PART 2

Week 6: The Web

1 HTTP

1.1. Setup

1.2. Exploring HTTP

1.3. Online research

The fragment part of a URL.
The URI fragment is the optional part at the end of a URL which starts with a hash (#) character. It lets you refer to a specific part of the document you've accessed.
source: How to Share Links that Anchor to Any Text on a Webpage
URL: hash property - Web APIs | MDN

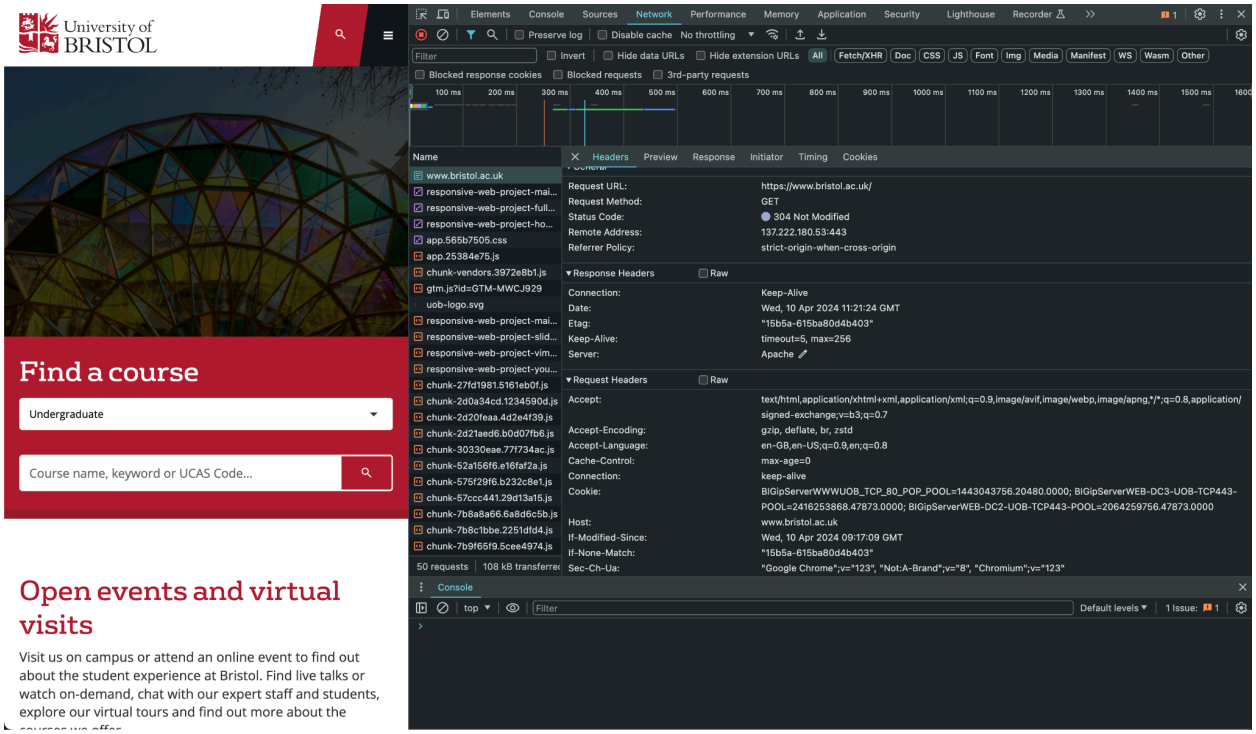
The Accept header sent by the HTTP client.
The Accept request HTTP header indicates which content types, expressed as MIME types, the client is able to understand
source: Accept - HTTP | MDN

The User-agent header sent by the HTTP client. What does your browser send?
The User-Agent request header is a characteristic string that lets servers and network peers identify the application, operating system, vendor, and/or version of the requesting user agent.
source: User-Agent - HTTP | MDN

How do you encode a path that contains a space in an URL? Which other characters are "special" and need to be treated this way in paths?
URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign or with %20.

Lots of special characters listed here:
HTML URL Encoding Reference.

Which web server is the University of Bristol using for www.bristol.ac.uk, according to the headers? Read up online on this server and the organisation of the same name that maintains it.
Apache.
https://httpd.apache.org/



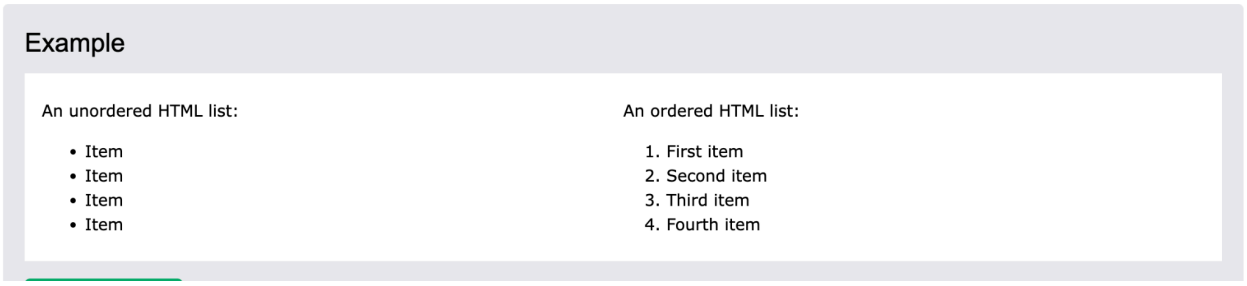
source:

1.4. A server in Java

2 HTML5

Video

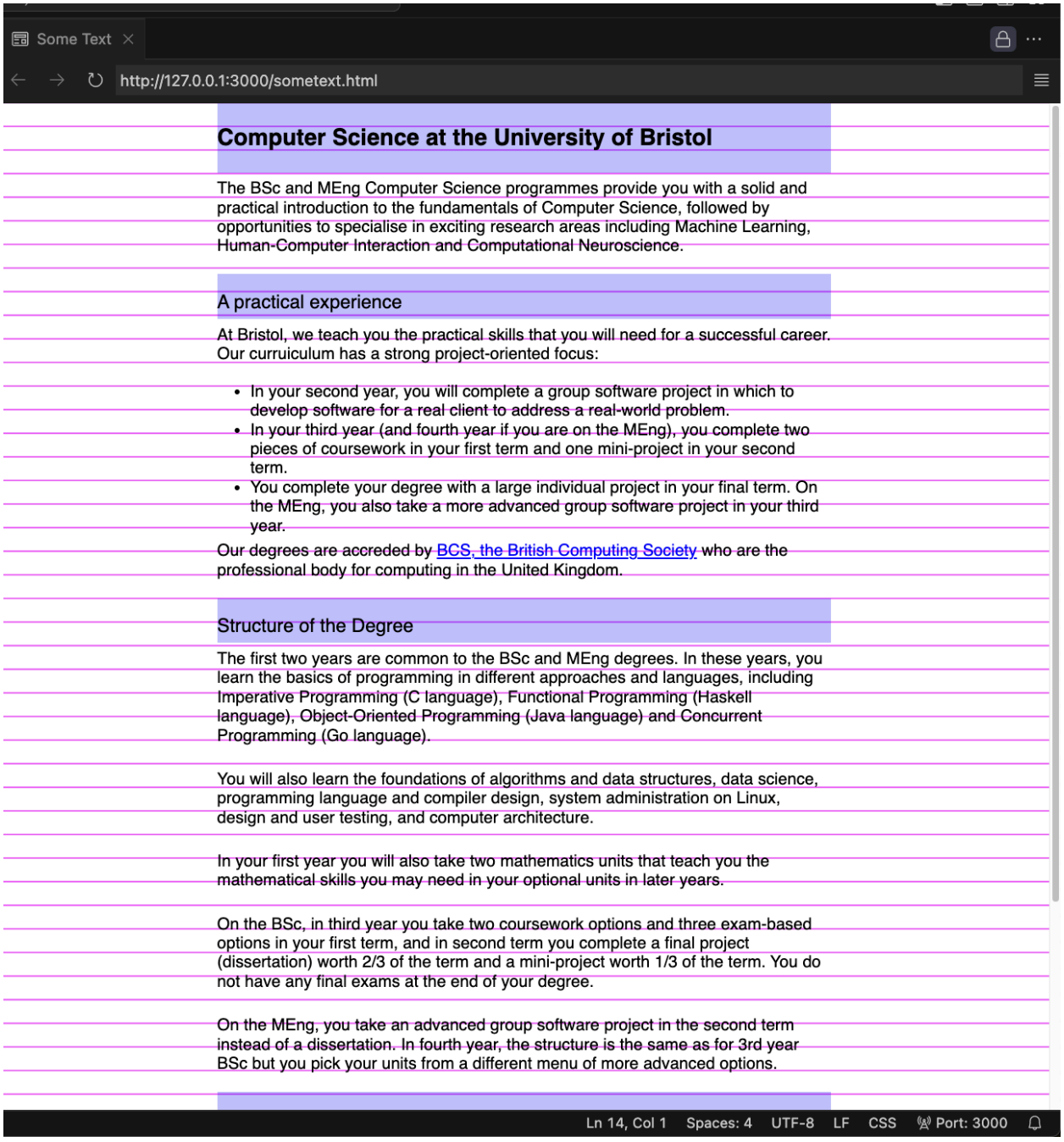
What's the difference between an unordered list <ul> and an ordered list ?



2.1. Basic HTML5

2.2. Templates

Week 7: CSS




```
body {
    margin: 0 auto;
    max-width: 40em;
    line-height: 125%;
    font-size: small;
    font-family: sans-serif;
    background-image: url("baseline.png");
```

