Web Programming

Lecturer: Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

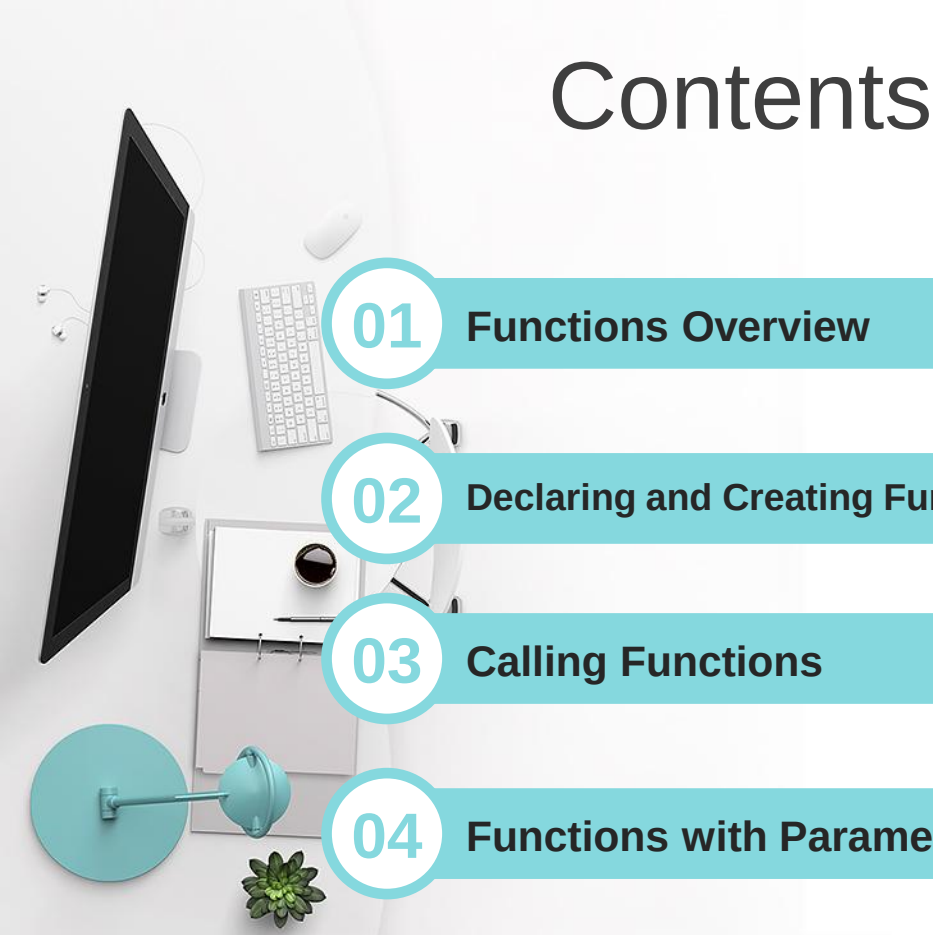
Content

- Lesson 1: Function
- Lesson 2: Object

Functions

Reusable parts of Code

Contents



1. Functions Overview

What is a function?

What is a Function?

- A function is a kind of building block that solves a small problem
  - **A piece of code** that **has a name** and can **be called from other code**
  - Can **take parameters** and **return a value**

- Functions allow programmers to construct large programs from simple pieces

Why to Use Functions?

- More **manageable** programming
  - **Split large** problems into **small pieces**
  - **Better organization** of the program
  - Improve code **readability** and **understandability**

- Avoiding repeating code

  Improve code **maintainability**

- Code **reusability**

  Using existing functions several times

2. Declaring and Creating Functions

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");
}
```

Ways of Defining a Function

Functions can be defined in 3 ways:

- By **function expression**

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

- By **function declaration**

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

Ways of Defining a Function

Functions can be defined in 3 ways:

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

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

  - Example:

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

3. Calling Functions

Executing the Function Code

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
```

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();
}
```

4. Functions with Parameters

Passing information to functions

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

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");
    }
}
```

