Pertemuan - 1

Function

Anonymous Function

Anonymous function, juga dikenal sebagai fungsi tak bernama atau fungsi tanpa nama, adalah fungsi dalam JavaScript yang tidak memiliki nama identifikasi yang diberikan. Mereka sering digunakan sebagai argumen dalam pemanggilan fungsi atau sebagai ekspresi fungsi dalam kode.

Berikut adalah contoh sintaksis umum untuk anonymous function dalam JavaScript:

```
// Anonymous function assigned to variable
let fungsi = function (name) {
    console.log("My Name is " + name);
};

fungsi("Faisal");
// Output
// My Name is Faisal

// Anonymous function as an argument
someFunction(function () {
    console.log("This is Anonymous function");
});

someFunction();
// Output
// This is Anonymous function
```

Berikut adalah penjelasan dari masing-masing contoh penggunaan Anonymous function:

1. Anonymous function assigned to variable: Dalam contoh ini, sebuah function ditetapkan ke dalam sebuah variabel. Variabel tersebut nantinya dapat digunakan untuk memanggil Anonymous function ini di tempat lain dalam codingan. Berikut contohnya:

```
let fungsi = function (name) {
    console.log("My Name is " + name);
};

fungsi("Faisal");
```

2. Anonymous function as an argument Anonymous function sering digunakan sebagai parameter untuk function yang lain. Contoh paling umum adalah penggunaan fungsi callback. Berikut contohnya:

```
let sum = (num1, num2, operation) => {
    let result = sum(num1, num2);
    console.log("The result of " + num1 + " + " + num2 + " is " + result);
};

sum(3, 4, function (x, y) {
    return x + y;
});
// Output
// The result of 3 + 4 is 7
```

Dalam contoh di atas, kita memiliki fungsi calculate yang menerima dua angka dan sebuah fungsi anonymous sebagai argumen operation. Fungsi calculate memanggil fungsi operation dengan argumen yang diberikan dan mencetak hasilnya.

Anonymous function juga dapat digunakan dengan metode array seperti **map()**, **filter()**, atau **forEach()**. Berikut adalah contoh penggunaannya:

```
let names = ["Ahmad", "Faisal", "Hidayat"];

let name = names.forEach((name) => console.log(name));

// Output
// Ahmad
// Faisal
// Hidayat
```

Dalam contoh di atas, kita menggunakan method **forEach()** untuk mencetak isi dari array bernama *names* dengan menggunakan variabel *name* sebagai tampungannya.

Keuntungan penggunaan anonymous function adalah kemampuannya untuk membuat kode lebih fleksibel dan modular. Mereka memungkinkan kita untuk mengirimkan logika atau tindakan tertentu sebagai argumen tanpa perlu mendefinisikan fungsi terpisah secara eksplisit.

Arrow Function

Arrow function adalah fitur sintaksis yang diperkenalkan dalam ECMAScript 6 (ES6) untuk mendefinisikan fungsi dalam JavaScript dengan sintaksis yang lebih singkat dan ekspresif. Arrow function juga dikenal sebagai lambda function atau fat arrow function. Mereka menyederhanakan penulisan fungsi dan memiliki perilaku yang sedikit berbeda dari fungsi konvensional.

Berikut adalah beberapa sintaks umum untuk Arrow Function pada javascript:

```
// Arrow Function with parentheses (for multiple parameters)
let myFunction = (param1, param2) => {
    // Function body
};
```

```
// Arrow Function with single parameter
let myFunction = (param) => {
    // Function body
};

// Arrow Function with implicit return (single expression)
let myFunction = () => expression;

// Arrow Function with implicit return (object literal)
let myFunction = () => ({ key: value });
```

Berikut adalah penjelasan dari masing-masing penggunaan Arrow Function diatas:

1. Arrow Function with parentheses (for multiple parameters): Ini adalah bentuk arrow function yang digunakan ketika fungsi memiliki lebih dari satu parameter. Tanda kurung diperlukan saat ada lebih dari satu parameter. Berikut contohnya:

```
let myFunction = (a, b) => {
    return a + b;
};

console.log(myFunction(5, 3));
// Output
// 8
```

2. Arrow Function with single parameter: Ketika fungsi hanya memiliki satu parameter, tanda kurung sekitar parameter tersebut dapat dihilangkan. Berikut contohnya:

```
let myFunction = (num) => {
    return num * num;
};

console.log(myFunction(5));
// Output
// 25
```

3. Arrow Function with Implicit Return (Single Expression) Jika fungsi hanya memiliki satu pernyataan atau ekspresi, kita dapat menghilangkan kata kunci return dan kurung kurawal. Fungsi akan secara otomatis mengembalikan hasil ekspresi tersebut. Berikut contohnya:

```
let myFunction = (a, b) => a * b;

console.log(myFunction(5, 3));
// Output
// 15
```

4. Arrow Function with Implicit Return (Object Literal) Jika kita ingin mengembalikan sebuah objek literal secara implisit dari arrow function, kita perlu menggunakan tanda kurung di sekitar objek literal tersebut. Berikut contohnya:

```
let createPerson = (name, age) => ({ name, age });

console.log(createPerson("Faisal", 19));
// Output
// { name: "Faisal", age: 19 }
```

Keuntungan penggunaan arrow function meliputi sintaksis yang lebih singkat dan pengikatan this yang berbeda. Dalam arrow function, this terikat pada konteks lingkungan yang mengelilinginya (lexical this), bukan pada objek yang memanggilnya. Ini dapat menghindari perubahan nilai this secara tidak sengaja dalam konteks yang berbeda.

