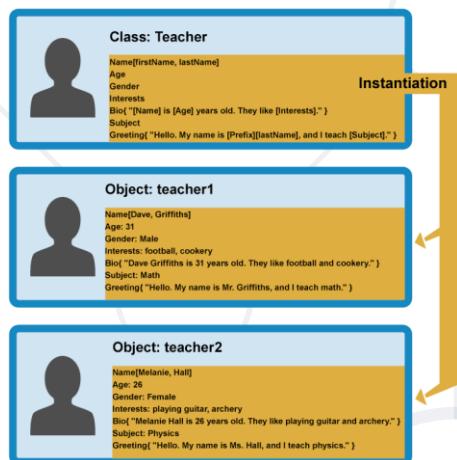


Objects and Classes

Using Objects and Classes
Defining Simple Classes



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

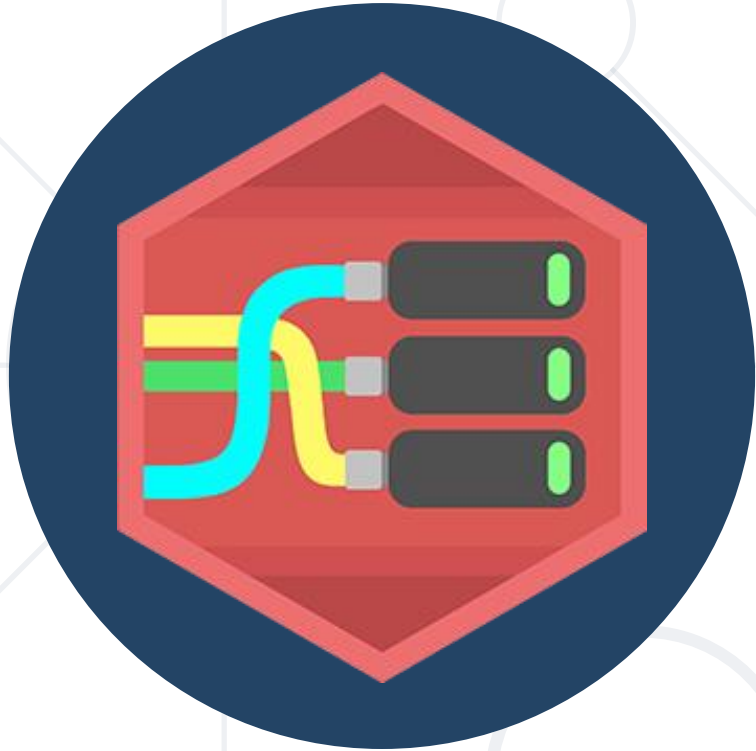
Table of Contents

1. Objects (definition, properties, and methods)
2. Reference vs. Value Types
3. Execution context (this)
4. JSON
5. Associative Arrays
6. Classes



sli.do

#js-front-end




Objects

Definition, Properties and Methods

What Are Objects ?

- **Structure** of related data or functionality
- Contains **values** accessed by **string keys**
 - Data values are called **properties**
 - Function values are called **methods**



Object	
'name'	'Peter'
'age'	20

Property name (key)

Property value

- You can **add** and **remove** properties **during runtime**

- We can create an object with an **object literal**

```
let person = { name: 'Peter', age: 20, height: 183 };
```

- We can define an **empty object** and **add properties** later

```
let person = {};  
person.name = 'Peter';  
person.age = 20;  
person.hairColor = 'black';
```

```
person['lastName'] = 'Parker';
```

Access and set
properties using
string indexing

Problem: Person Info

- Create an **object** that has a first name, last name, and age
- **Return** the object at the end of your function

"Peter",
"Pan",
20



firstName: Peter
lastName: Pan
age: 20

"Jack",
"Sparrow",
"unknown"



firstName: Jack
lastName: Sparrow
age: unknown

Solution: Person Info

- Create an object
- Set the properties **firstName**, **lastName**, and **age**
- Return the created object using the **return** keyword

```
function personInfo(firstName, lastName, age) {  
  let person = {};  
  person.firstName = firstName;  
  // TODO: Add other properties  
  return person;  
}
```