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}`);
}
```

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
}
```

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);
```

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.`);
    }
}
```

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
```

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}`);
}
```

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);
}
```

5. The arguments Object

Access to all function parameters

arguments Object

Every function in JavaScript has an **implicit parameter (arguments)**

- It **holds information about the function** and **all the parameters** passed to the function
- No need to be explicitly declared

  It exists in every function

```
function printArguments() {
    var i;
    for(i in arguments) {
        console.log(arguments[i]);
    }
}
printArguments(1, 2, 3, 4); //1, 2, 3, 4
```

The arguments Object

- The arguments object is not an array

  It just has some of the array functionality

- If in need to iterate it, better parse it to an array:

```
function printArguments() {
    var i, args;

    args = [].slice.apply(arguments);
    for(i in args) {
        console.log(args[i]);
    }
}

printArguments(1, 2, 3, 4); //1, 2, 3, 4
```

6. Returning Values From Functions

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();
```

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;
}
```

The return Statement

- The return statement:
  - **Immediately terminates** function's execution
  - **Returns specified expression** to the caller

- To terminate function execution, use just:

  return;

- Return can be used several times in a function body

  To return a different value in different cases

The return Statement

Example

Check if a number is prime:

```
function isPrime(number) {
    var divider, maxDivider;

    number = +number;
    maxDivider = Math.sqrt(number);
    if (number < 2) return false;

    for(divider = 2; divider <= maxDivider; divider += 1) {
        if(number % divider === 0) {
            //Divider found, no need to continue execution;
            return false;
        }
    }

    //All dividers tested and none is found
    //The number is prime
    return true;
}
```

Exercise

Sum of Even Numbers

Calculate the sum of all even numbers in an array

```
function sum(numbers) {
    var number, sum = 0;

    for (number of numbers) {
        if (0 === number % 2) {
            sum += number;
        }
    }
    return sum;
}
```

7. Function Overloading

Many functions with the same name

Function Overloading

JavaScript does **not support** function overloading

i.e. functions with the same name hide each other

```
function print(number) {
    console.log(`Number: ${number}`);
}

function print(number, text) {
    console.log(`Number: ${number}\nText: ${text}`);
}

print(2);
```

Function Overloading in JavaScript

- Function overloading in JavaScript **must be faked**

  i.e. make it look like overloading

- Many ways of fake function overloading exist
  - Different number of parameters
  - Different type of parameters
  - Options parameter (preferred)

Function Overloading: Different Number of Parameters

A simple **switch by the length** of the arguments

```
function printText (number, text) {
    switch (arguments.length) {
        case 1 : console.log (`Number: ${number}`);
            break;
        case 2 :
            console.log (`Number: ${number}`);
            console.log (`Text: ${text}`);
            break;
    }
}

printText (5); //logs 5
printText (5, "Lorem Ipsum"); //logs 5 and Lorem Ipsum
```

Function Overloading: Different Types of Parameters

A **switch on the type** of the parameter

```
function printValue (value) {
    switch (typeof value) {
        case "number" : console.log(`Number: ${value}`); break;
        case "string" : console.log(`String: ${value}`); break;
        case "object" : console.log(`Object: ${value}`); break;
        case "boolean" : console.log(`Number: ${value}`); break;
    }
}

printValue (5);
printValue ("Lorem Ipsum");
printValue ([1, 2, 3, 4]);
printValue (true);
```

Function Overloading with Default Parameters

- In JavaScript, **all parameters are optional**

  i.e. functions can be invoked without them

- Yet, there is a reason behind requesting parameters

  Maybe the function's behavior depends on it?

- **Default parameters are checked** in the function body

  If the parameter is not present - assign a value

