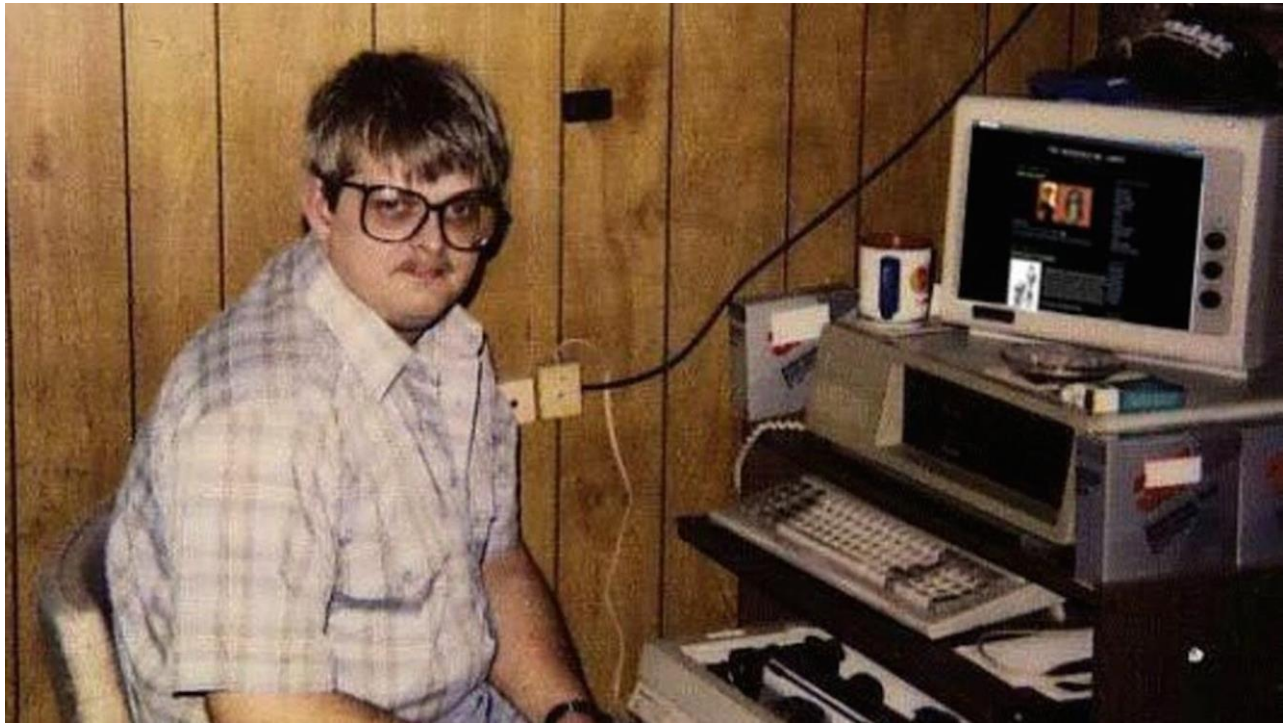
Javasta JavaScriptiin

Pikasukellus JavaScript-kielen koukeroihin TypeScriptillä

Aino Haikala 2024

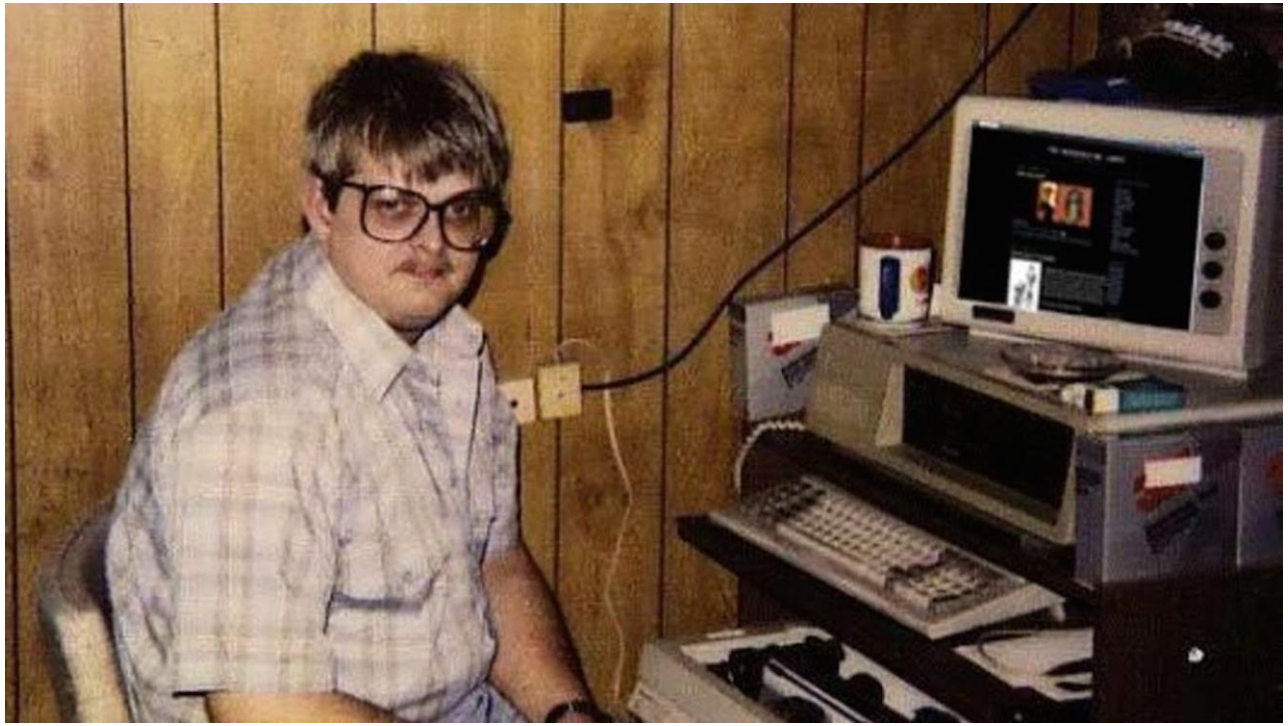
- Asiaa kahlataan läpi paljon.
- Tavoite tälle sessiolle ei ole, että poistutte täysinoppineina.
- Hyvin mahdollisesti olette aika hämmentyneitä ja sekaisin.
- **Tavoite on, että olette joskus kuullut näistä asioista ja osaatte kaivaa tarvittaessa aiheesta tietoa.**
- Joukossa on sellaista tietoa, joita ette välttämättä tarvitse vielä muutamaan vuoteen, jos silloinkaan.
- Ps. Esimerkeissä on käytetty magic stringejä. Älä tee niin IRL.

