M01-H02

Function dan Struktur Data in JavaScript

Function, Obyek, Array, Iterables, Map dan Set, WeakMap dan WeakSet, Date dan Time

Function

```
function name([param[, param[, ... param]]]) {
   statements
}
```

- The concept of wrapping a piece of program in a value or block code.
- a way to structure larger programs, to reduce repetition, to associate names with subprograms, and to isolate these subprograms from each other.
- **Defining functions:**
  - The function declaration
  - declaration / anonymous function
  - generator function
  - arrow function

Ref : devzilla, eloquentjavascrip

Obyek and array as Data Structures

- Built from :
  - primitive type (Numbers, Booleans, strings ...),non Primitive type (Object, Array, function)
- Array : a list of values between square brackets separated by comma

  Syntax : let arrayName = [value1, value2 ...]

- Object : place to store keyed collections of various data and more complex entities. Syntax :

```
let user = new Object(); // "object constructor" syntax
let user = {prop1 : valu1, prop2: valu2 };  // "object literal" syntax
```

Obyek and array as Data Structures

- Object Method : Obect.keys, Object values, Object.entrys, Object.fromEntries(array)

```
let user = {
  name: "John",
  age: 30
};
Object.keys(user) // ["name", "age"]
Object.values(user) // ["John", 30]
Object.entries(user) // [ ["name","John"], ["age",30] ]
var x = [ ["prop1", 2] , ["prop2", 3]]
Object.fromEntries(x)
```

https://javascript.info/iterable

Iterables

The concept to makeany object useable in a loop

```
let range = {
  from: 1,
  to: 5
};
```

```
for (let num of range) {
  alert(num); // 1, then 2, 3, 4, 5
}
```

Builtin iterable : string

```
// ADD Iterable
// 1. call to for..of initially calls this
range[Symbol.iterator] = function() {

  // ...it returns the iterator object:
  // 2. Onward, for..of works only with this iterator, asking it for next values
  return {
    current: this.from,
    last: this.to,

    // 3. next() is called on each iteration by the for..of loop
    next() {
      // 4. it should return the value as an object {done:.., value :...}
      if (this.current <= this.last) {
        return { done: false, value: this.current++ };
      } else {
        return { done: true };
      }
    }
  };
};
```

Ref : jsinfo

Map and Set

- The Map object holds key-value pairs. Any value (both objects and primitive values) may be used as either a key or a value.
- keys are not converted to strings
- Methos and property : new Map(), map.set(key, value), map.get(key), map.has(key), map.delete(key),map.clear(), map.size
- Set objects : collections of values.

```
let recipeMap = new Map([
  ['cucumber', 500],
  ['tomatoes', 350],
  ['onion',    50]
]);

let map = new Map();

map.set('1', 'str1');   // a string key
map.set(1, 'num1');     // a numeric key
map.set(true, 'bool1'); // a boolean key

map.get(1)
map.get('1')
```

Ref : javascript.info

Map

	Map	Object
Key Types	A `Map`'s keys can be any value (including functions, objects, or any primitive).	must be either a `String` or a `Symbol`.
Key Order	The keys in `Map` are ordered. a `Map` object returns keys in order of insertion.	not ordered.
Performance	Performs better in scenarios involving frequent additions and removals of key-value pairs.	Not optimized for frequent additions and removals of key-value pairs.
Iteration	A `Map` is an iterable, so it can be directly iterated.	Iterating over an `Object` requires obtaining its keys in some fashion and iterating over them.

Ref : javascript.info

WeakMap and WeakSet

- WeakMap object is a collection of key/value pairs in which the keys are weakly referenced. The keys must t be objects and the values can be arbitrary values.

```
const wm1 = new WeakMap(), wm2 = new
WeakMap(), wm3 = new WeakMap();

const o1 = {}, o2 = function() {}, o3
= window; wm1.set(o1, 37); wm1.set(o2,
'azerty'); wm2.set(o1, o2); // a value
can be anything, including an object
or a function wm2.set(o3, undefined);
wm2.set(wm1, wm2); // keys and values
can be any objects. Even WeakMaps!
```

```
wm1.get(o2); // "azerty"
wm2.get(o2); // undefined, because
there is no key for o2 on wm2
wm2.get(o3); // undefined, because
that is the set value wm1.has(o2);
// true wm2.has(o2); // false
wm2.has(o3); // true (even if the
value itself is 'undefined')
wm3.set(o1, 37); wm3.get(o1); // 37
wm1.has(o1); // true wm1.delete(o1);
wm1.has(o1); // false
```

Ref : [javascript.info]

WeakSet

- WeakSet objects are collections of objects. Just as with Sets, each object in a WeakSet may occur only once; all objects in a WeakSet's collection are unique.

```
const ws = new WeakSet(); const foo = {};

const bar = {}; ws.add(foo);

ws.add(bar); ws.has(foo); // true

ws.has(bar); // true

ws.delete(foo); // removes foo from the set

ws.has(foo); // false, foo has been removed

ws.has(bar); // true, bar is retained
```

Ref : javascript.info

Date and time

- Let's meet a new built-in object: Date. It stores the date, time and provides methods for date/time management.
- we can use it to store creation/modification times, to measure time, or just to print out the current date

```
let now = new Date();
// 0 means 01.01.1970 UTC+0
//milliseconds
let Jan01_1970 = new Date(0);
let Jan02_1970 = new Date(24 * 3600 * 1000);
//from string init
let date = new Date("2017-01-26");
new Date(year, month, date, hours, minutes, seconds, ms)
```

- Method accessor : getFullYear(), getMonth(), getDate(), getHours(), getMinutes(), getSeconds(), getMilliseconds()

Ref : javascript.info

“Before software can be reusable it first has to be usable.” ~ Ralph Johnson ~

Don't waste your life without praying …….

Let get started