# CSCI2720 MIDTERM YUSIHONG

## *Hypertext Markup Language*
<!DOCTYPE html><!–comment–>
<tag: element name, attribute name & value>content</element>
<b>, <i>, <u>, <sub>, <sup>, <pre>, <ol>, <ul>, <li>
<table>, <th><td>, <tr><td>
<section>, navigation bar<nav>
<a href="csci2720.html" target="_blank">Webapp</a> (new tab)
intro(id="intro")
<img src="cuhk.jpg" width="100" height="150">
<video control width="500">
<source src="2720ver1.mp4" type="mp4">
</video>
< &lt; > &gt; blank space &nbsp;

## *Bootstrap*
twitter, 2011, v5.2.0
<link href=".../bootstrap.min.css" rel="stylesheet"
integrity="..." crossorigin="anonymous">
sm ≥576px, md 768, lg 992, xl 1200, xxl 1400
"col-sm-12"
.border-start, .text-warning (yellow), .bg-success (green),
.d-block (block-level display), .mx-2 (margin on x-directions)
<i class="bi bi-file-music"></i> (icon)

## *Cascading Style Sheets*
<link rel="stylesheet" href="style1.css" >
p{color:orange;}
specific one, later one, !important
p:nth-child(3), p::first-letter
id, .class, :hover (mouse over)
px (pixel), em (font size), rem (root font size)
%, vh (1% viewport height), vw
#rrggbb #ffff00(yellow)
html element for css: div, span(inline)
display:none (occupy no space), visibility:hidden (has space)
position: absolute, fixed, relative, sticky
padding(internal space), margin(ext), border(line around)
{padding: top, right, bottom, left;}
*responsive web design*
#square {width: 200px;height: 200px;background: yellow;}
transforms: #square {transform: translateX(300px)
rotate(45deg) skew(-30deg);}
transition: {transition: width 2s ease-in, height 1s
ease-out, background 3s linear;}
animation: keyframes myani {
0% { font-size: 10px; background: yellow; }
100% { font-size: 40px; background: red; }}
#square { aniamtion-name:myani; animation-duration:5s;}
border-radius, text-shadow, box-shadow, column-count
display: block, inline, none, flex, grid
SASS $main-color:green;
custom.scss import "other.scss"; body{...}
scss: include color-scheme(dark) {...}
css: media (prefers-color-scheme: dark) {
or <link rel="stylesheet" href="dark.css"
media="(prefers-color-scheme: dark)">

## *Javascript*
var,let, const, (letter,number,_,$)
!undeclared variables are created as globals
string, number, bigint, boolean(not 0 are true), undefined, symbol, null
typeof x; Number(x);
array []; comment // /**/
for (x of arr) {x...}; for (x in dict) {dict[x]...};
hello = () => {};
window.screen location history alert confirm prompt
*Typescript* a:(string|number)

## *Input and Form*
<form action="" method="get" class="form-example">
<div class="form-example">
<label for="name">Enter your name: </label>
<input type="text" name="name" id="name" required></div>
<div class="form-example">
<input type="submit" value="Subscribe!"></div></form>
type: text,password,email,search,tel,url,color,checkbox,radio
<textarea>, <select><option></></>
<fieldset><legend></></>, <button>
value="initial", readonly, disableed, required, autofocus
GET: inside request URL string ( 2k) (seen by other, risky)
POST: in HTTP request body ( 1MB to 2GB)

## *Asynchronous and Fetch*
event loop (callback queue => call stack)
setTimeout(func(), time);, setINterval, clearTimeout()
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)
myResolve(); // when successful
myReject(); // when error });
// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
function(value) { /* code if successful */ },
function(error) { /* code if some error */ });
let promiseChain = new Promise((resolve,reject)=>{...};
promiseChain().then().then().catch((err)=>{...});
fetch('http...').then().catch().finally();
old method: AJAX (asynchronous JavaScript and XML)
async function f() { await f1(); await f2(); ...}
async function f() { try{} catch(err){} finally{} }
Promise.all([promise1, promise2, promise3]).then((values) =>
{console.log(values);}); // [3, 42, "foo"]

