

Lite repetition

Vi kommer att börja dagen med att repetera vad vi har gjort hittills innan vi gemensamt går igenom en laboration.

Vad är ett scriptspråk?

Man skiljer på kompilerande och interpreterande språk.

Kod skriven i tex C# eller Java kompileras innan den kan köras. Det innebär att den omvandlas till maskinnära kod som skiljer sig från källkoden. Detta kallas **kompilerande språk**.

Ett scriptspråk kompileras inte innan, utan koden där körs direkt när den exekveras. Då behövs någon form av scripttolk (interpretator) som kan omvandla koden till instruktioner. Detta **kallas för interpreterande språk**.

Webbsidan- Tre lager som samverkar

Det som händer i en webbläsare när en sida visas består av 3 lager som samverkar.

HTML - beskriver strukturen på sidan dvs vilka rubriker som finns, hur menyer/navigeringen ser ut, vilka texter som skall stå på sidan.

CSS - beskriver layouten på sidan dvs färg och form

Javascript - interaktiviteten dvs så fort en sida inte skall vara statisk utan det skall hända något i webbläsaren. Det kan tex vara när användaren gör något på sidan.

Hur körs koden ?

- Javascript togs från början fram för att köras i en webbläsare. Alla stora webbläsare har en inbyggd javascriptmotor (tolk) som gör att det går att köra kod.
- Detta förändrades 2009 när Node.js kom. Det är en runtimemiljö för att köra javascriptkod utanför en webbläsare. Den används oftast på webbservern.
- Google har tagit fram en javascriptmotor till Chrome som heter V8. Den används även av Node.JS för att köra kod.

JS - Editorer

- Finns en stor mängd editorer där det går att skriva och testa JavaScript-kod. Här är några av de vanligaste:
 - Visual Studio Code
 - Sublime
 - Atom
 - JS Fiddle

Versioner av Javascript

- Kan variera mellan olika webbläsare vilket stöd de har för olika funktionalitet som finns i olika versioner
- ECMAScript 11 senaste versionen, den kom 2020.
- <https://en.wikipedia.org/wiki/ECMAScript>

Ett språk i förändring

- Tanken från början var bara att ha ett språk för att göra enklare saker i webbläsaren.
- Idag byggs den en annan typ av applikationer och det sätt som webben används på har förändrats.
- De senaste versionerna av Javascript innehåller därför mycket nya saker och språket förändras en hel del.

ECMAScript 6 - 2015

- Significant new syntax for writing complex applications, including class declarations (`class Foo { ... }`), ES6 modules like `import * as moduleName from " ... ";` `export const Foo;`
- *iterators* and *for...of* loops
- Python-style *generators*
- *arrow function expression* `(() ⇒ { ... })`
- *let* keyword for local declarations
- *const* keyword for constant local declarations
- binary data, typed arrays, new collections (maps, sets and WeakMap)
- promises
- number and math enhancements, reflection, proxies
- *template literals* for strings.

ECMAScript 7 - 2016

- block-scoping of variables and functions
- destructuring patterns (of variables)
- proper tail calls
- exponentiation operator `**` for numbers
- `await`, `async` keywords for asynchronous programming
- `Array.prototype.includes`

ECMAScript 8 - 2017

- `Object.values`
- `Object.entries`
- `Object.getOwnPropertyDescriptors`
 - functions for easy manipulation of Objects
- `async/await` constructions which use generators and promises
- additional features for concurrency and atomics

ECMAScript 9 - 2018

- rest/spread operators for object literals (three dots: ...identifier)
- asynchronous iteration
- Promise.prototype.finally
- additions to RegExp.

```
let object = {a: 1, b: 2}
```

```
let objectClone = Object.assign({}, object) // before ES9
```

```
let objectClone = {...object} // ES9 syntax
```

```
let otherObject = {c: 3, ...object}
```

```
console.log(otherObject) // -> {c: 3, a: 1, b: 2}
```

ECMAScript 10 - 2019

- `Array.prototype.flat`, `Array.prototype.flatMap`, changes to `Array.sort` and `Object.fromEntries`.
- `Array.sort` is now guaranteed to be stable, meaning that elements with the same sorting precedence will appear in the same order in the sorted array.
- `Array.prototype.flat(depth=1)` flattens an array to a specified depth, meaning that all subarray elements (up to the specified depth) are concatenated recursively.

ECMAScript 11 - 2020

- BigInt primitive type for arbitrary-sized integers
- nullish coalescing operator
- globalThis object.
- The nullish coalescing operator, ??, returns its right-hand side operand when its left-hand side is null or undefined.

```
false ?? "string" // -> false
```

```
NaN ?? "string" // -> NaN
```

```
undefined ?? "string" // -> "string"
```

- Optional chaining allows you to access for the nested nodes in an object without having a AND check at each level.
- An example is `const zipcode = person?.address?.zipcode`. If any of the properties are not present, `zipcode` will be undefined.

