

JavaScript & TypeScript

By Fei

Creating Object in JS

JS

01

Object Literal | (-): memakan banyak resource

```
1 let mahasiswa = {  
2   'name': 'Fei',  
3   'energy': 10,  
4   makan: function (portion) {  
5     this.energy = this.energy + portion;  
6     console.log(`Halo ${this.name}, selamat makan`);  
7   }  
8 }
```

02

Function Declaration | (-): masih terdapat duplikasi
namun dapat menggunakan Object.create

```
1 const methodMhs = {  
2   eat: function (portion) {  
3     this.energy += portion;  
4     console.log(`Halo ${this.name}, selamat makan`);  
5   },  
6   play: function (hour) {  
7     this.energy -= hour;  
8     console.log(`Halo ${this.name}, selamat bermain`);  
9   },  
10  sleep: function (hour) {  
11    this.energy += hour * 2;  
12    console.log(`Halo ${this.name}, selamat tidur`);  
13  },  
14};  
15
```

```
15  
16 function mahasiswa(name, energy) {  
17   let mhs = {};  
18   mhs.name = name;  
19   mhs.energy = energy;  
20   mhs.eat = methodMhs.eat;  
21   mhs.play = methodMhs.play;  
22   mhs.sleep = methodMhs.sleep;  
23   // kelemahan: setiap ada perubahan method harus didefinisikan juga  
24   return mhs;  
25 }  
26  
27 let fei = mahasiswa('Fei', 10);  
28
```



```
1 function mahasiswa(name, energy) {  
2   /* memberitahukan parent objectnya jadi tidak perlu mendeklarasikan  
3    * setiap perubahan yang ada */  
4   let mhs = Object.create(methodMhs);  
5   mhs.name = name;  
6   mhs.energy = energy;  
7  
8   return mhs;  
9 }
```

Prompt: Object creation in JS

03

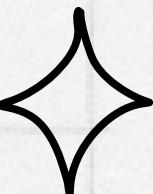
Constructor Function dengan Prototype

```
1 function mahasiswa(name, energy) {  
2   this.name = name;  
3   this.energy = energy;  
4 }  
5  
6 mahasiswa.prototype.eat = function (portion) {  
7   this.energy += portion;  
8   return `Halo ${this.name}, selamat makan`;  
9 }  
10  
11 mahasiswa.prototype.play = function (hour) {  
12   this.energy -= hour;  
13   return `Halo ${this.name}, selamat bermain`;  
14 }  
15  
16 let fei = new mahasiswa('Fei', 15);
```

04

Class

```
1 class mahasiswa {  
2   constructor(name, energy) {  
3     this.name = name;  
4     this.energy = energy;  
5   }  
6  
7   eat(portion) {  
8     this.energy += portion;  
9     console.log(`halo ${this.name}, selamat makan`);  
10 }  
11  
12   play (hour) {  
13     this.energy -= hour;  
14     console.log(`halo ${this.name}, selamat main`);  
15   }  
16  
17 let fei = new mahasiswa('Fei', 20);
```



Execution Context, Hoisting, and Scope

Pada JavaScript terdapat 2 phase:

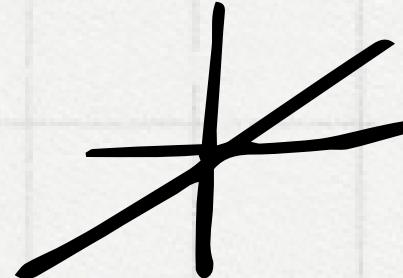
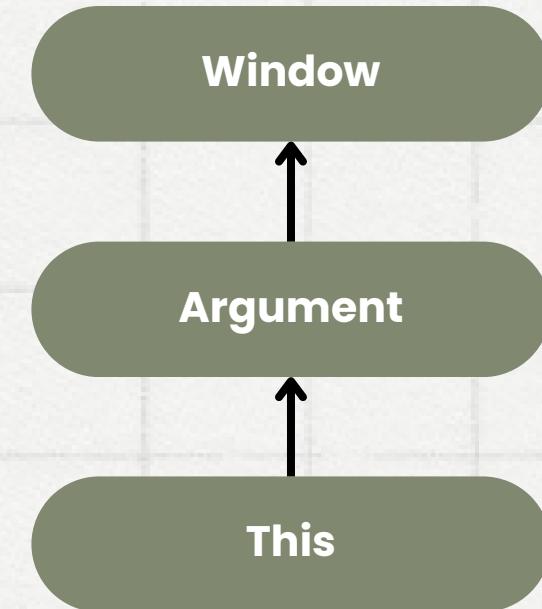
1. Creation phase
2. Execution phase

