A Primer on Back-end JavaScript

Luca Mottola
`luca.mottola@polimi.it`

(version 0.1)

JavaScript

- Why?
  - Initially created to “make web pages alive”
  - Fully integrated with HTML and CSS
  - Rapidly gained adoption as web apps proliferated
- In JavaScript, “simple things are done simply”
- Today, not just web pages, but mobile applications and backend processing as well!
- Our interest: we use JavaScript for developing IoT applications
  - Programs outside of a normal browser

- Note: JavaScript has no relation to Java

Outline

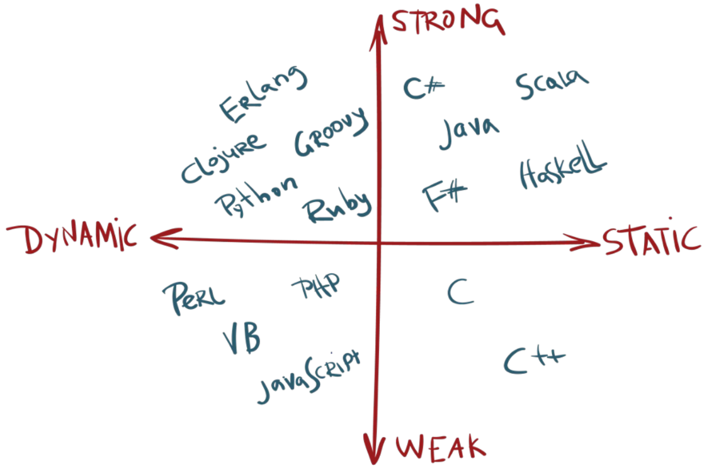
- Basics
- Functions and objects

Basics

Key Features (1/2)

- JavaScript is **interpreted**
  - Programs run on top of an **engine** without directly translation to machine code
  - Advantages: flexibility, ease of modification, …
  - Disadvantages: speed and (run-time) debugging
  - Other examples: Python, Ruby, …
- Most efficient engines are in **browsers**
  - V8 in Chrome, Opera, and Edge
  - SpiderMonkey in Firefox
  - JavaScriptCore in Safari
- Engines provide **protection** and **sandboxing**

Key Features (2/2)



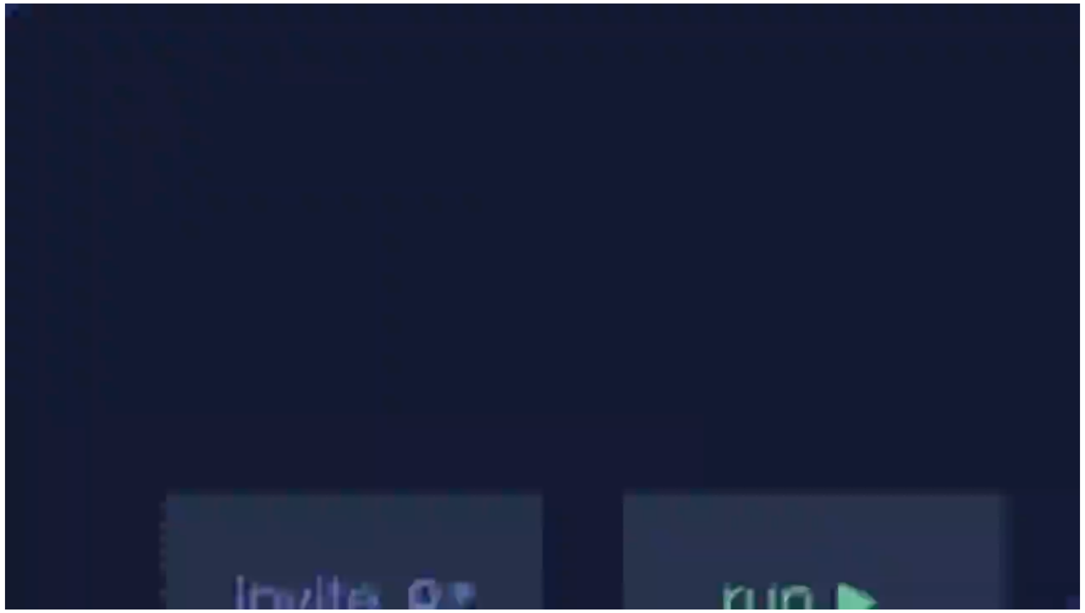
- JavaScript is **dynamically typed**
  - Variable declaration does **not** define the type
  - Types are inferred by the interpreter based on value
  - Variables may **change their type** as the program executes
  - Type checking **can only happen at run-time**, leaving bugs undiscovered
- Orthogonal dimension: **strongly typed** vs. **weakly typed** languages
  - Determines how "strict" are the type checks