JavaScript kodbibliotek och ramverk

Det finns bibliotek med färdig JS kod som man kan ladda ned och använda sig av för att bygga lösningar.

För att ”slippa uppfinna hjulet på nytt” och snabbare bygga lösningar.

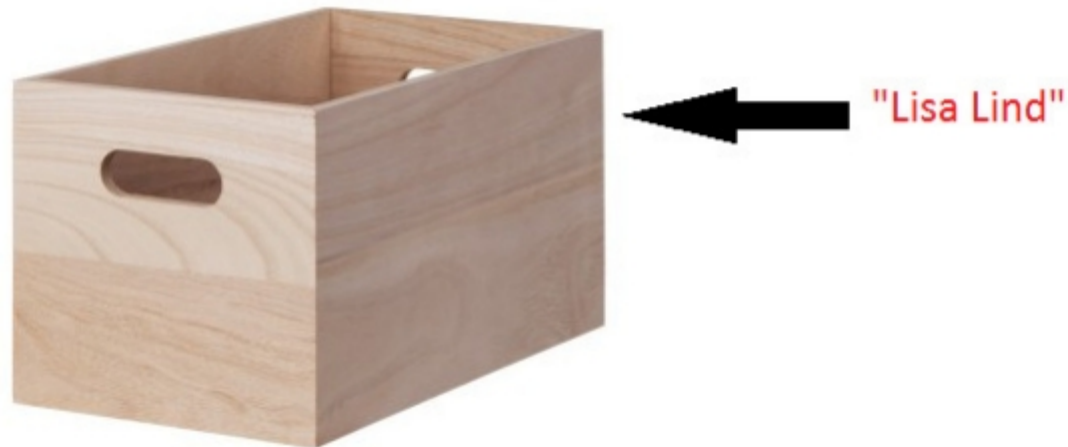
Bibliotek har ofta färdiga funktioner inom ett specifikt område.

Ramverk har ofta färdiga modeller för hur man ska bygga hela applikationer. Exempel på ramverk: AngularJS, React, Knockout, Ember, Vue.

Deklarera en variabel

Här deklarerar en variabel och tilldelas samtidigt ett värde. Det gör att den kan användas senare i programkoden.

```
let student_name = "Lisa Lind";
```



Datatype

- The latest ECMAScript standard defines nine types:
 - Six Data Types that are primitives, checked by typeof operator:
 - *undefined* : typeof instance === "undefined"
 - *Boolean* : typeof instance === "boolean"
 - *Number* : typeof instance === "number"
 - *String* : typeof instance === "string"
 - *BigInt* : typeof instance === "bigint"
 - *Symbol* : typeof instance === "symbol"
 - Structural Types:
 - *Object* : typeof instance === "object". Special non-data but Structural type for any constructed object instance also used as data structures: new Object, new Array, new Map, new Set, new WeakMap, new WeakSet, new Date and almost everything made with new keyword;
 - *Function* : a non-data structure, though it also answers for typeof operator: typeof instance === "function". This is merely a special shorthand for Functions, though every Function constructor is derived from Object constructor.
 - Structural Root Primitive:
 - *null* : typeof instance === "object". Special primitive type having additional usage for its value: if object is not inherited, then null is shown;

Aritmetiska operatorer

Operator	Description	Example
+	Addition	$x = y + 2$
-	Subtraction	$x = y - 2$
*	Multiplication	$x = y * 2$
/	Division	$x = y / 2$
%	Modulus (division remainder)	$x = y \% 2$
++	Increment	$x = ++y$ $x = y++$
--	Decrement	$x = --y$ $x = y--$

Kommentarer

- Används för att förklara en viss kodrad eller för att markera att vissa rader inte skall köras.

