Ennakkotehtävät

Tehtävä 1

Yhdistäisin luokat Cat ja Dog yhdeksi luokaksi, jonka nimi voisi olla vaikka Animals. Saman tekisin funktioille PrintCats ja PrintDogs. Uuden funktion nimeksi tulisi PrintAnimals. Nykyisellään nämä luokat ja funktiot ovat lähes identtisiä, joten mielestäni ne voisi ihan hyvin yhdistää yllä mainitulla tavalla.

Tämän ratkaisun ansiosta koodi muuttuu selkeämmäksi lukea ja ylläpito helpottuu. Jos ohjelman toimintaa halutaan muuttaa (esim. lisätä uusia ominaisuuksia eläimille), niin riittää, että muutokset tehdään yhteen luokkaan. Vanhassa mallissa samat asiat pitäisi kirjoittaa kahteen kertaan sekä kissoille että koirille. Tai vielä useampia kertoja, mikäli haluttaisiin lisätä muita eläinlajeja. Koodissa rivit lisääntyisivät ja mikä pahinta, ne olisivat käytännössä samaa koodia.

Lisäksi tuossa alkuperäisessä animals.ts koodissa kissoilla ja koirilla on eri ominaisuudet, koska koirilla on väri, mutta kissoilla ei. Tämäntyyppiset jutut hankaloittavat koodin lukemista, sillä lukija ei voi olla täysin varma siitä, että onko kyseessä virhe vai tarkoituksella tehty juttu. Joka tapauksessa yksi Animals-luokka takaisi sen, että kaikilla eläimillä olisi varmasti samat ominaisuudet.

Alla esimerkki toteutuksesta:

```
export class Animal {
    constructor(_name: string, _age: number, _color: string) {
        this.name = _name;
        this.age = _age;
        this.color = _color;
    }

    get(): object {
        return {
            "name": this.name,
            "age": this.age,
            "color": this.color
        };
    }
}

export function PrintAnimals(animals: Animal[], addheader: boolean): string {
    var html: string;
    var data: object;
    html = "";

    if (addHeader) {
        data = animals[0].get();
        var keys = Object.keys(data);

        html += "<div class='row header'>";

        for (var key in keys) {
            html += "<div class='col-1'>"+ keys[key] +"</div>";
        }

                    html += "</div>";
            }

    for (var i in animals) {
        data = animals[i].get();

        var values = Object.values(data);
        html += "<div class='row'>";

        for (var val in values) {
            html += "<div class='col-1'>" + values[val] + "</div>";
        }

        html += "</div>";
    }

    return html;
}
```

Tehtävä 2

Päivämääräväliä halutaan muokata

Tässä voisi antaa käyttäjälle mahdollisuuden valita päivämääräväli esimerkiksi HTML:n input typen avulla tähän tyyliin:

```
<input type="date" id="startDate">
<input type="date" id="endDate">
```

Asiakkaan koodin puolella nämä valinnat tallennettaisiin muuttujiin:

```
const startDate = document.getElementById("startDate").value;
const endDate = document.getElementById("endDate").value;
```

Kun HTML:n input type daten päivämäärät napataan näin, ne ovat muodossa yyyy-mm-dd, joten ne käyvät sellaisenaan Nuuka Open API:lle eikä niitä tarvitse pyöritellä mitenkään. Tämän jälkeen niille tehtäisiin tarvittavat oikeellisuustarkistukset joko asiakkaan tai palvelimen puolella. Tehdään nyt tässä esimerkissä asiakkaan puolella, koska mielestäni ei ole järkevää lähettää pyyntöä ollenkaan, jos päivämäärät vaikkapa rikkovat URLin tai tuottavat tulokseksi tyhjän JSON-taulukon. Muun muassa tällaisia juttuja voitaisiin tarkistaa:

- Päivämäärät eivät ole samoja.
- Alkupäivämäärä on ennen loppupäivämäärää.
- Päivämäärät eivät ole tulevaisuudessa.
- Päivämäärät ovat yhteensopivia API:n kanssa. (Esim. Nuuka Open API:n mukaan: "The most precise hourly data is generally available from 2015 onwards." Eli tässä tapauksessa ennen 2015 olevia päivämääriä ei sallittaisi.)

Kun päivämäärien oikeellisuus on varmistettu, ne voidaan lisätä yhteen merkkijonoon ja erottaa vaikka pilkulla, jonka avulla ne voidaan parsia palvelimella taas erillisiksi päivämääriksi:

```
const userInput = startDate + "," + endDate;
```

Ja sitten merkkijono userInput laitetaan menemään palvelimelle fetchin mukana:

```
const url = "http://127.0.0.1:5000/";
const options = {
    method: "POST",
    body: userInput
};
fetch(url, options)
```

Palvelimella käyttäjän syöttämistä tiedoista otetaan koppi tällä tavalla:

```
@electricityApp.route("/", methods = ["POST", "GET"])
def main():
    if (request.method == "POST"):
        req = request.data.decode("utf8") #User input stored in req
```

Tämän jälkeen käyttäjän syöte parsitaan taulukkoon, jonka jälkeen aloituspäivämäärä löytyy user_inputs[0] ja lopetuspäivämäärä user_inputs[1].

```
def parse_input(user_input):
    user_inputs = user_input.split(",")
    return user_inputs

user_inputs = parse_input(req)
```

