- Calculate 2^{30} ($2*2*...*2$ multiplied 30 times)

```
2 ** 30
```

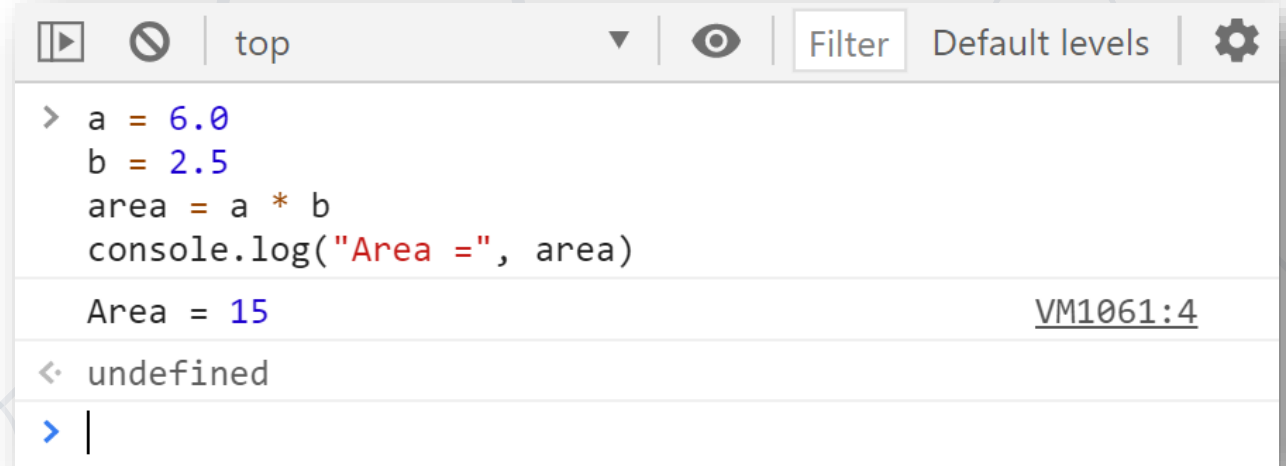
Look on the Internet for
"JavaScript exponentiation"



Problem: Area of Rectangle

- Calculate the **area of a rectangle** with sides **6.0** and **2.5**

```
a = 6.0  
b = 2.5  
area = a * b  
console.log("Area =", area)
```



```
> a = 6.0  
b = 2.5  
area = a * b  
console.log("Area =", area)  
  
Area = 15  
< undefined  
> |
```

Reading Input Data

```
a = prompt("Enter a:")  
b = prompt("Enter b:")  
console.log("Area = " + a * b)
```

Enter a:

OK

Cancel

Enter b:

OK

Cancel

```
> a = prompt("Enter a:")  
b = prompt("Enter b:")  
console.log("Area = " + a * b)  
  
Area = 15
```

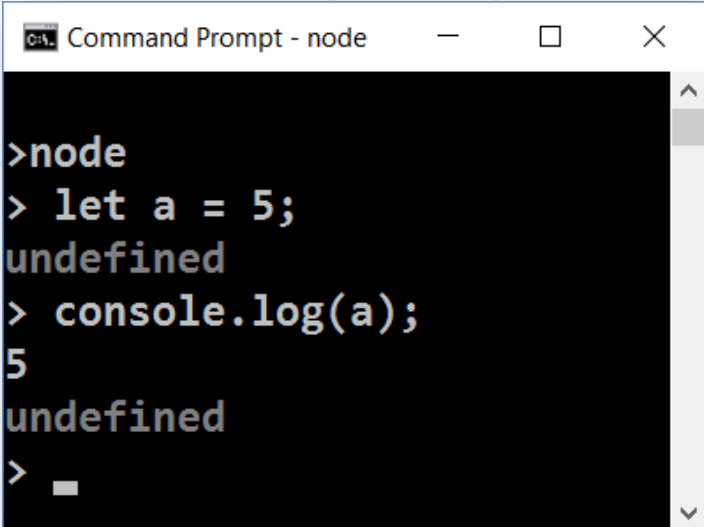



Setting up Node.js + VS Code

Running JavaScript in VS Code

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: 18.16.0 (includes npm 9.5.1)


