**Nama : Heria Yunita Putri  
NIM : L200140063  
Kelas : A**

1. **Daftar Tipe Data**

Tabel berikut menunjukkan tipe data, termasuk ukuran dan range.

|  |  |  |
| --- | --- | --- |
| **Tipe Data** | **Ukuran** | **Range** |
| **Byte** | 1 byte | 0 sampai 255 |
| **Boolean** | 2 bytes | **True** atau **False** |
| **Integer** | 2 bytes | -32,768 sampai 32,767 |
| **Long** (long integer) | 4 bytes | -2,147,483,648 sampai 2,147,483,647 |
| **Single** (single-precision floating-point) | 4 bytes | -3.402823E38 sampai -1.401298E-45 untuk nilai negarif; 1.401298E-45 sampai 3.402823E38 untuk nilai positif |
| **Double** (double-precision floating-point) | 8 bytes | -1.79769313486231E308 sampai -4.94065645841247E-324 untuk nilai negarif; 4.94065645841247E-324 sampai 1.79769313486232E308 untuk nilai positif |
| **Currency** (scaled integer) | 8 bytes | -922,337,203,685,477.5808 sampai 922,337,203,685,477.5807 |
| **Decimal** | 14 bytes | +/-79,228,162,514,264,337,593,543,950,335 tanpa nilai desimal; +/-7.9228162514264337593543950335 dengan 28 tempat disebelah kanan desimal; nilai terkecil adalah +/-0.0000000000000000000000000001 |
| **Date** | 8 bytes | 1 Januari 100 sampai 31 Desember 9999 |
| **Object** | 4 bytes | Referensi **Object** |
| **String** (variable-length) | 10 bytes + panjang string | 0 sampai dengan 2 milyar |
| **String** (fixed-length) | Panjang String | 1 sampai dengan 65,400 |
| **Variant** (with numbers) | 16 bytes | Nilai numerik sampai range dari **Double** |
| **Variant** (with characters) | 22 bytes + panjang string | Range sama dengan variable-length **String** |
| User-defined (using **Type**) | Jumlah yang diperlukan oleh elemen | Range setiap elemen sama dengan range tipe data-nya. |

1. **Nama Fungsi dan Kelompoknya**

Pemrograman fungsional merupakan paradigma pemrograman yang melihat struktur dan elemen dari kode program sebagai kumpulan dari fungsi matematis yang saling berinteraksi.

[Javascript Secara Fungsional](https://bertzzie.com/knowledge/javascript/Javascript-Fungsional.html) :

1. [Fungsi Rekursif](https://bertzzie.com/knowledge/javascript/Javascript-Fungsional.html#fungsi-rekursif)
2. [Fungsi Tingkat Tinggi](https://bertzzie.com/knowledge/javascript/Javascript-Fungsional.html#fungsi-tingkat-tinggi)
3. [Currying](https://bertzzie.com/knowledge/javascript/Javascript-Fungsional.html#currying)
4. [Fungsi Teraplikasi Parsial](https://bertzzie.com/knowledge/javascript/Javascript-Fungsional.html#fungsi-teraplikasi-parsial)

Penjelasan lebih lanjutnya :

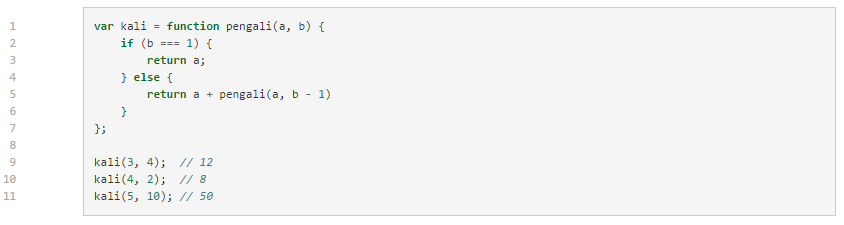
1. Fungsi Rekursif

Fungsi rekursif merupakan sebuah fungsi yang memanggil dirinya sendiri, baik secara langsung maupun tidak langsung, salah satu teknik penyelesaian masalah yang sangat berguna. Ketika menyelesaikan masalah secara rekursif, umumnya kita memecah-mecah masalah besar menjadi banyak masalah yang lebih kecil, dan menyelesaikan masalah kecil tersebut dengan fungsi rekursif.