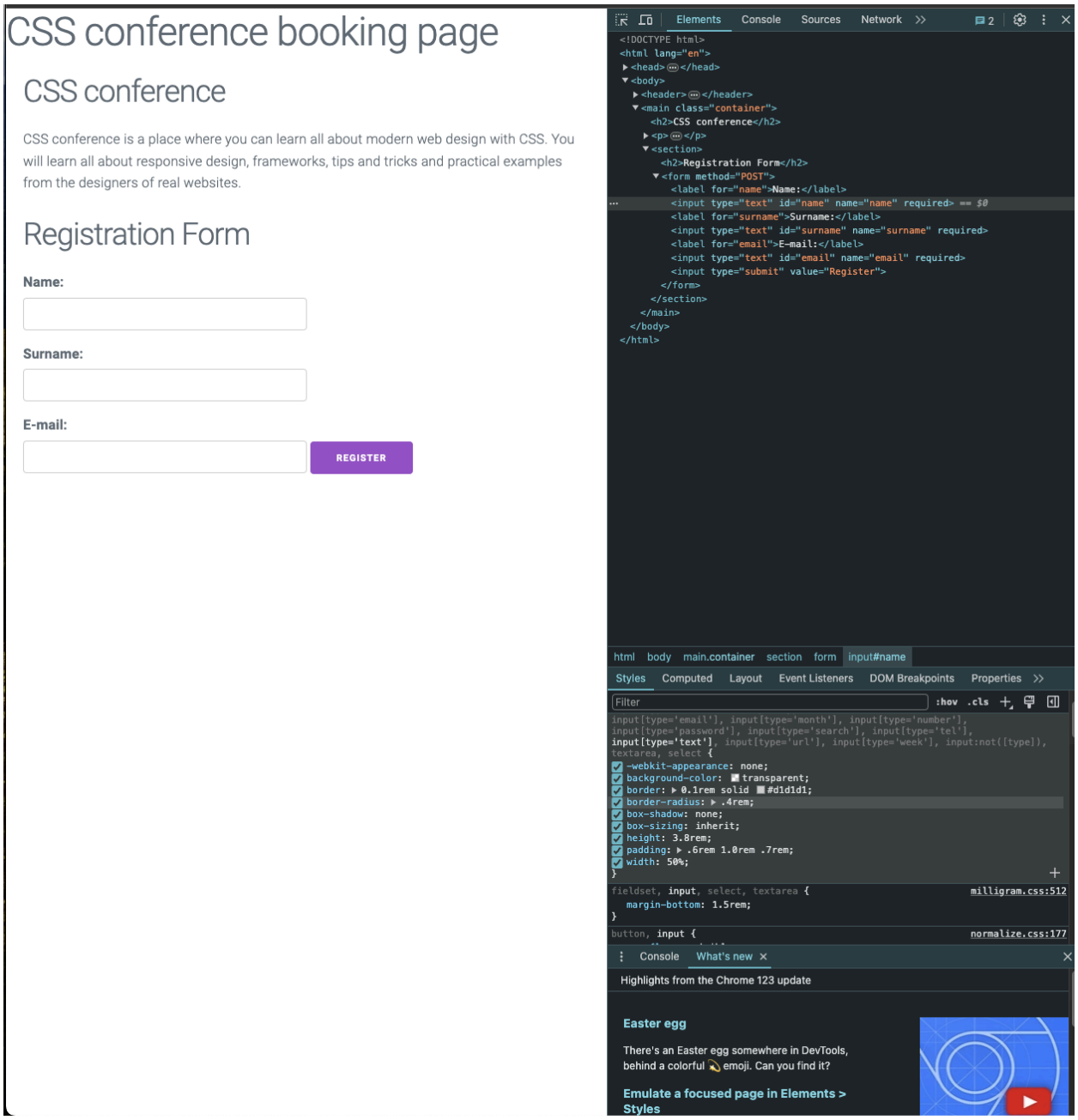
```
}
h1{
    font-size: 150%;
    font-weight: bold;
    background-color: rgba(0, 0, 255, 0.25);
    padding-top: 4%;
    padding-bottom: 4.2%;
}
h2{
    font-size: medium;
    background-color: rgba(0, 0, 255, 0.25);
    padding-top: 3%;
    padding-bottom: 1.2%;

}
p{
    font-size: 14px;
    padding-top: 1%;
    padding-bottom: 3%;
}
ul{
    padding-left: 2em; /* the bullets appear in the padding */
    list-style-type: disc;
    list-style-position: outside;
    font-size: 14px;
}
a{
    color:blue;
}
```

