JavaScript-1

Nesli Erdogmus

neslierdogmus@iyte.edu.tr

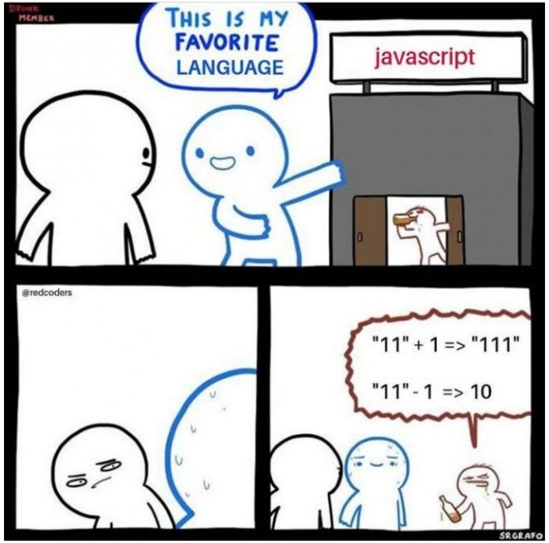
Introduction

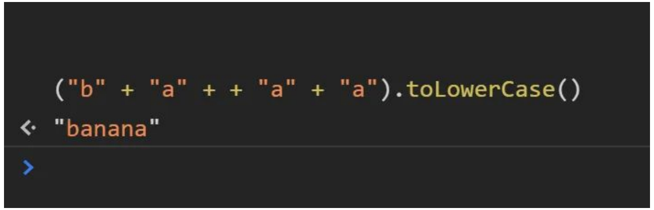
- **These slides assume that you are already familiar with the Python programming language.**

- JavaScript was invented in 1995 by Brendan Eich (who was with Netscape at the time) specifically as a programming language for the browser.
  - Originally code named “Mocha” the first version of JavaScript was written in just 10 days.
  - It was released largely unchanged as LiveScript and later renamed to JavaScript.
  - Today the language is controlled by Ecma International with various vendors providing their own implementation of the standardized language known as ECMAScript.
  - ECMAScript 2022, the 13th and current version, was released in June 2022.

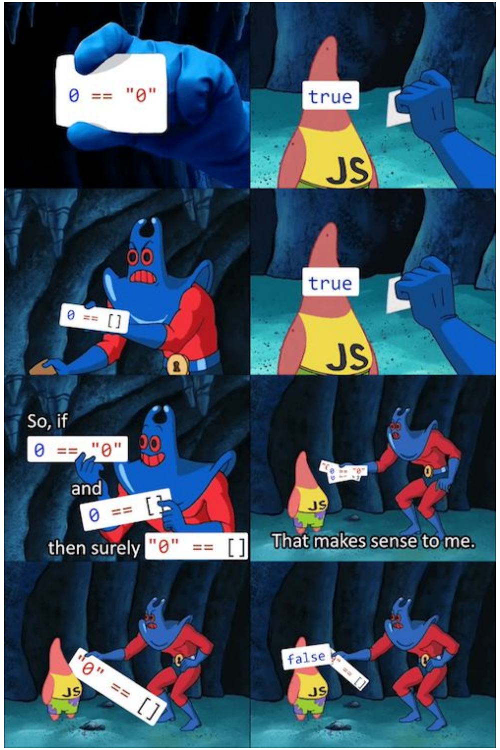
Introduction

- “JavaScript is ridiculously liberal in what it allows.”
  - Eloquent JavaScript, Marijn Haverbeke.





```
("b" + "a" + + "a" + "a").toLowerCase()
<· "banana"
>
```



Introduction

- Client-side development
  - Also referred to as front-end development, it is a type of development that involves programs that run on a client's or user's device.
  - Client-side developers focus on creating the part of a website with which the user interacts.
  - HTML, CSS, JS
- Server-side development
  - Also called back-end development, it is a type of development that involves programs that run on a server.
  - Web browsers, or clients, interact with web servers to retrieve information.
  - Java, Python, SQL, PHP and recently also JS!

Introduction

- **Client-side development**
  - Also referred to as front-end development, it is a type of development that involves programs that run on a client's or user's device.
  - Client-side developers focus on creating the part of a website with which the user interacts.
  - HTML, CSS, JS
- Server-side development
  - Also called back-end development, it is a type of development that involves programs that run on a server.
  - Web browsers, or clients, interact with web servers to retrieve information.
  - Java, Python, SQL, PHP and recently also JS!

