BKPM

(BUKU KERJA PRAKTIK MAHASISWA)

TIF3607

WORKSHOP SI BERBASIS WEB

SEMESTER 3

Penyusun:
Khafidurrohman Agustianto, S.Pd., M.Eng.
NIP. 19911211 201803 1 001
NIDN. 0011129102

D4 TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI JEMBER
2019/2020

KEMENTRIAN RISET, TEKNOLOGI DAN PENDIDIKAN TINGGI
POLITEKNIK NEGERI JEMBER

LEMBAR PENGESAHAN

WORKSHOP SI BERBASIS WEB

Penulis,

Khafidurrohman Agustianto, S.Pd., M.Eng.
NIP. 19911211 201803 1 001

Mensetujui,	**Mengetahui,**
Ketua Jurusan	**Kepala Program Studi**
Hendra Yufit Riskiawan, S.Kom., M.Cs.	**Elly Antika, S.T., M.Kom.**
NIP. 19830203 200604 1 003	**NIP. 19781011 200501 2 002**

DAFTAR ISI

VERSI BAHASA INDONESIA

ATURAN PRAKTIK WORKSHOP SI BERBASIS WEB

- Masing-masing kelompok membuat Repo. di GitHub untuk Project-nya dan mengundang dosen pengampu dalam Repo.
- Kelas Dilaksanakan 2X/Minggu, dengan ketentuan:
  - Pertemuan pertama digunakan untuk mebahas materi
  - Pertemuan kedua digunakan untuk berkonsultasi dan melaporkan hasil workshop
- Tidak boleh terlambat
  - Keterlambatan dihitung setelah dosen hadir di kelas
- Berpakaian sopan
  - Tidak memakai sandal/Sepatu-Sendal/Sejenis
  - Tidak memakai Baju dan Celana Robek
  - Tidak memakai Kaos/Oblong/T-Shirt
  - Tidak memakai Topi/Penutup Kepala Lainya
- Dilarang makan tetapi diperbolehkan minum, dengan tetap menjaga jarak aman dengan komputer
- Boleh tidur di kursi jika mengantuk

PENILAIAN PRAKTIK WORKSHOP SI BERBASIS WEB

- Presensi Perkuliahan/Attitude
- Jumlah Commit Masing-Masing Contributor
- Melihat Kontrobusi dari Masing-Masing Anggota Kelompok
- Kesesuaian Perencaan dengan Produk Akhir
  - TIF3603 Manajemen Basis Data
  - TIF3604 Interaksi Manusia dan Komputer
  - TIF3605 Pemodelan Sistem Informasi
  - TIF3606 Workshop Pengembangan Perangkat Lunak
  - TIF3607 Workshop Sistem Informasi Berbasis Web
- Penilaian Demo Aplikasi di Akhir Semester
- Penguasaan Masing-Masing Anggota

KELENGKAPAN PRAKTIK WORKSHOP SI BERBASIS WEB

- GitHub Desktop
- Visual Studio Code (Disarankan), atau Bisa Menggunakan Aplikasi Sejenis
- XAMPP
- Composer
- Hosting (disediakan Politeknik Negeri Jember)
- Domain (disediakan Politeknik Negeri Jember)
- ATK (disediakan Politeknik Negeri Jember)
- Media Backup RAM dan Penyimpanan Lokal (SSD) (disediakan Politeknik Negeri Jember)

PRAKTIK 1: Pengenalan dan Instalansi Alat Pengembangan WEB (Git, GitHub, XAMPP dan Sublime Text 3)

