



KINGSLAND  
UNIVERSITY

## Data Types and Variables



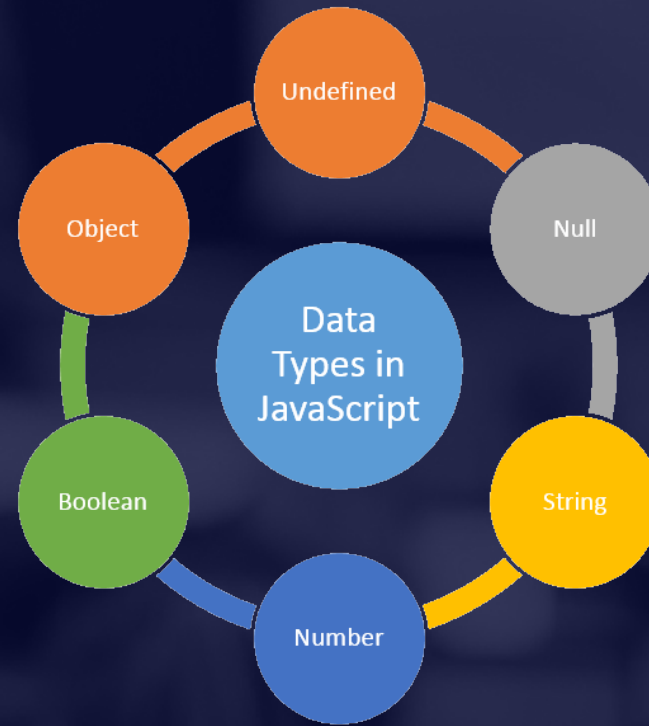
# Types of Operators



# Table of Contents

- ✔ What is a data type?
- ✔ Let vs. Var
- ✔ Strings
- ✔ Numbers
- ✔ Booleans
- ✔ Arrays and Objects
- ✔ typeof operator
- ✔ Undefined and Null