```
//only the str parameter is required
function getRandomValue(str, start, end) {
  start = start || 0;
  end = end || str.length;
  //function code
}
```

Function Overloading: Options parameter

- To create functions with options parameter
  - Create the function **take a single parameter**
  - **Each parameter is a property** of the options parameter

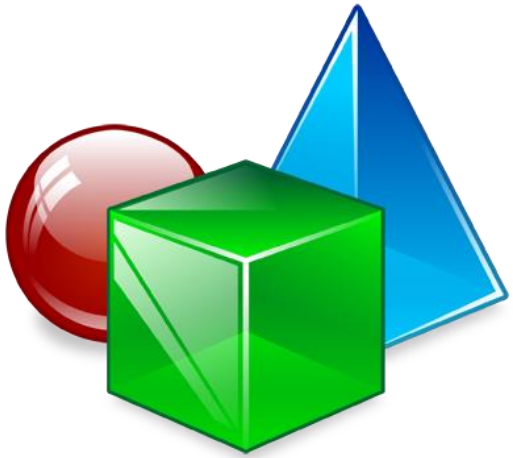
- Example:

```
function getRandomValue(opt) {
    var min = +opt.min || Number.MIN_VALUE;
    var max = +opt.max || Number.MAX_VALUE;

    return (Math.random() * (max - min + 1) + min) | 0;
}

console.log(getRandomValue({min: 0, max: 15}));
```

