Lesson - 13

Introduction to JavaScript

SkillUp by Dobrea Vladislav

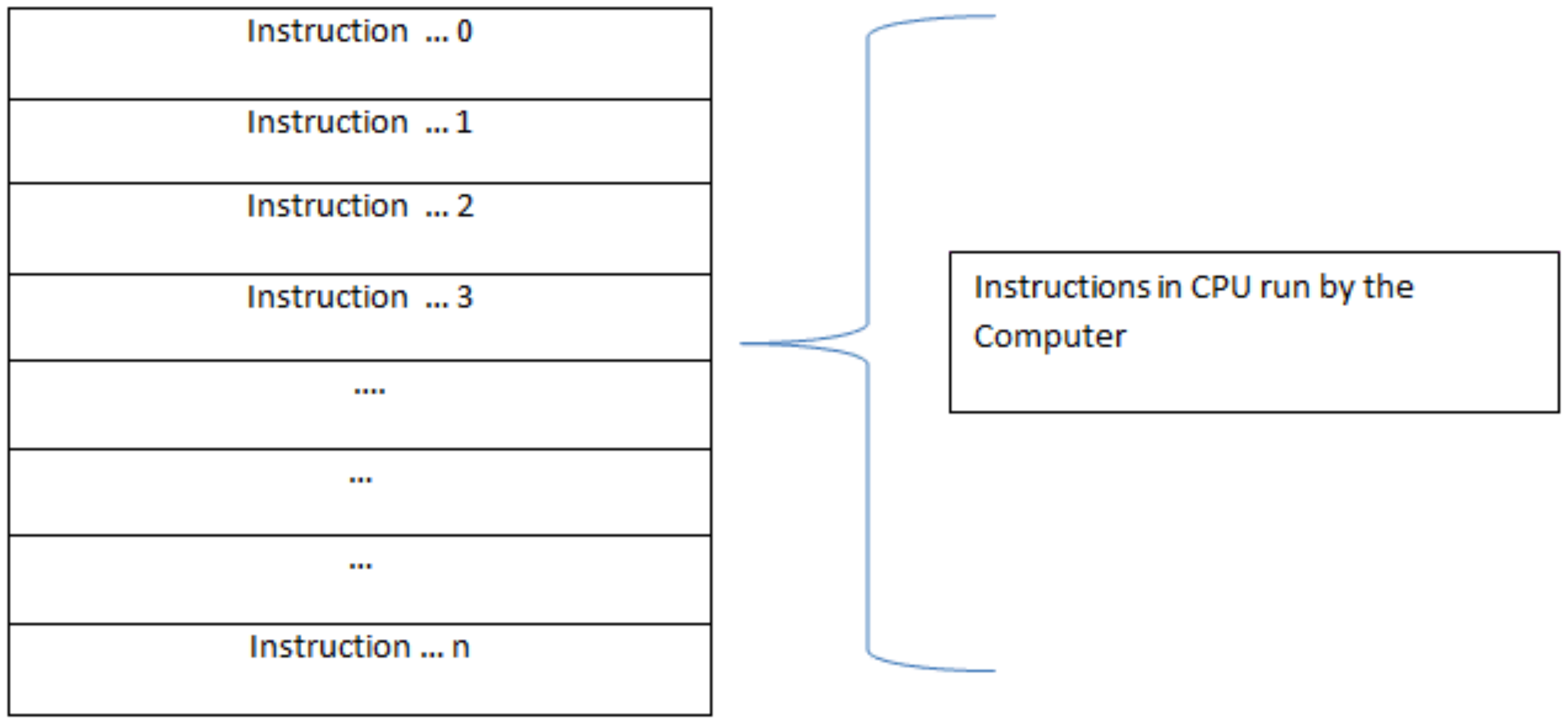
Lesson Plan

- What is program
- About JavaScript
- Variables
- Data types
- Code Structure
- Script files

What is program?

A program is a set of instructions that the computer executes.

(To be more precise, a program is a set of instructions loaded in the CPU that the CPU executes to achieve an outcome).



