JS Objects

Objects as associative arrays or dictionaries

Agenda

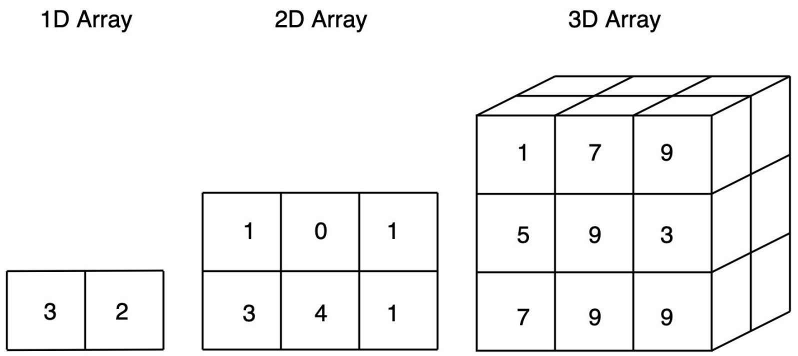
- Multi Dimensional Arrays
- Functions
- Objects
- Questions

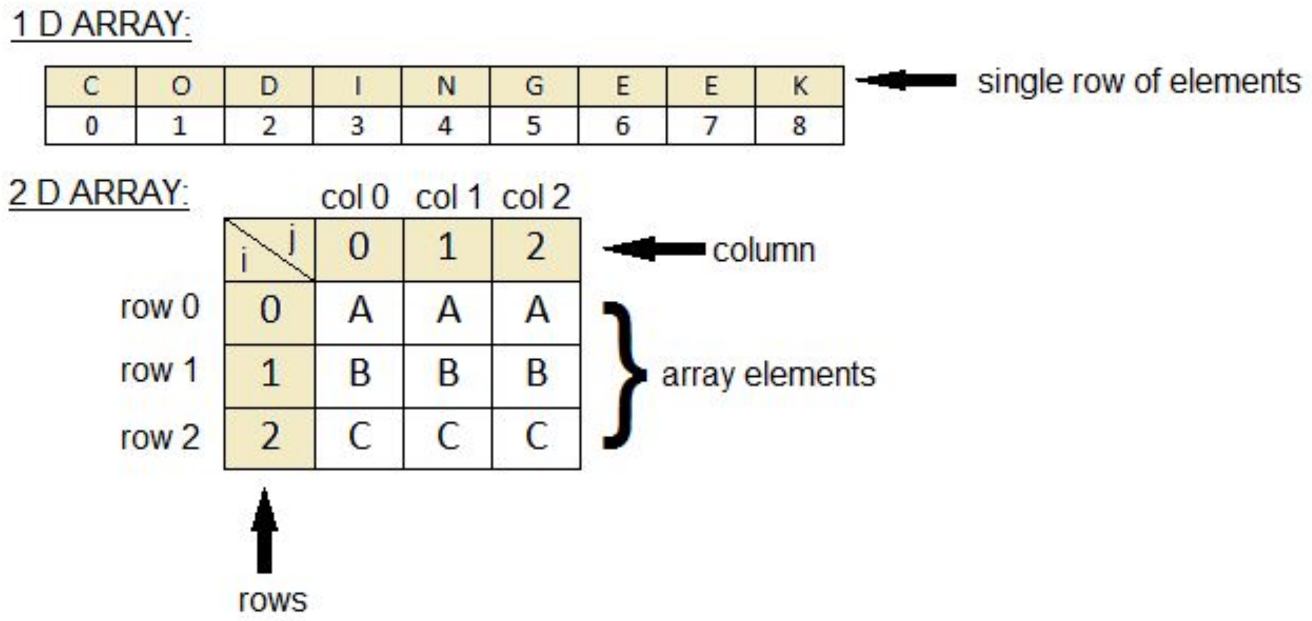
Learn Objectives

- You understand why a multidimensional array is needed
- You know how to iterate over a multidimensional array
- You know how to separate functions from main logic (different files) and why
- You know how to handle complex data better by using javascript objects

