

Lite repetition

for... of

- En *for... of*-loop loopar igenom alla värden i en array.

```
let students = [  
  "Lisa",  
  "Jessica",  
  "Ali",  
  "Stina"  
];  
  
for(student of students) {  
  console.log(student);  
};
```

Lisa
Jessica
Ali
Stina



for... in

- *For... in* fungerar som *for... of*, men variabeln kommer då att innehålla det nuvarande indexet istället för det nuvarande värdet.

```
let students = [  
  "Lisa",  
  "Jessica",  
  "Ali",  
  "Stina"  
];  
  
for(student in students) {  
  console.log(student);  
};
```

0

1

2

3

forEach

- *forEach* fungerar som for... of men tar en funktion som parameter.
- Vad kallas denna typ av icke namngivna funktioner?

```
let students = [  
  "Lisa",  
  "Jessica",  
  "Ali",  
  "Stina"  
];  
  
students.forEach(function(element) {  
  console.log(element);  
});
```

Lisa

Jessica

Ali

Stina

Lägga till värde i array

- Arrayer kan använda sig av speciella array-metoder. En av dessa är push som används för att lägga till ett värde, en låda till arrayen.

```
let students = [  
  "Lisa",  
  "Jessica",  
  "Ali"  
];  
  
students.push("Stina");
```

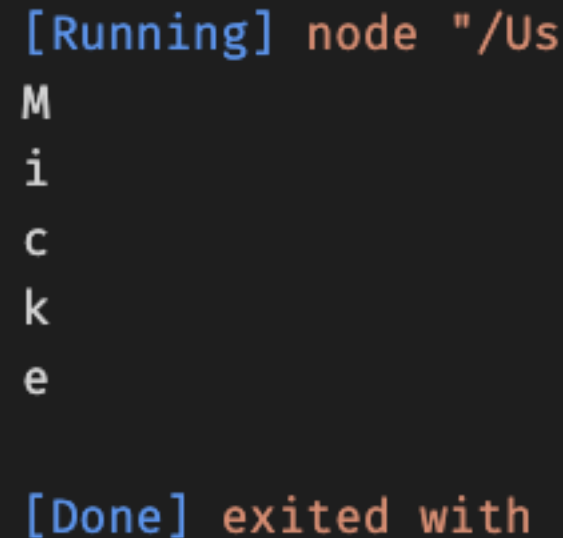
Ta bort värde från array

- `pop` - tar bort från slutet av en array och returnerar värdet.
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop
- `shift` - tar bort från början av en array och returnerar värdet.
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/shift
- `splice` - tar bort från en array för ett specifikt index och returnerar värdet / värdena.
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice
- `filter` - returnerar en ny array med filtrerade element från en array. (Mer om denna senare.)
 - <https://love2dev.com/blog/javascript-remove-from-array/>

Strings

- En sträng är faktiskt en array av tecken.

```
let name = "Micke";  
  
for ( let char of name ) {  
    console.log(char);  
}
```



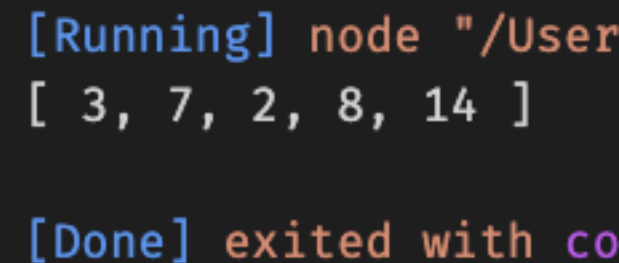
```
[Running] node "/Us  
M  
i  
c  
k  
e  
  
[Done] exited with
```

Rest

- Ibland vill man kunna ta emot ett okänt antal parametrar. Då kan man använda en `rest`-parameter. Den anges med tre punkter.

```
function sum (...numbers) {  
    console.log(numbers);  
}
```

```
sum (3, 7, 2, 8, 14);
```



```
[Running] node "/User  
[ 3, 7, 2, 8, 14 ]  
  
[Done] exited with co
```


Default-värde

- Parametrar kan innehålla default-värden, värden som används om man inte skickar någon parameter.

```
function add_numbers(a=2, b=5) {  
    return a + b;  
}
```

```
console.log(add_numbers());          // 7  
console.log(add_numbers(4));        // 9  
console.log(add_numbers(5, 7));     // 12
```

Sortering av arrayer

- En array kan sorteras med metoden sort.
Det fungerar lite olika för tal och strängar.

