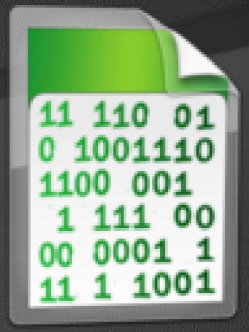
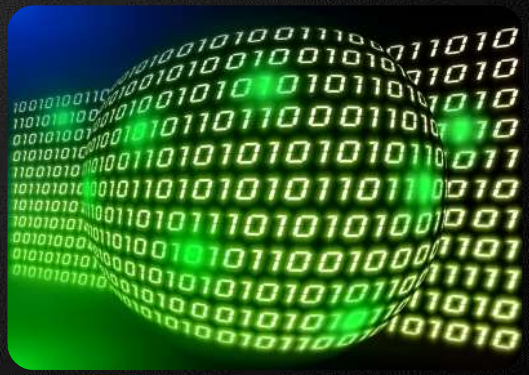
Telerik Academy



Data Types and Variables



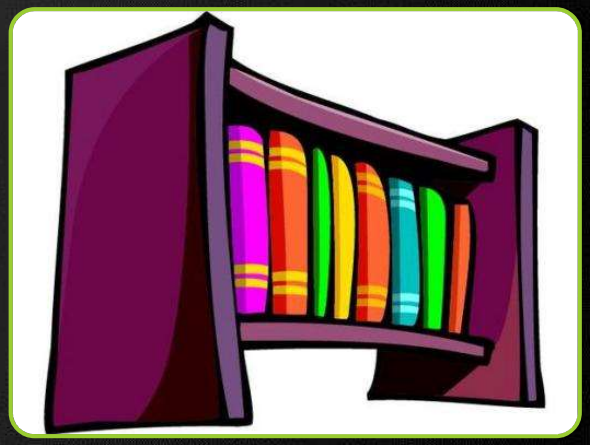
JavaScript Fundamentals

Telerik Software Academy

https://telerikacademy.com

Telerik Academy

Table of Contents



Telerik Academy

Data Types in JavaScript

Telerik Academy

How Computing Works?

- Computers are machines that process data
  - Data is stored in the computer memory in variables
  - Variables have name, data type and value
- *Example* of variable definition and assignment in JavaScript:

```
let count = 5;
```

Variable value

Variable name

Telerik Academy

What Is a Data Type?

- A data type:
  - Is a domain of values of similar characteristics
  - Defines the type of information stored in the computer memory (in a variable)
- *Examples*:
  - Positive integers: 1, 2, 3, ...
  - Alphabetical characters: a, b, c, ...

