Lecture date: 20 March 2017

JavaScript

* **Client-side scripting**
* Running a program in the environment of the browser
* Behavior of application
* **Why?**
  + CSS and HTML are passive, no computations, loops, etc.
  + With JS, applications are more powerful

**Associating scripts**

* **External linking**
  + <script type=’text/javascript’ src=’script.js’></script>
* **Embedded**
* **Inline script**
  + Utilize Event Handler (ex. console.log(“inline...“);
* Words that start with on ex. onclick

Process – associate 🡺 fetch 🡺 parse 🡺execute

**Types of Codes**

1. **Top level / global code** – outside the function, executes on the fly
2. **Function** – when function is invoked

* Externally linked scripts (Execution of script may be … )

1. **Defer** – fetch the script then continue rendering the HTML
2. **Async** – execution will be asynchronous

Process: Fetch script 🡺 continue rendering 🡺 if script is available/ready 🡺 compile 🡺 execute script 🡺 continue rendering

<noscript> - alternative content when browsers doesn’t support scripts

**\*Javascripts can be disabled in browsers since scripts can be security exploits**

**Interacting with content**

* **Window content**
  + Enable interaction with the document
  + **Examples:**
    - h.innerText
    - h.innerHTML – java <em> script </em>
    - h.outerHTML - <h1 id=’h’> Java <em> script </em> </h1>
    - h.lang - ‘en-US’
    - h[“lang”] – ‘en-UK’
    - h.setAttribute(‘data-extra’, ‘value’)
    - h.title – access attributes special for one/two property
    - h.style.color = value(‘red’)
      * p.style=’color:blue’
      * p.style.color=’red’
      * p.style.backgroundColor=’yellow’
      * p.style[‘color’]
    - h.className = ‘green-text’
    - h.style = ‘‘-> remove style
* **getElementBy ..**
  + - **getElementByID -** <p id=’ ’>
    - **getElementsByClassName -** <p class=’ ’>
    - **getElementsByName -** <a name=’ ‘>
    - **getElementsByTagName -** <a name= ‘ ‘>
    - **getElementsByNS**
* **querySelector ..**
* **document.querySelector(‘p’)**
* output: <p> loren ipsum </p>
* **document.querySelectorAll(‘p’)**
* output: [p, p, p, p, p]
* **document.querySelectorAll(‘h1+p’)**
  + - h1 immediately followed by p
* **matches**
* **p.matches(‘p’)**
* output: true
* **p.matches(‘body p’)**
* output: true
* **Window** – global object
  + - Examples:
      * window.navigator – browser vendor
      * window.screen.width – physical screen
      * window.document / document

**Dynamic content creation**

* **h1 – document.createElement(‘h1’)**
* output: <h1></h1>
* **document.body.appendChild(h1)**
* output: <h1></h1>
* **t1 = document.createTextNode(‘paningit’)**
* output: “paningit”

**h1.insertBefore(t1, t)**

* **t2 = document.createTextNode(‘pamalit’)**

**h1.replaceChild(t2,t)**

* **h1.removeChild(t1)**
* **document.append(‘abc’,’xyz’) –** experiment technology, may not be universally supported and may be remove in the future

Lecture date: 27 March 2017

**Programmatic construct of JS**

* **Script** – manipulate content
* **JS – imperative** 🡺 sequence of instructions are decided by programmer, variables are storage areas
* **Data type** – different values that can be associated and possible operations that can be applied
* Simple/Primitive
  + - * Boolean – control structure

|  |  |
| --- | --- |
| Falsy Values | Truthy values |
| Not really false but is equivalent to true | Not really true but is equivalent to false |
| 0 | Other numbers |
| Empty string | Other strings |
| Undefined |  |
| null |  |

* **In JS ..**
  + No differentiation between integer and float
    - 0b – binary/decimal
    - 0 – octal
    - 0x – hexa
* No char type
* ‘abc’, “xyz”, ‘c’ are strings
* Template literals
* **`** (back quote) – multiple lines
* String interpolation
  + a = 100; `price: ${a}` 🡺 `price: 100`
* **var** – keyword saying that you are declaring a variable
* Ex. var x = 10;
* Same with **function**
* Ex. function max();
* **Dynamically typed**
  + **Good:** flexibility
  + **Bad:** difficult to determine whether the program is doing correct or not