Java vs. JavaScript

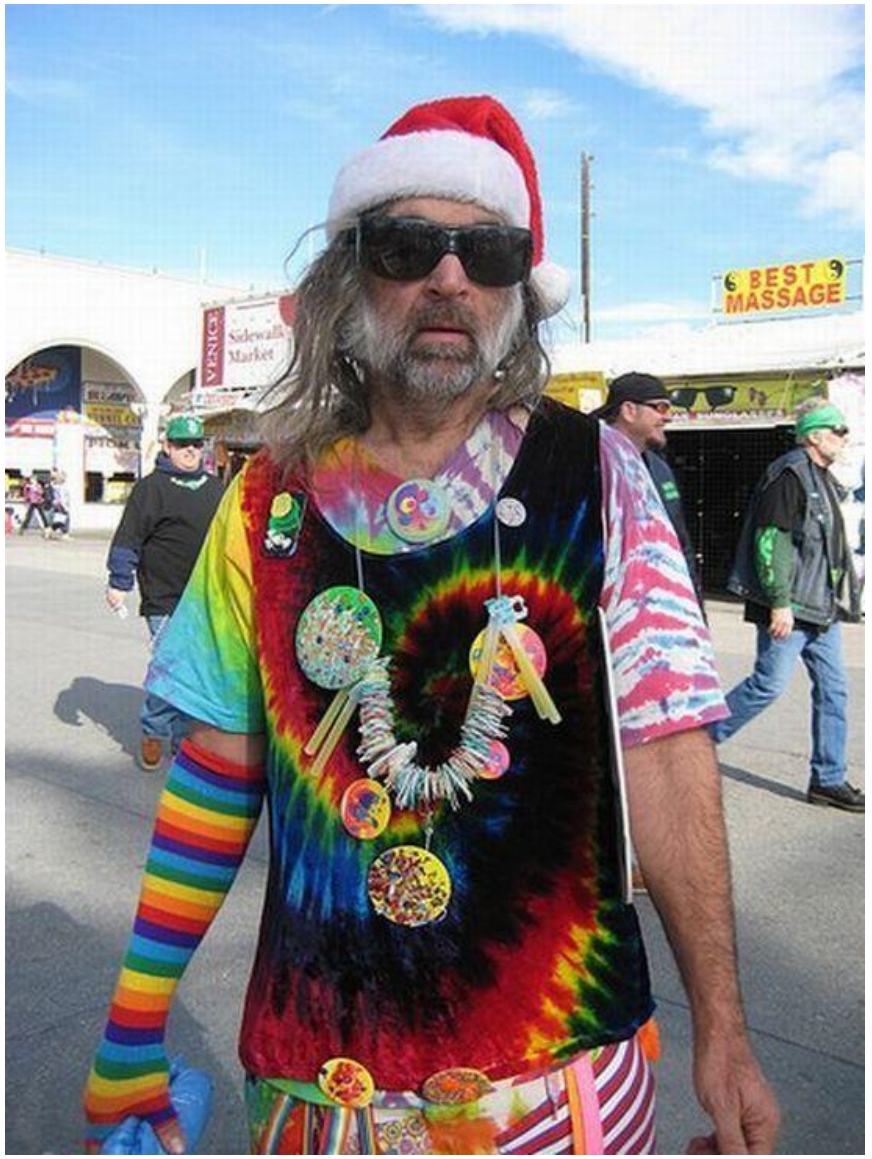


https://www.industryconnect.org/wp-content/uploads/2016/04/Computer-Nerd.jpg

Java vs. JavaScript



https://www.industryconnect.org/wp-content/uploads/2016/04/Computer-Nerd.jpg



https://i.pinimg.com/564x/00/1b/2f/001b2f96812d01aff68f40ccceff5bd3.jpg

Typescript

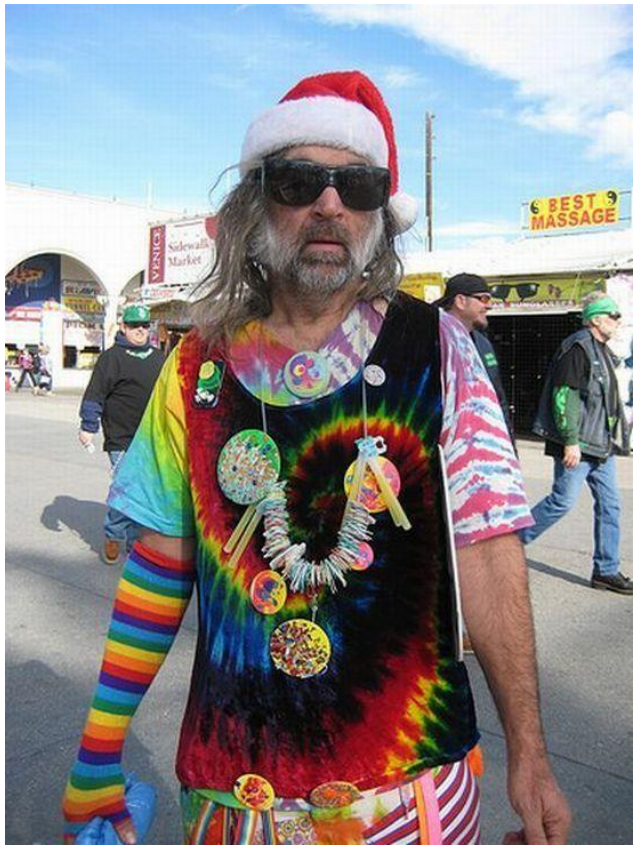
https://wellgroomedgentleman.com/wp-content/uploads/2023/10/10._The_purest_hipster_style_mustache_of_a_Vic.width-800.jpg

Ne ajat, jolloin web-frontti oli kasa huonosti skriptattua spagettia ovat (onneksi) olleet jo kauan sitten ohi.
Koodimäärän kasvaessa myös käytäntöjä on ollut pakko parantaa.

