

# Systemutveckling Ramverk

25 HP

# Kursupplägget

Föreläsning			Innehåll (FM)	Övningar (EM)	Inlämning
1	mån 25 mars	D	Introduktion Typescript (Basic Types, Type Inferrence, Variable Declarations, Iterators and Generators)	TypeScripts Hemsida TypeScript in 5 minutes + övning	
2	ons 27 mars	V	TypeScript forts. (Functions, Classes, Interfaces, Modules)		
3	fre 29 mars	V	Introduktion React (SPA, Virtuell DOM, Kap. 1-6)	Reacts Hemsida Main Concepts inklusive övningar i CodePen Kodövning (RP)	Inlämning 1 ges ut
4	mån 1 apr.	D	React Playground (1-initial-setup + 2-layout)		
5	ons 3 apr.	V	React forts. (Kap. 7-12)		
6	fre 5 apr.	V	Create React App TS Intro	Övning "Todo App"	Handledning
7	tis 9 apr.	D	React Playground (3-navigation-with-state) React Playground (4-navigation-with-routes)	Kodövning (RP)	
8	ons 10 apr.	V	React Playground (5-code-splitting) React Playground (6-error-boundary)	Kodövning (RP)	Inlämning 1 lämnas in
9	fre 12 apr.	V	React Playground (7-portals)	Kodövning (RP)	Inlämning 2 ges ut
10	mån 15 apr.	D	React Playground (8-code-split-app-start)	Kodövning (RP)	
11	tis 16 apr.	D	React Playground (9-api-lib-axios)	Kodövning (RP)	Handledning
12	tis 23 apr.	D	React Playground (10-context)	Kodövning (RP)	
13	tors 25 apr.	V	Tentaplugg		Inlämning 2 lämnas in
14	fre 26 apr.	V	TENTAMEN		



slack #ramverk

Ramverken vi kommer fokusera på är..

# React

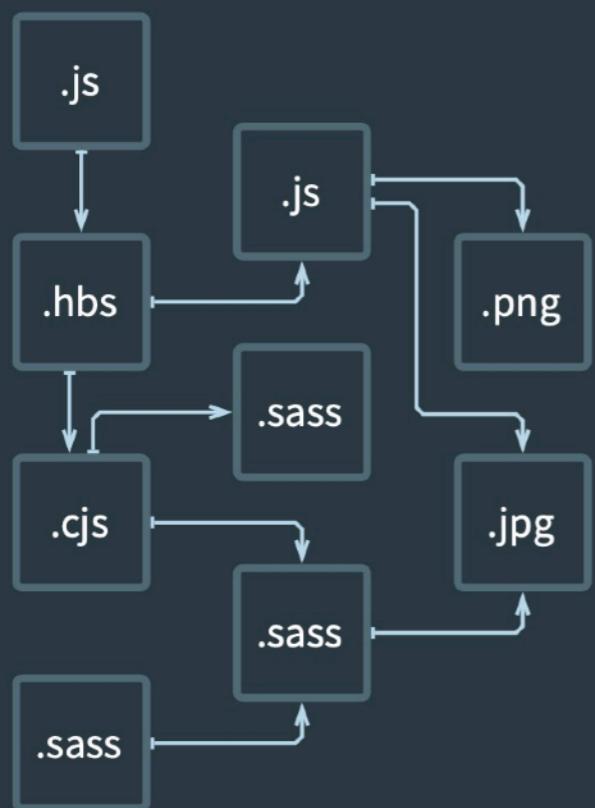
A JavaScript library for building user interfaces

# TypeScript

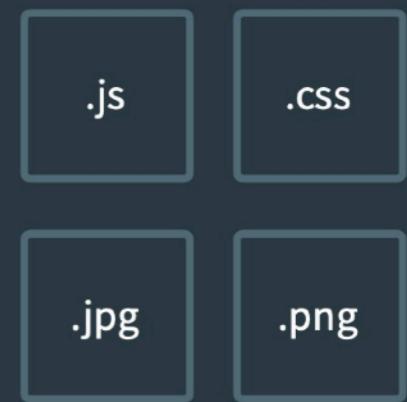
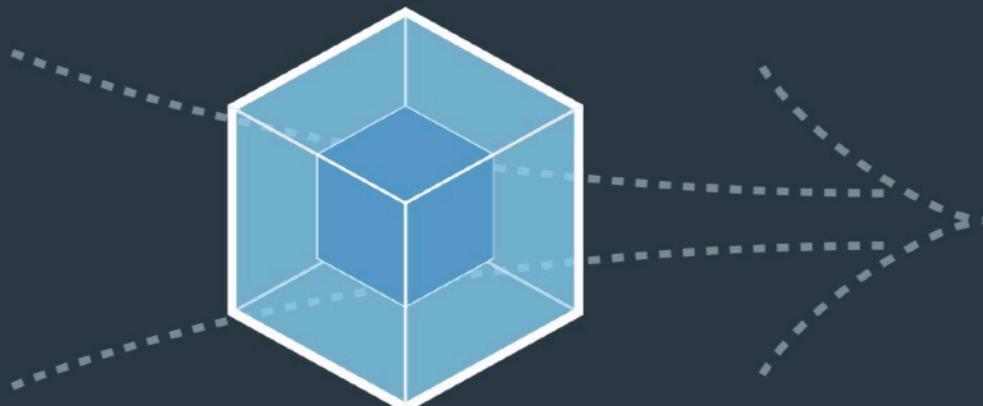
JavaScript that scales.

