Help to complete the tasks of this exercise can be found on the chapter 8 ”Bugs and Errors” and chapter 9 “Regular Expressions” of our course book “Eloquent JavaScript” (3rd edition) by Marijin Haverbeke. The aims of the exercise are to learn to handle exceptional error situations and to use regular expressions to validate and modify strings.

Embed your theory answers, drawings, codes, and screenshots directly into this document. Always immediately after the relevant question. Return the document into your return box in itsLearning by the deadline.

Remember to give your own assessment when returning this document.

It’s also recommendable to use Internet sources to supplement the information provided by the course book.

The maximum number of points you can earn from this exercise is 10 + 1 = 11.

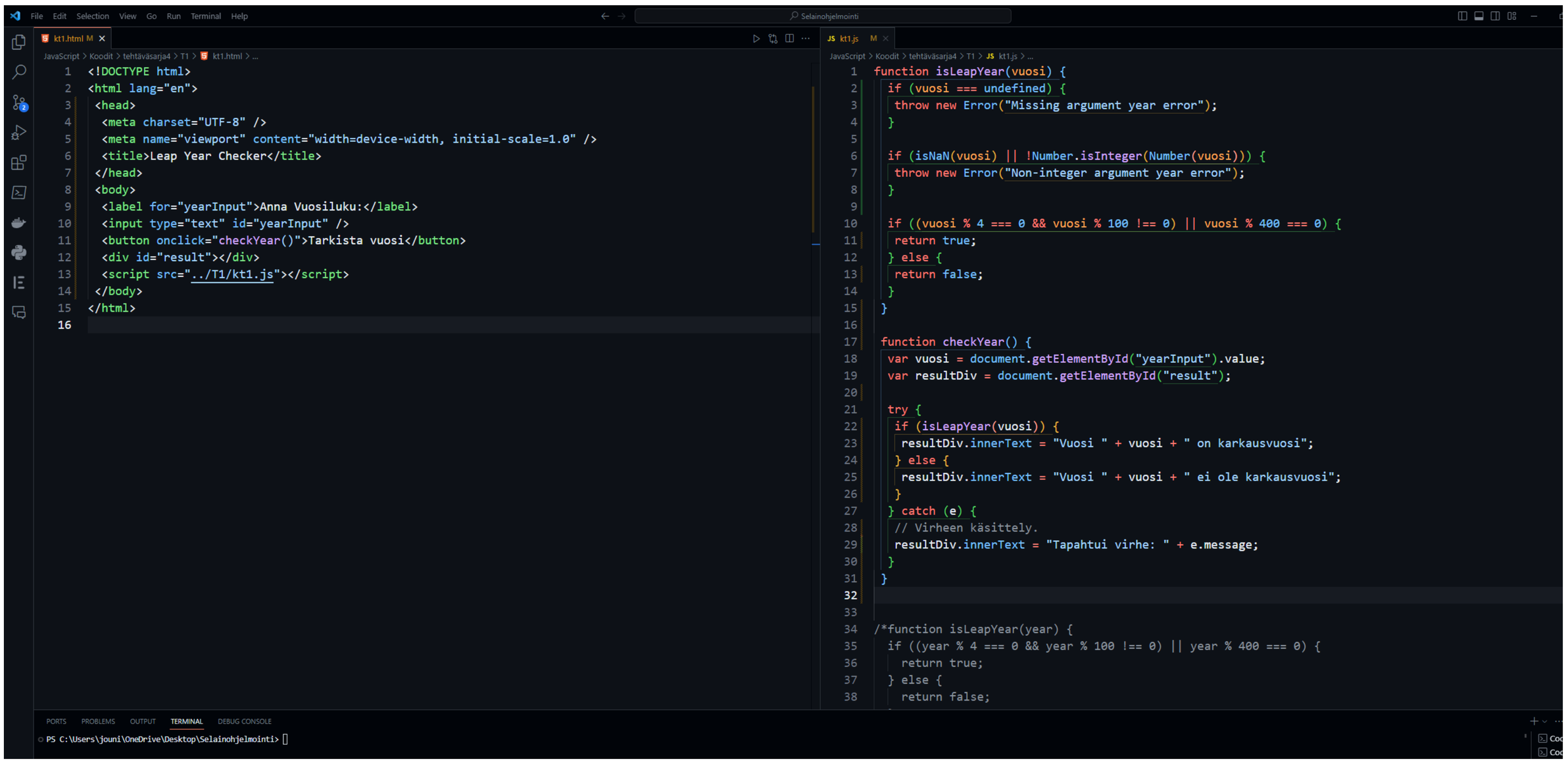
Tasks:

1. Adding error handling into a function. (1 point)

In the practical exercise 2 the task 1 was to program a function isLeapYear. The function accepted one argument year.

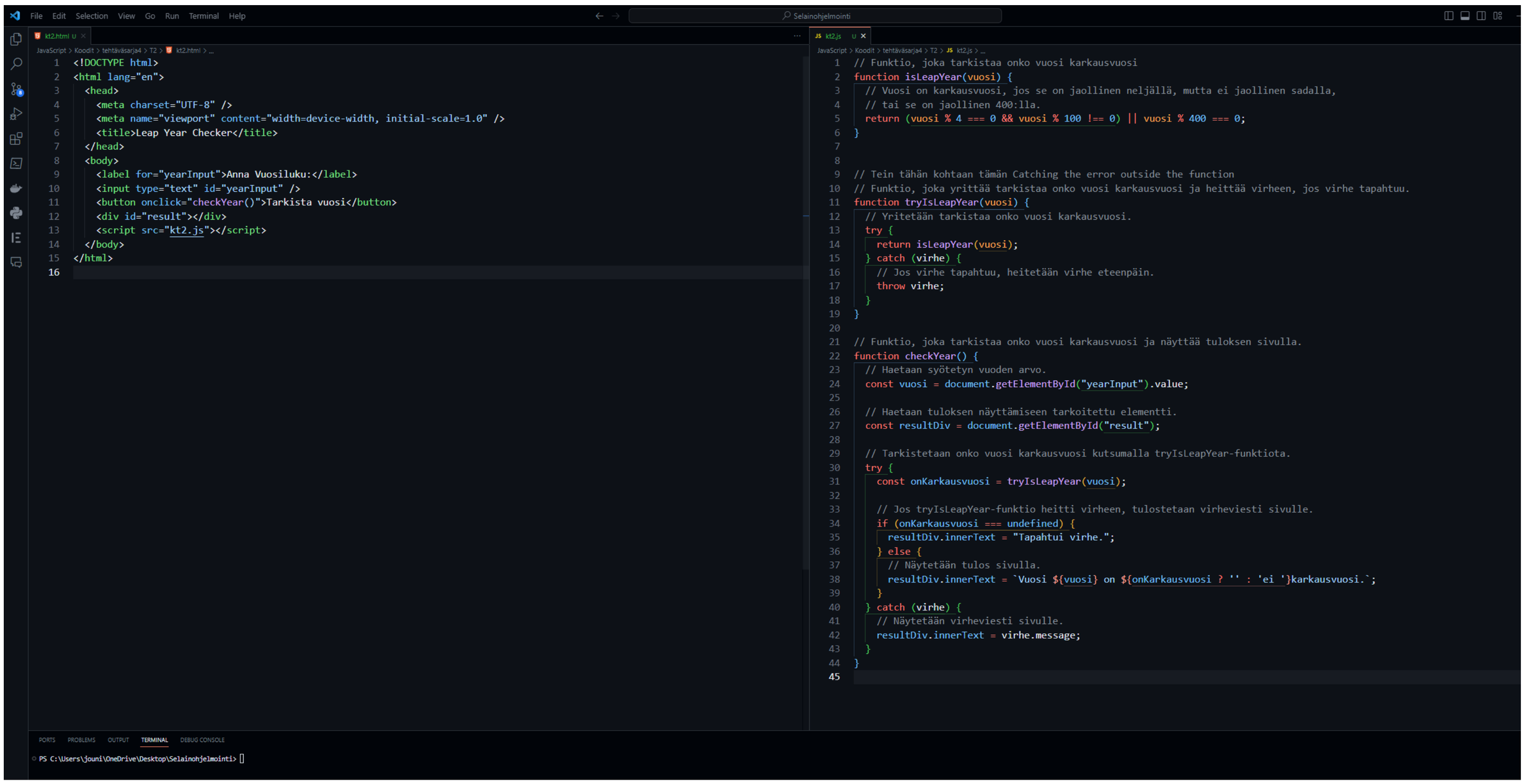
Develop this function further so that it can throw two different kinds of errors when called incorrectly. If the function is called without a parameter, then an error with a message “Missing argument year error” is thrown. If the function is called with non-integer argument, then an error with a message “Non-integer argument year error” is thrown. In this case, write the try-catch statement inside the function and output the error message on the console.