Typescript

https://wellgroomedgentleman.com/wp-content/uploads/2023/10/10._The_purest_hipster_style_mustache_of_a_Vic.width-800.jpg

Paitsi, että… typescript ei ole kieli vaan JS laajennos. Pohjalla kaikki on edelleen JavaScriptiä.



1. Ajon aikana ei ole typescriptiä, vain JS

- Mitä tämä palauttaa?

```
function sumNumbers (a: number) {
  return a + a;
}
```

1. Ajon aikana ei ole typescriptiä, vain JS

- Mitä tämä palauttaa?

```
function sumNumbers (a: number) {
  return a + a;
}
```

- Vastaus: Riippuu mitä syötät sisään. sumNumbers(2) => 4, sumNumbers(”2”) => 22.

2. JavaScriptissä ei ole luokkia

JavaScript may be a bit confusing for developers coming from Java or C++, as it's all dynamic, all runtime, and it has no static types at all.

Everything is either an object (instance) or a function (constructor), and even functions themselves are instances of the Function constructor.

Even the "classes" as syntax constructs are just constructor functions at runtime.

”MDN”

JavaScriptissä on käytössä prototypaalinen periytyminen

```
const cat = {
  scratch: function() { console.log("scratch scratch")},
  meow: function() { console.log("meow, meow")},
    __proto__: mammal
}
```

```
const mammal = {
  live: function() {console.log("do mammal things")},
  __proto__: animal
}
```

```
const animal = {
  live: function() {console.log("is living");}
}
```

- Ajatelkaa linkitettyä listaa, jossa on objekteja linkitettynä toisiinsa.
- Tämä on tietoa, jota ette välttämättä usein tarvitse, mutta se on hyvä tietää.
- (Tällä dialla on hieman yksinkertaistettu selkeyden vuoksi)

Miten saa koirat maukumaan?

Miten saa koirat maukumaan?

```
let cats = ["mirri", "miisu"];
let arrayPrototype = cats.__proto__;

arrayPrototype.meow = function() {console.log("MEOW MEOW")}

cats.meow();
cats.forEach(i => console.log(i));

let dogs = ["musti", "murre"];
dogs.meow();

let numbers = [1, 2, 3];
numbers.meow();
```

- **Jos stackoverflow ehdottaa koskemaan prototyyppiin, älä tee sitä. Ongelmasi voi ratkaista fiksumminkin.**

Typescriptin tekijöillä on ollut paljon mielipiteitä sallituista syntakseista. Et pysty tekemään laillisesti TS:llä kaikkea mitä JS:llä.

Typescript on ystävä.

=> **Kun typescript valittaa, kuuntele**.
Vältä any:n ja ts-ignoren käyttöä.

3. Skriptikielet usein ankkatyypittävät

- Duck typing. Jos se vaakkuu kuin ankka ja vaappuu kuin ankka, se on ankka.
- Myös Typescriptin tapa tyypittää eroaa Javan vastaavasta.

```ts
type Mouse = {
  name: string,
  color: "ruskea" | "harmaa"
}

const logMouse = (mouse: Mouse) => {
  console.log(`${mouse.name} on ${mouse.color}`)
}

const mikki: Mouse = {
  name: "Mikki",
  color: "ruskea"
} as Mouse

logMouse(mikki);
```

- Castaus on tässä turha TypeScript pitää jo muuttujaa hiirenä.
- Tämä on vs kohtele tätä kuin.

```ts
const mirri = {
  name: "Mirri",
  purr: () => console.log("PURRR, PURRR")
} as any as Mouse

logMouse(mirri);
```

- Ohitatte tässä castayksellä TypeScriptin tuoman suojan.