- Functions within a JavaScript object are called **methods**
- We can **define** methods using several syntaxes:

```
let person = {  
  sayHello: function() {  
    console.log('Hi, guys');  
  }  
}
```

```
let person = {  
  sayHello() {  
    console.log('Hi, guys');  
  }  
}
```

- We can **add** a method to an already defined object

```
let person = { name: 'Peter', age: 20 };  
person.sayHello = () => console.log('Hi, guys');
```

- Get array of all property **names** (keys)

```
Object.keys(cat); // ['name', 'age']
```

cat	
'name'	'Tom'
'age'	5

- Get array with of all property **values**

```
Object.values(cat); // ['Tom', 5]
```

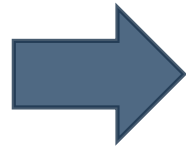
- Get and array of all properties as **key-value tuples**

```
Object.entries(cat); // [['name', 'Tom'], ['age', 5]]
```

Problem: City

- Receive an object, which holds **name**, **area**, **population**, **country**, and **postcode**
- Loop through all the keys and print them with their values

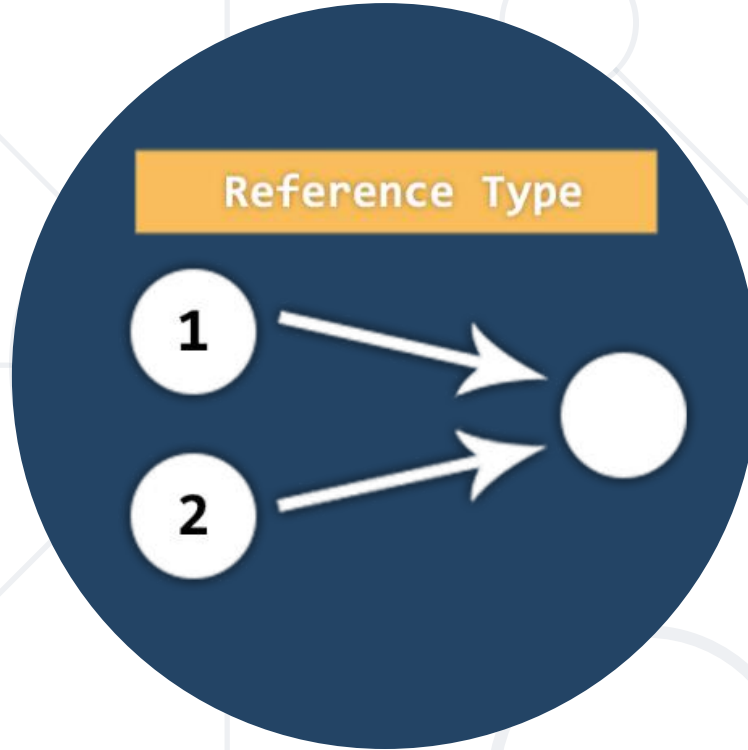
Sofia
492
1238438
Bulgaria
1000



name -> Sofia
area -> 492
population -> 1238438
country -> Bulgaria
postCode -> 1000

- Get the object **entries**
- Loop through the object **entries** using **for-of** loop
- Print the object **keys** and **values**

```
function cityInfo(city) {  
  let entries = Object.entries(city);  
  for (let [ key, value ] of entries) {  
    console.log(` ${key} -> ${value} `);  
  }  
}
```



Value vs. Reference Types

Memory Stack and Heap


Reference vs. Value Types

- JavaScript has 7 data types that are copied by **value**:
 - **Boolean, String, Number, null, undefined, Symbol, BigInt**
 - These are **primitive types**
- JavaScript has 3 data types that are copied by having their **reference** copied:
 - **Array, Objects, and Functions**
 - These are all technically Objects, so we'll refer to them collectively as Objects




Example: Reference vs. Value Types

pass by reference

cup = 

fillCup()

pass by value

cup = 

fillCup()

