II. Front-End Technologies

Structuring and design of web content

1 Basic Technologies

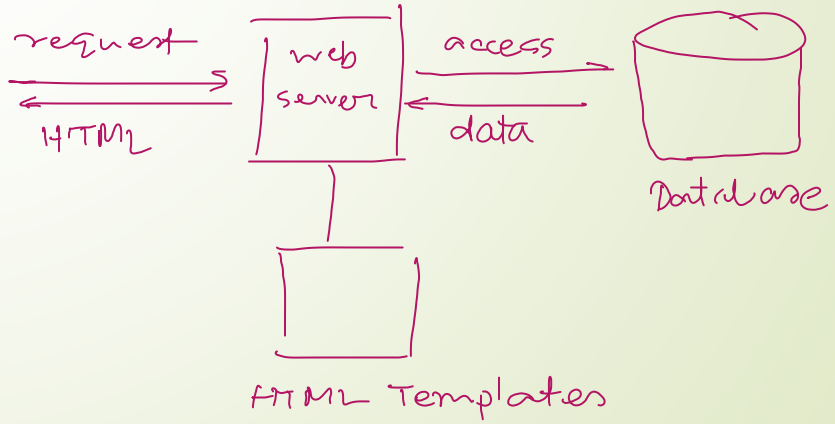
- Hypertext Markup Language
- Cascading Style Sheets

2 Interactive Web

- JavaScript
- Critical Rendering Path
- Json & Ajax
- jQuery
- React

1 Basic Technologies

HTML, CSS and their enhancements

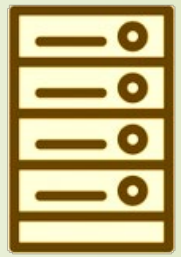


II.1

Fundamental Concepts

Main areas in web development

- front end
  - focused on creating interfaces, i.e. the collection of elements on the web page that the user is able to see and interact with
- back end
  - opposite of front end, i.e. performing processing of information



Front End

- Markup and web languages such as HTML, CSS and Javascript
- Asynchronous requests and Ajax
- Specialized web editing software
- Image editing
- Accessibility
- Cross-browser issues
- Search engine optimisation

Back End

- Programming and scripting such as Python, Ruby and/or Perl
- Server architecture
- Database administration
- Scalability
- Security
- Data transformation
- Backup

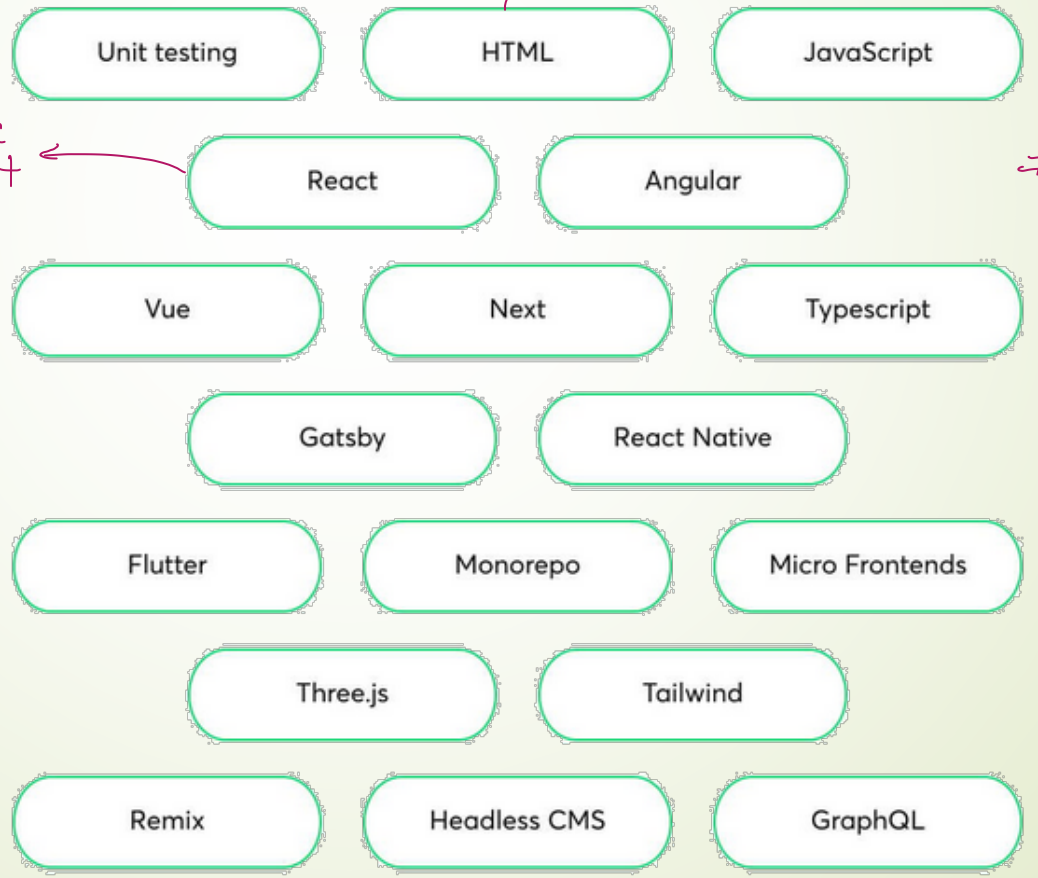
II.1

Fundamental Concepts

CSS – Corporate identity

Java script – manipulating DOM implement user interaction in the client

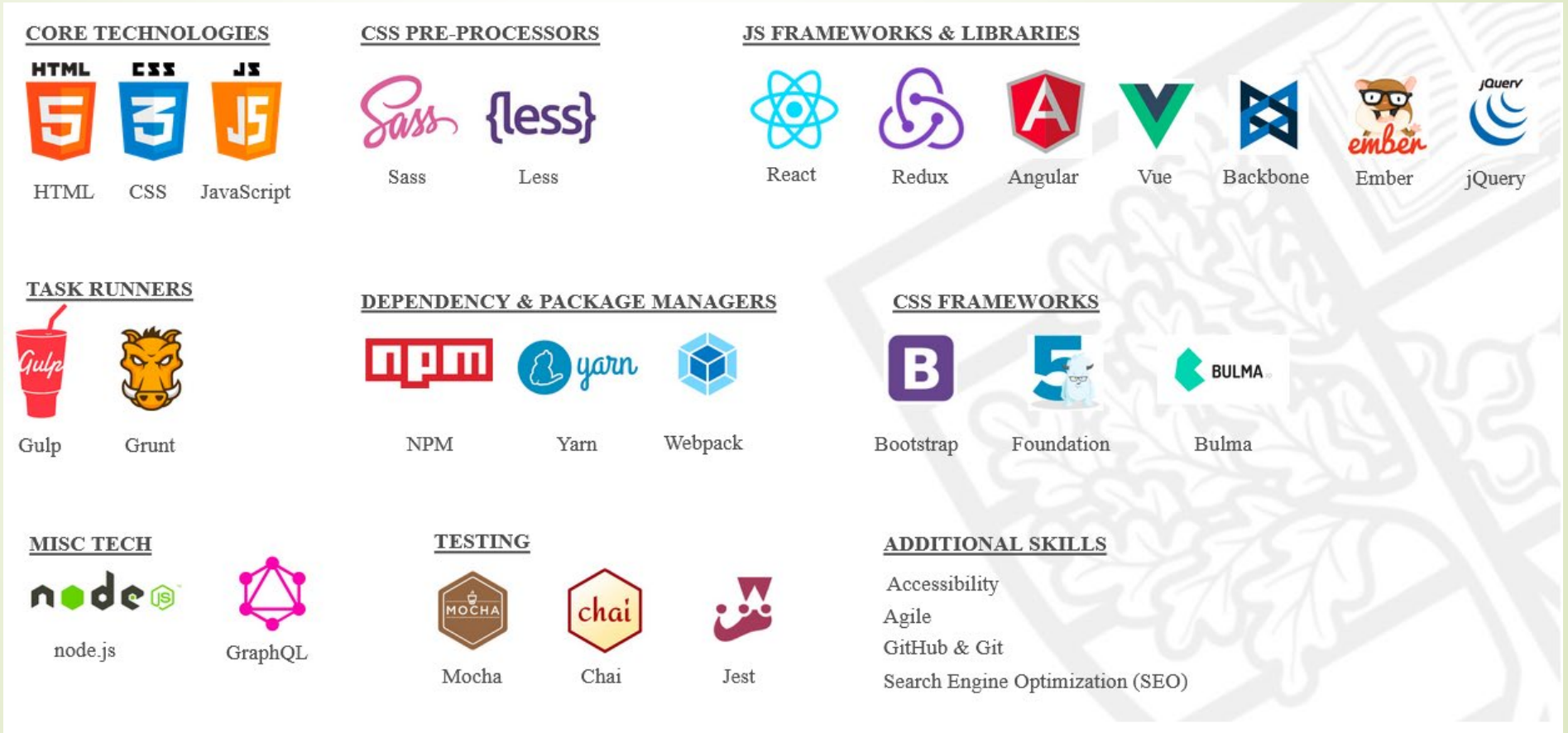
Top Front End Technologies



DOM – Document Object Model

Fundamental Concepts

Top Front End Technologies



II.1

Hypertext Markup Language

HTML

- most widely used language to write web pages
- hypertext refers to the way in which Web pages are linked together
- describes the content and structure of information on a web page
  - not the same as the presentation (appearance on screen)
- surrounds text content with opening and closing tags
- each tag's name is called an element
  - syntax: <element> content </element>
  - example: This is a paragraph
- stricter, more standard version XHTML

II.1

Hypertext Markup Language

XHTML

- EXtensible HyperText Markup Language
- a stricter, more XML-based version of HTML
- developed to make HTML more extensible and flexible to work with other data formats (such as XML)
- important differences:
  - elements must always be properly nested
  - elements must always be closed
  - elements must always be in lowercase
  - attribute names must always be in lowercase
  - attribute values must always be quoted

Hypertext Markup Language

XHTML – Structure of a page

- DOCTYPE
  - used to declare the DTD (Document Type Definition)
- xmlns attribute
  - specifies the xml namespace
  - provide a method to avoid element name conflicts

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Title of document</title>
</head>
<body>

  some content here...

