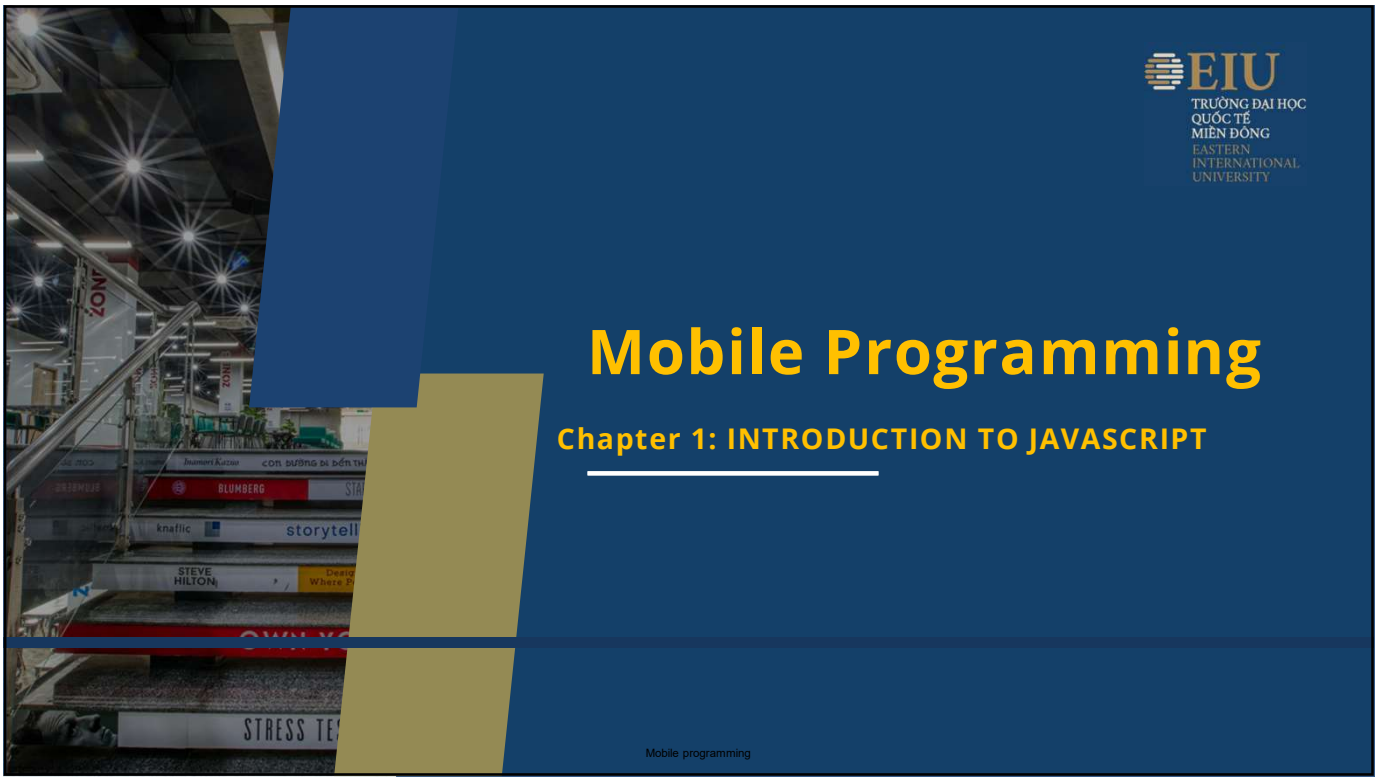


1

JavaScript Syntax

The JavaScript syntax is similar to C#, C++, Java

- **Operators** (+, *, =, !=, &&, ++,…)
- **Variables** (typeless)
- **Conditional statements** (if, else)
- **Loops** (for, while)
- **Arrays** (my_array[]) and **associative arrays** (my_array['abc'])
- **Functions** (can return value)

Note:

A **semicolon** at the end of a line indicates where a statement ends; it is only absolutely required when you need to separate statements on a single line.