# Webpack

bundle your app

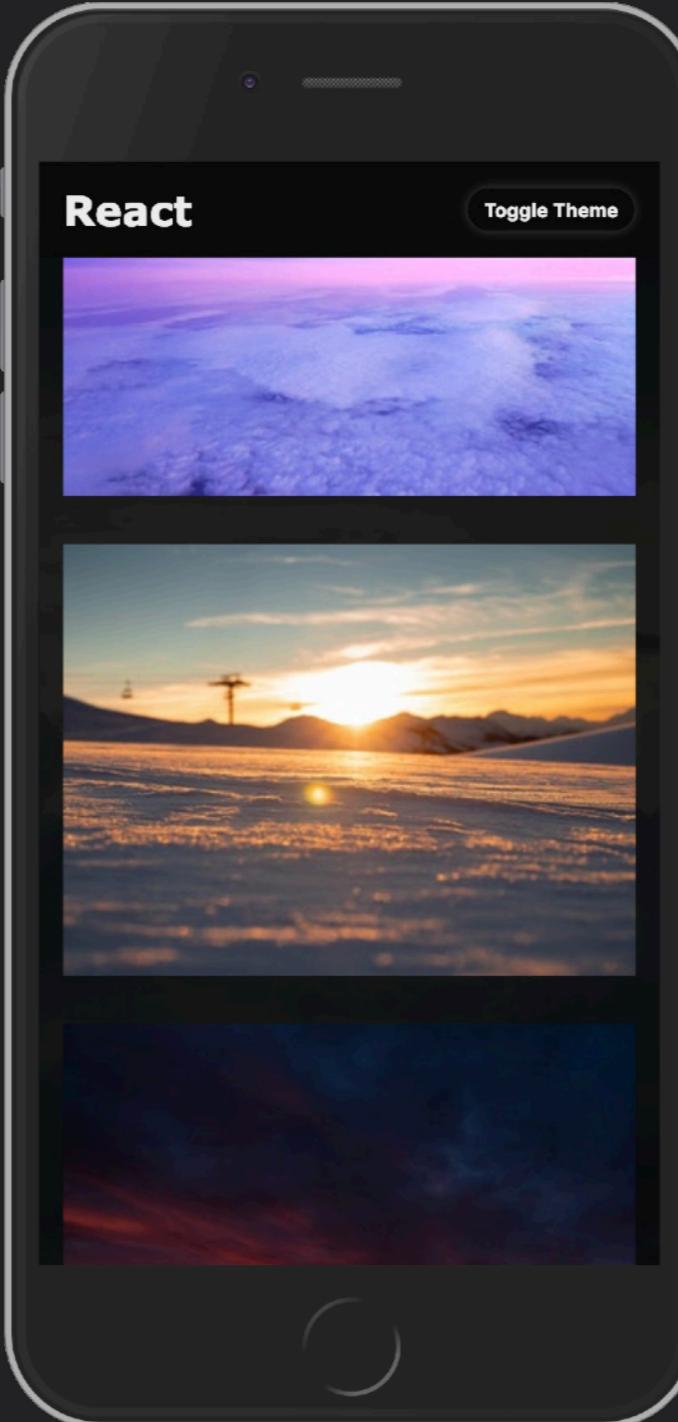
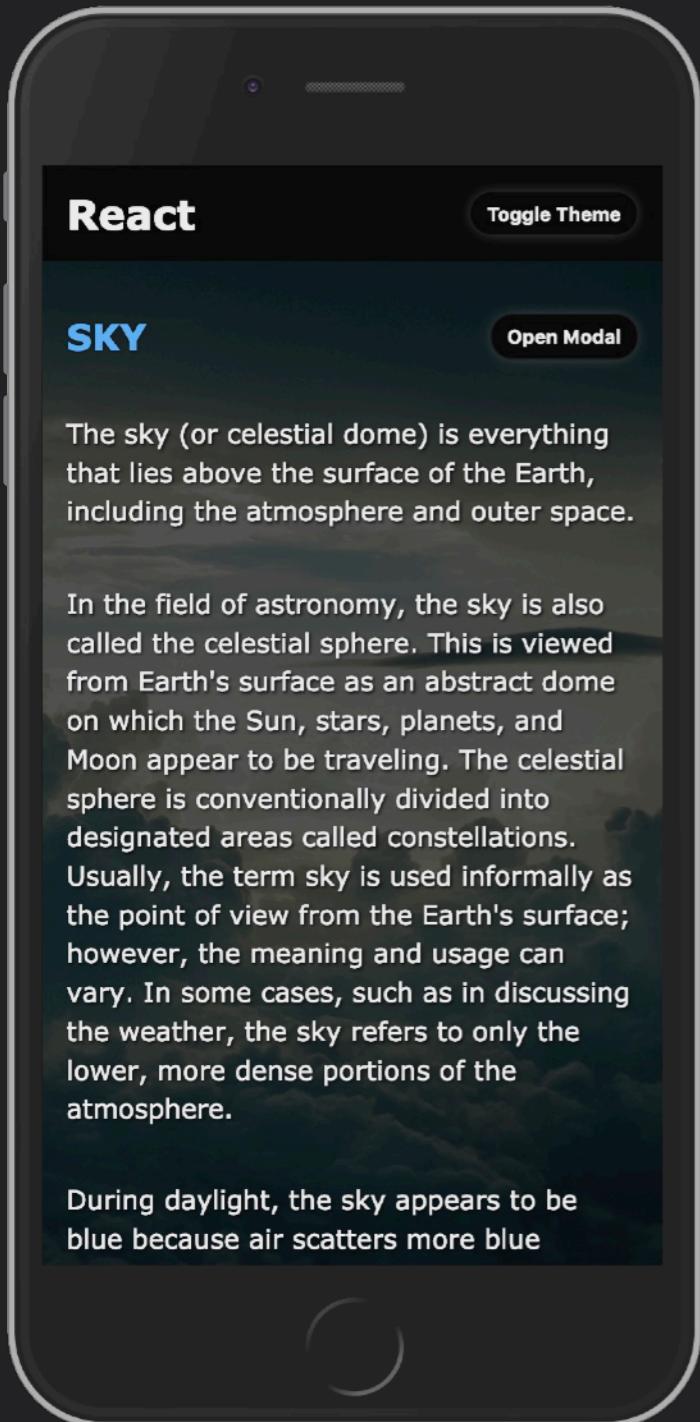