</body>
</html>
```

II.1

Hypertext Markup Language

XHTML – Structure of a page

- head
  - contains machine-readable information (metadata) about the document, like its title, scripts, and style sheets
- body
  - contains the page's content

Hypertext Markup Language

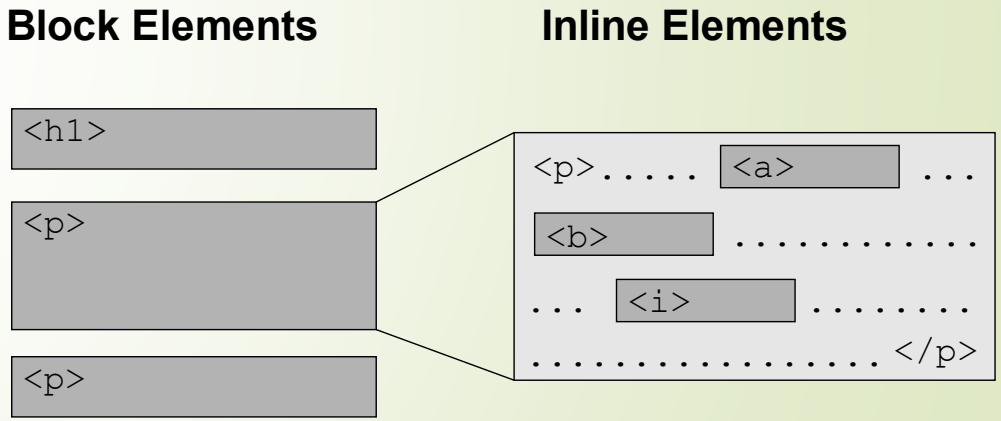
XHTML – Inside head

- meta-tag
  used to provide such additional information
  - Name
    Name for the property. Can be anything. Examples include, keywords, description, author, revised, generator etc.
  - content
    Specifies the property's value
- style- resp. link-tag
  used to specify embedded CSS resp. external CSS
- script-tag
  used to load JavaScript code

Hypertext Markup Language

XHTML – Block and inline elements

- block elements
  meant to structure the main parts
  of your page, by dividing your content
  in coherent blocks
  start with a new line and capture
  the available full width
- Inline elements
  meant to differentiate part of a text,
  to give it a particular function or
  meaning
  Inline elements usually comprise a single or few words



Hypertext Markup Language

XHTML – Basic Tags

- heading tags <h1>...</h1>
- paragraph tag ...
- line break tag
- centering content ...
- preserve formatting <pre> ... </pre>
- computer <code> ... </code>
- list tags <ul>, <ol>,

Hypertext Markup Language

XHTML – Basic Tags

- a-element
  specifies a link

```
<a href="http://www.thi.de" target="_blank">BGU</a>
```

- types of links
  - links from one website to another website
  - links from one page to another page on the same website
  - links from one part of a web page to another part of the same page
  - other types of links, such as those that start up your email program and compose a new email to someone

Hypertext Markup Language

XHTML – Basic Tags

- `href`-attribute can be:
  - an absolute URL which points to another website, such as `href="http://www.thi.de"`
  - a relative URL which points to a file within a website
  - a link to an element with a specified id within the current web page

    ```
    href="#top"
    ```

  - a combination of a URL and location within the page

    ```
    href="index.html#top"
    ```

Hypertext Markup Language

XHTML – Grouping

- `div`-element
  - allows to group a set of elements together in one block-level box
  - used to indicate a logical section or area of a page
  - use case of the `div`-element: to create an empty container
- `span`-element
  - acts like an inline equivalent of the `div`-element
    - contain a section of text where there is no other suitable element to differentiate it from its surrounding text
    - contain a number of inline elements
  - can control the appearance of the content of inner elements, using CSS

Hypertext Markup Language

XHTML – Attributes

- defines the characteristics of an HTML element and is placed inside the element's opening tag
- are made up of two parts: a name and a value

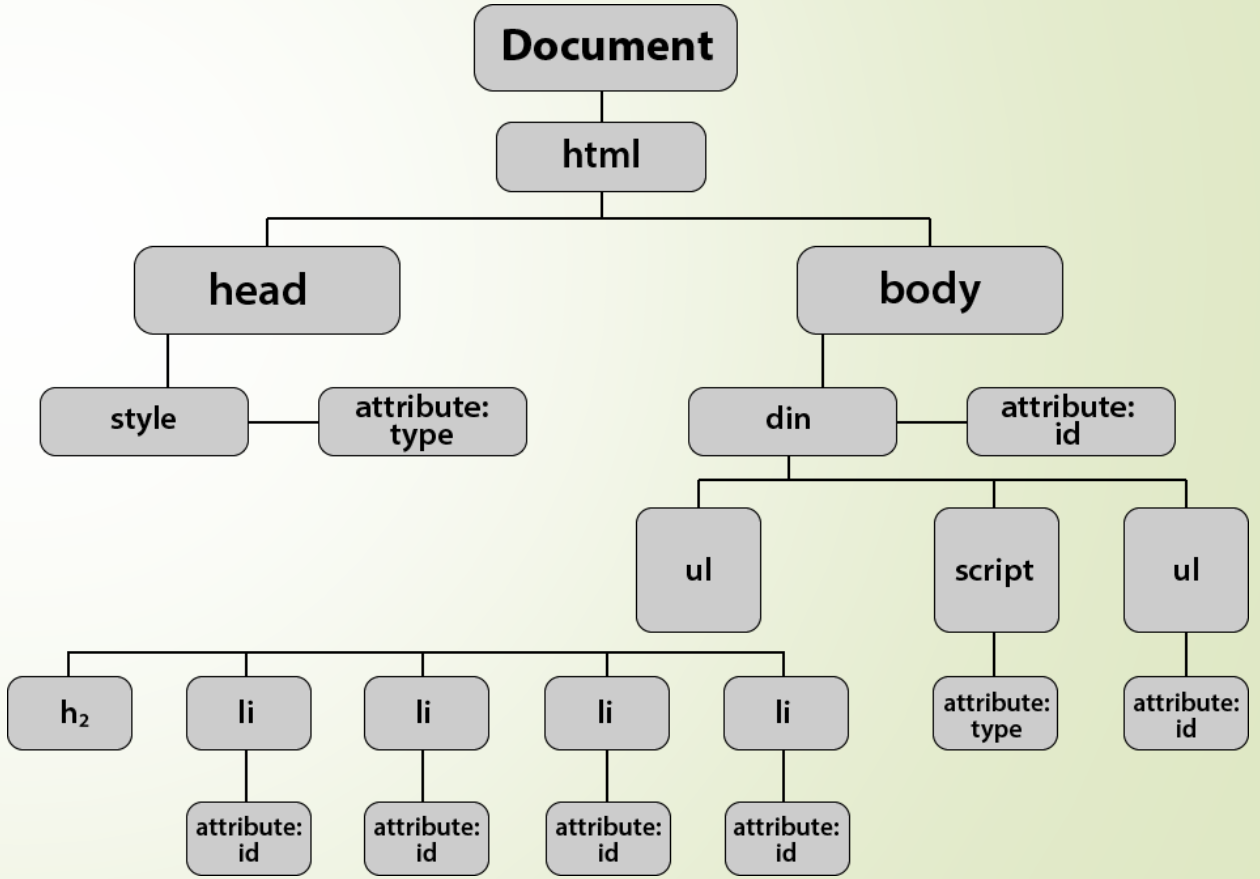
```
<p disabled="disabled"> ... </p>
```

- core attributes
  - id — unique identifier
  - title — the element's title
  - class — specifies the style sheet category
  - style — specifies Cascading Style Sheet (CSS) rules within the element

Parsing HTML

DOM - Document Object Model

- standard operations
  for traversing and modifying
  document elements arranged
  in a hierarchy of objects



Parsing HTML

xml.dom — The DOM API

- Objects

Interface	Section	Purpose
DOMImplementation	*DOMImplementation Objects*	Interface to the underlying implementation.
Node	*Node Objects*	Base interface for most objects in a document.
NodeList	*NodeList Objects*	Interface for a sequence of nodes.
DocumentType	*DocumentType Objects*	Information about the declarations needed to process a document.
Document	*Document Objects*	Object which represents an entire document.
Element	*Element Objects*	Element nodes in the document hierarchy.
Attr	*Attr Objects*	Attribute value nodes on element nodes.
Comment	*Comment Objects*	Representation of comments in the source document.
Text	*Text and CDATASection Objects*	Nodes containing textual content from the document.
ProcessingInstruction	*ProcessingInstruction Objects*	Processing instruction representation.

II.1

Parsing HTML

xml.dom — The DOM API

- Methods (link to Python Standard Library)
  - traversing the tree
  - manipulating the tree
  - finding an element

II.1

HTML Forms

Formulas

- forms contain special control elements like input text box, check boxes, radio-buttons and submit buttons
- `<form>` tag is used to defines an HTML form

Attributes	Description	Syntax & Example
action	defines URL of the program or page where action will be performed on form data.	`action="page2.php"`
method	specify the HTTP method to send form data.	`method="get"`
target	specify the target window or frame	`target="_blank"`
enctype	specify the encoding of form data (for media files)	`enctype="multipart/form-data"`

II.1

HTML Forms

Formulas

- form elements are defined inside the <form>-tag
  - <input>-tag
    - type attribute specifies different input elements
  - button
    - simple button
    - submit button
    - reset button

Attribute	Description	Syntax & Example
Text Box	used to define text box that allow user to enter some text.	input type="text"
Radio Button	defines radio button that allow users to select any one option or choice	input type="radio"
Checkbox	defines checkbox that allow users to select multiple option or choices	input type="checkbox"
Button	defines normal buttons for users for some action	input type="button"
Reset Button	defines button that reset form data when user click on it.	input type="submit"
Submit Button	defines Form submission button that submit form data when user clicks on it	input type="reset"

II.1

Realizing a Website

Design Steps

- Plan
  - collection of all possible information (requirements) regarding the website
  - define goals and objectives in order to achieve the objectives
- Design
  - determine the structure of the web site