Älkää castatko, ellei ole ihan oikeasti tarvetta.

Miten toteuttaisit seuraavan rajapinnan?

```
export interface TableItem {  1 inheritor
    getKeyField(): string

    getField(key: string): string

}
```

Tässä muutama mahdollisuus

```
class ItemImpl1 implements TableItem {

  constructor() {}

  getKeyField() {
    return "foo";
  }
  getField(key: string): string {
    return key;
  }
}


const item = new ItemImpl1();
item.getKeyField();
```

```
const item3: TableItem = {
  getKeyField: () => "foo",
  getField: (key: string) => key,
}

item3.getKeyField();
```

Rakas java-koodari,

Kaiken ei tarvitse olla luokkia
… tai luokissa. On ihan ok laittaa
moduulin juuritasolle vaikkapa
funktio tai useampi.

(Juuritasolla voi siis olla myös vaikka useampi luokka samassa tiedostossa, jos siltä tuntuu)

4. Truthyt ja falsyt

- !!a && show(a) on reactissa yleinen tapa näyttää a, jos a on annettu.

- Mitä tämä tulostaa?

```
const isDefined = (value: any) => {

  return !!value;
}

const logAllDefinedValues = (myObj: {[key: string]: any}): void => {

  for(const [key, value] of Object.entries(myObj)) {
    if (isDefined(value)) {
      console.log(key);
    }
  }
}

const definedOrNot = {
  a: "foo",
  b: undefined,
  c: null,
  d: true,
  e: false,
  f: 10,
  g: 0
}

logAllDefinedValues(definedOrNot);
```

Truthyt ja falsyt

- **Varo erityisesti nollia!**

Asiakas: ”Jostain syystä kaikki luvut ei näy taulukossa”.

```
const isDefined = (value: any) => {

  return !!value;
}

const logAllDefinedValues = (myObj: {[key: string]: any}): void => {

  for(const [key, value] of Object.entries(myObj)) {
    if (isDefined(value)) {
      console.log(key);
    }
  }
}

const definedOrNot = {
  a: "foo",
  b: undefined,
  c: null,
  d: true,
  e: false,
  f: 10,
  g: 0
}

logAllDefinedValues(definedOrNot);
```

- Bonus:

Mitä eroa on

0 == false ja 0 === false

(vinkki, ekassa tehdään tyyppikonversio)

Käyttäkää aina === (, jos teillä ei ole jotain erikoista syytä käyttää ==).

5. Function scope ja block scope

- Mitä tämä tulostaa? Entä jos olisi käytetty let tai const?

```
var a = 32;

function secretOfLife() {

  if (true) {
    var a = 42;
  }

  console.log(`Secret of life is: ${a}`);
}

secretOfLife();
```

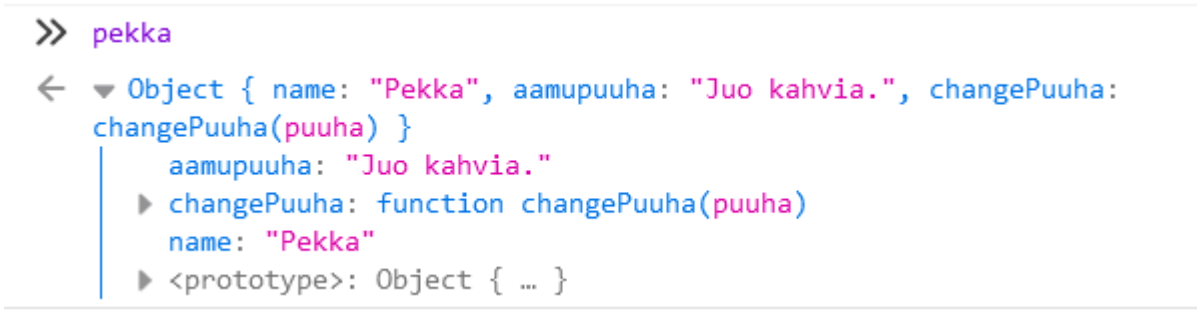
- JavaScriptissä muuttujien näkyvyys eroaa Javasta.
  - Avainsanoja: Function scope, block scope, global scope.
- Moduulien kanssa on huoletonta, koska muuttujista on hankala tehdä vahingossa globaaleja.

