# JavaScript ohjelmointi

## *JS kirjoitussäännöt & ketjutus*

MARGIT TENNOSAARIN MATERIAALISTA
LAURA JÄRVISEN MUOKKAAMA

# ‘use strict’;

- käytä tiedoston tai funktion alussa
- aktivoi JavaScriptissä tiukan tilan (strict mode), joka auttaa välttämään virheitä tekemällä kielestä tarkemman ja estämällä esimerkiksi julistamattomien muuttujien käytön.

# Nimeämiskäytännöt

- Vaikka nimeäminen ei vaikuta itse koodin toimintaan, johdonmukainen nimeäminen tekee koodista helpommin ymmärrettävää
- **Muuttujat ja funktiot:** Käytä camelCase-kirjoitustapaa (esim. userName, getUserInfo).
- **Luokat:** Käytä PascalCase-kirjoitustapaa (esim. Car, UserProfile).
- **Vakiot:** Käytä UPPER_CASE-kirjoitustapaa (esim. MAX_VALUE).

# Hyvät funktiot

- Yksi funktio - Yksi tehtävä
- Hillitse paramterien määrää - jos niitä on monta, voisiko ne olla olio?
- Nuolifunktiot silloin, kun ne ovat selkeitä käyttää

# Kommentointi

- Koodi puhuu puolestaan - älä kommentoi kaikkea
- Mutta lisää kommentteja! Kommenttien avulla voit jäsennellä koodia eri osioihin ja selventää monimutkaisempia tapahtumia. Hyvät kommentit ovat arvokas lahja

# ESLint ja muut lintterit

- Lintterit analysoivat lähdekoodia, tunnistavat virheitä ja tyyliongelmia sekä auttavat pitämään koodin yhdenmukaisena ja laadukkaana.
- Lintteri (ESLint) tarkistaa koodista virheitä ja sääntöjen rikkomuksia, kun taas formatointityökalu (Prettier) ainoastaan muotoilee koodin automaattisesti yhtenäiseksi.
- Prettier ja ESLint oletusasetukset ovat usein ristiriidassa. Jos näitä ei ratkaise asetuksia muuttamalla, koodi on jatkuvasti “rikki” vaikka se on kunnossa.

# Vältä taikalukuja ja -merkkijonoja

- Koodiin ei pidä kirjoittaa suoraan epämääräisiä numero- tai merkkijonoarvoja, vaan ne pitäisi määritellä vakioina, jotta koodi on selkeämpää ja helpommin ylläpidettävää.

```
// Huono käytäntö
let discount = total * 0.2; // Mitä 0.2 tarkoittaa?

// Hyvä käytäntö
const DISCOUNT_RATE = 0.2;
let discount  = total * DISCOUNT_RATE;
```

# Refaktoroi

- Refaktorointi tarkoittaa ohjelmiston lähdekoodin parantamista ja selkeyttämistä muuttamatta sen ulkoista toimintaa.
- Refaktoroinnin tavoitteena on tehdä koodista helpommin ylläpidettävää, luettavaa ja tehokkaampaa.

*Ketjutus*

# Mikä ketjutus?

Tämä ei ole kokonaan uusi asia, vaan ketjustusta ollaan käytetty jo aiemmin.

Ketjutus mahdollistaa useiden metodien yhdistämisen pisteen avulla yhdeksi selkeäksi ja helposti luettavaksi lauseeksi, mikä tekee koodista tehokkaampaa ja tiiviimpää.

Kirjoita koodia, jonka itse ymmärrät. Eli ei tarvitse ketjuttaa vain ketjuttamisen takia, jos se muuttaa koodin liian monimutkaiseksi.

# Ketjutusesimerkki (.)

- Jos ketju mahtuu yhdelle riville, se pidetään samalla rivillä
- Jos ketju ei mahdu yhdelle riville, ketjun eri vaiheet rivitetään omille riveilleen alkamaan pisteellä sisennyksen jälkeen

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

const evenSquares = numbers
  .filter(n => n % 2 === 0)    // suodatetaan parilliset
  .map(n => n * n)             // korotetaan neliöön
  .join(', ');                 // yhdistetään merkkijonoksi

console.log(evenSquares);      // "4, 16"
```

# Valinnainen ketjutus (?.)

- Valinnainen ketjutus (?.) mahdollistaa syvälle sisäkkäisten ominaisuuksien, taulukkojen alkioiden tai funktiokutsujen turvallisen käyttämisen: jos jokin välivaihe on null tai undefined, koko lauseke palauttaa undefined sen sijaan, että *heittäisi virheen**.
- Tämä on erittäin hyödyllistä silloin, kun tiedon saatavuudesta ei voida olla varmoja. Eli jos tietoa pyydetään esimerkiksi toiselta palvelulta rajapinnan kautta.

# *Virheenkäsittelyä ensi viikolla

- Valinnaisessa ketjutuksessa ja JS kirjoittamissääntöjen ohjeissa on mainittu *virheen heittäminen.* Tätä käsitellään ensi viikolla

# Harjoitellaan

Löydät tehtäviä JavaScriptin kirjoittamisen ja ketjuttamisen harjoittelemiseen viikon 8 kansiosta

https://github.com/bc-web-ohjelmistokehitys/WP25K-JS

Huom! Opettele itse - älä pyydä tekoälyä ratkaisemaan tehtävää puolestasi.