- Develop
  - write all the code that brings the content to life and makes the site a reality
- Validate
- Release

II.1

Realizing a Website

Design Steps

- How to structure of a Web Site
  - when planning web pages, first list all the information (as headings) to put on them
  - bring all the headings that belong together into related groups: each group will require a separate page, or group of related pages, which will be individual parts of your web site
  - if there are to be a lot of web pages, it is advisable to have each group of pages kept in a separate folder
  - remember that any individual pages, which contain a lot of data, should be broken down into smaller pages, linked together
- Often
  - need to organize information, spreading it over several pages that are organized and linked via an initial "home/index page".

Realizing a Website

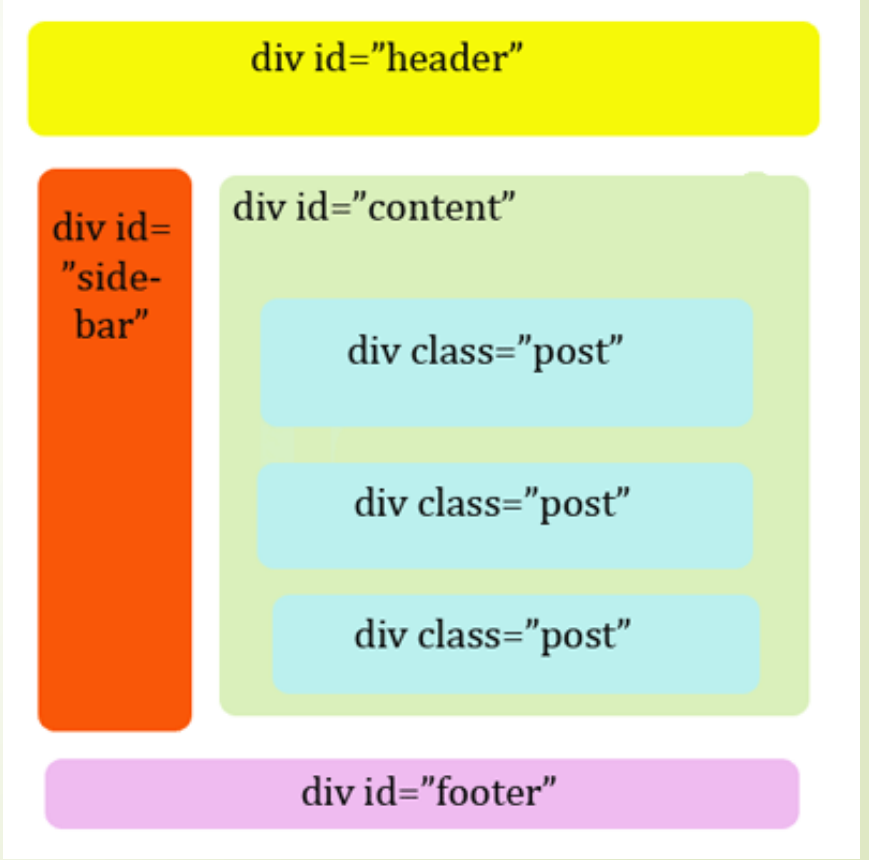
Design Steps - A typical page structure

- Only very few sites that don't at least loosely follow this pattern:
  - header at the top of the page, usually containing the top level heading of the page, and/or a company logo
  - main content column, which holds the main content of functionality
  - one or more sidebars, holding peripheral content that is either related to the page's main content and changes as new pages are loaded, or is always relevant and persists across the whole site
  - navigation menu going across or down the page. This is often put in a sidebar, or may form part of the header.
  - a footer that goes across the bottom of the page and contains secondary information such as copyright information and contact details

Realizing a Website

Design Steps - A typical page structure

- structuring mechanisms
  - `<div>` is a block container
  - `<span>` is an inline container
- characteristics
  - have no inherent content or dimensions, only that of their contents, until they are styled via CSS or manipulated by JavaScript
  - they also have no semantics, and the only way you can identify them is by giving them a class or id attribute



II.1

Realizing a Website

Intro to semantic HTML

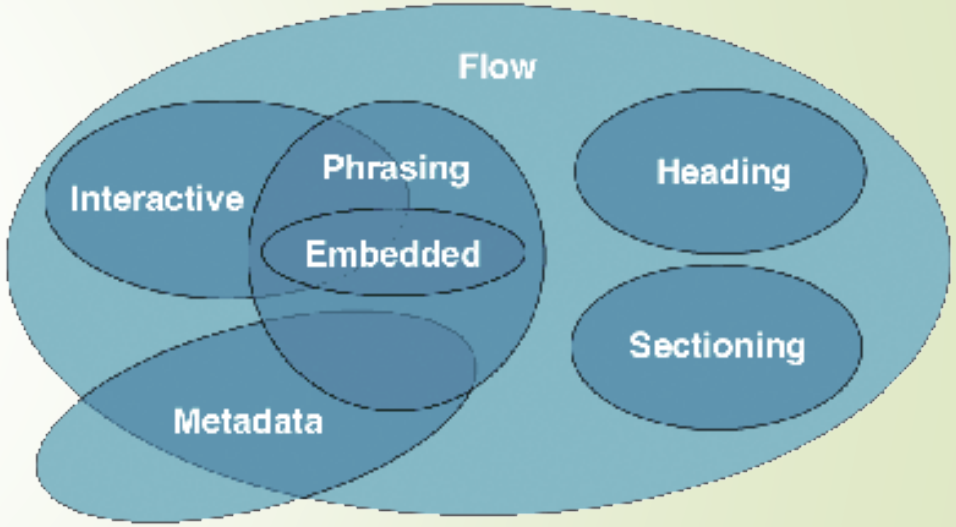
- disadvantages of div's
  - accessibility
  - readability
  - consistency and standards

- Now:
  - HTML5 was introducing a standardized set of semantic elements
  - a semantic element clearly describes its meaning to both the browser and the developer

HTML5

HTML5 – Content model

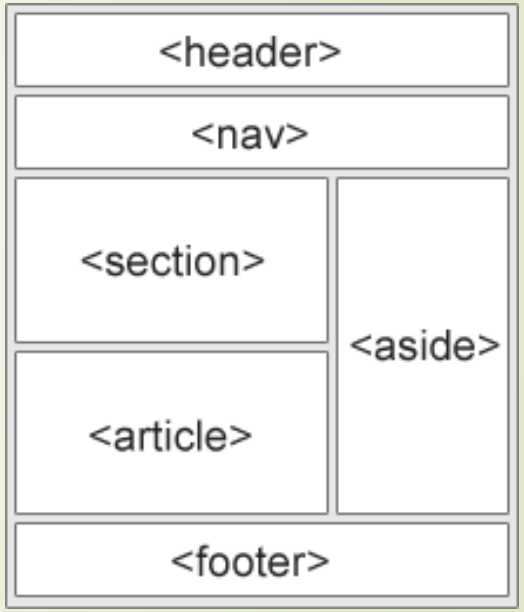
- Metadata content, e.g. link, script
- Flow content, e.g. span, div, text
- Sectioning content, e.g. aside, section
- Heading content, e.g. h1
- Phrasing content, e.g. span, img, text
- Embedded content, e.g. img, iframe, svg
- Interactive content, e.g. a, button, label
  Interactive content is not allowed to be nested



HTML5

HTML5 – Semantic elements

- `<header>` defines a header for the document or a section
- `<footer>` defines a footer for the document or a section
- `<nav>` defines navigation links in the document
- `<main>` defines the main content of a document
- `<section>` defines a section in the document
  the spec defines this as "a thematic grouping of content, typically with a heading
- `<article>` defines an article in the document
- `<aside>` defines content aside from the page content



HTML5

HTML5 – Form improvements

- Web Forms 2.0
  - extension to the forms features
  - form elements and attributes in HTML5 provide a greater degree of semantic mark-up

Type	Description
datetime	A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601 with the time zone set to UTC.
datetime-local	A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601, with no time zone information.
date	A date (year, month, day) encoded according to ISO 8601.
month	A date consisting of a year and a month encoded according to ISO 8601.
week	A date consisting of a year and a week number encoded according to ISO 8601.
time	A time (hour, minute, seconds, fractional seconds) encoded according to ISO 8601.
number	This accepts only numerical value. The step attribute specifies the precision, defaulting to 1.
range	The range type is used for input fields that should contain a value from a range of numbers.
email	This accepts only email value. This type is used for input fields that should contain an e-mail address. If you try to submit a simple text, it forces to enter only email address in email@example.com format.
url	This accepts only URL value. This type is used for input fields that should contain a URL address. If you try to submit a simple text, it forces to enter only URL address either in http://www.example.com format or in http://example.com format.

HTML5

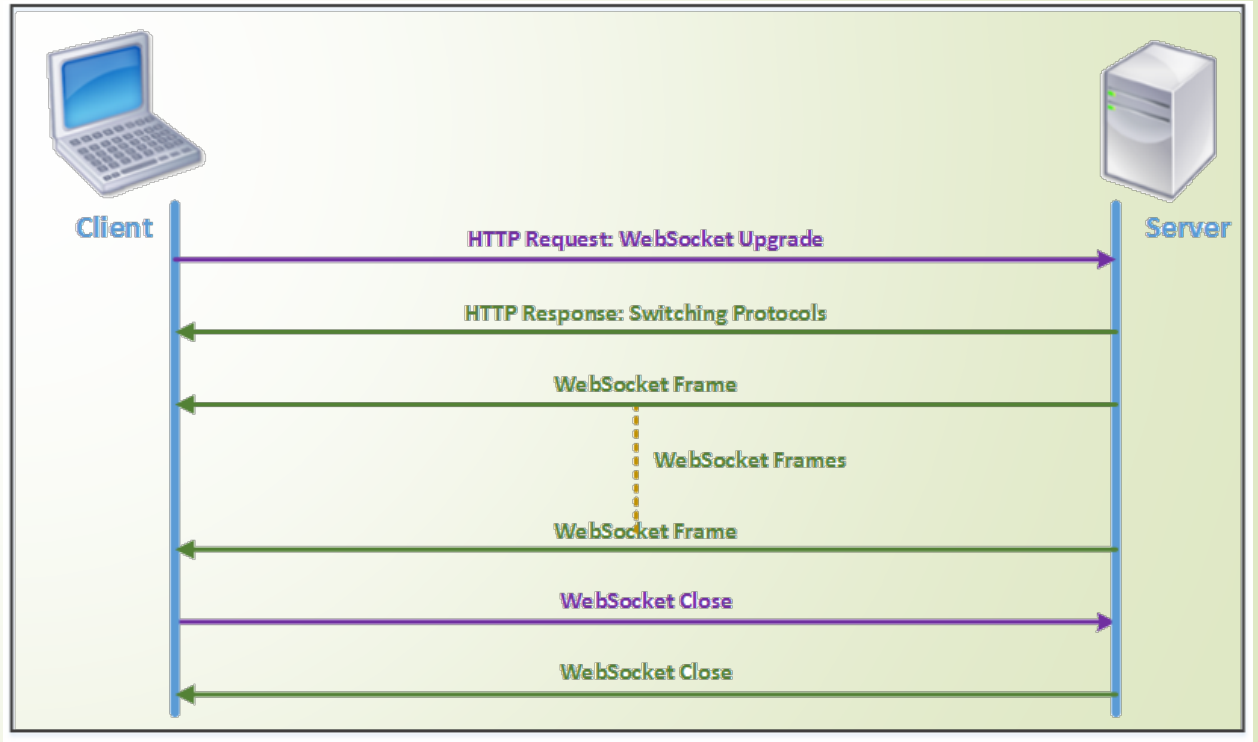
HTML5 – Websocket

- allows the browser to create a socket client (inside a web page)
- defines a full-duplex single socket connection over which messages can be sent between client and server
- two-way communication without expensive/annoying client-side polling
- ideal for low-latency persistent connections, e.g. for real-time Web applications
- requires server-side support and client-side support (JavaScript)

HTML5

HTML5 – Websocket

- starts off as a HTTP request, which indicates that it wants to "upgrade" to the WebSocket protocol
- if the server can understand it, then the http connection is switched into a WebSocket connection
- once connected, data is transmitted (bidirectionally) via frames



HTML5

HTML5 – Web Storage

- store some information locally on the user's computer, similar to cookies, but it is faster and much better than cookies and is no more secure than cookies
- information stored in the web storage isn't sent to the web server as opposed to the cookies where data sent to the server with every request

HTML5

HTML5 – Web Storage

- two types of web storage, which differ in scope and lifetime
  - local storage
    to store data for your entire website on a permanent basis
  - session storage
    to store data on a temporary basis

Example