Namun, perlu diperhatikan bahwa arrow function tidak dapat digunakan dalam beberapa situasi, seperti sebagai konstruktor objek atau ketika kita membutuhkan arguments yang merupakan objek array-like dari argumen fungsi.

Perlu diingat bahwa arrow function adalah salah satu fitur yang diperkenalkan dalam ES6, sehingga dukungan untuk arrow function mungkin tidak ada pada lingkungan runtime yang lebih lama.

Callback Function

Callback function merupakan konsep fundamental dalam JavaScript yang melibatkan penggunaan fungsi sebagai argumen dan dipanggil kembali (callback) pada waktu yang sesuai dalam suatu proses. Callback function digunakan untuk menangani tindakan atau respons setelah suatu tugas atau operasi selesai dilakukan.

Berikut adalah penjelasan lebih detail tentang callback function beserta contoh penggunaannya:

1. Menggunakan Callback Function sederhana:

```
function greeting(name, callback) {
    console.log("Hello, My Name is " + name);
    callback();
}

function sayGoodbye() {
    console.log("Goodbye!");
}

greeting("Faisal", sayGoodbye());
// Output
// Hello, My Name is Faisal
// Goodbye!
```

Dalam contoh ini, kita memiliki fungsi greeting yang menerima sebuah nama dan callback function sebagai argumen. Setelah fungsi greeting mencetak salam, callback function sayGoodbye dipanggil untuk mencetak pesan perpisahan.

2. Menggunakan Callback Function dalam Asynchronous Operations:

```
function fetchData(callback) {
    setTimeout(function() {
        var = "Data has been fetched"
        callback(data)
    }, 2000)
}

function displayData(data) {
    console.log(data)
}

fetchData(displayData)
// Output
// Data has been fetched
```

Dalam contoh ini, fungsi fetchData mensimulasikan pengambilan data dari sumber eksternal dengan menggunakan fungsi setTimeout untuk menunda eksekusi. Setelah data berhasil diambil, callback function displayData dipanggil untuk menampilkan data.

3. Menggunakan Anonymous Function sebagai Callback Function:

```
function processArray(array, callback) {
    for (let i = 0; i < array.length; i++) {
        callback(array[i]);
    }
}

let numbers = [1, 2, 3];

processArray(numbers, (numbers) => {
    console.log(numbers);
});
// Output
// 1
// 2
// 3
```

Dalam contoh ini, kita memiliki fungsi processArray yang menerima sebuah array dan callback function sebagai argumen. Fungsi tersebut mengiterasi setiap elemen array dan memanggil callback function untuk setiap elemen dan mencetaknya.

4. Menggunakan Callback Function sebagai Error Handling:

```
const fetchData = (callback, errorCallback) => {
    // Simulate an error
    let error = true;

    if (error) {
        errorCallback("Error fetching data.");
    } else {
        let data = "Data has been fetched";
        callback(data);
    }
};

const displayData = (data) => {
    console.log(data);
};

const displayError = (error) => {
    console.log("Error: " + error);
};

fetchData(displayData, displayError);
// Output
// Error fetching data
```

Dalam contoh ini, fungsi fetchData mensimulasikan pengambilan data dengan memeriksa kondisi error. Jika terjadi error, callback function displayError dipanggil dengan pesan error. Jika tidak ada error, callback function displayData dipanggil dengan data yang berhasil diambil.

Callback function sangat berguna dalam pemrograman JavaScript, terutama dalam pengolahan data asynchronous, pemanggilan API, dan event handling. Mereka memungkinkan kita untuk menjalankan tindakan setelah suatu tugas selesai atau saat terjadi peristiwa tertentu.

Rest Parameter (...)

Rest parameter adalah fitur yang diperkenalkan dalam ECMAScript 6 (ES6) yang memungkinkan sebuah fungsi untuk menerima sejumlah variabel argumen sebagai array. Rest parameter ditandai dengan penggunaan tanda titik tiga (ellipsis) **(...)** di depan parameter terakhir dalam definisi fungsi. Hal ini memungkinkan kita untuk mengirim sejumlah argumen yang tidak terbatas ke dalam fungsi dan mengaksesnya sebagai array di dalam fungsi.