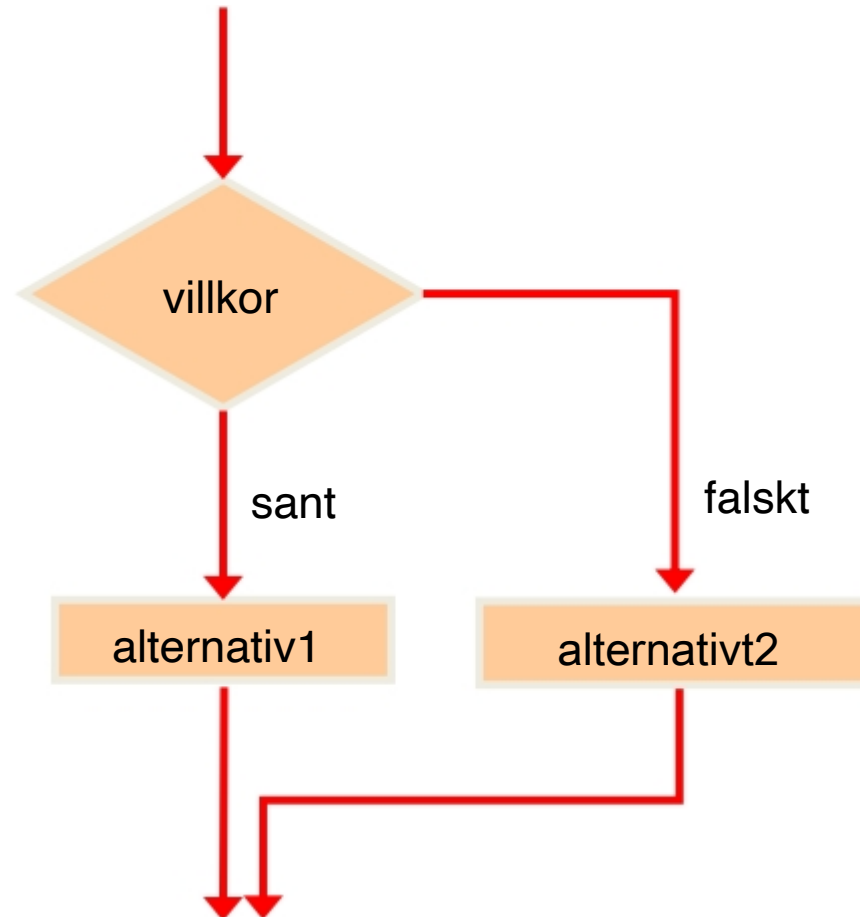
```
// Detta är en kommentar på en rad  
/*
```

```
    En kommentar som startas och  
avslutas, kan gå över flera rader.
```

```
*/
```

IF sats - grunden

```
if (villkor)
{
    alternativ1;
}
else
{
    alternativ2;
}
```



Jämförelse operatorer

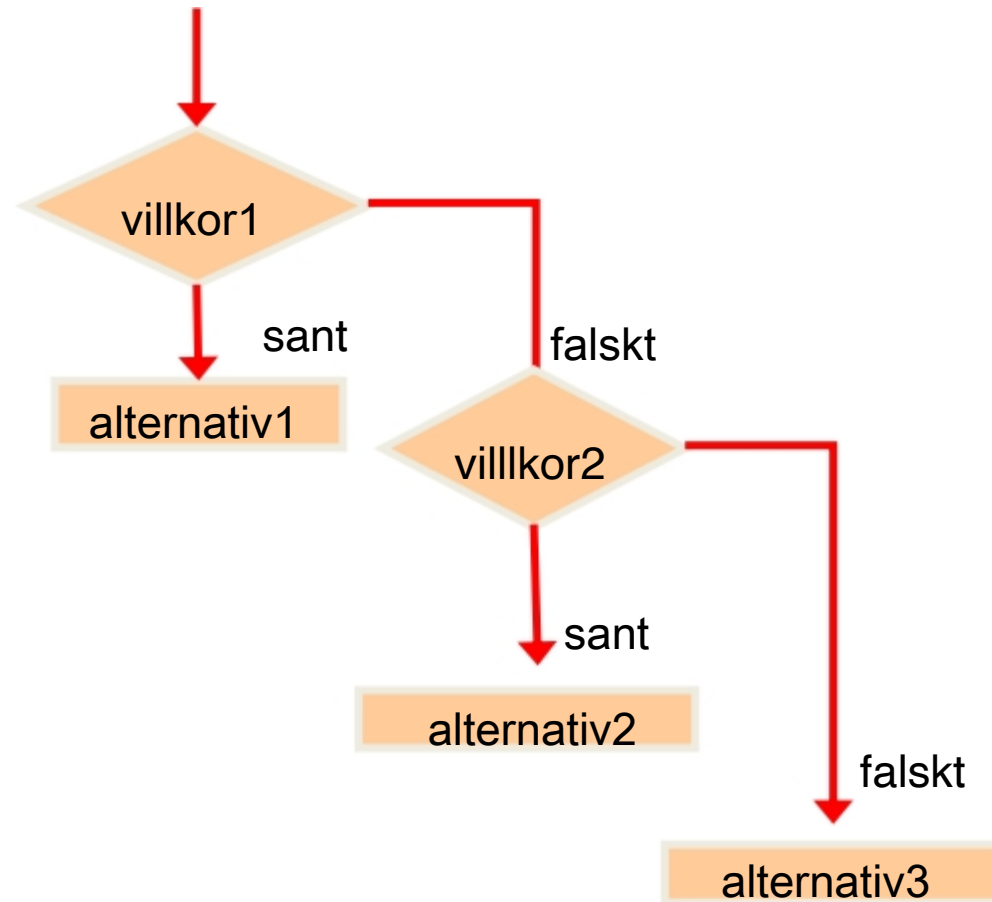
Operator	Beskrivning
<	Mindre än
>	Större än
<=	Mindre än eller lika med
>=	Större än eller lika med
==	Lika med
!=	Ej lika med

Logiska operatorer

Operator	Beskrivning
&&	OCH
	ELLER
!	INTE

IF - ELSE IF - ELSE

```
if (villkor1)
{
    alternativ1;
}
else if (villkor2)
{
    alternativ2;
}
else
{
    alternativ3;
}
```



Vad är en loop ?

