



OpenLibrary.id
Perpustakaan Digital Terbuka

Belajar Cepat Typescript

Gun Gun Febrianza

This is My Contribution for Open Library Concept.



Sebuah konsep Perpustakaan Digital Terbuka untuk membantu mempermudah siapapun untuk mengakses ilmu pengetahuan dan mempelajari ilmu pengetahuan. OpenLibrary.id adalah sebuah gerakan dan konsep pemikiran yang penulis usung sebagai wadah tempat untuk mengabdikan kepada masyarakat melalui kontribusi karya tulis. Karya tulis yang diharapkan dapat membantu agar minat baca jutaan pemuda-pemudi di Indonesia terus meningkat. Sebab penulis percaya **dengan membaca peluang keberhasilan hidup seseorang kedepannya akan menjadi lebih besar dan membaca dapat membawa kita ketempat yang tidak pernah kita sangka-sangka yaitu tempat yang lebih baik dari sebelumnya.**

Penulis sadar gerakan ini memerlukan penulis-penulis lainnya agar tujuannya bisa tercapai dan jangkauan manfaatnya bisa lebih luas lagi. Semakin banyak penulis dari berbagai bidang keilmuan akan semakin berwarna manfaat hasil karya tulis yang bisa diberikan untuk masyarakat. Maka dari itu penulis secara terbuka mengundang siapapun yang ingin bergabung menjadi penulis di gerakan *Indonesia Open Library*, agar bisa bertemu dan saling bersilaturahmi.

***First Published 24 April 2017,
Under License Attribution-NonCommercial-ShareAlike 4.0 International.***

Belajar Cepat *Typescript* version 1.0

"Seseorang yang ingin mendapatkan mutiara harus berani menyelam kedalam samudra yang amat dalam"

- Al – Mutannabi

Metode Belajar

Ada satu hal yang harus anda ketahui, jika ingin membaca buku ini anda harus **siap untuk susah atau menikmati proses belajar** yang akan anda lakukan, sebab anda tidak akan mendapatkan ilmu pengetahuan jika tidak siap untuk susah atau meninggalkan zona nyaman. Kenapa kita harus siap susah payah dalam belajar? Kenapa kita harus meninggalkan zona nyaman?

Seperti yang dikatakan **Imam Syafii** bahwa jika seumur hidup kita tidak ingin merasakan hinanya kebodohan maka kita harus merasakan pahitnya pendidikan. (Belajar dan Menuntut Ilmu). Penekanan ini ditegaskan lagi oleh **Sayyidina Ali bin abu thalib**, *“Knowledge is not attained in comfort”* yang artinya bahwa ilmu pengetahuan tidak akan bisa didapatkan melalui kenyamanan.

Setelah melakukan beberapa kali penelitian metode belajar saya membuat kesimpulan ada 5 Langkah dasar yang harus kita lakukan agar kita bisa faham apa yang ingin kita pelajari.

1. Baca
2. Fahami
3. Hafal
4. Praktek/Catat
5. Pastikan Faham dan Hafal (verifikasi akhir)

Metacognition

Tapi sebelum mengeksekusi semua hal yang ada diatas masih ada yang harus diketahui yaitu seberapa jauh **Metacognition** anda untuk belajar. *Metacognition* adalah soal seberapa besar kepekaan anda menganggap suatu hal adalah sesuatu yang sangat penting. Jika itu benar benar sangat penting kita akan mengingatnya (tersimpan dalam *long term memory*) tanpa harus menghafalnya dan timbul rasa motivasi yang besar untuk belajar.

Sebagai contoh dokter memberikan anda resep obat, namun dokter memberi peringatan jika anda lupa instruksi resepnya maka anda akan gagal dalam melakukan pengobatan yang berujung kematian. Saking takutnya anda pasti **mengingatnya** karena anda menganggapnya ini benar benar PENTING. Begitulah cara agar otak kita berada dalam optimal untuk memahami sesuatu.

Metacognition juga akan membuat kita terbayang tentang efek domino, jika saya tidak faham ini maka saya tidak akan faham itu? jika saya tidak faham itu saya tidak akan faham hal hal ini? jika saya tidak faham ini bagaimana saya bisa bernilai dan berhasil? Dan seterusnya...

Reason

Seorang adik kecil bertanya kepada saya,
Kenapa kaka selalu bisa semangat dan optimis belajar?

Hening sejenak, entah kenapa pertanyaan sederhana ini sulit sekali dijawab. Tapi ini benar benar sulit sebab prosesnya kompleks. Jadi sambil tertawa saya mengucapkan,

"kalau tidak optimis, entar kaka rugi bandar!"

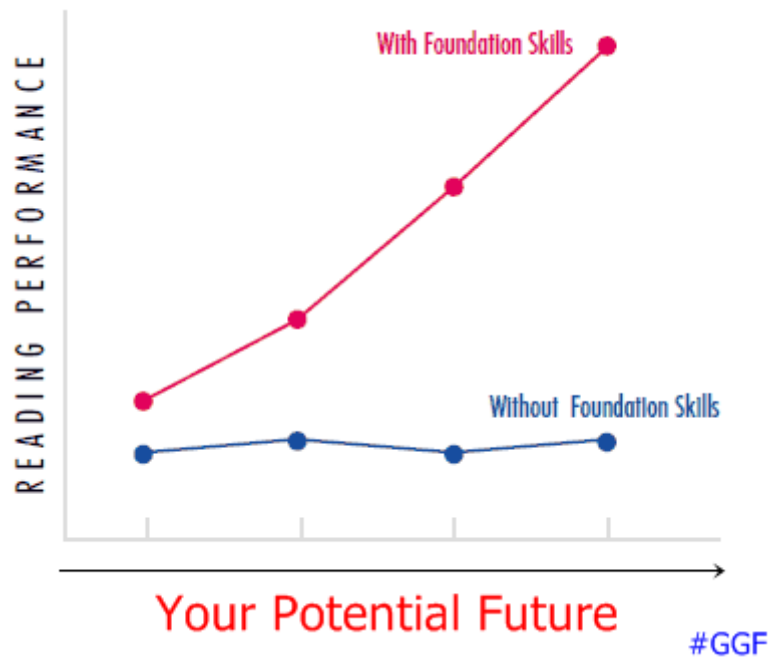
Di malam yang hening, saya memikirkan pertanyaan tadi.
Mencoba menyederhanakan solusi tanpa kehilangan substansi.
Solusi yang bisa mendekati solusi universal,
ya ini komitmen yang telah saya lalui.

Temukan alasan, kenapa kamu harus mencapai cita-citamu.
Temukan alasan, kenapa kamu harus mampu melakukannya.
Temukan alasan, kenapa kamu harus percaya diri.
Temukan alasan, kenapa kamu tidak takut pada kegagalan.
Temukan alasan, kenapa kamu berani berjuang dalam ketidakpastian.
Temukan alasan, kenapa kamu berani berjuang susah payah.
Temukan alasan, kenapa kamu siap berjuang dalam kerumitan.
Temukan alasan, kenapa kamu siap menghadapi orang-orang yang akan mempersulit perjuanganmu?

"Reason is the most powerful human driver in the world"

[#GGF](#) - CTO IEL.

Matthew Effect in Reading



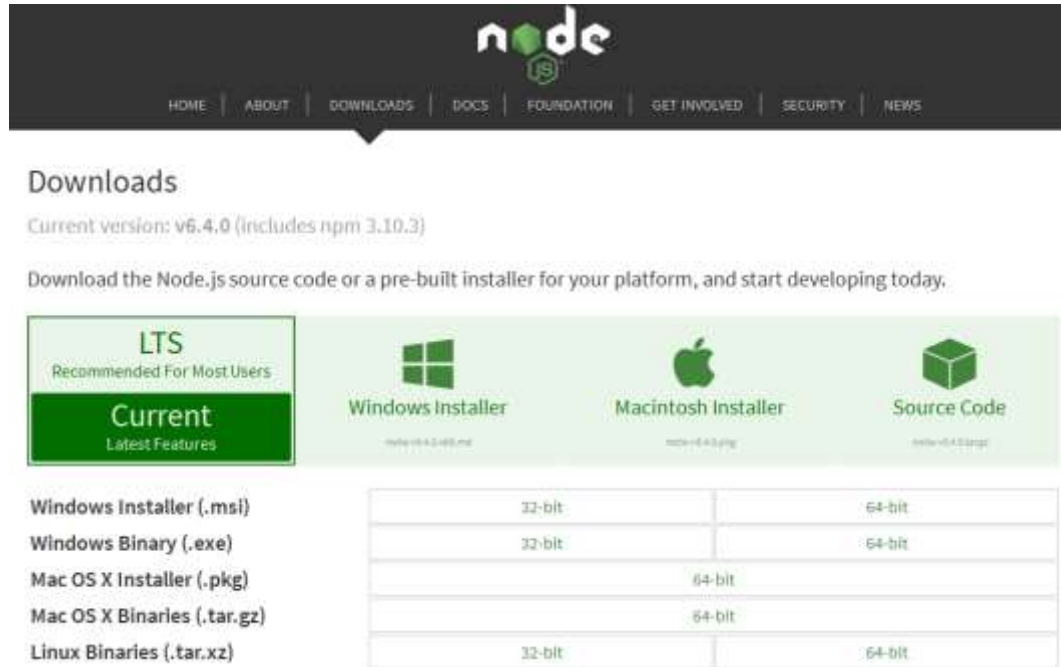
Bagi penulis membaca adalah ladang segala keberuntungan sebab ia membawa kehidupan kita ke arah yang tidak disangka-sangka dan tidak diduga-duga, sebuah kehidupan yang lebih baik dari sebelumnya. Memperbanyak membaca, memahami dan praktis artinya memperbaiki kehidupan yang hendak kita miliki.

Persiapan

Apa saja yang harus disediakan untuk membaca buku ini?

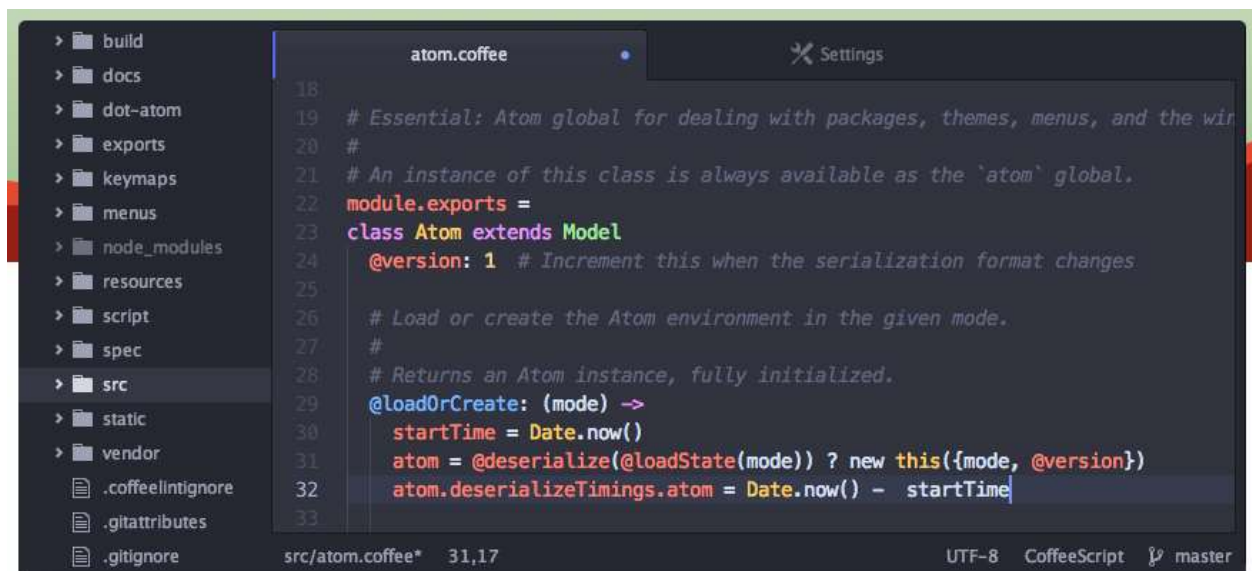
1. *Install Node.js*

Silahkan *download* di <https://nodejs.org/en/download/current/>



2. *Install Atom Editor*

Silakan *download* di <https://atom.io/>



3. *Install Google Chrome atau Firefox.*

Silahkan anda *googling* kunjungi situs resmi *firefox* atau *google*. Penulis menyarankan anda menggunakan *browser firefox*, namun jika anda sudah terbiasa dengan *google chrome developer tool* silahkan menggunakan *google chrome*.

Untuk Siapa Buku Ini?

Untuk siapapun yang telah mempelajari bahasa pemrograman *javascript* dan ingin mengembangkan kemampuannya agar bisa lebih dalam mempelajari dunia *web* dan *mobile application*. *Programmer* yang sudah mempelajari konsep *object oriented programming* akan lebih cepat memahami buku ini.

Konvensi Penulisan?

1. Setiap tulisan yang di beri **bold**, bermakna agar pembaca fokus pada konteks yang sedang dibahas di dalam buku ini.
2. Untuk setiap terminologi baru yang muncul untuk pertama kali akan diberi warna biru dan bold, contoh ***compiler***.
3. Setiap kode pemrograman atau perintah dalam sebuah *command line* akan disimpan di dalam sebuah box berwarna orange seperti dibawah ini :

```
var nama: string = "Gun Gun Febrianza";
```

4. Jika sebuah potongan kode pemrograman dan perintah dalam *command line* ditulis diantara suatu paragraf, misal untuk menjelaskan sebuah fungsi maka bentuk font yang akan digunakan adalah *font* consolas dengan ukuran 10 dan efek bold seperti pada contoh di bawah ini **fungsiTulisNama()** .
5. Setiap informasi penting yang harus dicatat dan diingat oleh pembaca akan disimpan kedalam sebuah tabel seperti pada tabel di bawah ini :

Notes
Jangan lupa simpan <i>file</i> , agar data tersimpan jika listrik padam !