Please note that an argument value 0 should not be considered as an error.



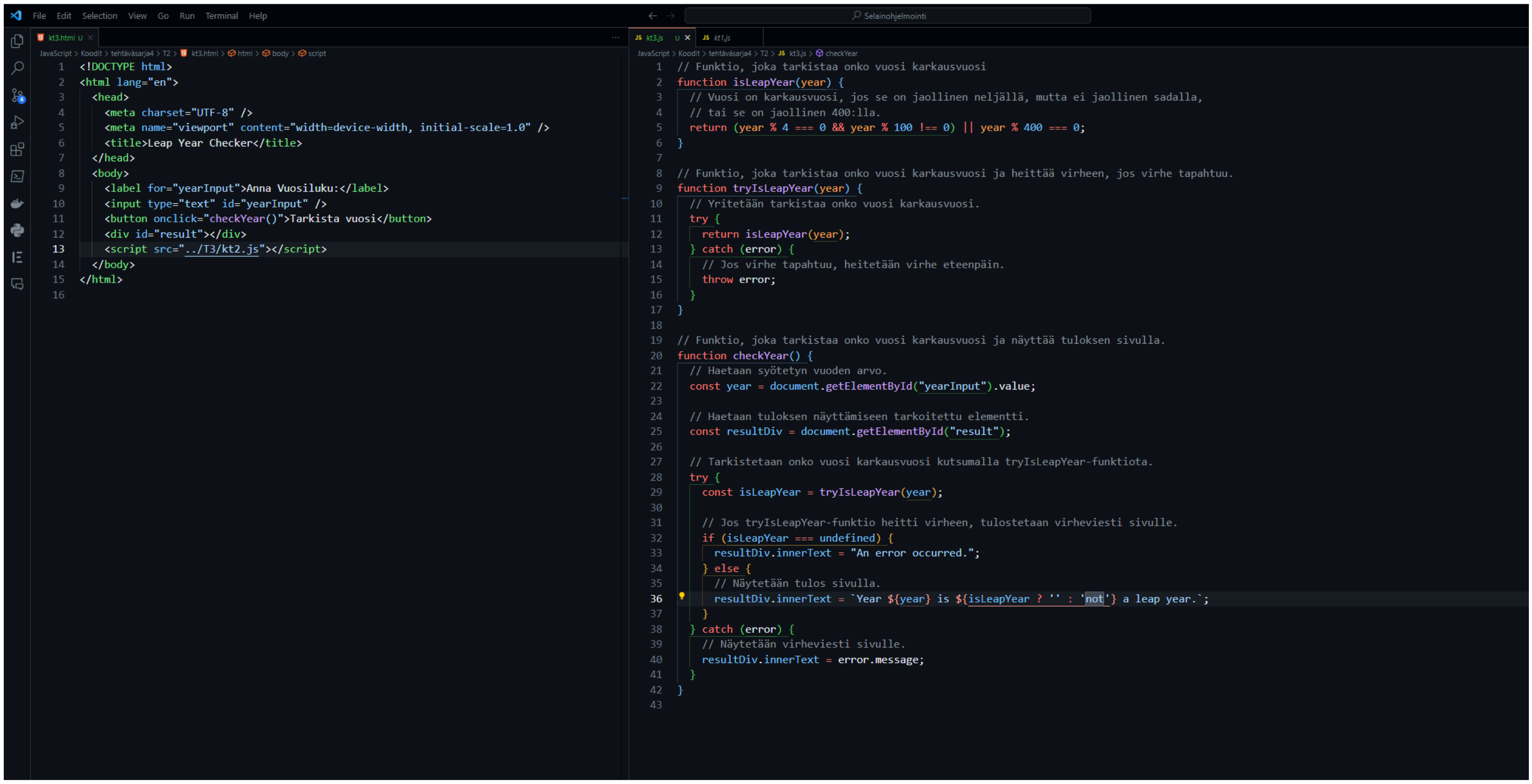
2. Catching the error outside the function. (1 point)

