|  |  |
| --- | --- |
| NAMA | Rezky Desta Kusuma Wardana |
| NIM | L200140066 |
| KELAS | A |

1. **Daftar Type Data dalam Java Script**
2. **Tipe data primitive atau tipe data Sederhana**

Tipe data primitive adalah Tipe data yang mampu menyimpan satu nilai tiap satu variable

Contoh tipe data primitive :

* **Numeric** : Tipe data numeric digunakan pada variabel atau konstanta untuk menyimpan nilai dalam bentuk angka.
  + Integer : Merupakan tipe data berupa bilangan bulat
  + Real : bilangan yang berisi titik desimal atau jenis bilangan pecahan
* **Karakter (char)** : Karakter merupakan tipe data yang hanya mampu menyimpan 1 digit karakter.  Ukuran untuk tipe data karakter adalah 1 byte (1 byte = 8 bit). Adapun macam karakter yang ada sejumlah 256 macam karakter yaitu dari kode karakter (ASCII), 0 sampai dengan 255
* **Boolean** : Merupakan tipe data logika, yang berisi dua kemungkinan nilai: TRUE (benar) atau FALSE (salah)

1. **Tipe Data String :** Merupakan suatu data yang menyimpan array (larik), sebagai contoh 'ABCDEF' merupakan sebuah konstanta string yang berisikan 6 byte karakter. Ukuran Tempat untuk tipe data ini adalah 2 s/d 256 byte, dengan jumlah elemen 1 s/d 255
2. **Tipe Null :** Tipe Null digunakan untuk merepresentasikan variabel yang tidak diberi nilai awal (inisialisasi).
3. **Daftar nama fungsi dan Kelompoknya**
4. **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:

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.

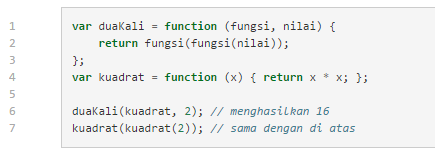
Fungsi rekursif cocok digunakan untuk perhitungan terhadap objek-objek yang tidak kita ketahui jumlahnya. Misalnya, teknik rekursif sangat efektif digunakan untuk penelusuran pohon (dalam konteks pengembangan web: DOM - Document Object Model milik HTML). Misalnya, kita dapat membuat sebuah fungsi untuk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

1. **Fungsi Tingkat Tinggi**

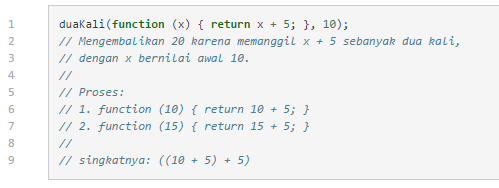
Fungsi tingkat tinggi (*higher-order function*, *functor*) pada konsep PF adalah sebuah fungsi yang minimal melakukan salah satu dari dua hal berikut:

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

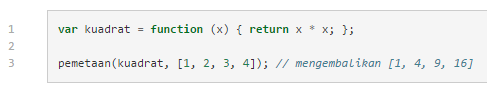
Konsep fungsi tingkat tinggi biasanya digunakan untuk membangun abstraksi pada kode yang dibangun, sehingga konsep ini biasanya digunakan dalam pengembangan pustaka (*library*). Misalnya, kita dapat membuat sebuah fungsi yang akan memanggil fungsi lain sebanyak dua kali, seperti berikut:



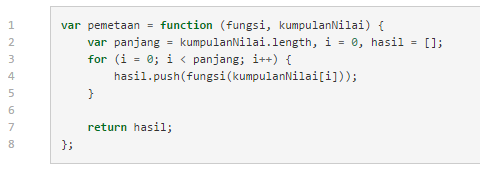
Fungsi duaKali menerima sebuah fungsi lain melalui parameter pertama fungsi, dan kemudian memanggil fungsisebanyak dua kali. duaKali merupakan fungsi tingkat tinggi karena ia menerima fungsi lain yang dispesifikasikan pengguna sebagai parameter. Kita juga dapat mengirimkan fungsi anonim kepadanya:



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:

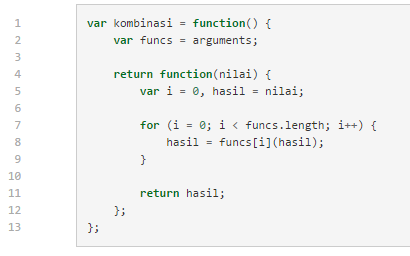


Seperti yang dapat dilihat dari kode di atas, fungsi pemetaan mengambil parameter pertama berupa fungsi yang akan diaplikasikan ke semua elemen dalam daftar nilai yang diberikan pada parameter kedua. Adapun deklarasi dari fungsi pemetaan adalah sebagai berikut:



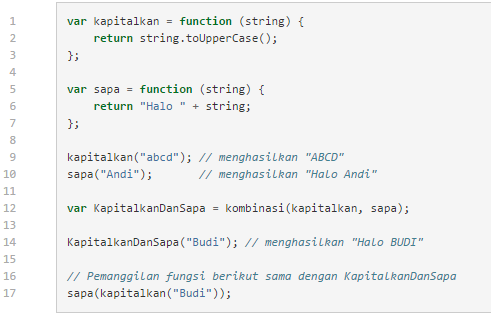
Cara kerja dari fungsi pemetaan cukup jelas dan sederhana. Kita hanya menelusuri isi dari kumpulanNilai satu per satu, dan kemudian memanggil fungsi dengan parameter berupa isi elemen kumpulanNilai. Hasil pemanggilan fungsi ini kemudian kita simpan ke dalam hasil, dan terkahir kita kembalikan nilai hasil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

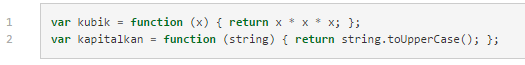
Untuk mempermudah pengertian, mari kita langsung lihat contoh kasus penggunaan kombinasi:



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**

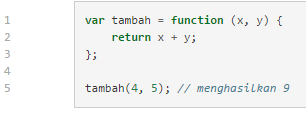
Currying merupakan teknik di mana kita mengubah sebuah fungsi dengan beberapa argumen menjadi serangkaian fungsi yang hanya memiliki satu argumen saja. Sederhananya, sebuah fungsi dengan dua argumen:

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

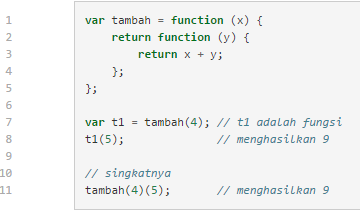
kita ubah menjadi:

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

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

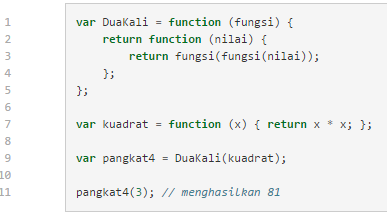


Dapat kita ubah menjadi fungsi yang hanya memiliki satu argumen dengan currying seperti berikut:



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.

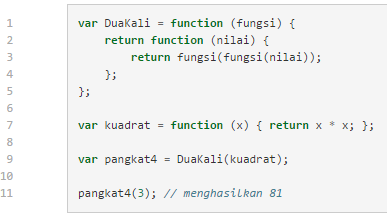
Pada penggunaan fungsi curry sendiri, kita memanggil fungsi sebanyak **dua kali**, yaitu dengan mengirimkan parameter pertama dan mendapatkan fungsi baru dan kemudian memanggil fungsi baru tersebut sambil mengirimkan parameter kedua. Teknik currying seperti ini memang sekilas terlihat hanya memperumit 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.