- **Älä laita var. Laita const, aina jos pystyt. Jos et pysty, laita let.**
- **Kaikkea ei tarvitse tästäkään aiheesta tietää, mutta tunne se, mitä käytät.**

6. This – uhh huhh…

- Mitä tämä tulostaa?

```
const pekka = {
  name: "Pekka",
  aamupuuha: "Juo kahvia." ,
  changePuuha: function(puuha: string) {
      this.aamupuuha = puuha;}
}

const changePuuhaCallBack = pekka.changePuuha
changePuuhaCallBack("Käy suihkussa");
console.log(pekka.aamupuuha)
```



Vastaus: Pekka juo edelleen kahvia. This viittaa kutsussa window-objektiin, ei pekkaan.

”One of the trickiest part of Javascript language is this mechanism. It is used to **refer to the object that is currently executing the code**. ”

Korjata voi useammalla tavalla

```
const changePuuhaCallBack2 = pekka.changePuuha.bind(pekka);
changePuuhaCallBack2("Lähde lenkille");
console.log(pekka.aamupuuha) //Lähde lenkille.
```

For the sake of simplicity tosiaan…

For the sake of simplicity, there is a list of questions that you should answer while evaluating the value of `this`.
The order of the questions is really important.

- is the function called with new?
  Then `this` refers to a constructor object.
- is the function called with call, apply or bind?
  Then `this` refers to the context passed as the argument call, apply or bind methods
- is the function called as a method, ie: obj.method()?
  Then `this` refers to the object "before the dot" in the method call
- is the function called in the global scope?
  Then `this` refers to the global object, which is `window` in browsers.

(https://dev.to/nikolasbarwicki/is-this-keyword-a-problem-1ind)

Miten lähestyä avainsanaa *this*

- Hyvää matskua aiheesta löytyy paljon, esim: https://www.freecodecamp.org/news/the-this-keyword-in-javascript/
- Yleensä ongelmia tulee, kun käyttää callback:ejä tai eventtien kanssa.
- Tällöin ongelman debuggaaminen on usein hankalampaa kuin korjaaminen.
  - Jos joku on mystisesti undefined, tarkista voiko kyseessä olla this.
  - Bind, apply, call, () => {}

- **Opettele peruskäyttö. Käytä siellä, missä tiedät miten se toimii.** (…tai yksi vaihtoehto on myös olla käyttämättä this:iä).

7. Event loop

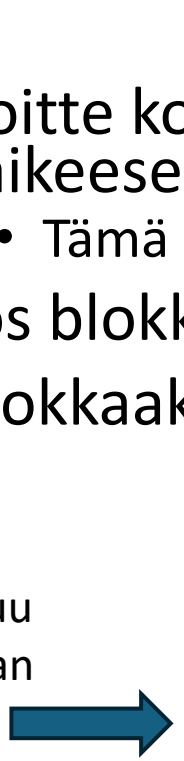
- Voitte koodatessa ajatella, että selaimessa on käytössä vain yksi säie. Siihen säikeeseen poimitaan tehtäväjonosta uusia tehtäviä suoritettavaksi.
  - Tämä on yksinkertaistus monellakin tavalla.
- Jos blokkaatte säikeen, blokkaatte käyttäjän toiminnan.
- Blokkaako tämä käyttäjän toiminnan?

```
async function sumNumbers (to: number): Promise<number> {

  let sum = 0;

  for (let i = 1; i <= to; to++) {

      sum = sum + i;
  }

  return sum;
}

await sumNumbers(100000);
```

7. Event loop

- Voitte koodatessa ajatella, että selaimessa on käytössä vain yksi säie. Siihen säikeeseen poimitaan tehtäväjonosta uusia tehtäviä suoritettavaksi.
  - Tämä on yksinkertaistus monellakin tavalla.
- Jos blokkaatte säikeen, blokkaatte käyttäjän toiminnan.
- Blokkaako tämä käyttäjän toiminnan? Vastaus: kyllä.

