

# Web Developer

Programmazione - Javascript e Typescript

Docente: Shadi Lahham

# Objects

Flexible data structures

Shadi Lahham - Web development

Objects

# Objects

Objects are a data type that let us store a collection of properties and methods

```
let objectName = {  
  propertyName: propertyValue,  
  propertyName: propertyValue,  
  ...  
};
```

```
let aboutMe = {  
  city: 'Madrid, ES',  
  hair: 'brown'  
};
```

# Objects

Objects are a data type that let us store a collection of properties and methods

```
let objectName = {  
  propertyName: propertyValue,  
  propertyName: propertyValue,  
  ...  
};
```

```
let myCat = {  
  age: 8,  
  furColor: 'grey',  
  likes: ['water', 'milk', 'meat'],  
  birthday: {'month': 7, 'day': 17, year: 1994}  
};
```

# Objects Access - dot notation

```
let aboutMe = {  
  city: 'Madrid, ES',  
  hair: 'brown'  
};
```

*// access properties using 'dot notation'*

```
let myCity = aboutMe.city;
```

*// non-existent properties will return undefined*

```
let myGender = aboutMe.gender;
```

# Objects Access - bracket notation

```
let aboutMe = {  
  city: 'Madrid, ES',  
  hair: 'brown'  
};
```

*// access properties using 'bracket notation'*

```
let myHair = aboutMe['hair'];
```

*// non-existent properties will return undefined*

```
let myGender = aboutMe['gender'];
```

*// properties can also be accessed with variable keys*

```
let myProperty = 'hair';
```

```
let myHair = aboutMe[myProperty];
```

# Changing Objects

Use dot or bracket notation with the assignment operator to change objects

```
let aboutMe = {  
  city: 'Madrid, ES',  
  hair: 'brown'  
};
```

Change existing properties:

```
aboutMe.hair = 'blue';
```

Or add new properties:

```
aboutMe.gender = 'female';
```

You can also delete properties:

```
delete aboutMe.gender;
```



# Arrays of Objects

*// since arrays can hold any data type, they can also hold objects*

```
let myCats = [
  {name: 'Cleo',
   age: 8},
  {name: 'Simba',
   age: 1}
];

for (let i = 0; i < myCats.length; i++) {
  let myCat = myCats[i];
  console.log(myCat.name + ' is ' + myCat.age + ' years old.');
```

# Objects as Arguments

*// just like other data types, objects can be passed into functions*

```
let myCat = {  
  age: 8,  
  furColor: 'grey',  
  likes: ['water', 'milk', 'meat'],  
  birthday: {'month': 7, 'day': 17, 'year': 1994}  
};  
  
function describeCat(cat) {  
  console.log('This cat is ' + cat.age + ' years old with ' + cat.furColor + ' fur.');}  
  
describeCat(myCat);
```

# Object methods

*// object properties can also be functions. Object functions are called 'methods'*

```
let myCat = {  
  age: 8,  
  furColor: 'grey',  
  meow: function() {  
    console.log('meowww');  
  },  
  eat: function(food) {  
    console.log('Yum, I love ' + food);  
  },  
  sleep: function(numMinutes) {  
    for (let i = 0; i < numMinutes; i++) {  
      console.log('z');  
    }  
  }  
};
```

# Object methods

*// invoke object methods using dot notation*

```
myCat.meow();  
myCat.eat('brown mushy stuff');  
myCat.sleep(10);
```

# This in methods

*// when a method is invoked, `this` refers to the object containing the method.*

```
let myCat = {  
  age: 8,  
  name: 'Cleo',  
  sayName: function() {  
    console.log('I am ' + this.name);  
  },  
};
```

```
myCat.sayName();
```

# Object.keys()

*// object.keys() Lists all the property names of an object in an array*

```
let myCat = {  
  age: 8,  
  furColor: 'grey',  
  meow: function() {  
    console.log('meowww');  
  },  
  sleep: function(numMinutes) {  
    for (let i = 0; i < numMinutes; i++) {  
      console.log('z');  
    }  
  }  
};
```

```
Object.keys(myCat); // ['age', 'furColor', 'meow', 'sleep']
```

# Built-in Objects

Javascript has a lot of useful built-in objects

## Array

`Array.isArray()`

## Number

`Number()`

`Number.parseInt()`

`Number.parseFloat()`

## Date

`Date.UTC()`

`Date.now()`

`Date.parse()`

## Math

Many useful functions



# In depth

Detailed usage



# Copying by value or by reference

```
// copying by value (primitives)
```

```
let a = 5;
```

```
let b = a; // b is now a copy of a
```

```
b = 10; // changing b won't affect a
```

```
console.log(a); // output: 5
```

```
console.log(b); // output: 10
```

```
// copying by reference (objects)
```

```
let obj1 = { name: 'John' };
```

```
let obj2 = obj1; // obj2 is now a reference to obj1
```

```
obj2.name = 'Jane'; // changing obj2 will also affect obj1
```

```
console.log(obj1.name); // output: Jane
```

```
console.log(obj2.name); // output: Jane
```