A Program is mainly a set of instructions

Let’s begin with a little math

Here is a simple equation
Z = X + Y
Suppose, the value of X is 100 and Y = 200, then
we know Z = 300.

Now here comes the twist of the programming world which shatters down the known world of an aspiring beginner.

What if we change the equation to: **Y = X + Y**

Believe it or not, this is a valid expression in almost all programming languages.

A programming language is a language which the computer understands. It is just like any other language with its own syntax and constructs. The programmer uses it to make the computer follow his/her instructions.

What we need to run a program?

The two major components to execute a program are
- *CPU*
- M*emory (RAM)*

CPU (Central Processing Unit) runs a set of instructions.
RAM (Random Access Memory) works as a temporary storage to help the CPU achieve the desired output/outcome.

Why we are discussing these boring hardware stuff before moving on to programming? To make you a better programmer by getting your fundamentals right.

How to solve equation?

Let me go back to the famous expression: **Z = X + Y**

Suppose we know **X = 100, Y = 200** and we ask the computer to calculate the value of **X + Y**

Here is how we achieve it:

Since we have three **variables**, we need to **declare** them in our program.
Most good programming languages requires that you declare the variables explicitly.

Therefore we write

```
var X
var Y
var Z
```

This is called variable declaration.

Since we know X is 100 and Y is 200, we write:

```
X = 100
Y = 200
```

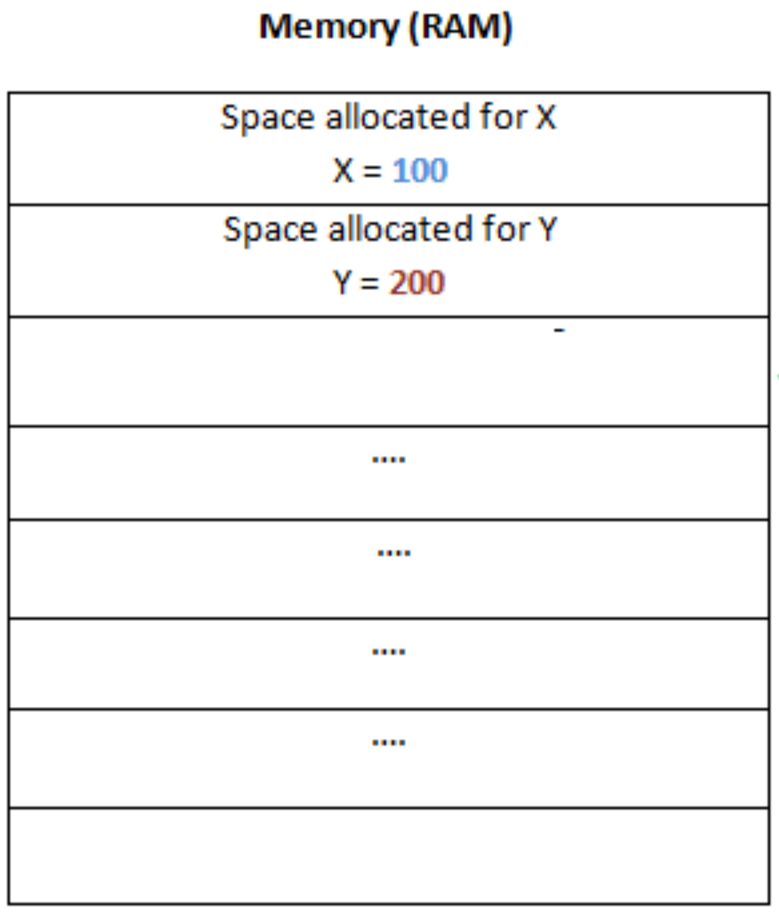
This is called variable assignment.

Now it is very important to understand what goes on behind the scene, otherwise the mind blowing stuff we will be dealing with later (Y = X + Y) would be hard to make sense of.

Behind Scenes

The moment we write var X or var Y, a space is allocated in RAM because unlike mathematical variables which exist mainly in our brains, the variables of a program resides in an actual physical location.