2

2

The Math Object

Math Properties

```
Math.E        // returns Euler's number
Math.PI       // returns PI
Math.SQRT2    // returns the square root of 2
Math.SQRT1_2  // returns the square root of 1/2
Math.LN2      // returns the natural logarithm of 2
Math.LN10     // returns the natural logarithm of 10
Math.LOG2E    // returns base 2 logarithm of E
Math.LOG10E   // returns base 10 logarithm of E
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_math_constants

3

3

The Math Object

Math Methods

Math.round(x)	Returns x rounded to its nearest integer
Math.ceil(x)	Returns x rounded up to its nearest integer
Math.floor(x)	Returns x rounded down to its nearest integer
Math.trunc(x)	Returns the integer part of x (new in ES6)
Math.pow()	
Math.sqrt()	
Math.abs()	
Math.min() and Math.max()	

https://www.w3schools.com/js/js_math.asp

4

4

Variable Characteristics

- A variable has:
  - Name
  - Value
- Example: let count = 5;
  - Name: counter
  - Value: 5

    Type of the counter's value: number

5

5

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;
```

6

6

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**

7

7

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

8

8

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
```

9

9

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

10

10

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!
```

11

11

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 (block) scope**, and cannot redeclare
  - **const** - like let, but cannot be reassigned

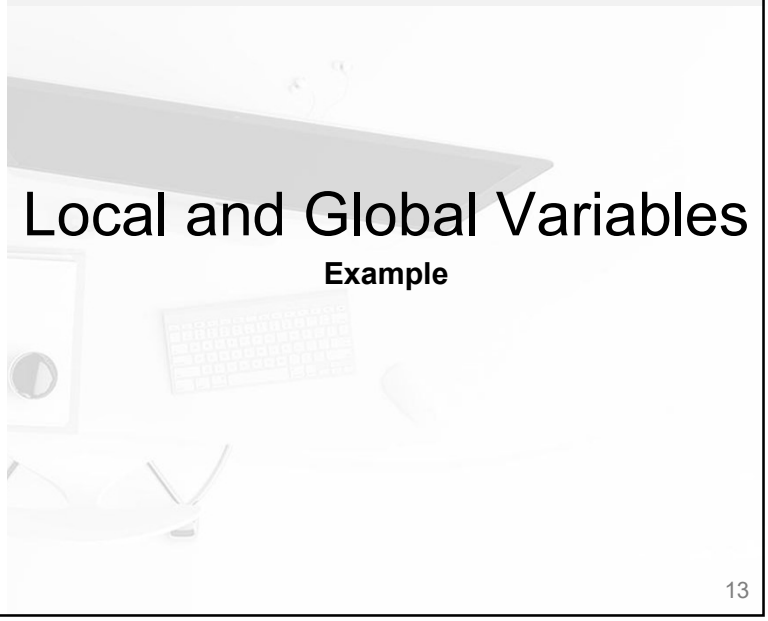
```
let a = 5; // a is local in the current scope
a = 'alabala'; // the same a is referenced here
```

- **Note:**
  - Duplicate variable declarations using **var** will not trigger an error
  - Variables declared by **let** have their scope in the block for which they are declared

12

12

```
function varTest() {
    var x = 1;
    {
        var x = 2;  // same variable!
        console.log(x);  // 2
    }
    console.log(x);  // 2
}

function letTest() {
    let x = 1;
    {
        let x = 2;  // different variable
        console.log(x);  // 2
    }
    console.log(x);  // 1
}

// If you use var to declare a variable
var myName = 'Chris';
var myName = 'Bob'; // You can do it

// If you use let to declare a variable
let myName = 'Chris';
let myName = 'Bob'; // You can't do it
```



Local and Global Variables

Example

13

13

Local and Global Variables

- **Global** variables
  - Declared **without** any **keyword**
  - Bad practice - **never do this**!

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

14

14

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;
```

15

15

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
```