Berikut adalah penjelasan lebih detail tentang rest parameter beserta contoh penggunaannya:

1. Penggunaan Rest Parameter:

```
const sum = (...numbers) => {
    let total = 0;

    for (let number of numbers) {
        total += number;
    }
```

```
    return total;
};

console.log(sum(1, 2, 3, 4, 5, 6, 7));
// Output
// 28
```

Dalam contoh ini, kita memiliki fungsi sum yang menggunakan rest parameter ...numbers. Rest parameter tersebut akan mengumpulkan semua argumen yang dikirimkan ke fungsi dan menyimpannya sebagai array numbers. Dalam tubuh fungsi, kita dapat melakukan iterasi melalui array numbers dan menjumlahkan semua angka untuk menghasilkan total.

2. Rest Parameter dengan parameter lain:

```
const multiply = (multiplier, ...numbers) => {
    let result = numbers.map((number) => number * multiplier);
    return result;
};

console.log(multiply(2, 1, 2, 3, 4, 5));
// Output
// [2, 4, 6, 8, 10]
```

Dalam contoh ini, kita memiliki fungsi multiply yang menggunakan rest parameter ...numbers setelah parameter multiplier. Rest parameter akan mengumpulkan argumen-argumen setelah multiplier dan menyimpannya sebagai array numbers. Dalam tubuh fungsi, kita menggunakan metode map() untuk mengalikan setiap elemen array numbers dengan multiplier dan mengembalikan array hasil perkalian.

3. Rest Parameter dengan Destructuring:

```
const getFullName = (first, last, ...middle) {
    console.log("First Name: " + first)
    console.log("Last Name: " + last)
    console.log("Middle Names: " + middle.join(", "))
}

getFullName("John", "Doe", "Adam", "Michael", "Smith")
// Output
// First Name: John
// Last Name: Doe
// Middle Names: Adam, Michael, Smith
```

Dalam contoh ini, kita memiliki fungsi **getFullName** yang menggunakan rest parameter **...middle** setelah parameter **last**. Rest parameter akan mengumpulkan semua argumen setelah **last** dan menyimpannya sebagai array **middle**. Dalam tubuh fungsi, kita mencetak nama depan, nama belakang, dan semua nama tengah

dengan menggunakan metode **join()** untuk menggabungkan elemen-elemen array menjadi satu string yang dipisahkan oleh koma.

Rest parameter memungkinkan fleksibilitas dalam mengirimkan sejumlah argumen ke dalam fungsi tanpa membatasi jumlahnya secara eksplisit. Dengan menggunakan rest parameter, kita dapat dengan mudah mengelola argumen-argumen tersebut sebagai array di dalam fungsi dan melakukan operasi yang sesuai.

Object

Optional Chaining(?.)

Optional chaining adalah fitur yang diperkenalkan dalam ECMAScript 2020 (ES2020) yang memungkinkan akses ke properti atau metode dari sebuah objek tanpa menghasilkan error ketika objek tersebut bernilai null atau undefined. Optional chaining ditandai dengan penggunaan tanda tanya (?) setelah objek yang sedang diakses.

Berikut adalah penjelasan lebih detail tentang optional chaining beserta contoh penggunaannya:

1. Akses properti dengan **Optional Chaining**

```
let person = {
    name: "Faisal",
    age: "19",
    address: {
        city: "Tangerang",
        country: "Indonesia",
    },
};

console.log(person.address?.city);
console.log(person.address?.region);
// Output
// Tangerang
// Undefined
```

Dalam contoh ini, kita memiliki objek person dengan properti **address**. Dengan menggunakan **optional chaining (?.)**, kita dapat mengakses properti **city** dan **zipCode** dari **address** dengan aman. Jika **address** atau properti yang diakses tidak ada, maka hasilnya adalah **undefined** tanpa menyebabkan error.

2. Akses method dengan **Optional Chaining**

```
let person = {
    name: "Faisal",
    age: "19",
    sayHello: function () {
        console.log(`Hi, My name is ${this.name}, I am ${this.age} years old`);
    },
};
```

```
console.log(person.sayHello?.());
console.log(person.sayGoodbye?.());
// Output
// Hi, My name is Faisal, I am 19 years old
// Undefined
```

Dalam contoh ini, kita memiliki objek **person** dengan method **sayHello** dan **sayGoodbye**. Dengan **optional chaining (?.)**, kita dapat memanggil method **sayHello** dan **sayGoodbye** dengan aman. Jika method yang diakses tidak ada, maka tidak akan terjadi error.

3. Optional Chaining dengan array

```
let numbers = [1, 2, 3, 4, 5];

console.log(numbers?.[2]);
console.log(numbers?.[10]);
// Output
// 3
// undefined
```

Dalam contoh ini, kita memiliki array **numbers**. Dengan **optional chaining (?.)**, kita dapat mengakses elemen array berdasarkan indeks dengan aman. Jika indeks yang diakses tidak ada, maka hasilnya adalah **undefined**.

Optional chaining sangat berguna dalam situasi di mana kita tidak yakin apakah properti atau method yang ingin kita akses ada atau tidak. Dengan menggunakan **optional chaining**, kita dapat menghindari error yang disebabkan oleh objek yang bernilai **null** atau **undefined**. Ini membantu meningkatkan ketahanan dan keamanan kode kita, serta memudahkan penanganan kasus-kasus di mana objek atau properti mungkin tidak ada.