Vad tar sort-metoden för typ
av parameter?

```
var numbers = [9,3,12,5,1,88];  
var sortedNumbers= numbers.sort(function (a, b){ return a - b});
```

```
var names = ["Kalle", "Lisa", "Anna", "Emma"];  
var sortedNames= names.sort( );
```

Sortering av arrayer

- Om vi anger en funktion kan vi själva bestämma hur sorteringen ska fungera.
- Principen för vår jämförelsefunktion är att JS kommer att använda den för att jämföra två värden och vi får själva bestämma vilket som ska komma först.

```
function compare(a, b) {  
  if (a is less than b by some ordering criterion) {  
    return -1;  
  }  
  if (a is greater than b by the ordering criterion) {  
    return 1;  
  }  
  // a must be equal to b  
  return 0;  
}
```

Lambda-funktioner

- Uttryck som skapar funktioner.

- Pre ES6:

```
var anon = function (a, b) { return a + b };
```

- ES6:

```
// Ovanstående exempel:
```

```
var anon = (a, b) => a + b;
```

```
// Eller
```

```
var anon = (a, b) => { return a + b };
```


```
// Om vi bara har en parameter kan vi skippa
```

```
// parenteserna
```

```
var anon = a => a;
```

Map

- `map ()` -metoden skapar en ny array med resultatet av att anropa en funktion för varje array-element.
- `map ()` -metoden anropar den angivna funktionen en gång per element i arrayen i ordning.




```
const myArray = [1,2,3,4];

const myArrayTimesTwo = myArray.map((value, index, array) => {
  return value * 2;
});

console.log(myArray); // [1,2,3,4];
console.log(myArrayTimesTwo); // [2,4,6,8];
```

Filter

- `filter` tar emot samma argument som `map` och fungerar liknande. Enda skillnaden är att callback-funktionen måste returnera `true` eller `false`. Om den returnerar `true` kommer arrayen att behålla elementet, om den returnerar `false` kommer det att filtreras bort.




```
const myArray = [1,2,3,4];

const myEvenArray = myArray.filter((value, index, array) => {
  return value % 2 === 0;
});

console.log(myArray); // [1,2,3,4];
console.log(myEvenArray); // [2, 4];
```

Reduce

- reduce tar en array och reducerar den till ett enda värde. Du kan t ex använda det för att räkna ut medelvärdet av alla värden i arrayen.



```
const myArray = [1,2,3,4];

const sum = myArray.reduce((acc, currValue, currIndex, array) => {
  return acc + currValue;
}, 0);

const avg = sum / myArray.length;

console.log(avg); // 2.5
```

Nu fortsätter vi...

Join

- `arr.join([separator])` används för att slå ihop värden i en array till en sträng.

```
var a = ['Wind', 'Water', 'Fire'];  
a.join();           // 'Wind,Water,Fire'  
a.join(', ');       // 'Wind, Water, Fire'  
a.join(' + ');      // 'Wind + Water + Fire'  
a.join('');         // 'WindWaterFire'
```

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/join

Split

- `str.split([separator[, limit]])`
används för att dela upp en sträng till en array.

```
const str = 'The quick brown fox jumps over the lazy dog.';
```

```
const words = str.split(' ');  
console.log(words[3]);  
// expected output: "fox"
```

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/split

Klasser

- Klasser är lite som ritningar eller mallar för objekt.
- Varje objekt som skapas från en klass, eller *instancieras*, kommer att innehålla alla de egenskaper och metoder som klassen innehåller. Detta exempel innehåller dock inget.

```
class Vehicle {  
    // Stuff  
}
```

Instansiering

- Man instansierar ett objekt genom nyckelordet *new*.

```
class Vehicle {  
    // Stuff  
}
```

```
let car = new Vehicle();  
console.log(car);
```

Uppgift

- Skapa en klass som heter *Student*.
- Skapa en variabel, *student1*, som innehåller ett objekt av klassen och skriv ut objektet.

Metoder

- Man skriver metoder i en klass lite annorlunda än man gör i objekt.

```
class Vehicle {  
  hello() {  
    console.log("hello!");  
  }  
}
```

```
let car = new Vehicle();  
car.hello();
```

Uppgift

- Skapa metoden *greeting* som skriver ut "Hello".
- Anropa metoden genom variabeln student1.