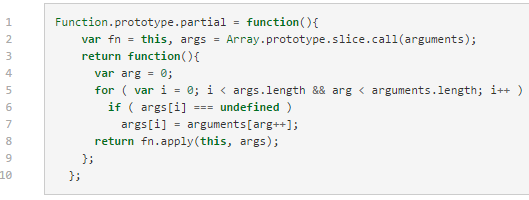
1. **Fungsi Teraplikasi Parsial**

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:

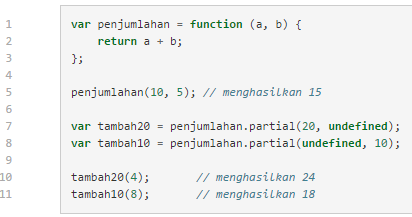


Jika fungsi DuaKali kita anggap sebagai sebuah fungsi yang memiliki dua parameter, maka fungsi pangkat4 adalah hasil penerapan fungsi teraplikasi parsial karena kita membuat fungsi baru melalui DuaKali dan parameter pertamanya. Singkatnya, fungsi teraplikasi parsial merupakan fungsi yang beberapa argumennya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

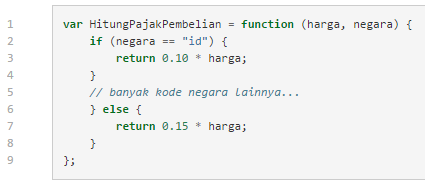
Pada contoh kode di atas, teknik fungsi teraplikasi parsial yang kita gunakan cukup kompleks. Terlebih dahulu kita harus membuat sebuah *curry*, dan kemudian memasukkan satu parameter terlebih dahulu. Tentunya keterbatasan ini akan menyulitkan kita, misalnya jika ingin mengisikan parameter pertama atau 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:



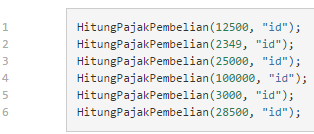
Fungsi partial di atas sangat sederhana dan gamblang. Kita hanya membuat sebuah fungsi baru, dan mengisikan argumen-argumen fungsi tersebut kepada fungsi lama, jika argumen bukan undefined. Berikut adalah contoh penggunaan fungsi partial:



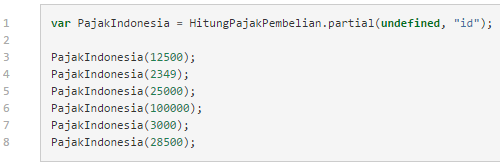
Fungsi teraplikasi parsial memiliki sangat banyak kegunaan, terutama jika kita ingin mengurangi perulangan dalam kode untuk perhitungan-perhitungan yang nyaris sama. Misalkan kita memiliki sebuah fungsi untuk menghitung pajak pembelian dari sebuah negara:



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.

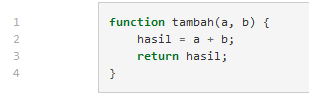
3. **Cara Membuat Fungsi pada Javascript**

Fungsi adalah sebuah blok kode yang mengeksekusi hanya bila Anda mengatakan itu untuk mengeksekusi.

Hal ini dapat terjadi ketika sebuah peristiwa, seperti ketika pengguna mengklik tombol, atau dari panggilan dalam naskah Anda, atau dari panggilan dalam fungsi lain.

Fungsi dapat ditempatkan baik di head dan di bagian <body> dokumen, pastikan bahwa fungsi ada, ketika panggilan dilakukan.

