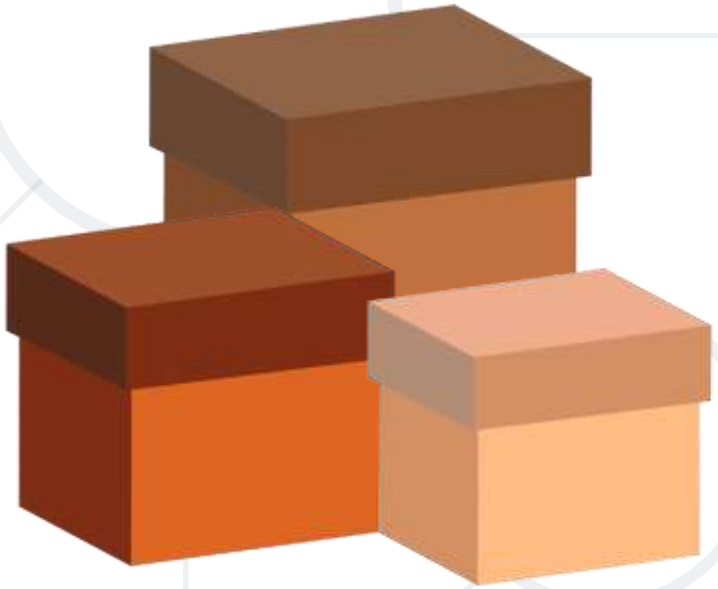
Data Types and Variables

Types of Operators



SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Content

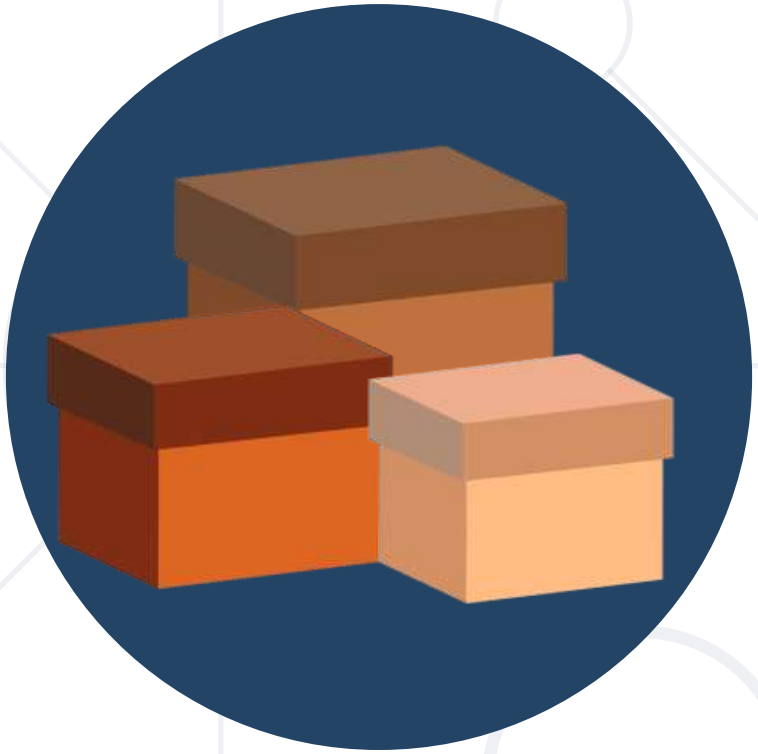
SoftUni Foundation

Have a Question?