Here is a conceptual diagram for better understanding:



Behind Scenes

X and **Y** are **variables,** and **variables** are like boxes in a room (which is our RAM) and they have the capabilities of holding things that we put in them. We might say Box with label **X** holds value **100** and Box with label **Y** holds value **200** because we had assigned the values through the assignment operator '=' to them.

To calculate the value of **X + Y**, we could simply write: **X + Y**

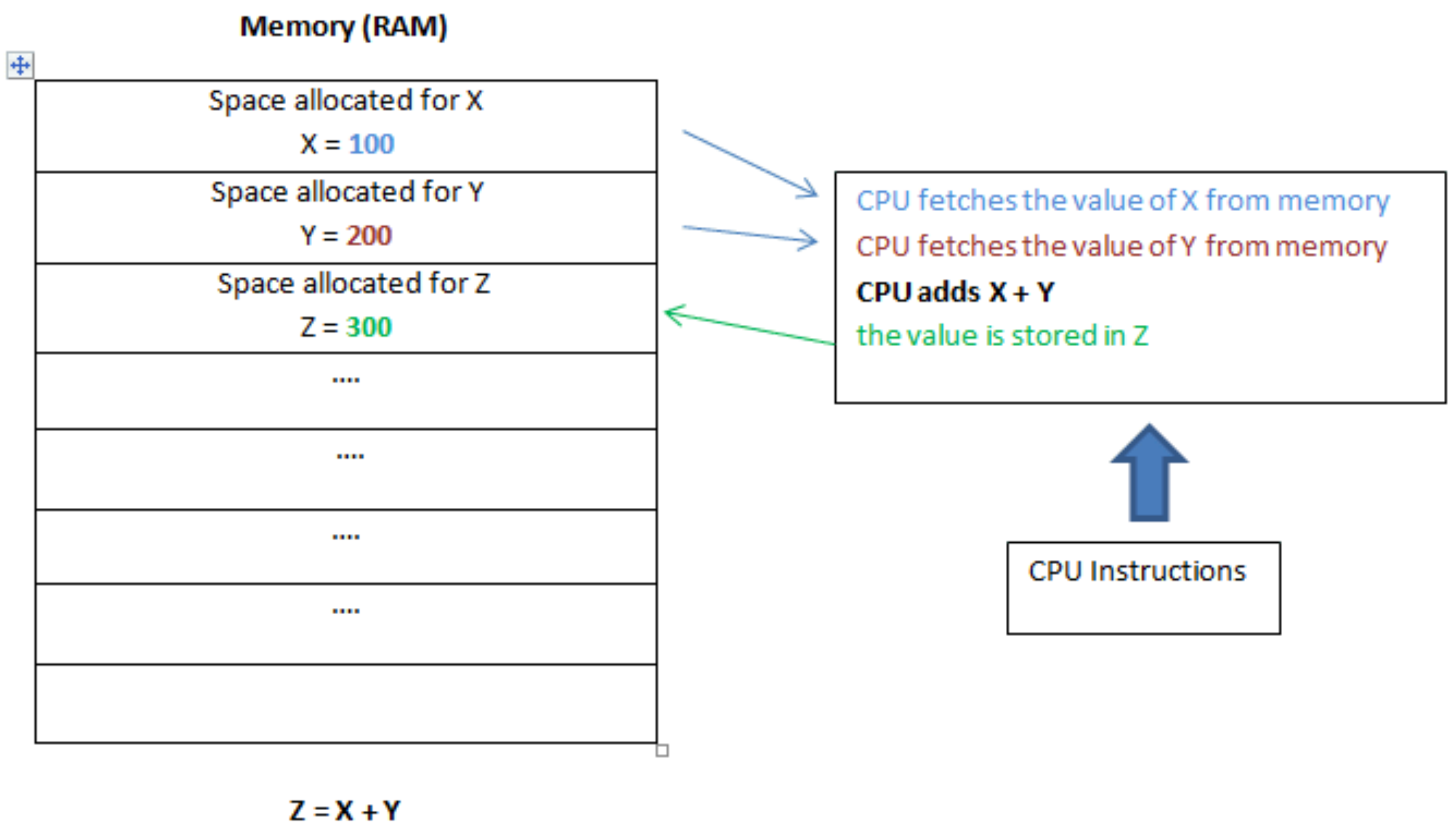
However, doing this operation (every mathematical operation takes place in the CPU) and not storing the result somewhere would mean the result of this operation will be lost forever.