```
<script>
// Check if the localStorage object exists
if(localStorage) {
    // Store data
    localStorage.setItem("first_name", "Peter");

    // Retrieve data
    alert("Hi, " + localStorage.getItem("first_name"));
} else {
    alert("Sorry, your browser do not support local storage.");
}
</script>
```

1

Basic Technologies

- Hypertext Markup Language
- Cascading Style Sheets

2

Interactive Web

- JavaScript
- Critical Rendering Path
- Json & Ajax
- jQuery
- React

II.1

Cascading Style Sheets

CSS – Cascading Style Sheets

- describe all the style that will be applied to the various elements (tags)
- come in three types:
  - inline = style attributes are used inline with the HTML tags
  - embedded (internal) = style information is in a `<style>` tag which is nested in the `<head>` tag of a single Web page
  - linked (external) = the style is in one .css file that is linked to multiple Web pages

II.1

Cascading Style Sheets

CSS – Cascading Style Sheets

- each CSS style definition has two components
  - the selector that defines the tag or the class that is addressed
  - the properties which define what the current style should
    - defined in braces {}-brackets
    - each statement is always terminated with a semicolon

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

Cascading Style Sheets

CSS – Cascading Style Sheets

- a distinction is made between the selectors
  - tag/element selector
    - `h1 {...}`
  - id selector
    - `#id {...}`
  - class selector
    - `.class_name {...}`
  - attribute selector
    - `[attribute] {...}`
    - `[attribute=value] {...}`

Cascading Style Sheets

CSS – Cascading Style Sheets

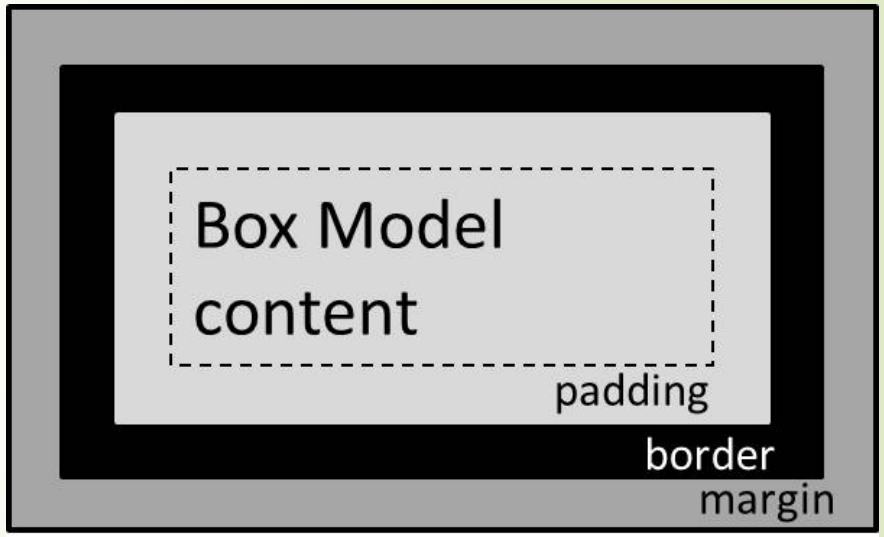
- grouping style
  - `tag1, tag2, tag3 { ... }`

- combinator selectors
  - descendant combinator (empty space)
    - `tag tag { ... }`
  - child combinator
    - `tag > tag { ... }`
  - adjacent sibling combinator
    - `tag + tag { ... }`

Cascading Style Sheets

CSS – Box model

- Border can have width, style, and color that you specify, is the only one that can be seen in a Web browser
- Padding is the space inside a border between the element content and the border, while margin is the space outside the border between the adjacent or parent elements and the border.



1

Basic Technologies

- Hypertext Markup Language
- Cascading Style Sheets

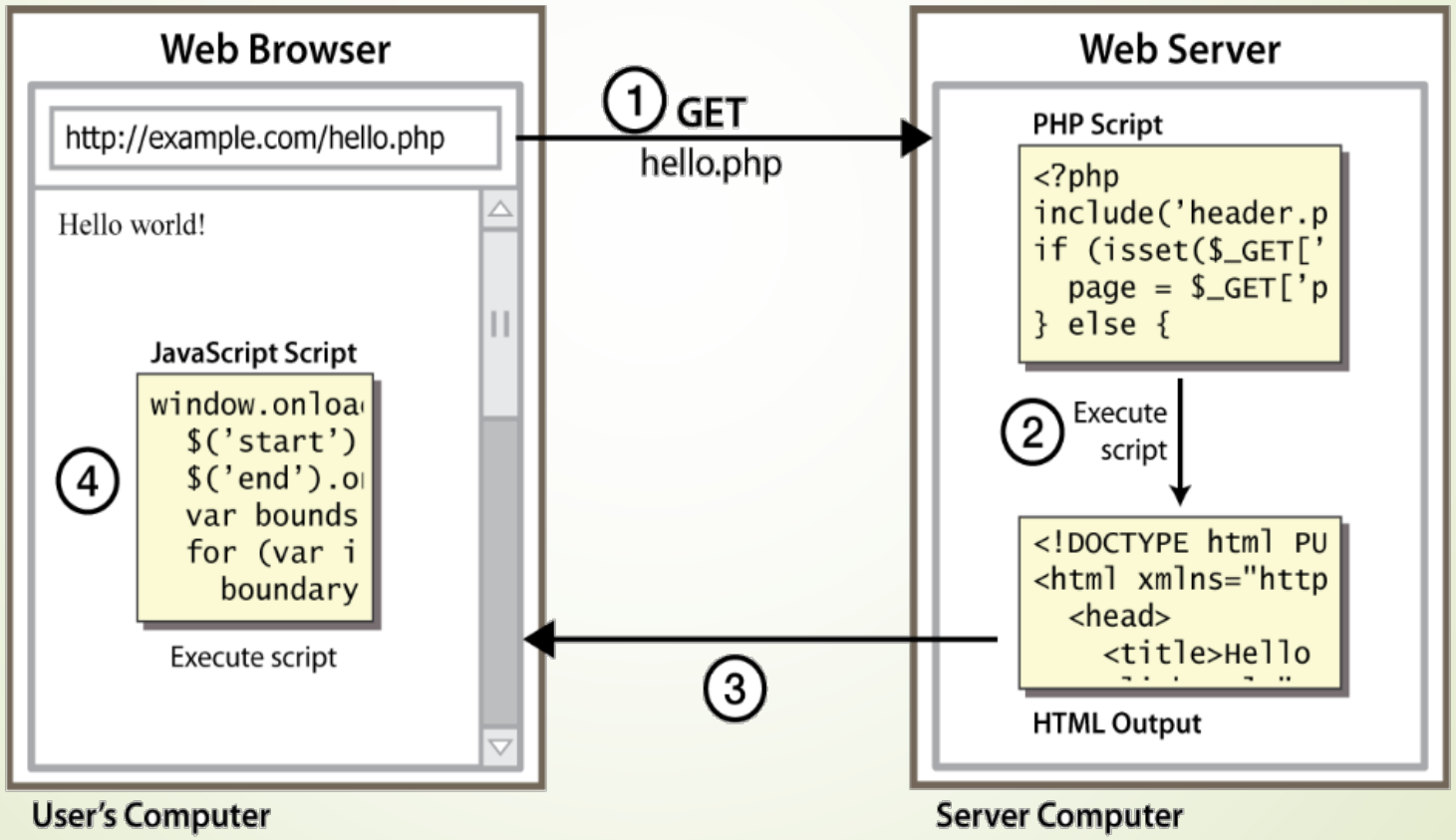
2

Interactive Web

- JavaScript
- Critical Rendering Path
- Json & Ajax
- jQuery
- React

JavaScript

Client-side Scripting



JavaScript

Client-side Scripting

- benefits:
  - usability

    can modify a page without having to post back to the server (faster UI)
  - efficiency

    can make small, quick changes to page without waiting for server
  - event-driven

    can respond to user actions like clicks and key presses

II.2

JavaScript

JavaScript

- a lightweight programming language ("scripting language")
  - used to make web pages interactive
  - insert dynamic text into HTML (e.g. a date)
  - react to events (e.g. user clicks on a button)
  - get information about a user's computer (e.g. browser type)
  - perform calculations on user's computer (e.g. form validation)

II.2

JavaScript

Attaching in HTML

- can be included in HTML using the <script> tag. Unlike CSS JavaScript can be included anywhere in the HTML document:

```
<script type="text/javascript">
    alert('This is an alert triggered by JavaScript');
</script>
```

- can be triggered by event handler attributes added to HTML tags. Event handlers respond to actions the user makes on the web-page.

```
<div onclick="alert('JavaScript Alert!');">
    Click Me
</div>
```

II.2

JavaScript

Basics

- variables, assignments
- special values: null and undefined
- concatenation and operators
- control structures

- Playgrounds/Sandboxes
  - https://playcode.io/
  - https://jsfiddle.net/

JavaScript

Basics

- string
  - usual functions

String length	String trim()
String slice()	String trimStart()
String substring()	String trimEnd()
String substr()	String padStart()
String replace()	String padEnd()
String replaceAll()	String charAt()
String toUpperCase()	String charCodeAt()
String toLowerCase()	String split()
String concat()	

JavaScript

Basics

- arrays
  - creation `var a = [10, 20, 30]`

Name	Description
concat()	Joins arrays and returns an array with the joined arrays
constructor	Returns the function that created the Array object's prototype
copyWithin()	Copies array elements within the array, to and from specified positions
entries()	Returns a key/value pair Array Iteration Object
every()	Checks if every element in an array pass a test
fill()	Fill the elements in an array with a static value
filter()	Creates a new array with every element in an array that pass a test
find()	Returns the value of the first element in an array that pass a test
findIndex()	Returns the index of the first element in an array that pass a test
forEach()	Calls a function for each array element
from()	Creates an array from an object
includes()	Check if an array contains the specified element
indexOf()	Search the array for an element and returns its position
isArray()	Checks whether an object is an array
join()	Joins all elements of an array into a string
keys()	Returns a Array Iteration Object, containing the keys of the original array
lastIndexOf()	Search the array for an element, starting at the end, and returns its position
length	Sets or returns the number of elements in an array
map()	Creates a new array with the result of calling a function for each array element
pop()	Removes the last element of an array, and returns that element
prototype	Allows you to add properties and methods to an Array object
push()	Adds new elements to the end of an array, and returns the new length
reduce()	Reduce the values of an array to a single value (going left-to-right)
reduceRight()	Reduce the values of an array to a single value (going right-to-left)

JavaScript

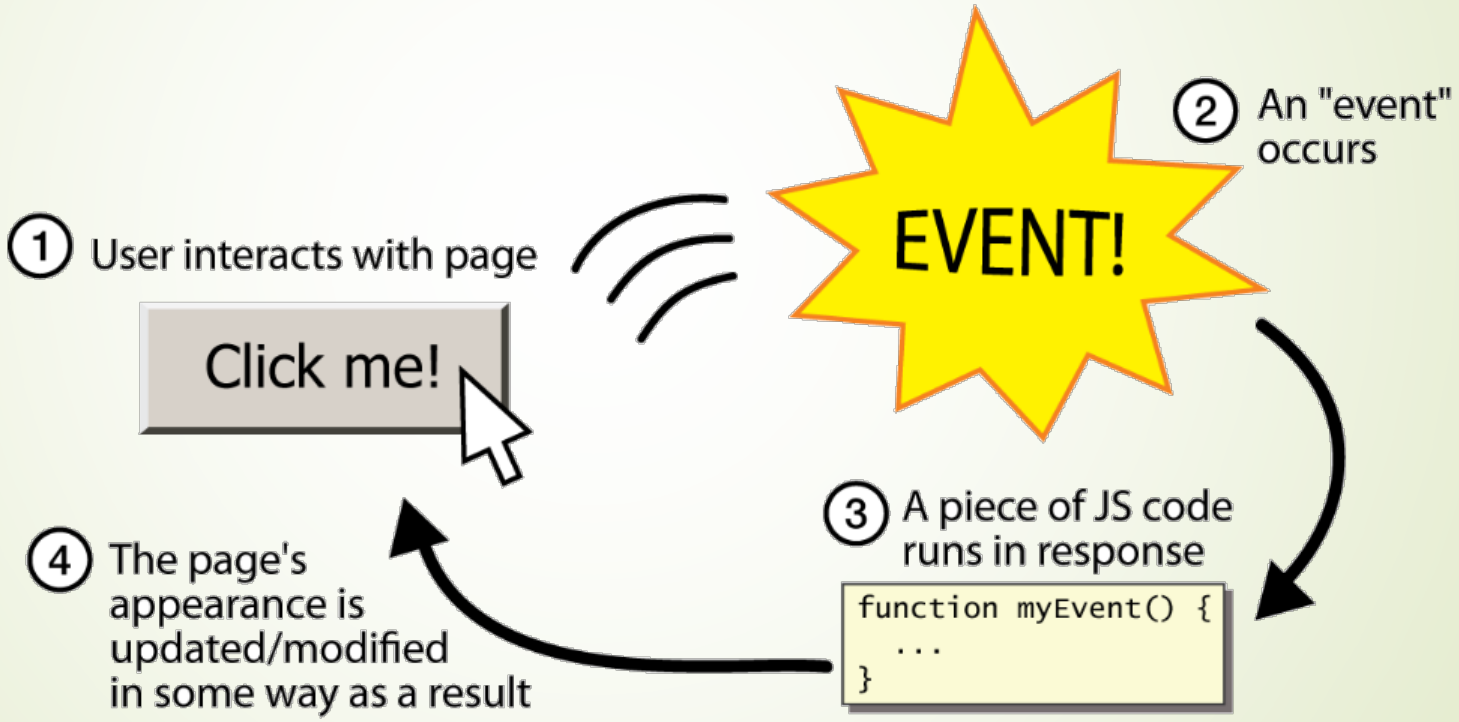
Functions

