Insegnamento di Tecnologie Web

CdS In Informatica

(A.A. 2022-23)

Esame scritto del 25/06/2023

Nome:

Cognome:

Matricola:

Corso di Studi

Solo se diverso da Informatica Triennale

Anno di frequenza

Come specificato nel piano di studi: o "2022-23" " oppure "precedente".

Attenzione:

- Questi computer sono limitati ad accedere solo ad alcuni siti: `eol.unibo.it`, `virtuale.unibo.it`, `developer.mozilla.org` e `getbootstrap.com`. Non funzionano Google, stack overflow, etc.
- *Rispondete solo negli spazi delimitati dai blocchi* ```` ``` qui la risposta ``` ````*, senza modificarli o eliminarli.*
- *Consegnate* solo *questo file. Copiate ed incollate dentro agli appositi spazi la vostra risposta per intero.*
- *Potete decidere se inserire il CSS inline nel file HTML o metterlo in un file esterno. Nel secondo caso inserite l'elemento nella posizione corretta e mettete il CSS in un blocco separato.*
- *You can use either English or Italian for your answers.*
- *Per favore, per favore, per favore: nesun erore di ortografia. Cuesta è un universita e non la squola elementale.*

Domanda #1 - Domande di base (6 punti totali)

> Queste risposte hanno ottenuto 1.00/1.00 (successivamente moltiplicato per i 6 punti dell'esercizio)

a) Semantic Web

Rappresentare in RDF i seguenti dati: "Umberto Eco era un professore italiano dell'Università di Bologna, nato il 1 maggio 1932. Informazioni su di lui sono reperibili al sito http://www.umbertoeco.com." Potete usare un formato a vostra scelta tra Turtle, XML-RDF o JSON-LD.

```
# gli url sono copiate dalle slides, puo' darsi
# che alcuni CURIE non esistano realmente, in tal
# caso bisognerebbe trovare la versione corretta equivalente online

@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix my: <url>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

my:umbertoeco
  rdf:type foaf:person;
  foaf:name "Umberto Eco";
  rdf:born_date "1932-05-01";
  rdf:is_alive "false"; # per dire che e' gia' morto ("ERA" verbo al passato)
  rdf:link "http://www.umbertoeco.com";
  rdf:nationality "italian";
  rdf:job my:job;

my:job
  rdf:type foaf:profession;
  rdf:name "professore universitario";
  rdf:work_place "universita' di Bologna";
```

b) Codifica caratteri

Quanti byte sono richiesti in UTF-8 per rappresentare le seguenti parole? NB: la parola in arabo è composta da 5 caratteri; la parola in giapponese da 2.

- Giapponese:
- Arabo:
- Tedesco: Stadt
- Portoghese: cidade
- Italiano: città
- Turco:

```
- Giapponese sono 6 bytes, tre bytes per carattere
- arabo sono 2 byte per carattere, quindi in totale sono 10 bytes.
```

```
- tedesco: 5 bytes, un byte per carattere
- cidade sono 6 bytes, un byte per carattere
- citta' sono 6 bytes, 4 per "citt" e 2 per "a" accentata.
- turco sono 2 byte per carattere, in totale sono 10 bytes.
```

c) Javascript

Spiegare il funzionamento della coercizione dei tipi (*type coercion*) in Javascript. Aggiungere un esempio (diverso da quello delle slide) sul modo in cui funziona. Mostrare un caso in cui può essere utile e un caso in cui è preferibile evitare la conversione automatica.

```
Javascript ha i tipi dinamici, quindi uno stesso nome puo' cambiare tipo durante il suo life
Durante le operazioni, i tipi sono convertiti in automatico, per esempio una operazione del
Pero' non sempre e' un esempio positivo perche' ad esempio se ho a=NaN, un valore not a numb
un tipo number, la conversione in automatico durante l'espressione a + "1" risulterebbe in "
un risultato molto sensato.

Per ovviare a questo genere di conversioni automatiche si potrebbe utilizzare Typescript, ch
```

d) CSS

Dato il seguente esempio, calcolare la dimensione del font per la stringa “sic filius”:

```
<style>
    body {
      font-size: 16px;
    }

    .container {
      font-size: 1.5em;
    }

    .child {
      font-size: 1.2em;
    }
  </style>

<body>
  <div class="container">
    <p>Lorem ipsum dolor sit pater</p>
    <p class="child">sic filius</p>
  </div>
</body>
```

```
il font size e' l'ultimo applicato ossia 1.2 em, quindi 16 *1.5 *1.2 definito dalla classe c
```

