Pemrograman Berbasis Web

Pertemuan 6

Nori Wilantika
Lya Hulliyyatus Suadaa
Yeni Rimawati

Politeknik Statistika STIS
Prodi DIV Komputasi Statistik

06

CLIENT-SIDE SCRIPTING (2)

Javascript;

FUNCTION

FUNCTION

Function definitions can take several different forms in JavaScript including:

```
function identifier(param1,param2, ...) {
  statements  }

identifier = function(param1,param2, ...) {
  statements  }
```

- Such function definitions are best placed in the head section of an HTML page or in a library that is then imported
- Function names are case-sensitive
- The function name must be followed by parentheses
- A function has zero, one, or more parameters that are variables
- Parameters are not typed
- The **return** statement can be used to terminate the execution of a function and to make the return value of the function
- Function can contain more than one return statement
- Different return statements can return values of different types. There is no return type for a function.

CALLING A FUNCTION

A function is called by using the function name followed by a list of arguments in parentheses

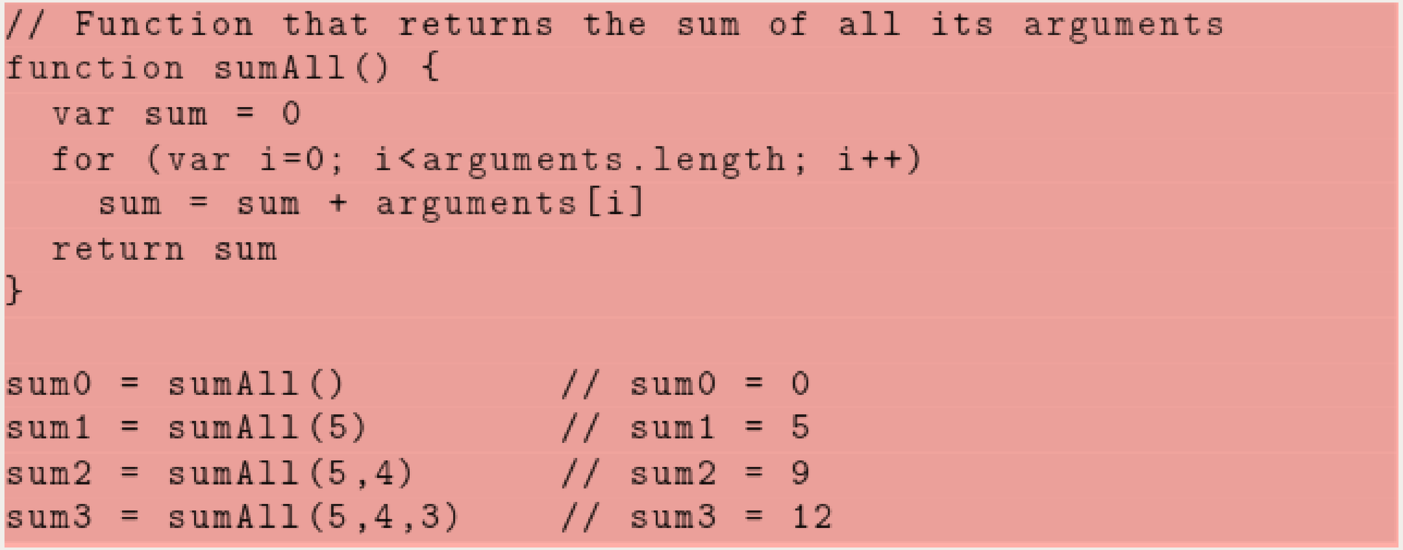
```
function identifier(param1, param2, ...) {
  ...
}
... identifier(arg1, arg2,...) ... // Function call
```

- The list of arguments can be shorter as well as longer as the list of parameters
- If it is shorter, then any parameter without corresponding argument will have value undefined

```
function sum(num1,num2) { return num1 + num2 }

sum1 = sum(5,4)       // sum1 = 9
sum2 = sum(5,4,3)     // sum2 = 9
sum3 = sum(5)         // sum3 = NaN
```