Our Engine: Node.js

- Node.js is an open-source cross-platform environment for back-end JavaScript
  - Uses the **V8 engine** as Chrome
  - Offers a rich set of **modules for networking and I/O**
  - Is based on **asynchronous event-driven** programming
  - Can be extended with **additional packages** using `npm`

Tools

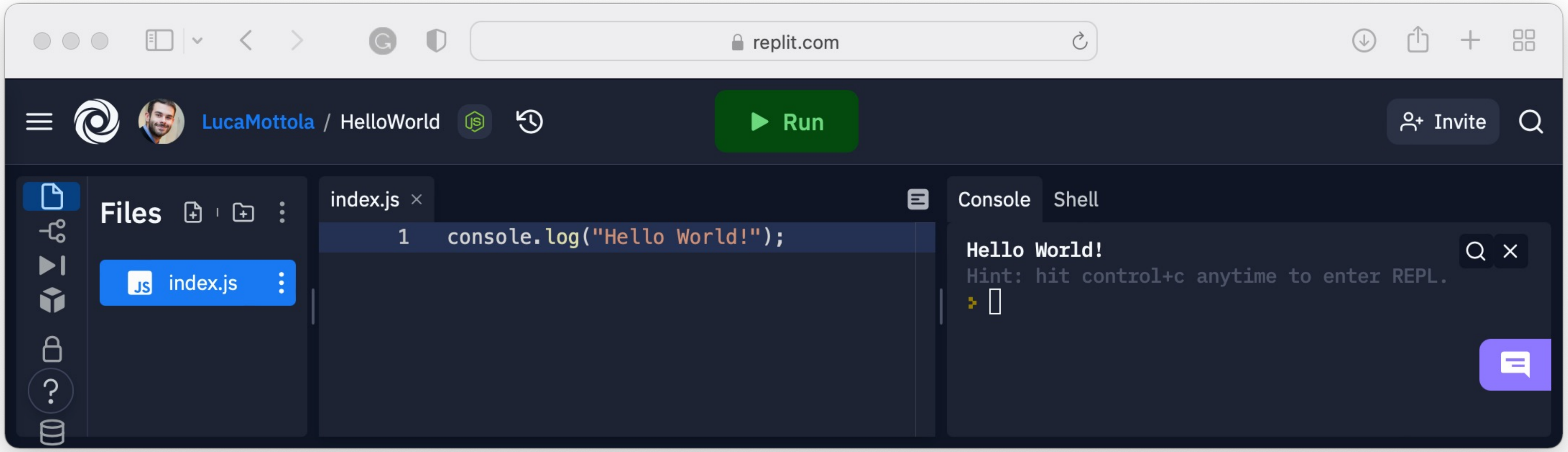
- Replit is a **browser-based IDE** supporting a variety of languages
  - Can be used for simple and complex projects
  - Has version control integration and tools for teamwork
- Note: we will work in a browser, but develop back-end JavaScript code running on Node.js
  - This is just the easiest option to start quickly



Our First JavaScript Program

```
console.log("Hello World!");
```

Using Node.js, the **log** method of the **console** class prints something out on the console



```
console.log("Hello World!");
```