- If a primitive type is assigned to a variable, we can think of that variable as **containing** the primitive value

```
let a = 10;      let c = a;  
let b = 'abc';   let d = b;
```

- They are **copied by value**

```
console.log(a, b, c, d);  
// a = 10 b = 'abc' c = 10 d = 'abc'
```


- Variables that are assigned a non-primitive value are given a **reference** to that value

```
let arr = [];  
let arrCopy = arr;
```

- That reference **points to a location** in memory
- Variables** don't contain the value but **lead to the location**




Methods and Context

Combine Data with Behavior

Object Methods

- Objects can also have **methods**
- Methods are **actions** that can be performed on objects
- Methods are stored in **properties** as **function** definitions



```
let person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: function (myAge) {  
    return `My age is ${myAge}!`  
  }  
};  
console.log(person.age(21)); // My age is 21!
```

- Related functions may be **grouped** in an object
- The object serves as a **function library**
 - Similar to built-in libraries like **Math**, **Object**, **Number**, etc.

```
// sorting helper  
const compareNumbers = {  
  ascending: (a, b) => a - b;  
  descending: (a, b) => b - a;  
};
```

- This technique is often used to **expose public API** in a module

Objects as **switch** replacement

- You will **almost never** see **switch** used in JS code
- **Named methods** are used instead

```
let count = 5;  
switch (command) {  
  case 'increment':  
    count++;  
    break;  
  case 'decrement':  
    count--;  
    break;  
  case 'reset':  
    count = 0;  
    break;  
}
```



```
let count = 5;  
  
const parser = {  
  increment() { count++; },  
  decrement() { count--; },  
  reset() { count = 0; }  
}  
  
parser[command]();
```

Shorter syntax for
object methods

Accessing Object Context

- Functions in JavaScript have **execution context**
 - Accessed with the keyword **this**
 - When executed as an **object method**, the context is a reference to the **parent object**



```
const person = {
  firstName: 'Peter',
  lastName: 'Johnson',
  fullName() {
    return this.firstName + ' ' + this.lastName;
  }
};
console.log(person.fullName()); // 'Peter Johnson'
```

Function Execution Context

- Execution context can be **changed** at run-time
- If a function is **executed outside** of its parent object, it will **no longer** have access to the object's content



```
const getFullName = person.fullName;  
console.log(getFullName()); // 'undefined undefined'  
const anotherPerson = { firstName: 'Bob',  
                        lastName: 'Smith' };  
anotherPerson.fullName = getFullName;  
console.log(anotherPerson.fullName()); // 'Bob Smith'
```

- **Further lessons** will explore more **context features**!

Problem: City Taxes

- Extend **Problem 1: City Record**
 - Add property **taxRate** with initial value **10**
 - Add **methods**:
 - collectTaxes()** increase **treasury** by (**population** * **taxRate**)
 - applyGrowth(percent)** increase **population** by percentage
 - applyRecession(percent)** decrease **treasury** by percentage
 - All values must be **rounded down** after calculation

Solution: City Taxes

```
function createRecord(name, population, treasury) {  
  return {  
    name, population, treasury,  
    taxRate: 10,  
    collectTaxes() {  
      this.treasury += this.population * this.taxRate;  
    },  
    applyGrowth(percent) {  
      this.population += Math.floor(this.population * percent / 100);  
    },  
    applyRecession(percent) {  
      this.treasury -= Math.floor(this.treasury * percent / 100);  
    },  
  };  
}
```



JSON

JavaScript Object Notation

-



JSON Usage

- Exchange data between **browser** and **server**
- JSON is a **lightweight** format compared to XML
- JavaScript has built-in functions to **parse JSON** so it's easy to use
- JSON uses **human-readable** text to transmit data



JSON Example

Brackets define a JSON

Keys are in double quotes

Keys and values separated by :

```
{  
  "name": "Ivan",  
  "age": 25,  
  "grades": {  
    "Math": [2.50, 3.50],  
    "Chemistry": [4.50]  
  }  
}
```

It is possible to have nested objects

In JSON we can have arrays

- We can convert object into **JSON** string using **JSON.stringify(object)** method

```
let text = JSON.stringify(obj);
```

