Typescript/Javascript Programmering

Introduktion

- Meet and greet
- Forventningsafstemning

Overblik

- Dag 1: Javascript/Ecmascript
- Dag 2: Typescript
- Dag 3: Projekt og opsamling på dag 1 og 2

Bogmærker

- Official documentation: typescriptlang.org
- Typescript Deep Dive: basarat.gitbook.io/typescript
- Jack Herrington Youtube channel
- Typescript Playground: https://www.typescriptlang.org/play
- Code Sandbox
- Can I Use

Hvad er Javascript/Ecmascript?

- Født i browseren
- Dynamisk typed (run-time)
- Fortolket/Interpreted
- Funktionelt- og objektorienteret
- Event-drevet
- Bundet af browser kompatibilitet

Tidslinie

- 1997: Ecmascript
- ES4 droppet
- 2009: ES5
- 2009: Første release af Node.js
- 2015: ES6 - "Next generation javascript"
- 2016-2022: Løbende årlige udgivelser: ES7, ES8, ES9, ES10, ES11, ES12, ES13

Hvad er Typescript?

- Supersæt af Javascript
- Statically typed (compile-time)

Supersæt af Javascript

Statically typed

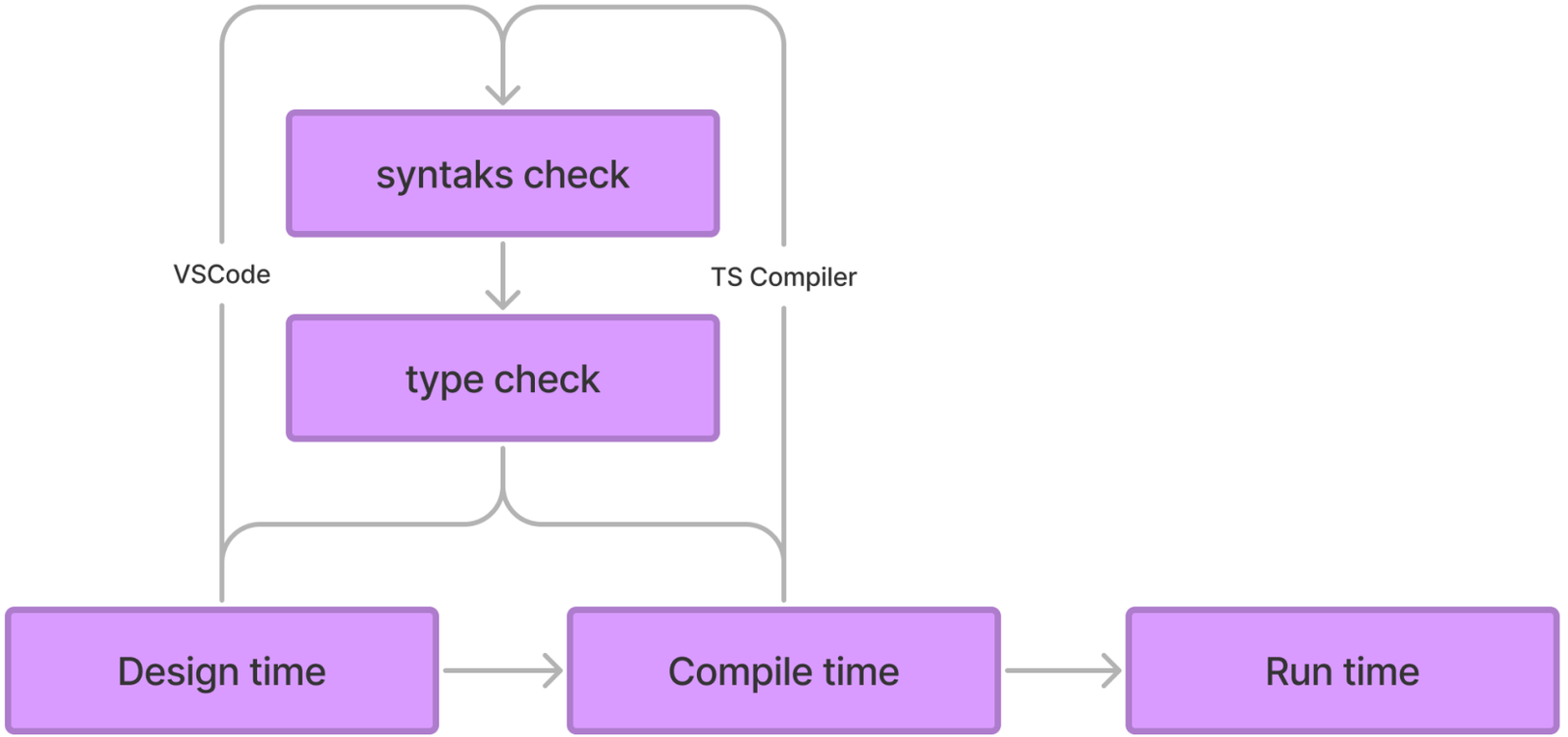
Dette er ok:

```
let x:number = 1;
const multiplyByTwo = (x:number) => {
  return x * 2;
}
```

