HTML + CSS + JS SL LIJIA XU

Animations

container → relative (2 /)
box → absolute

```
var pos = 0
var box = document.getElementById("box");
var t = setInterval(move, 10);
function move(){
  if(pos >= 150){
    clearInterval(t);
  } else{
    pos += 1;
    box.style.left = pos + "px";
  }
}
```

common events

```
onclick     onblur
onload      onfocus
onunload
onchange
onmouseover
onmouseout
onmousedown
onmouseup
```

event handler can be in line or assigned to elements

```
var x = document.getElementById("demo");
x.onclick = function(){
  document.body.innerHTML = Date();}
```

```
<button onclick="show()">
click me
</button>
<script>
  function show(){
    alert("Hi there");
  }
</script>
```

```
<body onload="doSome()">
window.onload = function(){}
```

onchange → text changes and focus lost from the element

```
<input type="text" id="name"
  onchange="change()">
<script>
  function change(){
    var x = document.getElementById(
    "name");
    x.value = x.value.toUpperCase();
  }
</script>
```

```
element.addEventListener(event, function, useCapture);
```

```
<button id="demo">Start</button>
<script>
  var btn = document.getElementById("demo");
  btn.addEventListener("click", myFunction);
  function myFunction(){
    alert(Math.random());
    btn.removeEventListener("click", myFunction);
} </script>
```

Event Propagation

bubbling → innermost element's event first
capturing → outtermost element's event first

ECMAScript 6 → a scripting language created to standardize JS

```
var a = 10;
const b = "hello";
let c = true;
```

Before ES6:

```
let name = 'David';
let msg = 'welcome' + name;
console.log(msg);
```

ES6:

```
let name = 'David';
let msg = `welcome${name}`;
console.log(msg);
```

loops before ES6

```
let arr = [1, 2, 3];
for(let k=0; k<arr.length;
k++){
  console.log(arr[k]);
}
```

Do Not use for...in for Arrays → only enumerable keys

```
let obj = {a:1, b:2, c:3};
for(let v in obj){ }
```

ES6: Map, Set, WeakMap, WeakSet

```
let list = ["x", "y", "z"];
for(let val of list){
  console.log(val);
}
```

```
for(let ch of "Hello"){
  console.log(ch)
}
```

before ES6:

```
function add(x, y){
  var num = x+y;
  console.log(sum);
}
```

ES6:

```
const add(x, y)=>{
  let sum = x+y;
  console.log(sum);
}
```

ES6 shorter version

```
const greet = x => "Welcome" + x;
```

No parameters ⇒ `const greet = () => alert("Hi");`

Before ES6:

```
var arr = [2, 3, 7, 8];
arr.forEach(function(el){
  console.log(el*2);
});
```

ES6:

```
const arr = [2, 3, 7, 8];
arr.forEach(v => {
  console.log(v*2);
});
```

functions like for...of Not working for older browsers

ES6 Array Destructing

```
let arr = ['1', '2', '3'];
let ["one", "two", "three"] = arr;
console.log(two);
```

```
"1" + 2 //12
"1" * 2 // 2
```

ES6 object destructing: unpacks properties into distinct variables

```
let a, b, c=4, d=8;
[a, b=6] = [2]; //a=2, b=6
[c, d] = [d, c]; //c=8 d=4
```

```
let obj = {h:100,
  s:true};
let {h, s} = obj;
```

keep h, s same

```
let a=() =>{
  return [1,3,2];
};
let [one, , two] = a();
// can skip one or more
```

without declaration, must have ()

```
let a, b;
({a, b} = {a:'A', b:'B'});
console.log(a+b) //AB
```

```
let {a,b} = {a:"A", b:"B"};
console.log(a+b);
```

assign with new names:

```
var o = {a:"A", b:"B"};
var {a: new, b: good} = o;
//new → "A", good → "B"
```

assign default values in case value unpacked is undefined

```
var obj = {id:42, name:
"A"};
// order Not matter
let {id=5, age=20} = obj;
```

before ES6: we can pass any number of arguments to the func and access it with arguments

ES6: rest parameter ...nums → spread operator

Spread operator ... args

```
function containsAll(arr, ...nums){
  for( let num of nums){
    if (arr.indexOf(num) === -1){
      return false;
    }
  return true;
}
```

before ES6:

```
function myFunc(w, x, y, z){
  console.log(x+y+z+w);}
var args = [1, 2, 3];
myFunc.apply(null,
args.concat(4));
```

with ES6:

```
const myFunc=(w, x, y, z)=>{
  console.log(w+x+y+z);};
let args = [1, 2, 3];
myFunc(...args, 4);
```

EX.

```
var dateFields = [1970, 0]
var date = new Date(...d-
-ateFields);
```

Spread in array literals

before ES6:

```
var arr = ["One", "Two", "Five"];
arr.splice(2, 0, "Three");
```

delete amount

with ES6:

```
let newArr = ['Three',
'Four'];
let arr = ['One', ...newArr];
```

spread in object literals

```
const obj1 = {a:"A", b:"B"};
const obj2 = {c:"C", d:"D"};
const clone1 = {...obj1};
const merged = {...obj1, ...obj2};
//{a:"A", b:"B", c:"C", d:"D"}
```

shallow cloning or meging object with Object.assign()

Class in ES6: only one constructor for each class

```
class Rectangle{
  constructor(height, width){
    this.height = height;
    this.width = width;
  }
}
```

prototype method

```
get area(){
return this.calcArea();
}
```

```
const square = new Rectangle(5,5);
```

```
static distance(a,b){
  const dx = a.x - b.y;
  const dy = a.y - b.y;
  return Math.hypot(dx, dy);
} // only called directly with class name
```

```
class Dog extends Animal{ }
super.speak();
```

ES6 Map key/value pairs

Map supports different data types
"1" and 1 are two different

keys for Map can be any type even objs

map.size → number of pairs in Map

```
set(key, value)
get(key)
has(key)
delete(key)
clear()
keys()
values()
entries()
```

```
let map = new Map();
map.set('k1', 'v1').set('k2', 'v2');
console.log(map.get('k1'));
console.log(map.has('k2'));
for(let kv of map.entries())
  console.log(kv[0] + ":" + kv[1]);
```

ES6: Set used to hold unique values

```
set.size
add(value)
delete(value)
has(value)
clear()
values()
```

```
let set = new Set()
set.add(5).add(9).add(15);
console.log(set.has(9));
for(let v of set.values())
  console.log(v);
```

ES6: Promises

A better way for asynchronous

before ES6:

```
setTimeout(function(){
  console.log("work1");
  setTimeout(function(){
    console.log("work2");
  }, 1000);
}, 1000);
console.log("End");
```

with ES6:

```
new Promise(fun-
-ction(resolve,
reject){
  // work
  if(sucess)
    resolve(result);
  else
    reject(Error("fa-
    -ilure"));});
```

resolve → method for success
reject → method for failure
It returns a Promise use → then