16

16

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
```

17

17

Floating-Point Types

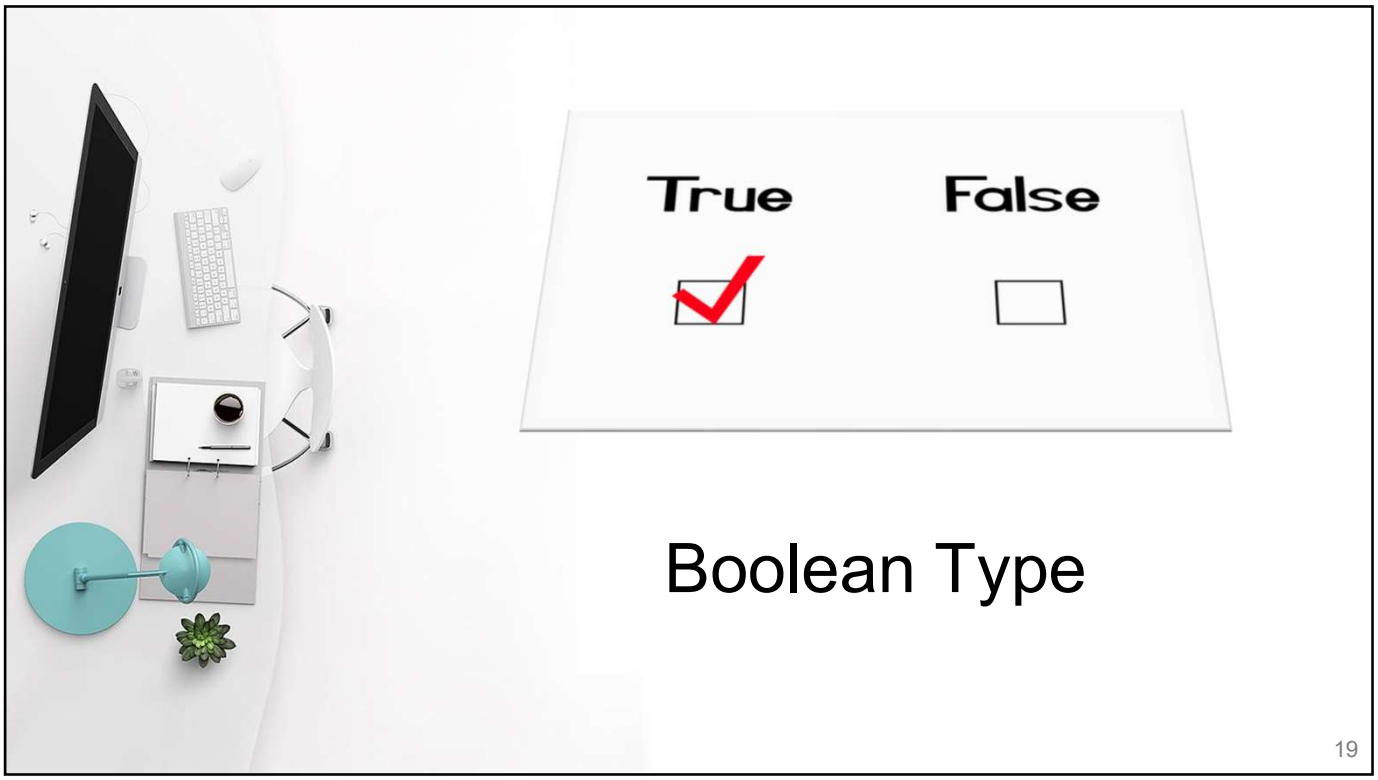
Example

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
```