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-v18.16.0-x64.msi


macOS Installer
node-v18.16.0.pkg


Source Code
node-v18.16.0.tar.gz

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

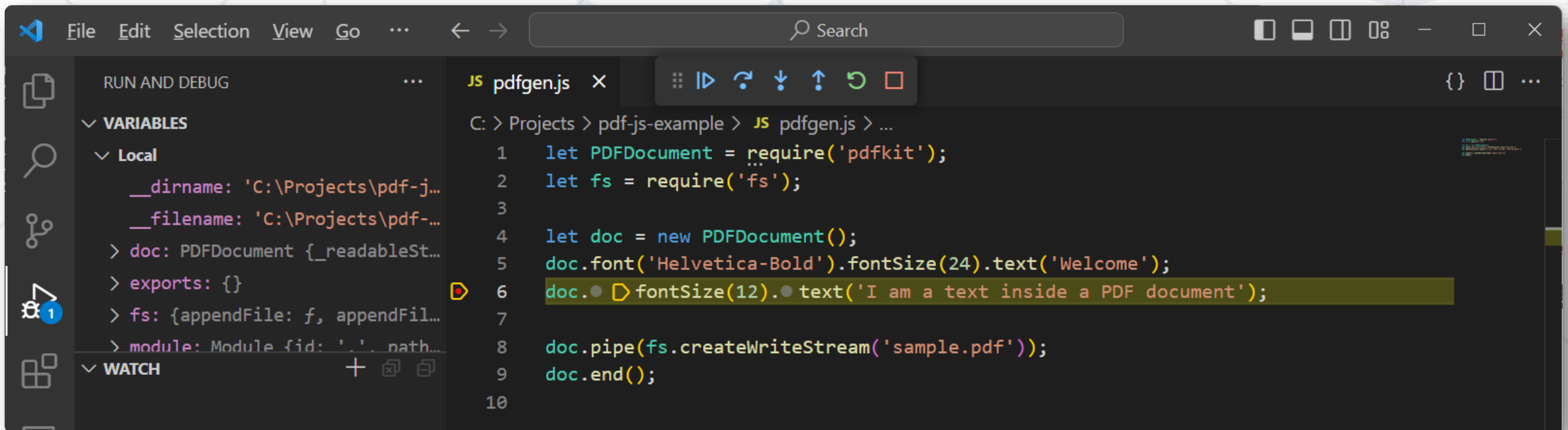
32-bit	64-bit
32-bit	64-bit
64-bit / ARM64	
64-bit	ARM64
64-bit	
ARMv7	ARMv8
node-v18.16.0.tar.gz	

What is an IDE?

- **IDE == Integrated Development Environment**
 - Write code, run the code, debug the code
 - Syntax highlighting, auto-complete, plugins
- **Examples:**
 - **Visual Studio** – IDE for C# and .NET
 - **PyCharm** – IDE for Python
 - **IntelliJ IDEA** – IDE for Java
 - **Web Storm** – IDE for HTML, CSS and JS