SoftUni Foundation

sli.do

#tech-fund



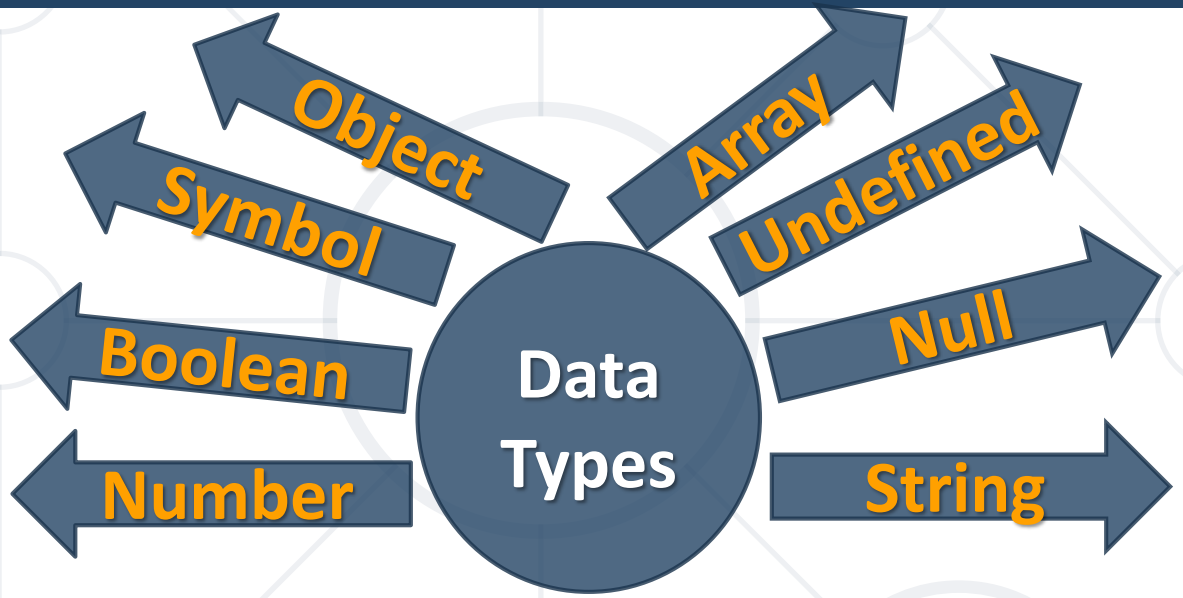
What is data type

Definition and examples

What is Data Type?

- A particular kind of **data item**, as defined by the values it can take, the programming language used, or the operations that can be performed on it
- After **ECMAScript** 2015 there are **seven** data types:
  - Six **primitive**: Boolean, Null, Undefined, Number, String, Symbol (new in ECMAScript 6)
  - and **Objects**

Examples




```
let number = 10;                                  // Number
let name = 'George';                              // String
let array = [1, 2, 3];                            // Array
let isTrue = true;                                // Boolean
let person = {name: 'George', age: 25};           // Object
let empty = null;                                 // Null
let unknown = undefined;                          // Undefined
```

Data Types are dynamic

- JavaScript is a loosely typed or a **dynamic** language. Variables are **not** directly associated with any particular value type, and any variable can be **assigned** (and **re-assigned**) values of all types:

```
let variable = 42; // variable is now a number
variable = 'bar';  // variable is now a string
variable = true;   // variable is now a boolean
```

Let vs. Var

local vs. global

Var and Let

- JavaScript variables are **containers** for storing data values.

- **var** - variables declared inside a block **{}** can be accessed from outside the block

```
{
    var x = 2;
}
console.log(x); // 2
```

- **let** - variables declared inside a block **{}** can not be accessed from outside the block

```
{
    let x = 2;
}
console.log(x) // undefined
```

Variables Scope

- The scope of a variable is the **region** of the program in which it is defined
  - Global Scope – **Global** variables can be accessed from anywhere in a JavaScript function

```
var carName = "Volvo";
// code here can use carName
function myFunction() {
    // code here can also use carName
}
```

Variables Scope (2)

- Function Scope – **Local** variables can only be accessed from inside the function where they are declared

```
function myFunction() {
    var carName = "Volvo";
    // only here code CAN use carName
}
```

- Block Scope - Variables declared inside a block **{}** can not be accessed from outside the block.

```
{
    let x = 2;
} // x can NOT be used here
```

Naming Variables

- Variable names are **case sensitive**
- Variable names must begin with a **letter** or **underscore** (_) character
- Variable names **can't** be one of JavaScript's reserved words like: break, const, interface, typeof, true etc.

```
firstName, report, config, fontSize, maxSpeed
```

```
foo, bar, p, p1, LastName, last_name, LAST_NAME
```

'ABC'

Strings

sequence of characters

What is a String?

- Used to represent **textual data**.
- Each element in the String occupies a **position** in the String.
- The **first** element is at **index 0**, the next at index 1, and so on.
- The **length** of a String is the number of elements in it.

Accessing element at index

```
let name = 'George';
console.log(name[0])  // 'G'
```

Strings are immutable

- Unlike in languages like C, JavaScript strings are **immutable**. This means that once a string is created, it is **not** possible to **modify** it.

```
let name = 'George';
name[0] = 'P';
console.log(name)  // 'George'
```

String Interpolation

- In JS we can use **template literals**. These are string literals that allow **embedded** expressions.

```
let name = 'Rick';
let age = 18;
console.log(`${name} = ${age}`);
// 'Rick = 18'
```

Use **back tick** to declare a strings

Place your **variables** after the **'$'** sign

Problem: Concatenate Names

- Receive two **names** as **string parameters** and a **delimiter**
- Print the names **joined** by the delimiter

`'John', 'Smith', '->'`	→	`John->Smith`
`'Jan', 'White', '<->'`	→	`Jan<->White`
`'Linda', 'Terry', '=>'`	→	`Linda=>Terry`

Solution: Concatenate Names

- Solution:

```
function solve(first, second, del) {
    console.log(`${first}${del}${second}`);
}
```

```
solve('John', 'Wick', '***')
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/1242

Problem: Right place

- You will receive **3 parameters** (string, symbol, string)
- Replace the underscore **'_'** in the **first word** with the **symbol**
- Compare both strings and print **"Matched"** or **"Not Matched"**

`'Str_ng', 'I', 'Strong'`	→	`Not Matched`
`'Str_ng', 'i', 'String'`	→	`Matched`

Solution: Right Place

- Solution:

```
function solve(str, symbol, result) {
    let res = str.replace('_', symbol);
    let output = res === result ? "Matched" : "Not Matched";
    console.log(output);
}
```

```
solve('Str_ng', 'I', 'Strong')
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/1242

123

Numbers

integer, float, double all in one

What is a Number?