Pada **creation phase** terjadi hoisting untuk mencatat seluruh variable dan function yang terdapat pada code namun variable belum diinisialisasi (masih kosong) dan function belum dipanggil.

Execution phase dijalankan setelah creation phase dengan aturan top-bottom. Jika terdapat function yang dipanggil, maka akan terbentuk local execution context.

Scope dari variable akan menentukan apakah variable tersebut akan diletakkan pada local function (this), object argument, atau global (window)

Urutan pengecekan:



Closure

Kombinasi function dan lexical scope dalam function yang dapat memiliki data dari parent function.

Manfaat closure:

- Membuat function factory
- Seolah-olah memiliki private method/variable

```
1  function salam(waktu) {  
2      return function(nama) {  
3          console.log(`Halo ${nama}, selamat ${waktu}`);  
4      }  
5  }  
6  
7 // function factory  
8 let selamatPagi = salam('pagi');  
9 let selamatSiang = salam('siang');  
10 let selamatMalam = salam('malam');  
11  
12 selamatPagi('Fei');  
13 selamatSiang('Fangeline');
```

★ Arrow Function

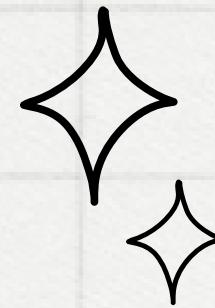
Digunakan untuk membuat regular function menjadi lebih singkat.

Arrow function tidak memiliki konsep this dan akan memanfaatkan lexical scope yang didapatkan dari parent functionnya.

Jika arrow function dibuat dengan menggunakan function expression, maka function tersebut tidak akan terkena hoisting.

```
22
> fei.sayHello
< f () {
    console.log(`Halo, nama saya ${this.nama} dan saya berumur ${this.umur} tahun`);
}
>
```

```
● ● ●
1 const Mhs = function() {
2     this.nama = 'Fei';
3     this.umur = 21;
4     this.sayHello = function () {
5         console.log(`Halo, nama saya ${this.nama} dan saya berumur ${this.umur} tahun`);
6     };
7
8     // arrow function yang langsung diinvoke
9     ((() => console.log(++this.umur))());
10 }
11
12 const fei = new Mhs();
```



Higher Order Function

JS

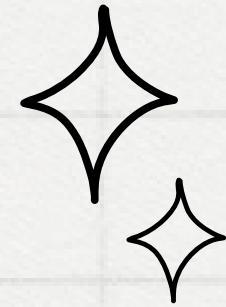
Merupakan function yang mengoperasikan function lain yang diterima sebagai argumen (callback) untuk mengabstraksi kompleksitas code yang terdapat dalam callback tersebut.

Contoh:

- 1.Array.prototype.map()
- 2.Array.prototype.filter()
- 3.Array.prototype.reduce()

```
● ○ ●  
1 // 1. filter  
2 const angka = [-1, 8, 9, 1, 4, -5, -4, 3, 2, 9];  
3 const filtered = angka.filter(a => a >= 3); //callback  
4  
5 console.log(filtered);  
6  
7 // 2. map  
8 const mapped = angka.map(a => a * 2);  
9 console.log(mapped);  
10  
11 // 3. reduce  
12 // 5 disini adalah nilai awal, defaultnya adalah 0  
13 const reduced = angka.reduce( (accumulator, value) => accumulator += value, 5);  
14 console.log(reduced);
```

```
▶ (5) [8, 9, 4, 3, 9]  
▶ (10) [-2, 16, 18, 2, 8, -10, -8, 6, 4, 18]
```