Domanda #2 - HTML + CSS (10 punti)

Scrivere il codice HTML e CSS cercando di riprodurre la seguente pagina web: E' composta da uno sfondo, una card su un livello superiore. La card contiene, oltre al testo, un input "Month" che contiene i mesi come opzioni e un input "Year" che consente solamente di inserire numeri. Il testo può essere un qualunque lorem ipsum. Insieme al presente file, dovresti vedere l'immagine per lo sfondo. Il logo "Be" non è necessario, così come non sono richiesti altri elementi oltre alla card e al titolo Behance a sinistra. * Si cerchi di riprodurre in maniera consistente il layout e si dedichi meno tempo al contenuto dei singoli elementi.

L'immagine di sfondo (background.jpg) è in uno zip scaricabile da EOL.

> Questo esercizio ha ottenuto 1.00/1.00, poi moltiplicato per 10 punti.

Codice HTML *(ed eventualmente CSS interno)*

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Hello world</title>
    <link rel="stylesheet" href="https://www.fabiovitali.it/TW/lib/bootstrap.min.css">
    <script src="https://www.fabiovitali.it/TW/lib/jquery-3.6.0.js"></script>
    <script src="https://www.fabiovitali.it/TW/lib/bootstrap.js"></script>
  </head>
  <style>
.back {
  width: 100vw;
  height: 100vh;

  background-image: url("immagini/background.jpg");
  background-size: cover;
  background-blend-mode: darken;
}
  </style>
  <body>
    <div class="back">
      <div class="row h-100">
        <div class="col">
          <div class="h-100 d-flex justify-content-center align-items-center">
            <img alt="logo" src="." >
            <h1 style=" color:white;" > BeHance </h1>
          </div>
```

```
        </div>

        <div class="col py-5">
          <div class="card w-100 h-100" >
            <div class="card-body">
              <h1 class="card-title py-3 fw-bold">Card title</h1>
              <p class="card-text pt-3">Lorem ipsum dolor sit amet, consectetur adipiscing e
                        eget pulvinar lobortis. Vestibulum ante ipsum primis in faucibus orc
                        et ultrice</p>

              <form>
                <label class="pb-1" style="font-weight: 200;"> Date of Birth
                <span class="badge rounded-pill bg-primary">!</span></label>

                <div class="input-group mb-3">
                <select class="form-select" aria-label="Default select example">
                  <option selected>Month</option>
                  <option value="1">January</option>
                  <option value="2">February</option>
                  <option value="3">march</option>
                  <option value="4">April</option>
                  <option value="5">May</option>
                  <option value="6">June</option>
                  <option value="7">July</option>
                  <option value="8">August</option>
                  <option value="9">September</option>
                  <option value="10">October</option>
                  <option value="11">November</option>
                  <option value="12">December</option>
                </select>
  <input type="number" class="form-control" placeholder="2022" aria-label="Year" >
</div>
<button type="submit" class="btn btn-primary" style="border-radius: 20px; float:right;">Cont
              </form>
            </div>
          </div>
        </div>
        <div class="col-2">
        </div>
      </div>
    </div>
    <div class="container border p-5 my-3">
      <h1>Hello world</h1>
  </body>
</html>
```

Codice CSS ***(solo se esterno)***

Domanda #3 - Javascript (12 punti)

> Questo esercizio è stato valutato 0.9/1.00 (poi moltiplicato per 12 punti) perché a Vitali importa solo che ci sia l'"idea", non che funzioni. Però credo di aver capito male alcuni pezzi della consegna, perdendo un punto.

Creare il front end di un'app che serve a tradurre testo in linguaggio naturale. Utilizzare l'ipotetica API "OpenTraslator".

Routes

Le routes dell'API sono le seguenti:

- `GET /languages`: restituisce un json con tutte le lingue disponibili, utilizzando il codice ISO 639-2. Il JSON appare così: `{ "eng": "English", "spa": "Spanish", "fra": "French", "deu": "German", "ita": "Italian", ... }`
- `POST /translate`: richiede body che contiene un testo, con lingua di partenza e lingua di arrivo.
  `{ "text": "Testo di origine", "source_language": "ita", "target_language": "deu" }`
- `POST /detect-language`: richiede un body che contiene un testo. Per ricevere solo la prima lingua rilevata si usi `/detect-language/first`. `{ "detected_languages": [ { "language": "eng", "confidence_score": 0.85 }, ... ] }`

Parte I Scrivere il codice HTML (se si vuole, anche CSS) e JS per gli input di scelta della lingua di partenza e della lingua di arrivo. Si può scegliere da un dropdown una lingua tra quelle disponibili, o scrivere in un input la lingua. L'input mostra il nome della lingua, non il suo codice ISO.

<!doctype html>

Hello world

Open this select menu One Two Three

Open this select menu One Two Three

