

# HTML Forms & Intro to CSS

Programming with  
Web Technologies



Auckland  
ICT Graduate School

# Last time

---

- HTML
  - Tables
  - HTML attributes
  - Anchors
  - Images & multimedia
- Git
  - Branching
  - Merging

# Today

---

- More HTML
  - Forms
- Intro to Cascading Style Sheets (CSS)
  - Styling elements, pages, websites
  - Colors & borders
  - Fonts
  - Basic CSS selectors

# HTML Forms



Auckland  
ICT Graduate School

# HTML forms

---

In short, HTML Forms allow input rather than just reading Web content

An HTML **form** is a section of the HTML document containing special elements — called **controls** — which can be used to obtain data from the user. This data can then be processed either in the browser or on the server

- *<http://www.w3.org/TR/html401/interact/forms.html>*

# Forms

---

We can use forms to allow users to input data which can be processed in the client, or sent to another site or application to process

- Login details (username + password)
- Registration details (name, contact info etc)
- Preferences for using the site
- Entering search queries
- Payment info for online purchases

There are a wide range of controls available to suit your needs

# Forms - Tags

Like with tables, forms have a number of tags and attributes associated with them. The first one we need to know about is:

`<form>` - Defines the start and end point of the form. Groups the form components

All controls and labels that make up the form need to be contained within the `<form>` tags

The `<form>` tag has two important, but optional attributes: `action` & `method`

More on these shortly

# Forms - Tags

`<input>` - An input element in a form, does not need a closing tag. Almost all form controls are inputs

- ☒ Male
- ☐ Female
- ☐ Other

Submit

First name:

Last name:

Enjoys pizza:



User password:

Birthdate:

November/2017

Mon	Tue	Wed	Thu	Fri	Sat	Sun
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3

All of the controls shown here are made using `<input>` tags

Made possible through gratuitous use of attributes, most importantly, the `type` attribute



# Forms - Text input

```
<input type="text">
```

A single-line text box

```
<input type="password">
```

A specialized text-box that masks input

```
<input type="tel">
```

A text-box for entering phone numbers

```
<input type="url">
```

A text-box for entering URLs

Many more specialized options exist. Useful to use these, as they give context to browsers and other clients

# Forms - Identifying and naming elements

If we are processing form information, then we need to know what values from each control means

Where did they come from, what do they represent

Use the `name=""` attribute to give a name to an input's value

```
<input type="text" name="firstname">
```

```
<input type="text" name="lastname">
```

Useful to software, but not so useful for the end-user. Would be useful to have some sort of `<label>` for these

# Forms - Labels

`<label>` - Provides a non-editable textual label for other controls

Labels are associated with other controls through use of the `for` attribute

The value of a `label`'s `for` attribute should match the value of another element's `id` attribute

All html elements can have an `id` attribute, they must be unique

```
<label for="fnameID">First Name</label>
```

First Name

```
<input type="text" id="fnameID" name="firstname">
```

# Forms - A basic form

```
<form>
  <label for="fnameID">First Name</label>
  <input type="text" id="fnameID" name="firstname">

  <label for="lnameID">Last Name</label>
  <input type="text" id="lnameID" name="lastname">
</form>
```

First Name

Last Name

# Forms - Grouping controls

Often a form will be made up of a number of sections, that contain related controls. We can group our controls into sections with a fieldset

`<fieldset>` - Groups form controls with a visual border

`<legend>` - A short textual label for a fieldset. Should be the first element inside a fieldset

```
<fieldset>  
  <legend>An example</legend>  
  Controls here  
</fieldset>
```



# Forms - Checkboxes

Checkboxes normally appear in a logical grouping in which 0 or more items can be selected. They are created using `<input>` controls with the `type` attribute set to `checkbox`

0 or more items can be selected by default, indicated by the presence of the `checked` attribute

The value sent to the processing application for a checked box can be controlled with the `value` attribute

Checkboxes can be grouped together by setting the `name` attributes of related inputs to the same value

# Forms - Checkboxes

```
<label for="ffCheckbox"> FireFox</label>
<input type="checkbox" name="browser_choice" value="firefox" id="ffCheckbox">
<label for="cCheckbox"> Chrome</label>
<input type="checkbox" name="browser_choice" value="chrome" id="cCheckbox">
<label for="sCheckbox"> Safari</label>
<input type="checkbox" name="browser_choice" value="safari" id="sCheckbox">
<label for="ieCheckbox"> Internet Explorer</label>
<input type="checkbox" name="browser_choice" value="ie" id="ieCheckbox">
```