6. Untuk hasil dari setiap proses yang muncul dalam sebuah terminal, akan disimpan di dalam tabel berwarna hitam dan teks warna putih :

```
nama:Maudy
```

```
namaKepanjangan:Ayunda
```

7. Untuk setiap interaksi dengan *keyboard* atau *mouse*, seperti *hotkey* atau *mouse click* maka instruksi akan diberi tanda dengan **bold** warna merah, contoh **CTRL+SHIFT+K**.

Feedback?

Jika ada feedback yang bersifat private silahkan diajukan melalui email saya gungunfebrianza@gmail.com

Kode Sumber?

<https://github.com/gungunfebrianza/learning-typescript>

Terdapat Kesalahan?

Silahkan ajukan isu :

<https://github.com/gungunfebrianza/learning-typescript>

Pertanyaan, Kritik dan Saran?

Seluruh pertanyaan silahkan diajukan melalui grup WebMobDevel.id :

<https://web.facebook.com/groups/1972240079700480/>

Pertanyaan sengaja dialihkan ke grup agar anda bisa mendapat bantuan dari sesama anggota lainnya. Bisa berdiskusi untuk berkenalan dengan developer lainnya untuk mempermudah proses belajar anda. Kritik dan saran terbaik dari pembaca akan saya tampilkan pada edisi revisi.

Kritik dan saran diperlukan agar *ebook* ini menjadi lebih baik lagi & terus berkembang.

Tentang Penulis?

Gun Gun Febrianza seorang pelajar muda dari kota Bandung yang aktif dalam gerakan penyayang lingkungan dan hewan, penulis sudah mulai menyukai bahasa pemrograman saat masih duduk di bangku SMA. Bahasa yang sempat penulis pelajari saat itu adalah pascal, cobol dan fortran.

Saat ini penulis adalah alumni S1 Teknik Informatika Universitas Komputer Indonesia. Penulis mengambil penelitian skripsi berjudul “*Compiler for Programming in Indonesian Language*” atau Kompiler Untuk pemograman dalam bahasa Indonesia yang diimplementasikan untuk *processor* x86-x64 Intel.

Penulis senang sekali belajar dan penelitian di dunia komputer, saat masih duduk dibangku kuliah penulis aktif menulis ebook pemrograman, membangun fablab dan riset di bidang teknologi *embedded system*. Alhamdulillah, penulis telah mendapatkan beasiswa dari seorang pengusaha dermawan, tahun depan tepat di tahun 2018 penulis berencana untuk melanjutkan studi ke University of Kaiserslautern di Jerman, dengan fokus study di bidang *embedded system*. Minta doanya dari para pembaca, sepulang dari sana insya allah akan lebih giat lagi menulis buku di bidang ilmu komputer.



1. Setup Learning Environment

1. Node.js

Setelah anda berhasil melakukan instalasi node.js anda sudah bisa menggunakan *javascript* sebagai bahasa *server-side*. Node.js bekerja sebagai *javascript* runtime yang akan membaca sebuah *javascript* code untuk keperluan *server*, sebuah *server* yang dibangun dengan *javascript*. Node.js berdiri di atas engine *javascript* yang telah dioptimasi untuk melakukan kompilasi dan eksekusi kode *javascript*. Engine *JavaScript* itu bernama *Google V8 Javascript engine* yang dibuat oleh tim *google developer* menggunakan bahasa pemrograman C++. Jika anda telah mempelajari C++ anda bisa menggunakan *engine* tersebut untuk diterapkan dalam aplikasi yang anda buat.



Kembali lagi ke node.js, di dalam node js terdapat *package* manager yang dapat mempermudah kita untuk mengembangkan setiap aplikasi yang ingin kita buat. *Package* manager untuk node.js bernama *Node Package Manager* atau disingkat *NPM*. Sekarang kita akan praktek untuk melakukan instalasi *typescript* menggunakan *NPM*, perhatikan perintah di bawah ini :

```
npm install -g typescript
```

Perintah di atas digunakan agar *NPM* melakukan instalasi *typescript package*, parameter *-g* digunakan agar program tersebut terinstals secara global dalam sistem bukan dalam sekup lokal misalkan dalam sebuah *folder*. Jika kita melakukan instalasi secara global perintah yang ada di dalam sebuah *package* manager terdaftar dalam sistem kita, sehingga kita bisa menggunakannya dimana saja melalui *command line*.

Setelah terinstal di dalam sistem, kita akan memiliki sebuah *typescript compiler* bernama *tsc* yang akan kita gunakan untuk mengkompilasi sebuah *typescript* kedalam sebuah *plain javascript*. Sekarang kita akan mencoba mengeksekusi perintah di bawah ini di dalam *command line* :

```
tsc -v
```

Maka akan muncul informasi versi dari *typescript compiler* yang kita miliki :

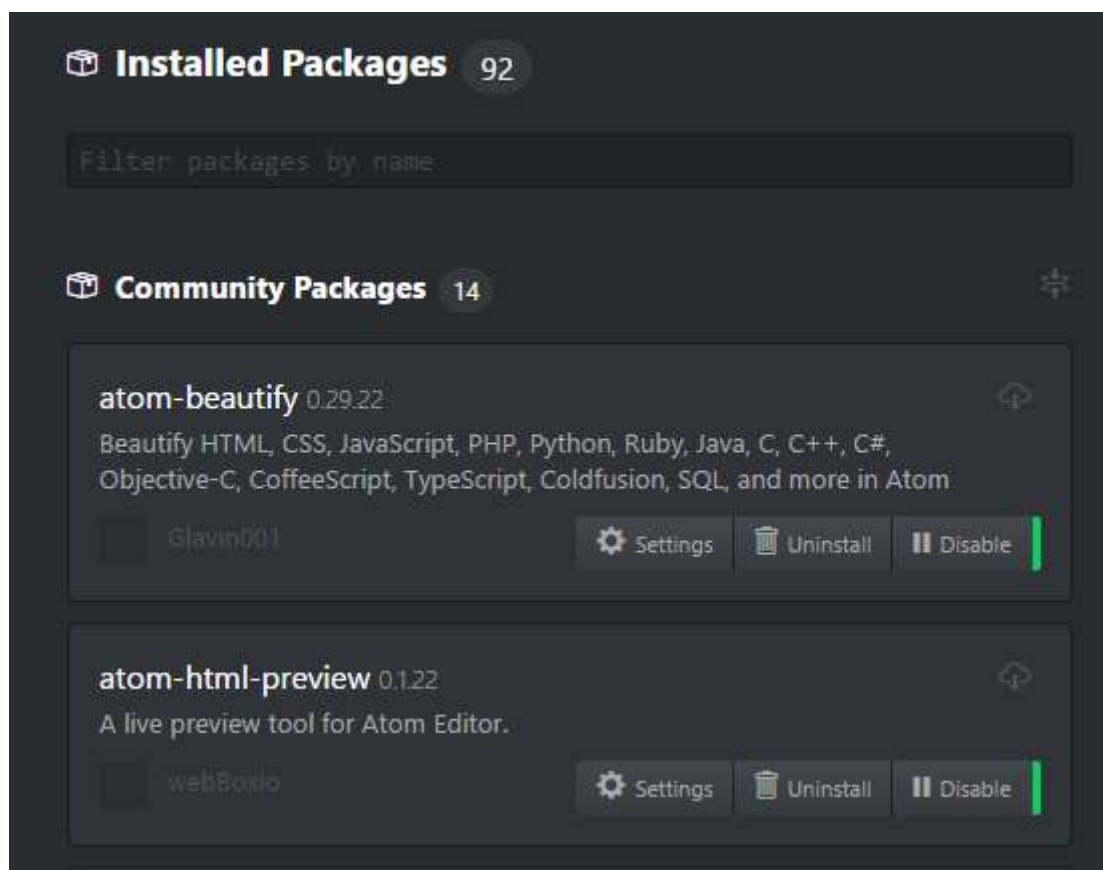
```
C:\Users\sambaladokrist>tsc -v
Version 2.2.2
C:\Users\sambaladokrist>
```

2. Atom Editor

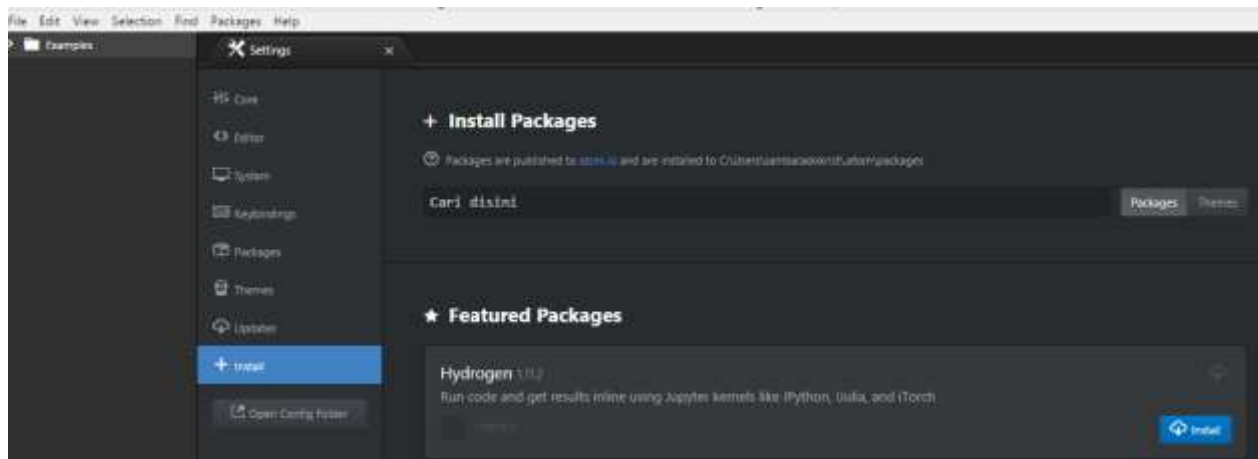
Atom Editor adalah kode editor yang *hackable*, maksudnya adalah kita bisa ikut mengembangkannya, mengatur apa saja yang harus ada di dalam kode editor, dan memaksimalkan fungsi kode editor ketika kita sedang bekerja untuk suatu bahasa pemrograman.

Jika anda telah berhasil melakukan instalasi, di dalamnya terdapat konsep **Package Manager** juga yang bisa kita gunakan untuk menambahkan beberapa fitur ke dalam *atom code editor*. Misal menambahkan fitur **Realtime HTML Viewer**, sehingga atom kode editor yang kita gunakan memiliki fitur untuk melihat tampilan web secara *realtime*. Mengubah tema dari kode editor dalam gelap ke terang dan banyak sekali *package* menarik lainnya yang bisa kita gunakan.

Di bawah ini penulis telah memasang **atom-beautify** dan **atom-html-preview** ke dalam atom kode editor sehingga jika kode pemograman untuk web penulis berantakan bisa dirapihkan, sehingga mudah untuk di baca dan **atom-html-preview** agar kode editor penulis memiliki fitur untuk melihat tampilan *web* secara *realtime*.



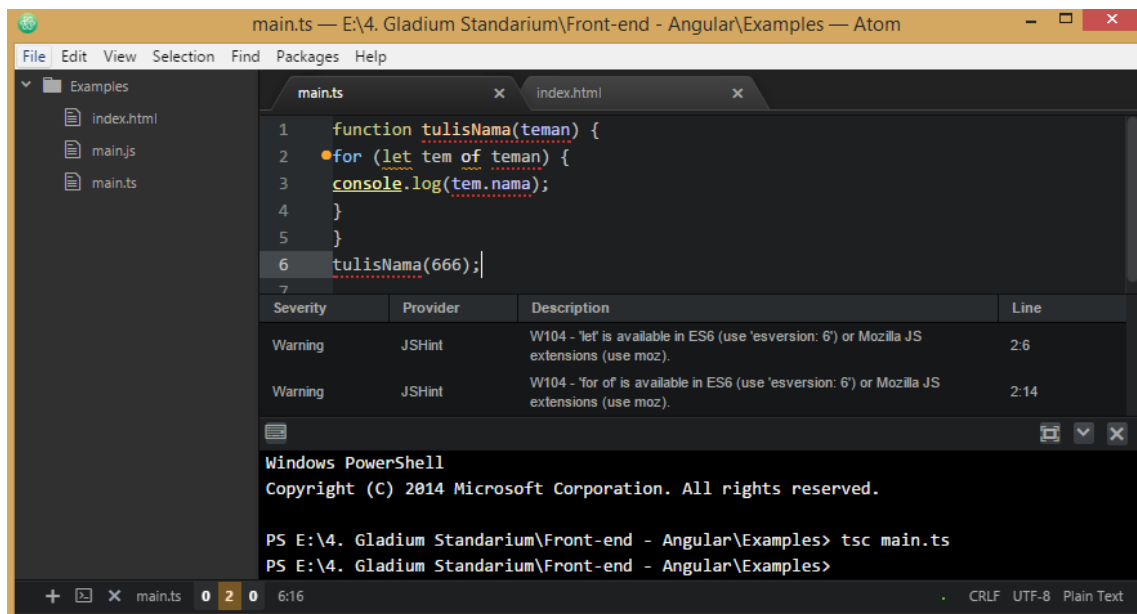
Jika anda ingin mencari dan memasang sebuah *package* pilih menu **File -> Settings** kemudian setelah muncul tab *settings* pilih menu *install* dan anda tinggal mengeksplorasi *package* yang ingin kita pasang. Cara nama *package*, jika sudah ditemukan baru tekan tombol *install* pada *package* yang telah anda pilih. Untuk mengetahui lebih detail tentang suatu *package*, cara penggunaan *package* dan *source code* klik nama *package* yang ingin anda ketahui.



Ada beberapa *Package* yang wajib untuk pembaca pasang :

1. atom-beautify
2. atom-html-preview
3. linter-csslint
4. linter-htmlhint
5. linter-jshint
6. minimap
7. minimap-athider
8. pigments
9. platformio-ide-terminal
10. highlight-selected
11. compare-files
12. atom-typescript

Di bawah ini adalah contoh tampilan kode editor penulis yang benar benar mempermudah proses menulis kode.



3. *Web Browser*

Anda bisa menggunakan *google chrome* atau *firefox*, pada buku ini penulis menggunak *browser firefox*.