Dette vil fejle:

```
let x:string = "1";
const multiplyByTwo = (x:number) => {
  return x * 2;
}
```

Compilation 1/2



Compilation 2/2

- Design time
  - Redigerer projektet i VSCode
- Compile time
  - Projektet konverteres til Javascript
- Run time
  - Projektet kører som Javascript i browser eller Node.js
  - Bemærk: Typescript typer eksisterer ikke i runtime

Hvorfor Typescript?

- Tillid
- Hurtige feedback loops
  - Design time fejl
    - Feedback i editor > "Fail fast"
  - Compile time fejl
- Behov for mindre valideringslogik og færre unit tests
- Future javascript now

Værktøjskasse

- Installer NPM og Node.JS
- Installer Yarn
- Installer VSCode

NPM 1/2

NPM er ansvarlig for pakkestyring (package manager) i Javascript. Vi bruger NPM til at installere og styre afhængigheder. Sørg for at have Node.js (minimum v.16.x) installeret.

Download Node.js

NPM 2/2

På Linux/Unix/Mac kan du anvende en package manager til at installere Node.js:

- Mac: HomeBrew
- Linux: APT

Yarn

- Yarn er et "lag" ovenpå NPM som tilbyder nogle forbedringer i forhold til bl.a. performance.
- Yarn er ikke et krav, men kodeeksempler i kurset vil tage udgangspunkt i Yarn, da Yarn har bevæget sig i retning af at blive den nye industri standard for udvikling i Javascript.

Installere Yarn globalt:

```
$ npm install yarn --global
```

VSCode

VSCode er ikke et krav (du kan f.eks. bruge WebStorm, eller terminal/VIM), MEN...

- VSCode er "bredt accepteret" som den bedste kode-editor og IDE
- God Typescript support
- Kører på alle platforme (og i browser med Github Codespaces)

Pause

Javascript syntaks

I den følgende sektion gennemgår vi grundlæggende Javascript syntaks.

Da Typescript er et supersæt af Javascript er alt hvad vi lærer her også gyldig Typescript.

Variable

Du kan bruge forskellige keywords `var`, `let` og `const` til at definere en variabel.

```
var a = "Jens";
console.log(a); // "Jens"
let b = "Børge";
console.log(b); // "Børge"
const c = "Hansen";
console.log(c); // "Hansen"
```

Variable - re-assignment

Hvis du anvender `var` eller `let` kan du ændre værdien af variablen senere i koden:

```
let a = "Jens";
let b = "Hansen";
b = "Jensen";
console.log(a + " " + b); // "Jens Jensen"
```

Variable - konstante værdier

Hvis du benytter `const` kan du ikke ændre værdien af variablen senere:

```
const a = "Jens";
const b = "Hansen";
b = "Jensen"; // Fejl
```