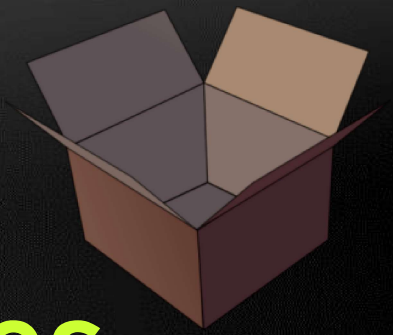
Telerik Academy

JavaScript Data Types

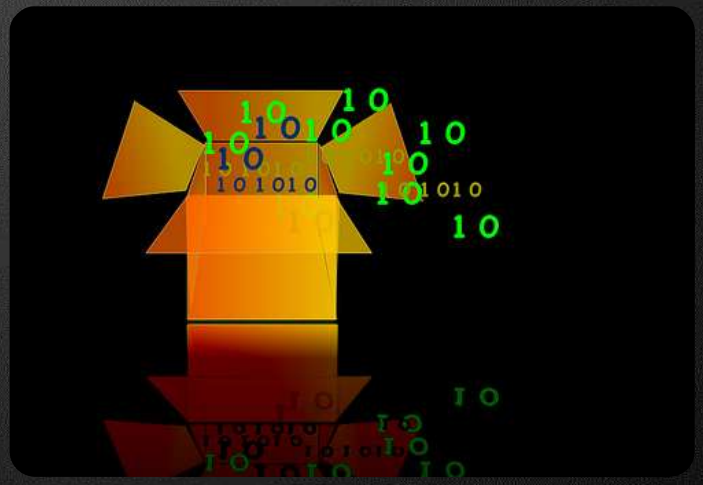
- JavaScript is weakly typed language
  - allows most operations on values without regards to their types
  - values have types, variables don't
  - variables can hold any type of value
  - All variables are declared with the keywords `var`, `let` or `const`

```
var count = 5; // variable holds an integer value
count = 'hello'; // the same variable now holds a string

var name = 'Telerik Academy'; // variable holds a string

let mark = 5.25 // mark holds a floating-point number
mark = true; // mark now holds a boolean value

const MAX_COUNT = 250; // name is a constant variable that holds a string
MAX_COUNT = 0; // error, cannot assign to a constant variable
```

Telerik Academy



Introducing Variables



Telerik Academy

What Is a Variable?

- A variable is a:
  - Placeholder of information that can usually be changed at run-time
  - A piece of computer memory holding some value
- Variables allow you to:
  - Store information
  - Retrieve the stored information
  - Manipulate the stored information

Telerik Academy

Variable Characteristics

- A variable has:
  - Name
  - Value
- *Example*:
  - Name: `counter`
  - Value: `5`
    - Type of the `counter`'s value: `number`

```
let counter = 5;
```

Telerik Academy

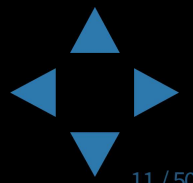
Numbers in JavaScript

Telerik Academy

Numbers in JavaScript

- All numbers in JavaScript are stored internally as double-precision floating-point numbers
- According to the IEEE-754 standard
  - Can be wrapped as objects of type `Number`
- *Example*:

```
let value = 5;
value = 3.14159;
value = new Number(100); // Number { 100 }
value = value + 1; // 101
let biggestNum = Number.MAX_VALUE;
```

Telerik Academy

Numbers Conversion

- Convert `floating-point` to `integer` number

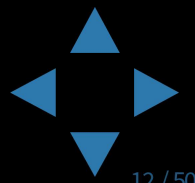
```
let valueDouble = 8.75;
let valueInt = valueDouble | 0; // 8
```

- Convert to `integer` number with rounding

```
let valueDouble = 8.75;
let roundedInt = (valueDouble + 0.5) | 0; // 9
```

- Convert `string` to `integer`

```
let str = '1234';
let i = str | 0 + 1; // 1235
```

Telerik Academy

Number Conversion

Demo

Telerik Academy

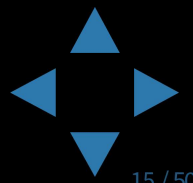
Integer numbers

Telerik Academy

What are Integer numbers?

- Integer numbers in JavaScript:
  - Represent whole numbers
  - Have range of values, depending on the size of memory used
- Integer values can hold numbers from `-9007199254740992` to `9007199254740992`
  - Their underlying type is a floating-point number (IEEE-754)

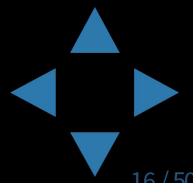
```
let studentsCount = 5;
let maxInteger = 9007199254740992;
let minInteger = -9007199254740992;
let a = 5, b = 3;
let sum = a + b; // 8
let div = a / 0; // Infinity
```

Telerik Academy

Integer numbers

Demo



Telerik Academy

Floating-Point numbers

Telerik Academy

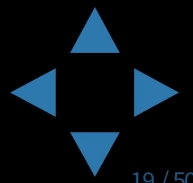
What are Floating-Point numbers?