Sebagai contoh, misalkan kita ingin membuat fungsi yang menghitung hasil kali dua bilangan. Secara sederhana, kita dapat mendefinisikan perkalian (sebuah masalah besar) sebagai penjumlahan (masalah kecil) yang dilakukan berulang kali, seperti berikut:

\[a \times b = \underbrace{a + a + a + ... + a}\_{\text{b kali}}\]

Melihat pemecahan masalah perkalian menjadi banyak penambahan seperti ini, kita dapat menuliskan fungsi Javascript untuk menghitung perkalian secara rekursif seperti berikut:



Pada baris pertama, kita membuat sebuah fungsi pengali dan kemudian menyimpan fungsi tersebut di dalam sebuah variabel kali. Pada pembuatan fungsi sebelumnya, kita tidak pernah memberikan nama fungsi ketika menggunakan fungsi ekspresi. Pemberian nama pada contoh ini penting, karena kita harus memanggil fungsi pengali di dalam dirinya sendiri nanti. Meskipun kita menggunakan nama fungsi dan nama variabel yang berbeda pada contoh ini, kita juga dapat menggunakan nama fungsi dan variabel yang sama. Penggunaan nama berbeda dilakukan untuk mempermudah pengertian, sementara di lapangan akan lebih sering ditemukan nama fungsi dan variabel yang sama.

Cara kerja fungsi pengali sangat sederhana. Kita terus menerus memanggil fungsi pengali dan mengurangi nilai b sampai nilai b menjadi 1. Langkah-langkah pemanggilan fungsi ini adalah seperti berikut:

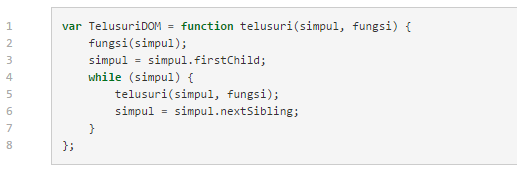


Dapat dilihat bagaimana yang kita lakukan sebenarnya hanyalah terus menerus mengurangi nilai b sembari menambahkan nilai a. Pengurangan nilai b ini sangat penting, karena tanpa mengurangi nilai b fugnsi tidak akan pernah selesai dieksekusi.

Secara umum, sebuah fungsi rekursif harus memiliki tiga komponen berikut untuk memastikan fungsi tersebut dapat berjalan dengan benar:

1. **Pemanggilan Diri Sendiri**. Sebagai syarat utama dari fungsi rekursif adalah, sesuai dengan namanya, pemanggilan kepada dirinya sendiri. Syarat pertama ini sudah pasti harus ada, sama seperti syarat sebuah mobil adalah memiliki minimal empat roda.
2. **Syarat berhenti**. Ketika sebuah fungsi memanggil dirinya sendiri, fungsi tersebut perlu tahu kapan ia harus berhenti. Tanpa adanya penanda kapan ia berhenti maka fungsi akan berjalan terus-menerus sampai memori habis atau dihentikan oleh pengguna. Pada fungsi kali di atas, syarat berhenti ada pada baris if (b === 1).
3. **Pergerakan Nilai ke Syarat Berhenti**. Untuk dapat mencapai syarat berhenti dari sebuah fungsi, tentunya harus ada sebuah nilai yang bertambah atau berkurang sampai syarat berhenti tersebut dicapai. Biasanya nilai yang bergerak ini disesuaikan dengan syarat berhentinya. Pada fungsi kali, kita mengurangi nilai b terus menerus pada baris return a + pengali(a, b - 1) karena nilai b yang menentukan apakah fungsi akan berhenti atau tidak. Pergerakan nilai ini juga biasanya dilakukan pada pemanggilan rekursif sebuah fungsi.