Variable - forskel på `var` og `let`

`var` er function scoped. `let` er block scoped.

```
function run() {
  var foo = "Foo";
  let bar = "Bar";
  console.log(foo, bar); // Foo Bar
  {
    var moo = "Mooo"
    let baz = "Bazz";
    console.log(moo, baz); // Mooo Bazz
  }
  console.log(moo); // Mooo
  console.log(baz); // ReferenceError
}
run();
```

Varable uden keyword

Hvis man deklarerer en variabel uden keyword bliver den automatisk tilføjet til global scope. Dette er en dårlig praksis og skal undgås, da du risikerer at overskrive andre variable i global scope - f.eks. fra et library du kunne have loaded.

```
// Gør aldrig dette
a = "Jens"; // Implicit global
// Skal du bruge en global variabel, så gør det eksplicit med window:
window.a = "Jens"; // Explicit global
```

Variable - best practice

- Brug `const` så ofte som muligt. Det er mere forudsigeligt.
- Brug `let` hvis du skal ændre værdien af variablen senere i koden.
- Undgå at bruge `var` med mindre du skriver kode der skal understøtte ES5, da function scope er forvirrende for udviklere med baggrund i andre sprog og er skyld i mange bugs.

Dynamiske typer

Javascript er et dynamisk sprog, hvilket betyder at du ikke behøver at definere datatypen af en variabel.

Datatypen bliver automatisk bestemt af værdien:

```
"Jens" // string
42 // number
true // boolean
```

Primitive datatyper i Javascript

Javascript har 7 primitive datatyper:

```
"Jens" // string
42 // number
true // boolean
null // null
undefined // undefined
Symbol("foo") // symbol
BigInt(9007199254740991) // bigint
```

Strenge (strings)

En `string` er en tekststreng. Og kan defineres med enkelt eller dobbelt anførselstegn.

```
"Jens"
'Peter'
'Streng der indeholder "dobbelte" anførselstegn';
"Streng der indeholder 'enkelte' anførselsetegn.";
```

Sammensatte strenge

```
const samlet = "Jens" + " " + "Peter"; // "Jens Peter"
const matematik = "1" + "2"; // "12"
```

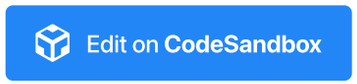
Template literals

- Anvender backticks ` i stedet for anførselstegn.
- Særlig syntaks til at indsætte variabler i strengen.

```
const firstName = "Jens";
const lastName = "Hansen";
const samlet = `${firstName} ${lastName}`; // "Jens Hansen"
const samlet2 = `Mit navn er ${firstName} ${lastName} og jeg bor i Odder.`; // "Mit navn er Jens Hansen og
```

Øvelse

Strenge assignment og template literals:



Pause

Tal i javascript (numbers)

- Modsat andre programmeringssprog anvendes typen `number` til både heltal og decimaltal.
- BigInt kan anvendes til at repræsentere heltal større end 2^53 - 1.

Aritmetik

Operator	Beskrivelse
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (rest)
++	Increment
--	Decrement

Aritmetik (eksempler)

```
1 + 2 // 3 // Addition
7 - 2 // 5 // Subtraction
2 * 2 // 4 // Multiplication
2 ** 3 // 8 // Exponentiation
2 / 2 // 1 // Division
3 % 2 // 1 // Modulus (rest)
let a = 1; a++; // 2 // Increment
let b = 10; b--; // 9 // Decrement
```

Konvertere strenge til tal 1/3

Hvis man på forhånd ved en streng indeholder et tal kan man konvertere den til et tal med `Number()`, `parseInt()` eller `parseFloat()`.

`parseInt()` returnerer det først heltal den finder i strengen:

```
parseInt("32px") // 32
```

Konvertere strenge til tal 2/3

`parseFloat()` returnerer det først decimaltal den finder i strengen:

```
parseFloat("32.2px") // 32.2
```

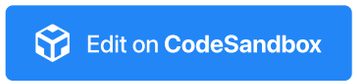
Konvertere strenge til tal 3/3

`Number()` forsøger at konvertere hele strengen til et tal:

```
Number("32") // 32
Number("32.2") // 32.2
Number("32px") // NaN
```

Øvelse

Artimetik og matematiske operatorer:



Equality

- Loosely equal: ==
  - Javascript konverterer værdierne til samme type før sammenligning.
- Strictly equal: ===
  - Javascript sammenligner værdierne uden konvertering.