ARGUMENT LISTS

Every JavaScript function has a property called **arguments**. The **arguments.length** can be used to determine the number of arguments.



```
// Function that returns the sum of all its arguments
function sumAll() {
  var sum = 0
  for (var i=0; i<arguments.length; i++)
    sum = sum + arguments[i]
  return sum
}

sum0 = sumAll()          // sum0 = 0
sum1 = sumAll(5)         // sum1 = 5
sum2 = sumAll(5,4)       // sum2 = 9
sum3 = sumAll(5,4,3)     // sum3 = 12
```

FUNCTIONS AS ARGUMENTS

JavaScript functions are objects and can be passed as arguments to other functions

```
function apply(f,x,y) {
  return f(x,y)
}
function mult(x,y) {
  return x * y
}
console.log('2 * 3 =',apply(mult,2,3))
```

```
2 * 3 = 6
```

```
console.log('2 + 3 =',
            apply(function(a,b) { return a + b },
            2,3))
```

```
2 + 3 = 5
```

IMMEDIATELY-INVOKED FUNCTION EXPRESSION (IIFE)

- IIFE: function expression yang langsung dipanggil saat pembuatannya.

```
(() => {
  // some initiation code
  let firstVariable;
  let secondVariable;
})();

// firstVariable and secondVariable will be discarded after the
// function is executed.
```

```
var b=2
(function(){
  b=3
})()
console.log(b)//=> 3
```

SCOPE

SCOPE

- Global Scope
- Local Scope
- Block Scope

GLOBAL SCOPE

- A variable declared outside a function, becomes GLOBAL.

- A global variable has global scope: All scripts and functions on a web page can access it.

- If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable, even if the value is assigned inside a function.

```
myFunction();

// code here can use carName

function myFunction() {
  carName = "Volvo";
}
```

GLOBAL SCOPE

In JavaScript, a variable can be declared after it has been used. In other words, a variable can be used before it has been declared.

```
x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                     // Display x in the element

var x; // Declare x
```

```
var x; // Declare x
x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x;                     // Display x in the element
```

LOCAL SCOPE

- Each function creates a new scope.
- Scope determines the accessibility (visibility) of these variables. In JavaScript there are two types of scope: Local Scope and Global Scope
- Variables defined inside a function are not accessible (visible) from outside the function.
- Variables declared within a JavaScript function, become LOCAL to the function. Local variables have Function scope: They can only be accessed from within the function.
- variables with the same name can be used in different functions.
- The lifetime of a JavaScript variable starts when it is declared. Local variables are created when a function starts, and deleted when the function is completed.

BLOCK SCOPE

- area yang dicakup oleh ***scope*** yang ada di dalam ***statement if-else*** atau ***looping***, atau bisa juga kita sebut dengan ***block statement***.

- ***block scope*** di JavaScript tidak menciptakan ***local scope*** baru di dalam ***scope*** yang ditempati sekarang.

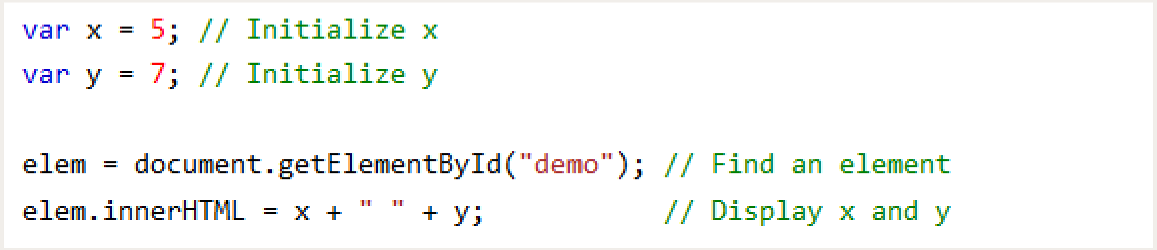
- ***Scoping*** di dalam ***if-else*** dan ***looping*** masih satu tempat dengan posisi didalam ***local/global scope*-nya** ditempatkan.