Multidimensional Arrays

1D, 2D, 3D, ... Worlds




1 D ARRAY:

C	O	D	I	N	G	E	E	K
0	1	2	3	4	5	6	7	8

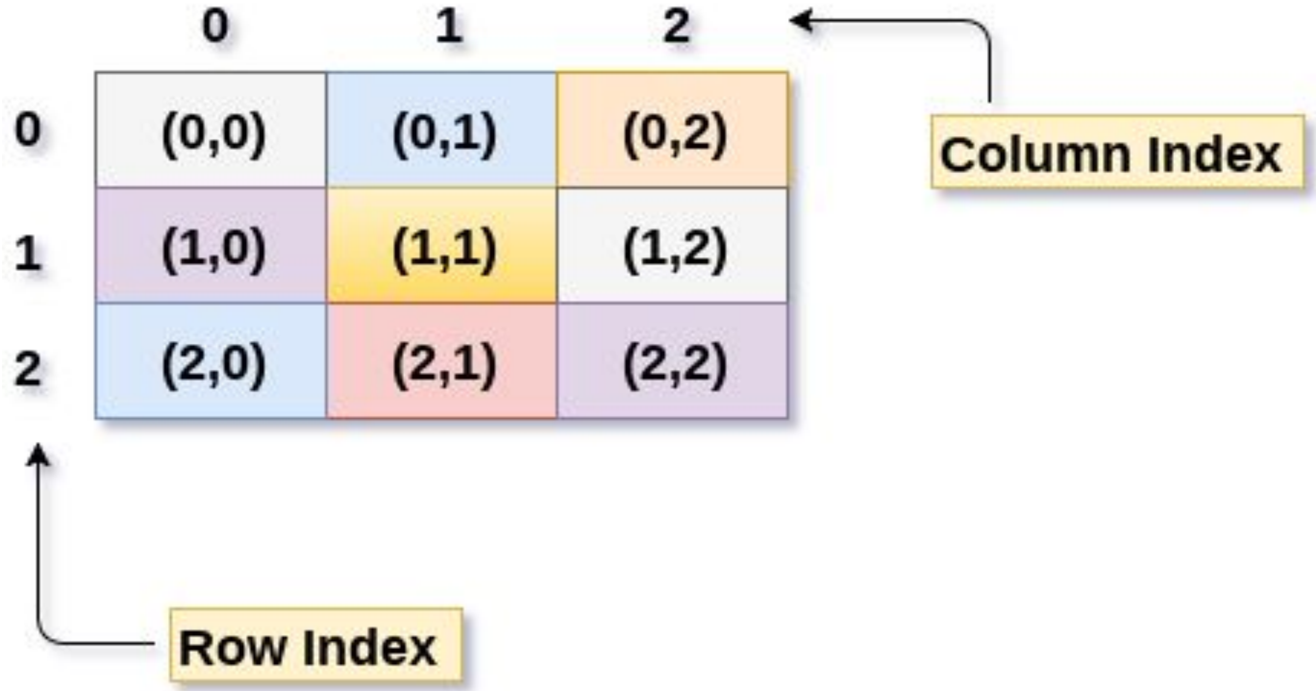
single row of elements

2 D ARRAY:

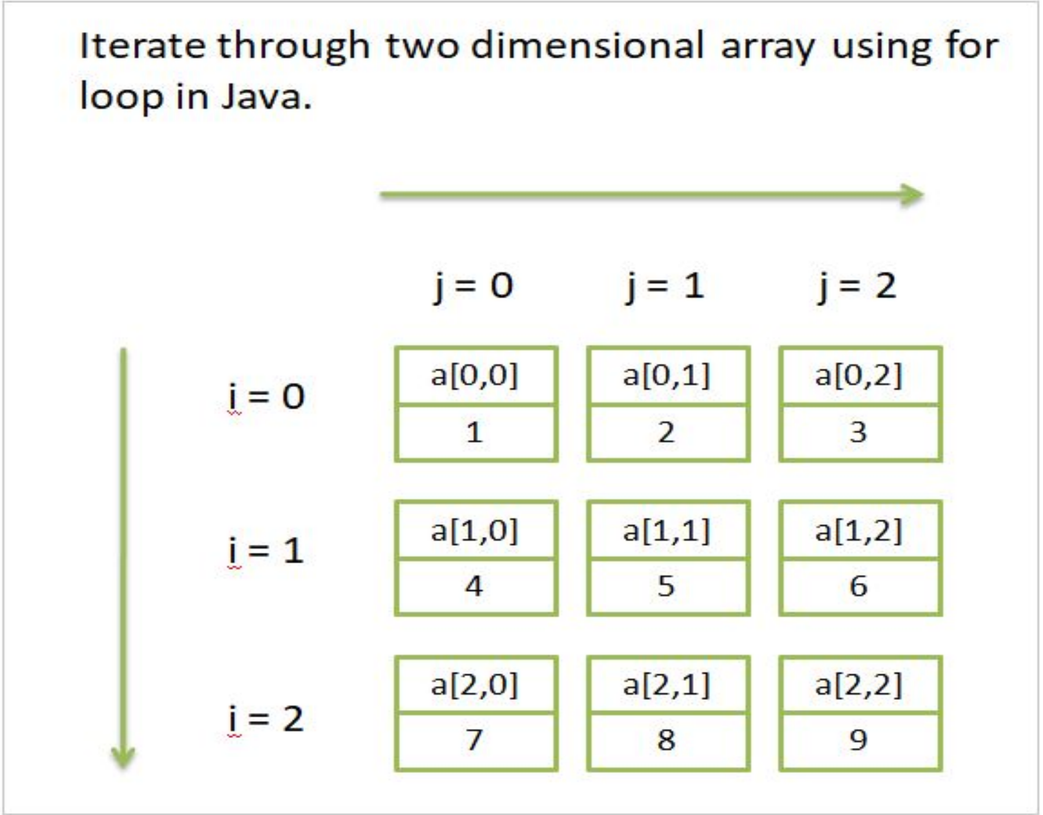
		col 0	col 1	col 2
	i \ j	0	1	2
row 0	0	A	A	A
row 1	1	B	B	B
row 2	2	C	C	C

column
array elements
rows

1D vs 2D vs 3D



access data in multidimensional array
(0,1) → array[0][1]



How to iterate a multidimensional array?

Where to use them?

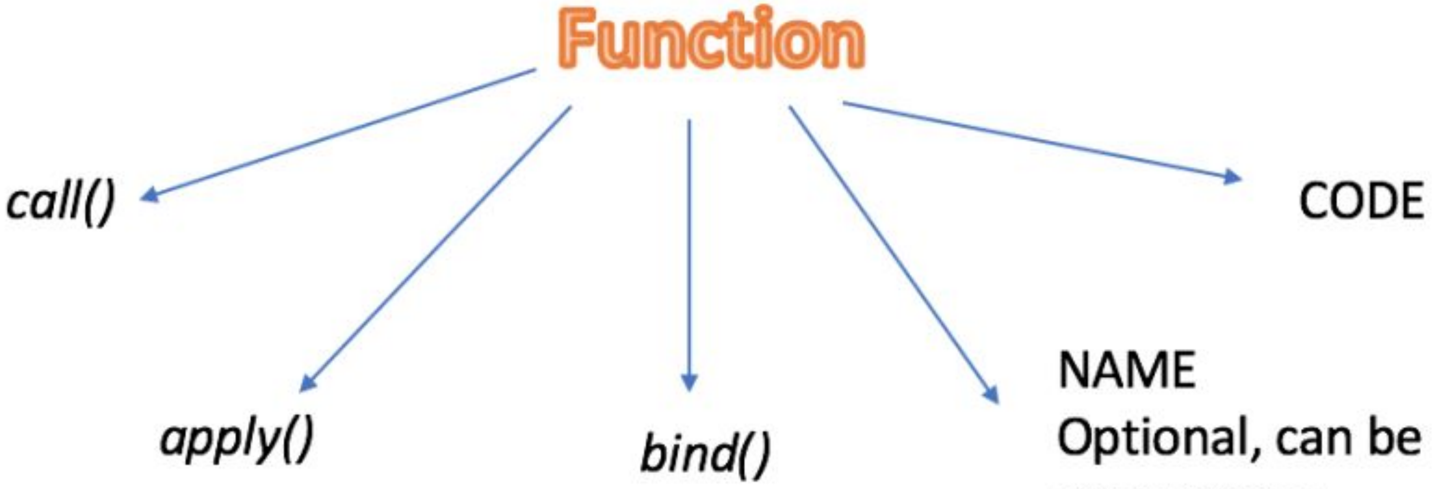
- As nested list; a list of lists
    - lecture notes of students
    - employee names of departments
- As a data model to a complex problem like;
    - matrix calculations
    - drawing a maze
    - showing a puzzle

functions

Subprograms

How to modularize them?

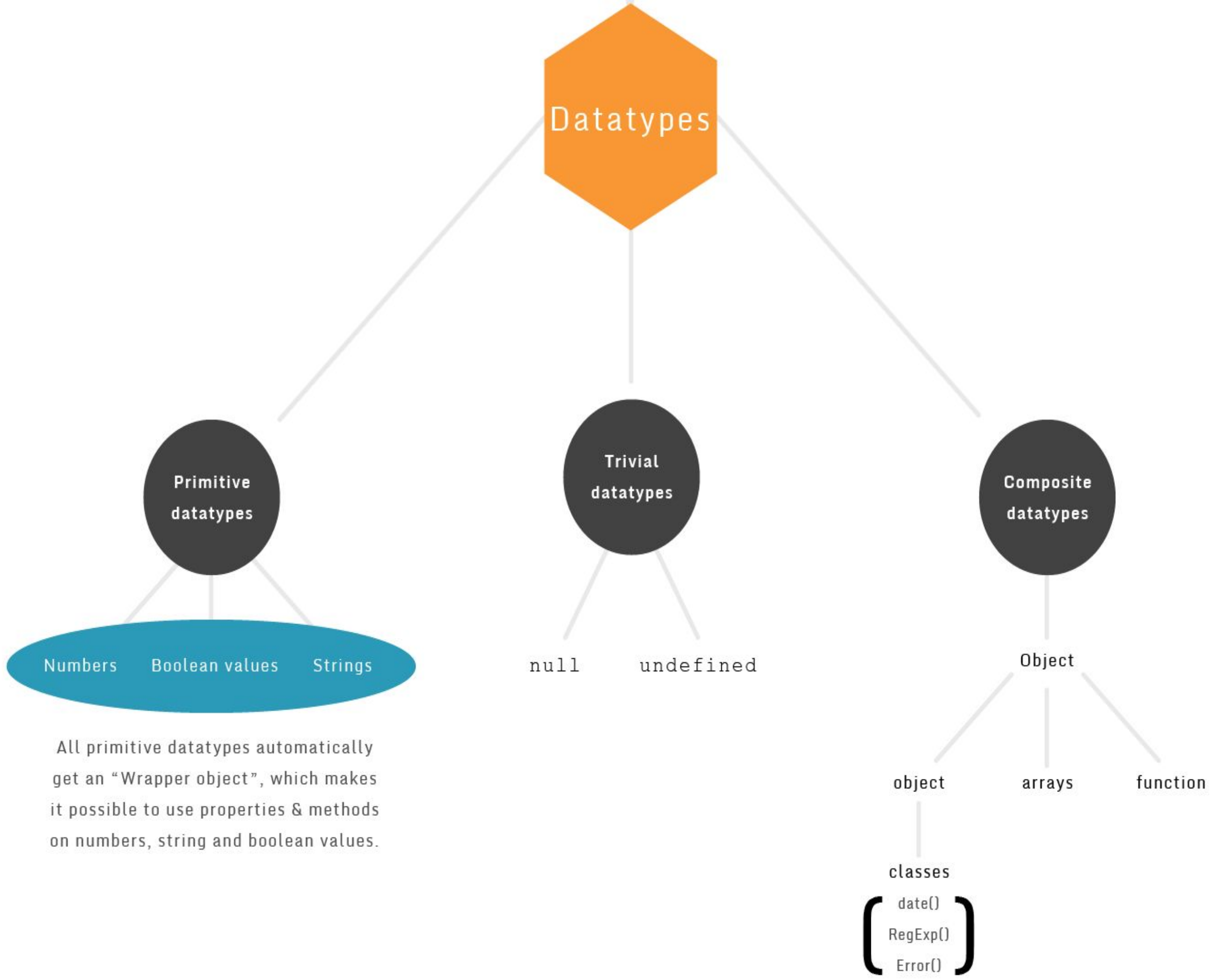
- Composition of functions
- A unit of work
- Separation of functions from the main code body
- Function libraries