Konstruktor

- Det finns en speciell metod i en klass som heter *constructor*. Den körs alltid när man skapar ett nytt objekt av klassen.

```
class Vehicle {  
    constructor() {  
        console.log("creating an object");  
    }  
}
```

```
let car = new Vehicle();
```


This

- Kommer ni ihåg nyckelordet *this*?

This

- Om vi vill använda en egenskap inuti objektet kan vi använda nyckelordet *this*.

```
let student = {  
  name: "Mikael Olsson",  
  print: function () {  
    console.log(this.name);  
  }  
}  
  
student.print();
```

```
[Running] node "/U  
Mikael Olsson  
  
[Done] exited with
```

This

- I klasser används *this* likadant. Vi använder *this* *inuti* klassen:
 - Om vi vill använda en egenskap.
 - Om vi vill anropa en metod.

Egenskaper

- Egenskaper skapas i konstruktorn i JS.

```
class Vehicle {  
  constructor() {  
    this.make = "Volvo";  
    this.model = "V70";  
  }  
}
```

```
let car = new Vehicle();
```

Egenskaper

- Likadant om vi vill komma åt egenskaper i andra metoder.

```
class Vehicle {  
  constructor() {  
    this.make = "Volvo";  
    this.model = "V70";  
  }  
  print() {  
    console.log("Denna bil är en " + this.make);  
  }  
}
```

Uppgift

- Lägg till egenskaperna *name* och *age* i din klass.
- Ändra metoden *greeting* så att den skriver ut "Hello, " och studentens namn.

Metoder

- Om vi vill använda en metod inuti en klass använder vi också this.

```
class Vehicle {  
  hello() {  
    console.log("hello!");  
  }  
  foo() {  
    this.hello();  
  }  
}
```

```
let car = new Vehicle();  
car.foo();
```

Uppgift

- Anropa metoden *greeting* från konstruktorn.

Konstruktor

- Konstruktor är en vanlig funktion. Den kan ta emot parametrar. Det är vanligt att använda dem till att sätta egenskaper.

```
class Vehicle {  
    constructor(make) {  
        this.make = make;  
    }  
}
```

```
let car = new Vehicle("Volvo");
```


Default-värden

- Det är vanligt att låta konstruktorn ta emot parametrar men att ge dem default-värden.

```
class Vehicle {  
    constructor(make = "Ford") {  
        this.make = make;  
    }  
}  
  
let car1 = new Vehicle();  
let car2 = new Vehicle("Volvo");
```

Uppgift

- Låt konstruktorn ta emot *name* och *age* som parametrar.
- Sätt en tom sträng och 0 som default-värden.

Uppgift

- Skapa en klass som heter *Subject* med egenskaperna *title* och *points*.
- Låt konstruktion ta emot *title* och *points* som parametrar med default-värden.
- Lägg till egenskapen *subjects* i *Student*-klassen. Låt den innehålla en tom array.
- Lägg till metoden *add_subject* i *Student*-klassen som lägger till ett objekt av klassen *Subject* i egenskapen *subjects*.

Egenskaper

- Vi läser en egenskap genom variabeln.egenskapen:
 - `console.log(car1.color);`
- Vi skriver till egenskapen på nästan samma sätt:
 - `car1.color = "blue";`
- Kan JS förstå när vi använder egenskapen för att läsa från den eller skriva till den?
- Ibland vill vi ha lite större kontroll över hur man skriver/läser egenskaper. Då kan man använda getters/setters.

Getters / setters

- The get syntax binds an object property to a function that will be called when that property is looked up.

```
const obj = {  
  log: ['a', 'b', 'c'],  
  get latest() {  
    if (this.log.length === 0) {  
      return undefined;  
    }  
    return this.log[this.log.length - 1];  
  }  
};
```

```
console.log(obj.latest);
```

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/get>

Getters / setters

- The set syntax binds an object property to a function to be called when there is an attempt to set that property.

```
const language = {  
  set current(name) {  
    this.log.push(name);  
  },  
  log: []  
};
```

```
language.current = 'EN';  
language.current = 'FA';
```

```
console.log(language.log);
```

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/set>

Getters / setters

- Getters och setters ser ut som egenskaper för ”användaren”, men är metoder. Det ger oss bättre möjligheter att kontrollera hur värden används, t ex för felkontroll.