FireFox ☐ Chrome ☐ Safari ☐ Internet Explorer ☐

# Forms - Radio buttons

Radiobuttons normally appear in a logical grouping in which only 1 item can be selected. They are created using `<input>` controls with the `type` attribute set to `radio`

Normally 1 item is selected by default, indicated by the presence of the `checked` attribute

The value sent to the processing application for a checked box can be controlled with the `value` attribute

Related radiobuttons should be grouped together by setting the `name` attributes of related radiobuttons to the same value



# Forms - Radio buttons

```
<label for="maleRadioID">Male</label>  
<input type="radio" checked id="maleRadioID" value="male" name="Gender">  
<label for="femaleRadioID">Female</label>  
<input type="radio" id="femaleRadioID" value="female" name="Gender">
```

Male ☒ Female ☐

# Forms - Select

A select, also known as a drop-down list, provides a list of items for the user to choose from

`<select>` - Groups options together to form a drop-down list. Should have a `name` attribute set to identify the selected option for the processing application

`<option>` - An option that exists within a `<select>`. The value sent to the processing application for a selected item can be controlled with the `value` attribute. The default option can be set with the `selected` attribute

# Forms - Select

```
<label for="countryID">Country</label>
<select id="countryID" name="countryName">
  <option value="nz" selected>New Zealand</option>
  <option value="aus">Australia</option>
  <option value="in">India</option>
  <option value="us">United States</option>
  <option value="other">other</option>
</select>
```



# Forms - More text input

---

The `textarea` control allows for multiline textual input by the user. By default, the textarea is shown in a monospaced (fixed width) font, and needs to be told via the `rows` and `cols` attributes how many rows and columns worth of text to show without scrollbars

If `rows` or `cols` limits are exceeded, the browser is responsible for wrapping text or adding scroll bars

Can optionally provide initial text as the content of the control

# Forms - More text input

```
<label for="profile_text"> About Text</label>
<textarea id="profile_text"
          name="profile_text_area"
          rows="4"
          cols="40">
    enter text here
</textarea>
```

About Text

enter text here

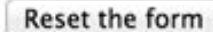
# Forms - Reset

The `reset` control is an input element with the `type` attribute set to `reset`, and is presented by the browser as a button

When clicked, all form controls are reset to their initial values. The browser handles this, we do not need to do this ourselves

By default, the button will appear with the text "reset", but if an alternative label is desired, it can be provided via the `value` attribute

```
<input type="reset" value="Reset the form">
```

A rectangular button with a light gray border and a subtle shadow, containing the text "Reset the form" in a dark gray sans-serif font.

# Forms - Submit

The `submit` control is an input element with the `type` attribute set to `submit`, and is presented by the browser as a button

When clicked, the values selected/provided in form controls are sent to the processing application

By default, the button will appear with the text "submit", but if an alternative label is desired, it can be provided via the `value` attribute

```
<input type="submit" value="Submit for processing">
```

A small, light gray button with rounded corners and a thin border. It contains the text "Submit for processing" in a dark gray, sans-serif font.

# Forms - Submitting

---

When the user clicks the 'Submit' button we want to have the form data processed in some way. This is normally done by a piece of program code e.g., JavaScript code executed in the browser, or alternatively by server-side code (e.g. a Servlet or PHP script) executed on a server

We might call this piece of code a form processing agent – and we need to identify the agent that the form data should be sent to

We identify the agent in the `action` attribute of the `<form>` element



# Forms - Submitting

The value of the action attribute is usually an absolute or relative URL of a server side script

```
<form action="https://trex-sandwich.com/echo/">  
    <!-- form content here -->  
</form>
```

```
<form action="../processing_script.jsp">  
    <!-- form content here -->  
</form>
```

# Forms - Methods

---

Along with the `action` attribute a `form` element has a `method` attribute which defines the HTTP method (verb) to use when communicating with the server

For forms, the verb is normally **GET** or **POST**

The method chosen affects the HTTP message header and determines the way in which the data will be transported to the server

The HTTP message header is a bit out of scope for now, but the way in which the data is transported is relevant to us

# GET vs POST

The browser is responsible for sending data from a form to the destination specified in the action attribute

If the **GET** method is used, `name=value` pairs are appended to the action attribute address, starting with a "?", separated by "&"s

`https://ex.com/form.jsp?fname=Peter&lname=Jackson&gend=male`

If the **POST** method is used the data is sent as the body of a message

Not visible in the browser/history. Useful for passwords!