Introduction

- The core client-side JavaScript language consists of common programming features.
- So-called Application Programming Interfaces (APIs) provide us with extra superpowers to use in our JavaScript code.
    - APIs are ready-made sets of code building blocks that allow a developer to implement programs that would otherwise be hard or impossible to implement.

APIs

- **Browser APIs** are built into your web browser, and are able to expose data from the surrounding computer environment, or do useful complex things. For example:
    - The DOM (Document Object Model) API allows you to manipulate HTML and CSS, creating, removing and changing HTML, dynamically applying new styles to your page, etc.
    - The Geolocation API retrieves geographical information.
    - The Canvas and WebGL APIs allow you to create animated 2D and 3D graphics.

APIs

- **Third party APIs** are not built into the browser by default, and you generally have to grab their code and information from somewhere on the Web. For example:
    - The Twitter API allows you to do things like displaying your latest tweets on your website.
    - The Google Maps API and OpenStreetMap API allows you to embed custom maps into your website, and other such functionality.

Language characteristics

- Interpreted versus compiled code
  - JavaScript is a lightweight interpreted programming language.
- Dynamic versus static code
  - JavaScript dynamically generates new content inside the browser on the client, e.g. creating a new HTML table, filling it with data requested from the server, then displaying the table in a web page shown to the user.

Inserting JavaScript

- There are three common ways of inserting JavaScript code in web page:
    - Inside an HTML tag script
    - In an external file
    - As a value of some HTML attributes

Inserting JavaScript

- There are three common ways of inserting JavaScript code in web page:
    - Internally, inside an HTML tag script

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: red;
      }
    </style>
    <script>
      // JavaScript goes here
    <script>
  </head>
  <body>
    <h1>Hello</h1>
    <p>It’s me...</p>
  </body>
</html>
```

Inserting JavaScript

- There are three common ways of inserting JavaScript code in web page:
    - In an external file

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: red;
      }
    </style>
      <script src="script.js" defer>
    <script>
  </head>
  <body>
    <h1>Hello</h1>
    <p>It’s me...</p>
  </body>
</html>
```

Inserting JavaScript

- There are three common ways of inserting JavaScript code in web page:
  - As a value of some HTML attributes

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: red;
      }
  </head>
  <body>
    <h1 onclick="createParagraph()>Hello</h1>
    <p>It’s me...</p>
  </body>
  <script>
      // Define createParagraph here
  <script>
</html>
```

Inserting JavaScript

- There are three common ways of inserting JavaScript code in web page:
    - As a value of some HTML attributes

      DO NOT USE inline JavaScript handlers. It is bad practice to pollute your HTML with JavaScript, and it is inefficient

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      p {
        color: red;
      }
  </head>
  <body>
    <h1 onclick="createParagraph()">Hello</h1>
    <p>It’s me...</p>
  </body>
  <script>
      // Define createParagraph here
  <script>
</html>
```

Event-driven programming

- The flow of the program is determined by events such as user interactions, sensor outputs, or messages from other programs or threads.
- Asynchronous programming allows for non-blocking code execution, meaning that code can be written without having to wait for certain tasks to be completed before continuing.
  - You can set up an **event listener** that will trigger a **callback function**. This callback function can handle the event, allowing for asynchronous code execution.
- Instead of including JavaScript in our HTML, we should use a pure JavaScript construct.

Examples

- Let’s use an online HTML editor to implement the examples on the following link:

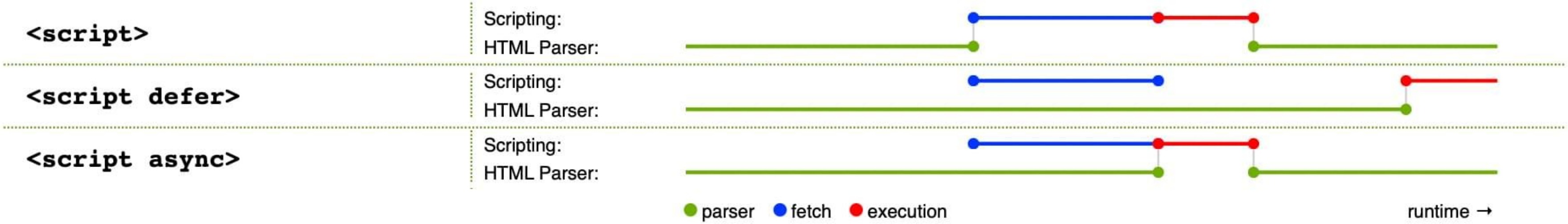
  https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript#how_do_you_add_javascript_to_your_page

Script loading strategies

- All the HTML on a page is loaded in the order in which it appears.
    - If you are using JavaScript to manipulate elements on the page (or more accurately, the Document Object Model), your code won't work if the JavaScript is loaded and parsed before the HTML you are trying to do something to.
- Previously on the internal JS example - an event listener, which listens for the browser's `DOMContentLoaded` event is used.
- Previously on the external JS example - the `defer` attribute, which tells the browser to continue downloading the HTML content once the `<script>` tag element has been reached is used.
- An **old-fashioned** solution to this problem used to be to put your script element right at the bottom of the body (e.g. just before the </body> tag).