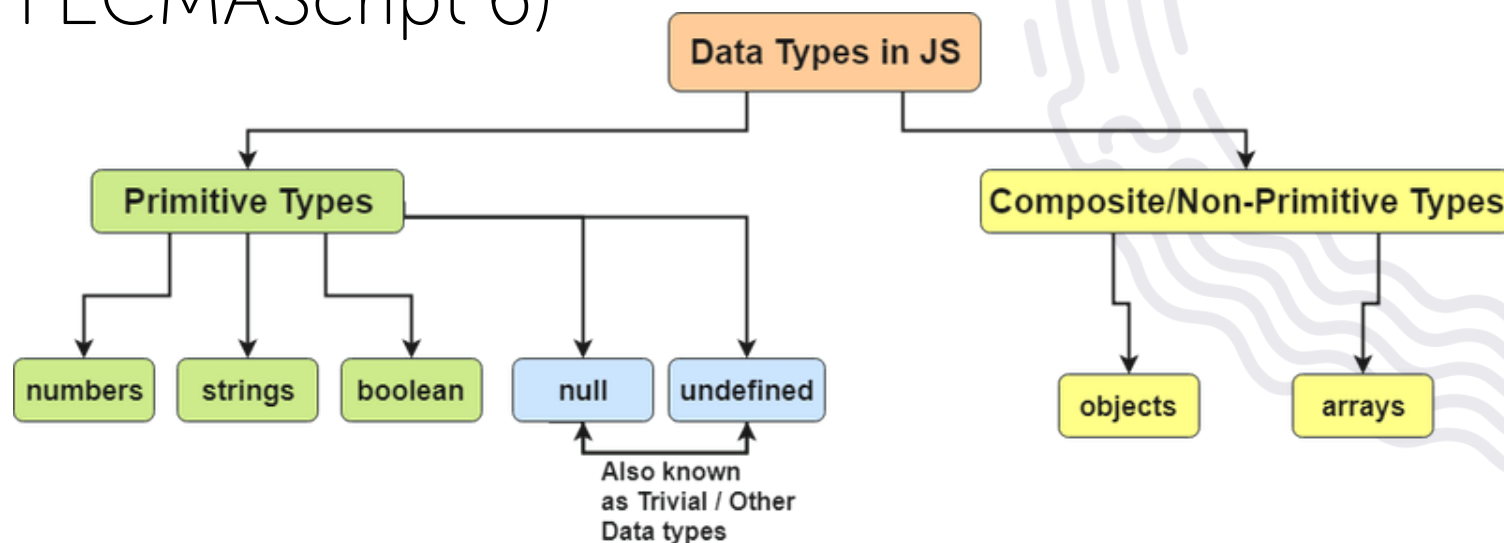


# What is Data Type

## Definition and Examples

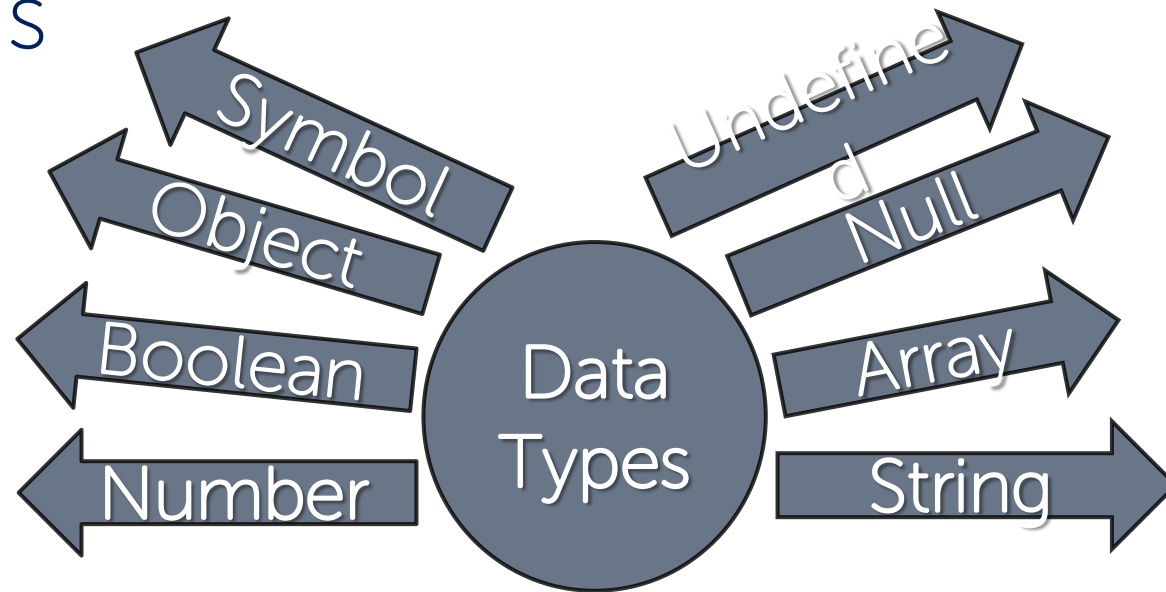
# What is a Data Type?

- The **data type** of a value is an attribute that tells what kind of data that value can have
- 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 **dynamic** language
- ✔ Variables are **not** directly associated with any particular value type
- ✔ 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 ⚡ var

Let vs. Var

Local vs. Global





# Var and Let

✓ **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  
`firstName, report, config, fontSize, maxSpeed`
- ✓ Variable names **can't** be one of JavaScript's reserved words like: **break, const, interface, typeof, true** etc.  
`foo, bar, p, p1, LastName, last_name, LAST_NAME`

'ABC'

Strings

Sequence of Characters



# What is a String?

- ✓ Used to represent **textual data**
- ✓ Each **symbol** 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

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

Accessing element at index



# 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'
```

Place your variables after the '\$' sign

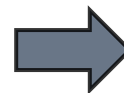




# Problem: Concatenate Names

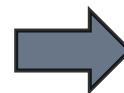
- ✓ Receive two **names** as **string parameters** and a **delimiter**
- ✓ Print the names **joined** by the delimiter

'John', 'Smith', '->'



John->Smith

'Jan', 'White', '<->'



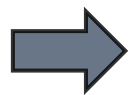
Jan<->White

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

## Problem: Right Place

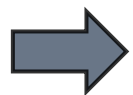
- ✓ You will receive **3 parameters** (string, symbol, string)
- ✓ Replace the underscore '\_' in the **first word** with the **symbol**
- ✓ Compare both strings and print "Matched" or "Not Matched"

'Str\_ng', 'I', 'Strong'



Not Matched

'Str\_ng', 'i', 'String'



Matched