Using Objects

Objects, Properties, Primitive and Reference Types

Contents



1. Object Types and Objects

Modeling Real-world Entities with Objects

What are Objects?

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

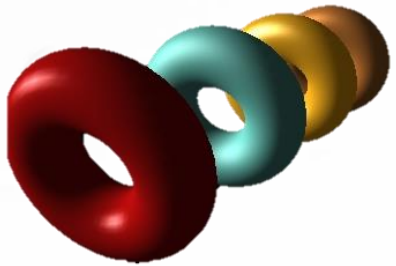
  Examples: bank, account, customer, dog, bicycle, queue

- **Real-world objects** have **states** and **behaviors**
  - Account states: holder, balance, type
  - Account behaviors: withdraw, deposit, suspend

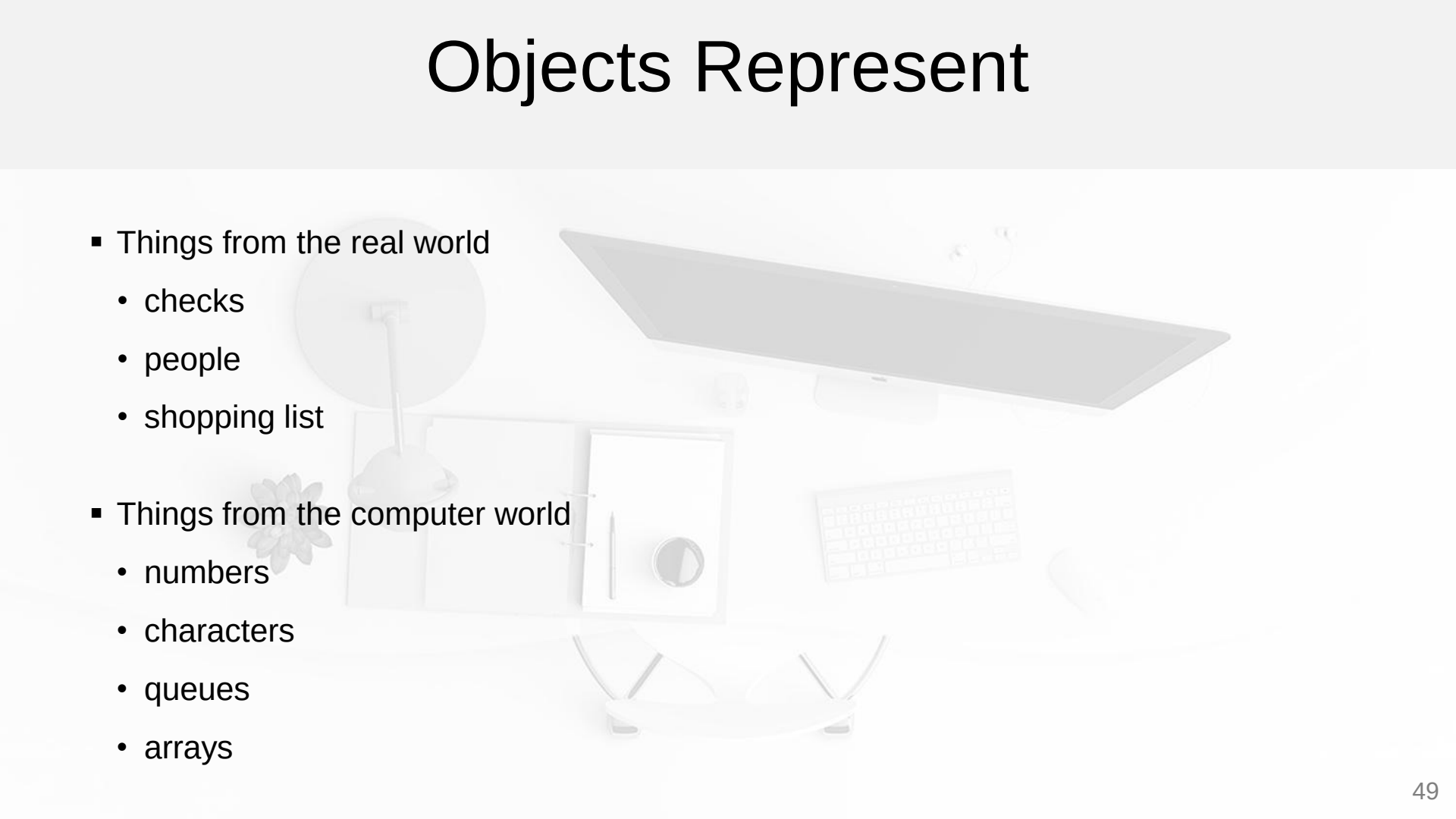
What are Objects?

- 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



Objects Represent



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