Fungsi rekursif cocok digunakan untuk perhitungan terhadap objek-objek yang tidak kita ketahui jumlahnya. Misalnya, teknik rekursif sangat efektif digunakan untuk penelusuran pohon (dalam konteks pengembangan web: DOM - Document Object Model milik HTML). Misalnya, kita dapat membuat sebuah fungsi untuk menelusuri DOM dan mengaplikasikan fungsi tertentu untuk tiap-tiap simpul (node) dalam pohon seperti berikut:



Pada fungsi TelusuriDOM di atas, kita mengunjungi setiap simpul elemen satu per satu. Kunjungan ke setiap simpul dilakukan melalui pengambilan anak pertama atau sepupu dari simpul, yaitu firstChild dan nextSibling. Setiap kali simpul masih memiliki anak atau sepupu, kita akan terus memanggil telusuri. Dengan cara kerja seperti ini, kita akan menelusuri pohon sedalam-dalamnya terlebih dahulu, baru kemudian melebarkan pencarian.

Syarat berhenti dari fungsi telusuri adalah ketika simpul sudah tidak memiliki anak atau sepupu lagi sehingga firstChild maupun nextSibling akan mengembalikan undefined dan menghentikan perulangan. Tepatnya, syarat berhenti ada pada baris while (simpul).

Pergerakan nilai ke syarat berhenti dari fungsi telusuri terjadi ketika kita memanggil firstChild ataupun nextSibling. Pada fungsi ini kita dapat melihat bagaimana yang terpenting dari syarat fungsi rekursif ini adalah bahwa nilai bergerak menuju ke syarat berhenti. Tidak penting apakah nilai bertambah atau berkurang, yang pasti adalah idealnya nilai akan terus bergerak sampai mencapai titik berhenti.

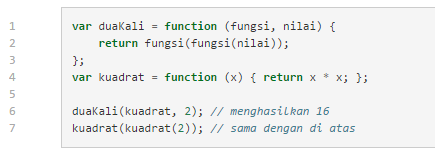
Terakhir, yang perlu diingat ketika menggunakan fungsi rekursif adalah bahwa penggunaan fungsi rekursif yang melakukan rekursi terlalu dalam (terlalu banyak memanggil diri sendiri) dapat menyebabkan *error* Stack Overflow. Hal ini karena setiap pemanggilan fungsi akan memakan memori sementara (*stack*) yang biasanya sangat kecil dan cepat habis. Pada beberapa bahasa pemrograman hal ini ditanggulangi dengan mengimplementasikan *tail recursion*, yang mengubah pemanggilan rekursif menjadi perulangan. Javascript sayangnya tidak mendukung hal ini. Karenanya, berhati-hatilah ketika menggunakan fungsi rekursif pada Javascript. Pastikan untuk tidak memanggil fungsi secara rekursif terlalu dalam.

1. Fungsi Tingkat Tinggi

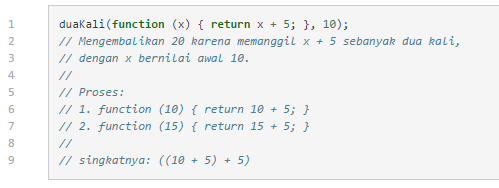
Fungsi tingkat tinggi (*higher-order function*, *functor*) pada konsep PF adalah sebuah fungsi yang minimal melakukan salah satu dari dua hal berikut:

1. Mengambil fungsi lain sebagai parameter, atau
2. Mengembalikan (menghasilkan) fungsi lain.

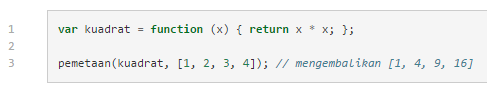
Konsep fungsi tingkat tinggi biasanya digunakan untuk membangun abstraksi pada kode yang dibangun, sehingga konsep ini biasanya digunakan dalam pengembangan pustaka (*library*). Misalnya, kita dapat membuat sebuah fungsi yang akan memanggil fungsi lain sebanyak dua kali, seperti berikut:



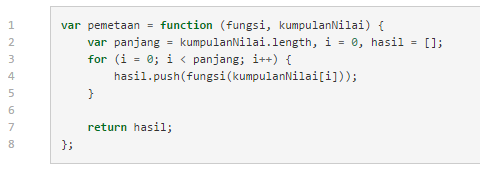
Fungsi duaKali menerima sebuah fungsi lain melalui parameter pertama fungsi, dan kemudian memanggil fungsisebanyak dua kali. duaKali merupakan fungsi tingkat tinggi karena ia menerima fungsi lain yang dispesifikasikan pengguna sebagai parameter. Kita juga dapat mengirimkan fungsi anonim kepadanya:



Contoh lain pemanfaatan fungsi tingkat tinggi yang mengambil fungsi lain sebagai parameter adalah fungsi pemetaan (map). Fungsi pemetaan memanggil fungsi tertentu kepada semua elemen dalam sebuah kumpulan nilai, dan mengembalikan hasil aplikasi fungsi tersebut ke dalam kumpulan nilai baru. Sederhananya, berikut adalah contoh pemanggilan fungsi pemetaan:

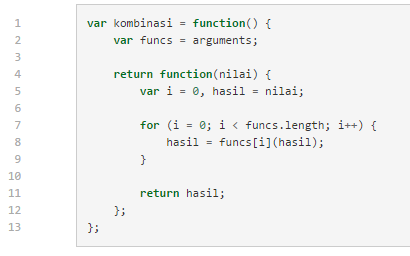


Seperti yang dapat dilihat dari kode di atas, fungsi pemetaan mengambil parameter pertama berupa fungsi yang akan diaplikasikan ke semua elemen dalam daftar nilai yang diberikan pada parameter kedua. Adapun deklarasi dari fungsi pemetaan adalah sebagai berikut:



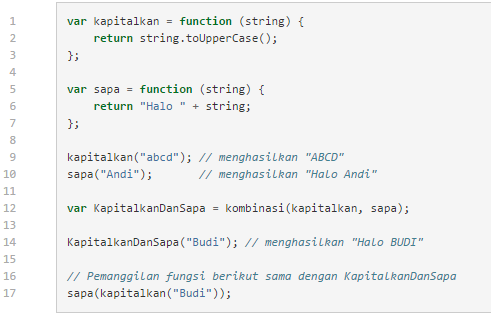
Cara kerja dari fungsi pemetaan cukup jelas dan sederhana. Kita hanya menelusuri isi dari kumpulanNilai satu per satu, dan kemudian memanggil fungsi dengan parameter berupa isi elemen kumpulanNilai. Hasil pemanggilan fungsi ini kemudian kita simpan ke dalam hasil, dan terkahir kita kembalikan nilai hasil kepada pengguna fungsi.

Memanfaatkan ide yang sama dengan fungsi pemetaan, kita dapat membuat sebuah fungsi yang menggabungkan beberapa fungsi lainnya menjadi fungsi baru. Konsep ini dikenal dengan nama komposisi fungsi (*function composition*). Implementasinya juga tidak terlalu rumit:



Fungsi kombinasi menerima daftar argumen berupa fungsi-fungsi yang ingin digabungkan. Fungsi ini memanfaatkan *closure* untuk menyimpan argumen-argumen yang dimilikinya ke dalam variabel funcs. Setelah menyimpan daftar fungsi yang digabungkan, kombinasi kemudian mengembalikan fungsi anonim yang menerima satu argumen saja: nilai. Fungsi baru ini kemudian menelusuri isi dari funcs, dan memanggil setiap fungsi di dala funcs satu per satu secara berurutan, dan mengembalikan nilai dari hasil pemanggilannya.

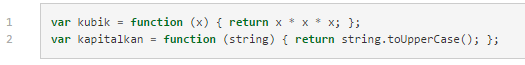
Untuk mempermudah pengertian, mari kita langsung lihat contoh kasus penggunaan kombinasi:



Meskipun dapat dikatakan sangat berguna, fungsi kombinasi memiliki beberapa keterbatasan, yaitu:

1. Fungsi yang dihasilkan hanya dapat menerima satu argumen saja. Hal ini jelas mengurangi fleksibilitas dari fungsi yang akan digabungkan.
2. Kita kehilangan konteks this pada fungsi yang dihasilkan. Hal ini akan menyulitkan kita jika kita ingin menggunakan teknik ini di dalam sebuah objek.
3. Fungsi-fungsi yang digabungkan harus mengembalikan nilai dengan tipe data yang dapat diterima oleh fungsi selanjutnya.