4. *Summary*

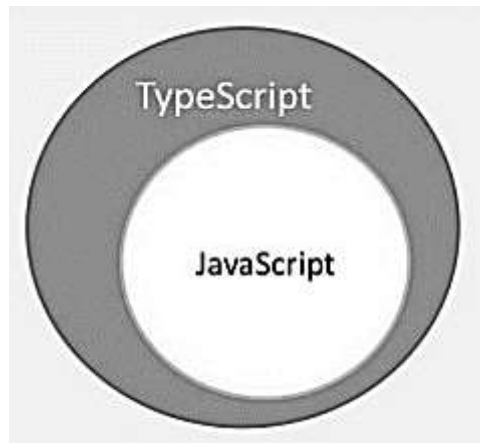
Hari ini kita telah mengenal :

1. Node.js
2. NPM
3. *Typescript* Kompiler
4. *Atom Editor*
5. *Package Manager Atom*

2. *Typescript Cheat*

1. Pengenalan *Typescript*

Typescript adalah bahasa open source yang dikembangkan oleh *microsoft*. *Typescript* sering kali disebut sebagai *superset* dari *javascript*, istilah *superset* yang dimaksud mengacu pada istilah *superset* dalam teori himpunan.



Jika anda melihat gambar di atas pasti faham, maksudnya adalah setiap *javascript code* adalah *typescript* dan terdapat beberapa kelebihan yang tidak tersedia dalam sebuah *javascript (Extended Javascript)*. Kelebihan ini digunakan untuk mempermudah pengembangan aplikasi *web* skala besar yang ditulis menggunakan bahasa pemrograman *javascript*. Di dalam *typescript* kita akan mempelajari konsep seperti *classes*, *modules*, *interface*, *generics*, dan *static typing* menggunakan *javascript*.

2. *Typescript Cheatsheet*

Dibawah ini adalah *cheatsheet* yang akan kita lihat dengan cepat,

Terdapat 3 Tipe data dalam *typescript* :

```
var salah: boolean = false;
var angka: number = 42;
var nama: string = "Anders";
```

Selain ketiga tersebut kita bisa menggunakan *tipe any* yang bisa di isi apa saja.

```
var hmm: any = 4;
hmm = "maybe a string instead";
hmm = false;
```

Contoh sebuah *collection*, di dalam *typescript* terdapat terdapat *typed array*

```
var list: number[] = [1, 2, 3];
```

Contoh sebuah *collection*, di dalam *typescript* terdapat *generic array*

```
var list: Array<number> = [1, 2, 3];
```

Contoh sebuah *Enumeration* :

```
enum Color {Red, Green, Blue};  
var c: Color = Color.Green;
```

Contoh sebuah *function* dengan return void :

```
function handsome(): void {  
    alert("Say me handsome");  
}
```

Contoh *lambda* dan *type inference*

```
var f1 = function(i: number): number { return i * i; }  
var f2 = function(i: number) { return i * i; }  
var f3 = (i: number): number => { return i * i; }  
var f4 = (i: number) => { return i * i; }  
var f5 = (i: number) => i * i;
```

Contoh *interface*, kita bisa membuat opsional *properties* yang ditandai dengan simbol ?

```
interface Manusia {  
    nama: string;  
  
    umur?: number;  
  
    makan(): void;  
}
```

Contoh *Object* hasil implementasi dari *interface* di atas

```
var m: Manusia = { nama: "Maudy", makan: () => {} };  
var m: Manusia = { nama: "Maudy", umur: 22, makan: () => {} };
```

Contoh *Interface* untuk sebuah *function*

```
interface fungsiCari {  
    (source: string, subString: string): boolean;  
}  
  
var data: fungsiCari;  
data = function(src: string, sub: string) {  
    return src.search(sub) != -1;  
}
```

Contoh sebuah *class* dengan *properties* dan *function*

```
class A {  
    x := number;  
    hitung() { return Math.sqrt(this.x * this.x + this.y * this.y); }  
}
```

Contoh sebuah *class* dengan *constructor*

```
class A {  
    x := number;  
    constructor(x: number, public y: number = 0) {  
        this.x = x;  
    }  
}
```



```
}
```

Contoh sebuah *class* dengan *static member*

```
class A {  
    x := number;  
    constructor(x: number, public y: number = 0) {  
        this.x = x;  
    }  
    static koordinat = new A(0, 0);  
}
```

Contoh sebuah *class* dengan *static member*

```
class A {  
    x := number;  
    constructor(x: number, public y: number = 0) {  
        this.x = x;  
    }  
    static koordinat = new A(0, 0);  
}
```

Contoh sebuah *object* dari *Class A*

```
var p1 = new A(10 ,20);  
var p2 = new A(25); //maka y akan bernilai 0
```

Contoh *inheritance* dan *overriding* dari *Class A*

```
class koordinat3D extends A {  
    constructor(x: number, y: number, public z: number = 0) {  
        super(x, y); //wajib menggunakan keyword super  
    }  
  
    hitung() {  
        var d = super.dist();  
        return Math.sqrt(d * d + this.z * this.z);  
    }  
}
```

Contoh sebuah *Modules*

```
module Geometry {  
    export class Kubus {  
        constructor(public lebarsisi: number = 0) {  
        }  
        area() {  
            return Math.pow(this.lebarsisi, 2);  
        }  
    }  
}  
  
var s1 = new Geometry.Square(5);
```

Contoh penggunaan alias untuk menggunakan sebuah *modules*

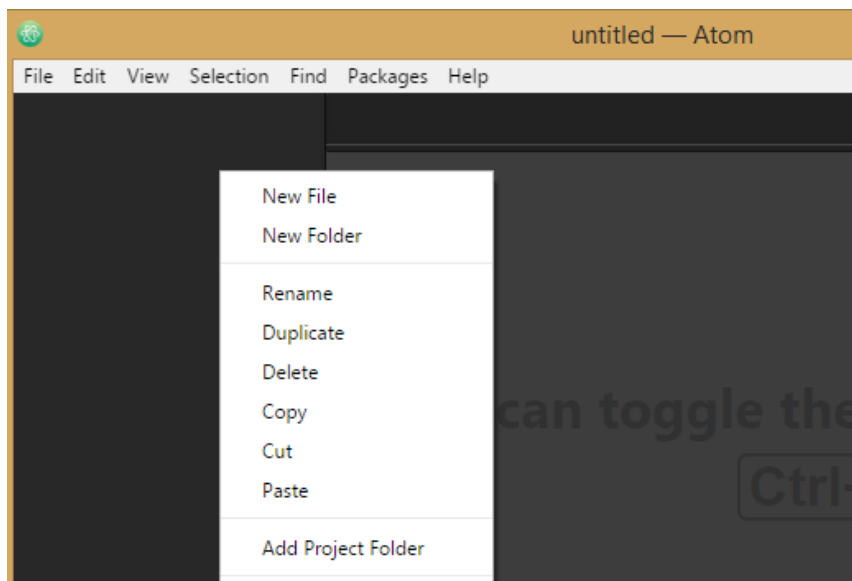
```
import G = Geometry;  
var s2 = new G.Square(10);
```

Contoh sebuah *generic* dalam *class*, *interface* dan *function*

```
class Tuple<T1, T2> {  
  constructor(public item1: T1, public item2: T2) {  
  }  
}  
  
interface Pair<T> {  
  item1: T;  
  item2: T;  
}  
  
var pairToTuple = function<T>(p: Pair<T>) {  
  return new Tuple(p.item1, p.item2);  
};  
  
var tuple = pairToTuple({ item1:"hello", item2:"world"});
```

3. Bermain dengan *Typescript*

Kita akan belajar bagaimana caranya mengkompilasi *typescript* ke dalam *javascript*. Sebelum itu pertama buka *atom editor* anda, klik kanan pada tab sebelah kiri sehingga muncul pilihan seperti yang ada pada gambar di bawah ini :



Pilih **Add Project Folder**, arahkan ke sebuah *folder* yang anda inginkan untuk menulis sebuah kode. Kemudian klik kanan lagi pilih *new file* dengan nama *main.js*. Maka akan muncul kode editor seperti pada gambar di bawah ini :

```

main.js
1  function tulisNama(teman) {
2    for (let tem of teman) {
3      console.log(tem.nama);
4    }
5  }
6  tulisNama(666);

```

Pada kode di atas terdapat sebuah *function* **tulisNama** yang memiliki 1 *parameter*, pada baris kode kedua terdapat perulangan untuk mendapatkan setiap nilai yang ada pada *object* teman dan menampilkannya satu-persatu sampai habis pada baris kode ketiga.

Kode di atas adalah kode yang salah, karena pada baris kode ke 6 fungsi **tulisNama** digunakan menggunakan *parameter number* 666, seharusnya sebuah *array*. Secara teoritis *runtime error* akan terjadi ketika *javascript* dieksekusi. Kemudian buatlah sebuah *file* bernama **index.html** dan tulis kode dibawah ini :

```

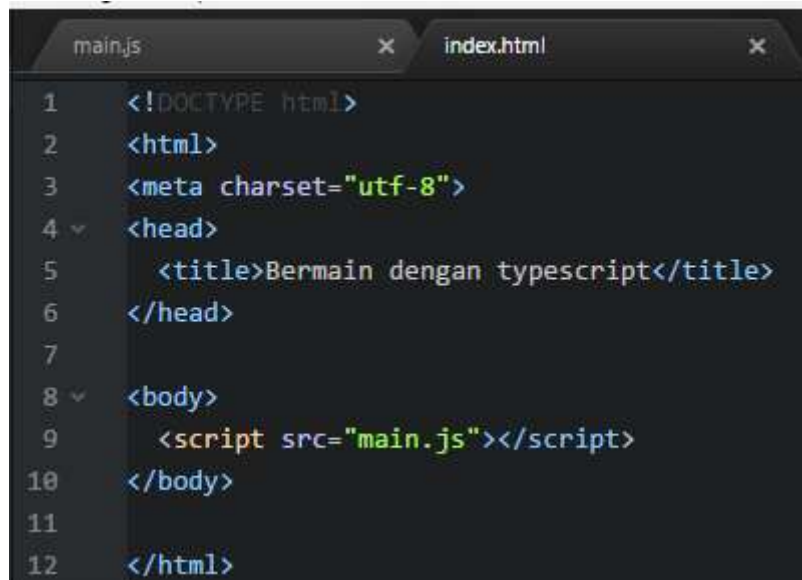
<!DOC TYPE html>
<html>
<meta charset="utf-8">
<head>
  <title>Bermain dengan typescript</title>
</head>
<body>
  <script src="main.js"></script>
</body>
</html>

```

Severity	Provider	Description	Line
Error	htmlhint	Doctype must be declared first.	1:1

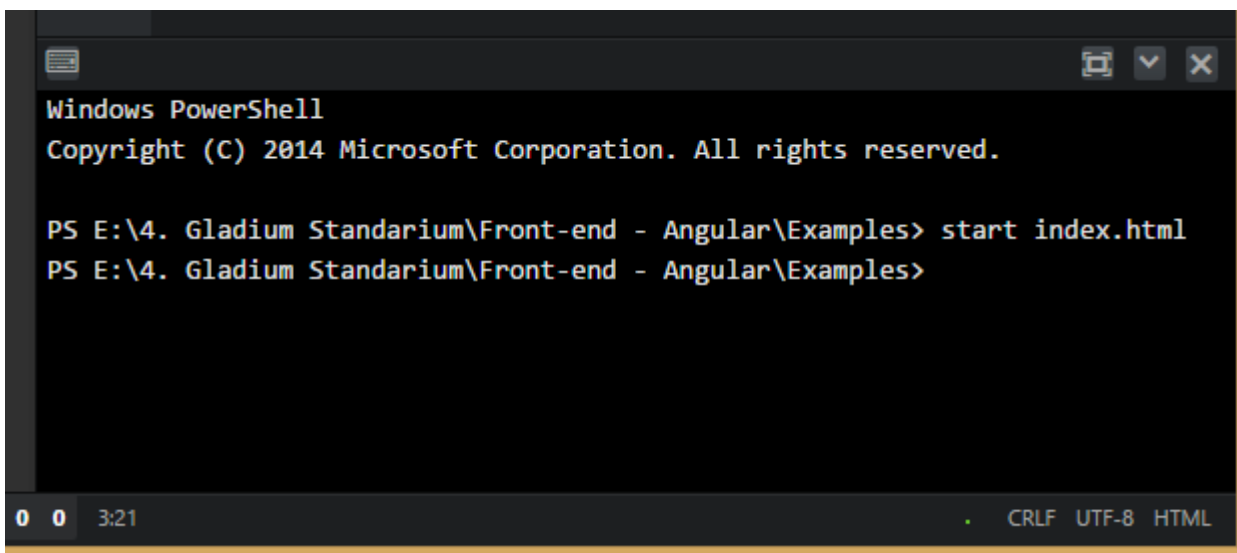
Pada kode editor diatas muncul informasi peringatan berwarna merah, karena penulis mengalami *typo* saat menulis **doctype**. Peringatan ini muncul karena penulis menggunakan *package* **htmlhint**, yang berfungsi sebagai *linter* untuk memeriksa kesalahan dalam kode HTML. Juga terdapat status bar yang memberitahu bahwa kesalahan terdapat pada lokasi baris ke 1.

Jika perhatikan baik baik kode di atas susunanya benar benar tidak rapih, kita bisa menggunakan *package* **atom-beautify**. Untuk memanggilnya pilih menu **Package -> Atom Beautify -> Beautify**, maka susunan kode akan rapih seperti pada gambar di bawah ini :



```
1  <!DOCTYPE html>
2  <html>
3  <meta charset="utf-8">
4  <head>
5    <title>Bermain dengan typescript</title>
6  </head>
7
8  <body>
9    <script src="main.js"></script>
10 </body>
11
12 </html>
```

Sekarang kita akan menggunakan *package* **PlatformIO IDE Terminal** agar bisa menggunakan *terminal powershell* di dalam *atom editor*. Lakukan instalasi, kemudian panggil dengan menekan **alt+shif+t** atau anda bisa memanggilnya melalui menu *package* dalam atom editor.