Template Literals

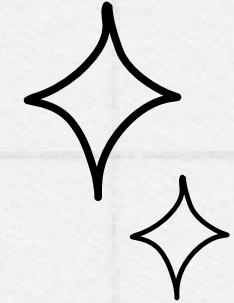
JS

Digunakan untuk membuat:

1. Multiline string
2. Embedded expression
3. Expression interpolation
4. HTML fragments
5. Tagged template:
 - a. Escaping HTML tags
 - b. Translation & internalization
 - c. Styled components

```
● ● ●  
1 // embedded expression and multiline string  
2 const nama = 'Fei';  
3 const umur = 21;  
4 console.log(`Halo, nama saya ${nama} !  
5 Saya berumur ${umur}`);  
6  
7 // Expression interpolation  
8 console.log(`Tahun depan saya akan berumur ${umur + 1}`);  
9  
10 // HTML fragments  
11 const mhs = {  
12   nama: 'Fei',  
13   umur: 21,  
14   email: 'fangeline@gmail.com'  
15 }  
16  
17 const element = `<div class="mhs">  
18   <h2>${mhs.nama}</h2>  
19   <span class="umur">Umur: ${mhs.umur}</span>  
20 </div>`;  
21 console.log(element);
```

```
● ● ●  
1 const name = 'Fei';  
2 const age = 21;  
3  
4 // rest parameter untuk menampung name dan age  
5 function coba(string, ...args) {  
6   let str = '';  
7   string.forEach((st, i) => {  
8     str += `${st}${args[i] || ''}`;  
9   });  
10  return str;  
11 }  
12  
13 // tag template akan memecah string yg ada dengan expression  
14 const str = coba`Halo, nama saya ${name}, saya berumur ${age} tahun`;  
15 console.log(str);
```



Destructuring Assignment

JS

Digunakan untuk membongkar isi dari array atau object dan dimasukkan ke dalam variable yang diinginkan. Tujuannya untuk mempermudah pengolahan data.

Destructuring dapat mempengaruhi penulisan return value dan argument yang terdapat dalam function.

```
● ● ●  
1 function calculation(a, b) {  
2     return [a + b, a - b, a / b, a * b];  
3 }  
4  
5 // urutan sangat berpengaruh  
6 const [tambah, bagi, kurang, kali] = calculation(9, 5);  
7 console.log(kurang);  
8  
9 // agar tidak berpengaruh bisa gunakan object  
10 function calculate(a, b) {  
11     return {  
12         add: a + b,  
13         sub: a - b,  
14         mul: a * b,  
15         div: a / b  
16     }  
17 }  
18  
19 const {mul, div, add, sub} = calculate(9,5);  
20 console.log(div);
```

```
● ● ●  
1 const mhs1 = {  
2     nama: 'fei',  
3     umur: 21,  
4     email: 'fei@gmail.com',  
5     nilai: {  
6         uts: 90,  
7         uas: 91  
8     }  
9 }  
10  
11 function cetakMhs({nama, umur, nilai:{uts}}) {  
12     return `nama saya ${nama} dan saya berumur ${umur} !  
13     nilai uts saya adalah ${uts}`;  
14 }
```

For..Of and For..In

For..of dan for..in memiliki tujuan yang sama yaitu untuk melakukan iterasi. Perbedaannya adalah for..of digunakan untuk melakukan iterasi pada array atau iterable object sedangkan for..in digunakan untuk melakukan iterasi pada enumerable, misalnya property pada object.

```
1 /* FOR..OF */
2 // string
3 const nama = 'Fei';
4 for(const n of nama) {
5     console.log(n);
6 }
```

```
1 /* FOR..IN
2 Membuat sebuah loop yang hanya dapat mengiterasi enumerable*/
3 const mahasiswa = {
4     nama: 'Fei',
5     umur: 21,
6     email: 'fei@gmail.com'
7 }
8
9 for(m in mahasiswa) {
10     console.log(mahasiswa[m]);
11 }
```

Spread Operator vs Rest Parameter

Spread operator dan rest parameter memiliki cara penulisan yang sama yaitu menggunakan (...arg) namun penggunaannya berbeda.