Penggabungan kedua fungsi ini misalnya:



Akan menghasilkan pesan kesalahan. Penyelesaian dari ketiga masalah di atas tidak akan dibahas, karena kita ingin berfokus pada konsep dari fungsi tingkat tinggi dan kombinasi. Solusi dari permasalahan akan diberikan kepada pembaca sebagai latihan.

1. Currying

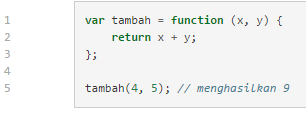
Currying merupakan teknik di mana kita mengubah sebuah fungsi dengan beberapa argumen menjadi serangkaian fungsi yang hanya memiliki satu argumen saja. Sederhananya, sebuah fungsi dengan dua argumen:

\[(x, y) \rightarrow x + y\]

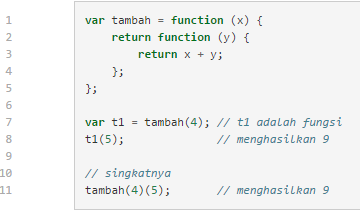
kita ubah menjadi:

\[x \rightarrow (y \rightarrow x + y)\]

Contoh currying dalam Javascript! Sebuah fungsi dengan dua argumen seperti berikut:

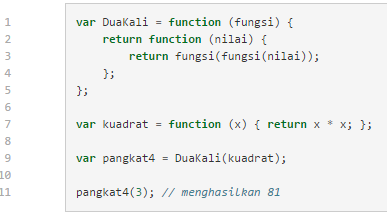


Dapat kita ubah menjadi fungsi yang hanya memiliki satu argumen dengan currying seperti berikut:



Pada teknik currying di atas, pada dasarnya kita mengubah fungsi tambah menjadi sebuah fungsi dengan satu argumen yang mengembalikan fungsi lainnya. Argumen yang diterima pada fungsi pertama ini (yaitu x) merupakan nilai pertama yang akan ditambahkan. Fungsi yang dikembalikan oleh fungsi awal kemudian menerima satu argumen lagi, yaitu argumen kedua (y), dan mengembalikan hasil penjumlahan yang benar.

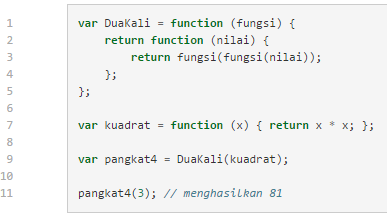
Pada penggunaan fungsi curry sendiri, kita memanggil fungsi sebanyak **dua kali**, yaitu dengan mengirimkan parameter pertama dan mendapatkan fungsi baru dan kemudian memanggil fungsi baru tersebut sambil mengirimkan parameter kedua. Teknik currying seperti ini memang sekilas terlihat hanya memperumit masalah tanpa keuntungan yang jelas. Currying merupakan tenik dasar pada PF, yang artinya kita dapat mengkombinasikan teknik ini dengan konsep lain dari PF untuk memaksimalkan kekuatannya. Misalnya, kita dapat menggunakan curry dengan fungsi tingkat tinggi untuk membuat fungsi yang membangun fungsi baru:



Walaupun sekilas terlihat sederhana, currying merupakan teknik yang sangat berguna untuk dipakai sebagai dasar pengembangan pustaka (library). Seperti yang dilihat pada contoh di atas, salah satu kegunaan utama currying adalah sebagai function compositor, atau fungsi yang membuat fungsi lainnya.