CONTOH SCOPE

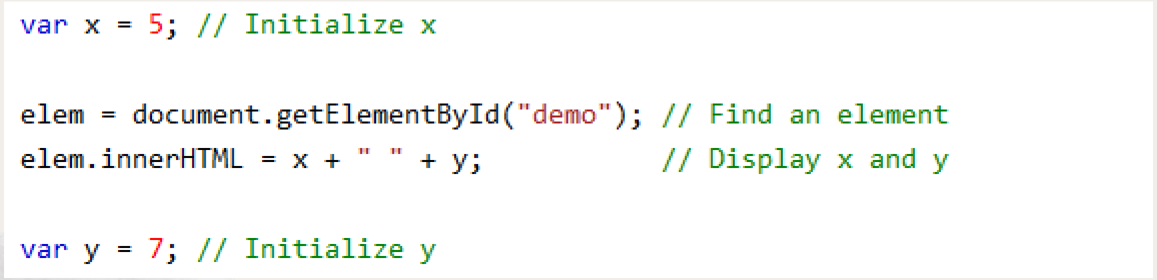
```
var x = 20; //global scope

function contohScope() {
  var x = 30; //local scope
  console.log(x); // 30

  if(true) {
    // kita tidak bisa mendeklarasikan ulang x untuk block statement
    // karena masih dianggap scope yang sama dengan function contohScope
    x = 40;
    console.log(x); // 40
  }
  console.log(x); // 40
}

contohScope();
console.log(x); // 20
```

HOISTING

- Hoisting is JavaScript's default behavior of moving declarations to the top.
- JavaScript moves all declarations to the top of the current scope (to the top of the current script or the current function).
- JavaScript only hoists declarations, not initializations.



```
var x = 5; // Initialize x
var y = 7; // Initialize y

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x + " " + y;           // Display x and y
```



```
var x = 5; // Initialize x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x + " " + y;           // Display x and y

var y = 7; // Initialize y
```

HOISTING

```
x = "Hello"
function f1() {
  console.log("1: " + x)
}
function f2() {
  console.log("2: " + x)
  x = "Bye"
  console.log("3: " + x)
}
f1()
f2()
console.log("4: " + x)
```

```
1: Hello
2: Hello
3: Bye
4: Bye
```

```
x = "Hello"
function f1() {
  console.log("1: " + x)
}
function f2() {
  console.log("2: " + x)
  var x = "Bye"
  console.log("3: " + x)
}
f1()
f2()
console.log("4: " + x)
```

```
1: Hello
2: undefined
3: Bye
4: Hello
```

HOISTING

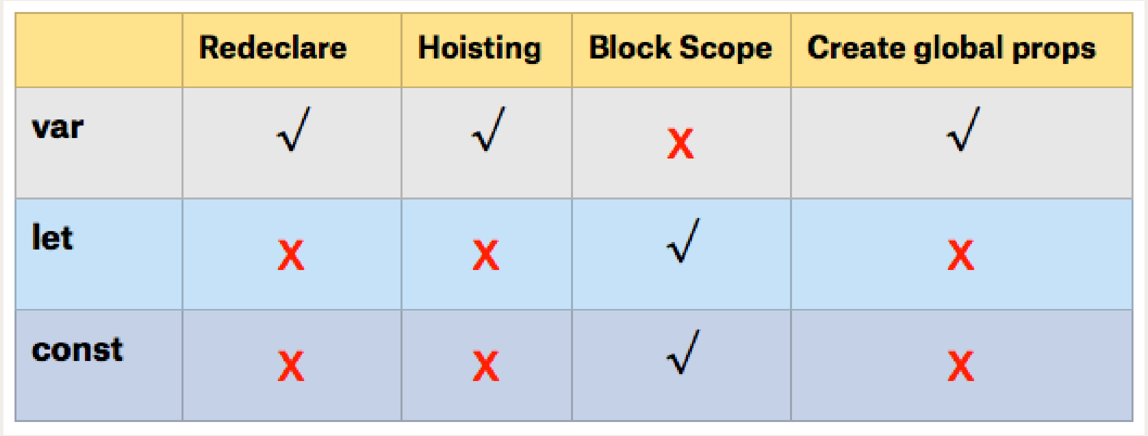
```
x = "Hello"


function f3(x) {
  x += ’!’
  console.log("1: " + x)
}
f3(’Bye’)
console.log("2: " + x)
f3(x)
console.log("2: " + x)
```

```
1: Bye!
2: Hello
1: Hello!
2: Hello
```