```
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')`

123

## Numbers

Integer, Float, Double – All in One



# What is a Number?

- ✓ There is **no specific** type for integers and floating-point numbers
- ✓ 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

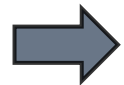
Trying to parse a string

# Problem: Integer or Float

✓ You will receive **3 numbers**

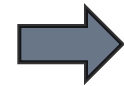
✓ Find their **sum** and print "{Sum} - {Integer or Float}"

9, 100, 1.1



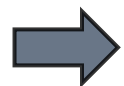
110.1 - Float

100, 200, 303



603 - Integer

122.3, 212.3, 5



339.6 - Float

```
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)

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
<code>==</code>	equal to (no type)	<code>if (day == 'Monday')</code>
<code>&gt;</code>	greater than	<code>if (salary &gt; 9000)</code>
<code>&lt;</code>	less than	<code>if (age &lt; 18)</code>
<code>===</code>	equal to (with type)	<code>if (5 === 5)</code>
<code>&gt;=</code>	greater than or equal (no type)	<code>if (6 &gt;= 6)</code>
<code>!==</code>	not equal (with type)	<code>if (5 !== '5')</code>
<code>!=</code>	not equal (no type)	<code>if (5 != 5)</code>





# 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)

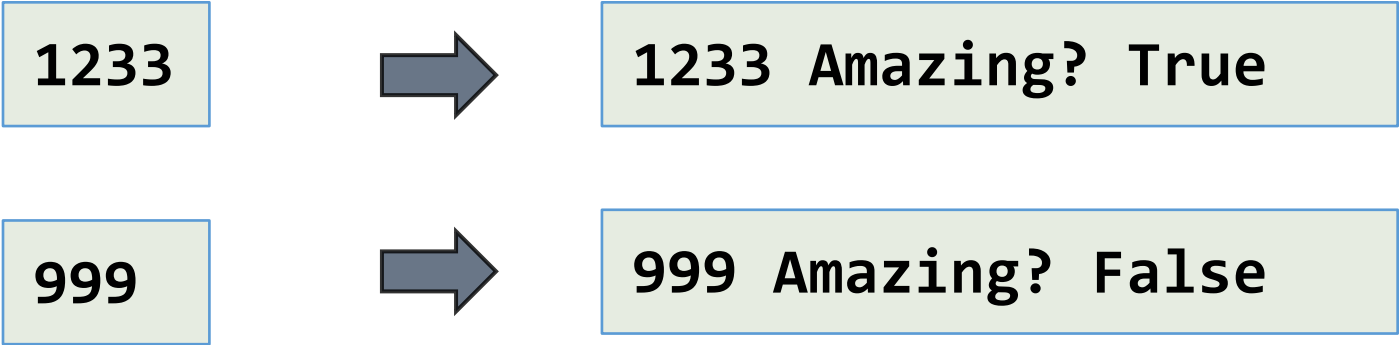
```
let x = 0;  
Boolean(x);           // false  
let x = -0;  
Boolean(x);           // false  
let x = '';  
Boolean(x);           // false  
let x = false;  
Boolean(x);           // false  
let x = null;  
Boolean(x);           // false  
let x = 10 / 'p';  
Boolean(x);           // false
```





# 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}**"





# Solution: Amazing Numbers

```
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`);  
}
```

[]

{ }

# 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

Type of Operator

Checking for a Type



# Definition and Examples

- ✓ Used to find the **type** of a **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

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

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

The background of the slide is a dark blue, blurred image of a classroom. In the foreground, the backs of several students' heads are visible as they sit at desks. In the background, a whiteboard is mounted on a wall, and other students are seated further back in the room.

# 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?





# License

- ✔ This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- ✔ Unauthorized copy, reproduction or use is illegal
- ✔ © Kingsland University – <https://kingslanduniversity.com>







THANK YOU