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)

Object Types

- Object Types provide the structure for objects

  Define their prototype, act as template
- Object Types define:
  - Set of **attributes**

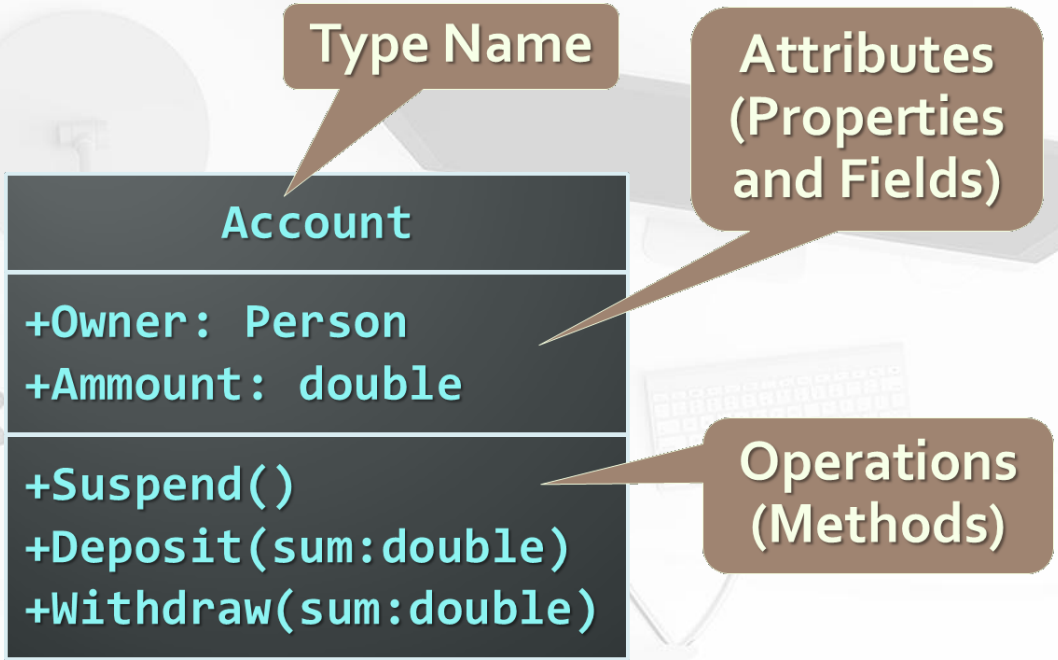
    ✓Represented by variables and properties

    ✓Hold their **state**
  - Set of actions - their **behavior**

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

Object Types

Example



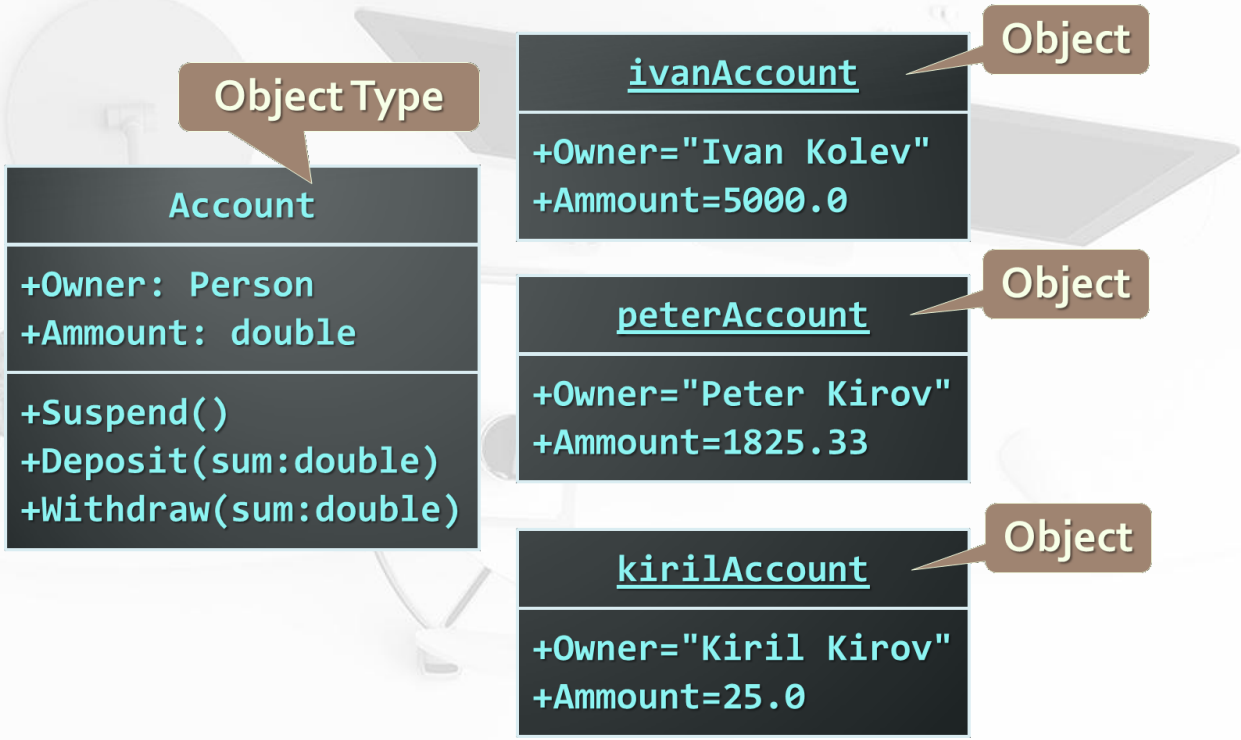
Objects

- 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

Objects

Example



2. Objects

Collection of fields and methods