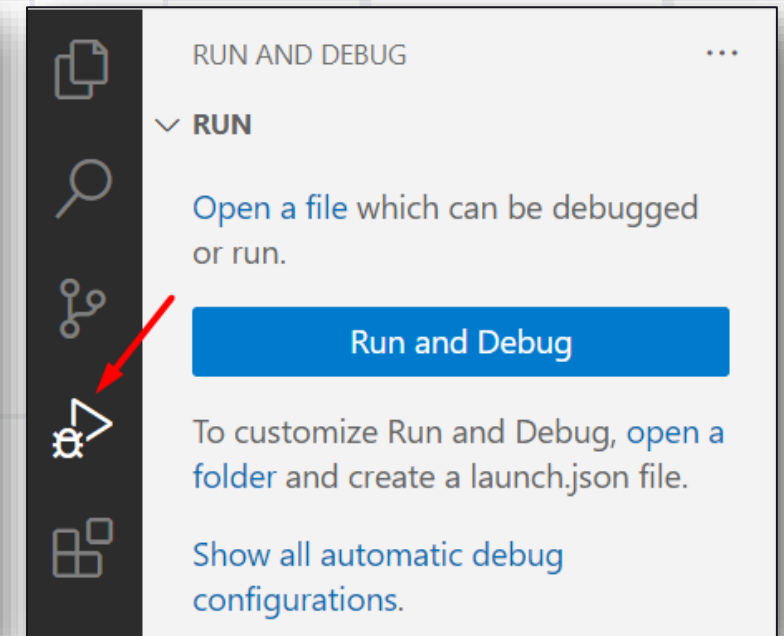
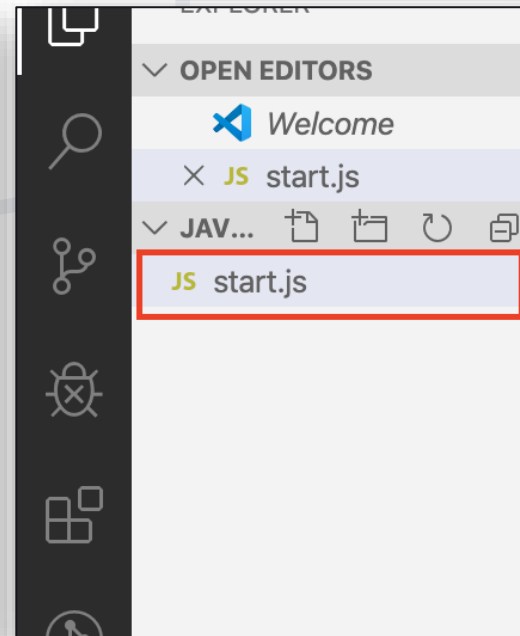
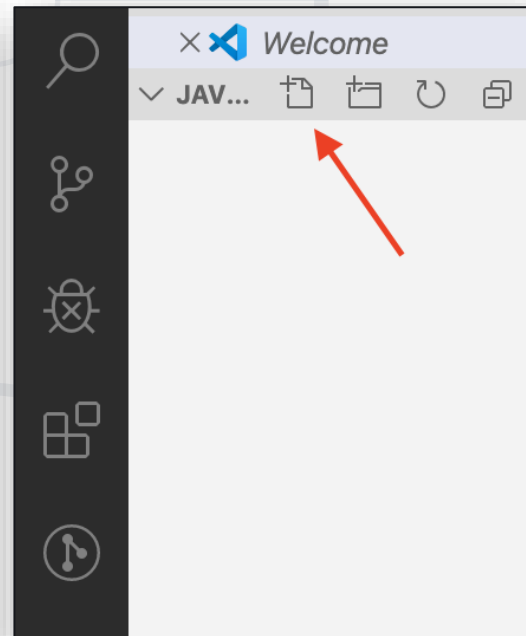
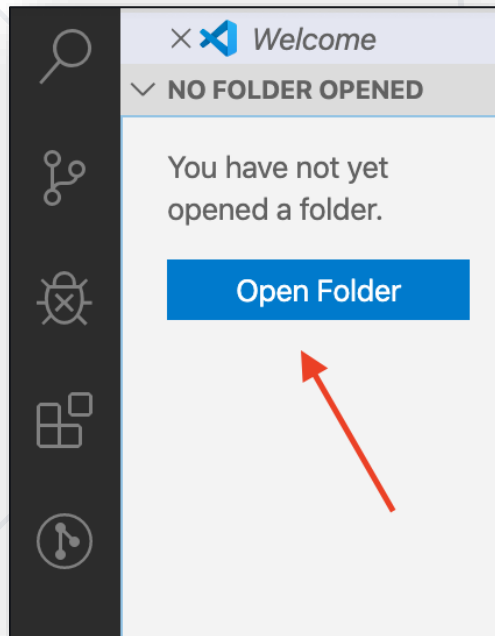
What is an IDE? What is VS Code?