MDN: Equality comparisons and sameness

Kontrolstrukturer - if

```
if (userInput % 2) {
  /* Hvis user input er ulige */
  return 'a';
}
```

Kontrolstrukturer - if/else

```
if (userInput % 2) {
  /* Hvis user input er ulige */
  return 'a';
} else {
  /* Hvis user input er ulige */
  return 'b';
}
```

Ternary operator

Samme princip som `if/else` men på en linie:

```
return userInput % 2 ? 'a' : 'b';
```

AND/OR

AND:

```
return a && b;
```

OR:

```
return a || b;
```

Nullish coalescing operator

```
let accountAmount = 0;
return accountAmount || 'No account found'; // 'No account found'
...
let accountAmount = 0;
return accountAmount ?? 'No account found'; // 0
```

Video by Web Dev Simplified

Pause

Iteration - loops

- Loops (iterators) anvendes når man vil gentage en stump kode flere gange.
- Der findes 3 forskellige typer loops i javascript:
  - `for`
  - `while`
  - `do while`

For loop

```
for (let i = 0; i < 10; i++) {
  console.log(i); // 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
}
```

While loop

```
let i = 0;
while (i < 10) {
  console.log(i); // 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  i++;
}
```

do...while loop

```
let i = 0;
do {
  console.log(i); // 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  i++;
} while (i < 10);
```

Øvelse

Iterators

Edit on **CodeSandbox**

Pause

Objekter

Et javascript objekt er en entitet der definerer egenskaber (properties) og data.

Eksempel:

```
let person = {
  name: "John Doe",
  age: 30,
  occupation: "Software Developer",
};
```

Indlejrede objekter

Et javascript objekt kan indeholde indlejrede objekter:

```
let person = {
  name: "John Doe",
  age: 30,
  occupation: "Software Developer",
  address: {
    street: "123 Main St",
    city: "Anytown",
    state: "CA",
    zip: "12345"
  },
};
```

Tilgå objekt værdier

Man kan tilgå objekt værdier på to måder:

```
// Dot notation
const name = person.name;
// Bracket notation
const name = person["name"];
// Nestede værdier
const street = person.address.street;
```

Object destructuring

Man kan anvende object destructuring som en genvej til at oprette en ny variabel og tildele den en værdi fra et objekt på samme tid. Man "plukker" værdien ud af objektet:

```
const { name } = person;
console.log(name); // "John Doe"
// Eller flere værdier på samme tid:
const { name, age, occupation } = person;
console.log(name, age, occupation); // "John Doe", 30, "Software Developer"
```

Arrays

Et javascript array er en liste af værdier.

```
const list = [1, 2, 3, 4, 5];
```

I javascript kan et array indeholde værdier af forskellige typer:

```
const list = [1, "to", { id: 3} , 4, "5"];
```

Array destructuring

Ligesom objekter kan man anvende destructuring med arrays:

```
const list = [1, 2, 3 , 4, 5];
const [first, second, third] = list;
console.log(first, second, third); // 1, 2, 3
```

JSON

- JSON står for JavaScript Object Notation
- Gemmes i en .json fil
- Bruges ofte til at gemme data til en fil eller til at udveksle data med et API

JSON

JSON ligner et javascript objekt, men bemærk at at alle værdier i JSON skal være i dobbelte anførselstegn:

```
{
  "name": "John Doe",
  "age": "30",
  "occupation": "Software Developer",
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA",
    "zip": "12345"
  }
}
```

JSON Parse

En json fil kan importeres i en javascript fil og konverteres til et object med JSON.parse();

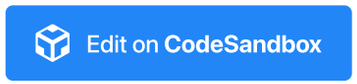
```
import person from './person.json';
const obj = JSON.parse(person);
```

Ligeledes kan man serialisere et objekt til JSON med JSON.stringify():

```
const obj = { name: "John Doe", age: 30 };
const jsontext = JSON.stringify(obj);
```