# Passing parameters: value vs. reference

```
function updateValues(num, obj, arr) {  
  num = 10;  
  obj.name = 'Jane';  
  arr.push(4);  
}
```

```
let a = 5;  
let person = { name: 'John' };  
let numbers = [1, 2, 3];
```

```
updateValues(a, person, numbers);
```

```
console.log(a); // output: 5 - unchanged because passed by value  
console.log(person.name); // output: jane - changed because passed by reference  
console.log(numbers); // output: [1, 2, 3, 4] - changed because passed by reference
```

# Invoke with bracket notation

```
let car = {  
  startEngine: function () {  
    console.log('Engine started.');  }  
};
```

```
// invoking the method using bracket notation  
car['startEngine']();
```

# Invoke with bracket notation

```
let car = {  
  startEngine: function (fuelType) {  
    console.log('Engine started. Fuel type: ' + fuelType);  
  },  
  stopEngine: function () {  
    console.log('Engine stopped.');  }  
};
```

*// invoking the methods using bracket notation*

```
car['startEngine']('gasoline');  
car['stopEngine']();
```

*// invoking the methods using dot notation*

```
car.startEngine('gasoline');  
car.stopEngine();
```

# Immutable Objects

*// myCat is a constant, but it's mutable so its properties can change*

```
const myCat = {  
  name: 'Cleo',  
  age: 8  
};
```

```
myCat.name = 'Adam';  
console.log(myCat.name); // 'Adam' not 'Cleo'
```

```
myCat = {}; // TypeError: Assignment to constant variable
```

# Immutable Objects

*// mutable objects can be changed over time, while immutable ones cannot be altered after creation*

```
const myCat = {  
  name: 'Cleo',  
  age: 8  
};
```

```
Object.freeze(myCat);
```

```
myCat.name = 'Sam';  
console.log(myCat.name); // 'Cleo' not 'Sam'
```

[Immutable - MDN](#)

[Object.freeze\(\)](#)

[JavaScript Immutability](#)

Your turn

# 1.Recipe

- Create an object to hold information on your favorite recipe. It should have properties for title (a string), servings (a number), and ingredients (an array of strings).
- On separate lines (one console.log statement for each), log the recipe information

**Bonus:** Create an array that holds several recipes and log them all



## 2. Reading list

- Create an array of objects, where each object describes a book and has properties for the title (a string), author (a string), and alreadyRead (a boolean indicating if you read it yet).
- Iterate through the array of books. For each book, log the book title and book author like so: "The Hobbit by J.R.R. Tolkien".
- Now use an if/else statement to change the output depending on whether you read it yet or not. If you read it, log a string like 'You already read "The Hobbit" by J.R.R. Tolkien', and if not, log a string like 'You still need to read "The Lord of the Rings" by J.R.R. Tolkien.'

## 3.Movie database

- Create an object to store the following information about a movie: title (a string), duration (a number), and stars (an array of strings).
- Create an Array of objects that can hold several movies.
- Create a function to print out the movie information like so: "Puff the Magic Dragon lasts for 30 minutes. Stars: Puff, Jackie, Living Sneezes."
- Test the function by printing one movie.
- Use the function to print all the movies in the Array.

## 4.Cash register

- Write a function called cashRegister that takes a shopping cart object.
- The object contains item names and prices (itemName: itemPrice).
- The function returns the total price of the shopping cart, e.g. :

// Input

```
let cartForParty = {  
  banana: "1.25",  
  handkerchief: ".99",  
  Tshirt: "25.01",  
  apple: "0.60",  
  nalgene: "10.34",  
  proteinShake: "22.36"  
};
```

// Output

```
cashRegister(cartForParty)); // 60.55
```

## 5.Credit card validation

- Write a function called “validateCreditCard” that checks credit card numbers according to the following rules:
  - Number must be 16 digits, all of them must be numbers
  - You must have at least two different digits represented (all of the digits cannot be the same)
  - The final digit must be even
  - The sum of all the digits must be greater than 16

Continues on next page >>>

## 5.Credit card validation

- The following credit card numbers are valid:
  - 9999-9999-8888-0000
  - 6666-6666-6666-1666
- The following credit card numbers are invalid:
  - a923-3211-9c01-1112 invalid characters
  - 4444-4444-4444-4444 only one type of number
  - 1111-1111-1111-1110 sum less than 16
  - 6666-6666-6666-6661 odd final number
- Hint
  - Remove the dash '-' from the input string before checking if the input credit card number is valid

Continues on next page >>>

## 5. Credit card validation

Call the function with several credit card numbers:

```
validateCreditCard('9999-9999-8888-0000');  
validateCreditCard('4444-4444-4444-4444');  
validateCreditCard('6666-6666-6666-1666');
```

The function returns an object saying that the credit card is valid, or what the error is:

```
{ valid: true, number: '9999-9999-8888-0000' }  
{ valid: false, number: 'a923-3211-9c01-1112', error: 'wrong_length' }
```

For each card check, print out the result to the log in this format:

```
=====  
= number : a923-3211-9c01-1112 =  
= valid   : false              =  
= error   : wrong length       =  
=====
```

# References

[JavaScript Objects](#)

[Working with objects - JavaScript](#)

More advanced:

[JavaScript Object Methods](#)

# References

[JavaScript Math Object](#)

[JavaScript Number Methods](#)