- **VS Code (Visual Studio Code)**
 - Universal IDE for JS, Python, Java, HTML, CSS, C# and others
 - <https://code.visualstudio.com>



Using Visual Studio Code

- **Visual Studio Code** is powerful text editor for JavaScript and others
- Creating an running a JS **project**:





HTML & CSS & JavaScript

Writing Simple Front-End Code

What is HTML?

- The **HTML** language describes Web content (Web pages)
 - Text with formatting, images, lists, hyperlinks, tables, forms, etc.
 - Uses **tags** to define **elements** in the Web page

Opening tag

```
<p>
```

```
<b>Document</b> content goes here...
```

```
</p>
```

Element

Closing tag

Document content goes here...

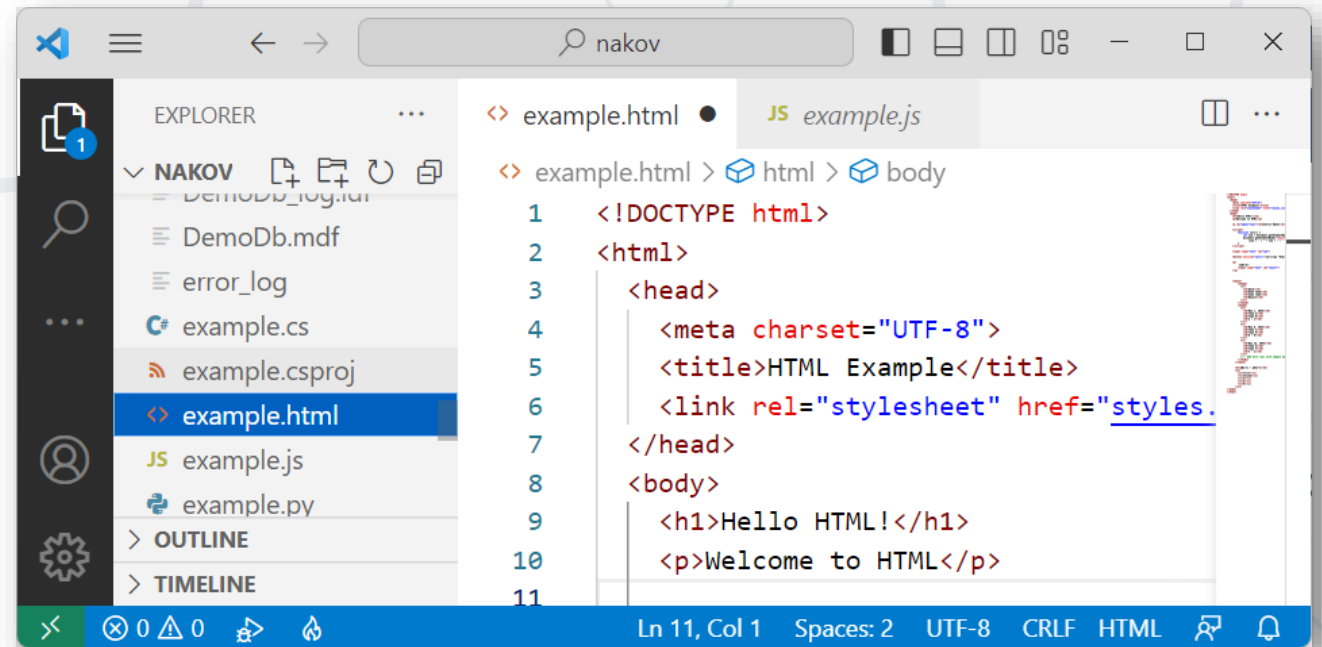
- **Visual Studio Code, Brackets, NetBeans**
 - Good free tools for HTML5, cross-platform