Säie jumittuu suorittamaan tätä pitkää looppia

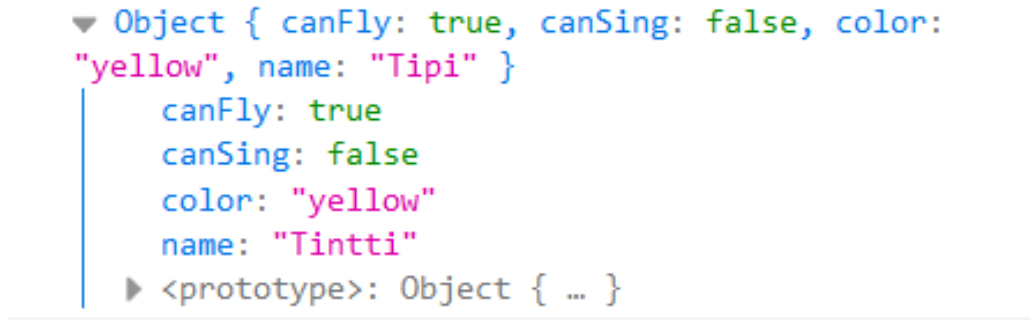


```
async function sumNumbers (to: number): Promise<number> {

  let sum = 0;

  for (let i = 1; i <= to; to++) {

      sum = sum + i;
  }

  return sum;
}

await sumNumbers(100000);
```

8. Console.log

- Mitä tämä tulostaa?

```
const bird = {
  canFly: true,
  canSing: false,
  color: "yellow",
  name: "Tipi"};
console.log(bird);
bird.name = "Tintti";
```



- Muistakaa tämä, kun ihmettelette, mitä koodissanne on vikaa, kun tulostuu väärä arvo.

Miten selviän?

- Tässä on tarkoituksella kerrottu kamalista asioista. No worries.

- Ensisijaisesti: Käytä yrityksesi ohjeita.
- Toissijaisesti: Ainon ohjeet, eli aloita hallitsemalla nämä:
  - Const ja let, === , truthyjen ja falsyjen osaaminen, objektit.
  - Arrow functiot () => {}
  - Yksi hyvä oikea funktiosyntaksi (arrow funktiolla ei voi tehdä kaikkea)
  - Luokat (https://www.typescriptlang.org/docs/handbook/2/classes.html)
  - Kivat jutut, kuten Typescriptin tyypityksen monipuolisuus, ??, erityisesti listojen käsittelyt (map, filter, for.. of yms.) monipuolisemmat javaan verrattuna.
- Osaa se, mitä käytät, lopun voit opetella joskus myöhemmin. Koodaa tiukkaa TypeScriptiä.

Miksi TypeScript on kivaa?

- Dokumentaatiot on yleensä huomattavasti parempia kuin Javan kanssa touhutessa.
- Hyvällä tavalla kevyttä ja vapaata.
- Mahdollistaa funktionaalisemman tyylin kokeilua tutunoloisessa ympäristössä.
- Sitä paitsi, tee tää Javalla

```ts
type DogType = {
  name: string,
  bark: () => string,
  isFurry: boolean,
  color: string
}

const myFunctionWithDefault = (name: string, bark: () => string, isFurry = true, color?: string): DogType => {

  return {
    name: name,
    isFurry: isFurry,
    bark: bark,
    color: color ?? "ruskea"

  }
}

const musti: DogType = myFunctionWithDefault("musti", () => "vuh vuh");
console.log(musti)

const haukku = () => "hau hau";
const peni: DogType = myFunctionWithDefault("peni", haukku, false, "punertava");
console.log(peni)

const myDogs: DogType[] = [musti, peni];
const dogNames = myDogs.map(dog => dog.name);
console.log(dogNames)
```

Tutustumisen arvoista:

- **MDN**
- https://www.typescriptlang.org/docs/handbook/intro.html
- https://github.com/labs42io/clean-code-typescript
- https://sbcode.net/typescript/
- https://refactoring.guru/design-patterns/typescript
- What the heck is event loop