Rest parameter digunakan untuk menerima argumen yang jumlahnya tidak pasti dalam sebuah function.

```
● ● ●  
1 function filterBy(tipe, ...unfiltered) {  
2   return unfiltered.filter(x => typeof x === tipe);  
3 }  
4  
5 console.log(filterBy('number', 1, 2, 'Fei', true, 5, false, 'Fei lagi'));
```

Spread operator digunakan untuk memecah iterables menjadi single element yang biasa digunakan untuk menggabungkan array.

```
● ● ●  
1 const mahasiswa = ['Fei', 'Jessie', 'Momo'];  
2 console.log(...mahasiswa[0]);
```

Asynchronous

JavaScript merupakan programming language yang berjalan dengan menggunakan single thread namun mensupport adanya non-blocking dan asynchronous concurrent thread yang berjalan agar dapat menjalankan task lain yang tidak perlu menunggu proses tertentu agar dapat dieksekusi.

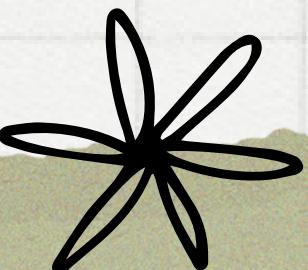
Terdapat beberapa async callback, yaitu DOM, ajax (XMLHttpRequest), setTimeout, setInterval.

Penggunaan callback yang terlalu banyak dengan library external dapat menyebabkan **callback hell**.

Contoh sederhana:

```
● ● ●  
1 console.log('satu');  
2 setTimeout(() => {  
3     console.log('dua')  
4 }, 5000);  
5 console.log('tiga');
```

```
satu  
tiga  
dua
```



Fetch

Sebuah method pada API milik JS untuk mengambil data secara asynchronous seperti ajax dan akan mereturn promise yang berisi response untuk merepresentasikan keberhasilan atau kegagalannya.

Parameter dari fetch adalah resource dan init (optional)

Aksi (3):

- Then: untuk menerima response resolve (fulfilled)
- Catch: untuk menerima response reject (rejected)
- Finally: untuk menandakan status pending sudah selesai

```
● ● ●  
1 fetch('http://www.omdbapi.com/?apikey=769bff8&i=' + imdb)  
2 .then(res => res.json())  
3 .then(res => console.log(res));
```

```
{Title: 'The Avengers', Year: '2012', Rated: 'PG-13', Released: '04 May 2012', Runtime: '143 min', ...}  
i  
Actors: "Robert Downey Jr., Chris Evans, Scarlett Johansson"  
Awards: "Nominated for 1 Oscar. 38 wins & 81 nominations total"  
BoxOffice: "$623,357,910"  
Country: "United States"  
DVD: "22 Jun 2014"  
Director: "Joss Whedon"  
Genre: "Action, Sci-Fi"  
Language: "English, Russian"  
Metascore: "69"  
Plot: "Earth's mightiest heroes must come together and learn to fight as a team if they are going to...  
Poster: "https://m.media-amazon.com/images/M/MV5BNDYxNjQyMjAtNTdiOS00NGYwLWFmNTAtNThmYjU5ZGI2YTI1XkEy  
Production: "N/A"  
Rated: "PG-13"
```

Error Handling

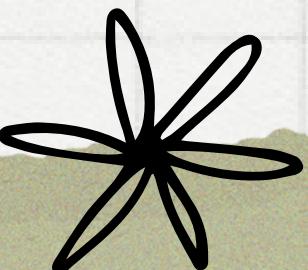
Error handling merupakan aksi yang dapat dilakukan untuk menghandle adanya error pada program (dapat dicustomize)

Pada fetch kita dapat menggunakan .then dan .catch untuk menghandle response yang didapatkan.