Return to the task above. Change the implementation so that the try-catch is moved to an enclosing operation tryIsLeapYear. The errors are still thrown inside the isLeapYear operation.



3. Throwing and catching the error outside the function. (1 point)

Return to the task above and change the implementation so that also throwing the error is done in the enclosing operation tryIsLeapYear.



4. Error handling best practices. (2 points)

a. In your operations logic, when should you use if statements and when error handling? Give a justified explanation. (0,5 points)

Virheidenhallinnan parhaat käytännöt:

Virheidenhallinnassa on tärkeää tarjota selkeitä virheviestejä, käsitellä virheitä oikealla tasolla sovelluksessa, ja hyödyntää lokitusta tehokkaasti vianmääritykseen. Lisäksi virheitä tulisi käsitellä arvokkaasti tarjoamalla käyttäjille ymmärrettävä palautusvirheiden sijasta.

IF-lauseiden ja Virheenkäsittelyn Käyttö:

Käytä IF-lauseita odotettavissa olevien ehtojen ja päätöksenteon yhteydessä. Virheenkäsittelyä taas käytetään poikkeuksellisissa tilanteissa ja odottamattomien virheiden hallintaan, mahdollistaen keskitetyn ja yhdenmukaisen lähestymistavan virheisiin sovelluksessa. Tämä auttaa varmistamaan sovelluksen vakauden ja joustavuuden ylläpidon.

b. Use Internet and write yourself a short list of the best practices of JavaScript error handling. (You don’t need elaborate on asynchronous code yet.) (1 point)

JavaScriptin virheenkäsittelyssä on hyvä käyttää try-catch-lohkoja virhealttiin koodin suojaamiseksi ja estämiseksi koko sovelluksen kaatumista. Tarjoa yksityiskohtaiset virheviestit, joissa on tietoa virheen luonteesta ja kontekstista, helpottamaan vianetsintää. Logita virheet, käsittele niitä asianmukaisesti sopivalla tasolla, ja käytä JavaScriptin Error-oliota tai luo omia virhe-objekteja. Validoi käyttäjän syötteet ja vältä virheiden tukahduttamista ilman asianmukaista käsittelyä. Lisäksi, kun käytät asynkronista koodia, hyödynnä Promisseja ja niiden **.catch()** -metodia virheiden käsittelyssä varmistaaksesi, että virheet käsitellään asianmukaisesti. Keskitetty virheenkäsittely voi myös auttaa ylläpitämään yhtenäistä virheenhallintaa sovelluksessasi.

c. Analyze the error handling solutions of the above tasks. What are the benefits and drawbacks? What do the best practices say about them? (0,5 points)

- **Hyödyt:**
  - Virheenkäsittely mahdollistaa hallitun reaktion odottamattomiin tilanteisiin.
  - Se auttaa estämään sovelluksen kaatumisen ja parantaa käyttäjäkokemusta.
- **Haitat:**
  - Virheenkäsittely voi tehdä koodista monimutkaisempaa, jos sitä käytetään liikaa tai epäselvästi.
  - Väärin käytettynä se voi piilottaa virheitä ja vaikeuttaa vianmääritystä.