MODULES WITH DEPENDENCIES



STATIC ASSETS

# React Playground



## Branches

- ⚡ master
- ⚡ 1-initial-setup
- ⚡ 2-layout
- ⚡ 3-navigation-with-state
- ⚡ 4-navigation-with-routes
- ⚡ 5-code-splitting
- ⚡ 6-error-boundary
- ⚡ 7-portals
- ⚡ 8-code-split-app-start
- ⚡ 9-api-lib-axios
- ⚡ 10-react-context

Idag

# TypeScript

JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain Javascript.

Any browser. Any host. Any OS. Open source.

<https://github.com/Microsoft/TypeScript>

**Typer, typtolkning, variabler, loopar, funktioner & övning!**

# TypeScript - Know How

TypeScript filer använder filendelsen **.ts**

För att en webbläsare skall kunna tolka koden måste den kompileras till JavaScript.

TypeScript har en inbyggt kompilator vilket låter oss göra om en **.ts** fil till en **.js** fil.

För att kompilera en fil används terminalkommandot **tsc**

Exempelvis så här: **tsc script.ts**

# TypeScript - Know How

TypeScript är ett typat superset av JavaScript.

Typning innebär att koden blir mer beskrivande och fel som vanligtvis uppstår i applikationen kan fångas under utvecklingsprocessen.

Applikationerna blir stabilare men även snabbare och roligare att utveckla för oss programmerare.

Felen fångas upp när TypeScript kompilerar koden och vi utvecklare får felen i terminalen istället.

Felen visas även direkt i IDE'n i form av röda sträck med detaljerad information om man hovrar med musen över felen.