Function as an object has some methods.

Objects – complex data holders

Objects

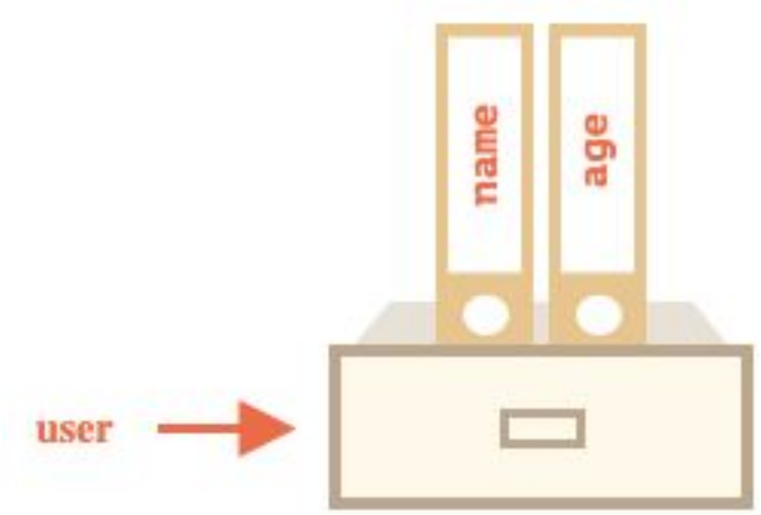


All primitive datatypes automatically get an "Wrapper object", which makes it possible to use properties & methods on numbers, string and boolean values.

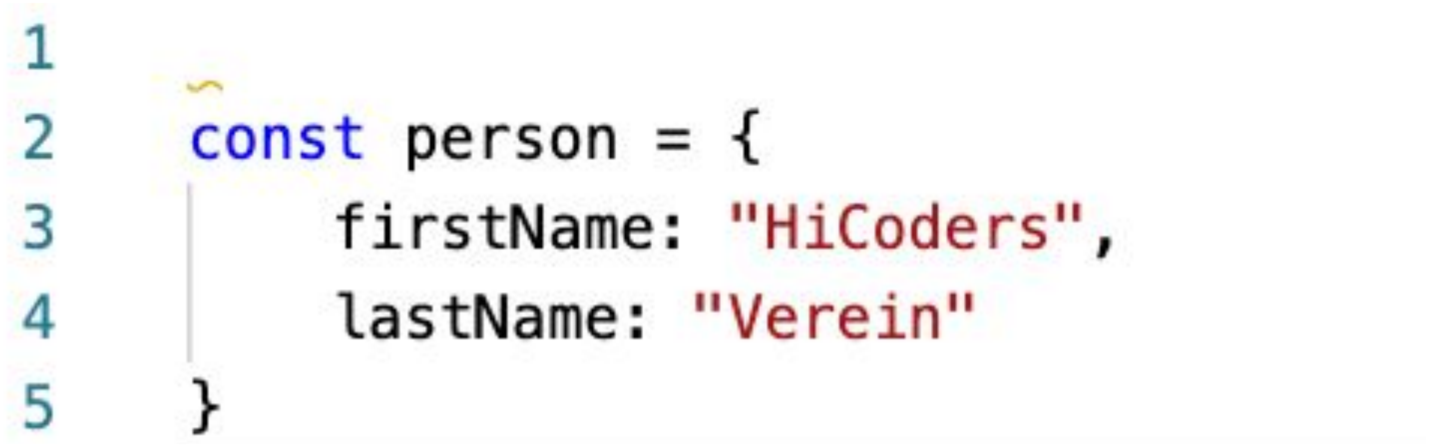
Objects

keep different data types together...

- Use etiquettes (key-value pairs) for easy handling
- It is easily understandable
- used for better isolation of the data belonging together.
- **used to create a better data model for solving the problem.**



Use key value pairs (etiquettes)



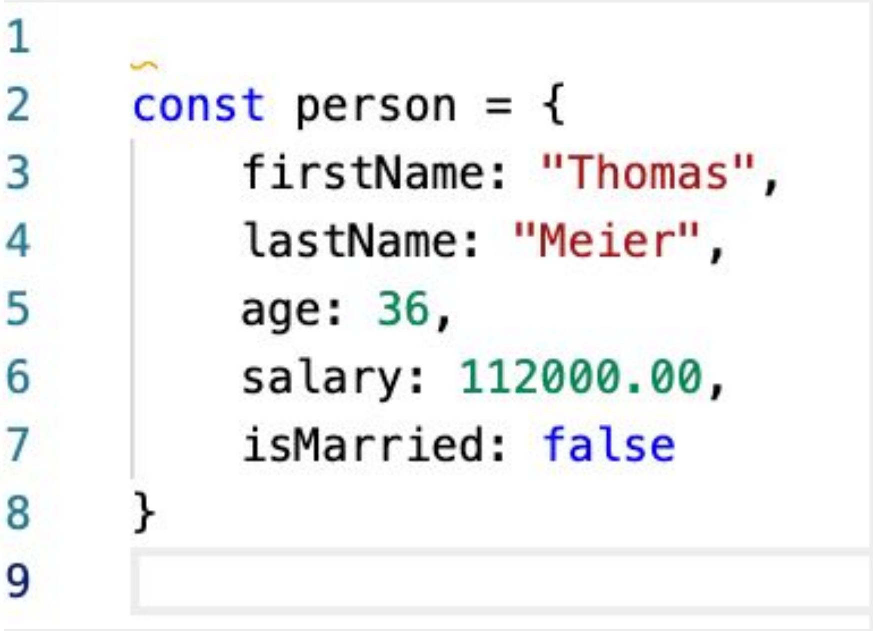
```
const person = {
    firstName: "HiCoders",
    lastName: "Verein"
}
```

How to define a javascript object?

