

Dr. Wittawin Susutti
wittawin.sus@kmutt.ac.th

CSS233 WEB PROGRAMMING I LECTURE 07 – JS#3

JavaScript Object

- In JavaScript, Objects may exist without Classes
 - Usually, Objects are created directly, without deriving them from a Class definition
- In JavaScript, Objects are dynamic
 - You may add, delete, redefine a property at any time
 - You may add, delete, redefine a method at any time
- In JavaScript, there are no access control methods
 - Every property and every method is always public (private/protected don't exist)
- There is no real difference between properties and methods

Object

- An object is an unordered collection of properties
 - Each property has a name (key), and a value
- You store and retrieve property values, through the property names
- Object creation and initialization:

```
let point = {x: 2, y: 5};  
let book = {  
  author: "Enrico",  
  title: "Learning JS",  
  for: "students",  
  pages: 520  
};
```

```
Object literals syntax:  
{"name": value,  
 "name": value}  
or:  
{name: value,  
 name: value}
```

Object Properties

Property names are:

- Identified as a string
- Must be unique in each object ^{ต้องไม่ซ้ำกัน}
- Created at object initialization
- Added after object creation
 - With assignment
- Deleted after object creation
 - With `delete` operator

Property values are:

- Reference to any JS value
- Stored inside the object
- May be arrays, other objects, ...
 - Beware: the object stores the reference, the value is outside
- May also be functions (methods)

Accessing properties

- Dot (.) or square brackets [] notation
- The . dot notation and omitting the quotes are allowed when the property name is a valid identifier, only.

`book.title` ^{ใช้บ่งชี้} or `book['title']`
`book['my title']` and not `book.my title`

```
let book = {  
  author : "Enrico",  
  title : "Learning JS",  
  for: "students",  
  pages: 340,  
  "chapter pages": [90,50,60,140]  
};
```

^{ใช้. หรือ}

```
let person = book.author;  
let name = book["author"];  
let numPages = book["chapter pages"];  
book.title = "Advanced JS";  
book["pages"] = 340;
```

^{เปลี่ยนค่าใน obj}

Objects as associative arrays

- The [] syntax looks like array access, but the index is a string
 - Generally known as associative arrays
- Setting a non-existing property creates it:

```
person["telephone"] = "0110901234";
```

```
person.telephone = "0110901234";
```

- Deleting properties

```
delete person.telephone;
```

```
delete person["telephone"];
```



Computed property names

- Flexibility in creating object properties
 - `{[prop]:value}` - creates an object with property name equal to the value of the variable `prop`
 - `[]` can contain more complex expressions: e.g., i-th line of an object with multiple "address" properties (address1, address2, ...): `person["address"+i]`
 - Using expressions is not recommended...
- Beware of quotes:
 - `book["title"]` - property called title
 - Equivalent to `book.title`
 - `book[title]` - property called with the value of variable `title` (if exists)
 - If `title=="author"`, then equivalent to `book["author"]`
 - No equivalent in dot-notation

Property access errors

- If a property is not defined, the (attempted) access returns undefined
- If unsure, must check before accessing
 - Remember: undefined is falsy, you may use it in Boolean expressions

↳ if false, undefined, false, null, etc. are logical

```
let surname = undefined;
if (book) {
  if (book.author) {
    surname = book.author.surname;
  }
}
```


Iterating over properties

- `for .. in` iterates over the properties

iterate property names

```
for(let a in {x: 0, y:3}) {  
  console.log(a) ;  
}
```

OP

x
y

```
let book = {  
  author : "Enrico",  
  pages: 340,  
  chapterPages: [90,50,60,140],  
};
```

```
for (const prop in book)  
  console.log(`${prop} = ${book[prop]}`);
```

OP

```
author = Enrico  
pages = 340  
chapterPages = 90,50,60,140
```

Iterating over properties

- All the (enumerable) properties names (keys) of an object can be accessed as an array, with:

```
let keys = Object.keys(my_object);
```

- All pairs [key, value] are returned as an array with:

```
let keys_values = Object.entries(my_object)
```

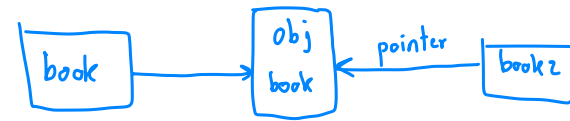
```
['author', 'pages']
```

```
[['author', 'Enrico'], ['pages', 340]]
```

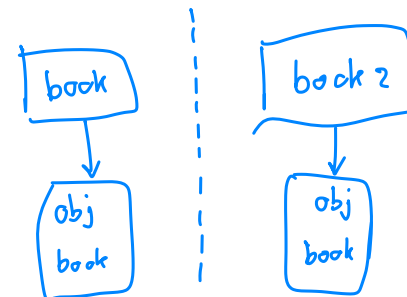
Copying objects

```
let book = {  
  author : "Enrico",  
  pages: 340,  
};  
let book2 = book; // ALIAS
```

reference



```
let book = {  
  author : "Enrico",  
  pages: 340,  
};  
let book3 = Object.assign({}, book); // COPY
```