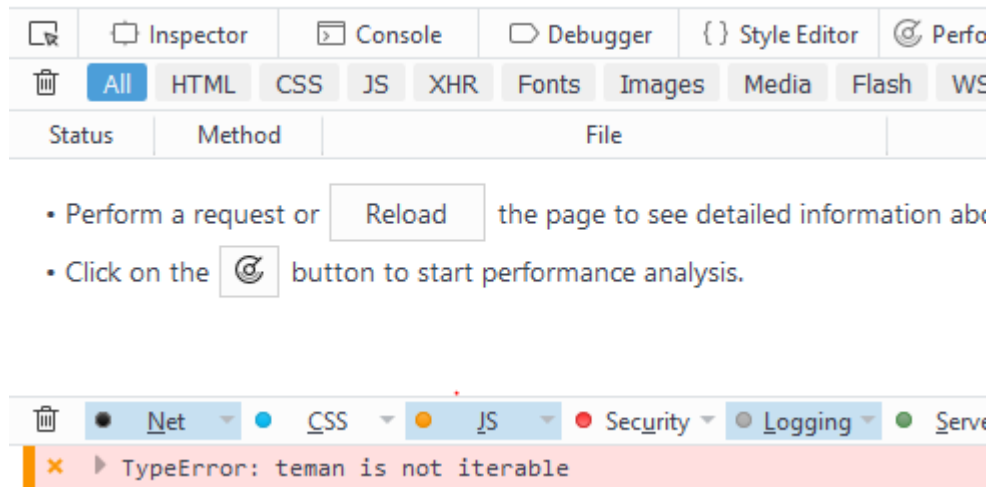
```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS E:\4. Gladium Standarium\Front-end - Angular\Examples> start index.html
PS E:\4. Gladium Standarium\Front-end - Angular\Examples>
```

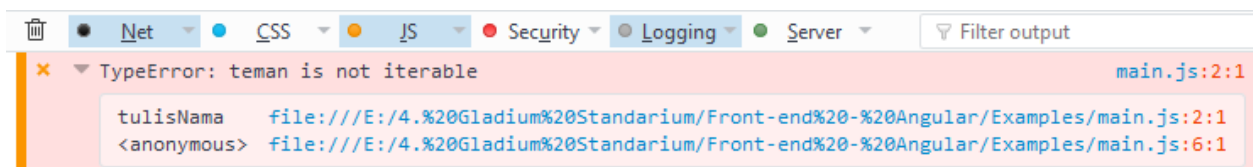
Kemudian dalam terminal eksekusi perintah seperti pada gambar di atas yaitu :

Start index.html

Maka *browser firefox* akan muncul dengan layar *blank* putih, kemudian tekan tombol **CTRL+Shift+Q** untuk melakukan *inspection* pada kode *javascript* yang telah kita buat sebelumnya. Jika, sudah maka akan muncul tab *inspector* seperti pada gambar di bawah ini :



Klik bulatan berwarna *orange* yang bertuliskan JS agar kita bisa mengetahui jika terdapat kesalahan kode *javascript* yang kita buat. Pada gambar di atas muncul pesan *error* yang bermakna bahwa **teman(parameter untuk fungsi tulisNama)** tidak bisa digunakan untuk melakukan iterasi. Kita bisa memperdetail pesan *error* tersebut dengan cara melakukan klik pada tulisan ***TypeError***.



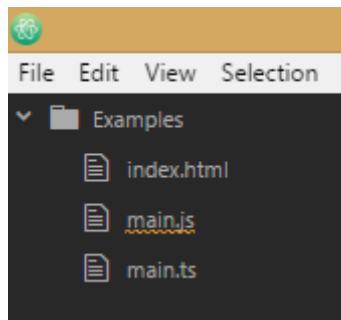
Terdapat informasi baris kode tempat terjadinya kesalahan.

Klik kanan **file main.js** dalam kode editor dan ubah namanya menjadi **main.ts**, kita akan mengkompilasi kode *typescript* kedalam *javascript*. Untuk melakukannya eksekusi perintah di bawah ini :

```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS E:\4. Gladium Standarium\Front-end - Angular\Examples> tsc main.ts
PS E:\4. Gladium Standarium\Front-end - Angular\Examples>
```

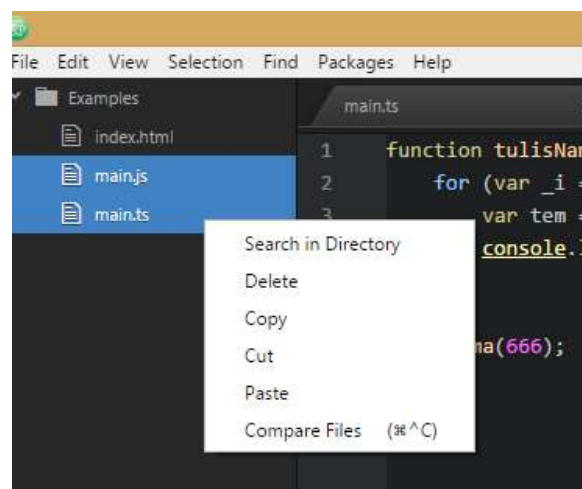
Maka akan muncul *file* baru bernama *main.js* seperti pada gambar di bawah ini :



Di bawah ini adalah kode hasil kompilasinya kedalam plain *javascript* :

```
main.ts x main.js x
1 function tulisNama(teman) {
2   for (var _i = 0, teman_1 = teman; _i < teman_1.length; _i++) {
3     var tem = teman_1[_i];
4     console.log(tem.nama);
5   }
6 }
7 tulisNama(666);
8
```

Jika anda sudah menggunakan *package compare files*, anda bisa melakukan *diff* untuk melihat perbedaan dari *file* main.ts dan main.js. Klik *file* main.ts dan main.js sambil menekan tombol CTRL kemudian klik kanan pilih menu *Compare Files* seperti pada gambar di bawah ini :



Maka akan muncul perbandingan kedua kode tersebut, silahkan luangkan waktu untuk memahami perbedaanya. Proses ini dilakukan menggunakan sebuah algoritma yang dikenal dengan sebutan *diff algorithm*. Teknik-teknik seperti ini dapat membantu anda menghadapi berbagai masalah dalam mengembangkan suatu aplikasi.

```
Packages Help
main.ts x main.js x main.js...main.ts x
1 function tulisNama(teman) {
2     for (var _i = 0, teman_1 = teman; _i < teman_1.length; _i++) {
3         var tem = teman_1[_i];
4         console.log(tem.nama);
5     }
6     for (let tem of teman) {
7         console.log(tem.nama);
8     }
9 }
10 tulisNama(666);
```

Jika anda ingin agar *typescript compiler* otomatis melakukan kompilasi setiap kali kita mengubah kode agar tidak perlu repot-repot memanggil perintah eksekusi, kita bisa mengeksekusi perintah kompilasi dengan *parameter -w*. *Parameter -w* artinya kompiler akan melakukan *watching* untuk memonitor jika ada perubahan kode kompilasi akan dilakukan. Eksekusi perintah di bawah ini :

```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS E:\4. Gladium Standarium\Front-end - Angular\Examples> tsc main.ts -w
8:09:22 PM - Compilation complete. Watching for file changes.

8:09:36 PM - File change detected. Starting incremental compilation...

8:09:36 PM - Compilation complete. Watching for file changes.
```

Jika kita mengubah salah satu kode atau hanya satu karakter saja mengubah *parameter* untuk fungsi **tulisNama** menjadi 66 dari 666 maka akan muncul informasi seperti pada gambar di atas. Kompilasi telah dilakukan. Untuk berhenti dari *watch mode*, tekan CTRL+C kemudian pilih y dan tekan enter.

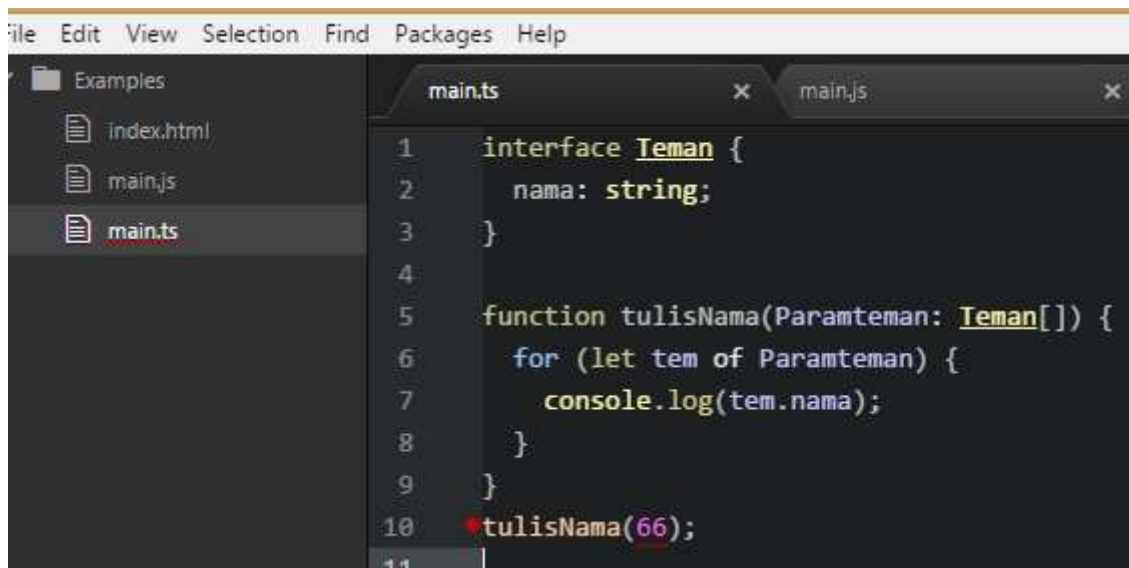
```
^CTerminate batch job (Y/N)? y
PS E:\4. Gladium Standarium\Front-end - Angular\Examples>
```

Jika anda ingin membersihkan layar di dalam terminal eksekusi perintah :

```
Cls
```

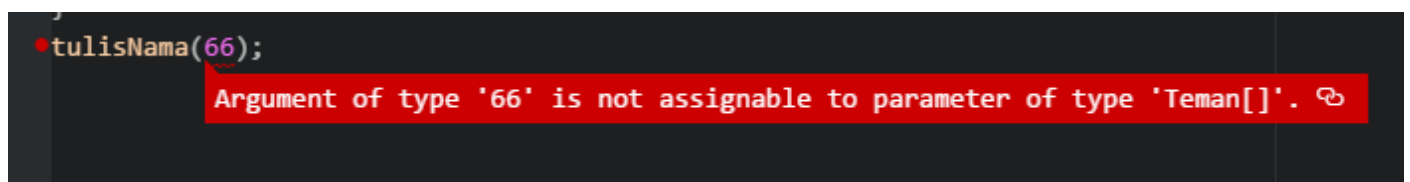
Dari kegiatan di atas kita bisa memahami bahwa *typescript* hanya digunakan saat proses *development*, sebab pada akhirnya *browser* hanya akan mengeksekusi kode *javascript*. Manfaatnya dalam kode yang kita tulis sebelumnya mungkin belum terasa, tapi perlahan anda akan segera mengetahuinya apalagi ketika kode yang anda miliki sudah berada dalam skala besar alis sudah ribuan baris kode.

Sekarang kita akan mencoba membuat sebuah *interface* bernama **Teman** yang akan digunakan sebagai syarat dalam *parameter* fungsi **tulisNama**. Di dalamnya terdapat properties **nama** dengan tipe data *string*. Simbol `[]` setelah *interface* **Teman** digunakan agar selanjutnya harus menerima tipe *array*. Perhatikan kode di bawah ini :



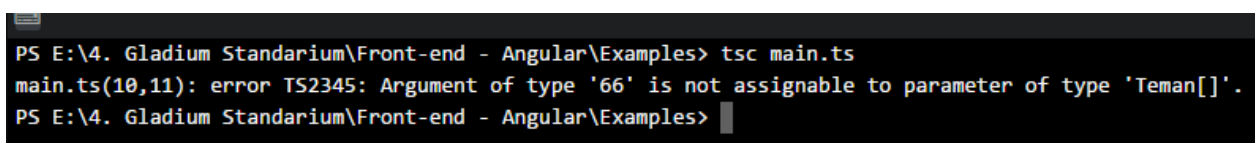
```
1 interface Teman {
2   nama: string;
3 }
4
5 function tulisNama(Paramteman: Teman[]) {
6   for (let tem of Paramteman) {
7     console.log(tem.nama);
8   }
9 }
10 tulisNama(66);
11
```

Jika anda sudah memasang *package* *atom-typescript* dalam *atom editor* maka kode *typescript* akan berwarna (*Syntax Highlight*) seperti pada gambar di atas. Selain itu terdapat *linter* juga yang akan memeriksa kesalahan penulisan kode *typescript*, pada gambar diatas terdapat tanda warna merah pada baris kode ke 10.