Uppgift

- Lägg till en getter och en setter för egenskapen subjects.
- Om användaren läser från egenskapen ska hen få en sträng som innehåller titeln på samtliga ämnen, separerade med kommatecken.
- Om användaren skriver till egenskapen ska ni kontrollera hur många ämnen subjects innehåller. Om subjects innehåller färre än tre ämnen ska ni lägga till det nya ämnet.

Arv

- Arv låter oss skapa klasser som ärver sitt innehåll från en annan klass.
- Vi kan t ex skapa en klass, Person, som innehåller namn, personnr, hårfärg och sånt som beskriver en person. Vi kan sedan låta klasserna Student, Teacher, Boss, Employee ärva från denna klass.
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/extends>

Super

- För att använda en metod från föräldraklassen använder vi super.
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super>

Uppgift

- Skriv klassen *Vehicle* med egenskaperna *make*, *model*, *color* och *no_of_wheels* med default-värdet 4.
 - Låt konstruktorn ta emot parametrar för samtliga egenskaper.
- Skriv klassen *Motorcycle* som ärver från *Vehicle*.
 - *no_of_wheels* ska ha default-värdet 2.
 - Skriv konstruktorn så att den kan skicka vidare alla konstruktor-parametrar till föräldraklassens konstruktor.
- Skapa objekt av båda klasserna och skriv ut dem.

Override

- Om både Vehicle och Motorcycle innehåller metoden print(), vilken metod kommer då att anropas?
- Den metod som hör till klassen. Dvs om objektet är av klassen Vehicle kommer den metoden att anropas, om objektet är av klassen Motorcycle kommer den metoden att anropas.
- "Motorcycle-print()" kan anropa "Vehicle-print()" genom att använda *super*.

Super

```
class Vehicle {  
    print() {  
        console.log("Vehicle");  
    }  
}  
class Motorcycle extends Vehicle {  
    print() {  
        console.log("Motorcycle");  
        super.print();  
    }  
}
```

Static

- Ibland vill man kunna skriva metoder som inte direkt hör till ett objekt. Det kan man göra genom att skriva statiska metoder med hjälp av nyckelordet *static*.
- Man anropar metoden genom att ange klassens namn.

```
class Vehicle {  
    static print() {  
        console.log("This is a common text, not  
associated with any vehicle in particular.");  
    }  
}  
  
Vehicle.print();
```

Static

- Man kan också göra statiska egenskaper. Vad har Rectangle.counter för funktion?

```
class Rectangle {  
  constructor() {  
    console.log("Creating rectangle");  
    Rectangle.counter++;  
  }  
}
```

- Statiska egenskaper fungerar inte i alla browsers/JS-motorer än.

Const

- En konstant är ungefär som en variabel, men den kan inte byta värde när den har blivit initierad.

```
const x = 5;  
x = 10; // Ger TypeError: Assignment to  
constant variable.
```


Math

- JS innehåller olika bibliotek med hjälp-funktioner, egenskaper och konstanter. Ett av dessa är Math.
- `Math.random()` ger ett slumpmässigt decimaltal som är 0 till 1 (men inte 1).
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random

Math

- `Math.round()` avrundar till närmaste heltal.
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/round
- `Math.floor()` avrundar nedåt till närmaste heltal.
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/floor
- `Math.ceil()` avrundar uppåt till närmaste heltal.
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/ceil

Math

- `Math.min()` returnerar det minsta av flera tal.
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/min
- `Math.max()` returnerar det största av flera tal.
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/max
- Läs mer om Math-biblioteket här:
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

Math.random()

- Vad gör följande funktion?

```
function getRandomInt(max) {  
    return Math.floor(Math.random() * Math.floor(max));  
}
```

- Uppgift: Skriv om funktionen så att den även tar en undre gräns, dvs att anropet `getRandomInt(5, 10)` ger ett slumpmässigt tal från 5 till 10.

Tärningar

- Skriv klassen *Die* med egenskapen *value*.
- Lägg till metoden *throw()* som ger tärningen ett nytt värde från 1 till 6.
- Anropa metoden *throw()* i konstruktorn.
- Skriv klassen *Dice* med egenskapen *dice*, som ska innehålla en array.
- Låt konstruktorn ta emot en parameter som säger hur många träningar vi vill skapa med default-värdet 5 och lägga till så många objekt av klassen *Die* i egenskapen *dice*.
- Lägg till metoden *throw()* som slår alla tärningarna med hjälp av metoden *throw()* i *Die*-klassen.