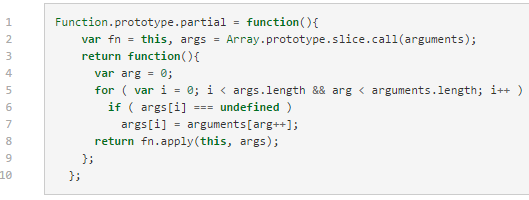
1. Fungsi Teraplikasi Parsial

Fungsi teraplikasi parsial (partially applied function) merupakan teknik pembuatan fungsi baru dari fungsi yang sudah ada dengan mengisikan beberapa parameter standar dari fungsi asal. Misalkan pada contoh sebelumnya:

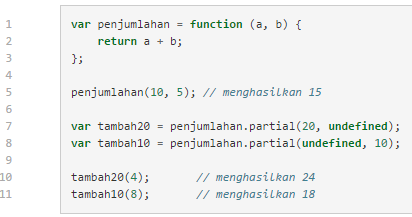


Jika fungsi DuaKali kita anggap sebagai sebuah fungsi yang memiliki dua parameter, maka fungsi pangkat4 adalah hasil penerapan fungsi teraplikasi parsial karena kita membuat fungsi baru melalui DuaKali dan parameter pertamanya. Singkatnya, fungsi teraplikasi parsial merupakan fungsi yang beberapa argumennya telah terisi.

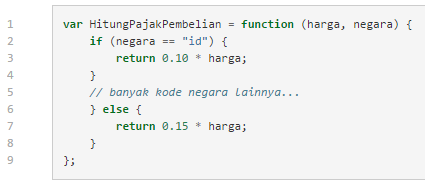
Pada contoh kode di atas, teknik fungsi teraplikasi parsial yang kita gunakan cukup kompleks. Terlebih dahulu kita harus membuat sebuah *curry*, dan kemudian memasukkan satu parameter terlebih dahulu. Tentunya keterbatasan ini akan menyulitkan kita, misalnya jika ingin mengisikan parameter pertama atau parameter yang ada di tengah dari 4 parameter. Untuk menanggulangi hal ini, kita dapat membuat fungsi yang melakukan aplikasi parsial tanpa memperhatikan urutan parameter, seperti berikut:



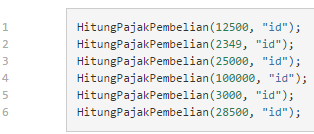
Fungsi partial di atas sangat sederhana dan gamblang. Kita hanya membuat sebuah fungsi baru, dan mengisikan argumen-argumen fungsi tersebut kepada fungsi lama, jika argumen bukan undefined. Berikut adalah contoh penggunaan fungsi partial:



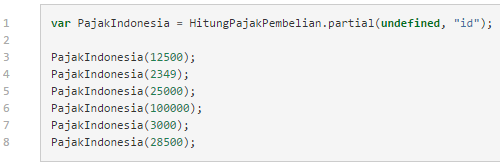
Fungsi teraplikasi parsial memiliki sangat banyak kegunaan, terutama jika kita ingin mengurangi perulangan dalam kode untuk perhitungan-perhitungan yang nyaris sama. Misalkan kita memiliki sebuah fungsi untuk menghitung pajak pembelian dari sebuah negara:



Ketika ingin menuliskan kode untuk melakukan banyak pembelian sekaligus pada satu negara, kita terpaksa akan harus mengirimkan parameter negara berkali-kali:



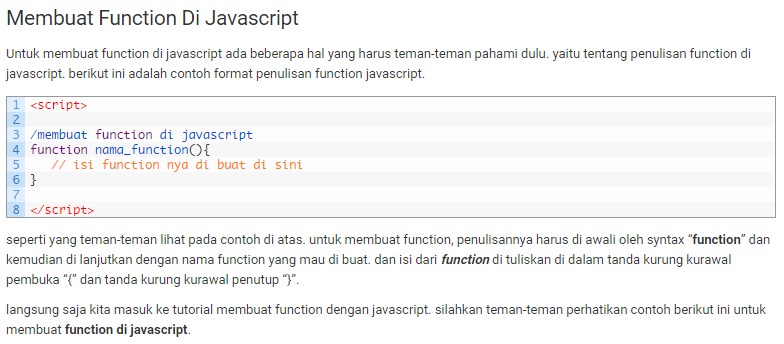
Ingat bahwa meskipun kita menggunakan perulangan untuk memasukkan kode di atas, pada dasarnya Javascript akan tetap mengirimkan parameter yang sama berulang kali. Untuk menyederhanakan kode dan pemanggilan fungsi, kita dapat menggunakan fungsi teraplikasi parsial untuk membuat fungsi baru seperti berikut:



Dengan begitu, parameter yang dikirimkan ke fungsi akan lebih sedikit, dan kode kita menjadi lebih sederhana.