```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello() {
            document.write ("Hello there!");
         }
      </script>
   </head>

   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "sayHello()" value = "Say Hello">
      </form>
      <p>Use different text in write method and then try...</p>
   </body>
</html>
```

JavaScript

Event-driven programming



JavaScript

Event-driven programming

- event handlers

  function that's called when an event occurs

  - inline event handlers

```
<a href="site.com" onclick="dosomething();">A link</a>
```

  - DOM on-event handlers

```
window.onload = () => {
  //window loaded
}
```

JavaScript

Event-driven programming

- event handlers
  - using addEventListener

```
window.addEventListener('load', () => {
  //window loaded
})
```

JavaScript

Event-driven programming

- event object

```
link.addEventListener('click', event => {
  // link clicked
})
```

  - target
    the DOM element that originated the event
  - type
    the type of event
  - stopPropagation()
    called to stop propagating the event in the DOM

JavaScript

BOM – Browser Object Model

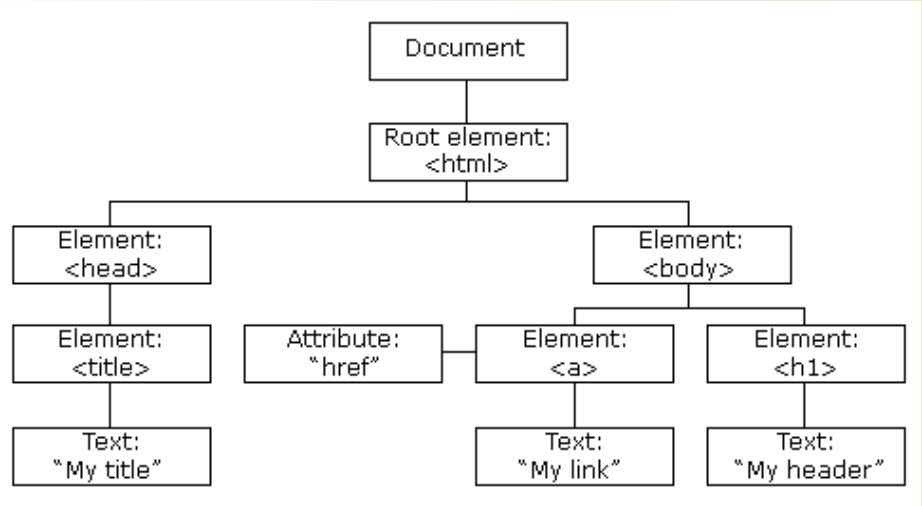
- objects in a Web document
  - window
    object of the browser, represents a window in a browser
    methods: alert, prompt, ...
  - window.screen
    contains information about the user's screen
  - window.location
    can be used to get the current page address (URL) and to redirect the browser to a new page
  - window.navigator
    contains information about the visitor's browser

II.2

JavaScript

DOM – Document Object Model

- JavaScript can
  - change all the HTML elements in the page
  - change all the HTML attributes in the page
  - change all the CSS styles in the page
  - remove existing HTML elements and attributes
  - add new HTML elements and attributes
  - react to all existing HTML events in the page
  - create new HTML events in the page



JavaScript

DOM – Document Object Model

- document
  is the owner of all other objects

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

JavaScript

DOM – Document Object Model

- example: form validation

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

```
 <form name="myForm" action="/action_page.php"
    onsubmit="return validateForm()" method="post">

    Name:
        <input type="text" name="fname">
        <input type="submit" value="Submit">
</form>
```

JavaScript

Objects

- are used to store keyed collections of various data and more complex entities
- an be created with figure brackets {…} with an optional list of properties ("key: value" pairs)

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true  // multiword property name must be quoted
};
```

- access via dot or square-bracket
- existence test: "in" operator
- "for ... in" loop

JavaScript

Objects

- methods in objects

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true  // multiword property name must be quoted

 sayHi: function() {
    alert("Hello");
  }
};
```

- this = "self in Python"

JavaScript

Objects

- constructor, operator "new"
  - technically are regular functions. There are two conventions though:
    - they are named with capital letter first
    - they should be executed only with "new" operator

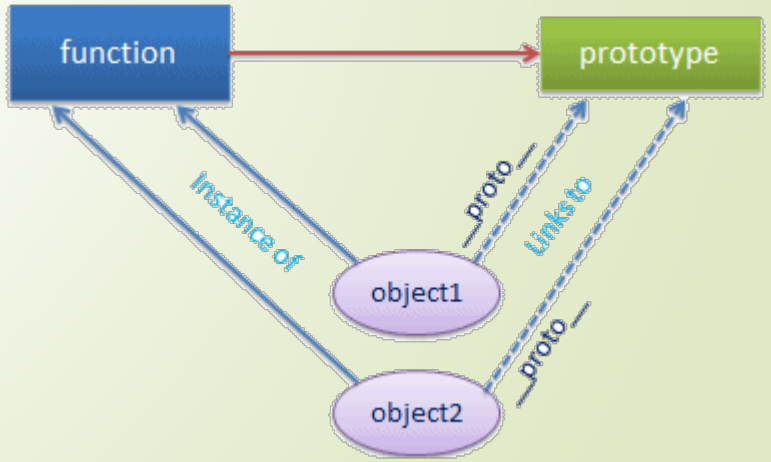
```
function User(name) {
  this.name = name;
  this.isAdmin = false;
}

let user = new User("Jack");
```

JavaScript

Objects

- prototype
  - an object that is associated with every functions and objects by default
    - function's prototype property is accessible and modifiable
    - object's prototype property is not visible
  - prototype object is special type of enumerable object to which additional properties can be attached to it which will be shared across all the instances of it's constructor function



JavaScript

Objects

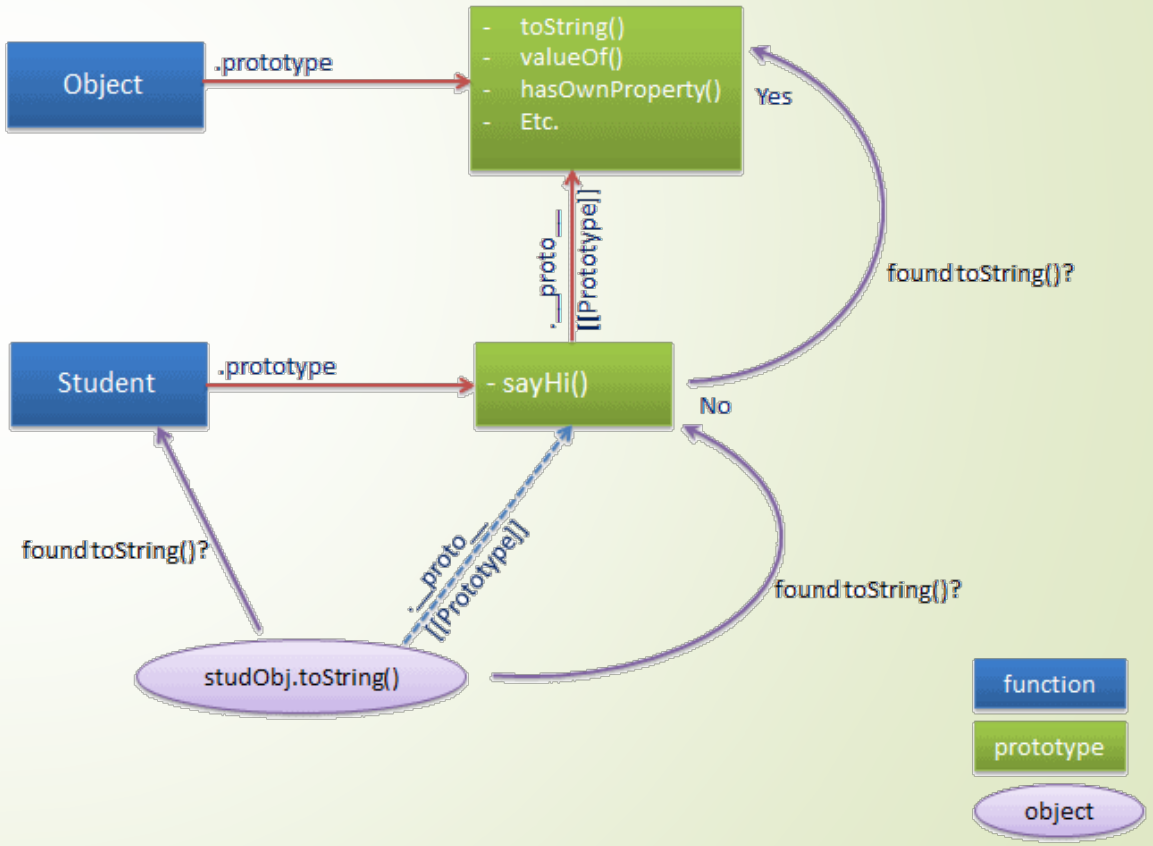
- prototype methods

Method	Description
hasOwnProperty()	Returns a boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain.
isPrototypeOf()	Returns a boolean indication whether the specified object is in the prototype chain of the object this method is called upon.
propertyIsEnumerable()	Returns a boolean that indicates whether the specified property is enumerable or not.
toLocaleString()	Returns string in local format.
toString()	Returns string.
valueOf	Returns the primitive value of the specified object.

JavaScript

Objects

- prototype usage
  - to find properties and methods of an object
  - to implement inheritance



JavaScript

Objects

- inheritance

```
function Person(firstName, lastName) {
    this.FirstName = firstName;
    this.LastName = lastName;
};

Person.prototype.getFullName = function () {
    return this.FirstName + " " + this.LastName;
}
```