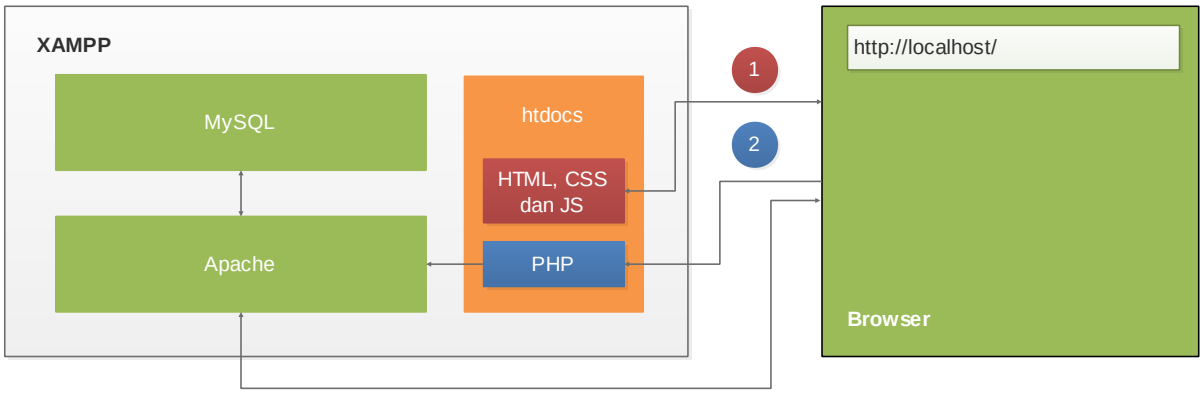
Matakuliah : Workshop SI berbasis WEB
Minggu Ke : 1
Waktu : 2 x 100 menit
Tema : Pengenalan dan Instalansi Alat Pengembangan WEB (Git, GitHub, XAMPP dan Sublime Text 3)

1. Kompetensi Dasar

A. Mahasiswa mampu memahami ekosistem pengembangan WEB
B. Mahasiswa mampu melakukan instalasi alat pengembangan WEB (Git, GitHUb, XAMPP dan Sublime Text 3)

2. Dasar Teori

A. Pengenalan Pengembangan WEB



Gambar 1. Gambaran Pengembangan WEB

Seperti pada proses pengembangan platform pada umumnya, pegembangan WEB juga memiliki beragam lingkungan pengembangan. Pada praktikum ini kita akan menggunakan LocalHost sebagai lingkungan pengembangan. Sehingga saudara dituntut untuk memahami mekanisme eksekusi (compiler) untuk asing-masing file.

Secara garis besar praktikum kita akan dibagi menjadi dua segmen, yaitu mengesekusi file yang diproses pada ***client side*/*browser*** (HTML, CSS dan JS) dan ***server side*** (PHP). Mekanisme ***client side*** ditunjukan oleh hubungan 1 pada Gambar 1, dimana untuk file HTML, CSS dan JS dapat langsung diesekusi oleh *browser* (file dapat dipanggil secara langsung). Sedangkang mekanisme ***server side*** permintaan oleh *browser* tidak dapat langsung diesekusi tetapi harus diproses terlebih dahulu oleh

server, dalam praktikum ini kita mengiakan *server Apache*. Server PHP yang digunakan sudah termasuk di dalam paket XAMPP.

B. XAMPP

- https://www.niagahoster.co.id/blog/cara-menggunakan-xampp/

XAMPP terdiri dari cross-platorm(X), berupa Apache, MySQL, PHP dan Perl. Ini adalah aplikasi yang memberikan solusi untuk Anda yang ingin menjalankan web server di lokal komputer ketika akan melakukan pengetesan aplikasi yang dibuat.

Sehingga Anda tidak perlu **membeli hosting** atau server untuk melakukan pengetesan website. **Catatan**: Saat ini XAMPP telah mengganti MySQL dengan MariaDB. Meskipun demikian, WordPress akan tetap bisa dijalankan dengan baik. MariaDB adalah hasil pengembangan dari MySQL, untuk keterangan lebih lanjut dapat Anda baca mealui tautan **berikut**.

Komponen pada XAMPP

- XAMPP (dengan PHP 7.2.1)
- Apache versi 2.4.29 adalah aplikasi web server default;
- MariaDB versi 10.1.30 adalah sistem manajemen database;
- PHP versi 7.2.1 adalah server side scripting untuk membuat aplikasi berbasis web;
- phpMyAdmin versi 4.7.4 adalah tool untuk menggunakan MySQL berbasis web;
- OpenSSL versi 1.1.0g adalah implementasi open-source dari dua protokol keamanan populer, yaitu SSL dan TSL ;
- XAMPP Control Panel versi 3.2.2 adalah kontrol panel sederhana untuk mengatur komponen berbeda pada XAMPP;
- Webalizer versi 2.23-04 adalah sebuah tool analitik untuk user log dan metrik penggunaan;
- Mercury Mail Transport System versi 4.63 adalah email server open source;
- FileZilla FTP Server versi 0.9.41 berfungsi untuk melakukan transfer file;
- Tomcat versi 7.0.56 adalah java servlet freeware untuk aplikasi Java;
- Strawberry Perl 7.0.56 Portable berfungsi untuk melakukan distribusi Perl.