Array

Method-Method Array

Berikut adalah method umum yang ada pada objek Array dalam JavaScript:

1. **concat()** : Menggabungkan dua atau lebih array dan mengembalikan array baru.
2. **join()** : Menggabungkan semua elemen array menjadi satu string menggunakan pemisah yang ditentukan, seperti , atau - .
3. **pop()** : Menghapus dan mengembalikan elemen terakhir array.
4. **push()** : Menambahkan satu atau lebih elemen ke akhir array.
5. **shift()** : Menghapus dan mengembalikan elemen pertama dari array.
6. **unshift()** : Menambahkan satu atau lebih elemen ke awal array.
7. **slice()** : Membuat salinan dangkal(shallow copy) dari sebagian array ke array baru.
8. **splice()** : Mengubah isi array dengan menghapus, mengganti, atau menambahkan elemen.
9. **forEach()** : Menjalankan fungsi callback untuk setiap elemen array.
10. **map()** : Membuat array baru dengan hasil pemanggilan fungsi callback pada setiap elemen dalam array.

11. **filter()** : Membuat array dengan elemen-elemen dari array awal yang lulus kondisi tertentu.
12. **find()** : Mengembalikan nilai dari elemen pertama dalam array yang memenuhi kondisi tertentu.
13. **reduce()** : Mengurangi array menjadi satu nilai tunggal dengan menjalankan fungsi callback pada setiap elemen.
14. **sort()** : Mengurutkan elemen-elemen array secara default atau dengan menggunakan fungsi pembanding.
15. **reverse()** : Membalikkan urutan-urutan elemen dalam array.
16. **indexOf()** : Mengembalikan indeks pertama dari elemen yang cocok dalam array.
17. **lastIndexOf()** : Mengembalikan indeks terakhir dari elemen yang cocok dalam array.
18. **includes()** : Memeriksa apakah array berisi elemen tertentu dan mengembalikan nilai boolean.
19. **some()** : Memeriksa apakah setidaknya satu elemen dalam array memenuhi kondisi tertentu.
20. **every()** : Memeriksa apakah semua elemen dalam array memenuhi kondisi tertentu.

Dalam pelatihan kali ini, kita hanya akan fokus dan mendalami 5 method saja, yaitu **forEach()**, **map()**, **filter()**, **find()**, dan **reduce()**.

forEach()

Method **forEach()** adalah salah satu metode bawaan dari objek Array dalam JavaScript yang digunakan untuk menjalankan sebuah fungsi callback pada setiap elemen dalam array. Fungsi callback akan dieksekusi sekali untuk setiap elemen dalam array tanpa mengembalikan nilai apapun.

Berikut adalah penjelasan lebih detail tentang metode **forEach()** beserta contoh penggunaannya:

```
array.forEach((currentValue, index, array) => {
    // statement
});
```

- **array** : Array yang ingin diiterasi.
- **(...) => {}** : Arrow function yang akan dieksekusi untuk setiap elemen dalam array.
- **currentValue** : Nilai saat ini dari elemen yang sedang diproses.
- **index** (opsional) : Indeks saat ini dari elemen yang sedang diproses.
- **array** (opsional) : Array asli yang sedang diiterasi.

Contoh penggunaan **forEach()** :

```
let numbers = [1, 2, 3, 4, 5]

numbers.forEach((number) => {
    console.log(number)
})
// Output
// 1
// 2
// 3
// 4
// 5
```

Dalam contoh ini, kita memiliki array **numbers**. Kita menggunakan method **forEach()** untuk menjalankan sebuah fungsi callback pada setiap elemen dalam array. Fungsi callback menerima satu argumen number, yang merupakan nilai saat ini dari elemen yang sedang diproses. Pada setiap iterasi, fungsi callback mencetak nilai number ke konsol.

Contoh lain dengan indeks dan array:

```
let fruits = ["Banana", "Orange", "Mango"]

fruits.forEach((fruit, index, array) => {
    console.log(`Fruit : ${fruit}`)
    console.log(`Index : ${index}`)
    console.log(`Array : ${array}`)
})
// Output
// Fruit : Banana
// Index : 0
// Array : Banana, Orange, Mango
// ...
```

Dalam contoh ini, kita menggunakan method **forEach()** pada array **fruits**. Fungsi callback menerima tiga argumen: **fruit** (nilai saat ini), **index** (indeks saat ini), dan **array** (array asli yang sedang diiterasi). Pada setiap iterasi, fungsi callback mencetak nilai buah, indeks, dan array ke konsol.