```
function Student(firstName, lastName, schoolNamee) {
    Person.call(this, firstName, lastName);

    this.SchoolName = schoolName;
}

Student.prototype = new Person();
Student.prototype.constructor = Student;

var std = new Student("James","Bond", "XYZ");

std.getFullName();
```

1

Basic Technologies

- Hypertext Markup Language
- Cascading Style Sheets

2

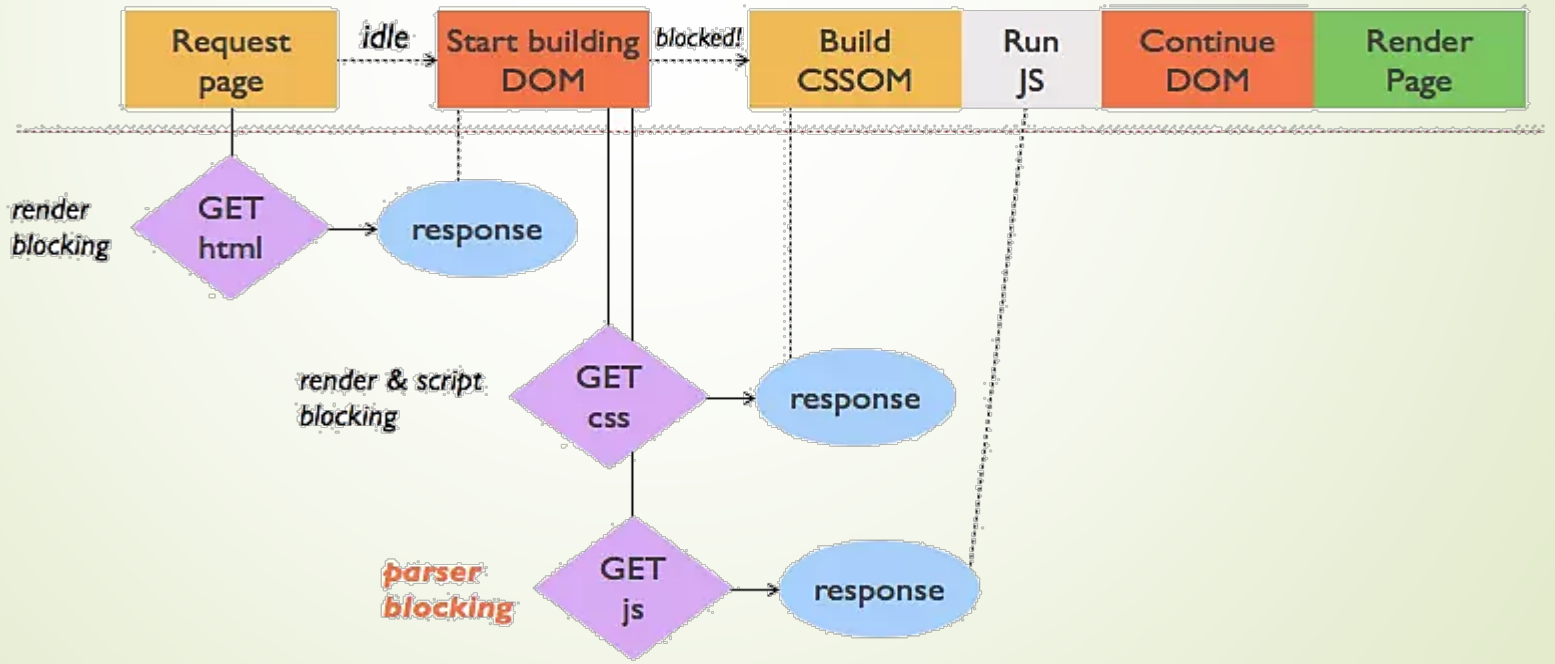
Interactive Web

- JavaScript
- Critical Rendering Path
- Json & Ajax
- jQuery
- React

Critical Rendering Path

Critical Rendering Path

- steps to turn "the code and resources required to render the initial view of a web page" into actual pixels on the screen



https://www.sitepoint.com/optimizing-critical-rendering-path/

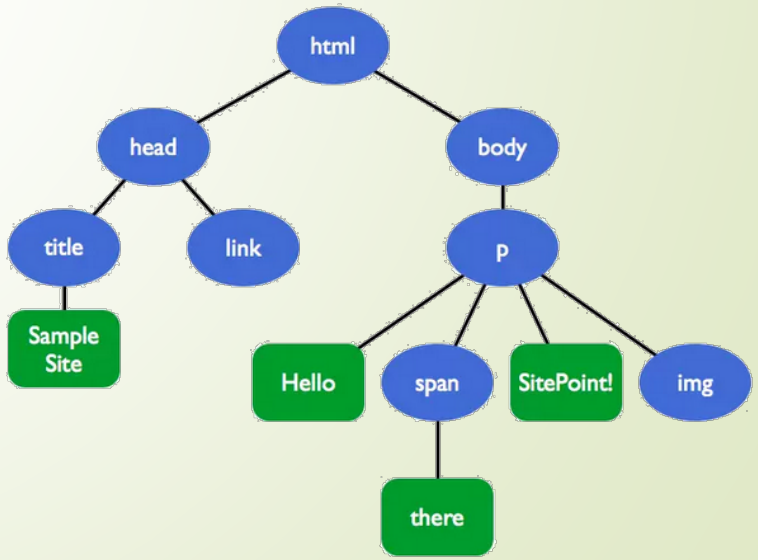
II.2

Critical Rendering Path

Critical Rendering Path

1. browser sends HTTP-request
2. browser receives HTML-response
3. browser parses stream of bytes into DOM-tree incrementally

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Sample Site</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <p>
      Hello <span>there</span> SitePoint!
      <img src="photo.jpg">
    </p>
  </body>
</html>
```



II.2

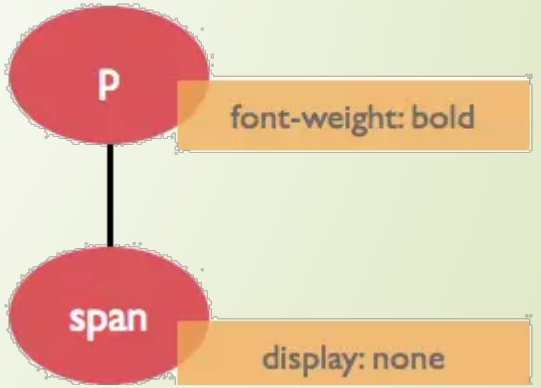
Critical Rendering Path

Critical Rendering Path

1. browser downloads CSS-files
2. CSS file is then parsed into the CSS Object Model, or CSSOM

CSS is render blocking!

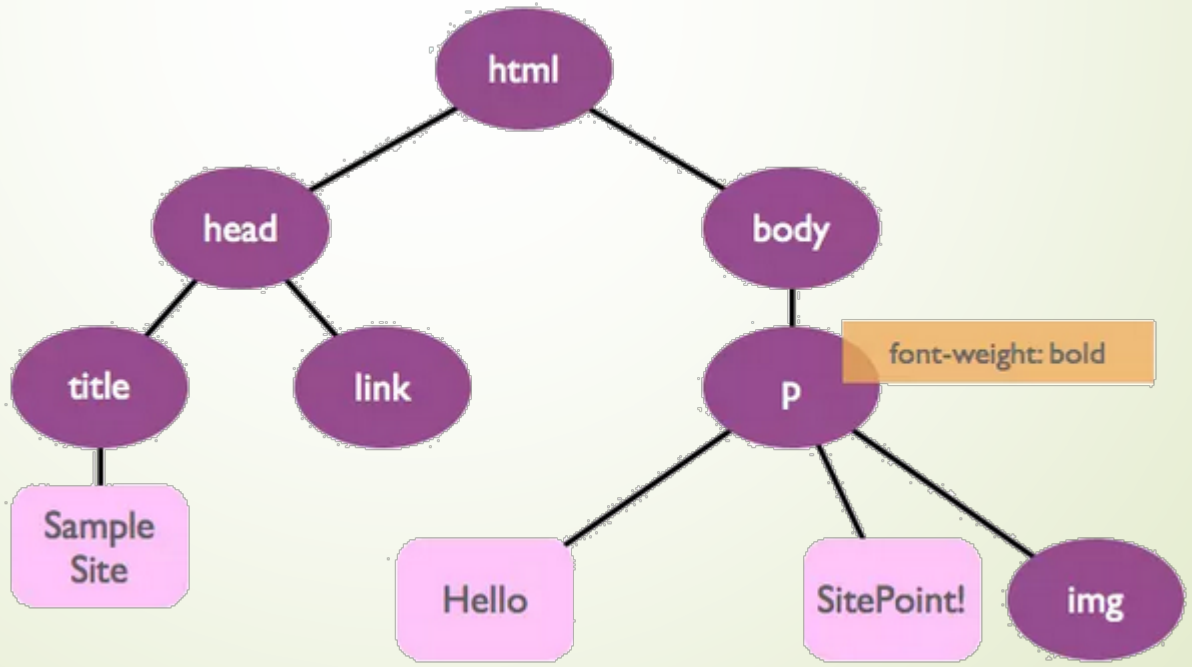
```
p { font-weight: bold; }
p span { display: none; }
```



Critical Rendering Path

Critical Rendering Path

1. browser builds a render tree
   structure combines DOM and CSSOM while only capturing visible elements



Critical Rendering Path

Critical Rendering Path

- JavaScript has a huge impact on the critical path
- scripts can both query and change the DOM as well as the CSSOM
  - JavaScript is parser blocking
  - CSS is also script blocking

```
<p>
  Hello <span>there</span>, SitePoint!
  <script>
    document.write('How are you?');
    var color = elem.style.color;
    elem.style.color = 'red';
  </script>
  <img src="photo.jpg">
</p>
```

Critical Rendering Path

Optimizing Critical Rendering Path

- minimize the bytes
  - minify, compress, and cache the assets as well as the HTML

CSS Minification Tools	JavaScript Minification Tools
▪ CSSnano ▪ CSSO ▪ UNCSS ▪ CSS-Minifier	▪ Closure Compiler ▪ UglifyJS2 ▪ YUI Compressor ▪ JS Compress

- minimize render blocking CSS
  - get CSS to the user as soon and as fast as possible
  - provide media information

II.2

Critical Rendering Path

Critical Rendering Path – Summary

- the notion of page speed has shifted from simple page loading to page rendering
- Critical Rendering Path comprises all steps to turn critical resources into a visible browser output: DOM and CSSOM, JavaScript, render tree, layout and paint phase
- HTML is render blocking, but the DOM can be built incrementally
- CSS is render and script blocking, treat it carefully and optimize it with inline styles or media queries
- JS is parser blocking, use it sparingly during the initial page load, defer execution or try to load it asynchronously
- don't forget that size still matters and minify, compress, cache

1

Basic Technologies

- Hypertext Markup Language
- Cascading Style Sheets

2

Interactive Web

- JavaScript
- Critical Rendering Path
- Json & Ajax
- jQuery
- React

II.2

JSON

JSON

- JavaScript Object Notation (JSON)
  Data format that represents data as a set of JavaScript objects
- natively supported by all modern browsers
  and libraries to support it in old ones
- not yet as popular as XML, but steadily rising due to its simplicity and ease of use

II.2

JSON

JSON

```
{
    "private":   "true",
    "from":      "Alice Smith (alice@example.com)",
    "to": [
            "Robert Jones (roberto@example.com)",
            "Charles Dodd (cdodd@example.com)"
    ],
    "subject":   "Tomorrow's event!",
    "message": {
            "language":  "english",
            "text":      "Hey guys, don't forget me!"
    }
}
```

II.2

JSON

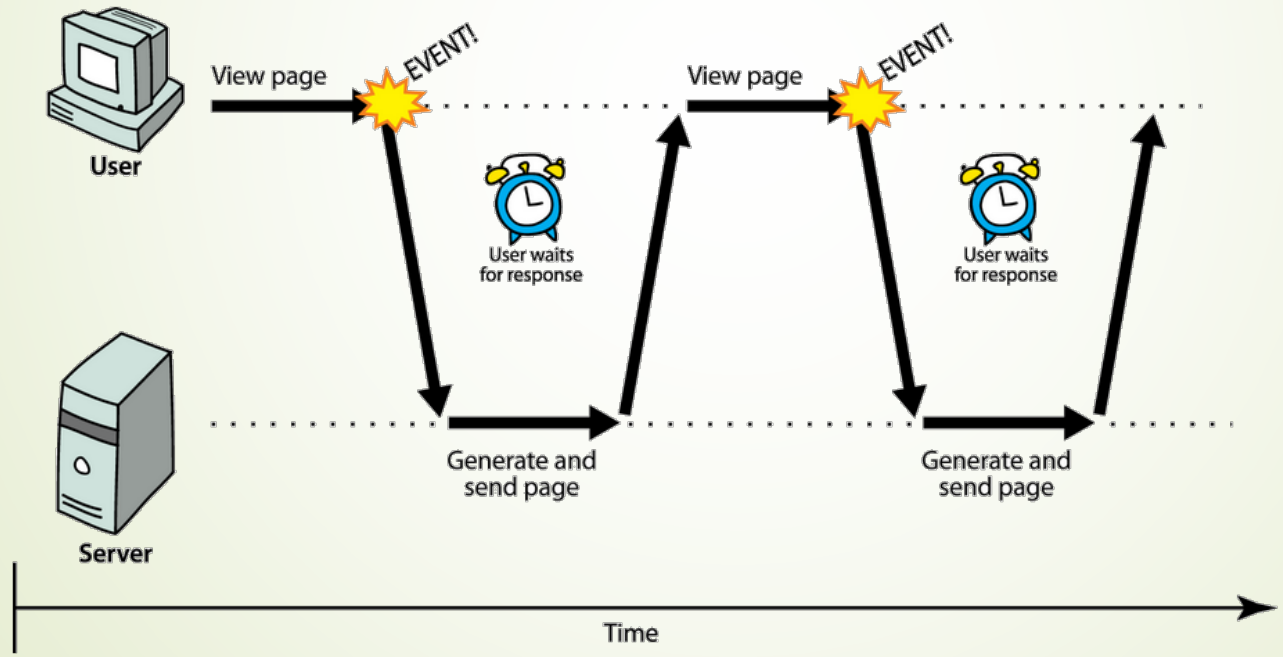
Browser JSON methods

- `JSON.parse(string)`
  converts the given string of JSON data into an equivalent JavaScript object and returns it
- `JSON.stringify(object)`
  converts the given object into a string of JSON data (the opposite of JSON.parse)