Objects

can include primitive data types

- String
- Number
- Boolean



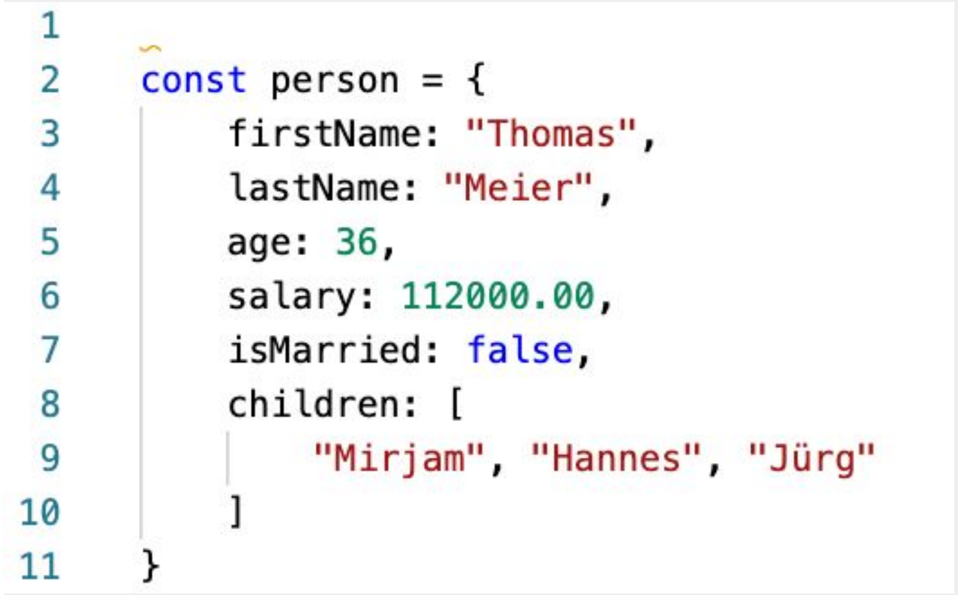
```

const person = {
    firstName: "Thomas",
    lastName: "Meier",
    age: 36,
    salary: 112000.00,
    isMarried: false
}

```

Objects

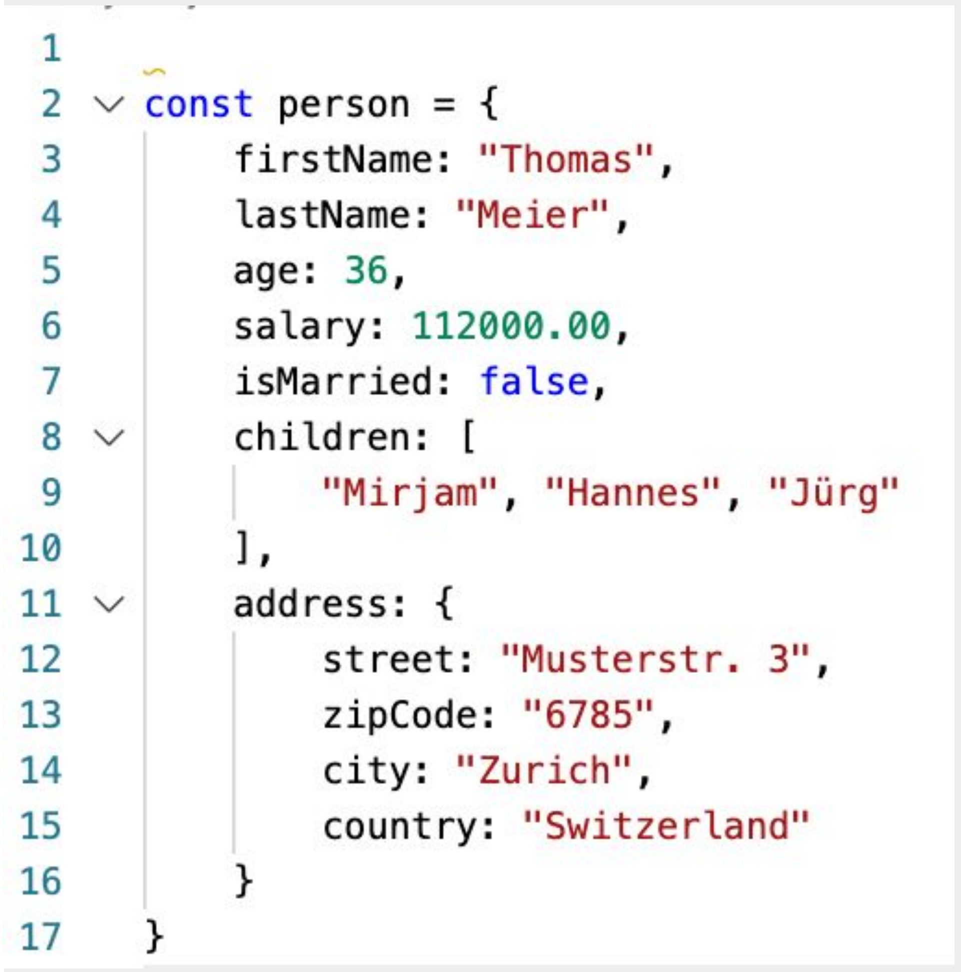
can contain arrays



```
const person = {
    firstName: "Thomas",
    lastName: "Meier",
    age: 36,
    salary: 112000.00,
    isMarried: false,
    children: [
        "Mirjam", "Hannes", "Jürg"
    ]
}
```

Objects

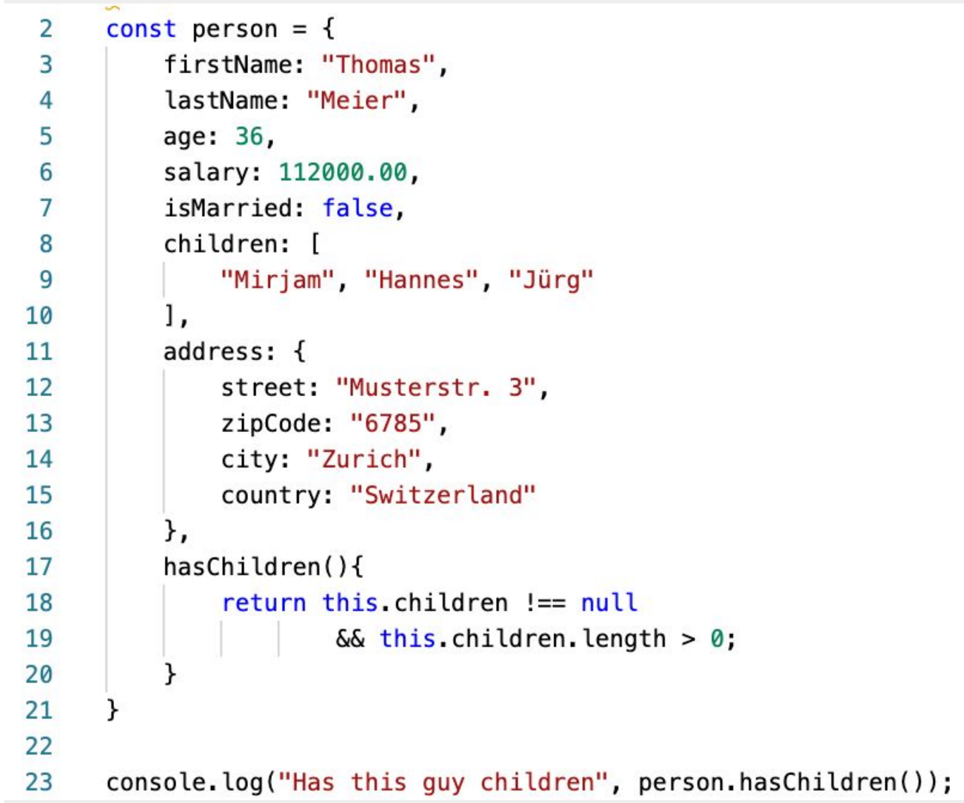
can contain other objects as well.



```
const person = {
      firstName: "Thomas",
      lastName: "Meier",
      age: 36,
      salary: 112000.00,
      isMarried: false,
      children: [
          "Mirjam", "Hannes", "Jürg"
      ],
      address: {
          street: "Musterstr. 3",
          zipCode: "6785",
          city: "Zurich",
          country: "Switzerland"
      }
}
```