Storing the result

Therefore we need to store the operation result somewhere and luckily we have enough room in our RAM to make provision for Z which will hold the value of the X + Y result.

Therefore, when we write: **Z = X + Y**

we are telling the computer to fetch the value of X and Y from RAM, add them in the CPU and then store the result in another space called Z.

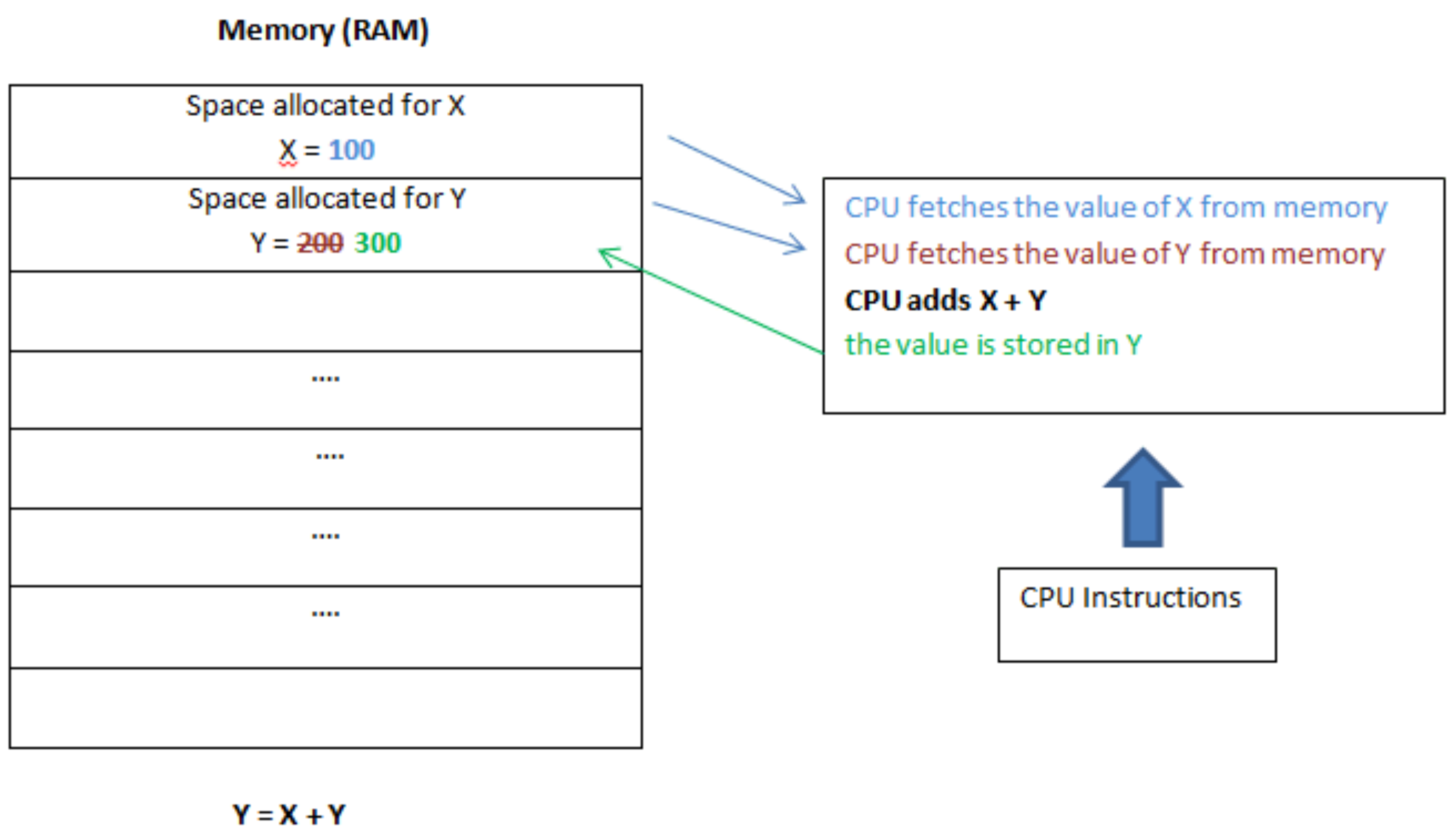


Y = X + Y

Now the expression Y = X + Y should make some sense now.