- **WebStorm**

- Powerful IDE for HTML, CSS and JavaScript
 - Paid product

- **Visual Studio**

- Many languages and technologies, Windows & Mac



HTML Page – Example

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Example</title>
  </head>
  <body>
    <h1>Hello HTML!</h1>
  </body>
</html>
```



Problem: Welcome to HTML

- Create your first HTML page
 - File name: **welcome.html**
 - Title: Welcome
 - Paragraph of text:
I am learning **HTML** and **CSS**!
- Hints:
 - Modify the code from the previous slide, use **** tag



- **CSS** defines styling of the HTML elements
 - Specifies fonts, colors, margins, sizes, positioning, floating, ...
 - CSS rules format: **selector { prop1:val1; prop2:val2; ... }**
- CSS rule example:

Selector

```
h1 {
```

```
font-size: 42px;
```

```
color: yellow;
```

```
}
```

Property

Value

Declaration

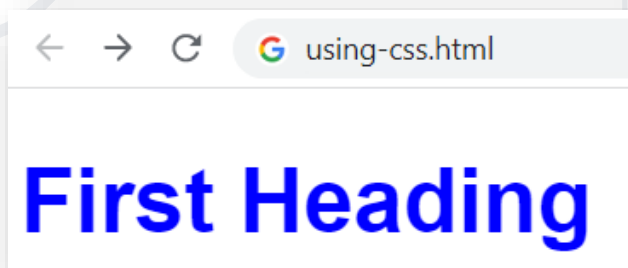
Combining HTML and CSS Files (External Style)

using-css.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet"
          href="styles.css">
  </head>
  <body> <h1>First Heading</h1>
  </body>
</html>
```

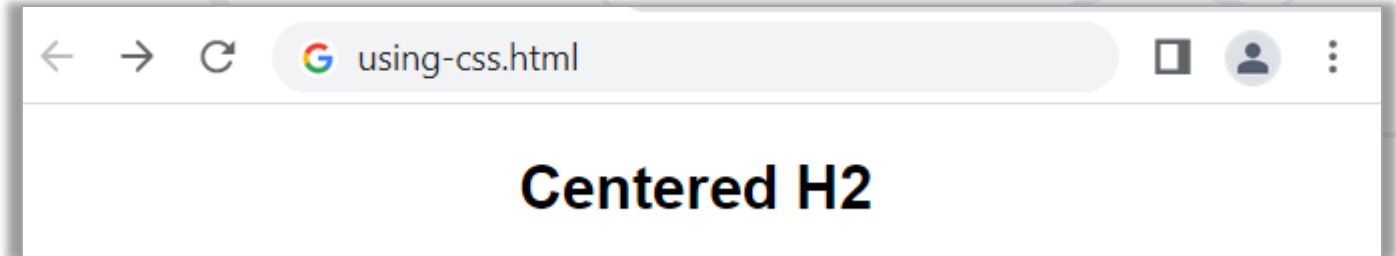
styles.css

```
/* CSS here */
h1 {
  font-size: 42px;
  color: blue;
}
```



- Uses the HTML class attribute, and is defined with a "."

```
.centered {  
  text-align: center;  
}
```



```
<h2 class="centered">Centered H2</h2>
```

- Style all paragraphs, having class "right"

Right-aligned
paragraph of text

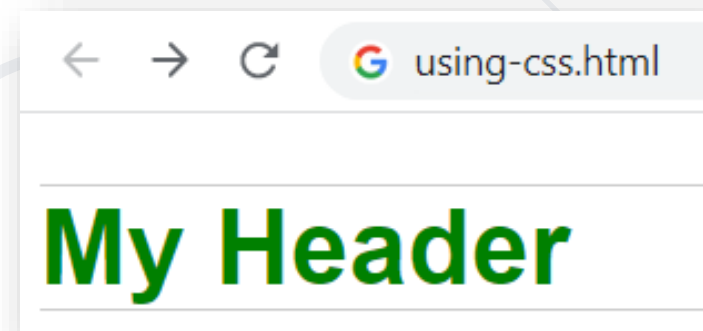
```
p.right {  
  text-align: right;  
}
```

```
<p class="right">Right-aligned  
paragraph of text</h2>
```