Sebuah fungsi pada Javascript dibuat dengan cara seperti berikut:

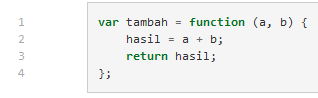


Cara penulisan fungsi seperti ini dikenal dengan nama *function declaration*, atau deklarasi fungsi. Terdapat empat komponen yang membangun fungsi yang baru kita definisikan di atas, yaitu:

* Kata kunci **function**, yang memberitahu Javascript bahwa kita akan membuat fungsi.
* **Nama fungsi**, dalam contoh di atas adalah tambah. Dengan memberikan sebuah fungsi nama maka kita dapat merujuk ke fungsi tersebut dengan nama yang diberikan. Harus diingat bawa nama fungsi bersifat *opsional*, yang berarti **fungsi pada Javascript tidak harus diberi nama**. Kita akan membahas tentang hal ini lebih dalam nanti.
* Daftar parameter fungsi, yaitu a, b pada contoh di atas. Daftar parameter ini selalu dikelilingi oleh tanda kurung (()). Parameter boleh kosong, tetapi tanda kurung wajib tetap dituliskan. Parameter fungsi akan secara otomatis didefinisikan menjadi variabel yang hanya bisa dipakai di dalam fungsi. Variabel pada parameter ini diisi dengan nilai yang dikirimkan kepada fungsi secara otomatis.
* Sekumpulan perintah yang ada di dalam kurung kurawal ({}). Perintah-perintah ini dikenal dengan nama badan fungsi. Badan fungsi dieksekusi secara berurut ketika fungsi dijalankan.

Penulisan deklarasi fungsi (*function declaration*) seperti di atas merupakan cara penulisan fungsi yang umumnya kita gunakan pada bahasa pemrograman imperatif dan berorientasi objek.

Tetapi selain deklarasi fungsi Javascript juga mendukung cara penulisan fungsi lain, yaitu dengan memanfaatkan ekspresi fungsi (*function expression*). Ekspresi fungsi merupakan cara pembuatan fungsi yang memperbolehkan kita melewatkan nama fungsi. Fungsi yang dibuat tanpa nama dikenal dengan sebutan fungsi anonim atau fungsi lambda. Berikut adalah cara membuat fungsi dengan ekspresi fungsi:



Terdapat hanya sedikit perbedaan antara ekspresi fungsi dan deklarasi fungsi:

* Penamaan fungsi. Pada deklarasi fungsi, kita langsung memberikan nama fungsi sesuai dengan sintaks yang disediakan Javascript. Menggunakan ekspresi fungsi kita pada dasarnya menyimpan sebuah fungsi anonim ke dalam variabel, dan nama fungsi adalah nama variabel yang kita buat. Perlu diingat juga bahwa pada dasarnya ekspresi fungsi *adalah* fungsi anonim. Penyimpanan ke dalam variabel hanya diperlukan karena kita akan memanggil fungsi nantinya.
* Ekspresi fungsi dapat dipandang sebagai sebuah ekspresi atau perintah standar bagi Javascript, sama seperti ketika kita menuliskan kode var i = 0;. Deklarasi fungsi merupakan konstruksi khusus untuk membuat fungsi. Hal ini berarti pada akhir dari ekspresi fungsi kita harus menambahkan ;, sementara pada deklarasi fungsi hal tersbut tidak penting.

Karena pada Javascript sebuah fungsi juga adalah sekaligus sebuah objek. Setiap kali kita menciptakan fungsi, pada dasarnya kita membuat sebuah objek Function baru, dengan nama yang kita berikan. Karenanya, secara eksplisit menuliskan bahwa kita membuat objek baru dan memperlakukan objek tersebut sama seperti perintah-perintah lain dalam program akan menyederhanakan kode program kita, yang pada akhirnya akan mempermudah kita mengerti kode kita nantinya.

Aturan pembuatan fungsi, baik ekspresi fungsi maupun deklarasi fungsi, sama dengan aturan penulisan ekspresi. Di mana kita dapat menuliskan ekspresi, kita dapat mendefinisikan fungsi juga. Karena aturan ini, maka kita juga dapat mendefinisikan fungsi di dalam fungsi lainnya. Fungsi yang berada di dalam fungsi lainnya memiliki akses terhadap semua variabel yang ada pada fungsi penampungnya. Keterhubungan fungsi di dalam fungsi ini dikenal dengan nama *closure*.