OBJECT BASICS

Seven data types called “primitive”, because their values contain only a single thing. object sama seperti array, yang menyimpan banyak nilai dalam 1 variabel, bedanya array diakses dengan index sedangkan object diakses dengan key.

An object can be created with figure brackets {…} with an optional list of properties. A property is a “key: value” pair, where key is a string/symbols (also called a “property name”), and value can be anything.

A. Syntax (check c. summary part 1)

An empty object can be created using one of two syntaxes:

```
let user = new Object(); // "object constructor" syntax
let user = {};  // "object literal" syntax
```

Property values are accessible using the dot notation:

```
// get property values of the object:
alert( user.name ); // John
alert( user.age ); // 30
```

```
let user = {
  name: "John",
  age: 30
};
```

To remove a property, we can use the delete operator:

```
delete user.age;
```

multiword key must be quoted:

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true
};
```

multiword key are only accessible using square bracket notation:

```
// set
user["likes birds"] = true;

// get
alert(user["likes birds"]); // true

// delete
delete user["likes birds"];
```

Square brackets allow to taking key from variable:

```
let user = {
  name: "John",
  age: 30
};

let key = prompt("What do you want to know about the user?", "name");

// access by variable
alert( user[key] ); // John (if enter "name")
```

The dot notation cannot be used in a similar way^^:

```
alert( user.key ) // undefined
```

most of the time, when property names are known and simple, the dot is used. And if we need something more complex, then we switch to square brackets.

- Computed Properties: use square brackets in an object literal

```
let fruit = prompt("Which fruit to buy?", "apple");

let bag = {
  [fruit]: 5, // the name of the property is taken
};

alert( bag.apple ); // 5 if fruit="apple"
```

^^That same as:

```
let fruit = prompt("Which fruit to buy?", "apple");
let bag = {};

// take property name from the fruit variable
bag[fruit] = 5;
```

- The last property in the list may end with a comma:

```
let user = {
  name: "John",
  age: 30,
}
```

That is called a "trailing" or "hanging" comma. Makes it easier to add/remove/move around properties, because all lines become alike.

- In real code, we often use existing variables as values for property names

```
function makeUser(name, age) {
  return {
    name: name,
    age: age,
    // ...other properties
  };
}

let user = makeUser("John", 30);
alert(user.name); // John
```

```
function makeUser(name, age) {
  return {
    name, // same as name: name
    age,  // same as age: age
    // ...
  };
```

- Property names limitations
  There are no limitations on property names, like "for", "let", "return" etc.

```
let obj = {
  for: 1,
  let: 2,
```

- 'in' operator (property existence test)
  Syntax:

```
"key" in object
```

  Example:

```
let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age exists
```

B. "for..in" loop

To walk over all keys of an object, there exists a special form of the loop: for..in.

Syntax:

```
for (key in object) {
  // executes the body for each key among object properties
}
```

Example:

```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // keys
  alert( key );  // name, age, isAdmin
  // values for the keys
  alert( user[key] ); // John, 30, true
```

- Integer properties are sorted (automatically)

```
let codes = {
  "49": "Germany",
  "41": "Switzerland",
  "44": "Great Britain",
  // ..,
  "1": "USA"
};

for (let code in codes) {
  alert(code); // 1, 41, 44, 49
}
```

Adding a plus "+" sign before each code to fix the issue

```
let codes = {
  "+49": "Germany",
  "+41": "Switzerland",
  "+44": "Great Britain",
  // ..,
  "+1": "USA"
};

for (let code in codes) {
  alert( +code ); // 49, 41, 44, 1
}
```

C. Summary part 1

Objects are associative arrays with several special features.

They store properties (key-value pairs), where:

- Property keys must be strings or symbols (usually strings).
- Values can be of any type.

To access a property, we can use:

- The dot notation: `obj.property`.
- Square brackets notation `obj["property"]`. Square brackets allow taking the key from a variable, like `obj[varWithKey]`.

Additional operators:

- To delete a property: `delete obj.prop`.
- To check if a property with the given key exists: `"key" in obj`.
- To iterate over an object: `for (let key in obj)` loop.

What we've studied in this chapter is called a "plain object", or just `Object`.

D. Object Methods

Object methods are actions that can be performed on objects.

A method is a **function definition** stored as a **property value**.