Päivämäärät liitetään osaksi URL-osoitetta tähän tyyliin:

```
api_url = ("https://helsinki-openapi.nuuka.cloud/api/v1.0/EnergyData/Daily/"
+ "ListByProperty?Record=LocationName&SearchString=1000%20Hakaniemen%20kauppahalli"
+ "&ReportingGroup=Electricity&StartTime=" + start_date + "&EndTime=" + end_date)
```

Kulutusdata haluttaisiinkin viikoittaisena

Tekisin palvelimelle funktion, joka järjestää datan halutulla tavalla. Käyttäjä voisi sitten valita käyttöliittymässä, että haluaako datan kuukausittaisena tai viikottaisena. Tieto valinnasta lähtisi fetchin mukana palvelimelle samaan tyyliin kuin käyttäjän syöttämät päivämäärät. Valinta käsiteltäisiin palvelimella ja sen mukaan API:n data vietäisiin joko arrange_monthly -funktiolle tai arrange_weekly -funktiolle. Seuraavalla sivulla on esimerkki yhdestä tavasta toteuttaa datan järjestely viikottaiseen muotoon.

Tämä funktio olettaa, että viikko alkaa maanantaina ja loppuu sunnuntaina. Silmukka toimii niin, että jos kohdalla oleva päivä ei ole maanantai, kulutusdataa summataan muuttujaan consumption. Kun maanantai osuu kohdalle, summa pistetään taulukkoon talteen, consumption-muuttujan uudeksi arvoksi tulee maanantain kulutusdata ja homma alkaa taas alusta. Silmukka laskee myös päivät, eli saamme tietää onko viikko täysi vai ei (tämä koskee lähinnä päivämäärävälin alku- ja loppupäätä) ja myös kunkin viikon alkamispäivän se tallentaa toivon mukaan selkeämmän raportoinnin aikaansaamiseksi. Funktio ottaa huomioon myös joitain erikoistapauksia, kuten sen, että aikavälin ensimmäinen päivä on maanantai. Ilman tätä tarkistusta funktio lisäisi tässä tapauksessa yhden turhan rivin tulostaulukkoon.

```
def arrange_weekly(data):
    """Arranges the given daily electricity consumption data to weekly format.
       Sums the consumption of each week and adds it to a results table.
       Week starts on monday and ends on sunday.
       The results table comprises the date of the 1st day of week,
       the number of days in the week, combined consumption for each week, and unit.

    Args:
        data (json): The given JSON data.

    Returns:
       list: Arranged data.
    """
    first_date = parser.parse(data[0]["timestamp"])
    first_day = first_date.weekday() # Monday = 0
    unit = data[0]["unit"]
    last_index = len(data) - 1
    consumption = 0
    count_days = 0
    if (first_day == 0): count_days = 1
    results = []
    results.append("1st day of week,Number of days in week,Consumption,Unit")

    for index, item in enumerate(data):
        date = parser.parse(item["timestamp"])
        day = date.weekday()
        value = item["value"]

        if (day != 0):
            consumption = consumption + value
            count_days = count_days + 1

        if ( (day == 0) and (count_days > 1) ):
            result = (format_isodate(first_date) + "," + str(count_days)
            + "," + str(round(consumption, 2)) + "," + unit)
            results.append(result)
            consumption = value
            count_days = 1
            first_date = date

        if (index == last_index):
            result = (format_isodate(date) + "," + str(count_days)
            + "," + str(round(consumption, 2)) + "," + unit)
            results.append(result)

    return results
```

Kulutustietoja tulisi saada haettua myös toisesta palvelusta

Tämänkin hoitaisin niin, että käyttäjä voisi valita palvelun esimerkiksi käyttöliittymässä olevan alasvetovalikon avulla. HTML:n avulla sen voisi toteuttaa jotenkin näin:

```
<label for="services">Choose a service:</label>
    <select id="services">
        <option value="nuuka">Nuuka Open API</option>
        <option value="service2">Service 2</option>
        <option value="service3">Service 3</option>
    </select>
```

Käyttäjän valinnasta otettaisiin sitten asiakkaan koodin puolella koppi:

```
const chosenService = document.getElementById("services").value;
```

Tämä valinta lähetettäisiin palvelimelle päivämäärien ja muiden käyttäjän valintojen mukana ja käsiteltäisiin siellä.

```
def get_url(choice) {
    nuuka_url = "https://helsinki-openapi.nuuka.cloud/api/....etc"
    if (choice == "nuuka"): return nuuka_url
    return ""

api_url = get_url(chosen_service);
if (url == ""):
        #TODO Error message
        return
```

Tässä toteutuksessa pitäisi huomioida se, että muiden palveluiden JSON-datan ominaisuudet saattavat olla erinimisiä kuin Nuuka Open API:ssa. Esimerkiksi alla oleva silmukka ei välttämättä tulostaisi jokaisen päivän sähkönkulutusta toisen palvelun datasta.

```
for item in data: #JSON content in data variable
    value = item["value"]
```

Jotta arrange_monthly -funktio olisi yleiskäyttöinen, eri palveluiden data pitäisi tuunata yhtenäiseen muotoon esimerkiksi erillisessä funktiossa ennen kuin se viedään arrange_monthlylle järjestettäväksi, tai arrange_monthly ottaisi parametreina vastaan nuo JSON-datan ominaisuuksien nimet. Tai sitten jokaisen palvelun datalle voisi tehdä erillisen järjestelyfunktion, mikä ei tunnu kovin hyvältä ratkaisulta, koska arvelen sen johtavan miltei identtisiin funktioihin. Mutta tällaiset asiat selviävät lopullisesti vasta siinä vaiheessa, kun pääsee oikeasti puuhastelemaan toisen palvelun datan kanssa.