What we are doing here is, fetch values of X and Y from RAM, add them in CPU and put the value back into Y.

Most computer programming languages are best read from right to left. **We just have to remember everything on the right side of an assignment operator ('=') is processed first and then the assignment happens.**



What is JavaScript?

JavaScript was initially created to “make web pages alive”.

The programs in this language are called *scripts*. They can be written right in a web page’s HTML and run automatically as the page loads.

Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.

In this aspect, JavaScript is very different from another language called Java.

Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.

The browser has an embedded engine sometimes called a “JavaScript virtual machine”.

Different engines have different “codenames”. For example:

- V8 – in Chrome and Opera.
- SpiderMonkey – in Firefox.
- …There are other codenames like “Trident” and “Chakra” for different versions of IE, “ChakraCore” for Microsoft Edge, “Nitro” and “SquirrelFish” for Safari, etc.

In browser JavaScript

Modern JavaScript is a “safe” programming language. It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it.

JavaScript’s capabilities greatly depend on the environment it’s running in. For instance, Node.js supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.

In-browser JavaScript can do everything related to webpage manipulation, interaction with the user, and the webserver.

For instance, in-browser JavaScript is able to:

- Add new HTML to the page, change the existing content, modify styles.
- React to user actions, run on mouse clicks, pointer movements, key presses.
- Send requests over the network to remote servers, download and upload files (so-called AJAX and COMET technologies).
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side (“local storage”).

What makes it unique?

There are at least *three* great things about JavaScript:

- Full integration with HTML/CSS.
- Simple things are done simply.
- Support by all major browsers and enabled by default.

What is Specification?

The ECMA-262 specification contains the most in-depth, detailed and formalized information about JavaScript. It defines the language.

But being that formalized, it’s difficult to understand at first. So if you need the most trustworthy source of information about the language details, the specification is the right place. But it’s not for everyday use.

A new specification version is released every year. In-between these releases, the latest specification draft is at https://tc39.es/ecma262/.

To read about new bleeding-edge features, including those that are “almost standard” (so-called “stage 3”), see proposals at https://github.com/tc39/proposals.

Also, if you’re in developing for the browser, then there are other specs covered in the second part of the tutorial.

Manuals

- **MDN (Mozilla) JavaScript Reference** is a manual with examples and other information. It's great to get in-depth information about individual language functions, methods etc.
  One can find it at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference.
  Although, it's often best to use an internet search instead. Just use "MDN [term]" in the query, e.g. https://google.com/search?q=MDN+parseInt to search for `parseInt` function.

- **MSDN** – Microsoft manual with a lot of information, including JavaScript (often referred to as JScript). If one needs something specific to Internet Explorer, better go there: http://msdn.microsoft.com/.
  Also, we can use an internet search with phrases such as "RegExp MSDN" or "RegExp MSDN jscript".

Variables

Most of the time, a JavaScript application needs to work with information. Here are two examples:

1. An online shop – the information might include goods being sold and a shopping cart.
2. A chat application – the information might include users, messages, and much more.

Variables are used to store this information.

A variable is a “named storage” for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the `let` keyword.

The statement below creates (in other words: *declares*) a variable with the name “message”:

```javascript
let message;
```

Now, we can put some data into it by using the assignment operator `=`:

```javascript
let message;

message = 'Hello'; // store the string
```

A real-life analogy

We can easily grasp the concept of a “variable” if we imagine it as a “box” for data, with a uniquely-named sticker on it.