```
<h1>Hello world</h1>
```

// script.js const openTranslatorBase = "someurl.com/api"

function showerror(message) { alert(message); }

function populateLanguages(data, selectEl) { selectEl.innerHTML = "Open this select menu" //andiamo a popolare le selezioni con le informazioni del server

Object.keys(data).forEach((key) => { selectEl.innerHTML += `<option value="${key}">${data[key]}</option>` }); }

function getLanguages() { fetch(`${openTranslatorBase}/languages`, { method: “GET”, }) .then((response) => { if (!response.ok) throw new Error(“error in the request”); return response.json(); }).then((data) => {

```
  const startLanguage = document.getElementById('l-partenza');
  const endLanguage = document.getElementById('l-arrivo');
  populateLanguages(data, startLanguage);
  populateLanguages(data, endLanguage);
}).catch(e => {
  if (e.message) showerror(e.message);
  else showerror("error in the request");
});
```

}

document.addEventListener(“load”, () => { const form = document.querySelector(‘#form-select’); form.addEventListener(‘submit’, (e) => { e.preventDefault(); // Prevent form from submitting and to refresh the page

```
const startLanguage = document.getElementById('l-partenza').value;
const endLanguage = document.getElementById('l-arrivo').value;
// do something with the values.
```

}) });

Parte II Partendo dal codice dell’esercizio precedente, aggiungere un input per il testo da tradurre e un box dove si mostra il testo tradotto nella lingua selezionata.

```
// html

    <!-- Sopra e' lo stesso, ci sono i due select -->
      <textarea id="l-input">
      </textarea>

      <div id="l-out">
      </div>
      <input type="submit" value="Submit">

    </form>
    <!-- Sotto e' lo stesso -->


// script.js

// sopra e' lo stesso codice
function askTranslation(callback) { // -> returns translated text
```

```
  const startLanguage = document.getElementById('l-partenza').value;
  const endLanguage = document.getElementById('l-arrivo').value;
  const inputText = document.getElementById('l-input').value;
  fetch(`${openTranslatorBase}/translate`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: {
      "text": inputText,
      "source_language": startLanguage,
      "target_language": endLanguage
    }
  })
    .then((response) => {
      if (!response.ok) throw new Error("error in the request");
      return response.json();
    }).then((data) => {
      callback(data);
    }).catch(e => {
      if (e.message) showerror(e.message);
      else showerror("error in the request");
    });
}

function showTranslation(text) {
  const outDiv = document.getElementById("l-out");
  outDiv.innerHTML = `<p> ${text} </p>`
}


document.addEventListener("load", () => {
  const form = document.querySelector('#form-select');
  form.addEventListener('submit', (e) => {
    e.preventDefault(); // Prevent form from submitting and page refresh

    askTranslation(showTranslation);
  })
});
```

Parte III Partendo dal codice dell'esercizio precedente, aggiungere un bottone che richiede il riconoscimento automatico del testo nel box input. Questo bottone disabilita il bottone di lingua di partenza.

- Se l'input è inferiore a 10 caratteri, informare l'utente che non sono sufficienti.

- Se l'input è tra i 10 e i 25 caratteri, mostrare le prime tre lingue per probabilità.
- Se l'input è superiore a 25 caratteri, utilizzare la prima lingua per probabilità e richiedere la traduzione nella lingua target.