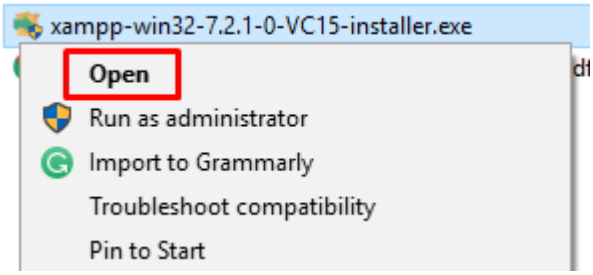
Cara Menggunakan XAMPP

Untuk menggunakan XAMPP sangatlah mudah, Anda hanya perlu mengunduhnya melalui website Apache Friends, kemudian lakukan instalasi dan jalakan modul yang ingin digunakan dengan menekan tombol start.

a). Lakukan Instalasi XAMPP

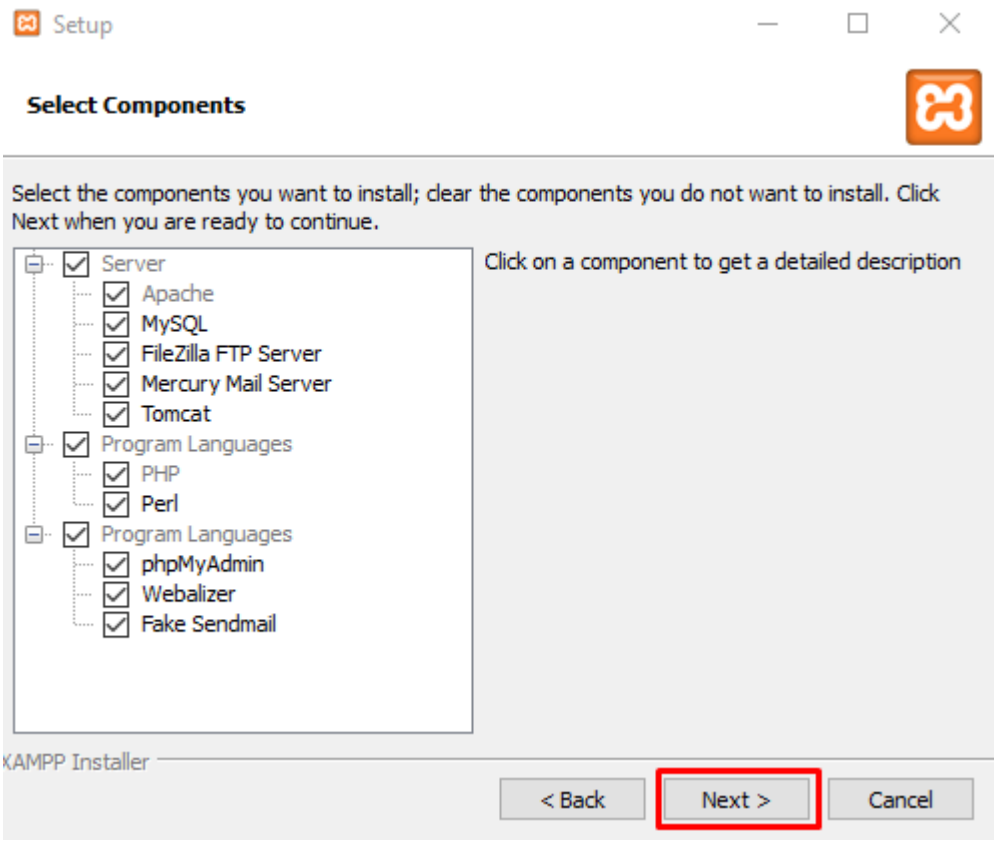
1. Silakan buka **website Apache Friends**, kemudian download sesuai sistem operasi Anda, pada contoh ini kami akan melakukan instalasi XAMPP di Windows.

2. Setelah selesai didownload, silakan lakukan instalasi XAMPP, dengan cara klik kanan pada file instalasi kemudian pilih **Open**.