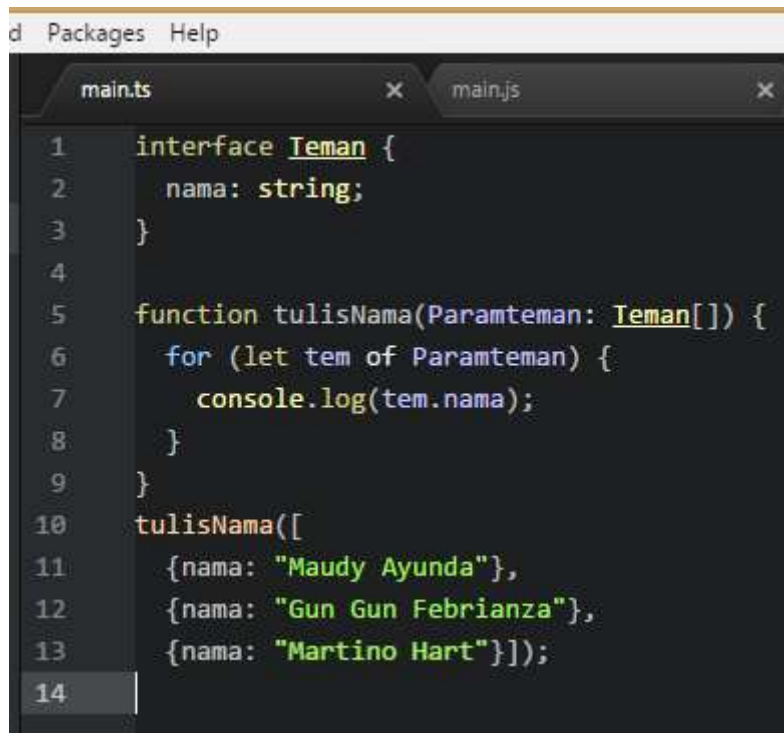
```
•tulisNama(66);
Argument of type '66' is not assignable to parameter of type 'Teman[]'.
```

Jika anda tetap memaksakan untuk melakukan kompilasi maka *error* tetap muncul :



```
PS E:\4. Gladium Standarium\Front-end - Angular\Examples> tsc main.ts
main.ts(10,11): error TS2345: Argument of type '66' is not assignable to parameter of type 'Teman[]'.
PS E:\4. Gladium Standarium\Front-end - Angular\Examples>
```

Kita bisa membetulkan kode diatas dengan memasukan sebuah objek *array* ke dalam *parameter* dari fungsi **tulisNama**. Perhatikan kode di bawah ini :



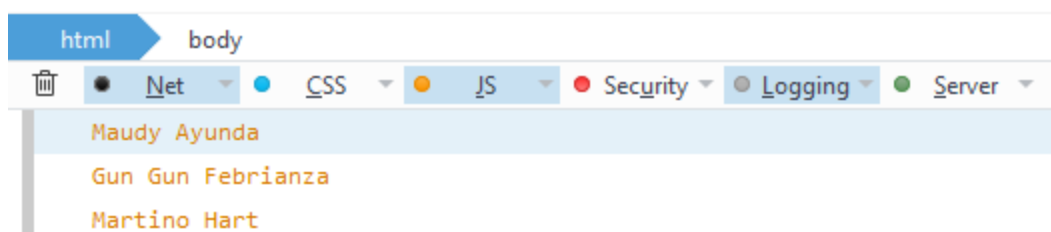
```
1 interface Teman {
2     nama: string;
3 }
4
5 function tulisNama(Paramteman: Teman[]) {
6     for (let tem of Paramteman) {
7         console.log(tem.nama);
8     }
9 }
10 tulisNama([
11     {nama: "Maudy Ayunda"},
12     {nama: "Gun Gun Febrianza"},
13     {nama: "Martino Hart"}]);
14
```

Di bawah ini adalah kode *javascript* hasil kompilasi dari kode *typescript* di atas :



```
1 function tulisNama(Paramteman) {
2     for (var _i = 0, Paramteman_1 = Paramteman; _i < Paramteman_1.length; _i++) {
3         var tem = Paramteman_1[_i];
4         console.log(tem.nama);
5     }
6 }
7 tulisNama([
8     { nama: "Maudy Ayunda" },
9     { nama: "Gun Gun Febrianza" },
10    { nama: "Martino Hart" }
11]);
```

Eksekusi perintah `start index.html`, kemudian inspeksi kita akan melihat 3 nama tertulis seperti pada gambar di bawah ini :

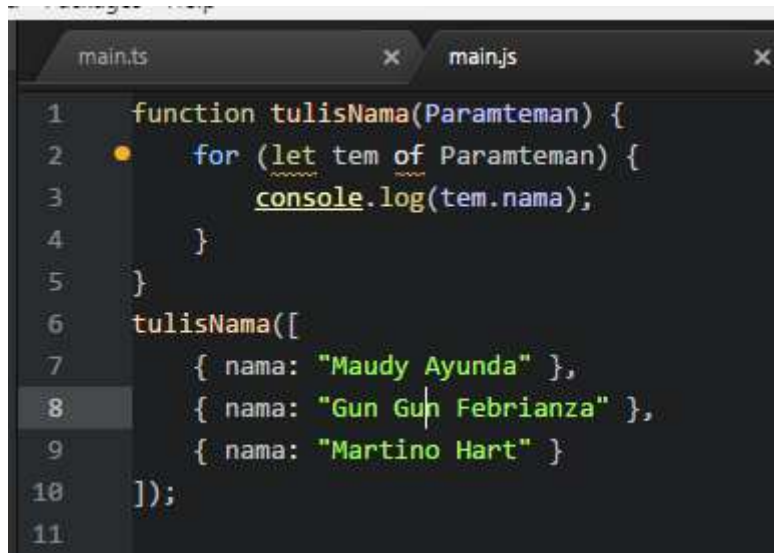


Disinilah kelebihan *typescript*, *interface* tidak ditampilkan di dalam kode *javascript*. Penggunaanya hanya sebagai *development process*. Secara default setelah kita melakukan kompilasi pada *typescript* hasil *javascript* yang diproduksi adalah *javascript* dengan ECMAScript versi 3 atau ES 3. Sementara **for-of-loop** yang digunakan pada *typescript* hanya tersedia pada *javascript* yang sudah mendukung ECMAScript versi

2015. Kita bisa menggunakan *parameter* `-t` saat kompilasi, agar kita bisa memilih hendak menggunakan versi ECMAScript 2015, 2016 atau 2017. Jika kita ingin menggunakan ECMAScript 2015 agar *javascript* yang diproduksi sudah mendukung **for-of-loop** maka eksekusi perintah di bawah ini :

```
PS E:\4. Gladium Standarium\Front-end - Angular\Examples> tsc main.ts -t "ES2015"
PS E:\4. Gladium Standarium\Front-end - Angular\Examples>
```

Di bawah ini kode *javascript* baru yang sudah mendukung ECMAScript 2015 :



```
1 function tulisNama(Paramteman) {
2   for (let tem of Paramteman) {
3     console.log(tem.nama);
4   }
5 }
6 tulisNama([
7   { nama: "Maudy Ayunda" },
8   { nama: "Gun Guh Febrianza" },
9   { nama: "Martino Hart" }
10 ]);
11
```

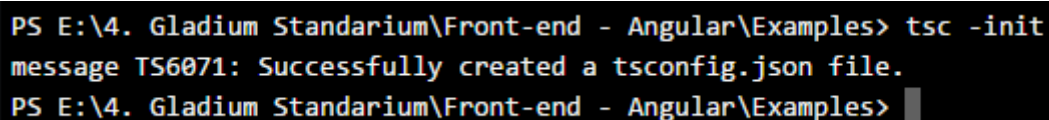
Jika ingin mengetahui opsi *parameter* yang tersedia untuk *typescript compiler* silahkan kunjungi

<https://www.typescriptlang.org/docs/handbook/compiler-options.html>

Jika kita menggunakan pengaturan kompiler di atas untuk aplikasi yang kita buat sendiri tidak menjadi masaaah, namun ada saatnya kita akan berkolaborasi dengan *developer* yang lainnya. Untuk menghindari perbedaan *parameter* dan opsi lainnya saat mengembangkan sebuah aplikasi, konfigurasi awal harus dibuat terlebih dahulu. Kita akan membuat sebuah *file* bernama **tsconfig.json** yang berfungsi untuk menyimpan pengaturan kompilasi pada *typescript compiler*. Eksekusi perintah di bawah ini :

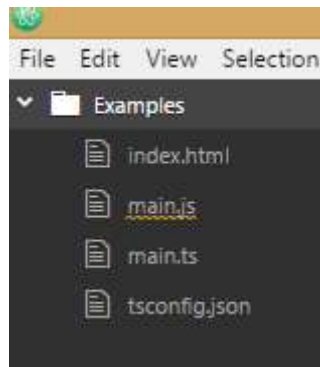
```
tsc -init
```

Maka akan muncul pesan seperti pada gambar di bawah ini :

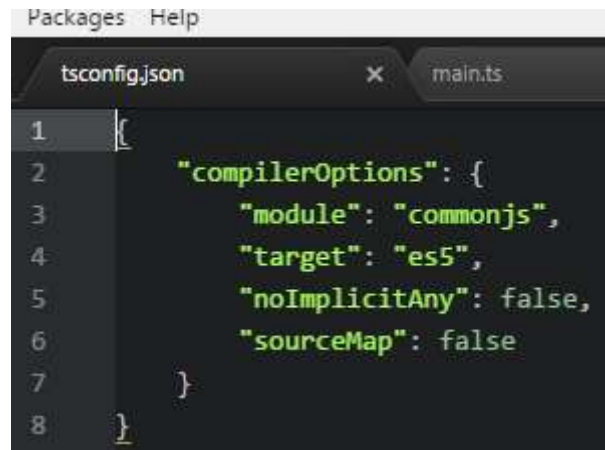


```
PS E:\4. Gladium Standarium\Front-end - Angular\Examples> tsc -init
message TS6071: Successfully created a tsconfig.json file.
PS E:\4. Gladium Standarium\Front-end - Angular\Examples>
```

Dan pada *project explorer* akan muncul *file* `tsconfig.json` seperti pada gambar di bawah ini :



Isi kode dari *file* tsconfigif.json :



Pada gambar di atas terdapat **option target** yang berfungsi untuk mengatur versi *javascript* yang akan diproduksi setelah kompilasi, kemudian terdapat **option module** yang akan kita pelajari di bab selanjutnya, **option noImplicitOnly** bisa diubah menjadi **true** jika anda ingin kompiler memberikan sebuah *error* pada *type* yang tidak diberi spesifikasi (secara otomatis kompiler akan memberinya sebagai *type any*) dan **option sourceMap** bisa diubah menjadi *true* jika anda ingin kompiler juga memproduksi **.js.map-files** yang berguna untuk debugging *typescript*.

Jika *file* tsconfig.json sudah tersedia intruksi kompilasi berubah menjadi lebih singkat yaitu

```
tsc
```

Kompiler akan secara otomatis melakukan kompilasi berdasarkan pengaturan dalam *file* **tsconfig.json**

Notes

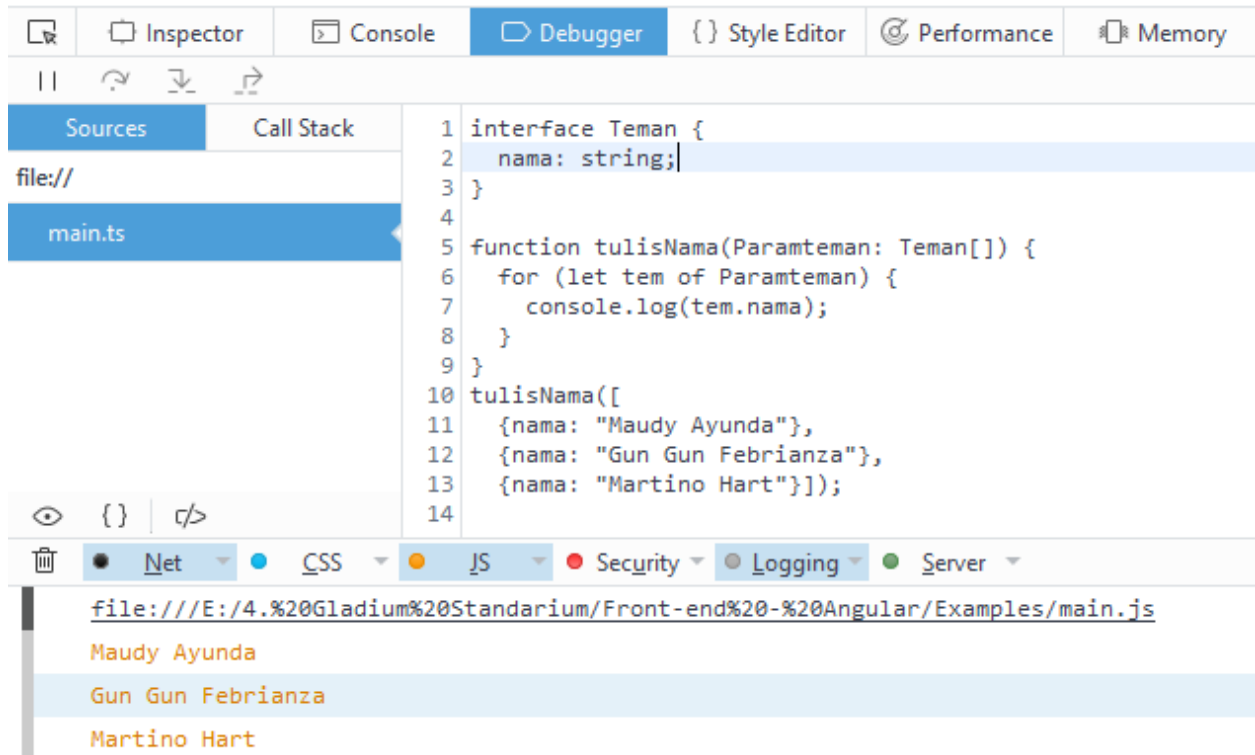
Jika anda ingin kompiler melakukan kompilasi menggunakan pengaturan yang ada di dalam tsconfig.json, cukup dengan perintah tsc saja tanpa harus menyebutkan nama *filenya* atau kompiler tidak akan mampu membaca pengaturan tsconfig.json

4. Debug *Typescript*

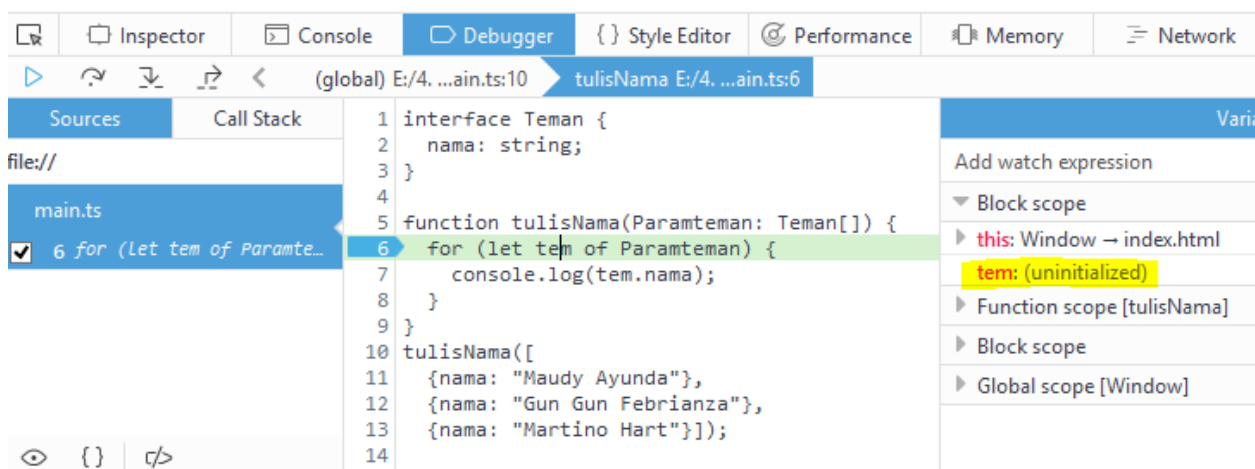
Tehnik Debug adalah tehnik yang sangat fundamental, penulis jadi teringat sebuah *quote* menarik :