- Iteration betyder upprepning. I programkod finns många exempel på när man behöver upprepa något flera gånger.
- En loop innebär att man upprepar något till dess att ett visst villkor är uppfyllt eller att man väljer att avbryta.

for-loop

Startvärde räknare

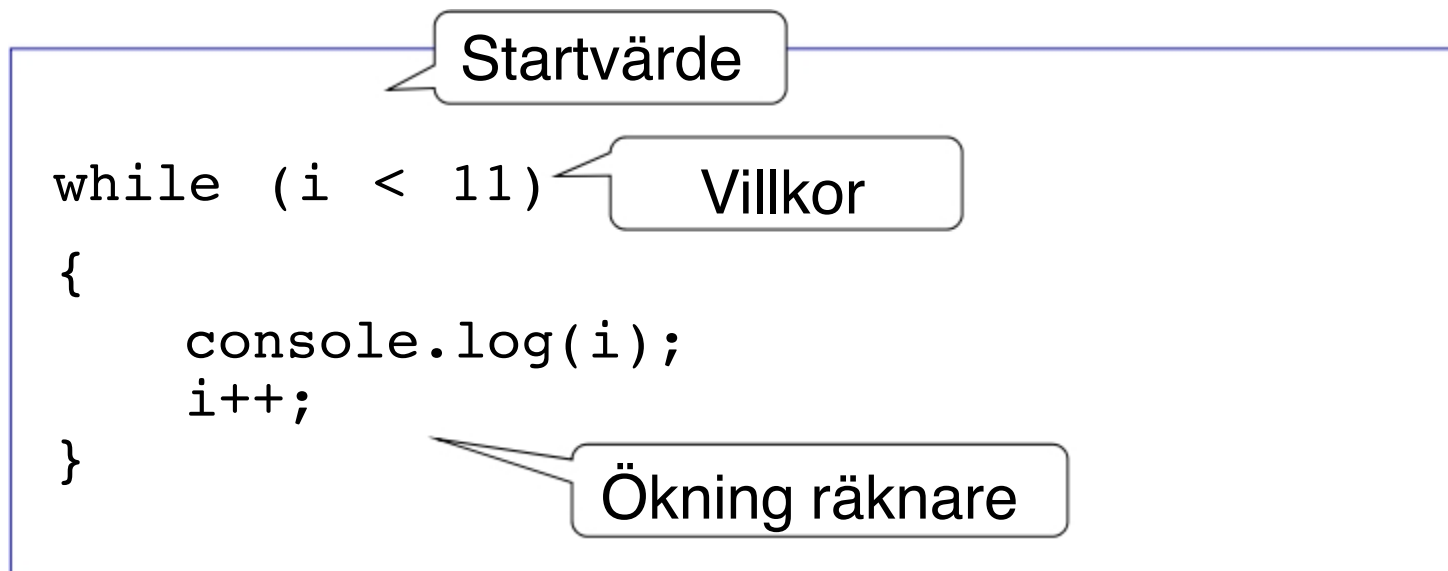
Villkor

Ökning räknare

```
for (let i = 1; i < 11; i++)  
{  
  console.log(i);  
}
```


while

- Ett annat sätt att skriva en loop är att använda `while` . Detta exempel gör samma sak som `for` loopen vi skrev tidigare.

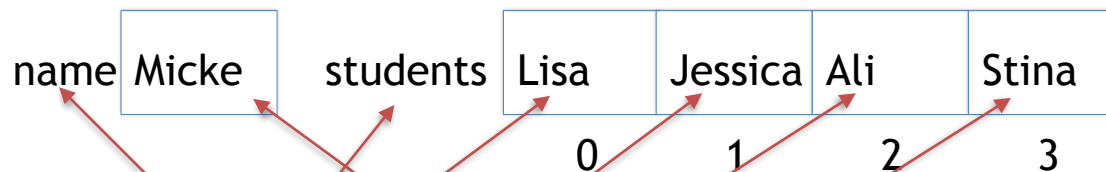


Kort om debugging

- VS Code har en inbyggd debugger.
 - <https://code.visualstudio.com/docs/editor/debugging>
- Nu kan vi följa koden och se vad olika variabler och uttryck innehåller.