18

18



Boolean Type

19

19

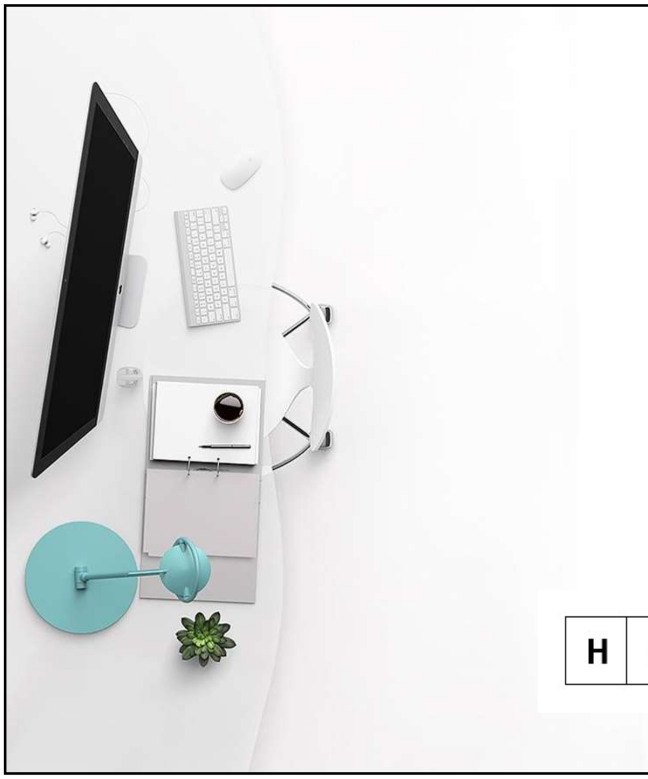
The Boolean Data Type

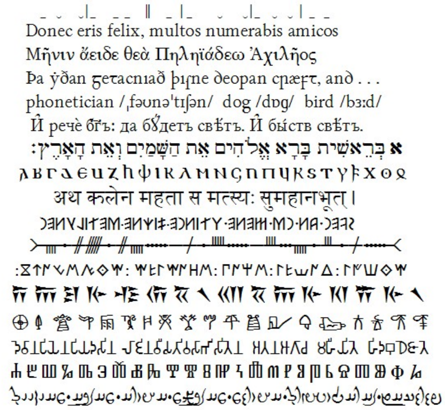
- Has **two possible values**:
  true and **false**
- Used in logical expressions

```
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));
```

20

20





String Type

H	e	l	l	o	,		J	S	!

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
```

Saying Hello

Example

Concatenating the two names of a person to obtain his full name:

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

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

23

23

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);
```

24

24

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 behavior:

  If a non-number string starts with a number, only the number is extracted:

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

25

25

Better String to Number Parsing

- parseInt and parseFloat are readable, but slow and show strange behavior
- 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

26

26

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
```

27

27

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
```

28

28

Operators by categories in JavaScript

Category	Operators
Arithmetic	+ – * / % ++ --
Logical	&& || ^ !
Binary	& | ^ ~ << >> >>>
Comparison	== != < > <= >= === !==
Assignment	= += -= *= /= %= |= ^= <<= >>=
Concatenation	+
Other	. [] () ?: new in , delete void typeof instanceof ...

29

29

Arithmetic Operators

Example

```
const squarePerimeter = 17;
const squareSide = squarePerimeter / 4;
const squareArea = squareSide * squareSide;

console.log(squareSide); // 4.25
console.log(squareArea); // 18.0625

let a = 5;
let b = 4;

console.log(a + b); // 9
console.log(a + b++); // 9
console.log(a + b); // 10
console.log(a + (++b)); // 11
console.log(a + b); // 11

console.log(12 / 3); // 4
console.log(11 / 3); // 3.6666666666666665
```

30

30

Arithmetic Operators

Example

```
console.log(11 % 3);   // 2
console.log(11 % -3);  // 2
console.log(-11 % 3);  // -2

console.log(1.5 / 0.0);  // Infinity
console.log(-1.5 / 0.0); // -Infinity
console.log(0.0 / 0.0);  // NaN

const x = 0;
console.log(5 / x);
```

31

31

Logical Operators

Example

Using the logical operators:

```
let a = true;
let b = false;

console.log(a && b); // False
console.log(a || b); // True
console.log(a ^ b); // True
console.log(!b); // True
console.log(b || true); // True
console.log(b && true); // False
console.log(a || true); // True
console.log(a && true); // True
console.log(!a); // False
console.log((5 > 7) ^ (a == b)); // False
```

32

32

Comparison Operators

- Comparison operators are used to compare variables

  ==, <, >, >=, <=, !=, **===**,!==
- For equality comparison, the use of === and !== is preferred

```
let a = 5;
let b = 4;

console.log(a >= b);    // True
console.log(a != b);    // True
console.log(a == b);    // False
console.log(0 == '');   // True
console.log(0 === ''); // False
```

33

Assignment Operators

Assignment operators are used to assign a value to a variable

=, +=, -=, *=, /=, ...

```
let x = 6;
let y = 4;

console.log(y *= 2); // 8

let z = y = 3;       // y=3 and z=3

console.log(z);      // 3
console.log(x |= 1); // 7
console.log(x += 3); // 10
console.log(x /= 2); // 5
```

34

Other Operators

- String concatenation **operator +** is used to **concatenate strings**
- If the second operand is not a string, it is converted to string automatically

```
let first = "First";
let second = "Second";

console.log(first + second); // FirstSecond

let output = "The number is : ";
let number = 5;

console.log(output + number); // The number is : 5
```