Objects

can contain also functions (surprise!).


```
const person = {
    firstName: "Thomas",
    lastName: "Meier",
    age: 36,
    salary: 112000.00,
    isMarried: false,
    children: [
        "Mirjam", "Hannes", "Jürg"
    ],
    address: {
        street: "Musterstr. 3",
        zipCode: "6785",
        city: "Zurich",
        country: "Switzerland"
    },
    hasChildren(){
        return this.children !== null
                && this.children.length > 0;
    }
}

console.log("Has this guy children", person.hasChildren());
```

How to access data within an object?

1

object.propertyName

```
console.log(person.firstName)
console.log(person.address.city)
```

2

object["propertyName"]

```
console.log(person["firstName"])
console.log(person["address"].city)
console.log(person["address"]["city"])
```

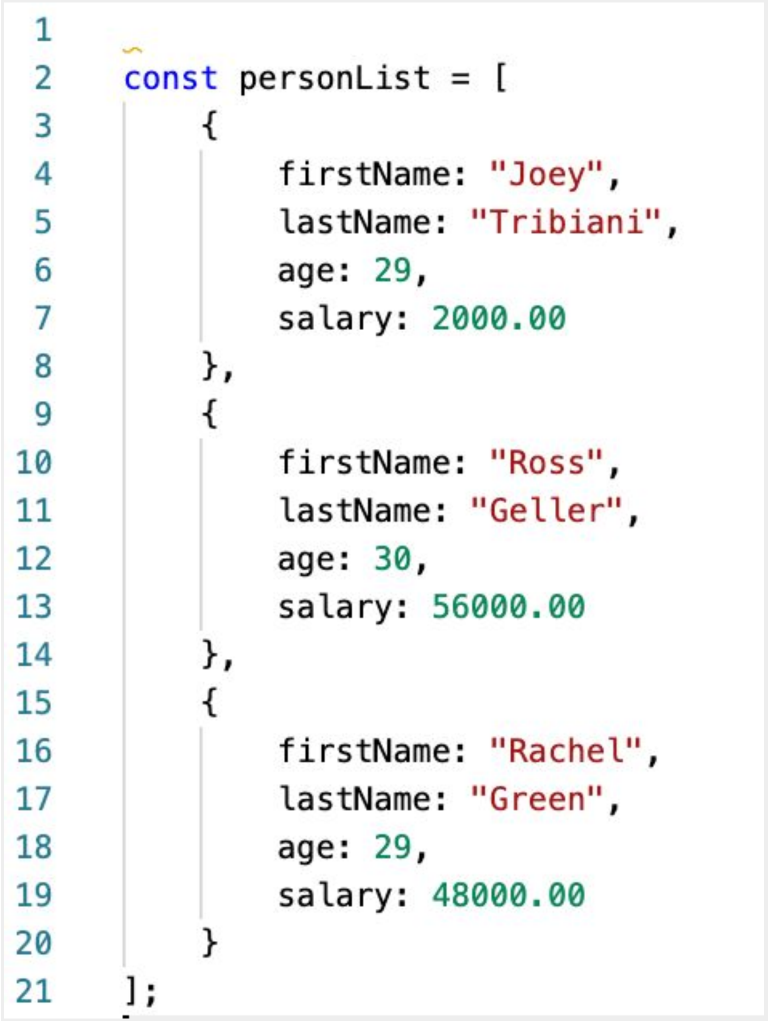
3

{propertyName} = object;

```
let {firstName, address} = person;
console.log(firstName);
let {city} = address;
console.log(city);
```

Objects

can be an array item as well



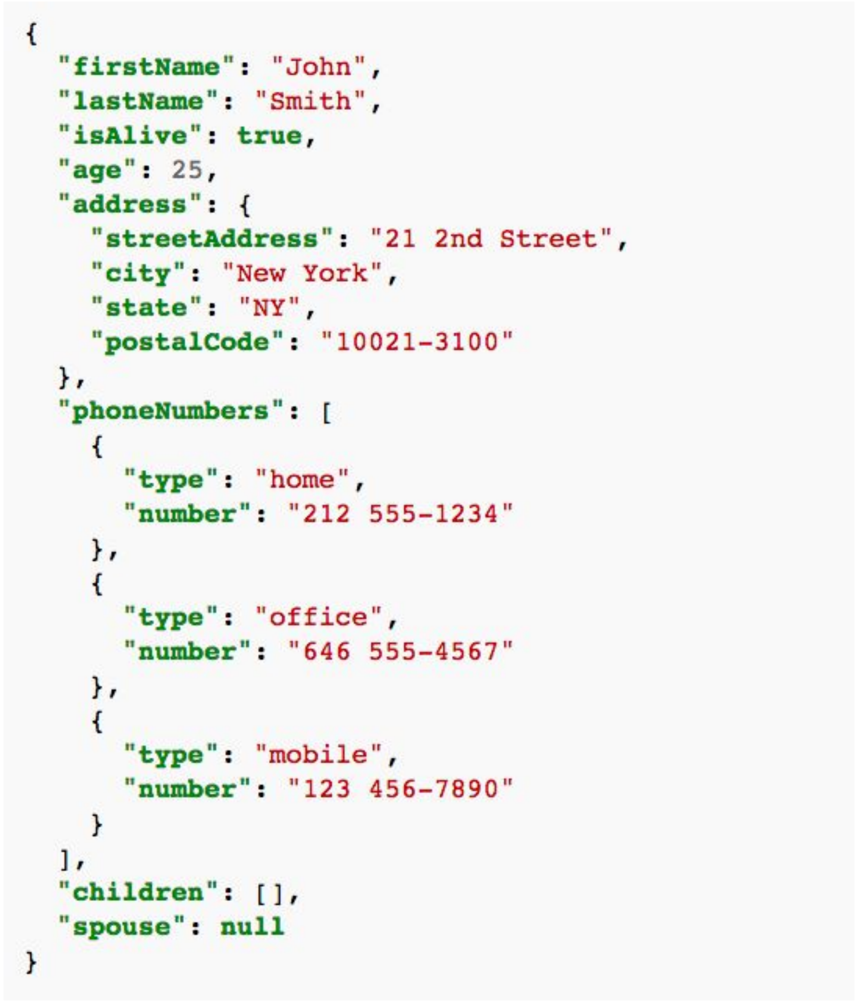
```
const personList = [
    {
        firstName: "Joey",
        lastName: "Tribiani",
        age: 29,
        salary: 2000.00
    },
    {
        firstName: "Ross",
        lastName: "Geller",
        age: 30,
        salary: 56000.00
    },
    {
        firstName: "Rachel",
        lastName: "Green",
        age: 29,
        salary: 48000.00
    }
];
```

Javascript
Array vs Object vs JSON

[] vs {} vs "{}"

let's try it!