Variable

Sejak ES6, Javascript memperkenalkan **let** dan **const** untuk urusan variabel.

- gunakan ***let*** hindari ***var***
- gunakan ***const*** untuk variabel yg nilainya statis atau tidak berubah



	Redeclare	Hoisting	Block Scope	Create global props
var	√	√	X	√
let	X	X	√	X
const	X	X	√	X

Permasalahan dengan var

- Problem 1: Duplikasi nama variabel

```
var foo ='hello1'
var foo ='hello2'
console.log(foo) //=> hello
```

tidak ada pesan error sama sekali ketika terjadi duplikasi variabel.

- Problem 2: Hoisting

```
x=10
var x
console.log(x) //=> 10
```

sering membuat bingung

Permasalahan dengan var

- Problem 3: Scope

```
var b=2
if (true){
  var b=3
}
console.log(b) //=> 3
```

var ternyata menjadi global variabel, meskipun ada di dalam scope

Sebelum ES6, solusinya yaitu membuat function scope:

```
var b=2
function myScope(){
 var b=3
}
myScope()
console.log(b) //=> 2
```

Atau dengan IIFE.

Permasalahan dengan var

- Problem 4: Lupa deklarasi var = global

```
(function(){
  b=3
})()
console.log(b)//=> 3
```

Variabel tanpa *deklarasi dan* langsung di *assign* akan di anggap menjadi global variabel.

Let

- Cara kerja let hampir sama dengan var, bedanya di scope.
- var adalah function scope, sedangkan let adalah block scope.

- Mengatasi problem 1 pada var:

```
let foo ='hello1'
let foo ='hello2' //=> TypeError: Duplicate declaration "foo"
console.log(foo)
```

- Mengatasi problem 2 pada var:

```
x=10
let x
console.log(x) //=> ReferenceError: x is not defined
```

Let

- Mengatasi problem 3 pada var:

```
let b=2
if (true){
  let b=3
}
console.log(b) //=> 2
```

Const

- Const atau Konstanta: Variabel yang sifatnya tetap atau dengan kata lain nama dan isi dari variabel tidak bisa di ubah

```
const KEY=123
KEY = 456 //=> TypeError: Assignment to constant variable.
const KEY=123
var KEY = 456 //=> TypeError: Assignment to constant variable.
```

- Tetapi tidak berlaku untuk object dan array

```
const obj = { id:1, name:'jhon'}
obj.location="medan"
console.log(obj) //=> { id:1, name:'jhon',location:'medan'}
obj={} //=> TypeError: Assignment to constant variable.

const arr=[1,2,3,4]
arr.push(5)
console.log(arr) //=> [1,2,3,4,5]
arr=[] //=> TypeError: Assignment to constant variable.
```

ARRAY

ARRAY

- Array JavaScript digunakan untuk menyimpan banyak nilai dalam satu variabel.

- Jika Anda memiliki daftar item (daftar nama mobil, misalnya), menyimpan mobil dalam variabel tunggal akan terlihat seperti ini:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

- Menggunakan array literal adalah cara termudah untuk membuat Array JavaScript.

```
var  arrayVar  =  []
var  arrayVar  =  [elem0,  elem1,  ... ]
```

- Contoh:

```
var cars = new Array("Saab", "Volvo", "BMW");
```

ARRAY

- Array adalah **object**.