35

35

Other Operators

Example

Using some other operators:

```
let a = 6;
let b = 4;

console.log(a > b ? 'a > b' : 'b >= a'); // a > b

let c = b = 3; // b = 3; followed by c = 3;

console.log(c); // 3
console.log(new Number(6) instanceof Number); // true
console.log(6 instanceof Number); // false
console.log((a + b) / 2); // 4
console.log(typeof c); // number
console.log(void(3 + 4)); // undefined
```

36

36

Expressions

Expressions are sequences of operators, literals and variables that are evaluated to some value

```
let r = (150 - 20) / 2 + 5; // r = 70

// Expression for calculation of circle area
let surface = Math.PI * r * r;

// Expression for calculation of circle perimeter
let perimeter = 2 * Math.PI * r;
```

37

37

The if-else Statement

- More complex and useful conditional statement
- Executes one branch if the condition is true, and another if it is false
- The simplest form of an **if-else** statement:

```
if (expression) {
        statement1;
} else {
        statement2;
}
```

38

38

if-else Statement

Example

Checking a number if it is odd or even

```
var s = '123';
var number = +s;
if (number % 2) {
    console.log('This number is odd.');
} else {
    console.log('This number is even.');
}

if (+str) {
    console.log('The string is a Number');
} else {
    console.log('The string is not a Number');
}
```

39

39

Multiple if-else-if-else-…

Sometimes we need to use another if construction in the else block

Thus, **else if** can be used:

```
var ch = 'X';
if (ch === 'A' || ch === 'a') {
    console.log('Vowel [ei]');
} else if (ch === 'E' || ch === 'e') {
    console.log('Vowel [i:]');
} else if  …
else …
```

40

40

The switch-case Statement

Selects for execution a statement from a list **depending on the value** of the switch expression

```
switch (day) {
    case 1: console.log('Monday'); break;
    case 2: console.log('Tuesday'); break;
    case 3: console.log('Wednesday'); break;
    case 4: console.log('Thursday'); break;
    case 5: console.log('Friday'); break;
    case 6: console.log('Saturday'); break;
    case 7: console.log('Sunday'); break;
    default: console.log('Error!'); break;
}
```

41

41

How to use a while loop?

- The simplest and most frequently used loop
- Has a **repeat condition**
  - Also called **loop condition**
  - Is not necessary strictly a **Boolean** value
  - Is evaluated to **true** or **false**
    - ✓5, 'non-empty', {}, etc. are evaluated as true
    - ✓0, '', null, undefined are evaluated as false

```
while (condition) {
    statements;
}
```

42

42

while loop

Example

```
let counter = 0;
while (counter < 10) {
    console.log('Number : ' + counter);
    counter += 1;
}
```

43

43

Using do-while loop

Another loop structure is:

- The block of statements is repeated

  While the Boolean loop condition holds

- The loop is always **executed at least once**

```
do {
    statements;
} while (condition);
```

44

44

do-while

Example

Calculating N!

```
let fact = 1,
    factStr = 'n! = ';

do {
    fact *= n;
    factStr += n + '*'
    n -= 1;
} while (n);

factStr += ' = ' + fact;
console.log(factStr)
```

45

for loops

- The typical for loop syntax is:

```
for (initialization; test; update) {
    statements;
}
```

- Consists of
  - Initialization statement
  - Test expression that is evaluated to Boolean
  - Update statement
  - Loop body block

46

Simple for loop

Example

Print all natural numbers up to N

A simple for-loop to print the numbers [0..9]

```
const N = 10;

for (let number = 0; number < N; number += 1) {
    console.log(number + ' ');
}
```

47

Nested Loops

Example

Print the following triangle:

```
1
1 2
...
1 2 3 ... N
```

```
const N = 7;
let result = '';

for(let row = 1; row <= N; row += 1) {
   for(let column = 1; column <= row; column += 1) {
      result += column + ' ';
   }

   result += '\n';
}

console.log(result);
```

48

for-in/for-of

Example

```
let language = "JavaScript";

        let text = "";
        for (let x of language) {
            text += x;
        }
        document.writeln(text);
```

Output: JavaScript

```
let language = "JavaScript";

        let text = "";
        for (let x in language) {
            text += x;
        }
        document.writeln(text);
```