...



```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

let's try it!

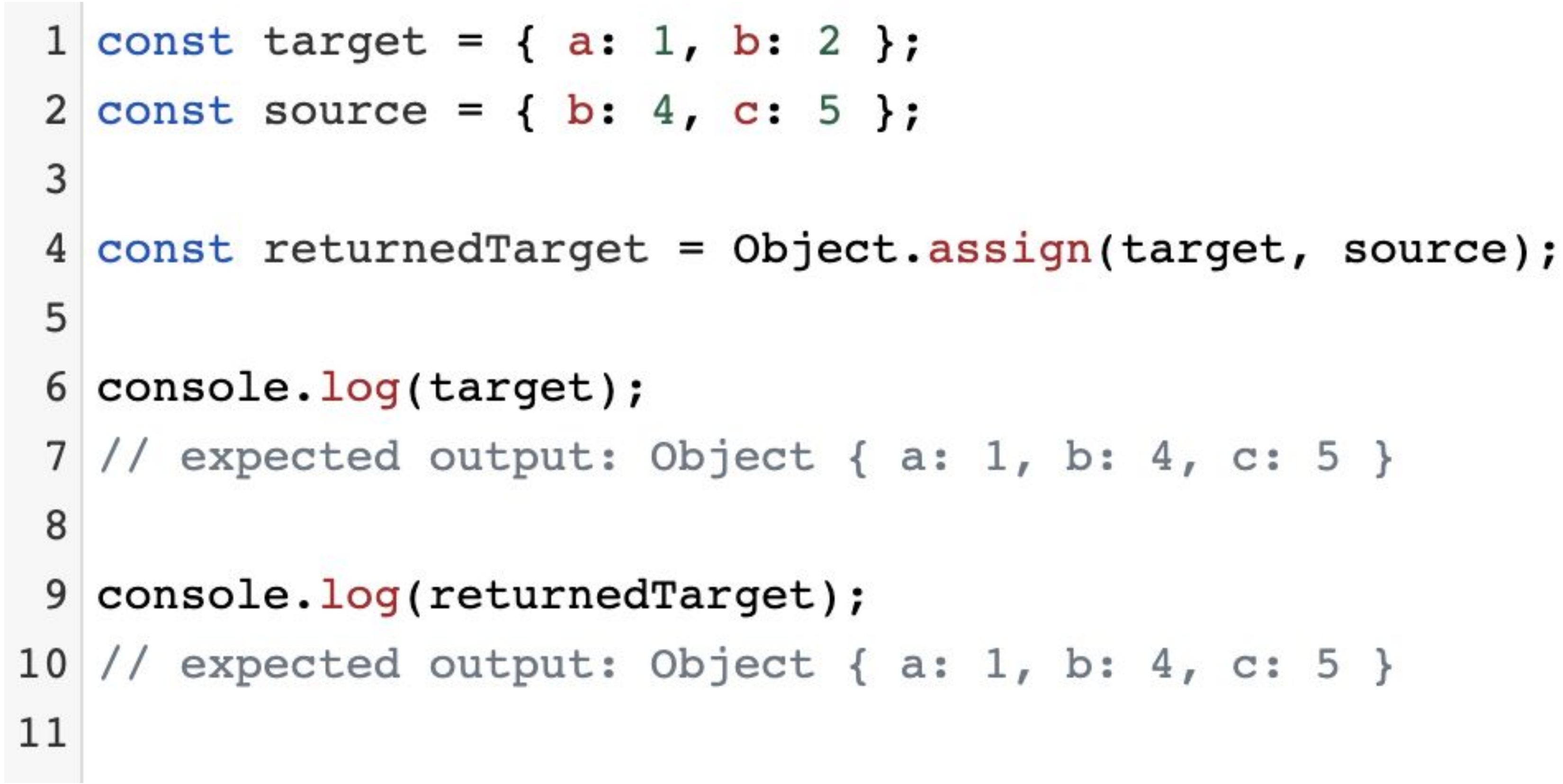
Queries on the structure

- How many phone numbers does this person have?
- In which city does this person live?
- Does this person have children?
- What is the name of this person's partner?
- During the day I want to reach this person, which phone number should I dial?

Object methods

Nützliche Methode:

- Object.create(obj)
- **Object.assign(obj, obj)**
- Object.values()
- Object.keys()
- Object.entries()



```javascript
const target = { a: 1, b: 2 };
const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);
// expected output: Object { a: 1, b: 4, c: 5 }

console.log(returnedTarget);
// expected output: Object { a: 1, b: 4, c: 5 }

```

Object.assign(...) → Copy an object into another one

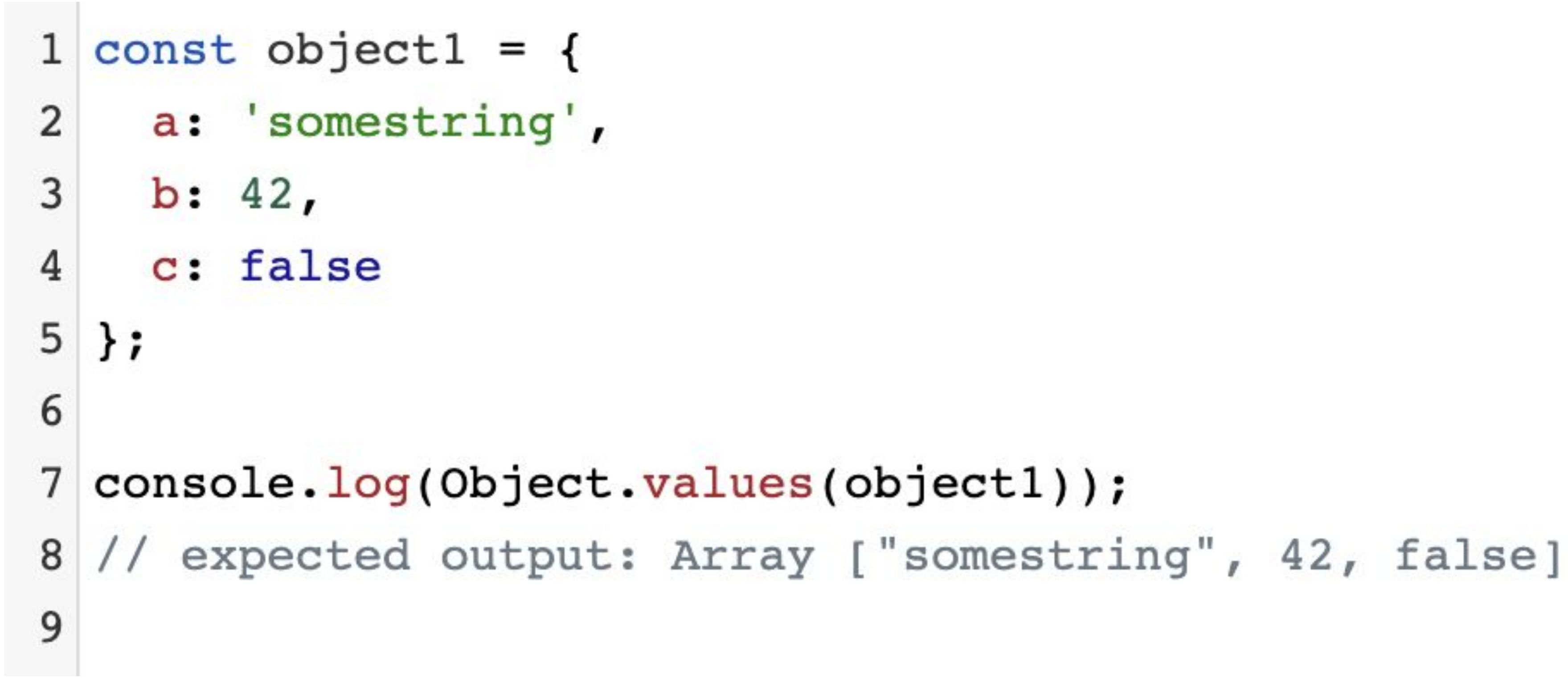
```javascript
const person = {
  isHuman: false,
  printIntroduction: function() {
    console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);
  }
};