3.2. Frameworks

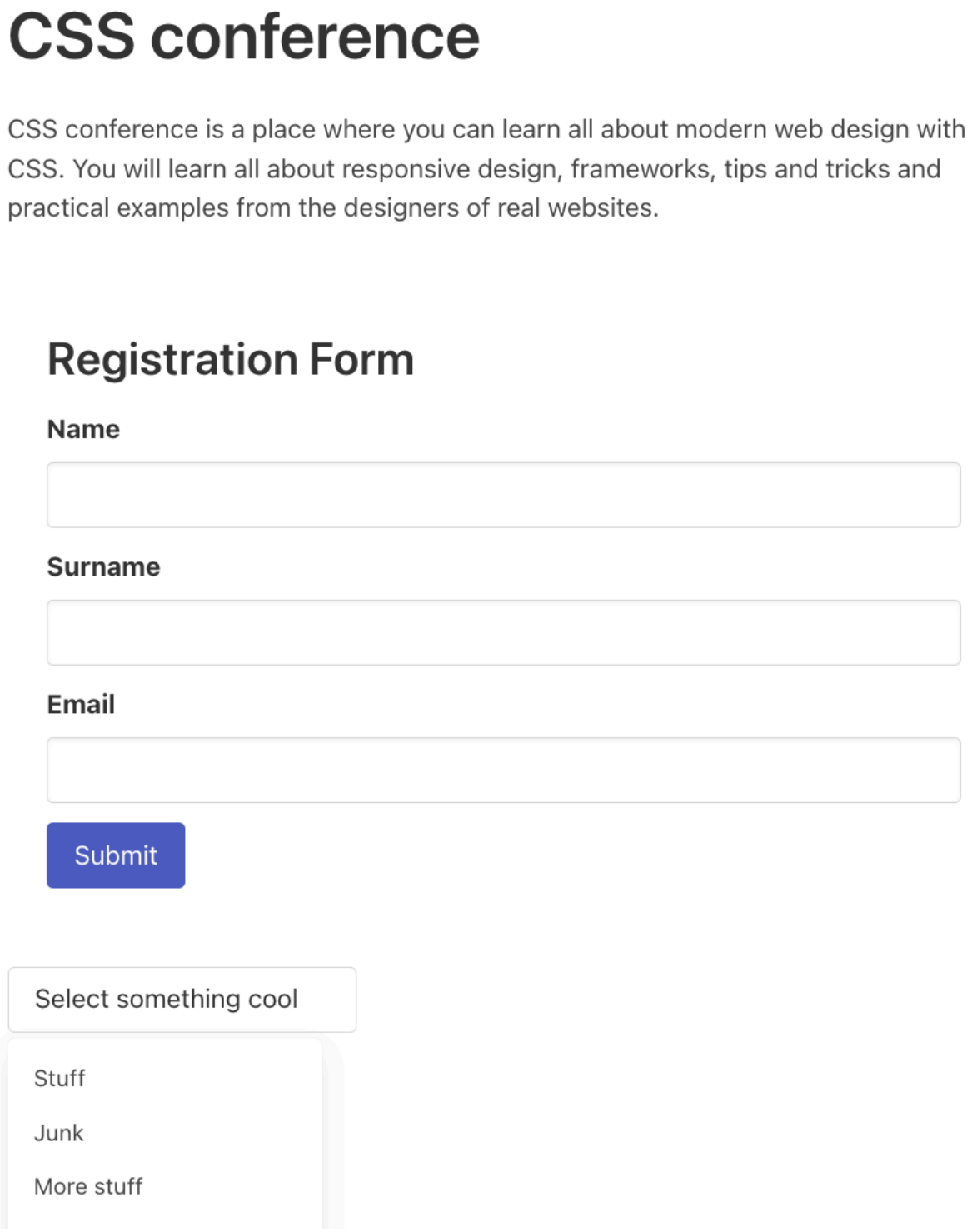
How does Milligram set the following?

- Size of heading fonts
    - font-size: 3.6rem;
- Form fields take up the full width of the container
    - width: 100%;
- Form labels and fields appear below each other

- Labels are closer to their own field than the field above
    - margin-bottom: .5rem;
- Size and centering of everything in a container on wide enough screens



Bulma
https://pastebin.com/QEGVjuR6

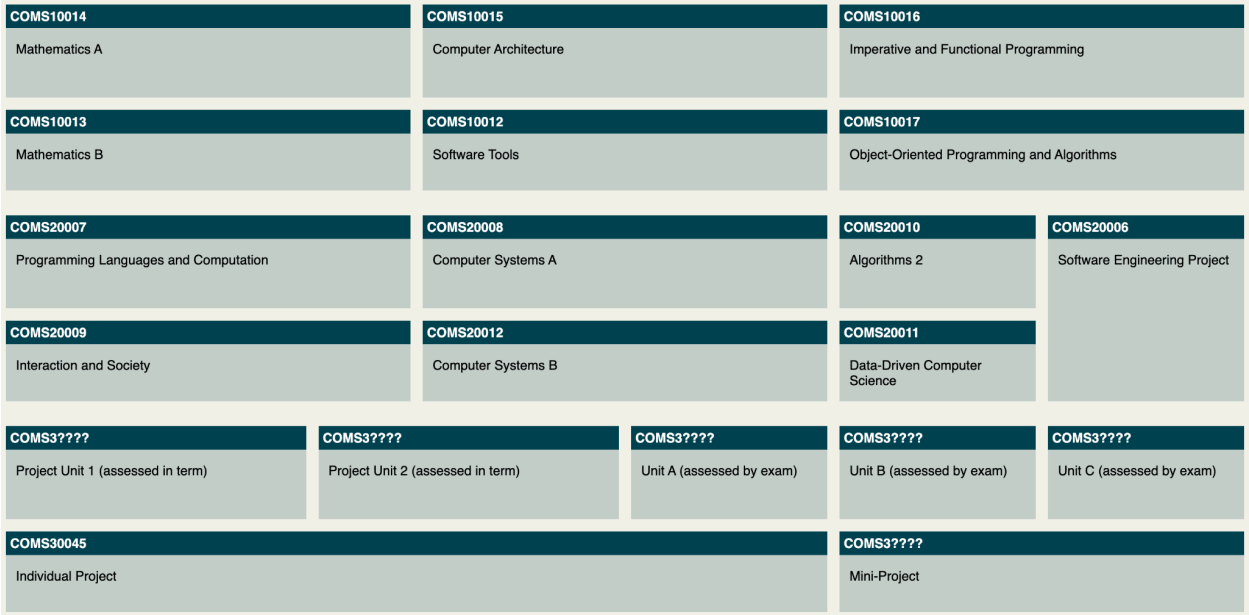


4 CSS grids

4.2. Curriculum exercise

Isaac's Solution:

The UoB CS BSc curriculum as a CSS grid




```css
main {
      max-width: 1500px;
      margin: 0 auto;
      display: inline-grid;

      grid-template-columns: repeat(12, 1fr);
      gap: 15px;
      grid-template-rows: repeat(12, 1fr);
}

body {
      font-family: sans-serif;
      background-color: rgba(112, 145, 53, 0.1);
}

.unit {
      background-color: rgba(0, 67, 79, 0.2);
      width: auto;
}

.unit b {
      display: block;
      background-color: rgb(0, 67, 79);
      color: white;
      padding: 5px;
```

```
}

.cp10 {
      grid-auto-flow: column;
      grid-column: span 2;
}

.cp15 {
      grid-auto-flow: column;
      grid-column: span 3;
}

.cp20 {
      grid-auto-flow: column;
      grid-column: span 4;
}

.cp40 {
      grid-auto-flow: column;
      grid-column: span 8;
}

.y1-tb1 {
      grid-row: 1;
}

.y1-tb2 {
      grid-row: 2;
      margin-bottom: 15px;
}

.y2-tb1 {
      grid-row: 3;
}

.y2-tb2 {
      grid-row: 4;
      margin-bottom: 15px;
}

.y3-tb1 {
      grid-row: 5;
}
```

```
.y3-tb2 {
      grid-row: 6;
      margin-bottom: 15px;
}

.y2-tb4 {
      grid-auto-flow: column;
      grid-column: span 2;
      grid-row: span 2;
      margin-bottom: 15px;
}

.unit b ~ p {
      padding-left: 12px;
      padding-right: 12px;
      align-content: center;
}
```

4.3. Trees exercise (responsive layout)

Isaac's Solution:

```
body {
    font-family: "Open Sans", sans-serif;
    margin: 0;
}

header {
    background-color: #04443c;
    color: white;
    margin: 0;
    padding-left: 10px;
    padding-right: 10px;
}

@media only screen and (max-width: 400px) {

.container {
      margin-left: 10px;
      margin-right: 10px;
      display: inline-grid;
```

```
      grid-template-columns: repeat(1, 1fr);
      grid-gap: 20px;
}

}

@media only screen and (min-width: 400px) {

.container {
      margin-left: 10px;
      margin-right: 10px;
      display: inline-grid;
      grid-template-columns: repeat(2, 1fr);
      grid-gap: 20px;
}

}

@media only screen and (min-width: 600px) {
      .container {
            margin-left: 10px;
            margin-right: 10px;
            display: inline-grid;
            grid-template-columns: repeat(4, 1fr);
            grid-gap: 20px;
      }
}

.featured {
      grid-column: span 2;
      padding-right: 15;
}

.card {
      margin: 0 auto;
    background-color: #167f5f;
    color: white;
    padding: 10px, 10px, 10px, 10px;
}

.card-image {
    max-width: 100%;
    height: auto;
```

```
}

.card span.latin-name {
    display: block;
    font-style: italic;
    padding: 10px;
    padding-bottom: 0;
}

.card span.common-name {
    display: block;
    padding: 10px;
    padding-top: 5px;
    font-weight: bold;
}
```

Week 8: Javascript

Note: this code requires you to replace YOUR_API_KEY with an API key. Otherwise it's a solution to the first part of the exercise.
Thanks to Sam for this solution.

```
const IMG_PATH = 'https://image.tmdb.org/t/p/w500'
const SEARCH_API =
`https://api.themoviedb.org/3/search/movie?api_key=YOUR_API_KEY&query="`
const API_URL_HOME =
'https://api.themoviedb.org/3/discover/movie?api_key=YOUR_API_KEY& \

include_adult=false&include_video=false&language=en-US&page=1&sort_by=popul
arity.desc';

const main =  document.getElementById("main");
const form = document.getElementById("form");
const search = document.getElementById("search");

async function getMovies(url) {

    const apiRes = await fetch(url);
    const resJson = await apiRes.json();
```

```
    if (!apiRes.ok) {
        console.error("Error fetching movies: ", apiRes.statusText);
    }
    showMovies(resJson.results);
}


function showMovies(movies) {
    main.innerHTML = ''

    movies.forEach((movie) => {
        const { title, poster_path, vote_average, overview } = movie

        const movieEl = document.createElement('div')
        movieEl.classList.add('movie')

        movieEl.innerHTML = `
            <img src="${IMG_PATH + poster_path}" alt="${title}">
            <div class="movie-info">
          <h3>${title}</h3>
          <span
class="${getClassByRate(vote_average)}">${vote_average}</span>
            </div>
            <div class="overview">
          <h3>Overview</h3>
          ${overview}
        </div>
        `
        main.appendChild(movieEl)
    })
}

function getClassByRate(vote) {
    if (vote >= 8) {
        return ".movie-info blue";
    } else if (vote >= 5) {
        return ".movie-info black"
    } else return ".movie-info orange";
}

// Get initial movies
getMovies(API_URL_HOME);
```

```
form.addEventListener('submit', (e) => {
    e.preventDefault()

    const searchTerm = search.value // we create a var with the search term

    if(searchTerm && searchTerm !== '') { // and if the term exists
        getMovies(SEARCH_API + searchTerm);

        search.value = ''
    } else {
        window.location.reload();
    }
})
```

Week 9: Web Scraping

7 Web Scraping

7.1. Crawling

1. Read man wget to understand what the -i --force-html and --spider options do. Download a copy of this webpage (the one you are currently reading) and use wget to test all the links on the page. Are there any broken links?

wget --spider -r -l 1 --force-html
https://cs-uob.github.io/COMSM0085/exercises/part2/scrape/crawl.html

-i file
--input-file=file
Read URLs from a local or external file. If **-** is specified as file, URLs are read from the standard input. (Use **./-** to read from a file literally named **-**.)
If this function is used, no URLs need be present on the command line. If there are URLs both on the command line and in an input file, those on the command lines will be the first ones to be retrieved. If **--force-html** is not specified, then file should consist of a series of URLs, one per line.
However, if you specify **--force-html**, the document will be regarded as **html**. In that case you may have problems with relative links, which you can solve either by adding "<base href="url">" to the documents or by specifying **--base=**url on the command line.