Output:0123456789

49

49

Declaring Arrays

Declaring an array in JavaScript

```
// Array holding integers
var numbers = [1, 2, 3, 4, 5];

// Array holding strings
var weekDays = ['Monday', 'Tuesday', 'Wednesday',
  'Thursday', 'Friday', 'Saturday', 'Sunday']

// Array of different types
var mixedArr = [1, new Date(), 'hello'];

// Array of arrays (matrix)
var matrix = [
  ['0,0', '0,1', '0,2'],
  ['1,0', '1,1', '1,2'],
  ['2,0', '2,1', '2,2']];
```

50

50

Array Methods

- **Array.reverse()**
  - **Reverses** the elements of the array
  - **Returns** a new arrays

```
var items = [1, 2, 3, 4, 5, 6];
var reversed = items.reverse();
//reversed = [6, 5, 4, 3, 2, 1]
```

- **Array.join(separator)**
  - **Concatenates** the elements with a separator
  - **Returns** a string

```
var names = ["John", "Jane", "George", "Helen"];
var namesString = names.join(", ");
//namesString = "John, Jane, George, Helen"
```

51

51

Concatenating Arrays

- **arr1.concat(arr2)**
  - **Inserts** the elements of **arr2** at the **end** of **arr1**
  - **Returns** a new array
  - **arr1** and **arr2** remain **unchanged**!

```
var arr1 = [1, 2, 3];
var arr2 = ["one", "two", "three"];
var result = arr1.concat(arr2);
//result = [1, 2, 3, "one", "two", "three"]
```

- **Adding the elements of an array to another array**

```
var arr1 = [1, 2, 3];
var arr2 = ["one", "two", "three"];
[].push.apply(arr1, arr2);
//arr1 = [1, 2, 3, "one", "two", "three"]
```

52

52

Sort() method

Example

- Sort numbers in ascending order:

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){ return a - b });
```

- Comparing string properties is a little more complex:

```
products.sort(function(a, b){
    let x = a.Name.toLowerCase();
    let y = b.Name.toLowerCase();
    if (x < y) {return -1;}
    if (x > y) {return 1;}
    return 0;
});
```

53

53

Sort() method

- Sorts the elements of an array in place.
- The default sort order is ascending, built upon converting the elements into strings.
- Syntax:

```
// Functionless
sort()

// Arrow function
sort((a, b) => { /* … */ } )

// Compare function
sort(compareFn)
```

CompareFn(a, b) return value	Sort order
> 0	Sort a after b
< 0	Sort a before b
=== 0	Keep original order or a and b

54

54

Declaring and Creating Functions

- Each function has a **name**
  - It is used to call the function
  - Describes its purpose
- Functions in JavaScript do not explicitly define return type

```
function printLogo() {
    console.log("JavaScript Fundamentals");
    console.log("Telerik Software Academy");
}
```

55

55

Ways of Defining a Function

Functions can be defined in 4 ways:

- **Using the constructor** of the Function object
  - Syntax:

```
new Function(functionBody)
new Function(arg0, arg1, […, argN,] functionBody)
```

  - E.g.

```
var print = new Function("console.log('Hello')")
const sum = new Function('a', 'b', 'return a + b');
```

- By **function expression**

```
var print = function() { console.log("Hello") };
var print = function printFunc() { console.log("Hello") };
```

56

56

Ways of Defining a Function

Functions can be defined in 4 ways:

- By **function declaration**

```
function print() { console.log("Hello") };
```

- By **arrow function expression**
  - Syntax:

```
param => expression
(param) => expression
(param1, paramN) => {
    expressions;
    return value;
}
```

  - Example:

```
const print = () => console.log("Hello");
const x = (x, y) => { return x * y };
```

57

Ways of Defining a Function

Example

```
// ES5
var multiply = function(x, y) {
    return x * y;
};

// ES6
var multiply = (x, y) => { return x * y };
```

```
//ES5
var phraseSplitterEs5 = function phraseSplitter(phrase) {
    return phrase.split(' ');
};

//ES6
var phraseSplitterEs6 = phrase => phrase.split(" ");

console.log(phraseSplitterEs6("ES6 Awesomeness"));  // ["ES6", "Awesomeness"]
```

```
//ES5
var docLogEs5 = function docLog() {
    console.log(document);
};

//ES6
var docLogEs6 = () => { console.log(document); }
docLogEs6(); // #document... <html> ….
```