- The **#id** selector styles the element with the specified **id**:

```
<h1 id="header">My Header</h1>
```

```
#header {  
  color: green;  
  border: 1px solid #CCC;  
  border-width: 1px 0;  
}
```



- Put a `<style>` element in the HTML `<head>` section

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .red {
      color:red;
    }
  </style>
</head>
</html>
```

```
<body>
  <p class="red">This is red</p>
</body>
```



- The **style** attribute in HTML elements

Attribute "style"

```
<h1 style="color:blue">This is a blue heading</h1>
```

Property

Value



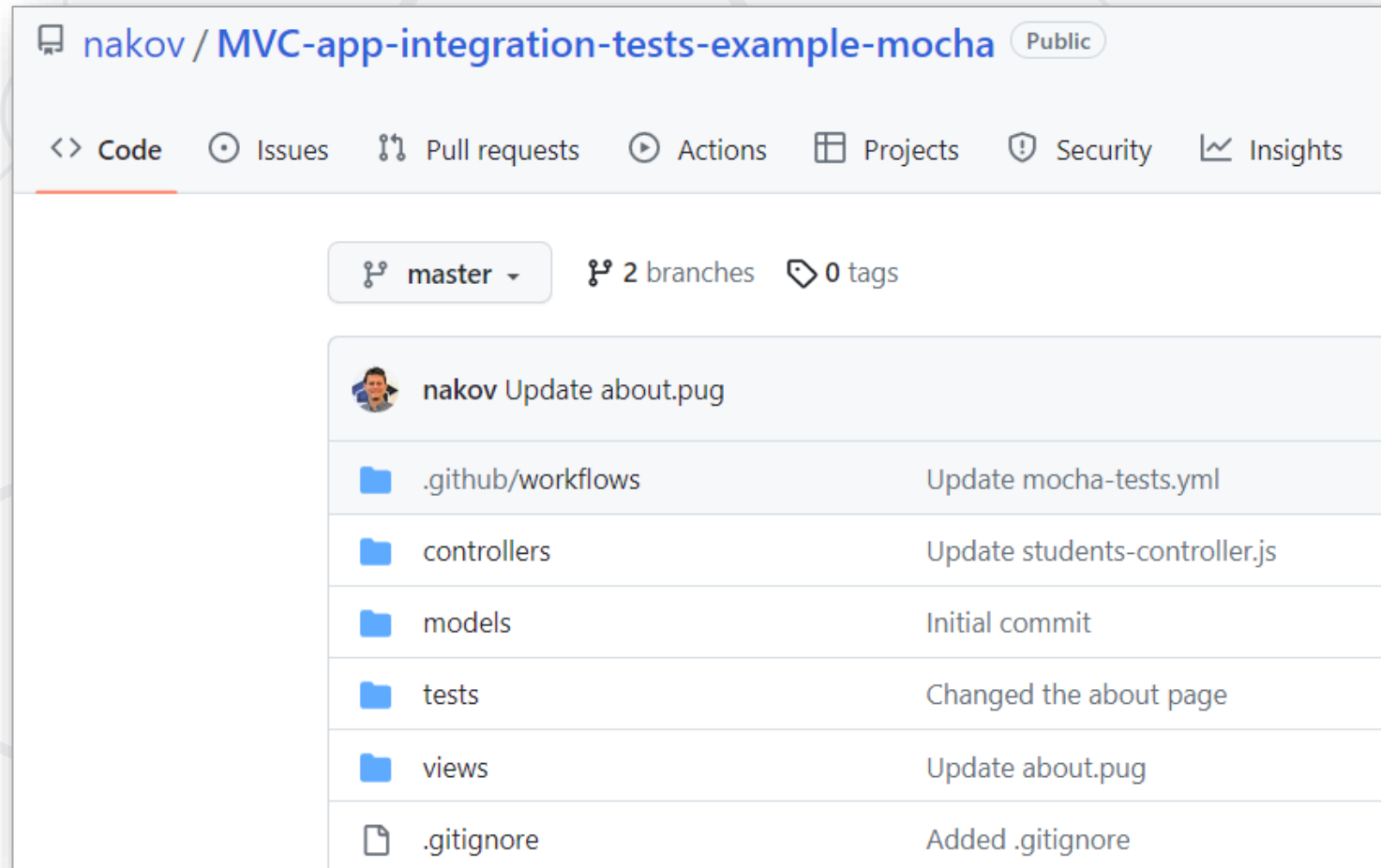


Real Software Project

Explore, Test and Run a Node.js Web App

Explore Project @ GitHub Repository

- Link to project: [nakov/MVC-app-integration-tests-example-mocha](https://github.com/nakov/MVC-app-integration-tests-example-mocha)



Running the Sample JS App