Teach a programmer to debug and he can do his work for a lifetime - by Chirag Gude

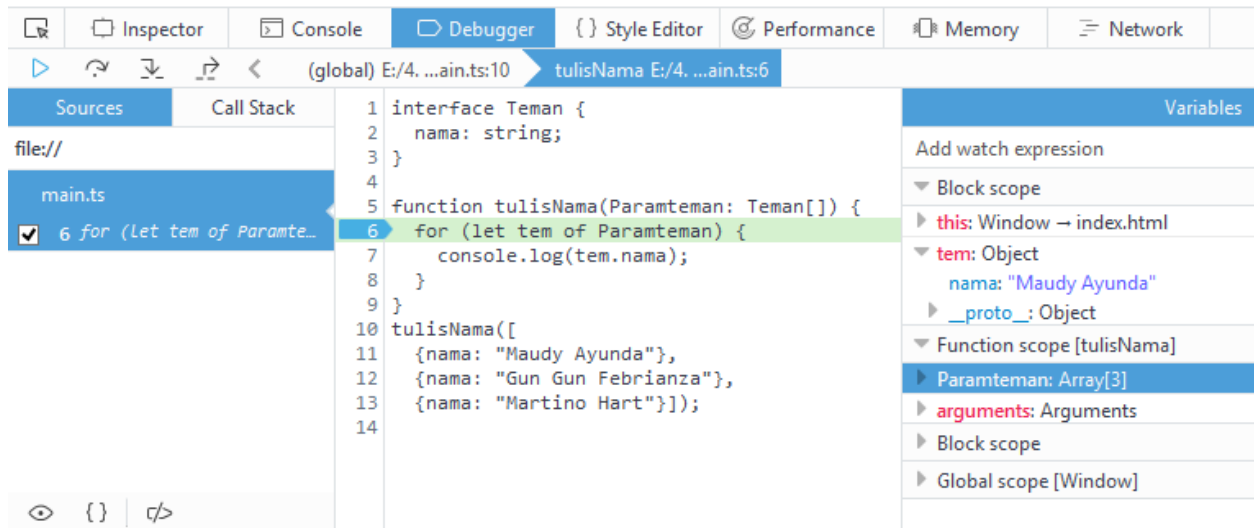
Saat kita membuka halaman **index.html** menggunakan *browser, firefox* akan secara otomatis mendeteksi **source map** yang telah kita buat sebelumnya. Perhatikan gambar di bawah ini :



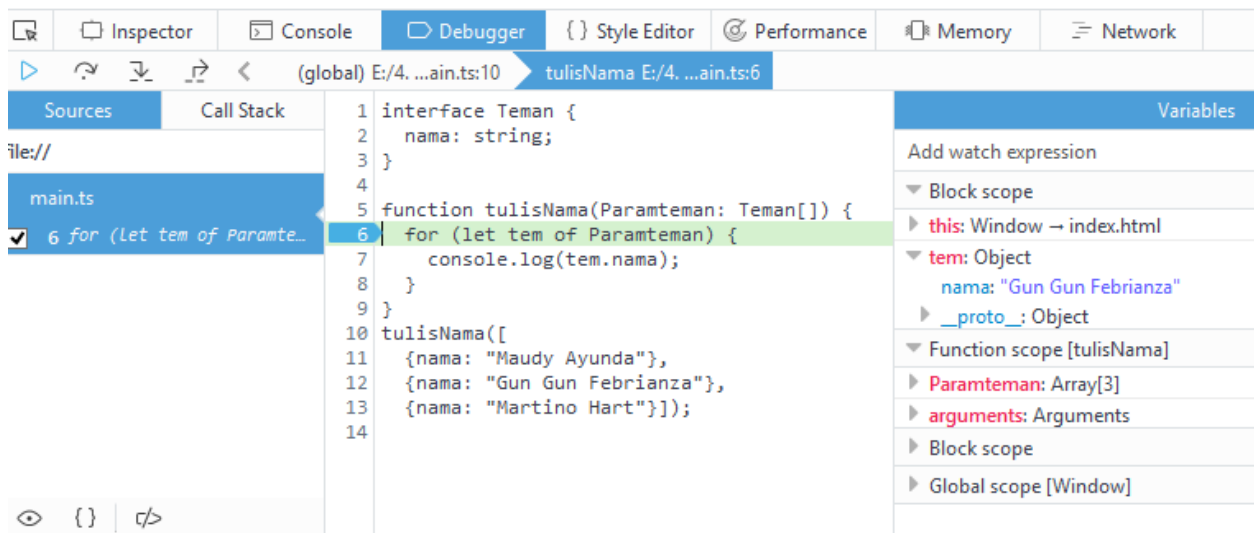
Untuk melakukan debugging pilih baris mana baris kode yang diperiksa, misal pada kasus kali ini kita ingin mengetahui setiap nilai yang tersimpan pada object **tem** di baris kode ke 6. Untuk melakukannya klik angka 6 sampai muncul tanda biru, kemudian **reload** halaman pada *firefox* dengan menekan tombol **F5**. Maka *browser* akan berhenti pada baris kode ke 6, nilai dari **object tem** masih kosong (perhatikan tanda warna kuning pada gambar di bawah) kita bisa melihatnya melalui tab variable disebelah kanan nilainya *uninnitialized*. Ini terjadi karena pertama *browser* harus mengetahui setiap fungsi yang ada di dalam kode *javascript* sebelum akhirnya fungsi tersebut dapat dipanggil.



Kemudian tekan tombol **F8** untuk melanjutkan *debugging* maka selanjutnya **object tem** akan terbaca dalam tab variable disebelah kanan seperti pada gambar di bawah ini :



Jika kita klik akan muncul *properties* yang dimiliki **object tem** yaitu *properties* **nama** dengan nilai sebuah string “Maudy Ayunda”. Lanjutkan lagi dengan menekan tombol **F8** maka akan muncul nilai *object* yang muncul dalam iterasi berikutnya seperti pada gambar di bawah ini :



Tekan **F8** lagi sampai selesai, teknik *debug* ini sangat bermanfaat untuk mengetahui jika terdapat kesalahan dalam kode yang telah kita buat. Kita bisa memeriksa proses yang terjadi sejenak demi sejenak di setiap baris kodenya untuk memastikan *input*, proses dan *output* disetiap *step* sesuai dengan yang kita inginkan. Sehingga jika terdapat kesalahan *logic* pada baris kode tertentu bisa kita lacak dengan mudah.

5. Summary

1. Memahami Berbagai Linter (HTML, CSS, Javascript dan Typescript) dalam Atom Editor
2. Memahami Penggunaan powershell terminal dalam Atom Editor untuk speed development.
3. Memahami Cara untuk melakukan Kompilasi kode typescript dan konfigurasi opsi kompilasinya
4. Memahami Pemanfaatan diff algorithm dalam package compare-files untuk Atom Editor.
5. Memahami Cara untuk melakukan debugging typescript.

3. Types

Kita akan mempelajari **types** dalam *typescript* lebih dalam pada bab ini. Anda mungkin sudah melihatnya sekilas pada bab 2 tentang *typescript cheat*. Pada bab ini kita akan belajar tentang *array*, *tuples*, *enums*, *any*, *union* dan bagaimana cara mengatasi permasalahan nilai *undefined* dan *null*. Terdapat opsi kompilasi “**Strict Null Checking**” saat kompilasi yang akan kita gunakan.

Selain materi di atas kita juga akan belajar tentang **type inference** yang secara otomatis akan memeriksa tipe dari variabel yang digunakan berdasarkan nilai yang diberikan. Lalu mempelajari konsep *type assertion* yang digunakan untuk melakukan *casting* seperti dalam bahasa *c#* atau *java*. Belum pernah mencoba keduanya? Jangan khawatir kita akan mencobanya di halaman selanjutnya.

1. Boolean

Tipe data boolean digunakan untuk menentukan sebuah nilai *logic*, bisa berupa *true* atau *false*.

```
let diaCantik: boolean = true;
```

Tipe data yang akan digunakan diletakkan setelah nama variabel dan diawali dengan simbol **colon** ‘:’

Notes

Keyword *let* digunakan untuk membuat sebuah variabel dalam scope lokal, anda akan mempelajarinya di halaman selanjutnya.

2. Type Inference

Saat kita menetapkan sebuah nilai pada sebuah variabel pada saat deklarasi variabel, *typescript* dapat melakukan proses *infer* variabel. Kita bisa membuat sebuah variabel tanpa harus menulis tipe data yang akan digunakan.

```
let diaCantik = true;
```

Jika kita hendak memisahkan deklarasi dan proses *assignment* atau penetapan nilai kita tidak bisa melakukan proses *infer* perhatikan kode di bawah ini :

```
let diaCantik;  
diaCantik = true; //nilai nya true  
diaCantik = "String"//nilai nya berubah menjadi string (tidak terjadi error)
```

Variabel *diaCantik* akan berubah menjadi tipe data **Any**. Sehingga jika kita ingin memisahkan deklarasi dan proses *assignment* kita harus memberikan tipe data yang akan digunakannya terlebih dahulu.

```
let diaCantik: boolean;  
diaCantik = true; //nilai nya true  
diaCantik = "String"//nilai nya berubah menjadi string (Semantic Error)
```

3. Number

Sebuah angka di dalam *typescript* merupakan tipe data **number**.

```
let tinggi: number = 900;
```

Javascript adalah bahasa yang *loosely typed language* yang tidak mengenal *type* data seperti *integer*, *short*, *long* atau *float* dan seterusnya. Begitu juga didalam *typescript* seluruh *number* representasinya di dalam sistem diubah menjadi 64-bit *floating point*. Namun begitu kita tetap bisa menyimpan sebuah nilai angka desimal dalam variabel *typescript*

```
let tinggi: number = 900.888;
```

Typescript juga mendukung *number-literal* dalam bentuk *hex*, *binary* dan *octal*. Perhatikan kode dibawah ini dan *prefix* yang digunakan :

```
let dec: number = 27;  
let hex: number = 0x001b;  
let binary: number = 0b11011;  
let octal: number = 0o0033;
```

4. String

Kita bisa menyimpan nilai *string* ke dalam sebuah variabel menggunakan *double* atau *single quote*. Perhatikan kode di bawah ini :

```
let nama: string = "Gun Gun Febrianza";  
nama = 'Gun Gun Febrianza';
```

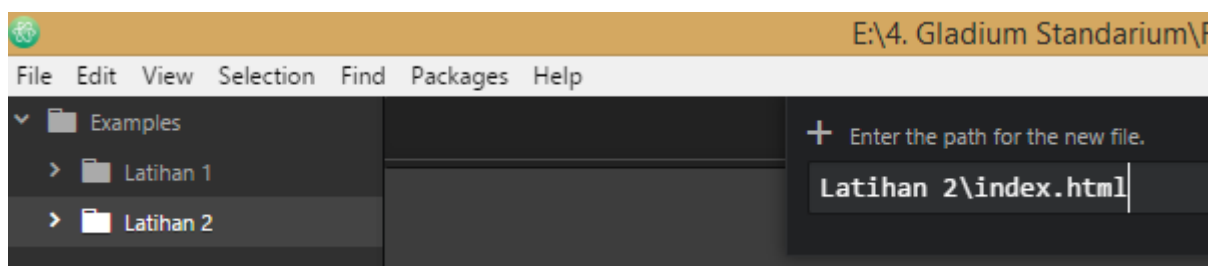
Typescript juga mendukung *template string* menggunakan *syntax* `${expression}`. Perhatikan kode di bawah ini :

```
va nama: string = "Gun Gun Febrianza";  
let nama: string = `Hallo ${nama}, kami doakan semoga anda selalu sehat.`;
```

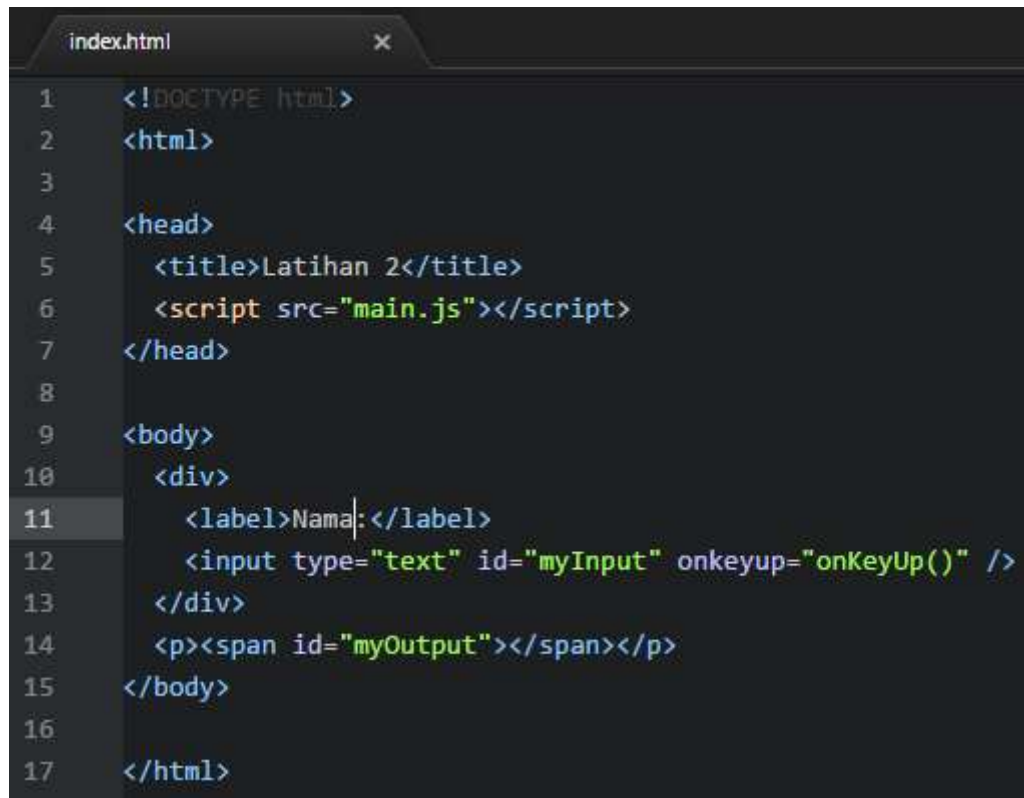
Jika kode *typescript* di atas dikompilasi ke dalam *javascript* maka outputnya adalah sebagai berikut :

```
var nama = "Gun Gun Febrianza";  
var nama = "Hallo" + nama + ", kami doakan semoga anda selalu sehat.";
```