If the file is an external one, the document will be automatically treated as **html** if the Content-Type matches **text/html**. Furthermore, the file's location will be implicitly used as base href if none was specified.

--force-html

When input is read from a file, force it to be treated as an HTML file. This enables you to retrieve relative links from existing HTML files on your local disk, by adding "<base href="url">" to HTML, or using the **--base** command-line option.

--spider

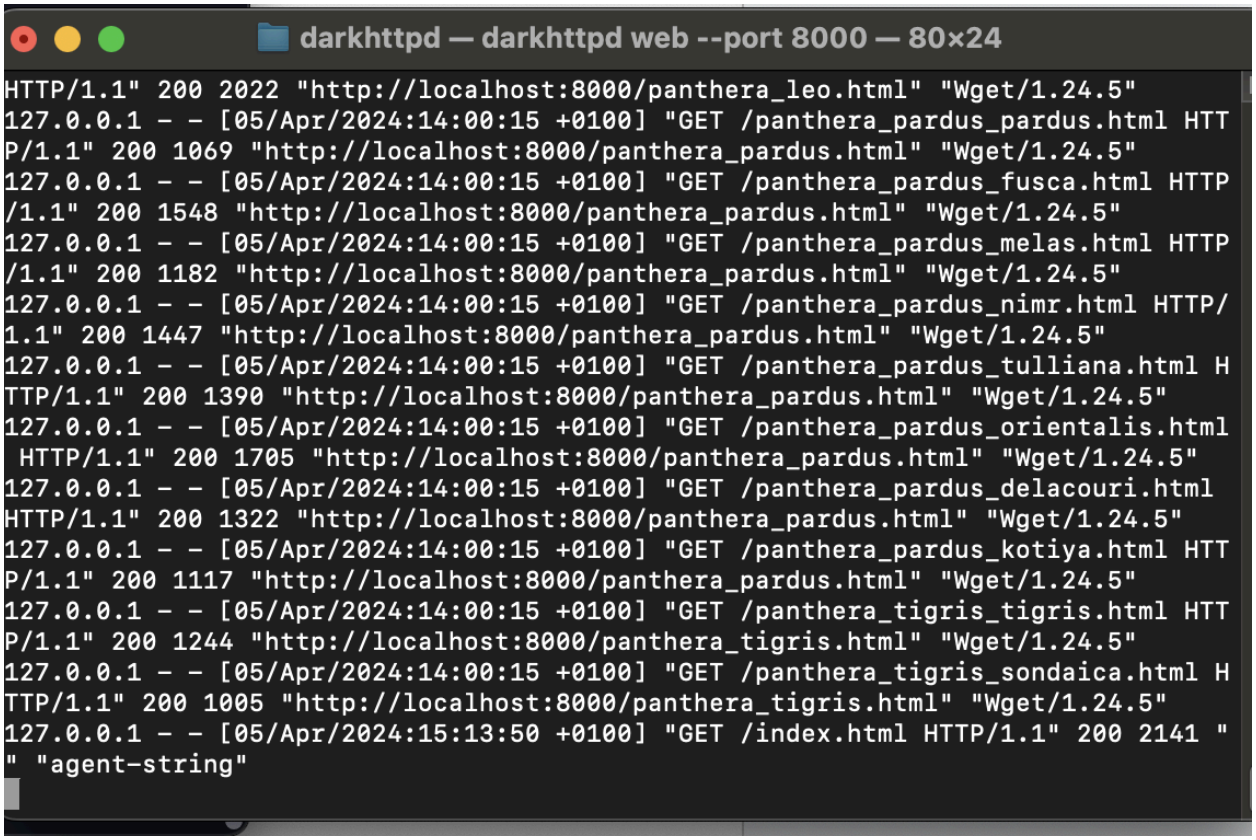
When invoked with this option, Wget will behave as a Web spider, which means that it will not download the pages, just check that they are there. For example, you can use Wget to check your bookmarks:

```
wget --spider --force-html -i bookmarks.html
```

This feature needs much more work for Wget to get close to the functionality of real web spiders

2. Tell wget to use a different user agent string in a request to your server running on localhost. Check what the request looks like to your server.

```
wget  --user-agent=agent-string localhost:8000/index.html
```



3. How would wget -r -l 1 http://example.com differ from wget -p http://example.com? (Hint: think about external resources).

wget -r -l 1 http://example.com tells wget to recursively download the website, but only to a depth of 1. This means wget will download the page at the given URL and any pages directly linked from it, but no further.

wget -p http://example.com instructs wget to download the webpage at the specified URL, along with all the files that are necessary to properly display the page, such as CSS, JavaScript, and images. This option is used for downloading a single page and its requirements for offline viewing.

The key difference is that -r -l 1 is for recursive download with a specified depth, which can include pages on the same domain, whereas -p is specifically for downloading all the components needed to render a single page properly, including external resources like stylesheets and images hosted on different domains.