# Basic Types

# TypeScript - Basic Types

## Boolean

```
let isDone: boolean = false;
```

## Number

```
let decimal: number = 6;
let hex: number = 0xf00d;
let binary: number = 0b1010;
let octal: number = 0o744;
```

# TypeScript - Basic Types

## String

```
let color: string = "blue";
color = 'red';
```

### ( *Template Strings* )

```
let fullName: string = `Bob Bobbington`;
let age: number = 37;
let sentence: string = `Hello, my name is ${ fullName }.

I'll be ${ age + 1 } years old next month.`;
```

# TypeScript - Basic Types

## Array

```
let list: number[] = [1, 2, 3];
```

## Tuple

```
// Declare a tuple type
let x: [string, number];
// Initialize it
x = ["hello", 10]; // OK
// Initialize it incorrectly
x = [10, "hello"]; // Error
```

Värden i en array eller tupel hämtas med samma syntax som för array'er i JS

```
console.log(x[0].substr(1)); // OK
console.log(x[1].substr(1)); // Error, 'number' does not have 'substr'
```

# TypeScript - Basic Types

## Any

```
let notSure: any = 4;
notSure = "maybe a string instead";
notSure = false; // okay, definitely a boolean
```

We may need to describe the type of variables that we do not know when we are writing an application. These values may come from dynamic content, e.g. from the user or a 3rd party library.

The **any** type is a powerful way to work with existing JavaScript, allowing you to gradually opt-in and opt-out of type-checking during compilation.

# TypeScript - Basic Types

## Void

```
function warnUser(): void {  
    console.log("This is my warning message");  
}
```

**void** is a little like the opposite of **any**: the absence of having any type at all.

# TypeScript - Basic Types

## Null and Undefined

```
// Not much else we can assign to these variables!
let u: undefined = undefined;
let n: null = null;
```

By default **null** and **undefined** are subtypes of all other types. That means you can assign **null** and **undefined** to something like **number**.

However, when using the **--strictNullChecks** flag, **null** and **undefined** are only assignable to **void** and their respective types. This helps avoid many common errors.

# TypeScript - Basic Types

## Type assertions

```
let someValue: any = "this is a string";  
  
let strLength: number = (someValue as string).length;
```

Sometimes you'll end up in a situation where you'll know more about a value than TypeScript does. Usually this will happen when you know the type of some entity could be more specific than its current type.

# Type Inference

# TypeScript - Type Inference

När variabler skapas kan vi låta TypeScript välja ut en lämplig typ för variablen åt oss. I exemplet nedan kommer varabeln **x** bli typad till **number**.

```
let x = 3;
```

I exemplet nedan kommer varabeln **input** bli typad till **number[ ]**.

```
let input = [1, 2];
```

# TypeScript - Type Inference

## Best Common Type

```
let x = [0, 1, null];
```

```
let zoo = [new Rhino(), new Elephant(), new Snake()];
```

När en bästa gemensan typ inte hittas, blir resultatet av typ-tolkningen en **Union Type**. I detta fallet **(number | null)[ ]** samt **(Rhino | Elephant | Snake)[ ]**.

Idealt hade kanske varit att själv typa variablen som en array av **Animal**

```
let zoo: Animal[] = [new Rhino(), new Elephant(), new Snake()];
```

# Variable Declarations

# TypeScript - Variable Declarations

**var**

```
var a = 10;
```

**let**

```
let hello = "Hello!";
```

**const**

```
const numLivesForCat = 9;
```

## A note about let

You may've noticed that so far, we've been using the `let` keyword instead of JavaScript's `var` keyword which you might be more familiar with. The `let` keyword is actually a newer JavaScript construct that TypeScript makes available. We'll discuss the details later, but many common problems in JavaScript are alleviated by using `let`, **so you should use it instead of `var` whenever possible.**

# TypeScript - Variable Declarations

# Problematik med var

```
for (var i = 0; i < 10; i++) {
    setTimeout(function() { console.log(i); }, 100 * i);
}
```

# TypeScript - Variable Declarations

## Problematik med var

```
for (let i = 0; i < 10 ; i++) {
    setTimeout(function() { console.log(i); }, 100 * i);
}
```

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

# TypeScript - Variable Declarations

## Destructuring (array)

```
let input = [1, 2];
let [first, second] = input;
console.log(first); // outputs 1
console.log(second); // outputs 2
```

I många fall kan koden bli mer läsbar om man väljer att bryta ut innehållet i en array/objekt till variabler.

```
let [first, ...rest] = [1, 2, 3, 4];
console.log(first); // outputs 1
console.log(rest); // outputs [ 2, 3, 4 ]
```

Detta är något vi kommer använda oss mycket av framöver i kursern när vi jobbar med React.

```
let [first] = [1, 2, 3, 4];
console.log(first); // outputs 1
```

# TypeScript - Variable Declarations

## Destructuring (objekt)

```
let o = {  
    a: "foo",  
    b: 12,  
    c: "bar"  
};
```

```
let { a, b } = o;
```

```
let { a, ...passthrough } = o;  
let total = passthrough.b + passthrough.c.length;
```

I många fall kan koden bli mer läsbar om man väljer att bryta ut innehållet i en array/objekt till variabler.