- Anda dapat memiliki variabel dengan tipe berbeda di Array yang sama. Anda dapat memiliki objek dalam Array. Anda dapat memiliki fungsi dalam Array. Anda dapat memiliki array dalam Array.

```
[2,'two',3,"three",1.1e1,true]
```

```
[2,[3,5],[7,[11]]]
```

- Each element has an index position, the first index position is 0.

```
[2,3,5,7,11]
 |  |  |  |  |
 0  1  2  3  4
```

ELEMEN ARRAY

- Mengakses elemen array: elemen array dapat diakses dengan mengacu pada nomor indeks.

```
var cars = ["Saab", "Volvo", "BMW"];
```

```
var name = cars[0];
```

- Mengubah elemen array:

```
cars[0] = "Opel";
```

- Dengan JavaScript, array lengkap dapat diakses dengan mengacu pada nama array

```
document.getElementById("demo").innerHTML = cars;
```

- The **length** property of an array returns the length of an array (the number of array elements).
- Arrays have no fixed length and it is always possible to add more elements to an array
- Accessing an element of an array that has not been assigned a value yet returns undefined

```
var array1 = [’hello’, [1, 2], function() {return 5}, 43]
console.log("1: array1.length = "+array1.length)
console.log("2: array1[3] ="+array1[3])
```

```
1: array1.length = 4
2: array1[3] = 43
```

```
array1[5] = ’world’
console.log("3: array1.length = "+array1.length)
console.log("4: array1[4] ="+array1[4])
console.log("5: array1[5] ="+array1[5])
```

```
3: array1.length = 6
4: array1[4] = undefined
5: array1[5] = world
```

```
array1.length = 4
console.log("6: array1[5] ="+array1[5])
```

```
6: array1[5] = undefined
```

```
var array2 = array1
array2[3] = 7
console.log("7: array1[3] ="+array1[3])
```