4. Look for 'Recursive Accept/Reject options' in the wget manpage. How would you get wget to crawl pages from multiple different domains?

-H

--span-hosts

Enable spanning across hosts when doing recursive retrieving.

5. Look up what -nc does. What is clobbering, and why would or wouldn't you want to do it?

-nc or --no-clobber.

By default, wget repeatedly download with a new name (example.html.1 etc). With no clobber wget will refuse to download new versions of that file.

Clobbering is something being overwritten on repeated download.

-nc

--no-clobber

If a file is downloaded more than once in the same directory, Wget's behavior depends on a few options, including **-nc**. In certain cases, the local file will be clobbered, or overwritten, upon repeated download. In other cases it will be preserved.

When running Wget without **-N**, **-nc**, **-r**, or **-p**, downloading the same file in the same directory will result in the original copy of file being preserved and the second copy being named file**.1**. If that file is downloaded yet again, the third copy will be named file**.2**, and so on. (This is also the behavior with **-nd**, even if **-r** or **-p** are in effect.) When **-nc** is specified, this behavior is suppressed, and Wget will refuse to download newer copies of file. Therefore, ""no-clobber"" is actually a misnomer in this mode---it's not clobbering that's prevented (as the numeric suffixes were already preventing clobbering), but rather the multiple version saving that's prevented.

When running Wget with **-r** or **-p**, but without **-N**, **-nd**, or **-nc**, re-downloading a file will result in the new copy simply overwriting the old. Adding **-nc** will prevent this behavior, instead causing the original version to be preserved and any newer copies on the server to be ignored.

When running Wget with **-N**, with or without **-r** or **-p**, the decision as to whether or not to download a newer copy of a file depends on the local and remote timestamp and size of the file. **-nc** may not be specified at the same time as **-N**.

A combination with **-O**/**--output-document** is only accepted if the given output file does not exist.

Note that when **-nc** is specified, files with the suffixes **.html** or **.htm** will be loaded from the local disk and parsed as if they had been retrieved from the Web.

7.2. BeautifulSoup

Take a look at some other examples from the BeautifulSoup documentation in particular regarding the use of the .find_all() method.

Use your interpreter to access a list of all the elements in the webpage, and figure out how to print out just the text contained within them.

```
>>> for line in soup.find_all('strong'):
...     print(line.get_text())
```

How would you use .find_all() to find all elements with a particular class value (e.g., 'container')? Would your method work if the div had multiple classes?

```
soup.find_all('div', {'class': 'container'})
```

no, for multiple classes we should use something like
soup.select('div.container')

INTIALLY:

```
1 from bs4 import BeautifulSoup
2 import os
3
4
5 for file in os.listdir('cattax'):
6   if file[-4:] == 'html':
7     soup = BeautifulSoup(open('cattax/'+file,'r'), features='html.parser')
8     print(soup.title.text + " : " + soup.h1.text)
9
```

Modify scrape.py so that it also prints out the contents of the 'info' paragraph in each page (this can be a second print statement). Run the script again to test that it works.

```
1 from bs4 import BeautifulSoup
2 import os
3
4
5 for file in os.listdir('cattax'):
6   if file[-4:] == 'html':
7     soup = BeautifulSoup(open('cattax/'+file,'r'), features='html.parser')
8     print(soup.title.text + " : " + soup.h1.text)
9     info_paragraph = soup.find('p', class_='info')
```

```
    if info_paragraph:
        print(info_paragraph.text)

```

~

Currently, the script prints something for every page. Modify it so it would only print something out for the leaf nodes -- those pages that don't have a 'container' element of their own.

```
from bs4 import BeautifulSoup
import os


for file in os.listdir('cattax'):
  if file[-4:] == 'html':
    soup = BeautifulSoup(open('cattax/'+file,'r'), features='html.parser')
    if(soup.find('div', class_='container')):
            continue
    print(soup.title.text + " : " + soup.h1.text)
    info_paragraph = soup.find('p', class_='info')
    if info_paragraph:
        print(info_paragraph.text)

~
```

Printing things out can be useful, but often we want to store values we scrape for later programmatic work. Rather than printing out information, create and update a Python dict for all leaf nodes where the dictionary keys are the titles of a page and the values are the corresponding content of the 'info' box. Run your script with python3 -i scrape.py and it will execute your script and then place you in an interactive session immediately following your script's execution. You can then check that the dictionary's contents are what you expect by interacting with the dict object in your interpreter.~

```
from bs4 import BeautifulSoup
import os

d = dict()
for file in os.listdir('cattax'):
  if file[-4:] == 'html':
    soup = BeautifulSoup(open('cattax/'+file,'r'), features='html.parser')
    if(soup.find('div', class_='container')):
            continue
    info_paragraph = soup.find('p', class_='info')
    if info_paragraph:
```

```
12          d.update({soup.title.text:info_paragraph})
13
```

~

Week 10: Practical Encryption

8 Practical Encryption

8.1. OpenSSL

What is the difference between your unencrypted file and the encrypted one? Try and read an encrypted file.
The file size is larger, and the text is unreadable due to weird characters:

Salted__◆G◆◆9Cw◆C0◆◆3◆◆>?◆◆:&◆

-rw-r--r-- 1 vagrant vagrant 32 Apr 28 10:16 mytext.enc
-rw-r--r-- 1 vagrant vagrant 6 Apr 28 10:16 mytext.txt

Alternatively, you can encode the encrypted file with base64.
openssl base64 -in mytext.txt -out mytext.encxt.enc -a
(the -a flag stands for ASCII).