```
const person = {
  name: "John",
  age: 30,
  job: "Developer",
  greet: function() {
    console.log("Hello, my name
  }
};
```

(hanya?) Object literal mendukung method.

- **This**

```
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
```

In the example above, `this` refers to the **person object**:

this.firstName means the **firstName** property of **person**.

this.lastName means the **lastName** property of **person**.

- Access Object Methods

```
objectName.methodName()
```

Example: `name = person.fullName();`

If you access the fullName property without (), it will return the function definition:

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

- Adding a method (same with adding property)

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

E. Intermediate/Advanced Array Method

1. Map(function(currentValue, index, arr), thisValue) : (creates new array)performing function to every array element. Example: (more in 5. Example)

```
function addOne(num) {
  return num + 1;
}
const arr = [1, 2, 3, 4, 5];
const mappedArr = arr.map(addOne);
console.log(mappedArr); // Outputs [2, 3, 4, 5, 6]
```

better if : `2 | const mappedArr = arr.map((num) => num + 1);`

2. Filter() : (creates new array)return (all) element that match the function.

```
function isOdd(num) {
  return num % 2 !== 0;
}
const arr = [1, 2, 3, 4, 5];
const oddNums = arr.filter(isOdd);
console.log(oddNums); // Outputs [1, 3, 5];
console.log(arr); // Outputs [1, 2, 3, 4, 5], original array
```

3. Reduce() : (creates new array) returns a single value(the function's accumulated result). This method is very useful for various operations such as adding values, multiplying values, combining objects, and so on.
   Syntax:

```
array.reduce(function(accumulator, currentValue, currentIndex, arr), initialValue)
```

- The callback function takes two arguments instead of one. The first argument is the `accumulator`, which is the current value of the result *at that point in the loop*. The first time through, this value will either be set to the `initialValue` (described in the next bullet point), or the first element in the array if no `initialValue` is provided. The second argument for the callback is the `current` value, which is the item currently being iterated on.
- It also takes in an `initialValue` as a second argument (after the callback), which helps when we don't want our initial value to be the first element in the array. For instance, if we wanted to sum all numbers in an array, we could call reduce without an `initialValue`, but if we wanted to sum all numbers in an array and add 10, we could use 10 as our `initialValue`.

Cara kerja reduce adalah loop through every element until the last element
- sebelum loop: accumulator merupakan parameter kedua dari reduce() ATAU elemen pertama dari array.
- setelah loop: accumulator merupakan hasil return dari loop sebelumnya ATAU hasil return loop saat ini(jika saat ini adalah loop terakhir).

Example: (more in 5. Example)

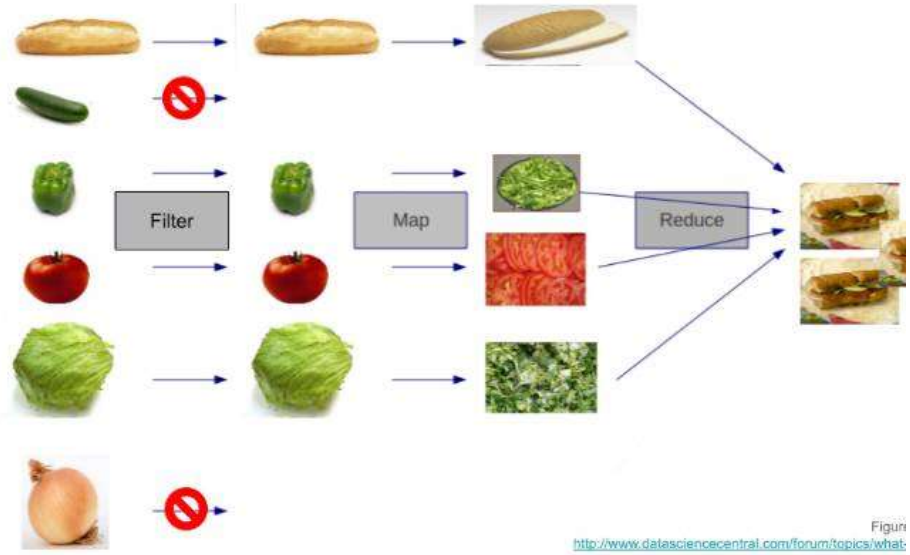
```
1  const arr = [1, 2, 3, 4, 5];
2  const productOfAllNums = arr.reduce((total, currentItem) => {
3    return total * currentItem;
4  }, 1);
5  console.log(productOfAllNums); // Outputs 120;
6  console.log(arr); // Outputs [1, 2, 3, 4, 5]
```

Penjelasan:
1) loop ke-1: total adalah 1 (karena 1 merupakan elemen pertama dari arr) & currentItem adalah 1.
2) return 1 (karena 1 * 1)
3) loop ke-2: total adalah 1 (karena loop sebelumnya me-return 1) & currentItem adalah 2
4) return 2 (karena 1 * 2)
5) loop ke-3: total adalah 2 (karena loop sebelumnya me-return 2) & currentItem adalah 3
6) return 6 (karena 2 * 4)
7) loop ke-4: total adalah 6 & currentItem adalah 4
8) return 24 (karena 6 * 4)
9) loop ke-5 total adalah 24 & currentItem adalah 5
10) return 120

    Output: 120 karena loop terakhir me-return 120

note: callback function is a function in reduce().

- Picture



Figur
http://www.datasciencecentral.com/forum/topics/what-

4. Sort() : mengurutkan elemen-elemen dalam sebuah array(sesuai urutan karakter Unicode)
   Syntax :