Array

- En array är en speciell datatyp som kan innehålla flera värden.
- Varje "låda" i en array har ett index, som alltid börjar på 0.



```
let name = "Micke";  
let students = [  
  "Lisa",  
  "Jessica",  
  "Ali",  
  "Stina"  
];
```

```
console.log(name);  
console.log(studenter[0]);
```

Skapa en array

- Vi kan skapa en array med ett visst antal värden.
- Vi kan använda metoden `fill()` för att fylla arrayen med ett visst värde.

```
1  let arrA = new Array(7);  
2  
3  console.log(arrA);  
4  
5  arrA.fill(3);  
6  
7  console.log(arrA);  
8
```

```
[Running] node "/Users/  
[ <7 empty items> ]  
[  
  3, 3, 3, 3,  
  3, 3, 3  
]  
  
[Done] exited with code
```

Objekt

- *Objekt* är (i sin enklaste form) variabler med "undervariabler".
- Dessa "undervariabler" kallas *egenskaper*.

```
▶ {name: "Micke", age: 42, shoe_size: 43}
```

```
>
```

```
let person = {};  
  
person.name = "Micke";  
person.age = 42;  
person.shoe_size = 43;  
  
console.log(person);
```

```
let person = {  
  name: "Micke",  
  age: 42,  
  shoe_size: 43  
};  
  
console.log(person);
```

Objekt som egenskap

- Egenskaper kan innehålla olika datatyper, även objekt.

```
let student = {  
  name: {  
    first: "Mikael",  
    last: "Olsson"  
  },  
  class: "JavaScript"  
}  
  
console.log(student);
```

```
[Running] node "/Users/micke/www/projects/ec education/javascript/e  
{ name: { first: 'Mikael', last: 'Olsson' }, class: 'JavaScript' }
```

```
[Done] exited with code=0 in 0.058 seconds
```

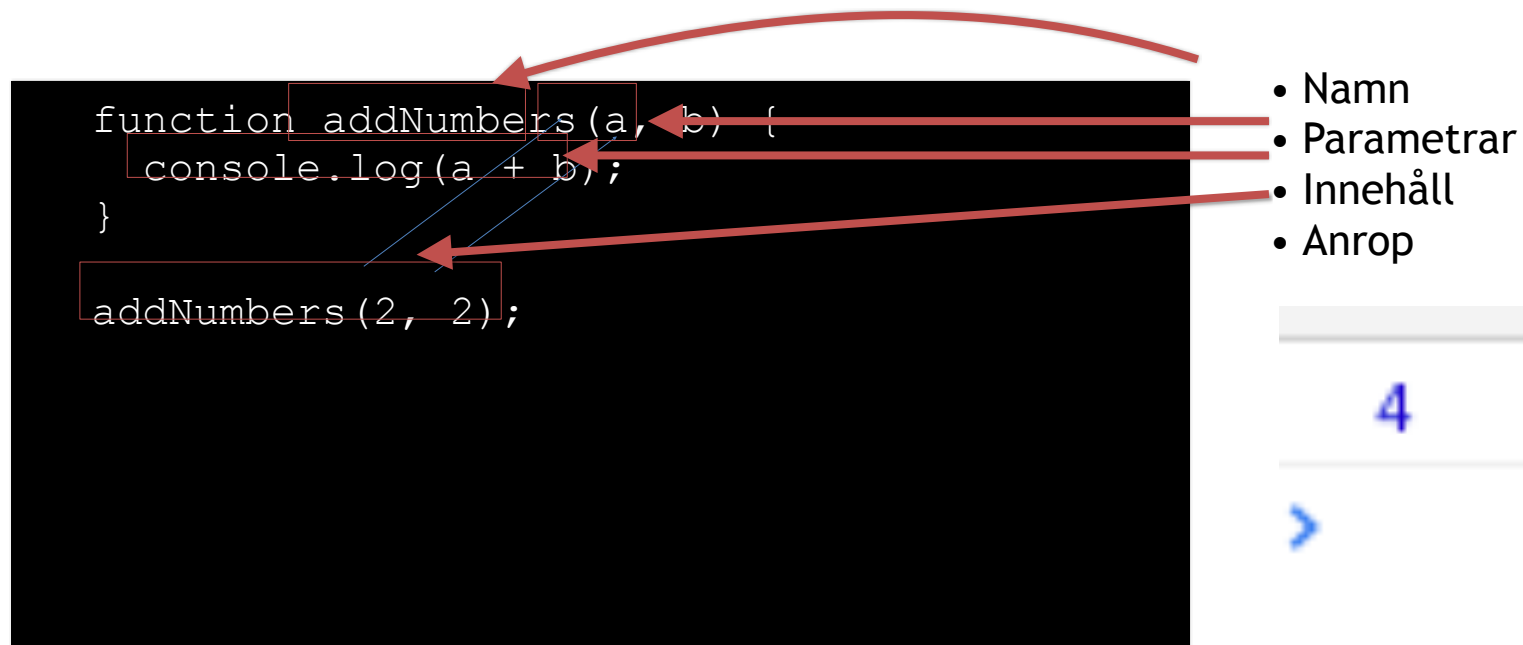
Skriva ut en egenskap

```
let student = {  
  name: {  
    first: "Mikael",  
    last: "Olsson"  
  },  
  class: "JavaScript"  
}  
  
console.log(student.class);  
console.log(student.name.first);
```

```
[Running] no  
JavaScript  
Mikael  
  
[Done] exited
```