Øvelse

Objekter og Arrays:



Pause

Funktioner

- Funktioner er en af de vigtigste byggesten i javascript.
- En funktion er en sekvens af instruktioner der er pakket som en enhed og kan genbruges.
- En funktion er en blok kode der udfører et stykke arbejde når den bliver kaldt.
- En funktion kan tage parametre og returnere en værdi.
- En funktion kan også modificere værdier udenfor sit eget scope (side effects)

Funktioner (syntaks)

```
function sum(a, b) { return a + b }
```

Anatomien af en funktion 1/2

```
function sum(a, b) { return a + b }
```

- Navn: sum
- Parametre: a, b
- Returværdi: a + b

Anatomien af en funktion 2/2

```
function capitalize(input) {
  const output = input.toUpperCase();
  return output;
}
```

- Navn: capitalize
- Parametre: input
- Returværdi: output

Anonyme funktioner

Anonym funktion:

```
function(a, b) {return a + b }
```

Anonym funktion med variabel:

```
const sum = function(a, b) {return a + b }
sum(1, 2); // 3
```

Arrow funktion

```
const capitalize = (input) => { return input.toUpperCase() }
```

Hvis der kun er en parameter kan paranteser undlades:

```
const capitalize = input => { return input.toUpperCase() }
```

Hvis funktionen returnerer på en linie kan blok og return statement undlades:

```
const capitalize = input => input.toUpperCase();
```

Funktioner (3/3)

Det kaldes en metode når en objekt- eller klasse property har en funktion som værdi:

```
const someObject = {
  val: 123,
  callMe: (a) => { return % a }
}
const result = someObject.callMe(2);
```

Funktioner - Default argumenter

Man kan definere default værdier for parametre:

```
const sum = (a = 1, b = 2) => a + b;
sum(7,3) // 10
sum() // 3
```

Hoisting

Virker:

```
doSomething();
function doSomething() { return "yes" }
```

Virker ikke:

```
doSomethingElse();
const doSomethingElse = () => { return "yes" }
```

Funktioner som returnerer funktioner

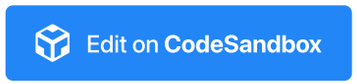
```
function calculator(operation) {
  if (operation === 'add') {
    return function(a, b) { return a + b; }
  } else if (operation === 'subtract') {
    return function (a, b) { return a - b };
  }
}
calculator('add)(1, 2); // 3
```

Funktion som tager en funktion som parameter

```
function processWithCallback(x, y, callback) {
  const result = x * y;
  callback(result);
  return result;
}
process(3, 7, (a) => { console.log(a)})
process(5, 2, (a) => { persistToDatabase(a)})
```

Øvelse

Funktioner, objekter og arrays



Pause

Guard clauses (1/2)

Without guard clause:

```
function processRequest(paramA, paramB) {
  if (typeof paramA !== 'undefined' || typeof paramB !== 'undefined' ) {
    const result = paramA + paramB;
    return result;
  } else {
    return null;
  }
}
```

Guard clauses (2/2)

With guard clause:

```
function processRequest(paramA, paramB) {
  if (typeof paramA === 'undefined' || typeof paramB === 'undefined' ) {
    return null;
  }
  return paramA + paramB;
}
```

Video by Web Dev Simplified

Spread syntax

"Udvide" et array eller et objekt til en liste af argumenter.

```
function sum(x, y, z) {
  return x + y + z;
}
const numbers = [1, 2, 3];
console.log(sum(...numbers));
// Expected output: 6
console.log(sum.apply(null, numbers));
// Expected output: 6
```

Kopiere objekt eller array med spread syntax

Et meget anvendt pattern er at kopiere et objekt eller et array med spread syntax:

```
// Objekt
const obj = { a: 1, b: 2, c: 3 };
const obj2 = { ...obj };
// Array
const arr = [1, 2, 3];
const arr2 = [...arr];
```