For instance, the variable `message` can be imagined as a box labeled `"message"` with the value `"Hello!"` in it:

We can put any value in the box.

"Hello!"
message

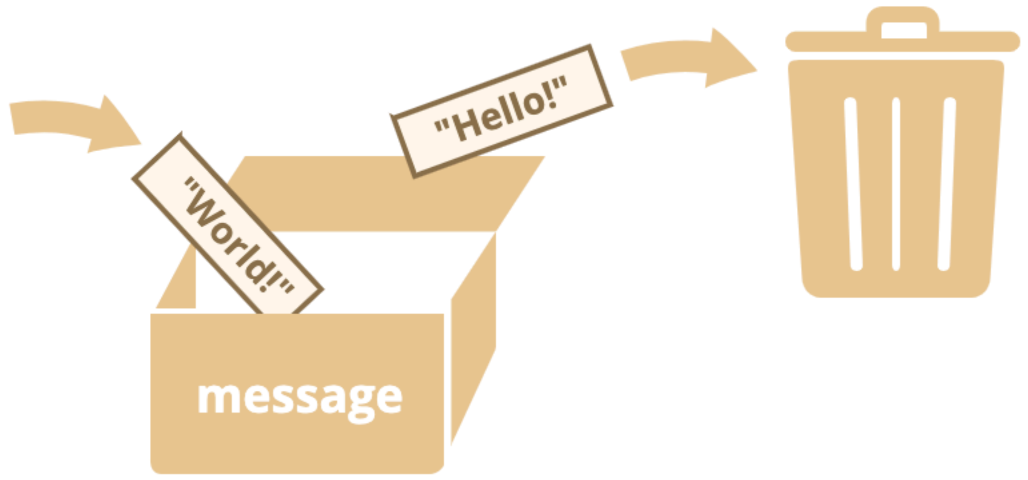
We can also change it as many times as we want:

```
let message;

message = 'Hello!';

message = 'World!'; // value changed

alert(message);
```



Variable naming

"There are 2 hard problems in computer science: cache invalidation, naming things, and off-by-1 errors."

There are two limitations on variable names in JavaScript:

1. The name must contain only letters, digits, or the symbols `$` and `_`.
2. The first character must not be a digit.

```
let userName;
let test123;
```

When the name contains multiple words, camelCase is commonly used. That is: words go one after another, each word except first starting with a capital letter: `myVeryLongName`.

What's interesting – the dollar sign `'$'` and the underscore `'_'` can also be used in names. They are regular symbols, just like letters, without any special meaning.

Examples of **incorrect** variable names:

```
let 1a; // cannot start with a digit

let my-name; // hyphens '-' aren't allowed in the name
```

Variable naming

Additional information to note when naming:

Reserved names
There is a list of reserved words, which cannot be used as variable names because they are used by the language itself.

For example: `let`, `class`, `return`, and `function` are reserved.

Case matters
Variables named `apple` and `AppLE` are two different variables.

Non-Latin letters are allowed, but not recommended
It is possible to use any language, including Cyrillic letters or even hieroglyphs, like this:

```
let имя = '...';
let 我 = '...';
```

Technically, there is no error here, such names are allowed, but there is an international tradition to use English in variable names. Even if we're writing a small script, it may have a long life ahead. People from other countries may need to read it some time.

Constants

To declare a constant (unchanging) variable, use `const` instead of `let`:

```
const myBirthday = '18.04.1982';
```

Variables declared using `const` are called “constants”. They cannot be reassigned. An attempt to do so would cause an error:

```
const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // error, can't reassign the constant!
```

Naming Constants

There is a widespread practice to use constants as aliases for difficult-to-remember values that are known prior to execution.

Such constants are named using capital letters and underscores.