Funktioner

- Funktionens beståndsdelar



Funktioner

- En funktion kan även returnera ett värde.

```
function addNumbers(a, b) {  
  return(a + b);  
}  
  
let result = addNumbers(2, 2);  
  
console.log(result);  
console.log(addNumbers(3, 3));
```



```
4  
6  
>
```

Spara en anonym funktion i en variabel

- En anonym funktion är en funktion utan namn.
- Vi kan spara en anonym funktion i en variabel.

```
let addNumbers = function (a, b) {  
  return a + b;  
}  
  
console.log( addNumbers( 3, 4 ) );
```

```
[Running] node  
7  
  
[Done] exited
```

Funktion som egenskap

- En egenskap i ett objekt kan innehålla en funktion. Den kallas då *metod*.

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

```
[Running] r  
hello  
  
[Done] exit
```

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
```

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* fungerer som for... of men tar en funktion som parameter.

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

Lisa

Jessica

Ali

Stina

Lägga till värde i array

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


Ta bort värde från array

- `pop` - tar bort från slutet av en array och returnerar värdet.
- `shift` - tar bort från början av en array och returnerar värdet.
- `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/>

Objekt

- *Objekt* är variabler med "undervariabler".
 - Man kan ha objekt i arrayer.
 - <http://jsfiddle.net/2r1ph08o/>

```
let students = [];
```

```
let person = {  
  name: "Micke",  
  age: 42,  
  shoe_size: 43  
};
```

```
students.push(person);
```

```
person = {  
  name: "Jessica",  
  age: 25,  
  shoe_size: 36  
};
```

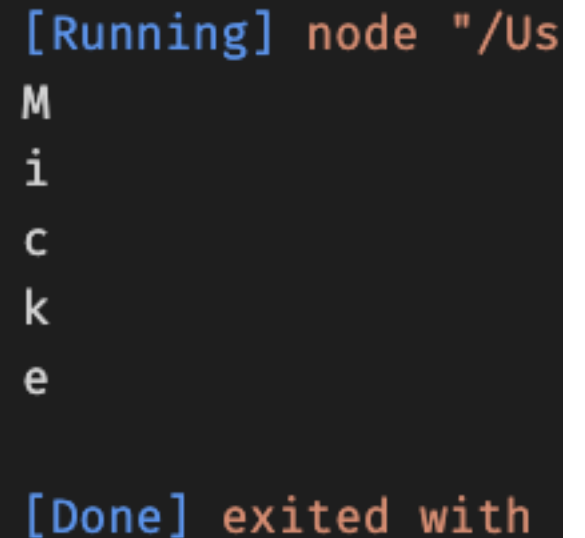
```
students.push(person);
```

```
console.log(students);
```

Strings

- En sträng är faktiskt en array av tecken.

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



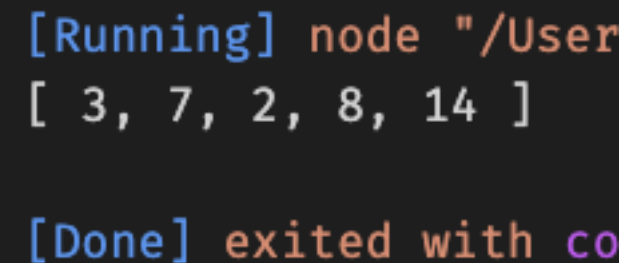
```
[Running] node "/Us  
M  
i  
c  
k  
e  
  
[Done] exited with
```

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
```

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);
```

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();
```


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);  
  }  
}
```

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();
```

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");
```


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>

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();
```

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

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.
- Skapa ett objekt av klassen, anropa *throw()* några gånger och skriv ut objektet efter varje gång så du ser att värdet ändras.

Tärningar

- Skriv klassen *Dice* med egenskapen *dice*, som ska innehålla en array.
- Låt konstruktorn ta emot en parameter som säger hur många tärningar vi vill skapa med default-värdet 5 och lägg 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.
- Skapa ett objekt av klassen, anropa *throw()* några gånger och skriv ut objektet efter varje gång så du ser att värdena ändras.

Nu fortsätter vi...

Entiteter

Klasser

Adressbok

Försök 1: Excel

	A	B	D	F	G	
1	Namn	Gatuadress	Tel hem	Arbetsgivare	Bil1	
2	Elin Nilsson	Kalmarsundsgatan 5	0321-321 54	Ulricehamns kommun	Volvo KCX 123	
3	Olle Andersson	Gatan 3	011-12 34 56			
4	Eva Ask	Vägen 5	013-98 65 32		Nissan PUK 456	
5						
6						
-						

Vad händer om vi vill lägga till ett nummer för en person?