Method **forEach()** berguna saat kita ingin menjalankan suatu tindakan atau operasi pada setiap elemen dalam array, seperti melakukan manipulasi data atau tindakan lain yang berkaitan dengan setiap elemen.

map()

Method **map()** adalah metode bawaan dari objek Array dalam JavaScript yang digunakan untuk membuat array baru dengan hasil pemanggilan fungsi callback pada setiap elemen dalam array yang sedang diiterasi. Fungsi callback akan menerima setiap elemen sebagai argumen dan mengembalikan nilai yang akan digunakan untuk membentuk elemen baru dalam array baru yang dihasilkan.

Berikut adalah penjelasan lebih detail tentang method **map()** beserta contoh penggunaannya:

```
let newArray = array.map((currentValue, index, array) => {
    // statement
    return result
})
```

- **array** : Array yang ingin diiterasi.
- **(...) => {}** : Arrow function yang akan dieksekusi untuk setiap elemen dalam array.
- **currentValue** : Nilai saat ini dari elemen yang sedang diproses.
- **index** (opsional) : Indeks saat ini dari elemen yang sedang diproses.
- **array** (opsional) : Array asli yang sedang diiterasi.
- **result** : Nilai yang akan digunakan untuk membentuk elemen baru pada array baru.

Contoh penggunaan **map()**:

```
let numbers = [1, 2, 3, 4, 5]

let squaredNumbers = numbers.map((number) => number * number)
console.log(squaredNumbers)
// Output
// [1, 4, 9, 16, 25]
```

Dalam contoh ini, kita memiliki array **numbers**. Kita menggunakan method **map()** untuk membuat array baru **squaredNumbers** dengan mengalikan setiap elemen dalam array **numbers** dengan dirinya sendiri. Fungsi callback menerima satu argumen **number** (nilai saat ini) dan mengembalikan hasil perkalian. Hasilnya adalah array baru **squaredNumbers** yang berisi nilai-nilai hasil kuadrat dari setiap elemen dalam array **numbers**.

Contoh lain penggunaan method **map()** dengan indeks dan array:

```
let names = ["Faisal", "Nanda", "Ali"]

let upperNames = names.map((name) => name.toUpperCase())
console.log(upperNames)
// Output
// ['FAISAL', 'NANDA', 'ALI']
```

Dalam contoh ini, kita menggunakan method **map()** pada array **names**. Fungsi callback menerima tiga argumen: **name** (nilai saat ini), **index** (indeks saat ini), dan **array** (array asli yang sedang diiterasi). Fungsi callback mengembalikan nilai berupa hasil **toUpperCase()** dari setiap elemen **name**. Hasilnya adalah array baru **upperNames** yang berisi semua elemen **names** dalam huruf besar.

Method **map()** berguna saat kita ingin membuat array baru dengan hasil transformasi atau pemetaan setiap elemen dalam array yang sudah ada. Dengan menggunakan **map()**, kita dapat dengan mudah memanipulasi atau mengubah setiap elemen array menjadi bentuk yang diinginkan.

filter()

Method **filter()** adalah metode bawaan dari objek Array dalam JavaScript yang digunakan untuk membuat array baru yang berisi elemen-elemen dari array asli yang memenuhi kondisi tertentu. Metode **filter()** akan menjalankan sebuah fungsi callback pada setiap elemen dalam array dan mengembalikan array baru yang berisi elemen-elemen yang lulus kondisi yang ditentukan.

Berikut adalah penjelasan lebih detail tentang metode **filter()** beserta contoh penggunaannya:

```
let newArray = array.filter((currentValue, index, array) => {
    // statement
    return condition
})
```

- **array** : Array yang ingin diiterasi.
- **(...) => {}** : Arrow function yang akan dieksekusi untuk setiap elemen dalam array.
- **currentValue** : Nilai saat ini dari elemen yang sedang diproses.
- **index** (opsional) : Indeks saat ini dari elemen yang sedang diproses.
- **array** (opsional) : Array asli yang sedang diiterasi.
- **condition** : Kondisi yang menetukan apakah elemen akan disertakan dalam array baru atau tidak. Jika kondisi mengembalikan **true**, elemen akan disertakan dalam array baru.

Contoh penggunaan method **filter()**:

```
let numbers = [1, 2, 3, 4, 5, 6, 7, 8]

let oddNumbers = numbers.filter((number) => number % 2 === 1)
console.log(oddNumbers)
// Output
// [1, 3, 5, 7]
```

Dalam contoh ini, kita memiliki array **numbers**. Kita menggunakan method **filter()** untuk membuat array baru evenNumbers yang berisi elemen-elemen dalam array numbers yang merupakan bilangan genap. Fungsi callback menerima satu argumen number (nilai saat ini) dan mengembalikan nilai boolean berdasarkan kondisi apakah bilangan tersebut genap (number % 2 === 0). Elemen-elemen yang memenuhi kondisi akan disertakan dalam array baru evenNumbers.

Contoh lain dengan menggunakan array:

```
let person = ["Faisal", "Anggi", "Achmad"]

let saram = person.filter((saram) => saram.length < 6)
console.log(saram)
// Output
// ['Anggi']
```

find()

Method **find()** adalah metode bawaan dari objek Array dalam JavaScript yang digunakan untuk mencari elemen pertama dalam array yang memenuhi kondisi tertentu. Method **find()** akan menjalankan sebuah fungsi callback pada setiap elemen dalam array dan mengembalikan nilai dari elemen pertama yang memenuhi kondisi yang ditentukan.