```
git clone https://github.com/nakov/MVC-app-integration-tests-example-mocha
```

```
cd MVC-app-integration-tests-example-mocha
```

```
npm install
```

```
npm test
```

```
npm start
```

```
Home page
  ✓ Page title (45ms)
  ✓ Students count (38ms)

View Students page
  ✓ Page title (40ms)
  ✓ Students list (38ms)

9 passing (1s)

[node1] (local) root@192.168.0.13 ~/MVC-app-integration-tests-example-mocha
$ npm start

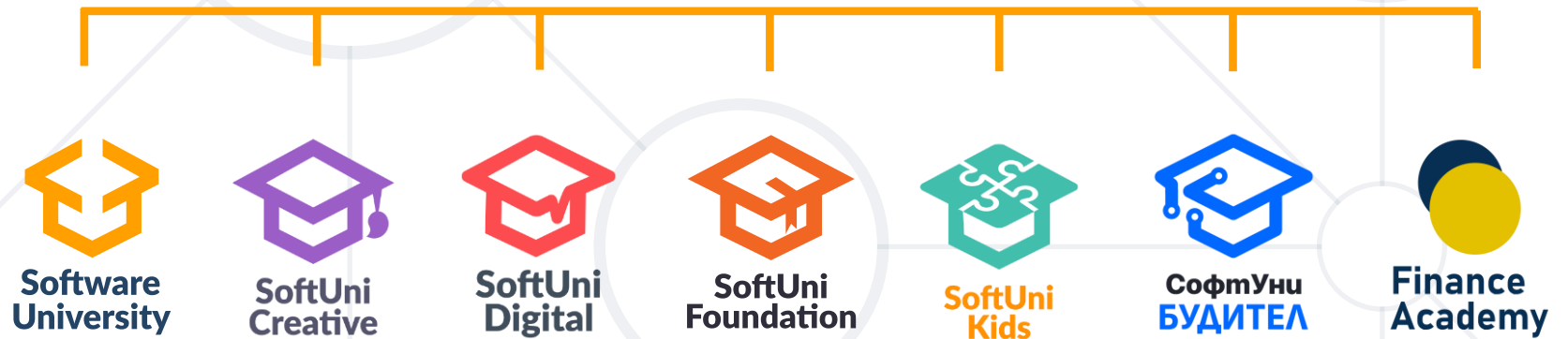
> start
> node index.js

App started. Listening at http://localhost:8080
```

- **Programming languages:** JS, Python, Java, C#
 - **Low-level & high-level** languages
 - **Scripting** vs. **compiled** languages
- Most popular programming languages: JavaScript, Python, Java, C#, C++, PHP
- IDE setup: Visual Studio Code for JS coding
- **HTML, CSS** and **JavaScript** – front-end coding
- Running a real-world Node.js software project



Questions?



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity



Software University



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