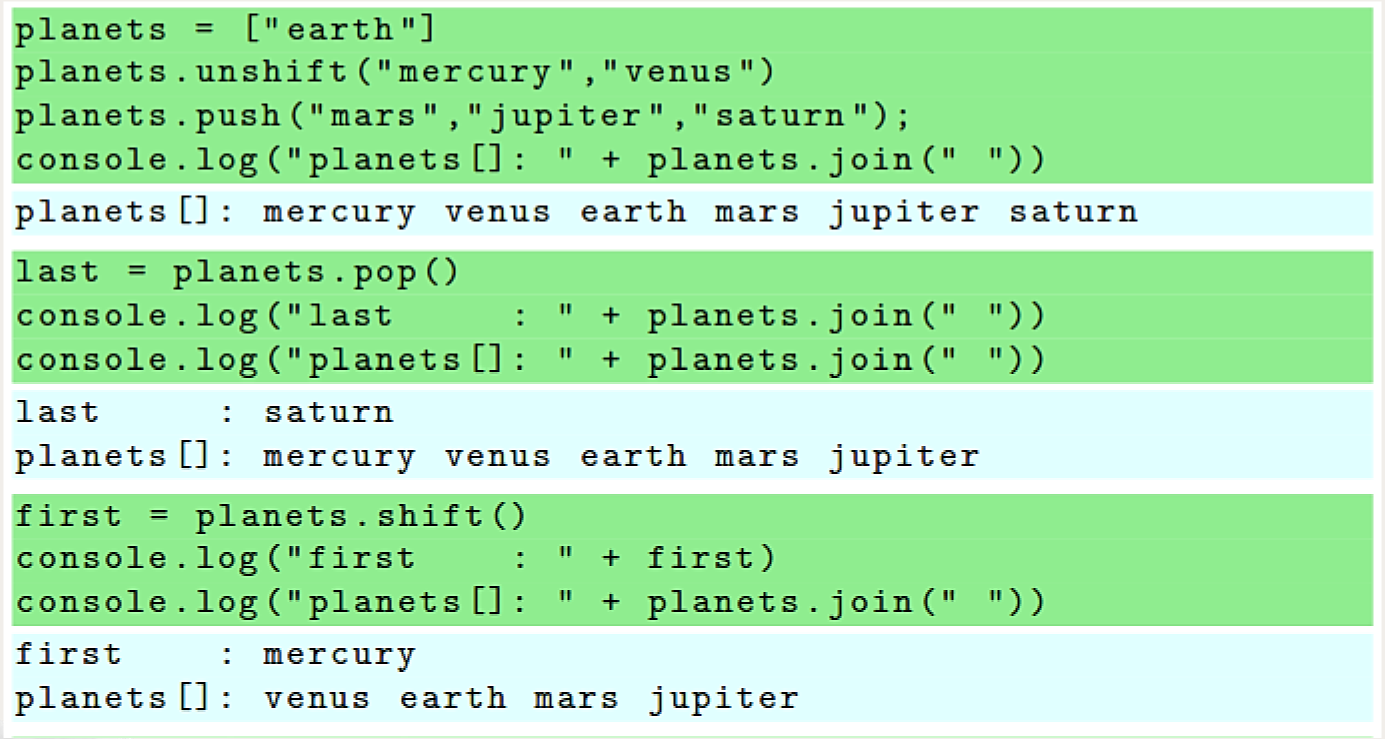
```
7: array1[3] = 7
```

ARRAY METHOD

- **pop( )** : menghilangkan elemen terakhir dari array. Return value: elemen yang dihilangkan.
- **push( )** : menambahkan elemen baru ke array (di akhir). Return value: panjang array yang
  baru.
- **shift( )** : menghilangkan elemen pertama array. Return value: elemen yang dihilangkan.
- **unshift( )** : menambahkan elemen baru ke array (di awal), Return value: panjang array yang
  baru.
- **splice( )**
- **slice( )**
- **toString( )**
- **join( )**

https://www.w3schools.com/jsref/jsref_obj_array.asp

CONTOH ARRAY



```
planets = ["earth"]
planets.unshift("mercury","venus")
planets.push("mars","jupiter","saturn");
console.log("planets[]: " + planets.join(" "))
```

```
planets[]: mercury venus earth mars jupiter saturn
```

```
last = planets.pop()
console.log("last     : " + planets.join(" "))
console.log("planets[]: " + planets.join(" "))
```

```
last     : saturn
planets[]: mercury venus earth mars jupiter
```

```
first = planets.shift()
console.log("first    : " + first)
console.log("planets[]: " + planets.join(" "))
```

```
first    : mercury
planets[]: venus earth mars jupiter
```

ARRAY ITERATION

- The recommended way to iterate over all elements of an array is a for-loop

```
for (index = 0; index < arrayVar.length; index++) {
     ... arrayVar[index] ...
}
```

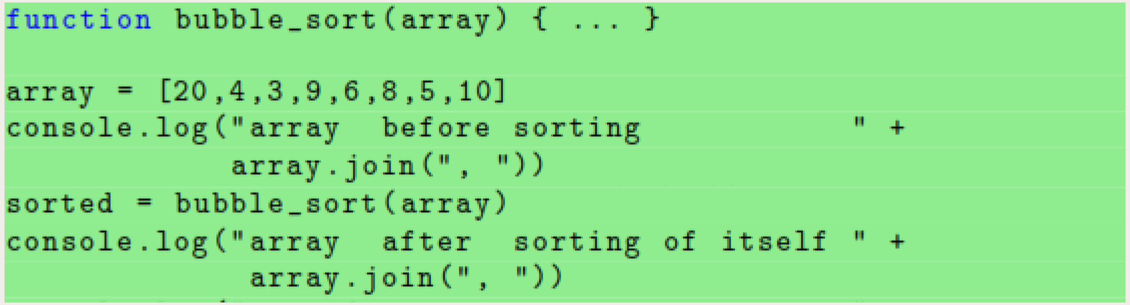
- An alternative is the use of the **forEach( )** method.

```
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```

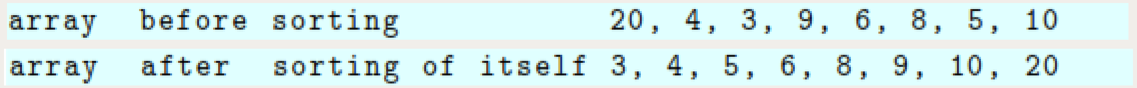
```
var fruits, text;
fruits = ["Banana", "Orange", "Apple", "Mango"];