## *JS DOM, Events and Objects*
*Document Object Model*
querySelectorAll(), getElementById(), document.images...
element.innerHTML, innerText, value, attribute, style.property
(navigation) let para = document.createElement("p");
(event) document.querySelector("#p1").onclick = ...;
(object, a collection of properties)
let stu1 = { name:"john", age:18 } (literal notation)
let stu1 = new Object(); stu1.name = ...; (define directly)
function Student(name,age){this.name=name;...};
let stu1 = new Student(...); (constructor, function/class)
*JSON* javascript object notation
JSON.parse(str); JSON.stringify(obj);

## *JS Function and Array*
parameter: list of input when define
argument: actual values at function call
missing argument will be given "undefined" value
more can be get by rest operatior ...: f(x,y,...more){}
declaration: function f1()...
expression: let f1 = function ()...
arrow let f1 = () => {}
invoking function (no pollution, isolate): ()=> f1()...
nested function (outside cannot see inside)
callback: function f1(callback1) { callback1();... }
generator function: function* f1(){ ...yield x+=2;}
let count = f1(2); count.next().value //2,4,6...
object methods: object can contain functions (as value)
*Array* verify: Array.isArray(x);
spread operator: c=[...a, 0, ...b]
separate variables: [a,b,...rest]=[...]
arr.slice(start(inclusive),end(exclusive))
arr.splice(start,count,[item to add])
// c: ["cyan","magenta","yellow","black"]
let c2 = c.splice(2,1,"red","green","blue");
// c2: ["cyan","magenta","red","green","blue","black"]
pop, push(item), shift, unshift(item)
arr.forEach((item,idx,arr)=>{...})
arr.indexOf(item, start), arr.lastIndexOf(item, start)
arr.includes(value), arr.find() //first, arr.filter() //more
arr.reverse(), arr.split/join, arr.map()
imperative programming: procedural, object-oriented
declarative programming: functional
let x = [1,2,3,4,5];
let transform = n => n*2;
let y = x.map(transform);
console.log(y); // [2,4,6,8,10]
result solely depends on the input
no side effect(screen output)/no variable mutation
curry, break a function taking multiple arguments
let add = (a,b) => a + b; add(3,4); // 7
let add2=a=>{return b=>{return a + b;};}; add2(3)(4);// 7
or elegant syntax: let add3 = a => b => a+b;

## React

Markup + Styling + Scripts (HTML + CSS + JavaScript)
fast, modular, scalable, flexible
<div id="app">React is rendering...</div>
const root =
ReactDOM.createRoot(document.querySelector('app'));
root.render(<App />);
css in react:
<div style={{ height: 10 }}>Hello World!</div>
class Welcome extends React.Component
{...render(){return()}}
*props* <Item subject="...">,
in Item class: {this.props.subject}
*state* constructor() { super();
this.state = { s1:"CSCI", s2:"CENG", s3:"AIST" };}
*event* onClick={this.f1}
Data are passed to children as props
Events are handled by parents, as the handler has been passed as props
If information needs to be passed to the parent, the technique of "lifting state up" could be used
controlled components:
event.preventDefault() to avoid default actions (e.g., submit)
*state in functional component*
const [time, setTime] = useState("morning");
*class component*
componentDidMount() { console.log("Mounted...");
setTimeout(()=>this.setState({count:this.state.count-1}),1000);}
componentDidUpdate() { console.log("Updated with count="
+this.state.count); if (this.state.count > 0)
setTimeout(()=>this.setState({count:this.state.count-1}),1000);}
*functional component*
useEffect(() => { console.log("Updating with count="+count);
if (count>0) setInterval(()=>setCount(count-1), 1000); });
*typescript*
type MyProps = { name: string; }
type MyState = { time: string; }
class Home extends React.Component<MyProps, MyState>
function Home2 (props:MyProps) :JSX.Element
const EasyHome:React.FC<MyProps> = (props:MyProps) =>

## NPM

npm init, install (-g), start, run build
npx create-react-app my-app

## SPA and Routing

web page: static content pages with hyperlinks
web applicaiton: + dynamic contents
*single-page application*
pagination: sense of control
infinite scrolling: content discovery
window.history: back(), forward(), go(num)//in the history list
pushState(data, title, url), replaceState(), onpopstate
window.location: assgin(), replace()