Berikut adalah penjelasan lebih detail tentang method **find()** beserta contoh penggunaannya:

```
let result = array.find((currentValue, index, array) => {
    // statement
    return condition
})
```

- **array** : Array yang ingin diiterasi.
- **(...) => {}** : Arrow function yang akan dieksekusi untuk setiap elemen dalam array.
- **currentValue** : Nilai saat ini dari elemen yang sedang diproses.
- **index** (opsional) : Indeks saat ini dari elemen yang sedang diproses.
- **array** (opsional) : Array asli yang sedang diiterasi.
- **condition** : Kondisi yang menetukan apakah elemen memenuhi kriteria yang dicari.

Contoh penggunaan **find()**:

```
let numbers = [1, 2, 3, 4, 5]

let evenNumber = numbers.find((number) => number % 2 === 0)
console.log(evenNumber)
// Output
// 2
```

Dalam contoh ini, kita memiliki array **numbers**. Kita menggunakan method **find()** untuk mencari elemen pertama dalam array **numbers** yang merupakan bilangan genap. Fungsi callback menerima satu argumen **number** (nilai saat ini) dan mengembalikan nilai **boolean** berdasarkan kondisi apakah bilangan tersebut genap **(number % 2 === 0)**. Method **find()** akan mengembalikan nilai dari elemen pertama yang memenuhi kondisi, yaitu angka 2.

Contoh lain dengan indeks array:

```
let persons = ["Faisal", "Anggi", "Achmad"]

let saram = persons.find((person) => person === "Faisal")
console.log(saram)
// Output
// ['Faisal']
```

Dalam contoh ini, kita menggunakan method **find()** pada array **fruits**. Fungsi callback mengembalikan nilai boolean berdasarkan kondisi apakah **person** sama dengan **'Faisal' (person === "Faisal")**. Method **find()** akan mengembalikan nilai dari elemen pertama yang memenuhi kondisi tersebut, yaitu **'Faisal'**.

Method **find()** berguna saat kita ingin mencari elemen pertama dalam array yang memenuhi kondisi tertentu. Dengan menggunakan **find()**, kita dapat dengan mudah menemukan elemen yang sesuai dengan kriteria pencarian kita tanpa harus melakukan iterasi manual melalui seluruh array.

reduce()

Method **reduce()** adalah metode bawaan dari objek Array dalam JavaScript yang digunakan untuk mengurangi atau menggabungkan semua elemen dalam array menjadi sebuah nilai tunggal. Method **reduce()** akan menjalankan sebuah fungsi callback pada setiap elemen dalam array dengan mengakumulasi nilai hasilnya.

Berikut adalah penjelasan lebih detail tentang method **reduce()** beserta contoh penggunaannya:

```
let result = array.reduce((accumulator, currentValue, index, array) => {
    // statement
    return updateAccumulator
}, initialValue)
```

- **array** : Array yang ingin diiterasi.
- **(...) => {}** : Arrow function yang akan dieksekusi untuk setiap elemen dalam array.
- **accumulator** : Nilai yang digunakan untuk mengakumulasi hasil reduksi.
- **currentValue** : Nilai saat ini dari elemen yang sedang diproses.
- **index** (opsional) : Indeks saat ini dari elemen yang sedang diproses.
- **array** (opsional) : Array asli yang sedang diiterasi.
- **updateAccumulator** : Nilai yang diperbarui untuk mengakumulasi hasil reduksi.
- **initialValue** (opsional) : Nilai awal yang akan digunakan sebagai **accumulator** pada iterasi pertama. jika tidak disediakan, maka **accumulator** akan menggunakan indeks pertama dalam array.

Contoh penggunaan **reduce()**:

```
let numbers = [1, 2, 3, 4, 5]

let sum = numbers.reduce((accumulator, currentValue) => accumulator +
currentValue, 0)
console.log(sum)
// Output
// 15
```

Dalam contoh ini, kita memiliki array **numbers**. Kita menggunakan method **reduce()** untuk mengakumulasi semua elemen dalam array **numbers** menjadi sebuah nilai tunggal, yaitu penjumlahannya. Fungsi callback menerima dua argumen: **accumulator** (nilai yang digunakan untuk mengakumulasi hasil) dan **currentValue** (nilai saat ini). Pada setiap iterasi, nilai **currentValue** ditambahkan ke **accumulator**. Nilai awal **accumulator** adalah 0 (diberikan sebagai argumen kedua dalam method **reduce()**). Hasil akhirnya adalah 15, yang merupakan hasil penjumlahan dari semua elemen dalam array **numbers**.