Immutable objekter og arrays

Når man tilstræber immutability er det vigtigt at kopiere objektet hver gang man modificerer det:

```
// Objekt
const obj = { a: 1, b: 2, c: 3 };
const obj2 = { ...obj, a: 4 };
```

Array Funktioner `Array.forEach()`

For each afvikler en stump kode for hvert element i et array.
Der returneres ikke noget fra forEach.

```
const list = [1, 2, 3, 4, 5];
list.forEach((item) => { console.log(item) }); // 1, 2, 3, 4, 5
```

Array Funktioner `Array.map()`

Map ligner forEach, men afviger ved at returnere et nyt array med de ændrede værdier.

```
const list = [1, 2, 3, 4, 5];
const newList = list.map((item) => { return item - 1 }); // [0, 1, 2, 3, 4]
```

Array Funktioner `Array.find()`

Find returnerer det første element i et array, der matcher et givent kriterie.

```
const list = [
  { id:1, name: 'first'},
  { id:2, name: 'second' }
];
const picked = list.find((item) => { return item.id === 2 });
console.log(picked); // { id:2, name: 'second' }
```

Array Funktioner `Array.filter()`

Filter returnerer et nyt array med alle elementer, der matcher et givent kriterie.

```
const list = [
  { id:1, name: 'first'},
  { id:2, name: 'second' },
  { id: 3, name: 'third' },
  { id: 4, name: 'fourth' },
  { id: 5, name: 'fifth' }
];
const picked = list.filter((item) => { item.id > 3 });
console.log(picked); // [{ id: 4, name: 'fourth' }, { id: 5, name: 'fifth' }]
```

Array Funktioner `Array.reduce()`

Reduce reducerer et array til en enkelt værdi.

```
const dates = [
  new Date('2020-03-04'),
  new Date('2018-04-03'),
  new Date('2022-01-01'),
  new Date('2020-05-06'),
  new Date('2021-02-01')
];
const latestDate = dates.reduce((latest, current) => {
  return current > latest ? current : latest;
});
```

Øvelse

Array funktioner

Edit on **CodeSandbox**

Classes

En klasse er et template for objekter. En klasse definerer et objekts egenskaber og metoder.

```
class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }
  greet() {
    return "Hello, " + this.greeting;
  }
}
let greeter = new Greeter("world");
```

Nedarvning

- En klasse kan arve egenskaber fra en anden klasse.
- En klasse kan kun arve fra en klasse ad gangen.

Nedarvning eksempel

```
class Animal {
  move() {
    console.log('Animal moved');
  }
}
class Dog extends Animal {
  bark() {
    console.log("Woof! Woof!");
  }
}
const dog = new Dog();
dog.bark();
dog.move(10);
dog.bark();
```

Public, private, and protected modifiers

- Public: kan tilgås fra alle steder (default)
- Private: kan kun tilgås indenfor klassen.
- Protected: kan kun tilgås indenfor klassen og dens subklasser.

Public, private, and protected modifiers

```
const fullNameMaxLength = 10;
class Employee {
  private _fullName: string = "";
  get fullName(): string {
    return this._fullName;
  }
  set fullName(newName: string) {
    if (newName && newName.length > fullNameMaxLength) {
      throw new Error("fullName has a max length of " + fullNameMaxLength);
    }
    this._fullName = newName;
  }
}
let employee = new Employee();
employee.fullName = "Bob Smith";
if (employee.fullName) {
```

Static properties

- Properties der er defineret som static, kan tilgås direkte fra klassen, ikke fra instanser af klassen.
- Bruges til generelle værdier der gælder for alle klasser.