Adressbok

	A	B	C	D	F	G
1	Namn	Gatuadress	Tel <u>arb</u>	Tel hem	Arbetsgivare	Bil1
2	Elin Nilsson	<u>Kalmarsundsgatan 5</u>	0321-123 45	0321-321 54	Ulricehamns kommun	Volvo KCX 123
3	Olle Andersson	Gatan 3		011-12 34 56		
4	Eva Ask	Vägen 5		013-98 65 32		Nissan PUK 456
5						
6						
7						

Adressbok

- Slöseri - flera fält är tomma
- Redundans - Samma värde förekommer flera gånger
- Oflexibelt - om vi behöver flera fält måste vi lägga till det för hela databasen
- Hur kan vi göra det bättre?

	A	B	C	D	E	F	G	H
1	Namn	Gatuadress	Tel <u>arb</u>	Tel hem	Tel mobil	Arbetsgivare	Bil1	Bil2
2	Elin Nilsson	Kalmarsundsgatan 5	0321-123 45	0321-321 54	070-123 456 78	Ulricehamns kommun	Volvo KCX 123	Hyundai PUF 321
3	Olle Andersson	Gatan 3		011-12 34 56				
4	Eva Ask	Vägen 5		013-98 65 32			Nissan PUK 456	
5								

ER-modellering (ERM)

- Entity-relationship model
 - Entities (entiteter)
 - Relationships (relasjoner)
 - Attributes (egenskaper)

Entitet

- Något med en egen existens, ett substantiv
- Kan vara ett fysiskt objekt eller en händelse



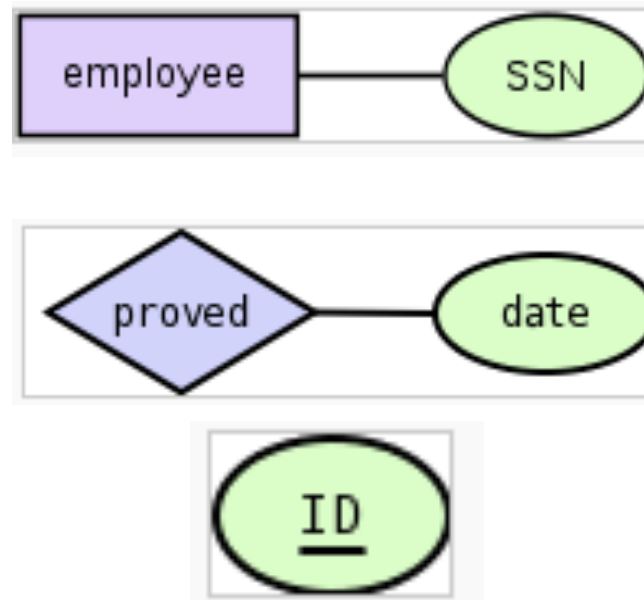
Två relaterade entiteter

Relationer

- Beskriver hur två eller flera entiteter hör ihop.
- Tänk verb som länkar ihop två eller flera entiteter.
 - Ett ägandeförhållande mellan ett företag och en dator.
 - Ett tillhörandeförhållande mellan en anställd och en avdelning.
 - Ett utförandeförhållande mellan en artist och en sång.

Egenskaper

- Entiteter och relationer kan ha egenskaper.



Uppdelning

- Vilka entiteter har vi i vår adressbok?

FirstName	LastName	Address	<u>Rooms</u>	Car 1	Car 2
Eva	Vik	Vägen 1		3 Volvo V70 – KXC122	Ford Ka – GRE479
Stina	Nilsson	Gatan 3		1 Ford Ka – ASD542	
Lars	Nilsson	Gatan 3		1	

Uppdelning

- Vilka entiteter har vi i vår adressbok?
 - Personer
 - Adresser/bostäder
 - Bilar

FirstName	LastName	Address	Rooms	Car 1	Car 2
Eva	Vik	Vägen 1		3 Volvo V70 – KXC122	Ford Ka – GRE479
Stina	Nilsson	Gatan 3		1 Ford Ka – ASD542	
Lars	Nilsson	Gatan 3		1	