Pada async await function kita dapat menggunakan try catch untuk menghandle error yang diberikan.

```
● ● ●  
1 searchbtn.addEventListener("click", async function () {  
2   const key = document.querySelector(".input-keyword");  
3  
4   try {  
5     const movies = await getMovies(key.value);  
6     console.log(movies);  
7  
8     updateUI(movies);  
9   } catch (error) {  
10     alert(error);  
11   }  
12});
```

```
● ● ●  
1 janji1  
2 .then(res => console.log('OK!: ' + res))  
3 .catch(res => console.log('NOT OK!: ' + res));
```

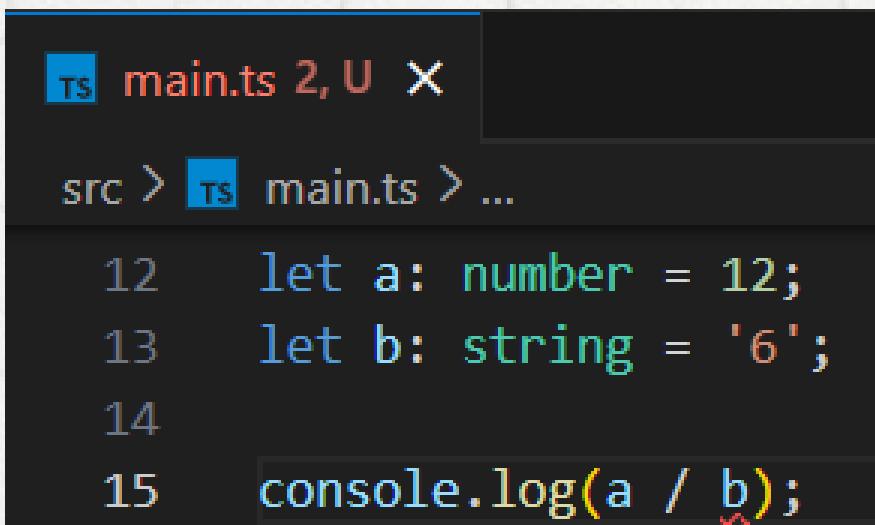


 TS

TypeScript —

TypeScript (TS) merupakan strongly typed programming language yang merupakan superset dari JavaScript. Dibuat oleh Microsoft pada tahun 2012 dengan tujuan untuk membuat bahasa pemrograman yang lebih scalable. JS disebut juga sebagai weakly typed programming language karena kita tidak perlu menspesifikasikan secara detail setiap tipe data dari variable yang ada.

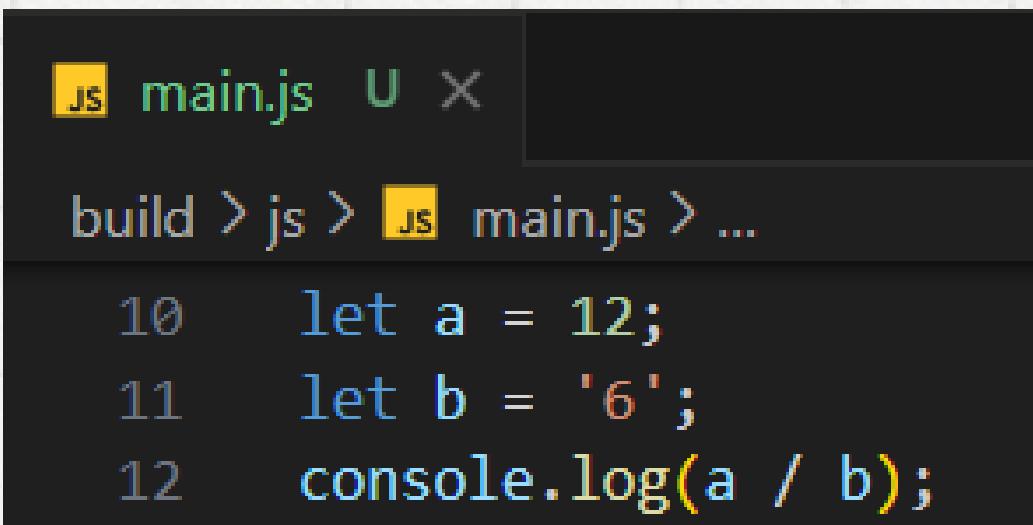
TypeScript digunakan untuk menangkap error yang terjadi saat development code dan mencegah terjadinya bug.



TS main.ts 2, U X

src > TS main.ts > ...