- There is **no specific** type for integers and floating-point numbers.
- In addition to being able to represent floating-point numbers, the number type has three symbolic values: **+Infinity**, **-Infinity**, and **NaN**(not-a-number).

```
let num1 = 1;
let num2 = 1.5;
let num3 = 'p';
console.log(num1 + num2)  // 2.5
console.log(num1 + num3)  // '1p'
console.log(Number(num3)) // NaN
```

Concatenation

Trying to parse a string

Problem: Integer and Float

- You will receive **3 numbers**
- Find their **sum** and print **"{Sum} – {Integer or Float}"**

`9, 100, 1.1`	→	`110.1 - Float`
`100, 200, 303`	→	`603 - Integer`
`122.3, 212.3, 5`	→	`339.6 - Float`

Solution: Integer or Float

- Solution:

```
function solve(num1, num2, num3) {
    let sum = num1 + num2 + num3;
    let output = sum % 1 === 0
        ? sum + ' - Integer' : sum + ' - Float';
    console.log(output);
}
```

```
solve(112.3, 212.3, 5)
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/1242

true

false

Booleans

conditions

What is a Boolean?

- Boolean represents a logical entity and can have two values: **true** and **false**.
- You can use the **Boolean()** function to find out if an expression (or a variable) is true:

```
Boolean(10 > 9)      // returns true
```

- Or even easier:

```
(10 > 9)             // also returns true
10 > 9               // also returns true
```

Comparisons and Conditions

Operator	Description	Example
==	equal to (no type)	if (day == 'Monday')
>	greater than	if (salary > 9000)
<	less than	if (age < 18)
===	equal to (with type)	if (5 === 5)
>=	greater than or equal (no type)	if (6 >= 6)
<==	less than or equal (with type)	if (10 <== 100)
!==	not equal (with type)	if (5 !== '5')
!=	not equal (no type)	if (5 != 5)

Booleans Examples

- Everything with a "value" is **true**

```
let number = 1;
if (number) {
  console.log(number) // 1
}
```

true

- Everything without a "value" is **false**

```
let number;
if (number) {
  console.log(number)
} else {
  console.log('false')  // false
}
```

false

Booleans Examples (2)

```
let x = 0;
Boolean(x);              // false
let x = -0;
Boolean(x);              // false
let x = '';
Boolean(x);              // false
let x = false;
Boolean(x);              // false
let x = null;
Boolean(x);              // false
let x = 10 / 'p';
Boolean(x);              // false
```

Problem: Amazing Numbers

- You will receive **a number,** check if it is **amazing**
- An amazing is a number, which **sum** of digits includes **9**
- Print it in format **"{number} Amazing? {True or False}"**

`1233`	→	`1233 Amazing? True`
`999`	→	`999 Amazing? False`

Solution: Amazing Numbers

- Solution:

```
function solve(num) {
    num =  num.toString();
    let sum = 0;
    for(let i = 0; i < num.length; i++)
        sum += Number(num[i]);
    let result = sum.toString().includes('9');
    console.log(result ? `${num} Amazing? True`
        : `${num} Amazing? False`);
}
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/1242

Arrays & Objects

Reference types

Definition and examples

- Arrays are used to store multiple values in a single variable.

```
let cars = ["Saab", "Volvo", "BMW"];
```

in square brackets, separated by commas.

- Objects containers for named values called properties or methods.

```
let person = {
  firstName:"John",
  lastName:"Doe",
  age:50,
  eyeColor:"blue"
};
```

in curly braces, properties are written as name : value pairs, separated by commas.

typeof

Typeof Operator

checking for a type

Definition and examples

- Used to find the **type of** a JavaScript **variable**.
- **Returns** the **type** of a variable or an expression:

```
console.log(typeof "")          // Returns "string"
console.log(typeof "John")      // Returns "string"
console.log(typeof "John Doe") // Returns "string"
console.log(typeof 0)           // Returns "number"
```

```
let number = 5;
if (typeof(n) === 'number') {
    console.log(number);  // 5
}
```

Undefined
Null

Undefined and Null

non-existent and empty

Undefined

- A variable without a value, has the value **undefined**. The **typeof** is also **undefined**.

```
let car;  // Value is undefined, type is undefined
```