Objects Overview

- JavaScript is designed on a simple object-based paradigm

  An object is a collection of **properties**

- An object property is association between a name and a value

  A value of property can be either a **method** (function) or a **field** (variable)

- Lots of predefined objects available in JavaScript

  Math, document, window, etc.

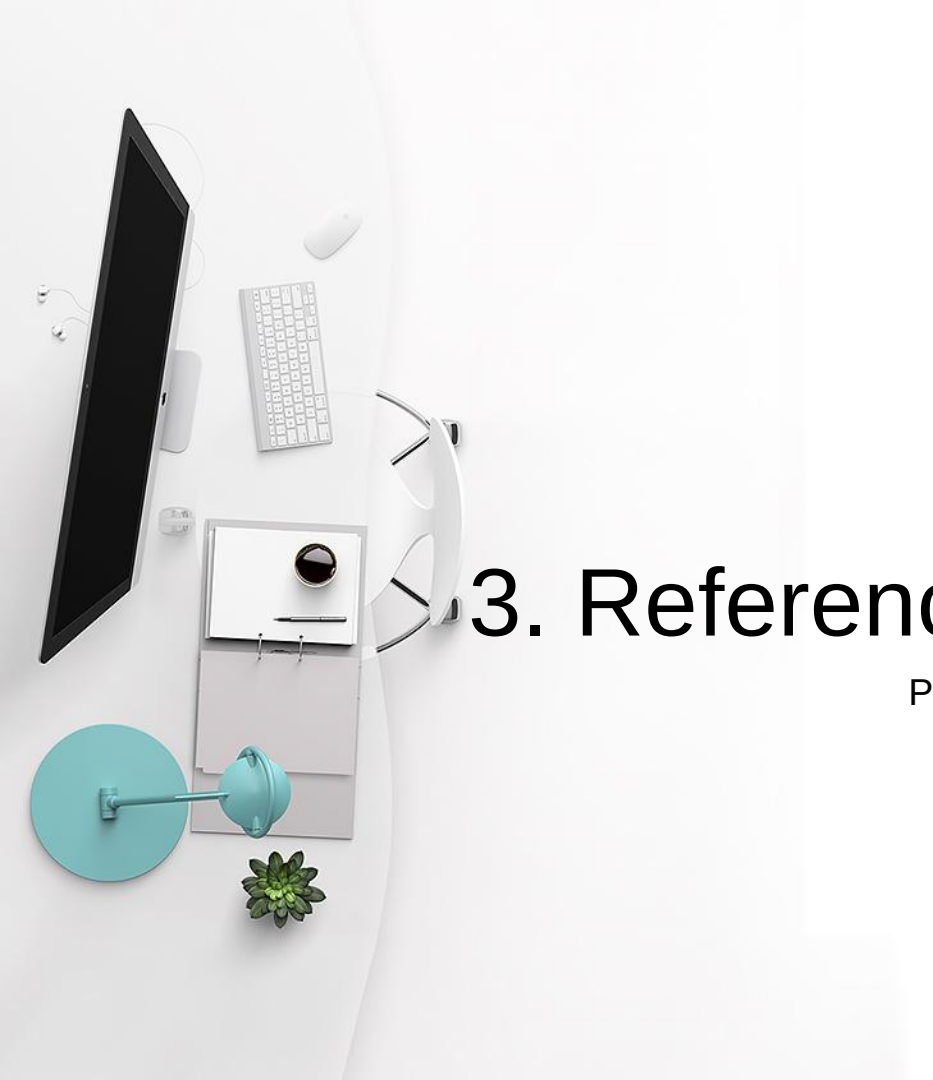
- Objects can be created by the developer

Object Properties

Each object has **properties**

- Properties are values attached to the object
- Properties of an object can be accessed with a **dot-notation** (. operator) or with **[] - indexer**:

```
let arrStr = arr.join(', '); // property join of Array
let length = arr.length;  // property length of Array
let words = text.split(' ');
let words = text['split'](' ');
```



3. Reference and Primitive Types

Passing by value, passing by reference

Reference and Primitive Types

- JavaScript is a **weakly typed** language

  Variables don’t have type, but their values do

- JavaScript has **six** different **types**:

  Number, String, Boolean, Null, Undefined and Object

- **Object** is the only **reference type**

  It is passed by **reference** (every time an object’s value is used, it’s used through a reference)

- **Number, String, Boolean, Null, Undefined** are **primitive** types

  Passed by **value** (they’re copied each time their value is used)

Reference and Primitive Types

- The primitive types are Boolean, Number, String, Undefined and Null

  All the other types are actually of type object

  Including arrays, dates, custom types, etc.

```
// all of those are true
console.log(typeof new Object() === typeof new Array());
console.log(typeof new Object() === typeof new Date());
console.log(typeof new Array() === typeof new Date());
```

- All types derive from Object

  Their type is object

Pass by value vs. Pass by reference

*pass by **reference***

cup =

fillCup( )

*pass by **value***

cup =

fillCup( )

www.penjee.com

Primitive Types

- Primitive types are passed **by value**

  When passed as argument

  - New memory is allocated
  - The value is copied in the new memory
  - The value in the new memory is passed

- Primitive types are initialized with type literals
- Primitive types have an object type **wrapper**

```
let number = 5, // Holds a primitive value of 5
    text = 'Hello there!', // Holds a primitive value
    numberObj = new Number(5); // Holds an object value of 5
```

Primitive Types

Example

Assign string values to two variables

- Create an object using their value
- Change the value of the variables
- Each object has its own value

```
let fname = 'Peter',
    lname = 'Johnson',
    person = { firstName: fname, lastName: lname };
lname = 'Peterson';
console.log(person.lastName) // Logged 'Johnson'
```

Reference Type

Object is the only **reference type**

When passed its value is used somewhere, it is not copied, but instead a reference to it is passed

```
let marks = [
        { subject : 'JavaScript', score : 4.50 },
        { subject : 'OOP', score : 5.00 },
        { subject : 'HTML5', score : 6.00 },
        { subject : 'Photoshop', score : 4.00 }
    ];

let student = { name: 'Doncho Minkov', marks: marks };
marks[2].score = 5.50;

console.log(student.marks);
// Logs 5.50 for HTML5 score
```

4. JavaScript Object Literal

Curly brackets {}

JavaScript Object Literal

JavaScript object literal is a simplified way to create objects

Using curly brackets:

```
let person = {
        firstName: 'Doncho',
        lastName: 'Minkov',
        toString: function () {
            return this.firstName + ' ' + this.lastName;
        }
    };

// object properties can be used:
console.log(person.toString());
// writes 'Doncho Minkov'
```

Creating Objects

- Let's make two people:

```
let minkov, georgiev;
minkov = {
    fname: 'Doncho',
    lname: 'Minkov',
    toString: function() {
        return this.fname + ' ' + this.lname;
    }
};
georgiev = {
    fname: 'Georgi',
    lname: 'Georgiev',
    toString: function() {
        return this.fname + ' ' + this.lname;
    }
};
```

- Object notations are great, but **repeating code** is not, right?

Object Building Function

- Using a function for building objects

  Just pass first and last name and get an object

  Something like a constructor

```
let minkov, georgiev;
function makePerson(fname, lname) {
    return {
        fname: fname,
        lname: lname,
        toString: function () {
            return this.fname + ' ' + this.lname;
        }
    }
}
minkov = makePerson('Doncho', 'Minkov');
georgiev = makePerson('Georgi', 'Georgiev');
```

- Much cooler, right?

5. JavaScript Object Properties

Dot-notation, associative arrays

JavaScript Object Properties

JavaScript objects are just a set of **key/value pairs**

- Each value can be accessed by its key
- Properties in objects are accessed using the **dot-notation** (**obj.property**)
- Yet properties can be used with brackets

  Like an array

```
document.write === document['write']
```

Associative Arrays

- Objects can be used as **associative arrays**
  - The key (index) is string instead of number
    - Also called dictionaries or maps
- Associative arrays don't have array properties
  - length, indexOf, etc.

```
function countWords(words) {
    let word,
        wordsCount = {};
    for (let i in words) {
        word = words[i].toLowerCase();
        if (!wordsCount[word]) { wordsCount[word] = 0; }
        wordsCount[word] += 1;
    }
    return wordsCount
}
```


Summary

Exercise

- **Exercise 1:** Write a function to show a 4-product list.

  Note: use layout in your Bootstrap Exercise
- **Exercise 2:** Write a function to delete a product in an array by product ID.


Q&A