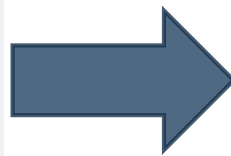
- We can convert JSON string into object using **JSON.parse(text)** method

```
let obj = JSON.parse(text);
```

Problem: Convert to Object

- Write a function, that receives a string in **JSON** format and converts it to object
- Print the entries of the object

```
'{  
  "name": "George",  
  "age": 40,  
  "town": "Sofia"  
'
```



```
name: George  
age: 40  
town: Sofia
```

Tips: Convert to Object

- Use **JSON.parse()** method to parse JSON string to an object
- Use **Object.entries()** method to get object's properties: names and values
- Loop through the entries and print them

```
function objConverter(json) {  
    // TODO: Use the tips to write the function  
}
```


Solution: Convert to Object

```
function objConverter(json) {  
  let person= JSON.parse(json);  
  
  let entries = Object.entries(person);  
  
  for (let [key, value] of entries) {  
    console.log(` ${key}: ${value}` );  
  }  
}
```

Problem: Convert to JSON

- Write a function that receives a first name, last name, hair color and sets them to an object.
- Convert the object to **JSON string** and print it.

```
'George',  
'Jones',  
'Brown'
```



```
{"name": "George", "lastName":  
"Jones", "hairColor": "Brown"}
```

Tips: Convert to JSON

- Create an object with the given input
- Use **JSON.stringify()** method to parse object to JSON string
- Keep in mind that the property name in the JSON string will be **exactly the same** as the property name in the object

```
function solve(name, lastName, hairColor){  
    // TODO: Use the tips and write the code  
}
```

Solution: Convert to JSON

```
function convertJSON(name, lastName, hairColor) {  
    let person = {  
        name,  
        lastName,  
        hairColor  
    };  
    console.log(JSON.stringify(person));  
}
```




Associative Arrays

Storing Key-Value Pairs

What is an Associative Array ?

- Arrays indexed by **string keys**
- Hold a set of pairs **[key => value]**
 - The key is a **string**
 - The **value** can be of **any** type



Key	Value
John Smith	+1-555-8976
Lisa Smith	+1-555-1234
Sam Doe	+1-555-5030

- An associative array in JavaScript is just an **object**
- We can declare it **dynamically**

```
let assocArr = {  
  'one': 1,  
  'two': 2,  
  'three': 3,  
  [key]: 6  
};
```

Quotes are used if the key contains **special characters**

```
assocArr['four'] = 4;
```

```
assocArr.five = 5;
```

```
let key = 'six';  
assocArr[key] = 6;
```

Valid ways to access **values** through **keys**

Using for – in

- We can use **for-in** loop to iterate through the keys

```
let assocArr = {};  
assocArr['one'] = 1;  
assocArr['two'] = 2;  
assocArr['three'] = 3;  
  
for(let key in assocArr) {  
    console.log(key + " = " + assocArr[key]);  
}
```

```
// one = 1  
// two = 2  
// three = 3
```



Problem: Phone Book

- Write a function that reads **names** and **numbers**
- Store them in an associative array and print them
- If the same name occurs, save the **latest** number

```
['Tim 0834212554',  
 'Peter 0877547887',  
 'Bill 0896543112',  
 'Tim 0876566344']
```



```
Tim -> 0876566344  
Peter -> 0877547887  
Bill -> 0896543112
```

Solution: Phone Book

```
function solve(input) {  
  let phonebook = {};  
  for (let line of input) {  
    let tokens = line.split(' ');  
    let name = tokens[0];  
    let number = tokens[1];  
    phonebook[name] = number;  
  }  
  for (let key in phonebook) {  
    console.log(`${key} -> ${phonebook[key]}`);  
  }  
}
```

Manipulating Associative Arrays

- Check if a key is **present**:

```
let assocArr = { /* entries */ };  
if (assocArr.hasOwnProperty('John Smith')) { /* Key found */ }
```