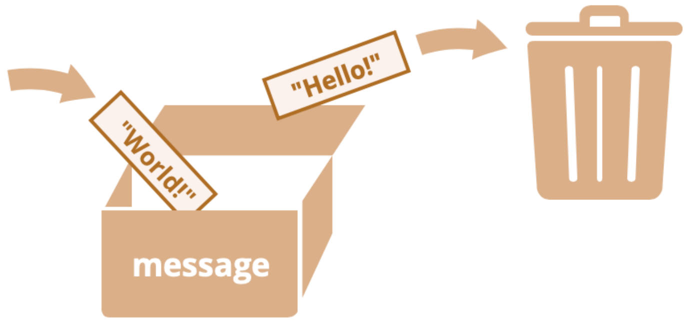
```
Hello World!
Hint: hit control+c anytime to enter REPL.
›
```

Variables and Constants

- **Variables** are named storage for data
  - Like in any other language…
  - They are initialized, read and written
  - Declaring twice triggers an error
  - Case matters, symbol $ and _ are legal
- **Constants** use **`const`** instead of **`let`**

"Hello!"
message

```
let message = ”Hello!”;
console.log(message);
```



```
let message = ”Hello!”;
message = ”World!”;
console.log(message);
```

Types and Values

- The data types are
  - Number (integer and float)
  - BigInt
  - String
  - Boolean
  - Objects (more on this later)
- The **`null`** value simply means "nothing"
  - Not a reference to a non-existing object (Java) or a null pointer (C)
  - The **`undefined`** value indicates a non-initialized variable
- The **`typeof`** operator returns the variable type
- Type conversion works as usual:
  - Example: **`let num = Number(string);`**

```
let message = ”Hello!”;
console.log(message);
console.log(typeof message);
message = 987;
console.log(message);
console.log(typeof message);
```

Variable **`message`** starts as a String, then becomes a Number type!

User Interaction

- To ask for user input from the command line, use **`prompt()`**
  - An optional message may be included, too..

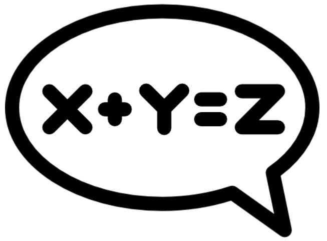
Variable **`message`** gets the user input after pressing return!

```
let message = ”Hello!”;
console.log(message);
message = prompt (”What is your name?”);
console.log(”Hello ” + message + ”!” );
```

Math Operators

- Math operations are **always safe**
  - With the usual precedence rules
  - We get **`NaN`** in the worst case
- Conversions work as expected
  - Conversion to String takes precedence
- Modify-and-assign and increment/decrement operators exist for arithmetic and bitwise operations

Comparisons



- All usual comparator are available (>, <, =>, <=, ==, …)
  - The result is of type Boolean
- String comparison works lexicographically
  - All following comparisons are **`true`**

    ```
    "Z" > "A"
    "Glow" > "Glee"
    "Bee" > "Be"
    ```
- When comparing different types, everything is converted to Number
  - This leads to funny consequences… make sure what type you are comparing with what else!
  - The value **`undefined`** cannot be compared to anything, yields **`false`**

A Complete Example

Variable **`message`** gets the user input after pressing return!

```
let message = ”Hello!”;
console.log(message);
message = prompt (”What is your name?”);
console.log(”Hello ” + message + ”!” );
```

Branching

```
if (date.getMonth()==0) {
  console.log("January");
} else if (date.getMonth()==1) {
  console.log("February");
} else {
  console.log("March and later..");
}
```

Curly brackets are not mandatory, but highly recommender anyways!

- Branching with **`if`** works the usual way
  - Logical operators too!
  - The **`and`** (**`or`**) operator evaluates to the first **`false`** (**`true`**) value, or the last one
- Consider the type conversion rules
  - **`0`**, **`""`**, **`null`**, **`undefined`**, and **`NaN`** all become **`false`**
  - Everything else becomes **`true`**
- Also **`switch`** is available..

Loops

```
for (let i=0; i<10; i++) {
  console.log(i);
  if (i==8) break;
}
```

Effectively executes only nine iterations!

- Usual loop operators **`while`**..., **`do`**...**`while`**, and **`for`** exist
  - Loops maybe broken with **`break`**
  - The rest of the current iteration may be skipped with **`continue`**