- Floating-point types:
  - Represent real numbers
  - Have range of values and precision
  - Can behave abnormally in the calculations

Telerik Academy

Floating-Point numbers

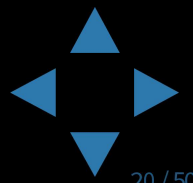
- Floating-point size depend on the platform
  - The browser and the OS
- 32-bit OS and browser have 32 bits for number, while 64-bit have 64 bits
  - It is good idea to use up to 32-bit numbers
    - Will always work on all platforms



Floating-Point Types - *Example*

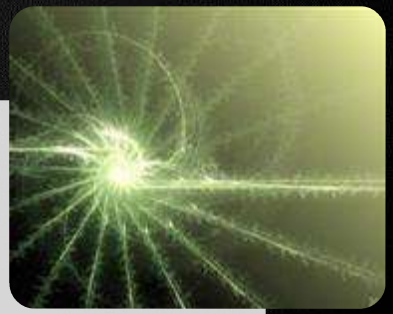
- The floating-point type can hold numbers from 5e-324 to 1.79e+308

```
let PI = Math.PI; // 3.141592653589793
let minValue = Number.MIN_VALUE; // 5e-324
let maxValue = Number.MAX_VALUE; // 1.79e+308
let div0 = PI / 0; // Infinity
let divMinus0 = -PI / 0; // -Infinity
let unknown = div0 / divMinus0; // NaN
```

Abnormalities in the Floating-Point Calculations

- Sometimes abnormalities can be observed when using floating-point numbers
  - Comparing floating-point numbers can not be performed directly with the `equals` operators (== and ===)
- *Example*:



```
let a = 0.1;
let b = 0.2;
let sum = 0.3;
let equal = (a+b === sum); // false!!!
console.log('a+b = '+ (a + b) + ', sum = ' +
  sum + ', sum == a+b? is ' + equal);
```

Telerik Academy

Floating-Point numbers

Demo

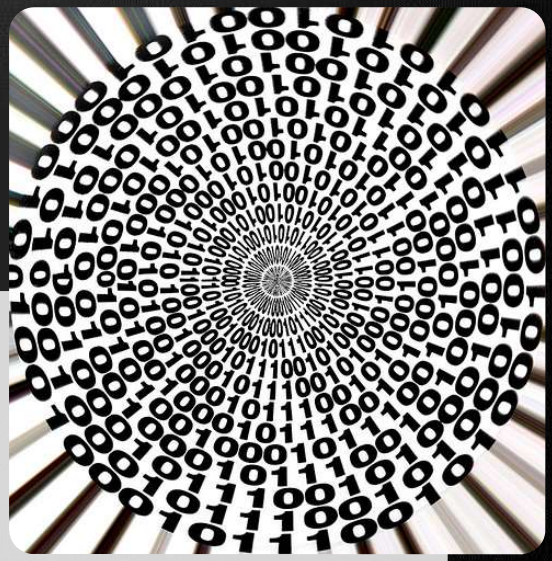
Telerik Academy

Boolean Type

Telerik Academy

The Boolean Data Type

- Has two possible values:
  - `true` and `false`
- Used in logical expressions



```
let a = 1;
let b = 2;
let greaterAB = (a > b);
console.log(greaterAB);  // false

let equalA1 = (a === 1);
console.log(equalA1);    // true

console.log((a !== b) && (b > 0));
```