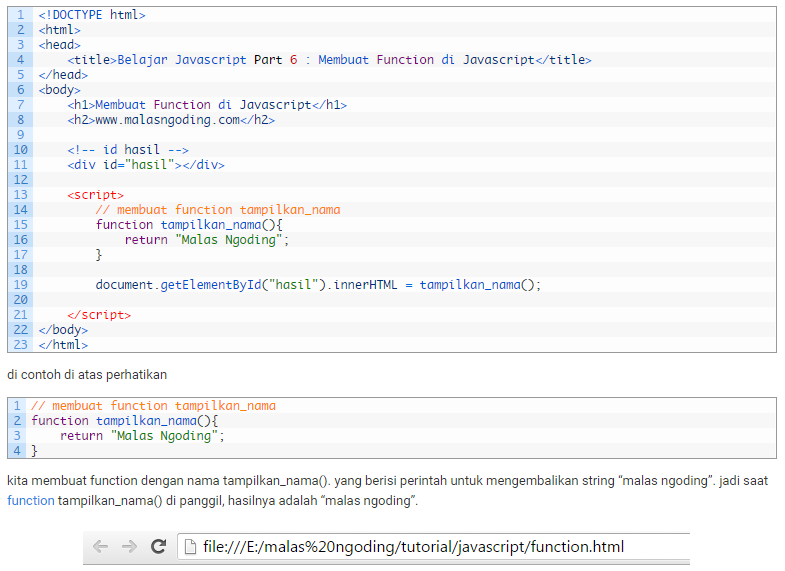
1. **Cara Membuat Fungsi pada Javascript**

Untuk membuat function di javascript ada beberapa hal yag harus dipahami dulu, yaitu tentang penulisan function di javasricpt. Berikut ini adalah contoh format penulisan function javascript :

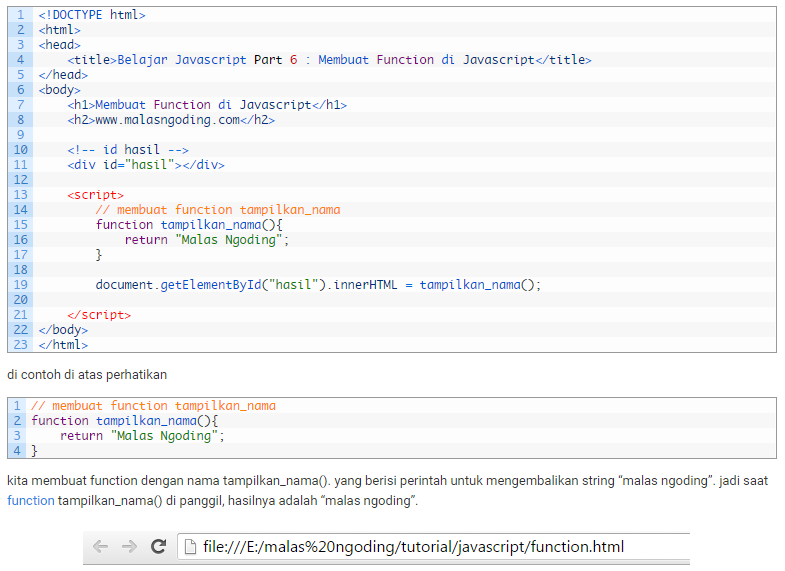


Seperti yang terlihat pada contoh di atas, untuk membuat function, penulisannya harus di awali oleh syntax “function” dan kemudian di lanjutkan dengan nama function yang mau dibuat da nisi dari function di tuliskan di dalam tanda kurung kuarawal pembuka “ { “ dan tanda kurung kurawal penutup “ } “.

Berikut tutorial membuat function dengan javascript. Perhatikan contoh berikut ini :



Pada contoh diatas perhatikan pada bagian ini :



Kita membuat function dengan nama tampilkan\_nama(), yang berisi perintah untuk megembalikan string “malas ngoding”, jadi saat function tampilkan\_nama(), maka hasilnya adalalah “malas ngoding”.