# Intro to CSS

# CSS (Cascading Style Sheets)

---

CSS is a language for describing presentational characteristics of elements in a document (commonly an HTML web page)

The idea is to separate description of document content/structure from description of layout/style. HTML code should define content and structure, while CSS code should define layout and style

A file containing CSS code is commonly referred to as a **stylesheet**

# CSS

---

## 3 main ways of using CSS:

- Inline
  - Using a style attribute
- Internal
  - In the head of the document
  - Using class attribute and element selectors
- External
  - In a separate CSS file
  - Using class attribute and element selectors

# Inline CSS

Uses the HTML **attribute** *style*. Contains semicolon separated *property:value* pairs within double quotes

```
<tag style="property: value [; property: value]*"></tag>
```

For example:

```
<p style="font-family: sans-serif; ">text</p>
```

```
<p style="font-family: sans-serif; font-size: 12pt">text</p>
```

# Internal CSS

Uses the HTML **tag** `<style>`, which appears in the `<head>` of the document. A selector is inserted that determines what elements will be affected, then a block of `property:value` pairs is placed inside braces

```
<style type="text/css">
  selector {
    property: value; [property: value;]*
  }
  ...
</style>
```



# External CSS

Uses the HTML **tag** `<link>`, which appears in the `<head>` of the document. This links to an external file that only contains CSS and includes it in your document

```
<link href="mystyle.css" rel="stylesheet" type="text/css">
```

The `href` can be any valid local, relative or external URL. When we are linking to a CSS file, the `relation` will always be `stylesheet`, and the `type` will always be `text/css`

# All together

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <link href="mystyle.css" rel="stylesheet" type="text/css">
  <style type="text/css">
    #custom { color: green; border: 2px solid black; }
    a:visited { color: blue; }
  </style>
<body>
  <p style="font-family: serif">Some content here</p>
  <a href="another.html">Some cool link</a>
  <div id="custom">
    <div id="external_target"></div>
  </div>
</body>
</html>
```

# Benefits of each approach

---

## Inline

- Quick changes, useful for simple once-off styles
- No extra files or tags

## Internal

- Removes duplication within pages
- Allows for styles that apply to multiple elements

## External

- Removes duplication across pages
- Allows for caching, stylesheet only needs to be downloaded once

# Benefits of each approach

---

## Inline

- Quick changes, useful for simple once-off styles
- No extra files or tags

## Internal

- Removes duplication within pages
- Allows for styles that apply to multiple elements

Generally, it is considered “best practice” to use external CSS where possible, only using other forms for very specific purposes or for prototyping.

## External

- **Removes duplication across pages**
- **Allows for caching, stylesheet only needs to be downloaded once**

# Basic styling

---

As with tags in HTML, there are a huge number of CSS properties, and it would not be practical to cover them all

There are however, a number of common properties that you will use regularly that we will look at

- Colors
- Borders
- Fonts

# Colors, Borders & Fonts



Auckland  
ICT Graduate School

# CSS colors

Colors can be represented in a number of ways using CSS

**Named colors** use [common color names](#) such as black, red, goldenrod, orangered, etc to represent colors. There are 140 of these that are supported

**RGB Hexadecimal** allows you to specify Red, Green and Blue 'channels' as Hexadecimal numbers, prefixed by a #. #DAA520 = Goldenrod

RGB or HSL **color functions** allow colors to be calculated within RGB or HSL spaces. `hsl(296, 59%, 28%)`, `rgba(120, 55, 60, 0.3)`

# Using colors

Colors can be used in a number of different places, but the most common uses are setting the foreground and background colors of an element

The `color` property sets the foreground (text) color

```
color: goldenrod;
```

```
color: rgb(255, 255, 60);
```

The `background-color` property sets the background color

```
background-color: #487;
```

```
background-color: hsl(122, 75%, 25%)
```



# Borders

Borders surround an element on all 4 sides, and can be set in a variety of colors, styles and thicknesses. These properties can be set individually, or at the same time; for all 4 sides at once, or for one side at a time

`border-style` is the most important border property, as by default it is set to `none`, so no other border changes will show up