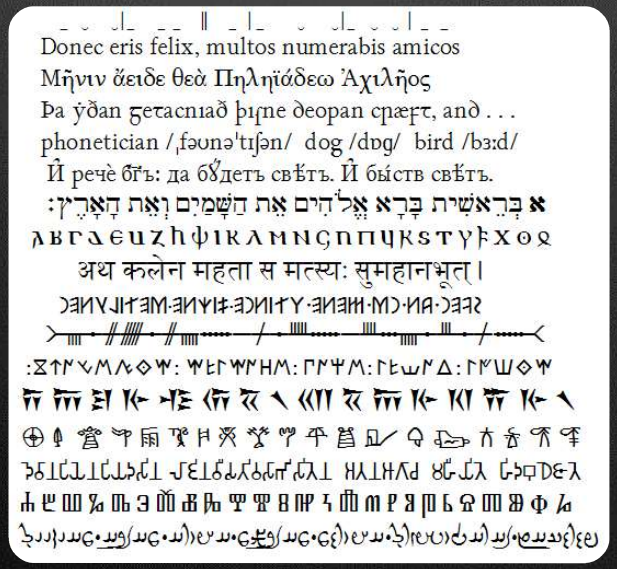
Telerik Academy

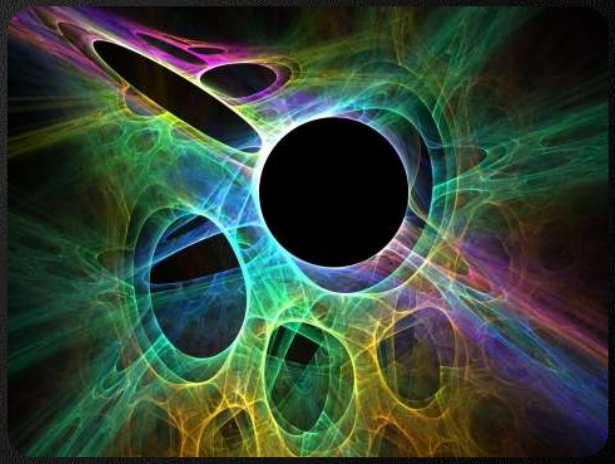
Boolean Type

Demo

Telerik Academy

String Type



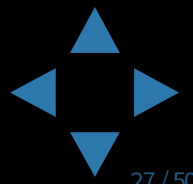


Telerik Academy

The String Data Type

- Represents a sequence of characters
- Strings are enclosed in quotes:
  - Both ' and " work correctly
  - ES6 also includes ` (ticks) for string interpolation
- Strings can be concatenated
  - Using the + operator

```
let s = 'Welcome to JavaScript';
let name = 'John' + ' ' + 'Doe';
let greeting = `${s}, ${name}`;

console.log(greeting); // Welcome to JavaScript, John Doe
```

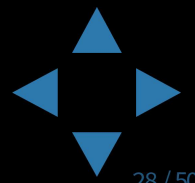


Telerik Academy

Saying Hello - *Example*

- Concatenating the two names of a person to obtain his full name:
  - *Note*: a space is missing between the two names! We have to add it manually

```
let firstName = 'Ivan';
let lastName = 'Ivanov';
console.log('Hello, ' + firstName + '!');

let fullName = firstName + ' ' + lastName;
console.log('Your full name is ' + fullName);
```

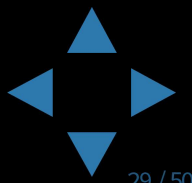



Telerik Academy

Strings are Unicode

- Strings are stored as Unicode
  - Unicode supports all commonly used alphabets in the world
    - E.g. Cyrillic, Chinese, Arabic, Greek, etc. scripts

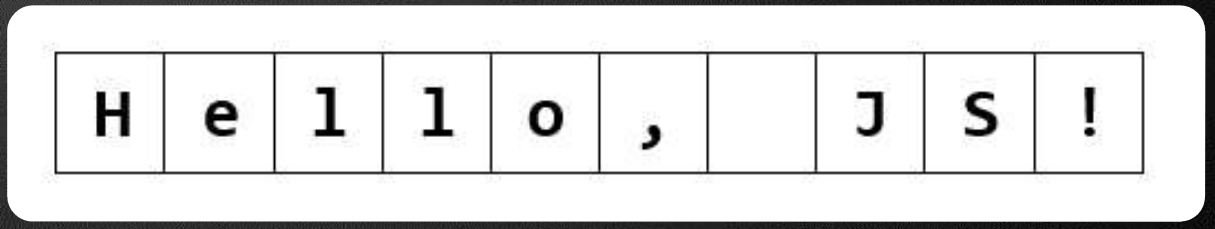
```
let asSalamuAlaykum = 'السلام عليكم';
alert(asSalamuAlaykum);

let кирилица = 'Това е на кирилица!';
alert(кирилица);

let leafJapanese = '葉';
alert(leafJapanese);
```