Uppdelning

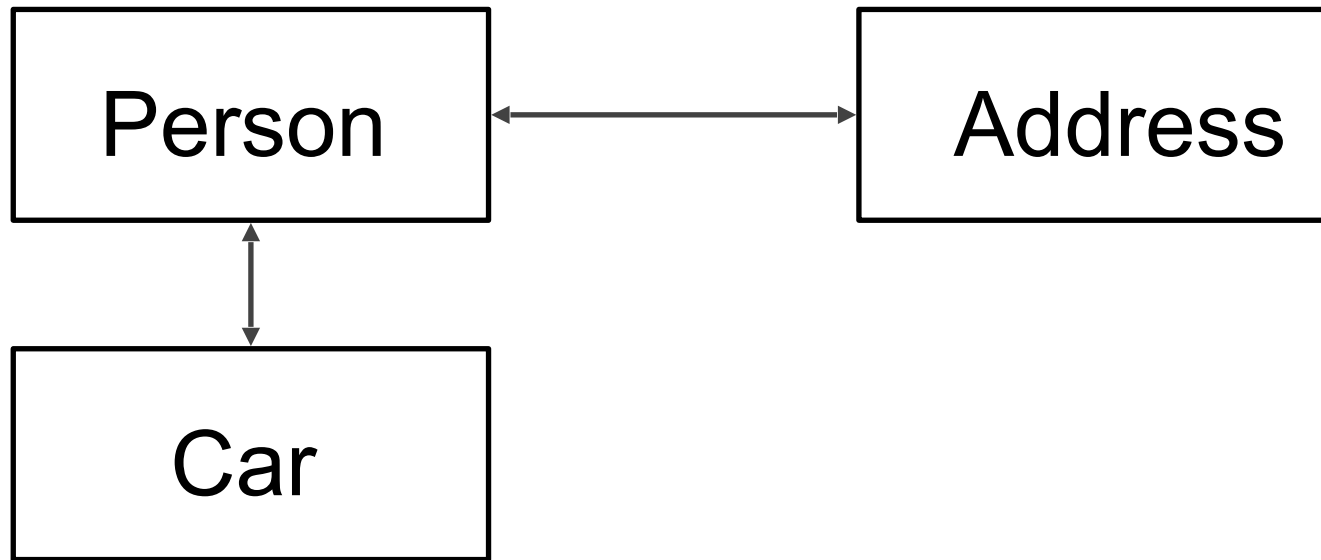
- Person
- House
- Car

FirstName	LastName
Eva	Vik
Stina	Nilsson
Lars	Nilsson

Address	<u>Rooms</u>
Vägen 1	3
Gatan 3	1

Car	<u>Registration</u>
Volvo V70	KXC122
Ford Ka	GRE479
Ford Ka	ASD542

Relation



Uppgift

- Gör en ER-modell över någon av följande:
 - Utbildning / kurser
 - TV-guide
 - Boksamling
- Övningen går ut på att identifiera lämpliga entiteter.

Klasser

- Klasser har oftast ett av två användningsområden.
 1. Entiteter, t ex Die. Egenskaper och metoder som beskriver en speciell sorts objekt.
 2. Organisation, t ex samlingar av objekt, metoder och egenskaper som hör ihop, t ex Dice eller en sammanhållande klass som håller ihop en applikation.

Labb 1

- Tanken är att ni ska samarbeta men lämna in varsitt svar.
- Vi kommer att titta på kriterierna tillsammans.
- Därefter vill jag att ni tillsammans funderar på hur man ska angripa problemet, vilka klasser och metoder som kan vara lämpliga, hur man ska lägga upp arbetet osv.
- Efter en gemensam genomgång av era planer kommer ni att få implementera lösningen i era grupper, ev med gemensamma genomgångar under dagen.
- Förhoppningsvis blir alla klara under dagen och kan lämna in sina labbar.

Labb 1 - e-handel

- Produkter ska ha egenskaper som sku (artikelnr), titel, beskrivning, antal i lager, pris. Den ska ha en metod som skriver ut produkten.
 - Kläder ska ha en klass som ärver från produkter och lägger till egenskapen storlek.
 - Leksaker ska ha en klass som ärver från produkter och lägger till egenskapen "ålder från".
- Det ska finnas en lager-klass som håller reda på alla produkter i butiken och hur många som finns i lager av varje produkt.
 - Det ska finnas en inventarie-metod som skriver ut en lista med alla produkter och hur många det finns av produkten.
 - Det ska finnas en metod som returnerar ett objekt baserat på sku. Om det t ex finns en produkt "jacka" med sku "456" så ska man kunna söka efter produkten genom att skicka artikelnumret till metoden och få tillbaka objektet.
- Det ska finnas en varukorg som innehåller 0- n produkter. Varukorgen ska höra ihop med en kund.
 - Varukorgen ska ha en metod för att skriva ut innehållet i korgen.
 - Varukorgen ska ha en metod för att räkna ihop summan av värdet av alla produkter i korgen.
 - Varukorgen ska ha metoder för att lägga till och ta bort produkter i varukorgen.
- Det ska finnas en kund-klass som håller reda på namn, orderhistorik och nuvarande varukorg.
 - Det ska finnas en köp-metod som lägger till varukorgens innehåll i kundens köphistorik med datum och minskar lagervärdet på produkterna.