Web Communication

Synchronous

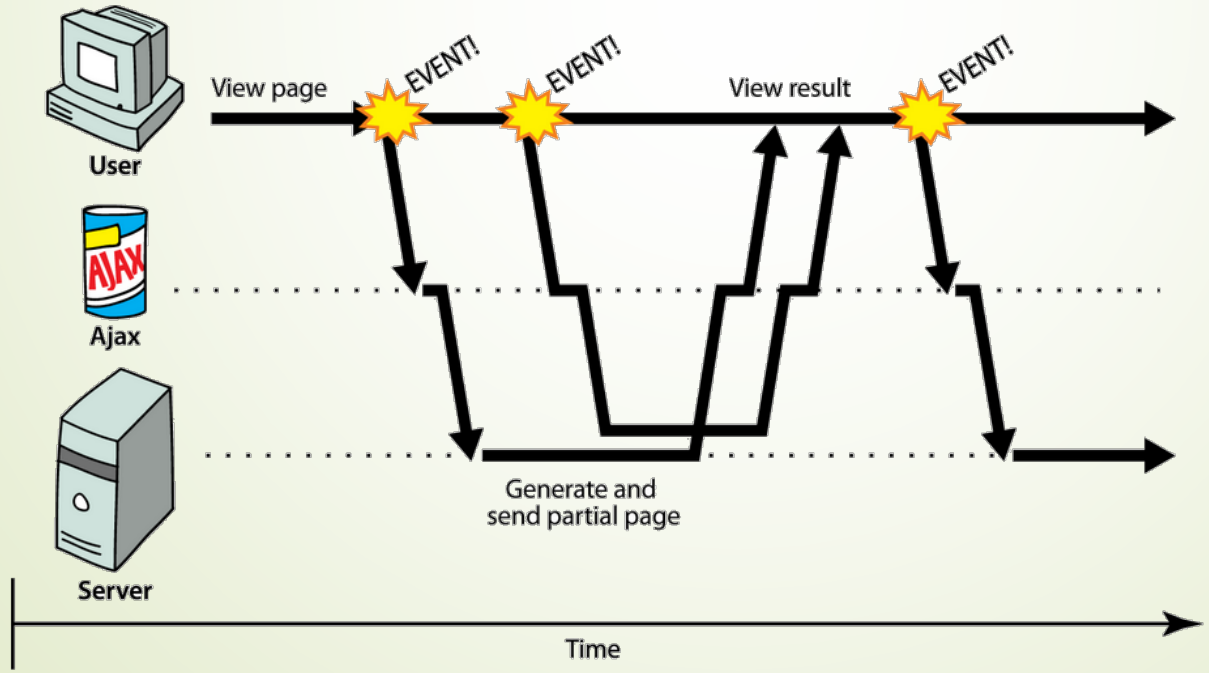
- pattern: click, wait, refresh



Web Communication

Asynchronous

- keep interacting with page while data loads
- communication pattern made possible by Asynchronous JavaScript and XML (AJAX)



II.2

AJAX

Technologies

- XHTML and CSS for presenting information
- DOM for dynamically interacting with and displaying the information presented
- XMLHttpRequest object to manipulate data asynchronously with the Web server
  - update a web page without reloading the page
  - request data from a server - after the page has loaded
  - receive data from a server - after the page has loaded
  - send data to a server - in the background
- XML, HTML, and XSLT for data interchange and manipulation
- JavaScript for binding data requests and information display

II.2

AJAX

XMLHttpRequest

- prepare data
- determine processing
- send request

```
var xrq = new XMLHttpRequest();

xrq.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        document.getElementById("demo").innerHTML =
            this.responseText;
    }
};
xrq.open("GET", "ajax_info.txt", true);
xrq.send();
```

- the response is available as a string or as a parsed XML document in the `responseText` and `responseXML` properties
- use JavaScript to use the response and update the current page's DOM

1 Basic Technologies

- Hypertext Markup Language
- Cascading Style Sheets

2 Interactive Web

- JavaScript
- Critical Rendering Path
- Json & Ajax
- jQuery
- React

jQuery

jQuery

- powerful javascript library
  - many functions to make programming tasks easier
  - uses short-hand notations
- jQuery is accessible as a global function and as an object instance
  - function "jQuery", abbreviated as "$"
- provides additional utility functions
  - frequent pattern: `$.fname(parameters)`
- supports event handlers
  - frequent pattern: `DOMObject.eventname(function)`
  - convenient pattern: using local anonymous functions

jQuery

jQuery - Selectors

- syntax is tailor-made for selecting HTML elements and performing some action on the element(s)
- basic syntax: `$(selector).action()`
  - a $ sign to define/access jQuery
  - a (selector) to "query (or find)" HTML elements
  - a jQuery action() to be performed on the element(s)

```
$(this).hide()          hides the current element
$("p").hide()           hides all <p> elements
$(".test").hide()       hides all elements with class="test"
$("#test").hide()       hides the element with id="test"
```

jQuery

JavaScript / jQuery - Selectors

`document.querySelector(`*`CSS selectors`*`)` | `$(`*`CSS selectors`*`).action()`

Selector	Example	Example description
.class	.intro	Selects all elements with class="intro"
.class1.class2	.name1.name2	Selects all elements with both *name1* and *name2* set within its class attribute
.class1 .class2	.name1 .name2	Selects all elements with *name2* that is a descendant of an element with *name1*
#id	#firstname	Selects the element with id="firstname"
*	*	Selects all elements
element	p	Selects all <p> elements

jQuery

JavaScript / jQuery - Selectors

`document.querySelector(`*`CSS selectors`*`)` | `$(`*`CSS selectors`*`).action()`

Selector	Example	Example description
element.class	p.intro	Selects all <p> elements with class="intro"
element,element	div, p	Selects all elements and all <p> elements
element element	div p	Selects all <p> elements inside elements
element>element	div > p	Selects all <p> elements where the parent is a element
element+element	div + p	Selects the first <p> element that is placed immediately after elements
element1~element2	p ~ ul	Selects every element that is preceded by a <p> element
[*attribute*]	[target]	Selects all elements with a target attribute
[*attribute=value*]	[target=_blank]	Selects all elements with target="_blank"

jQuery

jQuery – Example

```
$('div').each( function(index, value) {
  console.log('div${index}: ${this.id}');
});
```

jQuery

jQuery – Callback functions

- JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.
- a callback function is executed after the current effect is finished.
- typical syntax: `$(selector).hide(speed, callback);`

```
$("button").click( function() {
  $("p").hide("slow", function() {
    alert("The paragraph is now hidden");
  });
});
```

jQuery

jQuery – AJAX

```
$('#btn').click(function() {
    var selIdsText = $('#mysongs input:checked').map(function() {
        return $(this).parents('tr').children().first();
    }).text();
    $.ajax({
        type: 'POST',
        url: 'serverDummy.php',
        data: {selection: selIdsText},
        success: function(data) {
            ...
        }
    });
});
```

jQuery

Javascript versus jQuery

Javascript Code	jQuery Code

`window.onload = onDocumentReady;`
`function onDocumentReady() {`
`  // body of function`
`}`
	`$(document).ready(function(){`
	`  // body of function`
	`});`

`document.querySelector("nav ul")`	`$("nav ul");`
`document.querySelector("h1").innerHTML = "Welcome";`	`$("h1").text("Welcome");`

`var button = document.querySelector("button");`
`button.onclick = function() {`
`  // body of function`
`}`
	`$("button").click(function() { ...});`

`document.querySelector("h1").style.color = "red";`	`$("h1").css("color", "red")`

1

Basic Technologies

- Hypertext Markup Language
- Cascading Style Sheets

2

Interactive Web

- JavaScript
- Critical Rendering Path
- Json & Ajax
- jQuery
- React

II.2

JavaScript ES6

What is ECMAScript 6 (or ES6)

- ECMAScript 2015 (or ES6)
  sixth and major edition of the ECMAScript language specification standard
  defines the standard for the JavaScript implementation
- features
  - const
  - let versus var
    - var
      function-scoped and hoisted at the top within its scope
    - let
      block-scoped ({}) and they are not hoisted
  - for ... of - loop

II.2

JavaScript ES6

What is ECMAScript 6 (or ES6)

- Template Literals

```
let result = `The sum of ${a} and ${b} is ${a+b}.`;
```

- Default Values for Function Parameters
- Rest Parameters

```
function myFunction(a, b, ...args)
```

- Arrows Functions

```
var sum = (a, b) => a + b;
```

- Destructuring Assignment for Arrays and Objects

```
let fruits = ["Apple", "Banana"];
let [a, b] = fruits;
```

JavaScript ES6

What is ECMAScript 6 (or ES6)

- Classes
  - to create objects, implement inheritance by using the extends keyword, and reuse the code

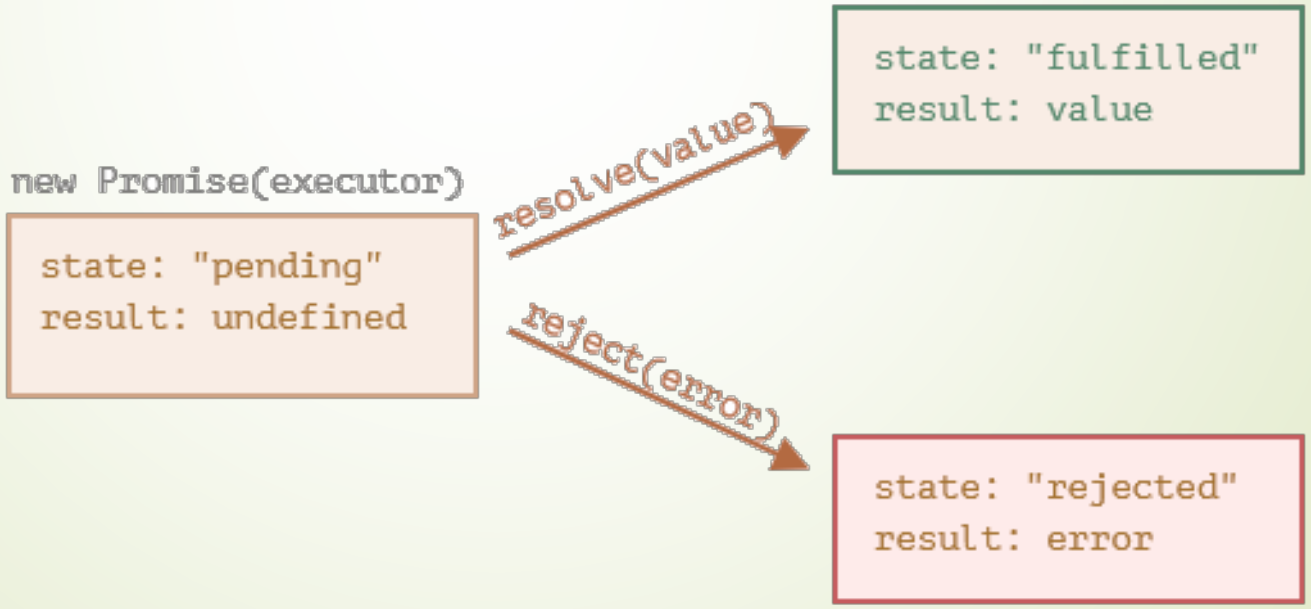
```
class Rectangle {
      constructor(...)
      ...
      getCoordinates()
}
class Square extends Rectangle {
      ...
}
```