Telerik Academy

String Data Type

Demo



Telerik Academy

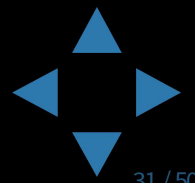
Parsing String to Number

- Strings can be parsed to numbers
  - Floating-point and rounded (integer)
- The trivial way to parse string to a number is using the functions `parseInt` and `parseFloat`:

```
let numberString = '123'
console.log(parseInt(numberString)); // prints 123
let floatString = '12.3';
console.log(parseFloat(floatString)); // prints 12.3
```

- `parseInt` and `parseFloat` exhibit stranger behaviour:
  - If a non-number string starts with a number, only the number is extracted:

```
let str = '123Hello';
console.log(parseInt(str)); // prints 123
```

Better String to Number Parsing

- `parseInt` and `parseFloat` are readable, but slow and show strange behaviour.
  - Better ways to parse string to numbers are as follows:
    - With rounding:

```
'123.3' | 0 -> returns 123
```

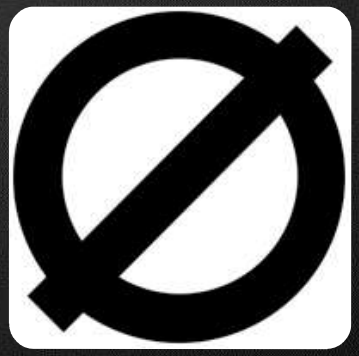
    - As is:

```
Number('123.3') -> returns 123.3
'123.3' * 1 -> returns 123.3
+'123.3' -> returns 123.3
```

Telerik Academy

Undefined and Null Values

Understanding 'undefined' in JavaScript

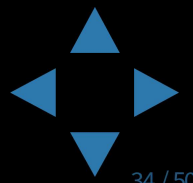


Telerik Academy

Undefined and Null Values

- JavaScript has a special value `undefined`
  - It means the variable has not been defined (no such variable in the current context)
- `undefined` is different than `null`
  - `null` represents an empty value

```
let x;
console.log(x); // undefined

x = 5;
console.log(x); // 5

x = undefined;
console.log(x); // undefined

x = null;
console.log(x); // null
```

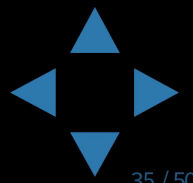


Telerik Academy

Checking a Variable Type

- The variable type can be checked at runtime:

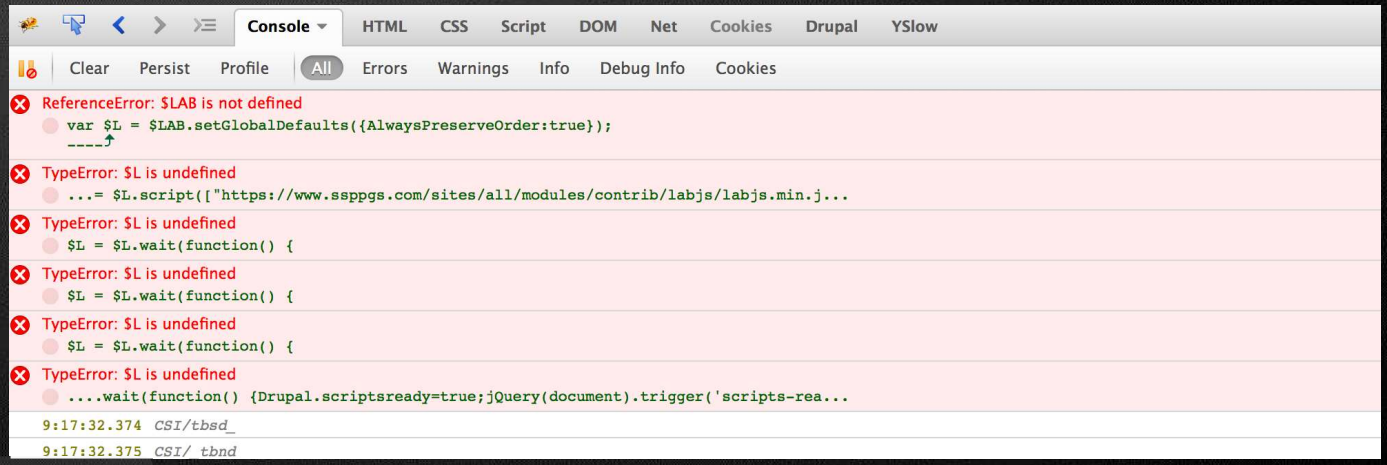
```
let x = 5;
console.log(typeof x); // number
console.log(x); // 5

x = new Number(5);
console.log(typeof x); // object
console.log(x); // Number {}

x = null;
console.log(typeof x); // object

x = undefined;
console.log(typeof x); // undefined
```