Detta är något vi kommer använda oss mycket av framöver i kursern när vi jobbar med React.

```
let { a: newName1, b: newName2 } = o;
```

(vi kan till och med döpa om variablerna)

# TypeScript - Variable Declarations

## Spread

Med spread operatorn ... (tre punkter) kan vi "plocka" ut allt innehåll i en array/objekt för att tex bygga upp ett nytt objekt.

```
let defaults = { food: "spicy", price: "$$", ambiance: "noisy" };
let search = { ...defaults, food: "rich" };
```

Detta kommer vi använda en hel del under kursen.

# Iterators & Generators

# TypeScript - Iterators & Generators

Precis som i JavaScript finns **while** loopar, **for** loopar, **for..of** lopar och **for..in** loopar i TypeScript.

# TypeScript - Iterators & Generators

**while**

```
let index = 0;
while (index < 100) {
|   console.log(index) // 0, 1, 2...
```

**for**

```
for (let i = 0; i < 100; i++) {
|   console.log(i) // 0, 1, 2...
```

# TypeScript - Iterators & Generators

## for..of

```
let someArray = [1, "string", false];

for (let entry of someArray) {
    console.log(entry); // 1, "string", false
}
```

# TypeScript - Iterators & Generators

## for..of vs for..in

```
let list = [4, 5, 6];

for (let i in list) {
    console.log(i); // "0", "1", "2",
}

for (let i of list) {
    console.log(i); // "4", "5", "6"
}
```

for..in loopar över index  
värdena in en array istället  
för innehållet i arrayen.

for..in loopen går även att  
använda på objekt om man  
vill gå över objektets  
properties.

# Functions

# TypeScript - Functions (basics)

## Funktioner i dess simplaste form (samma som i JS)

```
// Named function
function add(x, y) {
    return x + y;
}

// Anonymous function
let myAdd = function(x, y) { return x + y; };
```

Det går även att komma åt variabler som är deklarerade utanför funktionen precis som i JS. Detta kommer dock med en del nackdelar och bör i många fall undvikas men är oavsett viktigt att veta om.

```
let z = 100;

function addToZ(x, y) {
    return x + y + z;
}
```

# TypeScript - Functions (basics)

## Typed Functions

```
function add(x: number, y: number): number {
    return x + y;
}

let myAdd = function(x: number, y: number): number { return x + y; };
```

# TypeScript - Functions (basics)

## Writing the function type

```
let myAdd: (x: number, y: number) => number =  
  function(x: number, y: number): number { return x + y; };
```

När vi skriver funktionstypen är namnen bara till för läsbarheten och måste inte stämma överens. Vi hade lika gärna kunnat skriva så här:

```
let myAdd: (baseValue: number, increment: number) => number =  
  function(x: number, y: number): number { return x + y; };
```

# TypeScript - Functions (basics)

## Infering the types

```
// myAdd has the full function type
let myAdd = function(x: number, y: number): number { return x + y; };

// The parameters 'x' and 'y' have the type number
let myAdd: (baseValue: number, increment: number) => number =
  function(x, y) { return x + y; };
```

# Läsanvisningar

## TypeScript Handbook

<https://www.typescriptlang.org/docs/home.html>

### Kapitel.

Basic Types

Type Inference

Variable Declarations

Iterators & Generators

Functions

# Nästa lektion

Funktioner, klasser, interfaces, moduler  
och mer!

# Resten av dagen

## TypeScript in 5 minutes

<https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>

## Övningsuppgift

# Övningsuppgift (be creative)

**Skapa en valfri hemsida och lägg lite extra fokus på sidans design.**  
Det kan vara en ny portfolio sida, ett simpelt spel i canvas, eller  
något annat som du tycker är kul.

Försök att använda er av flera stycken TypeScript filer som sedan  
kompileras om till JavaScript filer och inkluderas i .html filen med  
hjälp av <script> taggen.

I slutet av lektionen kommer vi ha redovisningstid där ni som vill  
presentera det ni har gjort, får chans att göra det och berätta hur  
och vad ni har använt TypeScript till.

**Lycka till - ser fram emot att se vad ni skapar!**

# Tack