const me = Object.create(person);

me.name = 'Matthew'; // "name" is a property set on "me", but not on "person"
me.isHuman = true; // inherited properties can be overwritten

me.printIntroduction();
// expected output: "My name is Matthew. Am I human? true"
```

Object.create(...) → Create an object from given template

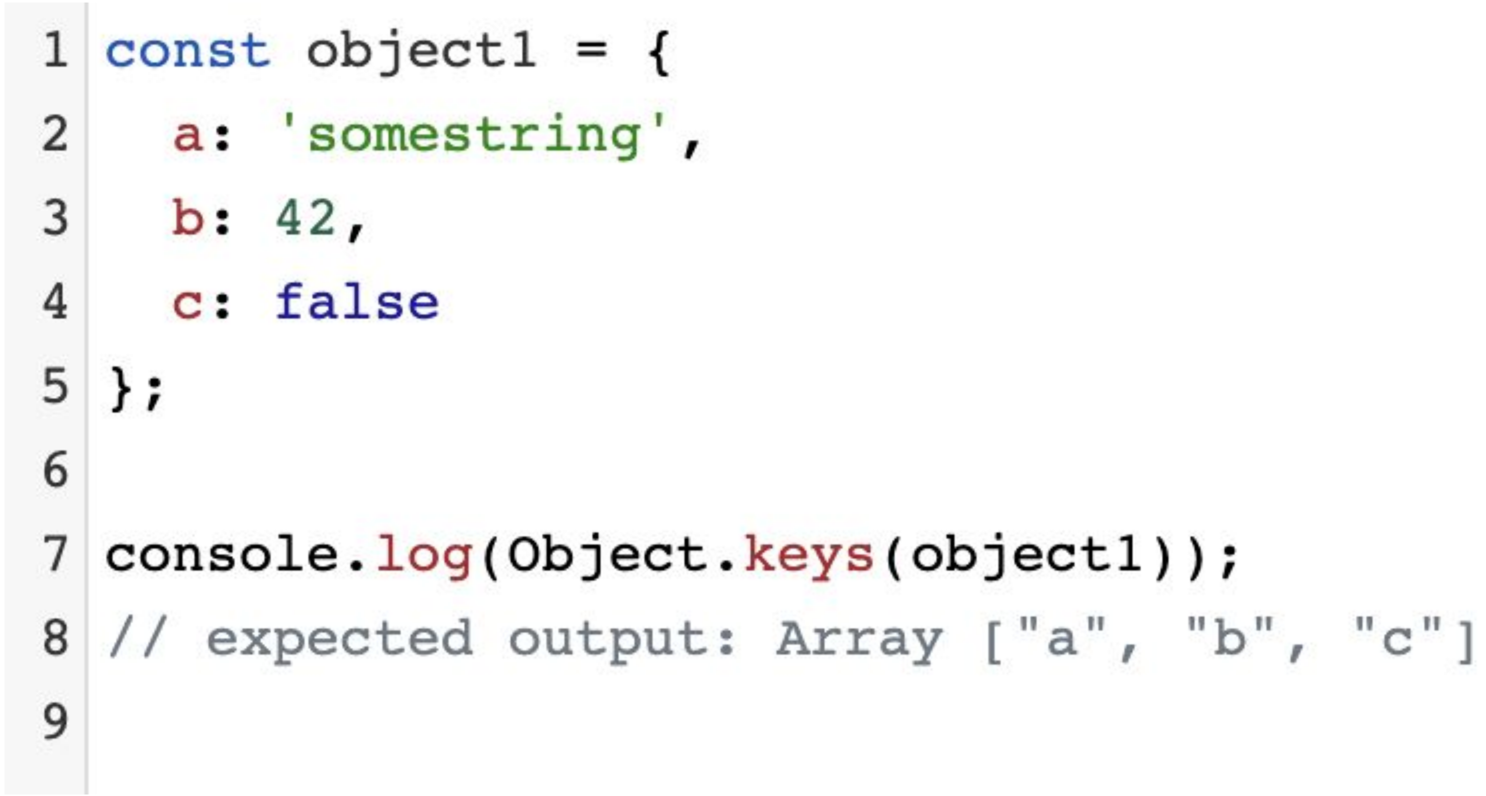


```javascript
const object1 = {
  a: 'somestring',
  b: 42,
  c: false
};

console.log(Object.values(object1));
// expected output: Array ["somestring", 42, false]

```

Object.values() → get object **values** as an array



```javascript
const object1 = {
  a: 'somestring',
  b: 42,
  c: false
};