Telerik Academy

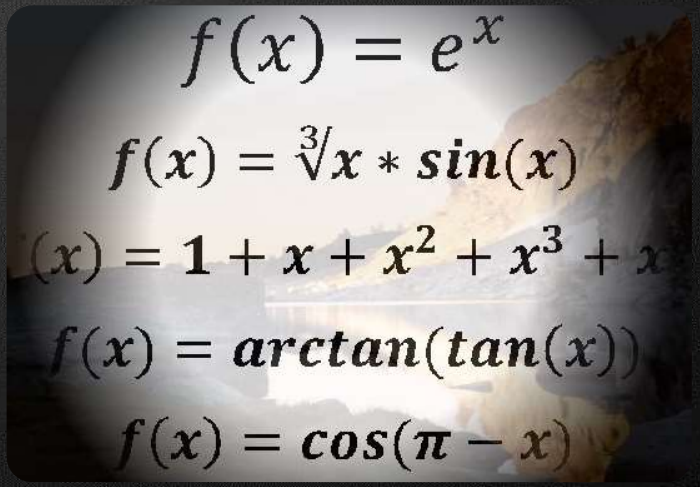
Undefined / Null / Typeof

Demo



Telerik Academy

Declaring and Using Variables



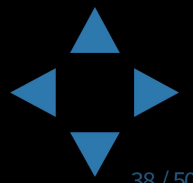
Telerik Academy

Declaring Variables

- When declaring a variable we:
  - Specify its name (called identifier)
  - May give it an initial value
- The syntax is the following:

```
<var | let | const> <identifier> [= <initialization>];
```

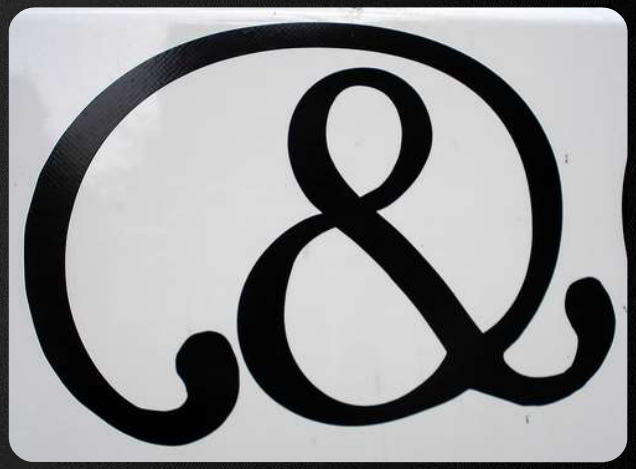
```
let emptyVariable;
var height = 200;
let width = 300;
const depth = 250;
```