### React Router

import React from 'react';
import ReactDOM from 'react-dom/client';
importBrowserRouter,Routes,Route,Linkfrom'react-router-dom';
return ( <BrowserRouter> <div> <ul> <li> <Link to="/"> Home </Link> </li> <li> <Link to="/about"> About </Link> </li> </ul> <hr/> <Routes> <Route path="/" element=<Home/> /> <Route path="/about" element = <About/> /> </Routes> </div></BrowserRouter>);
in js:
<script defer src="script.js"></script>
<script type="module" src="main.js"></script>
in react:
const OtherComponent = React.lazy(() => import('./OtherComponent'));
<Suspense fallback=<div>Loading...</div>>
<OtherComponent /></Suspense>

## Server, Client and HTTP

*request:*
request line: http method, request url, http version
header, body
http method: get, post, put, delete, head
header: user-agent, referer, cookie, content-type, accept...
body: indicated by content-type
*response:* status line... (200 ok, 404 not found)
header: location, cache-control, expires, Etag,
set-cookie, content-type, content-disposition

## Nodejs

const express = require('express');
const app = express();
app.all('/*', function (req, res) {res.send('Hello!'); });
const server = app.listen(3000);
res.sendFile(absolutepath)
serving static files: app.use(express.static('public'));
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded(extended: false));
req.body['loginId']...

### Middleware

exec, change, end, start new
app.get('/*', (req,res) => res.send("Hello World!"));
const requestTime = (req,res,next) => {
// modifying the req object
next(); // passing on to the next middleware};
app.use(requestTime);
app.all('/*', (req,res) => {console.log("LOG: " + req.requestTime); res.send("Hello!"); });
const fs = require('fs'); const os = require('os');

## Cookie, Session and Storage

*HTTP cookies* are data which a server-side script sends to a web client to persist for a period of time.
Cookies are embedded in HTTP headers.
Cookies are stored on the client device within the browser.
On every subsequent HTTP request, the web client automatically sends the cookies back to server.
for Personalization, Session Management, Tracking
drawback: seen by others, increase header size
expiration: expires (in GMT), max-age (have precedence)
scope: domain, path (default to /)
security: secure, httpOnly, signed
res.cookie(name, value, [options])
*Session*: the first time a web client visits a server, the server sends a unique session ID to the web client for the client to keep.
const session = require('express-session');
app.use(session({ secret: 'foobarbazz', cookie: { maxAge: 1200000 }}));
*client side web storage*: in k-v pair
window.localStorage, window.sessionStorage
Storage.setItem(k,v), getItem(k), removeItem(k), clear()

## Security

validation, verification, authentication, authorization
10 risks: broken access control, crypto graphic failures, injection, insecure design, security misconfiguration, vulnerable and outdated components, identification and authentication failures, software and data integrity failures, security logging and monitoring failures, server-side request forgery
http secure: authentication, encryption(ciphertext)
certificates: (certificate authority) domain verification, organization verification, extended verification
DDOS: distributed denial-of-service attack
layer 7: application layer(http)
layer 3 or 4: protocol attack (syn flood), volumetric attack (dns amplification)
Exhausting the resource
Password: hashing (salting), encryption (peppering)
http authorization schemes: basic, bearer(encrypted token), digest, HOBA (http origin-bound authentication)
JWT: JSON web tokens (header.payload.signature)
injection: input becomes code -> validated, sanitized
XSS: cross-site scripting
stored (snippet is inserted into input box), reflected (query string of url), DOM based (HTML fragment #)
corss-site threats: cross-site request forgery, clickjacking, cross-origin resource sharing (CORS)
CORS: different protocol, host, port
servers inform clients, clients preflight requests before sending

## *MongoDB*

const mongoose = require('mongoose');

mongoose.connect(url);

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function () { console.log("done"); });
const Schema = mongoose.Schema;

const UserSchema = Schema({k:v...});
type, required: boolean, unique: boolean
create, findOne, find, count, update, remove
User.find({ name: 'John' }, (err, results) => { ... });