- A variable can be emptied, by setting the value to **undefined**. The type will also be **undefined**.

```
let car = undefined;  // Value is undefined, type
is undefined
```

Null

- **Null** is **"nothing"**. It is supposed to be something that doesn't exist.
- The **typeof** null is an **object**.

```
let person = {
  firstName:"John",
  lastName:"Doe",
  age:50
};
person = null;
console.log(person);          // null
console.log(typeof(person)); // object
```

Null and Undefined

SoftUni Foundation

- **Null** is an assigned value. It means nothing.
- **Undefined** typically means a variable has been declared but not de fined yet.
- **Null** and **Undefined** are falsy values.
- **Undefined** and **Null** are equal in value but different in type:

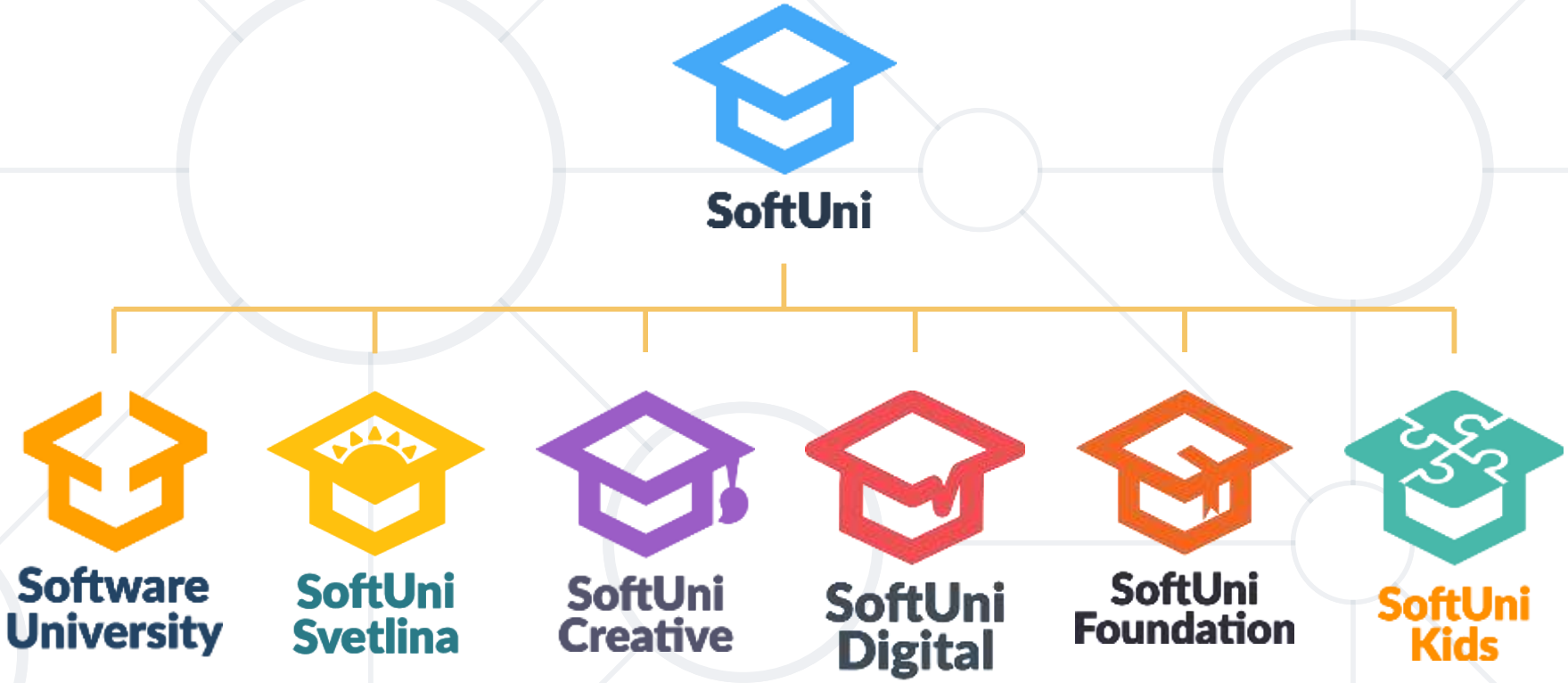
```
null !== undefined          // true
null == undefined           // true
```

Live Exercises

Summary

- There are **7 data types** in JavaScript: Number, String, Symbol, Null, Undefined, Object, Boolean
- **Let** is a local variable, **var** is a global variable
- With **typeof** we can receive the type of a variable
- **Null** is "nothing", **undefined** exists, but is empty

Questions?



https://softuni.bg/trainings/courses

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

LIEBHERR

æternity

codexio

SoftUni Organizational Partners

SoftUni Foundation

ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

WORLD OF MYTHS

Lukanet.com

codeXio

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license