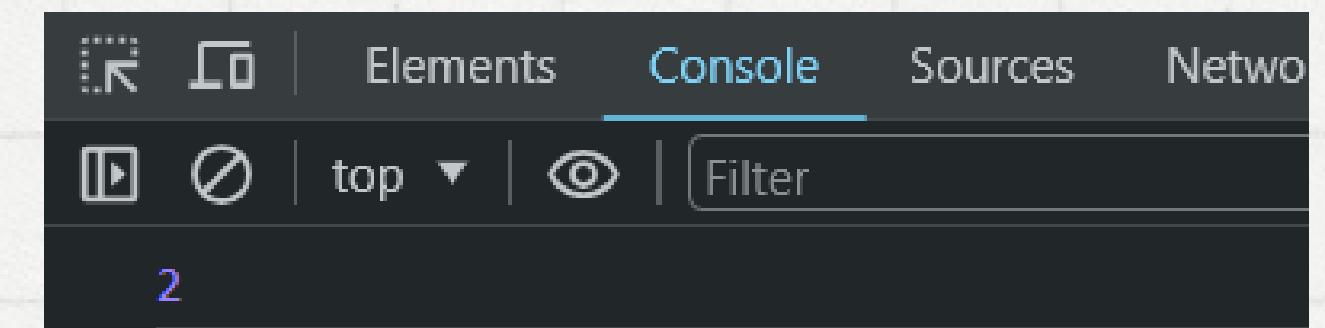
```
12 let a: number = 12;
13 let b: string = '6';
14
15 console.log(a / b);
```



JS main.js U X

build > js > JS main.js > ...

```
10 let a = 12;
11 let b = '6';
12 console.log(a / b);
```



Type and Aliases



Type digunakan untuk membuat alias yang dapat digunakan untuk membuat spesifikasi tipe data tertentu untuk mengimprove readability dari code.

```
1 type stringOrNumber = string | number;
2 type strOrNumArray = (string | number)[]
3
4 type Artist = {
5     // optional variable
6     name?: string,
7     active: boolean,
8     masterpieces: strOrNumArray
9 }
10
11 let userId: stringOrNumber;
12 userId = 'Fei'
13 userId = true;
```

Function and Type Assertion



Function pada TypeScript dapat menspesifikasi return type dan parameter typenya.

Type assertion digunakan untuk memberitahu TypeScript secara eksplisit tipe dari variable saat TypeScript tidak dapat menentukan secara implisit tipe dari variable tersebut



```
1 const sumAll = (a: number = 3, b: number, c: number = 5): number => a + b + c;  
2  
3 console.log(sumAll(10, 4, 10));  
4 console.log(sumAll(undefined, 5));
```



```
1 type One = string;  
2 type Two = string | number;  
3 type Three = 'hello';  
4  
5 let e: One = 'hello';  
6 let f = e as Two; //convert to less specific types  
7 let g = e as Three; //convert to more specific type
```

classes



Classes pada TypeScript dapat menggunakan extends untuk abstract class dan implements untuk interface.

```
● ● ●  
1  name: string,  
2  instrument: string,  
3  // method yg paramnya string dan return string  
4  play(action: string): string  
5 }  
6  
7 class Guitarist implements Musician {  
8   name: string;  
9   instrument: string;  
10  
11  constructor(name: string, instrument: string) {  
12    this.name = name,  
13    this.instrument = instrument;  
14  }  
15  
16  public play(action: string): string {  
17    return `${this.name} ${action} the ${this.instrument}`;  
18  }  
19 }  
20  
21 const Eddie = new Guitarist('Eddie', 'guitar');  
22 console.log(Eddie.play('strums'));
```

Index Signature



Index Signature digunakan untuk melakukan spesifikasi pada properties pada object yang belum diketahui akan memiliki property apa saja.