AJAX

Fetch API

- Promise – object

  contains both the producing code and calls to the consuming code



II.2

AJAX

Fetch API

- Promise – object
  - serves as a link between the executor (the "producing code" or "singer") and the consuming functions (the "fans"), which will receive the result or error. Consuming functions can be registered (subscribed) using the methods .then and .catch

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => resolve("done!"), 1000);
});

// resolve runs the first function in .then
promise.then(
  result => alert(result), // shows "done!" after 1 second
  error => alert(error) // doesn't run
);
```

AJAX

Fetch API

- fetch()
  - starts the process of fetching a resource from a server
  - returns a Promise that resolves to a Response object

```
fetch(url)
    .then(response => {
        // handle the response
    })
    .catch(error => {
        // handle the error
    });
```

II.2

TypeScript

What is TypeScript

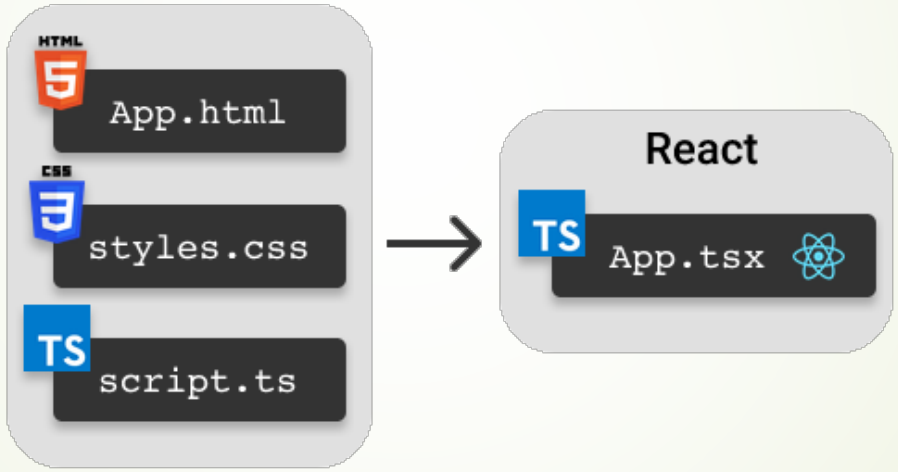
- a syntactic superset of JavaScript
- adds static typing
- uses compile time type checking

II.2

React

What is React

- open-source front-end JavaScript library for building user interfaces



- widely used as a base in building single-page websites and mobile applications
  - a single-page application is a web application or website that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of the default method of a web browser loading entire new pages

II.2

React

What is React

- advantages
  - reuse some components and the open-source instrument has many add-ons
  - the virtual DOM allows you to interact with much traffic
  - a flexible code structure
- disadvantages
  - no structured documentation

II.2

React

What is React

- objective
  display components in the browser with high performance
- React uses a so-called reconciliation algorithm
  optimizes the rendering, i.e. the graphical display of the web app in the browser, and thus the speed of the web app
  - components are translated into the DOM
  - every change to the DOM causes the browser to recalculate some of the CSS styles as well as the arrangement of the HTML elements
  - if only a certain part of the user interface is to change compared to the currently visible user interface, only this corresponding part should be re-rendered
  - React uses the web API provided by browsers to access the current DOM and creates a virtual DOM from it - a representation of the user interface stored in the browser cache

React

What is React

- Online-Editor https://codesandbox.io/s/

index.html

```
<body>
  <noscript>
    You need to enable JavaScript to run this app.
  </noscript>
  <div id="root"></div>
  <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
</body>
```

```
import App from "./App";

const rootElement = document.getElementById("root");
const root = createRoot(rootElement!);

root.render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

II.2

React

Features of React – React-Elements and JSX

- tags are React-elements, i.e. a JavaScript-object
  - JavaScript extended by XML-tags, called JSX
  - JSX is a template script combining the power of using HTML and JavaScript together
    - allows to make use of HTML and JavaScript in the same file and take care of the state changes in the DOM in an efficient manner
      - HTML for a UI
        each element in the DOM will have events to be handled, state changes, etc.
      - Javascript Expressions
        can be used in the jsx files using curly brackets

```
const Title = <h1>Hello world: {new Date().toISOString()}</h1>;
```

    - each React element is a JavaScript object that describes a specific HTML element as it appears in the DOM
  - example: babel website

React

Features of React – Virtual DOM

- problem of DOM
  recalculation of CSS and layout when DOM changes
- minimize the time it takes to repaint the screen
  - whenever a new element is added to the UI, a virtual DOM is created
  - if the state of this element changes, React would recreate the virtual DOM for the second time and compare with the previous version to detect which of the virtual DOM object has changed
  - it then updates only the object on the real DOM

React

Features of React – Components

- components are JavaScript functions returning React-elements
- make the code easy by splitting the logic into reusable independent code
- can be used as functions and as classes
- have a state and properties which makes life easy
- building an application with React, means to build a bunch of independent, isolated and reusable components
- components can be merged and thereby creating a complex user interface

React

Features of React – Components

- HTML-structs can be directly written in JavaScript

App.tsx

```
3   export default function App() {
4     return (
5       <div className="App">
6         <h1>Hello CodeSandbox</h1>
7         <h2>Start editing to see some magic happen!</h2>
8       </div>
9     );
10  }
```

- will be translated by React render-engine REACT-DOM to HTML-structs passed to the browser

React

Features of React – Components

- defining a new component

```
const LoadingText = () => {
  const isLoading = false;
  return
   <div>
   {isLoading ? <p>Loading...</p> : <h2>Ready</h2>}
   </div>;
};
```

```
export default function App() {
  return (
    <div className="App">
      <h1>React Example</h1>
      <h2>Time now: {new Date().toISOString()}</h2>
      <LoadingText />
    </div>
  );
}
```

React

Features of React – Components

- properties (props)
  - similar to HTML-attributes
  - passed to a React element or component by placing it in the opening tag

```
const LoadingText = (props) => {
  return
   <div>
  {props.isLoading ? <p>Loading...</p>: <h2>Ready</h2>}
   </div>;
};
```

```
export default function App() {
  return (
    <div className="App">
      <h1>React Example</h1>
      <h2>Time now: {new Date().toISOString()}</h2>
      <LoadingText isLoading={true}/>
    </div>
  );
}
```

React

Features of React – Components

- alternatives

```
function LoadingText(props) {
  return
   <div>{props.isLoading ? <p>Loading...</p>: <h2>Ready</h2>}</div>;
}
```

```
class LoadingText extends React.Component {
  render() {
    return
       <div>{props.isLoading ? <p>Loading...</p>: <h2>Ready</h2>}</div>;
   }
}
```

React

Features of React – Event Handler

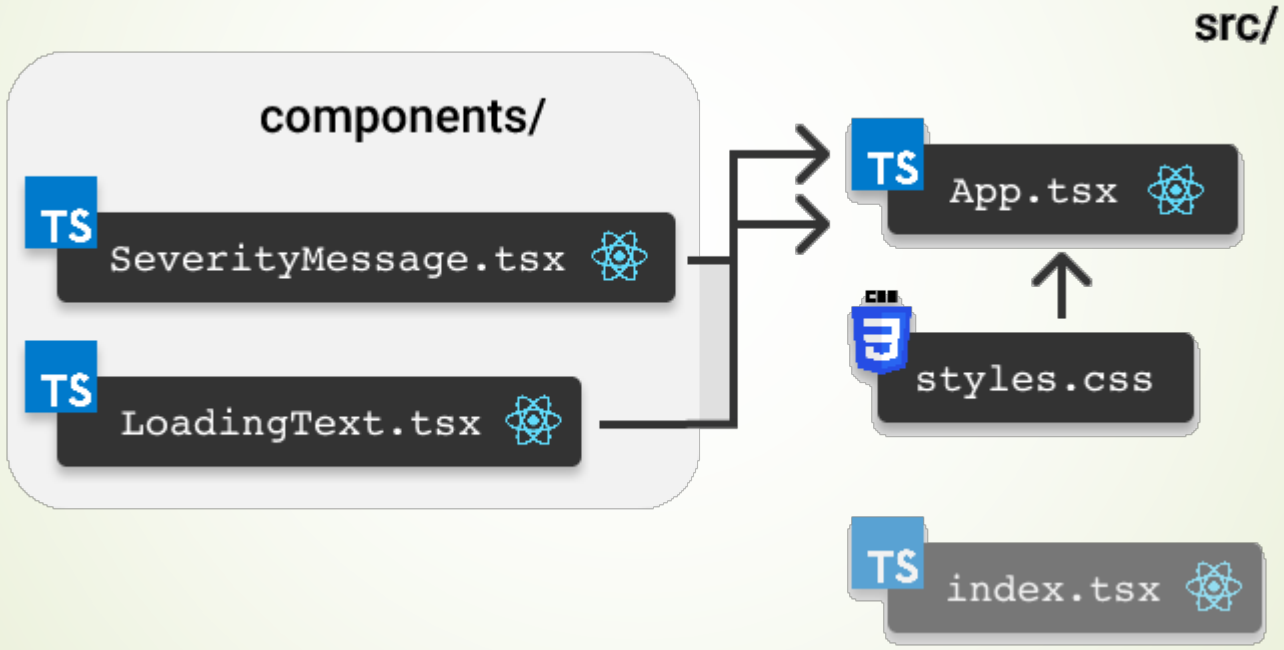
- onClick – prop

```
<button onClick={() => alert("A click")}>Click me</button>
```

- In JavaScript, functions are "first-class citizens"
  - functions can be passed to other functions as parameters
  - here, a callback function is passed as prop to a component

React

Project Organization



React

Project Organization

- TypeScript - component file
  - interface definition for props and their types
  - FC function component
    used to formulate React-components as functions returning JSX
    extends interface by children prop

```
import { FC } from "react";

interface LoadingTextProps {
  isLoading: boolean;
}

const LoadingText: FC<LoadingTextProps> = ({ isLoading }) => (
  <div>{isLoading ? <span>Loading...</span> : <h2>Ready</h2>}</div>
);

export default LoadingText;
```

React

Project Organization

- TypeScript - component file

```
import LoadingText from "./components/LoadingText";


export default function App() {
  return (
    <div className="App">
      <h1>React Example</h1>
      <h2>Time now: {new Date().toISOString()}</h2>
      <LoadingText isLoading={true}/>
    </div>
  );
}
```