text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
```

- The **forEach** method takes a **function** as an argument.
- **forEach( )** method calls the **function** once for each element in an array, in order.

ARRAY AND FUNCTION



```
function bubble_sort(array) { ... }

array = [20,4,3,9,6,8,5,10]
console.log("array  before sorting           " +
            array.join(", "))
sorted = bubble_sort(array)
console.log("array  after  sorting of itself " +
             array.join(", "))
```



```
array  before sorting           20, 4, 3, 9, 6, 8, 5, 10
array  after  sorting of itself 3, 4, 5, 6, 8, 9, 10, 20
```

ARRAY AND FUNCTION

```
  for (let i=0; i<array.length; i++) {
    for (let j=0; j<array.length-i; j++) {
      if (array[j+1] < array[j]) {
        // swap can change array because array is
        // passed by reference
        swap(array, j, j+1)
      }
    }
  }
  return array
}

function swap(array, i, j) {
  let tmp  = array[i]
  array[i] = array[j]
  array[j] = tmp
}
```

```
function bubble_sort(array) {
  function swap(i, j) {
    // swap can change array because array is
    // a local variable of the outer function bubble_sort
    let tmp = array[i];
    array[i] = array[j];
    array[j] = tmp;
  }
  for (var i=0; i<array.length; i++) {
    for (var j=0; j<array.length-i; j++) {
      if (array[j+1] < array[j]) swap(j, j+1)
  } }
  return array }
```

ARRAY AND COMPARISON

```
$array3 = ["1.23e2",5]
$array4 = ["12.3e1",true]
if ($array3 == $array4)
     console.log("The two arrays are equal")
else console.log("The two arrays are not equal")
```

Output: The two arrays are not equal

```
$array3 = ["1.23e2",5]
$array4 = ["1.23e2",5]
if ($array3 == $array4)
     console.log("The two arrays are equal")
else console.log("The two arrays are not equal")
```

Output: The two arrays are not equal

```
$array3 = ["1.23e2",5]
$array4 = ["12.3e1",true]
if (($array3[1] == $array4[1]) && ($array3[2] == $array4[2]))
     console.log("The two arrays are equal")
else console.log("The two arrays are not equal")
```

Output: The two arrays are not equal

OBJECT

OBJECT

In JavaScript, objects are king. If you understand objects, you understand JavaScript.

In JavaScript, almost "everything" is an object.

- Booleans can be objects (if defined with the new keyword)
- Numbers can be objects (if defined with the new keyword)
- Strings can be objects (if defined with the new keyword)
- Maths are always objects
- Dates are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are always objects

CREATING OBJECT

Object Literal

```
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

Object literal adalah daftar pasangan name:value (seperti age: 50) di dalam tanda kurung kurawal {}.

Keyword new

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

OBJECT CONSTRUCTOR

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.name = function() {return this.firstName + " " + this.lastName;};
}
```

It is considered good practice to name constructor functions with an upper-case first letter.

```
var myFather = new Person("John", "Doe", 50, "blue");
var myMother = new Person("Sally", "Rally", 48, "green");
```

PROPERTIES AND METHODS

- Accessing properties:

```
"My father is " + myFather.age + ". My mother is " + myMother.age + ".";
```

```
"My father is " + myFather['age'] + ". My mother is " + myMother['age'] + ".";
```

- The JavaScript **for...in** statement loops through the properties of an object. The block of code inside of the **for...in** loop will be executed once for each property.