5. What are the risks and benefits of error handling in general? Consider this carefully, because it is very important. (1 point)

Virheenkäsittelyn yleiset riskit sisältävät lisääntyneen monimutkaisuuden ja mahdollisen suorituskykytaakan. Puutteellinen virheenkäsittely saattaa altistaa herkkää tietoa, ja liian yleiset virheviestit voivat vaikeuttaa ongelmien tunnistamista. Toisaalta asianmukainen virheenkäsittely on kriittinen kaatumisten estämisessä, käyttäjäkokemuksen parantamisessa ja tietoturvan varmistamisessa. Se tarjoaa myös tukea vianmääritykseen ja ylläpitää sovelluksen luotettavuutta pitkällä aikavälillä. Tasapainoinen lähestymistapa on olennaista, jotta virheenkäsittely tuo enemmän hyötyä kuin riskejä sovelluskehityksessä.

6. Justify the existence of regular expressions. Why do we need them? Any drawbacks? (1 point)

Säännölliset lausekkeet ovat olennaisia tarkan kuvion tunnistamiseen, mahdollistaen tekstinkäsittelyn, validoinnin ja tehokkaan merkkijonon etsinnän ja korvaamisen. Ne tarjoavat kielestä riippumattoman tavan työskennellä merkkijonokuvioilla eri ohjelmointiympäristöissä. Kuitenkin niiden monimutkaisuus ja mahdolliset luettavuusongelmat, suorituskykyyn liittyvät huolenaiheet sekä oppimiskäyrä voivat olla haittoja, jotka kehittäjien tulee ottaa huomioon varmistaakseen niiden harkitun ja tehokkaan käytön koodissaan.

7. Understanding the functioning of regular expressions. (4 * 0,5 = 2 points)

Explain shortly:

a. How regular expressions consume their input? (What is the mechanism of matching?)

Säännölliset lausekkeet (regular expressions) käyvät läpi syötteensä tietyn säännön mukaisesti. Ne vertaavat säännöllistä lauseketta syötteen eri osiin ja yrittävät löytää vastaavuutta. Matching-mekanismi tarkoittaa sitä prosessia, jossa säännöllinen lauseke vertaillaan syötteen kanssa, ja jos se löytää vastaavuuden, se ilmoitetaan tai tallennetaan. Tämä mahdollistaa esimerkiksi tietyn kuvion tai merkkijonon löytämisen suuremmasta tekstimassasta.

b. What do parentheses around any part of the regular expression cause?

Sulkeet säännöllisen lausekkeen osan ympärillä merkitsevät ryhmää, mikä vaikuttaa siihen, miten lauseke sovitetaan syötteeseen ja miten siihen viitataan myöhemmin. Ryhmän käyttö mahdollistaa esimerkiksi tietyn osan merkityksen yhdistämisen tai toistamisen.

c. What is the difference between lazy matching and greedy matching in regular expressions?

Laiska (lazy) ja ahne (greedy) sovitus säännöllisissä lausekkeissa viittaavat siihen, miten lauseke yrittää sovittaa syötettä.

- **Ahne sovitus (greedy matching):** Yrittää sovittaa mahdollisimman pitkän osan syötteestä, mikäli se täyttää lausekkeen ehdot.
- **Laiska sovitus (lazy matching):** Yrittää sovittaa mahdollisimman lyhyen osan syötteestä, mikäli se täyttää lausekkeen ehdot.