```
border-style: solid;  
border-left-style: dashed;  
border-bottom-style: inset;
```

# Borders

---

`border-color` is used for setting the color of the border. You can use any of the color representations shown earlier

```
border-color: green;  
border-right-color: #5F6
```

`border-width` is used for setting the width of the border

```
border-width: 5px;  
border-top-width: 17px;
```

# Borders

If you were to set all of your borders at the same time using the `border-*` properties, you would need 3 lines of CSS. If you were to set each side individually with the `border-direction-*` properties, you would need 12 lines

The `border` property allows us to set the width, style and color of all borders, or a single border, in a 1 line

```
border: 15px solid red;  
border-left: 2px inset green;
```

# Fonts

Fonts can have a number of aspects of their appearances adjusted with CSS, including the size, weight and style of the font, as well as the font face itself

The five properties used to control fonts are:

- `font-family` – The “font” to be used
- `font-size` – The size of the font
- `font-style` – Settings like italics
- `font-weight` – Settings like bold
- `font-variant` – Settings like drop-caps

# Font family

The `font-family` property is used to select the font face. There are 5 basic options that are provided by browsers, but you can also specify fonts that may be installed on the client system, or that are defined as web fonts

The 5 basic options are shown here



The `font-family` property can specify several comma separated fonts, known as a **font stack**, that is evaluated left to right until a suitable font that the client has is found

```
font-family: Times new roman, Georgia, serif;
```

# Other font options

```
font-size: [medium | xx-small | x-small | small | large |  
x-large | xx-large | smaller | larger | number];
```

```
font-style: [normal | italic | oblique];
```

```
font-weight: [normal | bold | bolder | lighter | number];
```

```
font-variant: [normal | small-caps];
```

# Web fonts

- Recall the 5 basic fonts:   
The image shows five examples of the letters 'wi' in different font styles: serif, sans-serif, monospace, cursive, and fantasy. Each example is labeled with its respective font style name below it.
- For modern websites these are usually not enough - websites need their own look & feel.
- We can use CSS to easily import other font styles.

# Web fonts

The `@font-face` rule allows for a new font to be defined. This rule will contain a `font-family` property that gives a name to the new font, and a `src` property that provides a link to where the font file can be found

```
@font-face {  
    font-family: 'Fontasia';  
    src: url('/path/to/Fontasia.ttf');  
}
```

Once declared with font-face, you can use your new font in a font stack using the name provided in the `font-family` property (Fontasia in this case)



# CSS Selectors



Auckland  
ICT Graduate School

# CSS selectors

When defining CSS rule in an internal or external stylesheet, we need some way of indicating which attributes should be styled. We can do this using a **selector**. CSS3 supports many different types of selector, a few of which we will look at today.

When writing rule in a stylesheet, we write them like this

```
selector {  
    property: value;  
    ...  
}
```

# Type selector

A type selector matches all elements of a named HTML tag. The selector consists simply of the name of the HTML tag to which it should apply

```
h1 { color: blue; }
```

All `h1` elements will have blue text

```
td { font-weight: bold; }
```

Text inside all `td` elements will be bold

This is useful for setting default properties for tags, as it will apply to all tags of that type

# ID and class attributes

Tags in HTML can contain many different attributes, a number of which we have seen so far in this course. There are 2 attributes that nearly all tags support: `id` and `class`

Recall from previous lectures, `id` defined a unique identifier that could be used as a target for anchor bookmarks. The uniqueness of `id` is important and can be used for more than just anchors

The `class` attribute is similar, but does not need to be unique, and can contain more than one value separated by spaces

# Class selectors

A `class` selector matches all HTML tags that have a value in their `class` attribute that matches that specified in the selector. To indicate we are referring to a class, we prefix the class name with a period

```
.correct { background-color: green; }
```

All elements with the class "correct" will have a green background

```
.important { border-style: solid; }
```

All elements that have "important" in their class attributes will have solid borders

# ID selectors

---

An **id** selector matches the HTML element that has the same value in its **id** attribute as in the selector. To indicate we are referring to an id, we prefix the id name with a hash

```
#reset { font-family: serif; }
```

The element with the id "reset" will use a serified font for any text

# References

---

- [MDN Color](#)
- [MDN Text Styling](#)
- [MDN CSS Selectors](#)
- [MDN Div](#)
- [MDN Span](#)