Object.assign

เก็บไว้ target



```
let new_object = Object.assign(target, source);
```

- Assigns all the properties from the source object to the target one
- The target may be an existing object
- The target may be a new object: { }
- Returns the target object (after modification)

Beware! Shallow copy, only

```
let book = {  
  author : "Enrico",  
  pages: 340,  
};  
let study = {  
  topic: "JavaScript",  
  source: book,  
};  
  
let study2 = Object.assign({}, study);
```

Obj → Obj

source → pointer to book

Merge properties (on existing object)

- `Object.assign(target, source, default values, ...);`

```
let book = {  
  author : "Enrico",  
  pages: 340,  
};
```

เพิ่มค่าไป ใน obj ที่ก้มี

```
let book2 = Object.assign( book, {title: "JS"} );
```

Merge properties (on new object)

- `Object.assign(target, source, default values, ...);`

```
let book = {  
  author : "Enrico",  
  pages: 340,  
};
```

```
let book2 = Object.assign(  
  {}, book, {title: "JS"}  
);
```

כש'אנרש' מנל' אוב' ר'ומד'ן.

Copying with spread operator (ES9 – ES2018)

```
let book = {  
  author : "Enrico",  
  pages: 340,  
};
```

```
let book2 = {...book, title: "JS"};  
let book3 = { ...book2 } ;  
console.log(book2);
```

```
{ author: 'Enrico', pages: 340, title: 'JS' }
```

← สรุปรวตัวแปรอื่นๆ

```
const {a,b,...others} =  
{a:1, b:2, c:3, d:4};
```

```
console.log(a);  
console.log(b);  
console.log(others);
```

```
1  
2
```

```
{ c: 3, d: 4 }
```


Checking if properties exist

- Operator `in`
 - Returns `true` if property is in the object.

```
let book = {  
  author : "Enrico",  
  pages: 340,  
};  
  
console.log('author' in book);  
delete book.author;  
console.log('author' in book);  
  
true  
false
```

Object creation (equivalent methods)

- By object literal:

```
const point = {x:2, y:5};
```

- By object literal (empty object):

```
const point = {};
```

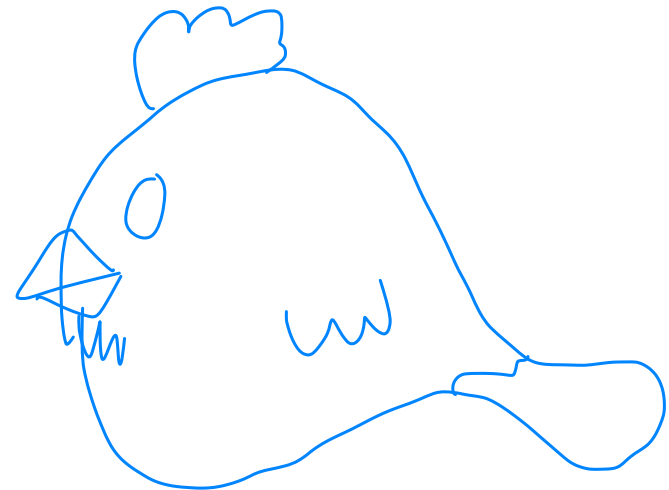
- By constructor:

```
const point = new Object();
```

- By object static method create:

```
const point = Object.create({x:2,y:5});
```

- Using a constructor function



Functions

- One of the most important elements in JavaScript
- Delimits a block of code with a private scope
- Can accept parameters and returns one value
 - Can also be an object
- Functions themselves are objects in JavaScript
 - They can be assigned to a variable
 - Can be passed as an argument
 - Used as a return value

Classic functions

```
function do(params) {  
    /* do something */  
}
```

```
function square(x) {  
    let y = x * x ;  
    return y ;  
}
```

```
let n = square(4) ;
```

Parameters

- Comma-separated list of parameter names
 - May assign a default value, e.g., `function(a, b=1) {}`
- Parameters are passed by-value
 - Copies of the reference to the object
- Parameters that are not passed in the function call get the value 'undefined'
- Check missing/optional parameters with:
 - `if(p===undefined) p = default_value ;`
 - `p = p || default_value ;`

Variable number of parameters

- Syntax for functions with variable number of parameters, using the ... operator (called “rest”)

```
function fun (par1, par2, ...arr) { }
```

- The “rest” parameter must be the last, and will deposit all extra arguments into an array

```
function sumAll(initVal, ...arr) {  
  let sum = initVal;  
  for (let a of arr) sum += a;  
  return sum;  
}
```

```
sumAll(0, 2, 4, 5); // 11
```



A blue line connects the first argument '0' in the function call to the 'initVal' parameter in the function definition. Another blue line connects the array of arguments [2, 4, 5] in the function call to the '...arr' parameter in the function definition.