```
// html

    <!-- Parte sopra lo stesso-->
      <div id="l-out">
      </div>

      <button type="button" class="btn btn-primary" id="l-rileva">Rileva testo</button>
      <input type="submit" value="Submit">

    </form>
    <!-- Parte sotto lo stesso-->

// script.js
// la parte di sopra e' lo stesso
function showTranslation(text) {
  const outDiv = document.getElementById("l-out");
  outDiv.innerHTML = `<p> ${text} </p>`
}

function disableInputSelection() {
  const startLanguage = document.getElementById('l-partenza');
  startLanguage.setAttribute("disabled", true);
}

function handleMultipleDetection(text, callback) {
  fetch(`${openTranslatorBase}/detect-language`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: {
      text
    }
  })
    .then((response) => {
      if (!response.ok) throw new Error("error in the request");
      return response.json();
    }).then((data) => {
      callback(data.slice(0, 3)); // assumo il ritorno sia array, tengo i primi 3
    }).catch(e => {
      if (e.message) showerror(e.message);
```

```
      else showerror("error in the request");
    });
}

function handleSingleDetection(text, callback) {
  fetch(`${openTranslatorBase}/detect-language/first`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: {
      text
    }
  })
    .then((response) => {
      if (!response.ok) throw new Error("error in the request");
      return response.json();
    }).then((data) => {
      callback(data.language);
    }).catch(e => {
      if (e.message) showerror(e.message);
      else showerror("error in the request");
    });
}

function makeLanguageDetection() {
  const inputText = document.getElementById('l-input').value;
  if (inputText.length < 10) {
    showerror("non sono sufficienti, almeno 10 caratteri");
  } else if (inputText.length < 25) {
    handleMultipleDetection(inputText, (languages) => {
      alert(`I linguaggi possibili sono ${languages}`);
    });
  } else {
    handleSingleDetection(inputText, (languageCode) => {
      // questo e' considerabile un hack per far funzionare il codice
      // di askTranslation correttamente
      document.getElementById('l-partenza').value = languageCode;
      askTranslation(showTranslation);
    });
  }
}

document.addEventListener("load", () => {
  const form = document.querySelector('#form-select');
  form.addEventListener('submit', (e) => {
```

```
    e.preventDefault(); // Prevent form from submitting and page refresh

    askTranslation(showTranslation);
  })


  const button = document.getElementById("l-rileva");
  button.addEventListener("click", () => {
    disableInputSelection();
    makeLanguageDetection();
  });
});
```

Domanda #4 - Framework (6 punti)

Si scriva il codice necessario ad avere un input testuale che si blocca oltre i 300 caratteri e che mostra costantemente il numero di caratteri presenti nell'input, il numero di parole e il numero di caratteri rimanenti.
Se i caratteri rimanenti sono >300, a scelta, si blocchi l'input dopo un alert, oppure si mostrino nell'output tutti i caratteri successivi con un colore diverso, o altro a vostra scelta purche sensato.

- Si parta da un framework a vostra scelta, e.g. React, Angular, Vue, Svelte, etc.
- Si può suddividere l'esercizio in più componenti, ma devono essere tutti richiamati nel componente padre.
- Il numero di parole si calcola quantomeno dividendo una stringa sulla base degli spazi. Ulteriori accorgimenti aumentano la valutazione.
- Il risultato deve apparire più o meno così:

  Questa soluzione ha ottenuto 6 punti, 1.00/1.00

```
// App.jsx
//
// suppongo sia il componente corretto dentro un npx create-react-app [nome]
import {useState} from 'react';


function Textbox() {
  // conterra' lo stato interno, poi aggiornato in volta in volta quando l'utente
  // immettera' input nella textarea.
  const [text,setText] = useState("");
  const maxSize = 300;
  const handleChange = (e) => {
    const newText = e.target.value;
    if (newText.length > maxSize) {
      alert(`non e' piu' possibile scrivere di piu'. Max ${maxSize} caratteri`);
    } else {
      setText(newText);
    }
  }; // si potrebbe anche utilizzare un useCallback, rendrebbe
  // un po' piu' efficiente dato che e' dipendente solo da text questo onchange.
  //

  return (
    <>
      <textarea value={text} onChange={setText} col="20"/>
      <div class="d-flex">
        <p>
        Caratteri: {text.length}
```

```
        </p>
        <p>
        Parole: {text.split(" ").length} // conto le parole dal numero di spazi presenti,
    // si potrebbe utilizzare una regex in modo che due "  " non siano contati come una paro
    // pseucodice:
    // regex che riconosce molteplici spazi contigui, e li compatta in uno
    // poi eseguire split come si fa sopra, ma avrei bisogno di documentazione per regex su
    // scrivere questo codice.
        </p>
        <p>
        Rimanenti: {maxSize - text.length}
        </p>
      </div>
    </>
  );
}

// funzione principale, che conterra' il componente figlio textbox
function App() {
  return (
    <> // uso un fragment che poi non sara' renderizzato\
    <label> Input testuale:
    <Textbox/>
    </label>
    </>
  );
}


export default App;
```