-rw-r--r-- 1 vagrant vagrant 32 Apr 28 10:16 mytext.enc
-rw-r--r-- 1 vagrant vagrant 9 Apr 28 10:20 mytextbase64.enc
-rw-r--r-- 1 vagrant vagrant 6 Apr 28 10:16 mytext.txt

This new file is only 9 bytes and looks like this:
aGVsbG8K

Is the file size larger than before? If so, why?
Because it has been encrypted - and contains more letters than before. (Hashed with salt.)

Utilize the -d parameter to decrypt the file, whether it's a binary stream or consists only of ASCII characters

```
openssl enc -base64 -d -in mytext.encxt.enc -out mytext.decoded
```

Using the openssl man pages, figure out how to use genrsa to generate a 1024-bit public-private key pair.

```
man openssl-genrsa

openssl genrsa -out private_key.pem  -numbits 1024
```

This generates a private key - the public key has to be extracted.

The public key is meant to be shared with others. Extract your public key with the OpenSSL tool into a .pem file using the -pubout parameter.

```
openssl rsa -in private_key.pem -pubout -out public_key.pem
```

Next, create an encrypted message using the OpenSSL -encrypt command.

```
openssl rsautl -encrypt -inkey bob_public.pem -pubin -in top_secret.txt
-out top_secret.enc
```

The crucial thing about this is that you can use a public key of someone else to do encryption of a message.

Then, we generate the RSA private key (keep note of the passphrase). Use OpenSSL with the genrsa command and the -aes256 parameter for a private key with size 2048.

```
openssl genrsa -aes256 -out private_key.pem
```

Next, generate a root certificate with OpenSSL using the req command and parameters -x509 for a digital signature and -sha256 for a hash function. Finally, make it valid for 30 days.

```
man openssl-req
openssl req -x509 -sha256 -days 30 -out rootCert.pem
```

technically we don’t need the 30d parameter because it’s the default.

The OpenSSL tool includes many utilities. We can use one of them to verify a secure connection to a server. To do so, we will use the s_client utility to establish client-to-server communication. Therefore, we can determine whether a port is open, if the server is configured to support SSL, and when the certificate expires.

```
man openssl-s_client
```

In the terminal, use s_client against a webserver with HTTPS.

```
openssl s_client -connect google.com:443
```

443 is the default ssl port
https://docs.pingidentity.com/r/en-us/solution-guides/htg_use_openssl_to_test_ssl_connectivity

What are the differences compared to other tools such as netcat or telnet?
The other tools aren't designed for ssl
Try running the command again with the -crlf and -brief flags. What is the difference?

-brief
Only provide a brief summary of connection parameter instead of the normal verbose output.

-crlf
This option translated a line feed from the terminal into CR+LF as required by some servers.

8.2. PGP

gpg --encrypt --sign --armor -r other_person@mail.com file_name
This command encrypts the message and signs it with your private key to ensure that the message originates from you. Keep in mind that this message will be encrypted using the recipient's public key, making it

unreadable to you (Unless you somehow obtain the recipient's private key!).

Tip: You can modify the command above to generate a message only you can read, using your private key. Think about how.

```
gpg --encrypt --sign --armor -r your_own_email@mail.com file_name
```

- Use the gpg command with the --search parameter to find out, for each key, the associated email address or User ID (username).

```
gpg --keyserver keyserver.ubuntu.com --search-key 413109AF27CBFBF9
```

- Use the gpg command with the --recv-keys parameter if you decide to import a key into your keyring.

```
gpg --keyserver keyserver.ubuntu.com --recv-keys ABCDEF0123456789
```

- Use the gpg command with the --fingerprint option to verify the authenticity of the keys.

```
gpg --keyserver keyserver.ubuntu.com --fingerprint ABCDEF0123456789
```

- Now, can you try searching based on the email addresses?

gpg --search bogwonch@bogwonch.net

- Can you find Joseph's old expired key?

```
gpg --keyserver keyserver.ubuntu.com --search bogwonch@bogwonch.net
```

Vagrantfile

updated with the stuff for the last lecture, otherwise the same.

```
Vagrant.configure("2") do |config|
  config.vm.box = "generic/debian12"
  config.vm.synced_folder ".", "/vagrant"
  #config.vm.network "forwarded_port", guest: 8080, host: 8080

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update -y
    apt-get install -y git git-man apt-file
    apt install -y gdb
```

```
    apt install -y maven
    apt install -y openjdk-17-jdk
    apt install -y unzip
    apt install -y maven
    apt install -y openjdk-17-jdk
    apt install -y mariadb-{server,client}
    apt-get install -y libreadline-dev
    apt install -y shellcheck
    apt install -y openssl
    apt install -y nginx
    systemctl start mariadb
    apt-get install gnupg
    systemctl enable mariadb
    mysqladmin -u root -p create mydatabase
    mysql -u root -p -e 'source /vagrant/sample-data.sql'

SHELL
config.vm.provision :shell, privileged: false, inline: <<-SHELL
    git config --global user.name "ali"
    git config --global user.email "jardine64@gmail.com"
    SHELL
  config.vm.provision "file", source: "~/.vimrc", destination: "~/.vimrc"
end
```