Contoh lain dengan indeks dan array:

```
let persons = ["Faisal", "Anggi", "Achmad"]

let combine = persons.reduce((accumulator, currentValue) => accumulator + " " +
currentValue, "")
console.log(combine)
// Output
// Faisal Anggi Achmad
```

Dalam contoh ini, kita menggunakan method **reduce()** pada array **persons**. Pada setiap iterasi, nilai **currentValue** ditambahkan ke **accumulator** dengan spasi sebagai pemisah. Hasil akhirnya adalah 'Faisal Anggi Achmad', yang merupakan hasil penggabungan semua elemen dalam array **persons**.

Method **reduce()** berguna saat kita ingin mengurangi atau menggabungkan semua elemen dalam array menjadi sebuah nilai tunggal, seperti menjumlahkan elemen, menggabungkan string, atau melakukan operasi matematika lainnya. Dengan menggunakan **reduce()**, kita dapat dengan mudah melakukan operasi yang kompleks pada array dengan hasil yang diinginkan.

Destructuring Assignment

Destructuring Assignment adalah fitur dalam JavaScript yang memungkinkan kita untuk "mendestruksikan" atau mengurai nilai dari array atau objek ke dalam variabel terpisah. Dengan **destructuring assignment**, kita dapat mengeluarkan nilai-nilai yang terkandung dalam struktur data tersebut dan menetapkannya ke variabel yang terpisah dengan sintaks yang lebih ringkas.

Destructuring Assignment dapat dilakukan pada array dan objek. Berikut adalah penjelasan lebih detail tentang **destructuring assignment**:

Destructuring Array

Dalam **destructuring array**, kita menentukan variabel-variabel yang ingin kita inisialisasi dengan nilai-nilai dari array yang sedang kita destructuring.

Contoh penggunaan destructuring array:

```
let numbers = [7, 6, 2003]

let [date, month, year] = numbers
console.log("Tanggal lahir saya " + date + "-" + month + "-" + year)
// Output Tanggal lahir saya 7-6-2003
```

Dalam contoh diatas kita mendestruksikan variabel **numbers** ke dalam variabel **date**, **month**, **year**.Nilai pertama dari array **numbers** akan diassign(dimasukkan) ke dalam variabel **date**, nilai kedua akan diassign ke variabel **month**, dan nilai ketiga akan diassign ke variabel **year**.

Destructuring Object

Dalam **destructuring object**, kita menentukan variabel-variabel dengan nama yang sesuai dengan properti-properti yang ingin kita inisialisasi dari objek yang sedang kita destructuring.

contoh penggunaan **destructuring object**:

```
let person = {
    nama: "Ahmad Faisal Hidayat",
    nim: "02042111003",
    email: "ahmadfaisalhidayat127@gmail.com"
}

let {nama, nim, email} = person
console.log("Nama       : " + nama)
console.log("NIM        : " + nim)
console.log("Email      : " + email)
```

```
// Output
// Nama     : Ahmad Faisal Hidayat
// NIM      : 02042111003
// Email    : ahmadfaisalhidayat127@gmail.com
```

Dalam contoh di atas, kita mendestruksikan object **person** ke dalam variabel **nama**, **nim**, **email**. Properti **nama** diassign ke variabel **nama**, properti **nim** diassign ke variabel **nim**, properti **email** diassign ke variabel **email**.

Destructuring Assignment dengan Default Value

Destructuring Assignment juga memungkinkan kita untuk memberikan nilai default pada variabel jika nilai yang sesuai tidak ada atau undefined.

Contoh penggunaan **destructuring assignment** dengan **default value**:

```
let numbers = [1]

let [number1, number2 = 2] = numbers
console.log(number1)
console.log(number2)
// Output
// 1
// 2
```

Dalam contoh di atas, kita mendestuksikan array **numbers** ke dalam variabel **number1** dan **number2**. Karena array hanya memiliki satu elemen, nilai **1** akan diassign ke variabel **number1**. Namun, karena tidak ada elemen kedua dalam array, variabel **number2** akan menggunakan **default value**, yaitu **2**.

Rest Property/Element

Rest property/element adalah fitur dalam JavaScript yang memungkinkan kita untuk mengumpulkan sisa properti dalam objek atau elemen dalam array menjadi satu variabel terpisah. **Rest property/element** ditandai dengan penggunaan sintaks **tiga titik (...)** di depan variabel yang bertindak sebagai penampung untuk sisa properti/element.

Berikut adalah penjelasan lebih detail tentang rest property/element:

Rest Property dalam object

Dalam **rest property** pada objek, kita dapat mengumpulkan sisa properti yang tidak ditentukan secara eksplisit menjadi satu variabel terpisah.