- **Remove** entries:

```
delete assocArr['John Smith'];
```

Problem: Meetings

- Write a function that reads **weekdays** and **names**
- Print a **success** message for every successful appointment
- If the same weekday occurs a second time, print a **conflict message**
- In end, print a list of all meetings
- See **example** input and output on **next slide**

Example: Meetings

- Parsing input and success/conflict messages

```
['Monday Peter',  
'Wednesday Bill',  
'Monday Tim',  
'Friday Tim']
```



```
Scheduled for Monday  
Scheduled for Wednesday  
Conflict on Monday!  
Scheduled for Friday
```

- Final list output

```
Monday -> Peter  
Wednesday -> Bill  
Friday -> Tim
```

Solution: Meetings

```
function solve(input) {  
  let meetings = {};  
  for (let line of input) {  
    let [weekday, name] = line.split(' ');  
  
    if (meetings.hasOwnProperty(weekday)) {  
      console.log(`Conflict on ${weekday}!`);  
    } else {  
      meetings[weekday] = name;  
      console.log(`Scheduled for ${weekday}`);  
    }  
  }  
  
  // TODO: Print result  
}
```

Sorting Associative Arrays


- Objects **cannot be sorted**; they must be converted first
 - Convert to **array** for **sorting**, **filtering** and **mapping**:

```
let phonebook = { 'Tim': '0876566344',  
                  'Bill': '0896543112' };  
  
let entries = Object.entries(phonebook);  
console.log(entries); // Array of arrays with two elements each  
// [ ['Tim', '0876566344'],  
//   ['Bill', '0896543112'] ]  
  
let firstEntry = entries[0];  
console.log(firstEntry[0]); // Entry key -> 'Tim'  
console.log(firstEntry[1]); // Entry value -> '0876566344'
```

The entry is turned into an array of **[key, value]**

- The **entries** array can be **sorted**, using a **Compare function**
 - To **sort by key**, use the **first element** of each entry

```
entries.sort((a, b) => {  
  keyA = a[0];  
  keyB = b[0];  
  // Perform comparison and return negative, 0 or positive  
});
```



- You can also **destructure** the entries

```
entries.sort(([keyA, valueA], [keyB, valueB]) => {  
  // Perform comparison and return negative, 0 or positive  
});
```


Problem: Sort Address Book

- Write a function that reads **names** and **addresses**
- Values will be separated by ":"
- If same name occurs, save the **latest** address
- Print list, **sorted** alphabetically by **name**

```
['Tim:Doe Crossing',  
'Bill:Nelson Place',  
'Peter:Carlyle Ave',  
'Bill:Ornery Rd']
```



```
Bill -> Ornery Rd  
Peter -> Carlyle Ave  
Tim -> Doe Crossing
```

Solution: Sort Address Book

```
function solve(input) {  
  let addressbook = {};  
  for (let line of input) {  
    let [name, address] = line.split(':');  
    addressbook[name] = address;  
  }  
  let sorted = Object.entries(addressbook);  
  sorted.sort((a, b) => a[0].localeCompare(b[0]));  
  // TODO: Print result  
}
```

- The **destructuring assignment** syntax makes it possible to unpack values from arrays, or properties from objects, into distinct variables.
- On the left-hand side of the assignment to define what values to unpack from the sourced variable.

```
const x = [1, 2, 3, 4, 5];  
const [y, z] = x;  
console.log(y); // 1  
console.log(z); // 2
```

```
obj = { a: 1, b: 2 };  
const { a, b } = obj;  
// is equivalent to:  
// const a = obj.a;  
// const b = obj.b;
```

- To **sort by value**, use the **second element** of each entry

```
entries.sort((a, b) => {  
  valueA = a[1];  
  valueB = b[1];  
  // Perform comparison and return negative, 0 or positive  
});
```

- You can also **destructure** the entries

```
entries.sort(([keyA, valueA],[keyB, valueB]) => {  
  // Perform comparison and return negative, 0 or positive  
});
```



Classes

Object Models

What are Classes?