For instance, let's make constants for colors in so-called "web" (hexadecimal) format:

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";

// ...when we need to pick a color
let color = COLOR_ORANGE;
alert(color); // #FF7F00
```

Benefits:

- `COLOR_ORANGE` is much easier to remember than `"#FF7F00"`.
- It is much easier to mistype `"#FF7F00"` than `COLOR_ORANGE`.
- When reading the code, `COLOR_ORANGE` is much more meaningful than `#FF7F00`.

Name things right

Talking about variables, there's one more extremely important thing.

A variable name should have a clean, obvious meaning, describing the data that it stores.

Variable naming is one of the most important and complex skills in programming. A quick glance at variable names can reveal which code was written by a beginner versus an experienced developer.

In a real project, most of the time is spent modifying and extending an existing code base rather than writing something completely separate from scratch. When we return to some code after doing something else for a while, it's much easier to find information that is well-labeled. Or, in other words, when the variables have good names.

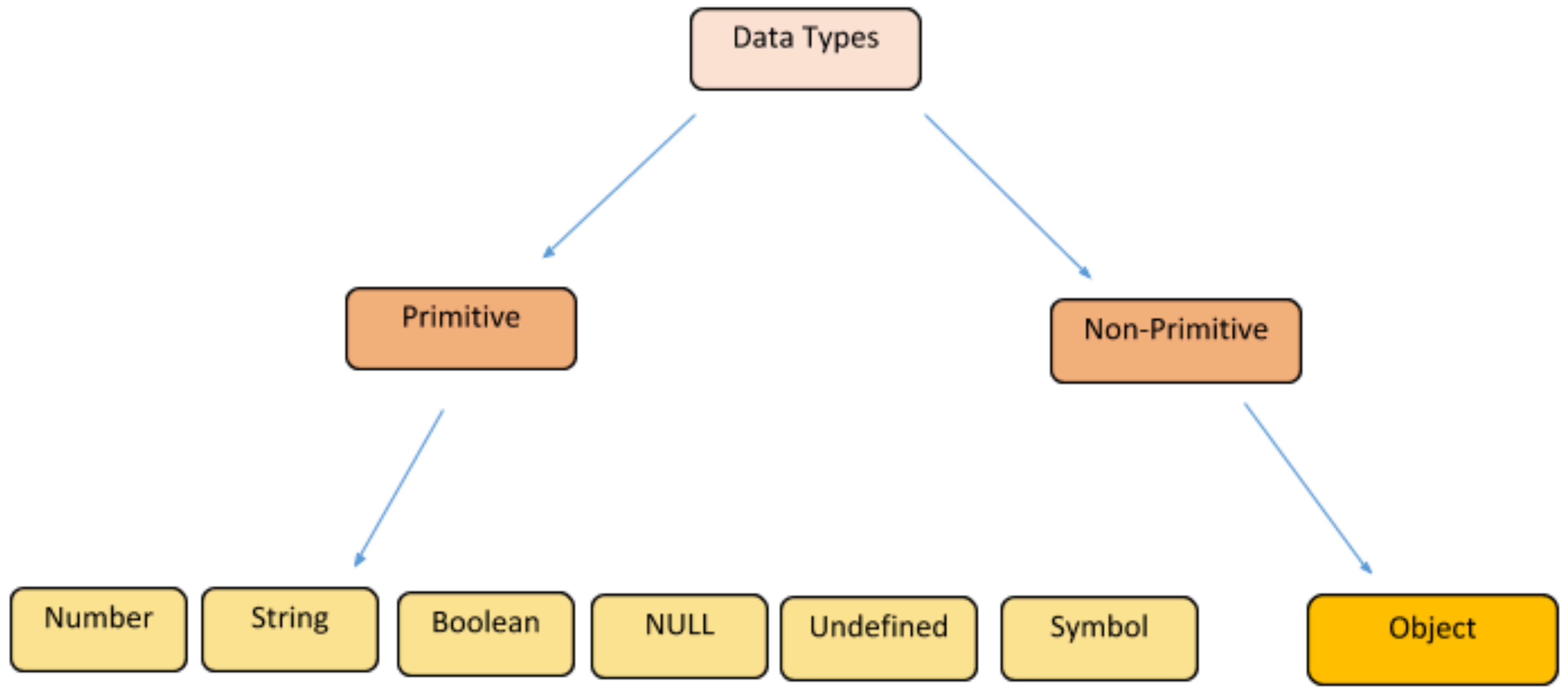
Please spend time thinking about the right name for a variable before declaring it. Doing so will repay you handsomely.

Some good-to-follow rules are:

- Use human-readable names like `userName` or `shoppingCart`.
- Stay away from abbreviations or short names like `a`, `b`, `c`, unless you really know what you're doing.
- Make names maximally descriptive and concise. Examples of bad names are `data` and `value`. Such names say nothing. It's only okay to use them if the context of the code makes it exceptionally obvious which data or value the variable is referencing.
- Agree on terms within your team and in your own mind. If a site visitor is called a "user" then we should name related variables `currentUser` or `newUser` instead of `currentVisitor` or `newManInTown`.

Data types

In computer science and computer programming, a **data type** or simply **type** is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.



Data types

There are 8 basic data types in JavaScript.

- `number` for numbers of any kind: integer or floating-point, integers are limited by $\pm 2^{53}$.
- `bigint?` is for integer numbers of arbitrary length.
- `string` for strings. A string may have one or more characters, there’s no separate single-character type.
- `boolean` for `true`/`false`.
- `null` for unknown values – a standalone type that has a single value `null`.
- `undefined` for unassigned values – a standalone type that has a single value `undefined`.
- `object` for more complex data structures.
- `symbol` for unique identifiers.

The `typeof` operator allows us to see which type is stored in a variable.

- Two forms: `typeof x` or `typeof(x)`.
- Returns a string with the name of the type, like `"string"`.
- For `null` returns `"object"` – this is an error in the language, it’s not actually an object.

Code Structure

Statements

Statements are syntax constructs and commands that perform actions.

We’ve already seen a statement, `alert('Hello, world!')`, which shows the message “Hello, world!”.

We can have as many statements in our code as we want. Statements can be separated with a **semicolon**.

Usually, statements are written on separate lines to make the code more readable

A semicolon may be omitted in most cases when a line break exists.

Code Structure - Comments

Comments

As time goes on, programs become more and more complex. It becomes necessary to add *comments* which describe what the code does and why.

Comments can be put into any place of a script. They don't affect its execution because the engine simply ignores them.

One-line comments start with two forward slash characters `//`.