```
//ES5
var setNameIdsEs5 = function setNameIds(id, name) {
    return {
        id: id,
        name: name
    };
};

// ES6
var setNameIdsEs6 = (id, name) => ({ id: id, name: name });

(setNameIdsEs6 (4, "Kyle"));   // Object {id: 4, name: "Kyle"}
```

58

Ways of Defining a Function

Example

```
var smartPhones = [
  { name: 'iphone', price: 649 },
  { name: 'Galaxy S6', price: 576 },
  { name: 'Galaxy Note 5', price: 489 }
  ];
  document.writeln(smartPhones.map(function(smartPhone){ return smartPhone.price;}));
  document.write("<br>")
  document.writeln(smartPhones.map(smartPhone => smartPhone.name));
```

```
Output: 649,576,489
        iphone,Galaxy S6,Galaxy Note 5
```

59

59

Calling Functions

- To call a function, simply use:
  - The function's name
  - Parentheses
  - A semicolon (;)

    Optional, but preferred

- This will execute the code in the function's body and will result in printing the following:

```
print();
// Hello
```

60

60

Calling Functions

A function can be called from:

- Any other function
- Itself (process known as **recursion**)

```
function print(){
    console.log("printed");
}

function anotherPrint(){
    print();
    anotherPrint();
}
```

61

61

Function Parameters

- To pass information to a function, you can use **parameters** (also known as **arguments**)
  - You can pass **zero or several** input values
  - Each parameter **has a name**
  - Parameters are assigned to particular values when the function is called
- Parameters change the function behavior depending on the passed values

62

62

Defining and Using Function Parameters

- Function's behavior depends on its parameters
- Parameters can be of **any type**
  - Number, String, Object, Array, etc.
  - Even Function

```
function printSign(number) {
    if (number > 0) {
        console.log("Positive");
    } else if (number < 0) {
        console.log("Negative");
    } else {
        console.log("Zero");
    }
}
```

63

63

Defining and Using Function Parameters

Functions can have **as many parameters as needed**:

```
function printMax(x, y) {
    var max;
    x = +x; y = +y;
    max = x;

    if (y > max) {
        max = y;
    }

    console.log(`Maximal number: ${max}`);
}
```

64

64

Defining and Using Function Parameters

If a function is called with missing arguments (less than declared), the missing values are set to **undefined**. It is better to assign a **default value** to the parameter.

```
// Method 1
function myFunction(x, y) {
    if (y === undefined) { y = 2; }
    return x * y;
}
function myFunction(x, y) {
    y = (typeof y !== 'undefined') ?  y : 1;
    return x * y;
}

// Method 2: ECMAScript 2015
function myFunction(x, y = 2) {
    // function code
}
```

65

65

Calling Functions with Parameters

- To call a function and pass values to its parameters:

  Use the function's name, followed by a list of expressions for each parameter

- Example:

```
printSign(-5);
printSign(balance);
printSign(2 + 3);
printMax(100, 200);
printMax(oldQuantity * 1.5, quantity * 2);
```

66

66

Defining and Using Function Parameters

The **find()** method returns the value of the first array element that passes a test function.

```
var numbers = [4, 9, 16, 25, 29];
var first = numbers.find(myFunction);

function myFunction(value, index, array) {
    return value > 18;
}
```

67

67

Functions Parameters

Example

Print the sign of a number

```
function printSign(number) {
    number = +number;

    if (number > 0) {
        console.log(`The number ${number} is positive.`);
    } else if (number < 0) {
        console.log(`The number ${number} is negative.`);
    } else {
        console.log(`The number ${number} is zero.`);
    }
}
```

68

68

Functions Parameters

Exercise

1. **Exercise 1:** Print the max between 2 numbers
2. **Exercise 2:** Printing Triangles

   Creating a program for printing triangles as shown below:

```
n = 6

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
n = 5

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

69

69

Functions Parameters

Exercise

1. **Exercise 1:**

   Print the max between 2 numbers

```
function printMax(x, y) {
    var max = x;

    if (max < y) {
        max = y;
    }

    console.log(`Maximal number: ${max}`);
}
```