- **Templates** for creating objects
- Defines **structure** and **behavior**
- An object created by the class pattern is called an **instance** of that class
- A class has a **constructor** – method called automatically to create an object
 - It **prepares** the new object for use
 - Can **receive** parameters and **assign** them to properties



Use the **class** keyword followed by a name

```
class Student {  
    constructor(name) {  
        this.name = name;  
    }  
}
```

The **constructor** is a special method for creating and initializing an object

- Creating a class:

this keyword is used to set a property of the object to a given value

```
class Student {  
  constructor(name, grade) {  
    this.name = name;  
    this.grade = grade;  
  }  
}
```

- Creating an **instance** of the class:

```
let student = new Student('Peter', 5.50);
```


- Classes can also have functions as property, called **methods**:

```
class Dog {  
  constructor(name) {  
    this.name = name;  
  }  
  
  speak() {  
    console.log(`${this.name} says Woof!`);  
  }  
}
```

this in the object
refers to itself

```
let dog = new Dog('Sparky');  
dog.speak(); // Sparky says Woof!
```

We access the
method as a regular
property

Problem: Cat

- Write a function that receives **array of strings** in the following format:
`'{cat name} {age}'`
- Create a class **Cat** that receives the **name** and the **age** parsed from the input
- It should also have a method named **meow()** that will print
`"{cat name}, age {age} says Meow"` on the console
- For each of the strings provided you must create a cat object

```
['Mellow 2', 'Tom 5']
```



```
Mellow, age 2 says Meow  
Tom, age 5 says Meow
```

- Create a class
- Set properties name and age
- Set property 'meow' to be a method that prints the result
- **Parse** the input data
- Create all objects using the class **constructor** and the parsed input data and store them in an array
- Loop through the array using **for...of** loop and invoke **.meow()** method

Solution: Cat

```
function catCreator(arr) {  
  // TODO: Create the Cat class  
  let cats = [];  
  for (let i = 0; i < arr.length; i++) {  
    let catData = arr[i].split(' ');  
    cats.push(new Cat(catData[0], catData[1]));  
  }  
  // TODO: Iterate through cats[] and invoke .meow()  
  // using for...of loop  
}
```



Live Exercises

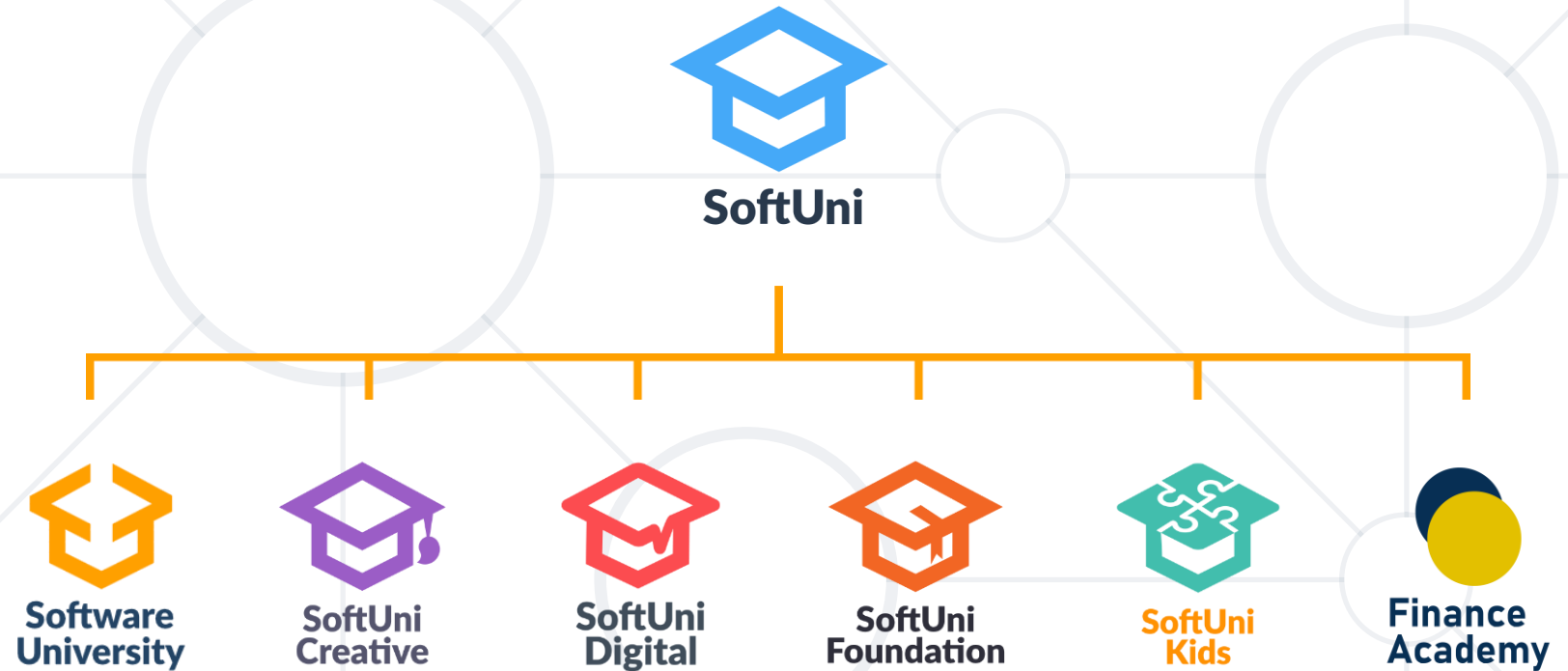
- Objects hold **key-value pairs**
 - Access value by indexing with key
 - **Methods** are functions
- **References** point to data in memory
- **Parse** and **stringify** objects in **JSON**
- **Classes** are templates for objects