console.log(Object.keys(object1));
// expected output: Array ["a", "b", "c"]
```

Object.keys() → get **keys** as an array

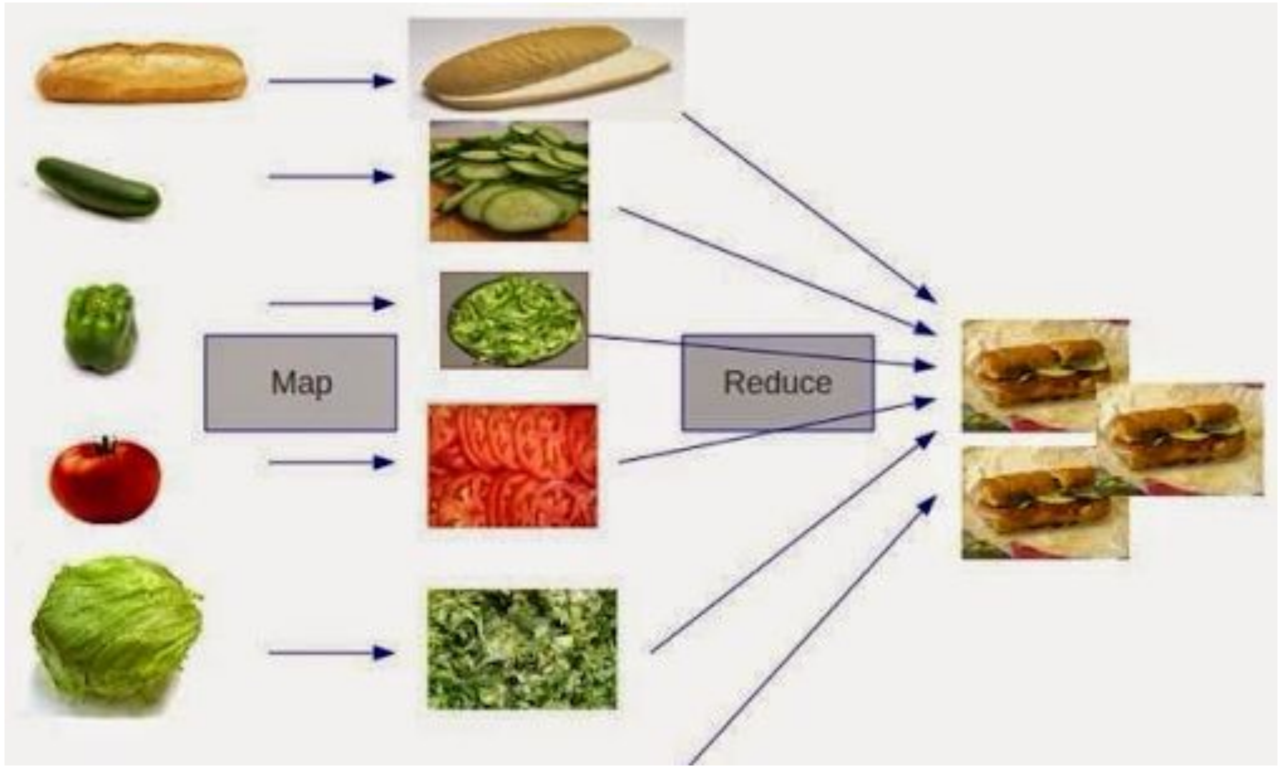
Functional extensions in arrays

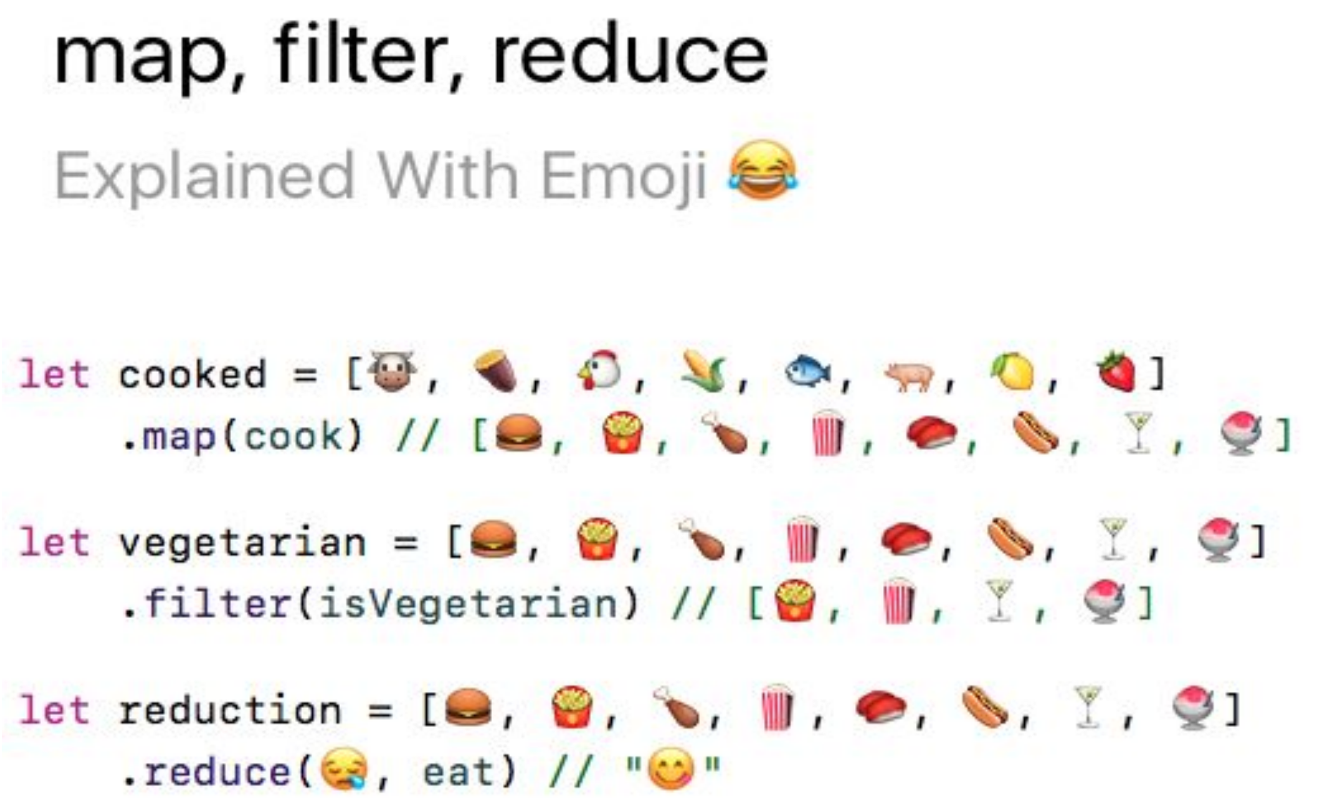
eliminate the need of for-loops

Functional Extensions

Mostly used:

- Array.prototype.find()
- Array.prototype.filter()
- Array.prototype.map()
- Array.prototype.reduce()
- Array.prototype.every()
- Array.prototype.some()


Map
Reduce



map, filter, reduce

Explained With Emoji 😂

```
let cooked = [🐮, 🍠, 🐔, 🌽, 🐟, 🐖, 🍋, 🍓]
    .map(cook) // [🍔, 🍟, 🍗, 🍿, 🍣, 🌭, 🍸, 🍧]

let vegetarian = [🍔, 🍟, 🍗, 🍿, 🍣, 🌭, 🍸, 🍧]
    .filter(isVegetarian) // [🍟, 🍿, 🍸, 🍧]

let reduction = [🍔, 🍟, 🍗, 🍿, 🍣, 🌭, 🍸, 🍧]
    .reduce(😪, eat) // "😋"
```

```
reduce([🍔, 🍟, 🍗, 🍿], eat)
=> 💩
```

```
const inventory = [
    {name: 'apples', quantity: 2},
    {name: 'bananas', quantity: 0},
    {name: 'cherries', quantity: 5}
];

const result = inventory.find( fruit => fruit.name === 'cherries' );

console.log(result) // { name: 'cherries', quantity: 5 }
```

find

```javascript
function isBigEnough(value) {
  return value >= 10;
}

var filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
// filtered is [12, 130, 44]
```

filter

```
var kvArray = [{key: 1, value: 10},
               {key: 2, value: 20},
               {key: 3, value: 30}];

var reformattedArray = kvArray.map(obj =>{
   var rObj = {};
   rObj[obj.key] = obj.value;
   return rObj;
});
// reformattedArray is now [{1: 10}, {2: 20}, {3: 30}],

// kvArray is still:
// [{key: 1, value: 10},
//  {key: 2, value: 20},
//  {key: 3, value: 30}]
```

map

```
[0, 1, 2, 3, 4].reduce(function(accumulator, currentValue, currentIndex, array) {
  return accumulator + currentValue;
});
```

reduce

```
function isBiggerThan10(element, index, array) {
  return element > 10;
}

[2, 5, 8, 1, 4].some(isBiggerThan10);  // false
[12, 5, 8, 1, 4].some(isBiggerThan10); // true
```

some

```
function isBigEnough(element, index, array) {
  return element >= 10;
}
[12, 5, 8, 130, 44].every(isBigEnough);   // false
[12, 54, 18, 130, 44].every(isBigEnough); // true
```

every

```
var arr1 = [1, 2, [3, 4]];
arr1.flat();
// [1, 2, 3, 4]

var arr2 = [1, 2, [3, 4, [5, 6]]];
arr2.flat();
// [1, 2, 3, 4, [5, 6]]

var arr3 = [1, 2, [3, 4, [5, 6]]];
arr3.flat(2);
// [1, 2, 3, 4, 5, 6]
```

flat

```
var arr1 = [1, 2, 3, 4];

arr1.map(x => [x * 2]);
// [[2], [4], [6], [8]]

arr1.flatMap(x => [x * 2]);
// [2, 4, 6, 8]

// only one level is flattened
arr1.flatMap(x => [[x * 2]]);
// [[2], [4], [6], [8]]
```

flatmap

Questions?