**ECMA Script**

* Ways to create data objects
* **Examples:** top-level code (outside function)
* a = 10;
* var b = 10;
* let c = 10;
* const d = 10;

**Standard Objects**

* **Functions**
  + Example:

function sayHello(){

console.log(‘hello’);

}

* **Abstraction** – name a method then call the method through its name
* **Function** = object
* **sayHello();** - invocation
* **sayHello** – referencing
* **anonymous function** – function without a name
  + - * Example:

(function(a,b){return a+b;})(10,20);

* Can throw anyting/try-and-catch
* **Sample codes:**

/\* add \*/

var add = new function(‘a,b’,’return a+b’);

var sum = add(10,20);

/\* subtract \*/

var subtract = function(a,b){

return a-b;

}

var difference = subtract(10,20);

/\* arrow syntax \*/

var multiply = (a,b) => {return a\*b;}

var product = multiply(10,20);

var divide = (a,b) => a/b;

var quotient = divide(20,10);

var triple = n => 3\*n;

var thrice = triple(5);

var zero = () => 0;

/\* recursive functions \*/

function factorial(n){

if (n<0){

throw “Invalid argument”;

} else if (n==0) {

return 1;

} else {

Return n\*factorial(n-1);

}

}

/\* nested function \*/

function outer(){

//codes here

function inner(){

//inner codes here

}

//some more codes here

}

/\* giving default values for variables \*/

function fn(a=10, b=20, c, d=40, e){

// codes here

}

/\* function rest parameters \*/

function fn(a, b, … others) {

console.log(`a=${a}`);

console.log(`b=${b}`);

console.log(`others=` ,others);

}

* **Arrays**
  + Dynamic in size
  + **Sample codes:**

var emptyArray = new Array();

var alsoEmptyArray = [];

var arrayWithLengthFive = new Array(5);

var arrayWithOneElementWithValueFive = [5];

var array = new Array(‘5’) – number of elements

var sameArray = [‘5’];

var multiDimArray = [

[‘apple’, ‘banana’, ‘orange’],

[4, 5, 6, [true, false, false], 7, 8],

[]; **// I don’t know how to close this array**

/\* array destructing..? \*/

var array=[1,2,3,4,5];

var [a,b,c,d,e] = array;

var [m,n,… others] = array;

var [, x ,, y] = array;

/\* array indices can be non-contiguous \*/

var array=[1,2,3,4,5]

array[10]=10;

for(let i in array){

console.log(i, array[i]);

}

* **Methods**
  + - **Mutator methods**
      * Change the value of array
      * Example:
        + reverse()

a = [1,2,3]

a.reverse();

🡺 [3,2,1]

* sort()

Ex. (v1,v2) /\* if v1<v2 then -, if v1>v2 then +, else empty set \*/

* splice()

splice(2, 1, 1, 2, 3)

🡺index, 1 element, \*add these elements

* **Accessor methods**
  + - * .join(“ <li></li> “)
  + **Heration methods**
    - * a.filter(function)

🡺 value, index, array

a = [1,5,4,8,9,10,12,4,6,7,9]

a.filter(v => v%2 == 0)

🡺 [4,8,10,2,4,6]

* a.every(v => v>0)

🡺 true

a.every(v => v%4 == 0)

🡺 false

* .forEach() – do it for each element
* .reduce – accumulator, v, i, a
  + Example:

a.reduce ((acc, v) => acc += v)

🡺 75 (sum of all elements)

* **Objects**
  + - Prototype-base
    - **Sample codes:**

var emptyObj = new object();

var alsoEmpty = {};

var student = {};

student.idno = 121;

student[‘name’] = ‘JDC’;

student.toString = function(){

return `${this.idno}. ${this.name}`;

}

console.log(JSON.stringify(student, null, 2);

S = localStorage.getItem(‘213134’);

* **Constructor**
  + ­function in ECMA script = class
  + **Sample code:**

function Person(name, age){

this.name = name;

this.age = age;

this.speak = function(){

console.log(`Hello! I’m ${this.name}`);

}

}

p = new Person(‘ako’, 100)

p1 = {name: ‘ikaw’, age:20}