Silahkan rapihkan kode sebelumnya yang telah kita buat masukan ke dalam *folder* Latihan 1 dan buatlah *folder* baru dengan nama Latihan 2, kemudian buatlah *file* html di dalam *folder* 2 dengan nama **index.html** seperti pada gambar di bawah ini :

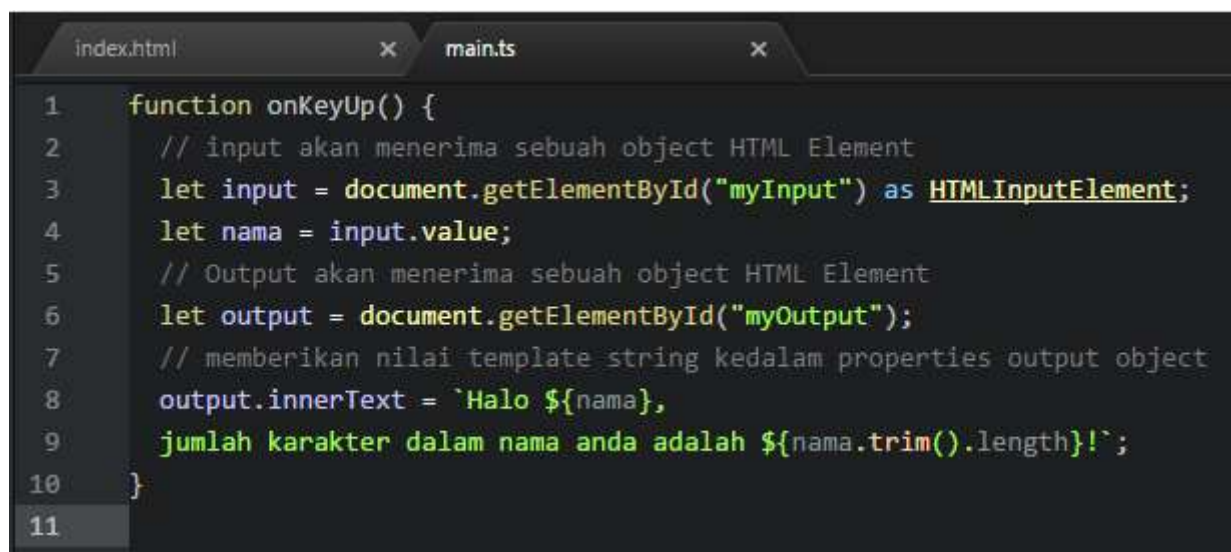


Tulis kode di bawah html seperti pada gambar di bawah ini atau anda dapat membuka contoh kodenya pada sumber kode yang telah penulis berikan di repository github.



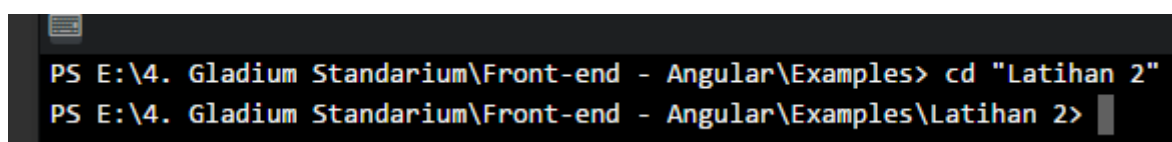
```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <title>Latihan 2</title>
6   <script src="main.js"></script>
7 </head>
8
9 <body>
10  <div>
11    <label>Nama: </label>
12    <input type="text" id="myInput" onkeyup="onKeyUp()" />
13  </div>
14  <p><span id="myOutput"></span></p>
15 </body>
16
17 </html>
```

Pada kode di atas, *javascript* **main.js** sengaja disimpan di dalam **tag header** agar tidak dieksekusi saat *browser* pertama kali memuat halaman. Kemudian pada baris kode ke 12 terdapat **kontrol input** yang akan merespon jika terjadi **event onkeyup**. Sebuah *event* yang akan memanggil *function* `onKeyUp()` dari kode *javascript* yang akan kita buat. Selanjutnya buatlah sebuah *file typescript* dengan nama `main.js` seperti pada gambar di bawah ini :



```
1 function onKeyUp() {
2   // input akan menerima sebuah object HTML Element
3   let input = document.getElementById("myInput") as HTMLInputElement;
4   let nama = input.value;
5   // Output akan menerima sebuah object HTML Element
6   let output = document.getElementById("myOutput");
7   // memberikan nilai template string kedalam properties output object
8   output.innerText = `Halo ${nama},
9   jumlah karakter dalam nama anda adalah ${nama.trim().length}!`;
10 }
11
```

Kemudian sebelum kompilasi jangan lupa untuk memindahkan posisi *current directory* :

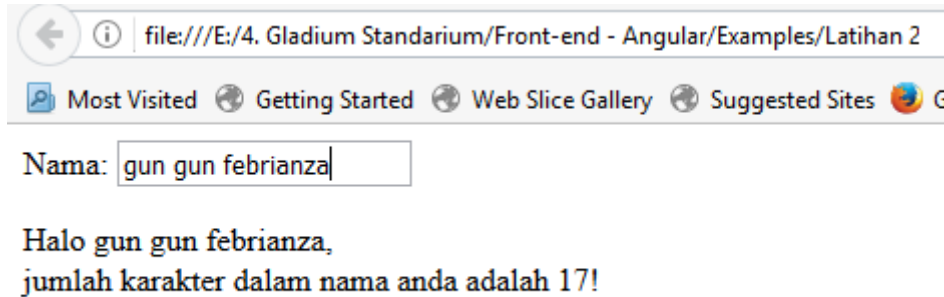


```
PS E:\4. Gladium Standarium\Front-end - Angular\Examples> cd "Latihan 2"
PS E:\4. Gladium Standarium\Front-end - Angular\Examples\Latihan 2>
```


Lakukan kompilasi dan panggil *file* index.html

```
PS E:\4. Gladium Standarium\Front-end - Angular\Examples\Latihan 2> tsc main.ts -t "ES2015"
PS E:\4. Gladium Standarium\Front-end - Angular\Examples\Latihan 2> start index.html
PS E:\4. Gladium Standarium\Front-end - Angular\Examples\Latihan 2> |
```

Jika kita membuka *browser* maka akan muncul halaman seperti pada gambar di bawah ini, masukan nama anda maka kode *javascript* akan dieksekusi secara otomatis oleh *browser*.



Di bawah ini adalah hasil *javascript* yang diproduksi setelah kompilasi, pada ES2015 konsep *template string* sudah dikenali :

```
index.html x main.ts x main.js x
1  function onKeyUp() {
2      // input akan menerima sebuah object HTML Element
3      let input = document.getElementById("myInput");
4      let nama = input.value;
5      // Output akan menerima sebuah object HTML Element
6      let output = document.getElementById("myOutput");
7      // memberikan nilai template string kedalam properties output ob
8      output.innerHTML = `Halo ${nama},
9      jumlah karakter dalam nama anda adalah ${nama.trim().length}!`;
10 }
```

Jika target kompilasi secara *default* menggunakan ES3 maka hasil produksi kode *javascript* yang dihasilkannya adalah sebagai berikut sebab konsep *template string* belum dikenali :

```
1  function onKeyUp() {
2      // input akan menerima sebuah object HTML Element
3      var input = document.getElementById("myInput");
4      var nama = input.value;
5      // Output akan menerima sebuah object HTML Element
6      var output = document.getElementById("myOutput");
7      // memberikan nilai template string kedalam properties output object
8      output.innerHTML = "Halo " + nama + ",\n jumlah karakter dalam nama anda adalah " + nama.trim().length + "!";
9  }
```

5. Array

Di dalam *Typescript* kita bisa menggunakan *array* dengan cara menambahkan sepasang *square bracket*.

```
let arrayNama: string[] = ["Gun Gun Febrianza", "Maudy", "Martino"];
```

Selain cara deklarasi *array* di atas kita juga bisa menggunakan deklarasi *array* menggunakan *generic array* class. Perhatikan kode di bawah ini :

```
let arrayNama: <string> = ["Gun Gun Febrianza", "Maudy", "Martino"];
```

Anda bisa memilihnya salah satu karena hasilnya sama saja.

Iterasi for..of dan for..in

Untuk melakukan iterasi nilai dalam sebuah *array*, kita bisa menggunakan for-of-loop. Perhatikan kode *typescript* di bawah ini :

```
let arrayNama: string[] = ["Gun Gun Febrianza", "Maudy", "Martino"];
for (let nama of arrayNama) {
  console.log(nama);
}
```

Di bawah ini adalah iterasi menggunakan for..in, jika kita menggunakan iterasi ini variabel index tidak akan mendapatkan nilai dari setiap *array* disetiap iterasi. Melainkan variabel index akan mendapatkan nilai index untuk setiap *array*.

```
Let arrayNama: string[] = ["Gun Gun Febrianza", "Maudy", "Martino"];
for (let index in arrayNama) {
  console.log(`${index} - ${arrayNama[index]}`);
}
```

Iterasi For..In untuk sebuah Object

Di bawah ini adalah implementasi iterasi for..in pada sebuah *object*, di bawah ini kita membuat interface *Teman* dengan dua properties di dalamnya. *Typescript* adalah bahasa yang **case-sensitive**, yang membedakan sebuah *identifier* maksudnya adalah teman != Teman (teman dengan Teman adalah beda).

```
interface Teman {
  nama:string;
  namaKepanjangan:string;
}

let teman: Teman = {nama:"Maudy", namaKepanjangan:"Ayunda"};

for (let propName in teman) {
  console.log(`${propNama}: ${teman[propNama]}`);
}
```

Jika kode di atas dieksekusi maka akan menghasilkan *output* seperti di bawah ini :

```
nama:Maudy
namaKepanjangan:Ayunda
```

Ada dua cara untuk mengakses sebuah *object* yaitu menggunakan *dot operator* atau *square bracket* seperti pada contoh kode di bawah ini :

```
let nama = teman.nama; // akses nilai properties objek, cara 1
nama = teman["nama"]; // akses nilai objek berdasarkan index, cara 2
```

6. Tuples

Sebuah *tuples* bekerja seperti *array* hanya saja bisa memiliki *type* yang berbeda. Di bawah ini terdapat sebuah *tuple* yang memiliki variabel *string* dan *boolean* :

```
let contohTuple: [string, boolean] = ["Gun Gun Febrianza", true];
```

Sebuah *type* *string* berada pada index ke 0 dan *boolean* ada pada index ke 1

7. Enum

Dengan *enum* kita bisa membuat naming yang mudah difahami untuk sebuah angka, *index* pada *enum* juga dimulai dari angka 0 dan seterusnya. Di bawah ini adalah contoh kode *enum* :

```
enum Arah { kiri, atas, kanan, bawah }
```

Pada *enum* di atas, *kiri* memiliki index 0, *atas* memiliki index 1, *kanan* memiliki index 2 dan *bawah* memiliki index 3. Kita juga bisa mengubah index dari angka awal dalam sebuah *enum*.

```
enum Arah { kiri = 1, atas, kanan, bawah }
```

Selain itu kita juga bisa menjelaskan seluruh index dalam *enum* secara *explicit* :

```
enum Arah { kiri = 1, atas = 2, kanan = 3, bawah = 4 }
```

Untuk mengakses nilai yang ada di dalam sebuah *enum* kita bisa memanggilnya dengan cara :

```
let nilai: number = Arah.kiri;
```

Atau menggunakan posisi index yang dimilikinya

```
let nilai: number = Arah[1];
```

8. Any Type

Terkadang kita membutuhkan sifat *dynamic typing* bukan hanya *static typing* seperti yang telah kita lakukan setiap kali hendak membuat sebuah variabel. Dalam *typescript* kita bisa menggunakan *any-Type*.

```
let diaCantik: any = true; // inisialisasi dengan boolean
console.log(typeof diaCantik);
diaCantik = "benar"; // nilai variabel diubah menjadi string
console.log(typeof diaCantik);
diaCantik.fungsiX();
```

Pada kode di atas *typeof* adalah *keyword* yang digunakan untuk mengetahui tipe data suatu variabel. *Any-Type* yang digunakan akan membuat kompiler untuk tidak memberikan *semantic error* saat memeriksa *type* yang hendak digunakan. *Any-type* digunakan saat kita menghadapi kasus dimana kita membutuhkan *function-parameter* yang bersifat implisit atau tidak diketahui. Di bawah ini adalah contoh *function* dengan *parameter* implisit. Dikatakan implisit karena kita tidak akan pernah tau apa saja nilai yang akan dimasukan kedalam *parameter*, karena *parameter* tidak memiliki *type* :

```
function tulisNama(teman) {  
    console.log(teman.namaKepanjangan);  
}
```