70

70

Functions Parameters

Exercise

2. Exercise 2: Printing Triangles

```
function pringTriangle(n) {
    var line;
    n = +n;

    for (line = 1; line <= n; line += 1) {
        printLine(1, line);
    }

    for (line = n-1; line >= 1; line -= 1) {
        printLine(1, line);
}

function printLine(start, end) {
    var line = "", i;
    start = +start;
    end = +end;
    for (i = start; i <= end; i += 1){
        line += " " + i;
    }
    console.log(line);
}
```

71

71

Returning Values from Functions

Every function in JavaScript returns a value

- Returns **undefined** implicitly
- Can be set explicitly
- The **return value can be of any type**
  - Number, String, Object, Function
  - Examples:

```
var head = arr.shift();
var price = getPrice() * quantity * 1.20;
var noValue = arr.sort();
```

72

72

Defining Functions That Return a Value

- Functions can return any type of data:

  Number, String, Object, etc.
- Use **return keyword** to return a result

```
function multiply (firstNum, secondNum) {
    return firstNum * secondNum;
}

function sum (numbers) {
    var sum = 0, number;
    for(number of numbers){
        sum += number;
    }
    return sum;
}
```

73

73

What are Objects?

- **Software objects** model real-world objects or abstract concepts

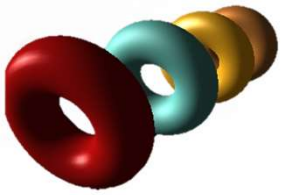
  Examples: bank, account, customer, dog, bicycle, queue
- **Real-world objects** have **states** and **behaviors**
  - Account states: holder, balance, type
  - Account behaviors: withdraw, deposit, suspend

74

74

What are Objects?

- How do software objects implement real-world objects?
  - Use variables/data/properties to implement states
  - Use methods/functions to implement behaviors
- An object is a software bundle of variables and related methods



75

75

Objects Represent

- Things from the real world
  - checks
  - people
  - shopping list
- Things from the computer world
  - numbers
  - characters
  - queues
  - arrays

76

76

What is an Object Type?

The formal definition of an object type:

Object types act as **templates** from which an instance of an object is created at run time. Types **define** the **properties** of the object and the **methods** used to control the object's behavior.

(Definition by Google)

77

77

Object Types

- Object Types provide the structure for objects

  Define their prototype, act as template
- Object Types define:
  - Set of **attributes**
    - ✓Represented by variables and properties
    - ✓Hold their **state**
  - Set of actions - their **behavior**

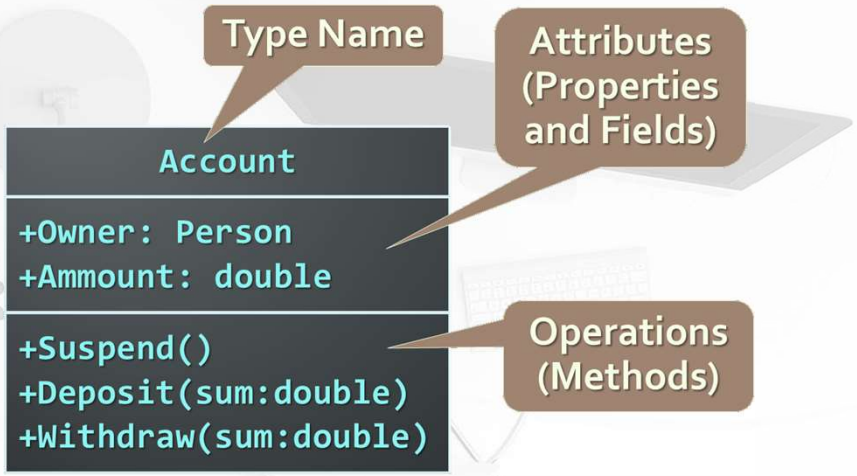
    Represented by methods
- A type defines the methods and types of data associated with an object

78

78

Object Types

Example



79

79

Objects

- An object is a **concrete instance** of a particular object type
- Creating an object from an object type is called **instantiation**
- Objects have state

  Set of values associated to their attributes
- Example:
  - Type: Account
  - Objects: Ivan's account, Peter's account

80

80

Exercise

1. Initialize an integer value N. Calculate the sum of intercept values less than N and output the result.

2. Initialize a array of fruits. Perform sorting and output results.
var fruits = ["orange", "banana","Chery","Mango","Apple"].

3. Given a string longer than 15 characters. Write a function to cut the string, retrieve the first 10 characters and add a "..." at the end of the string.