```
var person = {fname:"John", lname:"Doe", age:25};
var x;
var txt = "";

for (x in person) {
  txt += person[x] + " ";
}
```

```
John Doe 25
```

- Accessing methods:

```
"My father is " + myFather.name();
```

ADDING AND DELETING

Adding properties (assumed that the person object already exists):

```
myFather.nationality = "English";
```

Deleting properties (assumed that the person object already exists):

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
delete person.age;   // or delete person["age"];
```

Adding a method to an object:

```
myFather.name = function () {
  return this.firstName + " " + this.lastName;
};
```

PROTOTYPE

You cannot add a new property or method to an object constructor the same way you add a new property or method to an existing object.

```
Person.nationality = "English";
```

```
Person.prototype.nationality = "English";
```

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}

Person.prototype.name = function() {
  return this.firstName + " " + this.lastName;
};
```

DATES

The **Date** object can be used to access the (local) date and time
The **Date** object supports various constructors

The **Date** object supports various **constructors**

newDate()	current date and time
newDate(milliseconds)	set date to milliseconds since 1 January 1970
newDate(dateString)	set date according to dateString
newDate(year, month, day, hours, min, sec, msec)	

Methods provided by Date include

toString()	returns a string representation of the Date object
getFullYear()	returns a four digit string representation of the (current) year
parse()	parses a date string and returns the number of milliseconds since midnight of 1 January 1970

https://www.w3schools.com/js/js_dates.asp

REGULAR EXPRESSION

Reguler Expression adalah urutan karakter yang membentuk pola pencarian.
Pola pencarian dapat digunakan untuk pencarian teks dan operasi penggantian teks.

Regular expressions can be used to perform all types of text search and text replace operations.
Regular expressions can make your search much more powerful, or more complicated pattern.

```
if (/[^a –zA -Z0 -9_ -]/. test ( field ))
  return " Invalid character in username"

if /[^ a -zA -Z0 -9\.\@\_ -]/. test ( field ))
  return " Invalid character in username"
```

RegExp Pattern: ***https://www.w3schools.com/jsref/jsref_obj_regexp.asp***

RegExp Methods: **test()**, **exec()**

BEST PRACTICE

- Hindari/minimumkan penggunaan Global Variable. Contoh:

```
// Initiate counter
let counter = 0;

// Function to increment counter
function add() {
  counter += 1;
}

// Call add() 3 times
add();
add();
add();

// The counter should now be 3
```

Hati-hati, kode lain di dalam halaman dapat mengubah nilai counter

BEST PRACTICE

- Selalu deklarasikan local variable, jika tidak akan menjadi global variable
- Simpan deklarasi pada bagian atas kode dan inisialiasi variable ketika dideklarasikan
- Deklarasikan objek dan array dengan const
- Hati-hati dengan automatic type conversion
- Gunakan === pada saat comparison

```
0 == "";          // true
1 == "1";         // true
0 === "";         // false
1 === "1";        // false
```

- Gunakan default parameter dalam function untuk menghindari missing argument

```
function myFunction(x, y) {
  if (y === undefined) {
    y = 0;
  }
}
```

BEST PRACTICE

- Akhiri switch dengan default
- Hindari penggunaan Number, String, and Boolean sebagai Objects

```
let x = "John";
let y = new String("John");
(x === y) // is false because x is a string and y is an object.
```

```
let x = new String("John");
let y = new String("John");
(x == y) // is false because you cannot compare objects.
```

Referensi: https://www.w3schools.com/js/js_best_practices.asp

REFERENSI

Materi disadur dari:

1. COMP519 Web Programming (2020-21). Diakses pada Januari 2021, dari https://cgi.csc.liv.ac.uk/~ullrich/COMP519/
2. W3Schools Online Web Tutorials. Diakses pada Januari 2021, dari https://www.w3schools.com/

Thanks!

Do you have any questions?

wilantika@stis.ac.id
lya@stis.ac.id
yeni.rima@stis.ac.id