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg
 - Software University Foundation
 - softuni.foundation
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- Software University Forums
 - forum.softuni.bg



Questions?



SoftUni Diamond Partners

**SUPER
HOSTING
.BG**



**Coca-Cola HBC
Bulgaria**



POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers



AMBITIONED


**DRAFT
KINGS**



**SOFTWARE
GROUP**

createX



Postbank

Решения за твоето утре

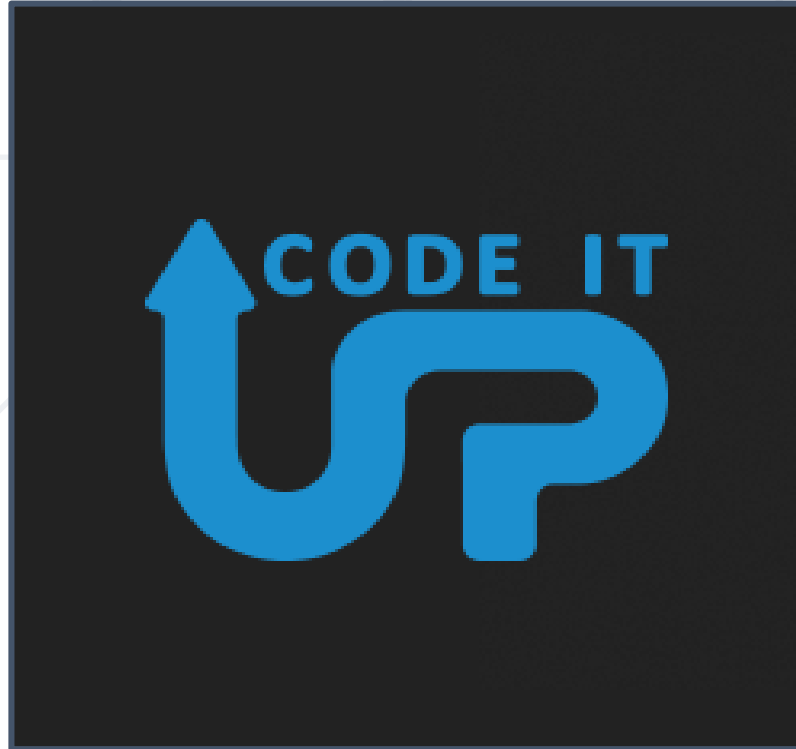


BOSCH

DXC
TECHNOLOGY



SmartIT



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