Function expression: indistinguishable

- The expression `function() {}` creates a new object of type 'function' and returns the result.
- Any variable may "refer" to the function and call it.
- You can also store that reference into an array, an object property, pass it as a parameter to a function, redefine it, ...

```
const fn = function(params) {  
    /* do something */  
}  
  
const fn = function do(params) {  
    /* do something */  
}
```

```
function square(x) {  
    let y = x * x ; return y ;  
}
```

```
let cube = function c(x) {  
    let y = square(x)*x ;  
    return y ; }  
let n = cube(4) ;
```

Arrow Function: just a shortcut

```
const fn = (params) => {  
    /* do something */  
}
```

```
function square(x) {  
    let y = x * x ;  
    return y ;  
}
```

```
let cube = function c(x) {  
    let y = square(x)*x ;  
    return y ;  
}
```

```
let fourth = (x) => { return square(x)*square(x); }  
let n = fourth(4) ;
```


Parameters in arrow functions

```
const fun = () => { /* do something */ } // no params
const fun = param => { /* do something */ } // 1 param
const fun = (param) => { /* do something */ } // 1 param
const fun = (par1, par2) => { /* smtg */ } // 2 params
const fun = (par1 = 1, par2 = 'abc') => { /* smtg */ } // default values
```

Nested functions

- Function can be nested, i.e., defined within another function

```
function hypotenuse(a, b) {  
    const square = x => x*x ;  
    return Math.sqrt(square(a) + square(b));  
}  
  
function hypotenuse(a, b) {  
    function square(x) {  
        return x*x;  
    }  
    return Math.sqrt(square(a) + square(b));  
}
```

- The inner function is scoped within the external function and cannot be called outside
- The inner function might access variables declared in the outside function

Closure: definition

- A closure is a name given to a feature in the language by which a nested function executed after the execution of the outer function can still access outer function's scope.
- Really: one of the most important concepts in JS

Closures

- JS uses lexical scoping
 - Each new functions defines a scope for the variables declared inside
 - Nested functions may access the scope of all enclosing functions
- Every function object remembers the scope where it is defined, even after the external function is no longer active → Closure

```
"use strict" ;  
function greeter(name) {  
    const myname = name ;  
    const hello = function () {  
        return "Hello " + myname ;  
    }  
    return hello ;  
}  
  
const helloTom = greeter("Tom");  
const helloJerry = greeter("Jerry");  
console.log(helloTom());  
console.log(helloJerry());
```

Using closures to emulate objects

```
"use strict" ;  
function counter() {  
    let value = 0 ;  
    const getNext = () => {  
        value++;  
        return value;  
    }  
    return getNext ;  
}
```

```
const count1 = counter() ;  
console.log(count1()) ;    1  
console.log(count1()) ;    2  
console.log(count1()) ;    3  
const count2 = counter() ;  
console.log(count2()) ;    1  
console.log(count2()) ;    2  
console.log(count2()) ;    3
```

Using closures to emulate objects (with methods)

```
"use strict";
function counter() {
  let n = 0;
  return {
    count: function() {
      return n++; },
    reset: function() { n = 0; }
  };
}
```

```
let c = counter(), d = counter();
```

```
c.count() 0
```

```
d.count() 0
```

```
c.reset() 0
```

```
c.count() 1
```

```
d.count() 1
```

Construction functions

- Define the object type
 - Use a capital initial letter
 - Set the properties with the keyword `this`
- Create an instance of the object with `new`

```
function Car(make, model, year) {  
    this.make = make;  
    this.model = model;  
    this.year = year;  
    this.isNew = ()=>(year>2000);  
}
```

```
let mycar = new Car('Eagle', 'Talon TSi', 1993);
```

Homework

- Create a function `createBankAccount(initialBalance)` that accepts an initial balance. This function must **return** an object representing a bank account. It must have the following methods:
 - `getBalance()`: Returns the current account balance.
 - `deposit(amount)`: Adds the specified amount to the balance.
 - `withdraw(amount)`: Subtracts the specified amount from the balance.

Requirements:

- A `withdraw` operation must not succeed (it should do nothing, or log a message) if the withdrawal `amount` is greater than the current `balance`.
- Create a function `createItemManager()` that returns an object for managing a list of items (strings). The object should have the following methods:
 - `addItem(item)`: Adds the item (string) to the list, but only if it's not already in the list.
 - `removeItem(item)`: Removes the item from the list, if it exists.
 - `listItems()`: Returns an array containing all items currently in the list.

Requirements:

- `addItem` should prevent duplicate items from being added.