Buatlah sebuah *folder* bernama Latihan 3 dan buat *file* main.ts dengan kode seperti di atas. Menulis kode secara eksplisit memiliki banyak sekali keuntungan termasuk ketika kita menggunakan *typescript*. Sebagai contoh jika kita ingin migrasi sebuah code base ukuran besar yang ditulis dengan *javascript* di dalamnya masih bisa terdapat sebuah *function* yang *parameternya* tidak memiliki *type* seperti kode di atas. Untuk menemukan setiap fungsi yang memiliki *parameter* implisit kita bisa mengubah **compiler option** **noImplicitAny** menjadi true. Eksekusi perintah **tsc -init** agar memproduksi *file* tsconfig.json kemudian peraturan **noImplicitAny** diubah menjadi true.

```
{  
  "compilerOptions": {  
    "module": "commonjs",  
    "target": "es2015",  
    "noImplicitAny": true,  
    "sourceMap": false  
  }  
}
```

Kompilasi lagi dengan perintah **tsc** saja, maka akan muncul pesan *error* seperti pada gambar di bawah ini :

```
PS E:\4. Gladium Standarium\Front-end - Angular\Examples\Latihan 3> tsc  
main.ts(1,20): error TS7006: Parameter 'teman' implicitly has an 'any' type.  
PS E:\4. Gladium Standarium\Front-end - Angular\Examples\Latihan 3>
```

Untuk membetulkannya kita harus menambahkan *type any* pada kode sebelumnya seperti di bawah ini :

```
function tulisNama(teman:any) {  
    console.log(teman.namaKepanjangan);  
}
```

9. Type Assertion

Terkadang kita lebih mengetahui kode yang kita tulis lebih dari kompiler. Diasumsikan anda tahu variabel **nama** memiliki nilai sebuah *string*, namun satu hal yang perlu dicatat bahwa *type* yang digunakan adalah *any*.

```
Let nama:any = "Gun Gun Febrianza "
```

Kemudian jika kita ingin mengakses salah satu *property* dari *string* tersebut misal **length property** untuk mengetahui jumlah karakter yang dimilikinya, maka tidak akan menghasilkan informasi apa apa karena variabel nama memiliki *type any*.

```
Let jumlahKarakter:number = nama.length;
```

Untuk mengatasi masalah ini agar kompiler *typescript* mengetahuinya kita bisa menggunakan **type assertion** seperti pada kode di bawah ini :

```
Let jumlahKarakter:number = (nama as string).length;
```

Selain menggunakan *keyword as* kita juga bisa menggunakan cara seperti pada kode di bawah ini :

```
Let jumlahKarakter:number = (<string>nama).length;
```

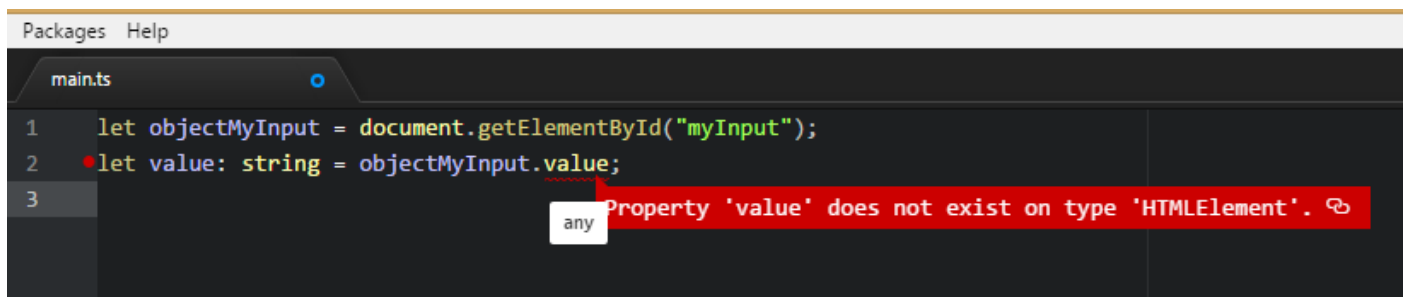
Kegiatan *type assertion* akan sering digunakan ketika kita menggunakan fungsi yang memiliki nilai *return* sesuai *base type* yang dimilikinya. Sebagai contoh jika kita menggunakan *method getElementById* maka nilai *return* yang dihasilkan adalah sebuah *HTML Element*. Namun begitu, *typescript compiler* tidak akan membaca kode HTML dalam *file* HTML yang kita miliki. Kita mungkin mengetahui mana *HTML element* yang ingin kita gunakan namun kompiler tidak. Sebagai contoh terdapat sebuah *HTML Element* dengan *attribut id my input* :

```
<input type="text" id="myInput"/>
```

Untuk mengakses *property* yang dimiliki *element* tersebut menggunakan *typescript*, kita akan menulis kode di bawah ini :

```
let objectMyInput = document.getElementById("myInput");  
let value: string = objectMyInput.value;
```

Maka akan muncul pesan *error* seperti pada gambar di bawah ini :



Buatlah *folder* bernama Latihan 4 dan buat sebuah *file* bernama main.ts dengan kode seperti di atas, atau melihat dari *source code* yang sudah penulis sediakan. Untuk membetulkan permasalahan di atas kita harus menggunakan *type assertion* seperti pada kode di bawah ini :

```
let value: string = (objectMyInput as HTMLInputElement).value;
```

Jika ingin sekupnya agar bisa digunakan dimana saja anda bisa langsung memberinya *type assertion* seperti pada gambar di bawah ini :

```
let objectMyInput = document.getElementById("myInput") as HTMLInputElement;
```

10. Union

Ada saatnya kita menginginkan variabel yang bisa disimpan dengan berbagai *type*, selain menggunakan *type any* kita bisa menggunakan *union type*. Sebuah *union type* adalah kombinasi dari berbagai *type*. Kode di bawah ini adalah variabel dengan kemampuan untuk bisa menampung dua *type* data sekaligus yaitu sebagai *boolean* atau *number*.

```
let var : boolean|number = true;  
isVisible = 1; // OK  
isVisible = "yes"; // akan menghasilkan error
```

Perhatikan kode di bawah ini, terdapat *function padLeft* yang memiliki *parameter padding* dengan *type any*. Pada kode di bawah ini *parameter padding* hanya bisa menerima sebuah *number* atau *string*. Jika anda memberikan nilai yang lain *typescript compiler* akan memaklumiya tanpa memberi pesan *error*.

```
function padLeft(value: string, padding: any) {  
  if (typeof padding === "number") {  
    return Array(padding + 1).join(" ") + value;  
  }  
  if (typeof padding === "string") {  
    return padding + value;  
  }  
  throw new Error(`membutuhkan string atau number`);  
}
```

Jika *function* diatas dieksekusi dengan *parameter* di bawah ini :

```
let indentedString = padLeft("Hello world", 4);
```

Maka hasilnya adalah :

```
"  Hello world"
```

Kemudian jika *function* diatas dieksekusi dengan *parameter* di bawah ini :

```
let indentedString = padLeft("Hello world", true);
```

Maka saat kompilasi *error* tidak akan muncul, namun *error* akan muncul saat proses *runtime*. Untuk mengatasi hal ini kita bisa menggunakan *union type*. Buatlah sebuah *folder* dengan nama Latihan 5 dan masukan kode di bawah ini kedalam *file main.ts* seperti pada gambar di bawah ini :



```
main.ts x  
1 function padLeft(value: string, padding: string | number) {  
2   if (typeof padding === "number") {  
3     return Array(padding + 1).join(" ") + value;  
4   }  
5   if (typeof padding === "string") {  
6     return padding + value;  
7   }  
8   throw new Error("membutuhkan string atau number");  
9 }  
10 let indentedString = padLeft("Hello world", true);  
11
```

Dengan *union type* kita bisa mendeteksi kesalahan pada saat kompilasi, pada kode editor di atas *linter* sudah mendeteksi akan terjadi kesalahan jika kita melakukan kompilasi. Jika kita tetap memaksakanya untuk dikompilasi maka akan muncul pesan *error* seperti pada gambar di bawah ini :


```
PS E:\4. Gladium Standarium\Front-end - Angular\Examples\Latihan 5> tsc main.ts
main.ts(10,45): error TS2345: Argument of type 'true' is not assignable to parameter of type 'string | number'.
PS E:\4. Gladium Standarium\Front-end - Angular\Examples\Latihan 5> █
```

11. Returning Void

Sebuah *type void* digunakan sebagai *return type* dari sebuah *function*. Dengan *void* sebuah *function* tetap menghasilkan *return* yang tidak memiliki nilai. Perhatikan kode di bawah ini :

```
function logIt(input: string): void {
    console.log(input);
}
```

Fungsi di atas hanya akan menampilkan *log* pada *browser* berdasarkan *input string* yang diberikan.

12. Never

Untuk *type never*, tipe ini digunakan pada nilai yang tidak akan pernah muncul. Sebagai contoh terdapat sebuah *function* yang selalu menghasilkan *error* dan memiliki *return type never*.

```
function doSomething(): never {
    throw new Error("Not implemented");
}
```

Never juga bisa digunakan pada sebuah variabel. Perhatikan kode di bawah ini :

```
function book(text: string, page: string | number): string {
    if (typeof page === "number") {
        return text + Array(appendix).join(" ");
    }

    if (typeof page === "string") {
        return text + page;
    }

    // <- Deklarasi variabel page disini
}
```

Jika kita mencoba membuat sebuah variabel setelah kedua *if statement* untuk menampung nilai *page* yang memiliki *type union* maka variabel tersebut akan memiliki *type never* karena kode tersebut tidak akan pernah dieksekusi. *Typescript compiler* akan mengetahuinya.

13. Undefined, Null & Strict Checking

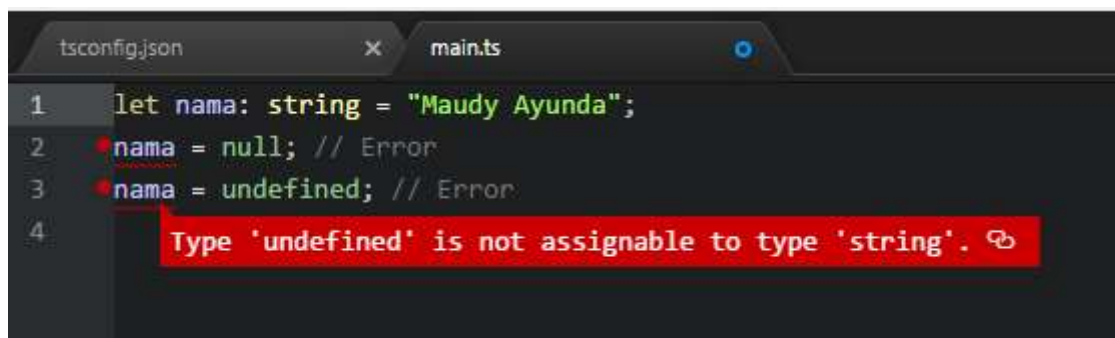
Nilai *undefined* dan *null* adalah nilai dalam *javascript* yang bisa mendatangkan banyak sekali *error*, pada *typescript* nilai *undefined* dan *null* masing masingnya memiliki *type*. Terdapat *type undefined* dan juga *type null*. Secara default *undefined* dan *null* adalah *subtype* dari seluruh *type* yang ada di dalam *typescript*, maksudnya adalah kita tetap bisa mengisi sebuah nilai *null* atau *undefined* kedalam sebuah *type* (misal *string*) seperti pada contoh kode di bawah ini :

```
let nama: string = "Gun Gun Febrianza";
nama = null; // OK
nama = undefined; // OK
```

Seperti yang sudah disebutkan sebelumnya bahwa *undefined* dan *null* bisa memancing banyak *error* salah satunya adalah *runtime-error*. Seandainya seseorang mengakses variabel yang bernilai *null* atau *undefined* saat *runtime* pasti akan terjadi *runtime error*. Untuk menghindari *error*, *typescript* memiliki opsi *strictNullChecks* kita bisa menggunakannya di dalam file *tsconfig.json-file* seperti pada kode di bawah ini :

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
    "noImplicitAny": false,
    "sourceMap": false,
    "strictNullChecks": true
  }
}
```

Dengan opsi *strictNullCheck* maka *types undefined* dan *null* bukan lagi menjadi *subtype* seluruh *type*, ini artinya kita tidak bisa menetapkan mereka kedalam sebuah *type* lagi. Buatlah *folder* dengan nama Latihan 6 dan buat *file* bernama **main.ts** dengan isi kode seperti gambar di bawah ini :



Saat anda belum membuat *file tsconfig.json* yang tidak memiliki opsi *strictNullCheck*, *linter* tidak akan mendeteksi potensi *error* saat kompilasi. Namun ketika anda membuatnya dengan mengeksekusi perintah **tsc -init**, kemudian menambahkan opsi *strictNullCheck* dengan nilai *true* di dalamnya maka akan muncul pesan *error* seperti pada gambar di atas. Bahwa *undefined* dan *null* tidak bisa digunakan pada sebuah *string*.

Namun jika anda ingin nilai *null* atau *undefined* tetap bisa digunakan meskipun konfigurasi *strictNullChecks* sedang aktif, maka kita bisa menggunakan *union type*. Seperti pada kode di bawah ini :

```
let nama: string | null = "Maudy Ayunda";
nama = null; // Ok
nama = undefined; // Error
```

14. Summary

1. Kita telah mempelajari *basic types* yang ada di dalam *typescript* seperti *number*, *boolean* dan *string*.
2. Kita telah mempelajari bagaimana melakukan *infer types* menggunakan *typescript compiler*.

3. Mempelajari sebuah *array*, melakukan iterasi sebuah *array* dengan ***for-of-loop*** dalam *typescript*.
4. Mempelajari sebuah *array* yang bisa memiliki berbagai *type* dalam *typescript* yaitu *tuple*.
5. Mempelajari sebuah *enum* untuk melakukan *enumeration* dalam *typescript*.
6. Mempelajari *dynamic code* yang tidak diketahui saat *compile-time* menggunakan *any-type*.
7. Mempelajari *type assertion* dalam *typescript*.
8. Mempelajari *union type* dalam *typescript*.
9. Mempelajari konfigurasi baru untuk kompilasi *typescript* dengan *parameter* `strictNullChecks-option`.

Lampiran

Reserved Keyword

break	as	any	switch
case	if	throw	else
var	number	string	get
module	type	instanceof	typeof
public	private	enum	export
finally	for	while	void
null	super	this	new
in	return	true	false
any	extends	static	let
package	implements	interface	function
new	try	yield	const
continue	do	catch	