Contoh penggunaan rest property pada objek:

```
let person = {
    nama: "Ahmad Faisal Hidayat",
    nim: "02042111003",
```

```
    email: "ahmadfaisalhidayat127@gmail.com",
    age: 19,
    ttl: "Tangerang, 07 Juni 2003",
    gender: "Man"
}

let {name, nim, email, ...detail} = person
console.log(name)
console.log(nim)
console.log(email)
console.log(detail)
// Output
// Ahmad Faisal Hidayat
// 0204211003
// ahmadfaisalhidayat127@gmail.com
// {age: 19, ttl: "Tangerang, 07 Juni 2003", gender: "Man"}
```

Dalam contoh di atas, kita mendestruksikan objek person ke dalam variabel **nama**, **nim**, **email**, dan **detail**. Properti **nama** akan diassign ke variabel **nama**, properti **nim** akan diassign ke variabel **nim**, properti **email** akan diassign ke variabel **email**, dan sisanya (**age**, **ttl**, dan **gender**) akan dikumpulkan ke dalam variabel **detail** sebagai objek terpisah.

Rest Element dalam Array

Dalam **rest element** pada array, kita dapat mengumpulkan sisa elemen yang tidak ditentukan secara eksplisit menjadi satu array terpisah.

Contoh penggunaan **rest element** pada array:

```
let numbers = [1, 2, 3, 4, 5]

let [number1, number2, ...rest] = numbers
console.log(number1)
console.log(number2)
console.log(rest)
// Output
// 1
// 2
// [3, 4, 5]
```

Dalam contoh di atas, kita mendestruksikan array **numbers** ke dalam variabel **number1**, **number2**, dan rest. Elemen pertama array **(1)** akan diassign ke variabel **number1**, elemen kedua **(2)** ke variabel **number2**, sedangkan sisa elemen **(3, 4, dan 5)** akan dikumpulkan ke dalam variabel rest sebagai array terpisah.

Rest property/element sangat berguna ketika kita ingin mengumpulkan sisa **properti/element** yang tidak perlu secara eksplisit ditentukan dalam proses destructuring. Dengan menggunakan **rest property/element**, kita dapat dengan mudah mengakses nilai-nilai yang tidak ditentukan tanpa harus menyebutkan **properti/element** satu per satu.

Destructuring assignment sangat berguna dalam mengakses nilai-nilai yang terkandung dalam array atau objek dengan sintaks yang lebih ringkas. Hal ini memudahkan kita untuk mendapatkan nilai-nilai yang dibutuhkan tanpa harus mengaksesnya satu per satu menggunakan indeks atau nama properti secara manual.

Spread (...)

Spread syntax atau **operator spread (...)** adalah fitur dalam JavaScript yang digunakan untuk "mengurai" atau memecah sebuah iterable **(seperti array, string, atau objek literal)** menjadi beberapa elemen terpisah. **Operator spread** digunakan untuk mengekspansi nilai-nilai dari iterable ke dalam konteks yang membutuhkan beberapa nilai terpisah.

Berikut adalah penjelasan lebih detail tentang spread syntax:

Spread pada Array

Dalam konteks array, **spread syntax** digunakan untuk memecah elemen-elemen dari array menjadi elemen terpisah. **Spread syntax** dapat digunakan saat membuat array baru, menggabungkan dua atau lebih array, atau mengirimkan argumen ke fungsi yang membutuhkan beberapa argumen terpisah.

Contoh penggunaan **spread pada array**:

```
let numbers = [1, 2, 3]

let newNumbers = [...numbers, 4, 5]
console.log(newNumbers)
// Output
// [1, 2, 3, 4, 5]
```

Dalam contoh di atas, **operator spread (...numbers)** digunakan untuk mengurai elemen-elemen dari array **numbers**. Kemudian, kita menggabungkan elemen-elemen tersebut dengan elemen lain **(4 dan 5)** menjadi sebuah array baru **(newArray)**.

Spread pada String

Dalam konteks string, **spread syntax** digunakan untuk memecah string menjadi karakter-karakter terpisah.

Contoh penggunaan **spread pada string**:

```
let str = "Hello"

let characters = [...str]

console.log(characters)
// Output
// ['H', 'e', 'l', 'l', 'o']
```

Dalam contoh di atas, **operator spread (...str)** digunakan untuk mengurai string **'Hello'** menjadi karakter-karakter terpisah. Hasilnya adalah array **characters** yang berisi karakter-karakter dalam string.

Spread pada Object Literal

Dalam konteks **objek literal**, **spread syntax** digunakan untuk menyalin properti-properti dari satu objek ke objek lain.

Contoh penggunaan **spread pada objek literal**:

```
let object1 = { name: "Ahmad Faisal Hidayat" }
let object2 = { nim: "02042111003" }

let combine = {...object1, ...object2}
console.log(combine)
// Output
// { name: "Ahmad Faisal Hidayat", nim: "02042111003" }
```

Dalam contoh di atas, **operator spread (...object1 dan ...object2)** digunakan untuk menyalin properti-properti dari objek **object1 dan object2** ke dalam objek baru **(combine)**, sehingga properti-properti dari kedua objek digabungkan menjadi satu objek.

Spread syntax sangat berguna dalam menggabungkan atau menyalin elemen-elemen dari iterable menjadi konteks yang membutuhkan nilai-nilai terpisah. Dengan menggunakan **operator spread**, kita dapat dengan mudah menambahkan, menggabungkan, atau menyalin nilai-nilai dengan sintaksis yang lebih ringkas.