```
// This comment occupies a line of its own
alert('Hello');

alert('World'); // This comment follows the statement
```

Multiline comments start with a forward slash and an asterisk `/*` and end with an asterisk and a forward slash `*/`

```
/* An example with two messages.
This is a multiline comment.
*/
alert('Hello');
alert('World');
```

The “script” tag

JavaScript programs can be inserted into any part of an HTML document with the help of the `<script>` tag.

```html
<!DOCTYPE HTML>
<html>

<body>

  <p>Before the script...</p>

  <script>
    alert( 'Hello, world!' );
  </script>

  <p>...After the script.</p>

</body>

</html>
```

The `<script>` tag contains JavaScript code which is automatically executed when the browser processes the tag

External scripts

If we have a lot of JavaScript code, we can put it into a separate file.

Script files are attached to HTML with the `src` attribute:

```
<script src="/path/to/script.js"></script>
```

Benefits of external scripts:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

Learning Resources

1. **Debugging**
    1. General: **https://www.w3schools.com/js/js_debugging.asp**
    2. RU version: **https://learn.javascript.ru/debugging-chrome**
    3. Video from Chrome: **https://www.youtube.com/watch?v=H0XScE08hy8**
2. **Scripts:**
    1. MDN: **https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script**
    2. W3C: **https://www.w3.org/TR/2011/WD-html5-20110525/scripting-1.htm**
    3. EN version: **https://javascript.info/hello-world**
    4. RU version: **https://learn.javascript.ru/external-script**
3. **JS Data types:**
    1. **https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures**
    2. RU version: **https://learn.javascript.ru/types-intro**
    3. EN version: **https://javascript.info/types**

Home Work

1. Create a new project in GitHub lesson-13-hw
2. Create a index.html file
3. Create a script.js file and attach it to index.html
4. Create variables for each Data Types(except Symbol), and with console.log method, using **typeof operator,** log each type.