Script loading strategies

- async and defer
  - With `async`, we get no guarantee that scripts will run in any specific order. It is best to use `async` when the scripts in the page run independently from each other and depend on no other script on the page.
  - Scripts loaded with the `defer` attribute will load in the order they appear on the page. They won't run until the page content has all loaded, which is useful if your scripts depend on the DOM being in place (e.g. they modify one or more elements on the page).



Exercise

- I want you to create a simple guess the number type game. It should choose a random number between 1 and 100, then challenge the player to guess the number in 10 turns. After each turn, the player should be told if they are right or wrong, and if they are wrong, whether the guess was too low or too high. It should also tell the player what numbers they previously guessed. The game will end once the player guesses correctly, or once they run out of turns. When the game ends, the player should be given an option to start playing again.

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash

Python vs JavaScript

In Python, the scope of a variable in a function is determined by assignment in that function or the innermost enclosing function definition in which it is assigned, unless this behavior is opted out of with the `global` or `nonlocal` keywords.

```
x = 1
def local_variable():
    x = 2
    print(x)

local_variable()  # prints 2
print(x)  # prints 1

def reference_global_variable():
    print(x)

reference_global_variable()  # prints 1

def modify_global_variable():
    global x
    x = 3
    print(x)

modify_global_variable()  # prints 3
print(x)  # prints 3
```

JavaScript uses variable declarations instead: if you want a local variable, you should use let (for bindings that can be reassigned) or const (for bindings that cannot).

```
let x = 1;
const localVariable = () => {
  const x = 2;
  console.log(x);
}
localVariable(); // logs 2
console.log(x) // logs 1

const referenceGlobalVariable = () => {
  console.log(x);
}
referenceGlobalVariable(); // logs 1

const modifyGlobalVariable = () => {

  x = 3;
  console.log(x);
}
modifyGlobalVariable(); // logs 3
console.log(x) // logs 3
```

Python vs JavaScript

- Python uses indentation to indicate a block of code.
- Python variables use function scope. Loop variables "leak" into the scope of the function.

- JavaScript uses brackets instead of indentation; the indentation of code should match the brackets, but it's a style convention.
- JavaScript's let and const variables declarations use block scope: bindings only apply to the current block and enclosed blocks, not the entire function.
- With let and const, functions declared in a loop will each close over their own loop variable!

Python vs JavaScript

```
# This will work:

if (true)
  a = 1

print(a);

// Try function declaration in a loop

functions = [];
for i in [1, 2, 3]:
    def print_num():
        print(i)

    functions.append(print_num)


functions[0]()  # prints 3
functions[1]()  # prints 3
functions[2]()  # prints 3
```

```
// This won't work:

if (true) {
  const a = 1;
}
console.log(a);

// Try function declaration in a loop

const functions = [];
for (const i of [1, 2, 3]) {
  const printNum = () => {
    console.log(i);
  }
  functions.push(printNum);
}

functions[0](); // logs 1
functions[1](); // logs 2
functions[2](); // logs 3
```

Python vs JavaScript

Python has **triple-quoted f-strings**.

```
s = f"""This is a Python triple-quoted
f-string.
It's similar to a template string in
JavaScript.
Code goes in here: {1 + 1}"""
```

JavaScript has **template literals**.

```
s = `This is a JavaScript template literal.
It's similar to a multi-line f-string in
Python.
Code goes in here: ${1 + 1}.`
```

A JavaScript **tagged template literal** is a template literal with a function tacked onto the front:

```
const x = 12;
const s = capitalize`my favorite numbers
are ${x} and ${42}.`;
```

```
const x = 12;
const s = capitalize(["my favorite numbers
are ", " and ", "."], x, 42);
```

Python vs JavaScript

- In Python, there are three distinct numeric types: integers, floating point numbers, and complex numbers.

- Most numbers you encounter in JavaScript are floats.
- Unless you're using the still-unusual BigInt, every number is what you'd call a 'double' in NumPy: a double-precision 64-bit floating point number.
    - Even when indexing into arrays!