Esimerkiksi, jos käyttää lauseketta **a.*b** ahneesti, se voisi sovittaa koko välillä olevan osan, joka alkaa 'a' ja päättyy 'b'. Laiskassa sovituksessa se yrittäisi löytää mahdollisimman pienen osan, joka täyttää ehdot.

d. When do you need to use RegExp constructor like new RegExp() instead of regular expression literal like / /?

RegExp-konstruktoria (new RegExp()) tarvitaan silloin, kun säännöllisen lausekkeen arvo ei ole tiedossa ennen ohjelman suoritusta, vaan se luodaan dynaamisesti. Toisin sanoen, jos haluat rakentaa säännöllisen lausekkeen merkkijonosta tai muista muuttujista ohjelman aikana, käytät RegExp-konstruktoria.

Esimerkki:

```
var patternString = "ab+c";

var regexLiteral = /ab+c/;  // Tämä on säännöllinen lauseke kirjoitettuna suoraan koodiin

var regexConstructor = new RegExp(patternString);  // Tämä luo säännöllisen lausekkeen dynaamisesti
```

Jos säännöllisen lausekkeen arvo tiedetään jo koodin kirjoitusvaiheessa, voit käyttää suoraa säännöllisen lausekkeen kirjallista muotoa (***/ /***). Tämä on yleensä suositeltavaa, kun mahdollista, koska se on lyhyempi ja luettavampi.

8. Using regular expressions. (2 points)

Develop a function buildRegisterNumber so that it can throw two different kinds of errors when called with incorrect argument values. The function takes two arguments: theLetters and theDigits. If the value of theLetters argument is not valid, then an error with a message “Invalid register number letters” is thrown. If the value of theDigits argument is not valid, then an error with a message “Invalid register number digits” is thrown. In case the arguments are valid, then a valid register number is returned.

Use regular expressions to validate the argument values.

Let’s agree that a valid register number obeys the following rules

- there are from two to three uppercase letters before a dash
- the letter W is not allowed
- a dash
- there are from one to three digits after the dash
- no leading zeros are allowed (no zeros before first non-zero digit)

Examples of valid register numbers:

AX-12

UUI-6

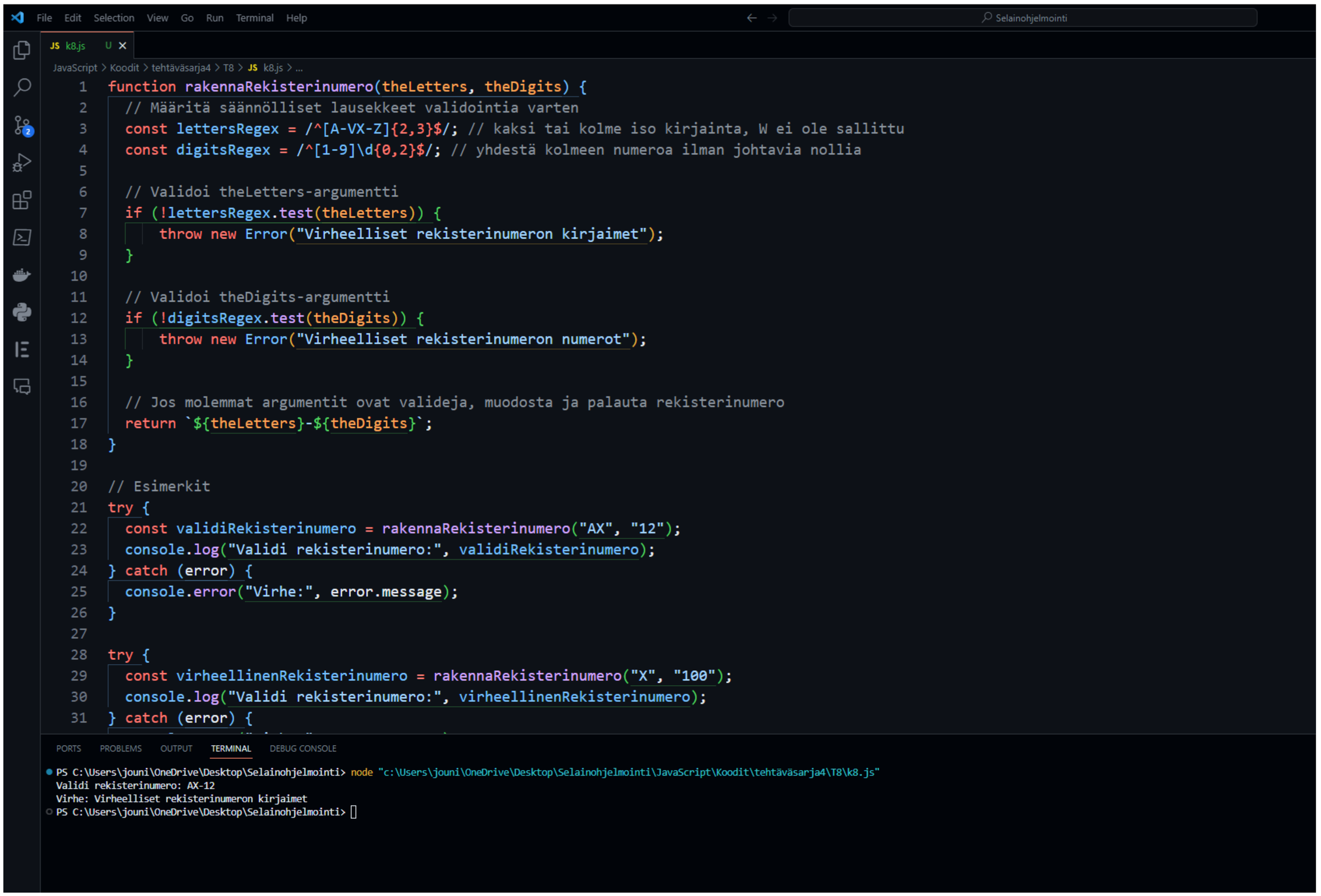
GFS-200

Examples of invalid register numbers:

X-100

YUT-020

WWW-100


File Edit Selection View Go Run Terminal Help
Selainohjelmointi
JS k8.js U
JavaScript > Koodit > tehtäväsarja4 > T8 > JS k8.js > ...
function rakennaRekisterinumero(theLetters, theDigits) {
// Määritä säännölliset lausekkeet validointia varten
const lettersRegex = /^[A-VX-Z]{2,3}$/; // kaksi tai kolme iso kirjainta, W ei ole sallittu
const digitsRegex = /^[1-9]\d{0,2}$/; // yhdestä kolmeen numeroa ilman johtavia nollia
// Validoi theLetters-argumentti
if (!lettersRegex.test(theLetters)) {
throw new Error("Virheelliset rekisterinumeron kirjaimet");
}
// Validoi theDigits-argumentti
if (!digitsRegex.test(theDigits)) {
throw new Error("Virheelliset rekisterinumeron numerot");
}
// Jos molemmat argumentit ovat valideja, muodosta ja palauta rekisterinumero
return `${theLetters}-${theDigits}`;
}
// Esimerkit
try {
const validiRekisterinumero = rakennaRekisterinumero("AX", "12");
console.log("Validi rekisterinumero:", validiRekisterinumero);
} catch (error) {
console.error("Virhe:", error.message);
}
try {
const virheellinenRekisterinumero = rakennaRekisterinumero("X", "100");
console.log("Validi rekisterinumero:", virheellinenRekisterinumero);
} catch (error) {
PORTS PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi> node "c:\Users\jouni\OneDrive\Desktop\Selainohjelmointi\JavaScript\Koodit\tehtäväsarja4\T8\k8.js"
Validi rekisterinumero: AX-12
Virhe: Virheelliset rekisterinumeron kirjaimet
PS C:\Users\jouni\OneDrive\Desktop\Selainohjelmointi>