Kita dapat menggunakan **keyof** untuk membantu mengakses property dari sebuah object

```
1 interface transactionObj {  
2     readonly [index: string]: number  
3     Pizza: number,  
4     Books: number,  
5     Job: number  
6 }  
7  
8 const todaysTransaction: transactionObj = {  
9     Pizza: -10,  
10    Books: -5,  
11    Job: 50,  
12    // Fei optional  
13    Fei: 21  
14 }
```

Generics

Generics merupakan placeholder yang digunakan agar kita dapat bekerja dengan functions, classes, maupun interfaces dengan berbagai jenis data types namun dengan masih memperhatikan type safety.

```
1  class StateObject<T> {  
2  
3      private data: T  
4  
5      constructor(value: T) {  
6          this.data = value;  
7      }  
8  
9      get state(): T {  
10         return this.data  
11     }  
12  
13     set state(value: T) {  
14         this.data = value  
15     }  
16 }  
17  
18 const store = new StateObject('fei');  
19 console.log(store.state)  
20 store.state = 'feiii'  
21 // store.state = 21  
22  
23 const myState = new StateObject<(string | number | boolean)[]> ([15, 16]);  
24 myState.state = ([ 'fei', 21, true])  
25 console.log(myState.state)
```

Utility Types

Utility Types merupakan generic types yang sudah di-predefined oleh TypeScript dan menyediakan operasi yang umum diperlukan pada data types tanpa harus melakukan konfigurasi tambahan

Utility types:

- Partial
- Required dan Readonly
- Pick and Omit
- Exclude dan Extract
- ReturnType
- Parameters
- Awaited

Partial - beberapa dari keseluruhan interface Assignment

```
● ● ●  
1 interface Assignment {  
2   studentId: string,  
3   title: string,  
4   grade: number,  
5   verified?: boolean  
6 }  
7  
8 const updateAsg = (assign: Assignment, propsToUpdate: Partial<Assignment>):  
9 Assignment => {  
10   return { ...assign, ...propsToUpdate }  
11 }
```

Required - keseluruhan Assignment

Readonly - tidak dapat assign value baru

```
● ● ●  
1 const assignGraded: Assignment = updateAsg(assign, { grade: 95 })  
2  
3 const recordAssignment = (assign: Required<Assignment>): Assignment => {  
4   return assign;  
5 }  
6 recordAssignment({...assignGraded, verified: true})  
7  
8 const assignVerified: Readonly<Assignment> = { ...assignGraded, verified: true }  
9 console.log(assignVerified)
```

Utility Types

TS

Pick - beberapa props dari Assignment

Omit - mengabaikan props tertentu dari Assignment obj



```
1 // AssignResult -> studentId dan grade
2 type AssignResult = Pick<Assignment, "studentId" | "grade">
3 const score: AssignResult = {
4   studentId: '1111111',
5   grade: 100,
6 }
7
8 // AssignPreview -> selain grade dan verified
9 type AssignPreview = Omit<Assignment, "grade" | "verified">
10 const preview: AssignPreview = {
11   studentId: '324234234',
12   title: 'Final Project',
13 }
```

ReturnType - memberikan return type berdasarkan parameter function

Parameters - memberikan parameter dengan type tertentu



```
1 type newAssign = ReturnType<typeof createNewAssign>
2 const createNewAssign = (title: string, points: number) => {
3   return { title, points }
4 }
5 const tsAssign: newAssign = createNewAssign("Utility Types", 800)
6
7 // Parameters
8 type assignParams = Parameters<typeof createNewAssign>
9 const assignArgs: assignParams = ["Generics", 100]
10 const tsAssign2: newAssign = createNewAssign(...assignArgs)
```

Prompt: Utility types in TS

Exclude - mengecualikan type tertentu dari union types

Extract - mengambil type tertentu dari union types



```
1 type LetterGrades = "A" | "B" | "C" | "D" | "E"
2 // adjustedGrade -> selain E
3 type adjustedGrade = Exclude<LetterGrades, "E">
4 // highGrades -> A dan B
5 type highGrades = Extract<LetterGrades, "A" | "B">
```

Awaited - digunakan pada ReturnType yang mereturn Promise



```
1 const fetchUsers = async (): Promise<User[]> => {
2   const data = await fetch(
3     'https://jsonplaceholder.typicode.com/users'
4   ).then(res => {
5     return res.json()
6   }).catch(error => {
7     if(error instanceof Error) console.log(error.message)
8   })
9
10  return data;
11 }
12
13 type FetchUsersReturnType = Awaited<ReturnType<typeof fetchUsers>>
14 fetchUsers().then(users => console.log(users))
```



**Thank
you!**

[GitHub Repo: JavaScript lanjutan & TypeScript](#)