Static properties

```
class Grid {
  static origin = { x: 0, y: 0 };

  calculateDistanceFromOrigin(point: { x: number; y: number }) {
    let xDist = point.x - Grid.origin.x;
    let yDist = point.y - Grid.origin.y;
    return Math.sqrt(xDist * xDist + yDist * yDist) / this.scale;
  }

  constructor(public scale: number) {}
}

let grid1 = new Grid(1.0); // 1x scale
let grid2 = new Grid(5.0); // 5x scale

console.log(grid1.calculateDistanceFromOrigin({ x: 10, y: 10 }));
console.log(grid2.calculateDistanceFromOrigin({ x: 10, y: 10 }));
```

Øvelse

Klasser og nedarvning

Edit on **CodeSandbox**

Del 2 - Typescript

Typescript - hurtigt igang

- Typescript Playground
- Code Sandbox

Basis typer

Everyday types

```
const name:string = "Jens" // string
const age:number = 42 // number
const married:boolean = true // boolean
```

Arrays

```
// Array der kun indeholder tal
const numbers:number[] = [1, 2, 3];
// Array der kun indeholder strings
const names:string[] = ["Jens", "Peter", "Hans"];
// Array der kan indeholde både tal og strings
const mixed:(number|string)[] = [1, "Jens", 2, "Peter", 3, "Hans"];
```

Tuple

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

Any

- Any er en type der kan indeholde alle typer.
- Man bør undgå at bruge any, da det fjerner mange af fordelene ved at bruge Typescript.

```
let obj: any = { x: 0 };
// None of the following lines of code will throw compiler errors.
// Using any disables all further type checking, and it is assumed
// you know the environment better than TypeScript.
obj.foo();
obj();
obj.bar = 100;
obj = "hello";
const n: number = obj;
```

Functions

```
// Parameter type annotation
function greet(name: string) {
  console.log("Hello, " + name.toUpperCase() + "!!");
}
// Return type annotation
function getFavoriteNumber(): number {
  return 26;
}
```

Object typer

```
// The parameter's type annotation is an object type
function printCoord(pt: { x: number; y: number }) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
}
printCoord({ x: 3, y: 7 });
```

Optional arguments

- Man kan markere argumenter som valgfrie ved at tilføje et ? efter argumentet.
- Det er ofte en god ide at bruge default argumenter sammen med optional arguments.

```
function printName(first: string; last?: string) {
  // ...
}
// With default argument
function printName(first: string; last?: string = "Hansen") {
  // ...
}
```

Type Alias

- En type med et navn
- Kan bruges til at gøre kode mere læselig
- Genbrug af type definitioner

Type Alias

```
type Point = {
  x: number;
  y: number;
};
// Exactly the same as the earlier example
function printCoord(pt: Point) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
}
printCoord({ x: 100, y: 100 });
```

Interface

- Alternativ til type alias
- Kan bruges til at beskrive objekter
- Interfaces kan bruge nedarvning

```
interface Point {
  x: number;
  y: number;
}
function printCoord(pt: Point) {
  console.log("The coordinate's x value is " + pt.x);
  console.log("The coordinate's y value is " + pt.y);
```

Type intersection

Man kan kombinere typer med type intersections:

```
type typeAB = typeA & typeB;
```

Union type

En union type er en type der kan indeholde flere typer.

```
function printId(id: number | string) {
  console.log("Your ID is: " + id);
}
```

Enum type

Brug unions i stedet for Enums: Stackoverflow

```
enum Status {
  Idle = "idle",
  Pending = "pending",
  Resolved = "resolved",
  Rejected = "rejected"
};
```

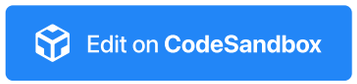
Readonly

- Man kan erklære en property i en type som readonly
- Fungerer både i interfaces og type aliases
- Readonly gør for properties hvad `const` gør for variabler.

```
interface Foo {
    readonly bar: number;
    readonly bas: number;
}
type Foo = {
    readonly bar: number;
    readonly bas: number;
}
```

Øvelse

Type aliases of interfaces



Types vs. Interfaces

Bør man bruge types eller interfaces?

typescriptlang.org Matt Pocock

Typescript Generics

- Generics er en måde at skrive kode der kan bruge flere typer
- Generics er en slags type funktion der kan tage en type som parameter

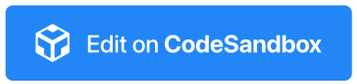
```
type MyGenericType = <T>(arg: T) => T;
```