Telerik Academy

Identifiers

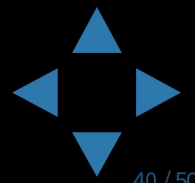
- Identifiers may consist of:
  - Letters (Unicode)
  - Digits [0-9]
  - Underscore '_'
  - Dollar '$'
- Identifiers
  - Can begin only with a letter, $, or an underscore
  - Cannot be a JavaScript keyword
- Variables / functions names: use `camelCase`



Telerik Academy

Identifiers

- Identifiers
  - Should have a descriptive name
  - It is recommended to use only Latin letters
  - Should be neither too long nor too short
- Names in JavaScript are case-sensitive
  - Small letters are considered different than the capital letters



Telerik Academy

Identifiers - *Examples*

- *Examples* of correct identifiers:

```
let New = 2; // Here N is capital
let _2Pac; // This identifier begins with _
let поздрав = 'Hello'; // Unicode symbols used
// The following is more appropriate:
let greeting = 'Hello';
let n = 100; // Undescriptive
let numberOfClients = 100; // Descriptive
// Overdescriptive identifier:
let numberOfPrivateClientOfTheFirm = 100;
```

- *Examples* of incorrect identifiers:

```
let new;    // new is a keyword
let 2Pac;    // Cannot begin with a digit
```


Telerik Academy

Assigning Values To Variables

Telerik Academy

Assigning Values

- Assigning values to variables
  - Is achieved by the = operator
- The = operator has
  - Variable identifier on the left
  - Value on the right
    - Can be of any value type
  - Could be used in a cascade calling, where assigning is done from right to left
- Variables declared with the `const` keyword cannot be reassigned after their initial assignment

Telerik Academy Assigning Values - *Examples*

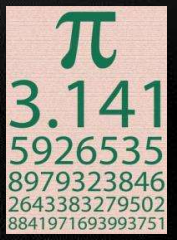
- Assigning values example:

```
let firstValue = 5;
let secondValue;
let thirdValue;

// Using an already declared variable:
secondValue = firstValue;

// The following cascade calling assigns
// 3 to firstValue and then firstValue
// to thirdValue, so both variables have
// the value 3 as a result:

thirdValue = firstValue = 3; // Avoid this!
```

Telerik Academy

Initializing Variables

- Initialization
  - Assignment of initial value
  - Must be done before the variable is used!
- Several ways of initializing a variable:
  - By using a literal expression
  - By referring to an already initialized variable
- Uninitialized variables are undefined

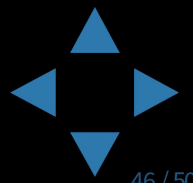


Telerik Academy

Initialization - *Examples*

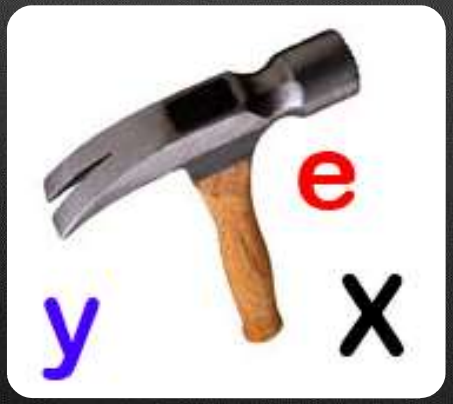
- *Example* of some initializations:

```
// This is how we use a literal expression:
let heightInMeters = 1.74;

// Here we use an already initialized variable:
let greeting = 'Hello World!';
let message = greeting;

// Use a result from an expression
const parsedNumber = parseInt('1239') + 1;
```

Telerik Academy

Assigning and Initializing Variables



Demo

Telerik Academy

Local and Global Variables

- Local variables - declared with the keywords `var`, `let` or `const`
  - `var` - the variable lives in the scope of the current function or in the global scope
  - `let` - the variables lives in the current scope
  - `const` - like `let`, but cannot be reassigned
- Global variables
  - Declared without any keyword
  - Bad practice - never do this!

```
let a = 5; // a is local in the current scope
a = 'alabala'; // the same a is referenced here
```

```
a = undefined;
a = 5; // the same as window.a = 5;
```