```
array.sort(compareFunction)
```

   - Mengurutkan Angka

```
let numbers = [4, 2, 5, 1, 3];
numbers.sort((a, b) => a - b);
console.log(numbers); // Output: [1, 2, 3, 4, 5]

numbers.sort((a, b) => b - a); // Descending
console.log(numbers); // Output: [5, 4, 3, 2, 1]
```

   - Mengurutkan String

```
let fruits = ["banana", "apple", "cherry"];
fruits.sort();
console.log(fruits); // Output: ["apple", "banana", "cherry"]
```

   - Mengurutkan Objek berdasarkan Properti

```
let items = [
  { name: 'John', age: 30 },
  { name: 'Alice', age: 25 },
  { name: 'Bob', age: 35 }
];

items.sort((a, b) => a.age - b.age);
console.log(items);
// Output: [
//   { name: 'Alice', age: 25 },
//   { name: 'John', age: 30 },
//   { name: 'Bob', age: 35 }
// ]
```

5. Example
    1. Map() : mapping array of obj to array of ob

```
let john = { name: "John", surname: "Smith", id: 1 };
let pete = { name: "Pete", surname: "Hunt", id: 2 };
let mary = { name: "Mary", surname: "Key", id: 3 };

let users = [ john, pete, mary ];
/*
usersMapped = [
  { fullName: "John Smith", id: 1 },
  { fullName: "Pete Hunt", id: 2 },
  { fullName: "Mary Key", id: 3 }
]
*/
let usersMapped = users.map(user => ({
  fullName: `${user.name} ${user.surname}`,
  id: `${user.id}`
}));
alert( usersMapped[0].id ) // 1
alert( usersMapped[0].fullName ) // John Smith
```

    2. Reduce()

Array of object to object

```
let users = [
   {id: 'john', name: "John Smith", age: 20},
   {id: 'ann', name: "Ann Smith", age: 24},
   {id: 'pete', name: "Pete Peterson", age: 31},
 ];
 let usersById = groupById(users);
 /*
 // after the call we should have:

 usersById = {
   john: {id: 'john', name: "John Smith", age: 20},
   ann: {id: 'ann', name: "Ann Smith", age: 24},
   pete: {id: 'pete', name: "Pete Peterson", age: 31},
 }
 */
 function groupById(users) {
   return users
       .reduce((obj, item) => {
           obj[item.id] = item;
           return obj;
       }, {});
 }
```

Sum the same array element

```
// Sum up the instances of each of these
const data = ['car', 'car', 'truck', 'truck', 'bike', 'walk', 'car', 'van', 'bike', 'walk', 'car', 'van', 'car', 'truck' ];

const transport = data.reduce((obj, item) => {
  if(!obj[item]) { //'obj[item]' memeriksa apakah properti 'item' (jenis transportasi) sudah ada dalam objek obj.
    obj[item] = 0;
  }
  obj[item]++;
  return obj;
}, {});

console.log(transport);
```

Object property processing:

```
const people = [
      {
        name: "Carly",
        yearOfBirth: 1942,
        yearOfDeath: 1970, //28
      },
      {
        name: "Ray",
        yearOfBirth: 1962,
        yearOfDeath: 2011, //49
      },
      {
        name: "Jane",
        yearOfBirth: 1912,
        yearOfDeath: 1941, //29
      },
    ];

const findTheOldest = function(people) {
    return people.reduce((oldest, currentPerson) => {
        let oldestAge = getAge(oldest.yearOfBirth, oldest.yearOfDeath);
        let currentAge = getAge(
            currentPerson.yearOfBirth,
            currentPerson.yearOfDeath
        );

        return oldestAge < currentAge ? currentPerson : oldest;
```