Typescript Generics

10 Tips for Mastering TypeScript Generics

Øvelse

Typescript Generics



Modulær Typescript

Typescript lokalt projekt

For at starte at basis Typescript projekt lokalt, kan du køre:

```
$ npm install -g typescript
$ mkdir typescript-test-project && cd $_
$ yarn add typescript --dev
$ yarn tsc --init
$ tsc [your-file.ts]
```

tsconfig.json (1/5)

Du kan køre `tsc` med "flag" for eksempel:

```
$ tsc app.ts util.ts --target esnext --outfile index.js
```

Hvilket "bygger" filerne `app.ts` og `util.ts` og sætter dem sammen til en enkelt fil, `index.js`

tsconfig.json (2/5)

Det er ikke praktisk at holde styr på alle compiler options med flag.

```
$ tsc --init app.ts util.ts --target esnext --outfile index.js
```

Med `--init` flaget kan du automatisk oprette en konfigurationsfil, tsconfig.json, hvor de flag du har sat bliver gemt.

tsconfig.json (3/5)

Hvis du kører tsc uden options vil typescript compileren kigger efter en tsconfig.json fil og anvende de options der er angivet.

Hvis du checker tsconfig.json ind i Git og sørger for at alle fremtidige ændringer opdateres i filen, er du sikker på at dine kolleger compiler med samme options som dig.

tsconfig.json (4/5)

- `tsconfig.json` kan være intimiderende
- Dokumentation: typescriptlang.org/tsconfig
- Vigtige options:
  - files
  - include
  - exclude
  - outDir / outFile

tsconfig.json (5/5)

- Vigtige options (fortsat):
  - strict
  - target
  - watchMode

Linting

- Eslint
- Tslint
- Prettier

Poetic

Poetic er en NPM pakke der gør det nemt at installere og konfigurere linters og prettier med forudbestemte basisindstillinger:

https://www.npmjs.com/package/poetic

```
$ npx poetic
```

TS Node

Med `ts-node` kan vi afvikle typescipt filer i et servermiljø eller vi kan starte en REPL for at eksperimentere og afprøve ting.

Afvikle en fil:

```
$ npx ts-node file.ts
```

Starte en REPL:

```
$ npx ts-node
```

Runtime validering

Library:

Jack Herrington Zod

Array metoder og objekter

Man kan ikke bruge Array metoder på objekter, men man kan konvertere Arrays til Objekt med `Object.entries`.

```
const timesheetByName = {
  michael: { hours: "237"},
  john: { hours: "252"},
  jane: { hours: "212"},
  michelle: { hours: "141"}
};
const timesheetEntries = Object.entries(timesheetByName);
const totalHours = timesheetEntries.reduce((sum, [key, value]) => {
  return sum + Number(value.hours);
}, 0);
```

Optional chaining

Optional chaining giver mulighed for at undgå at få en fejl, hvis et objekt eller et array er undefined. Eksempel:

```
const adventurer = {
  name: 'Alice',
  cat: {
    name: 'Dinah'
  }
};
const dogName = adventurer.dog?.name;
console.log(dogName);
// Expected output: undefined
console.log(adventurer.someNonExistentMethod?.());
// Expected output: undefined
```

Rest parameters

Giver adgang til at tage et ukendt antal parametre som et array.

```
function sum(...theArgs) {
  let total = 0;
  for (const arg of theArgs) {
    total += arg;
  }
  return total;
}
console.log(sum(1, 2, 3));
// Expected output: 6
console.log(sum(1, 2, 3, 4));
// Expected output: 10
```

Promises

Promises er en måde at håndtere asynkronitet på. En promise kan være i tre forskellige tilstande:

- Pending
- Fulfilled
- Rejected

Promises

Opret et Promise:

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("foo");
  }, 300);
});
```

Promises

```
const handleFulfilled = (value) => {
  console.log(value);
};
const handleRejected = (error) => {
  console.log(error);
};
myPromise.then(handleFulfilled, handleRejected)
```

Async / Await

```
const run = async function() {
  const result = await myPromise();
  console.log(result);
}
run();
```

Project

Byg et SDK for en biograf