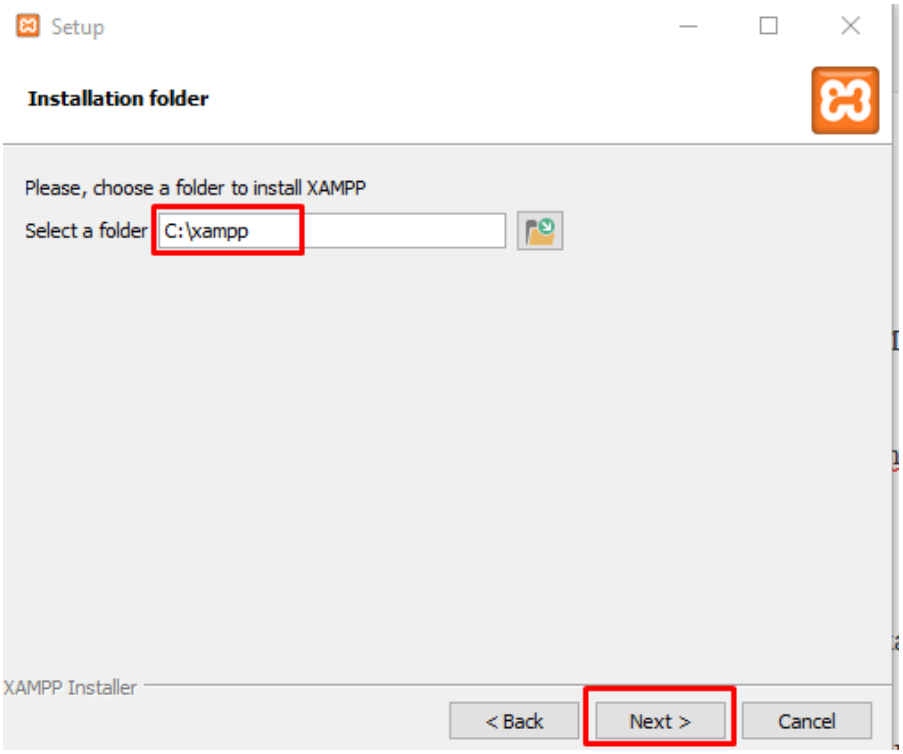


3. Jika pada saat melakukan instalasi muncul peringatan yang bertujuan untuk memastikan apakah Anda akan menginstal aplikasi ini, Silakan klik Ok/Yes untuk melanjutkan instalasi.

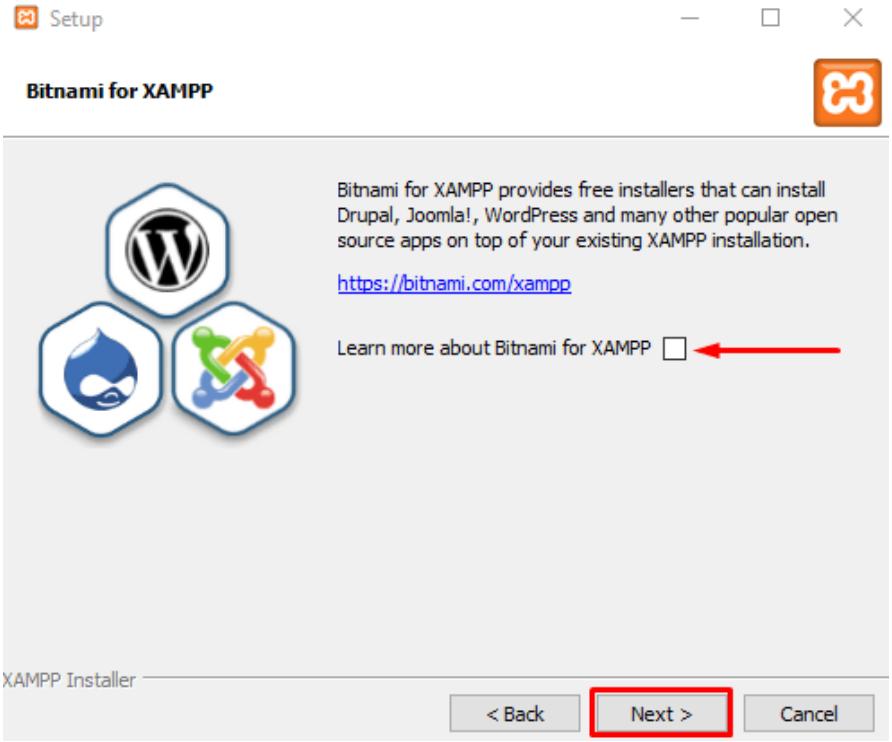
4. Selanjutnya akan tampil pilihan aplikasi apa yang akan Anda instal dan tidak ingin Anda instal. Beberapa aplikasi akan terinstal secara otomatis untuk menjalankan website seperti Apache dan PHP.



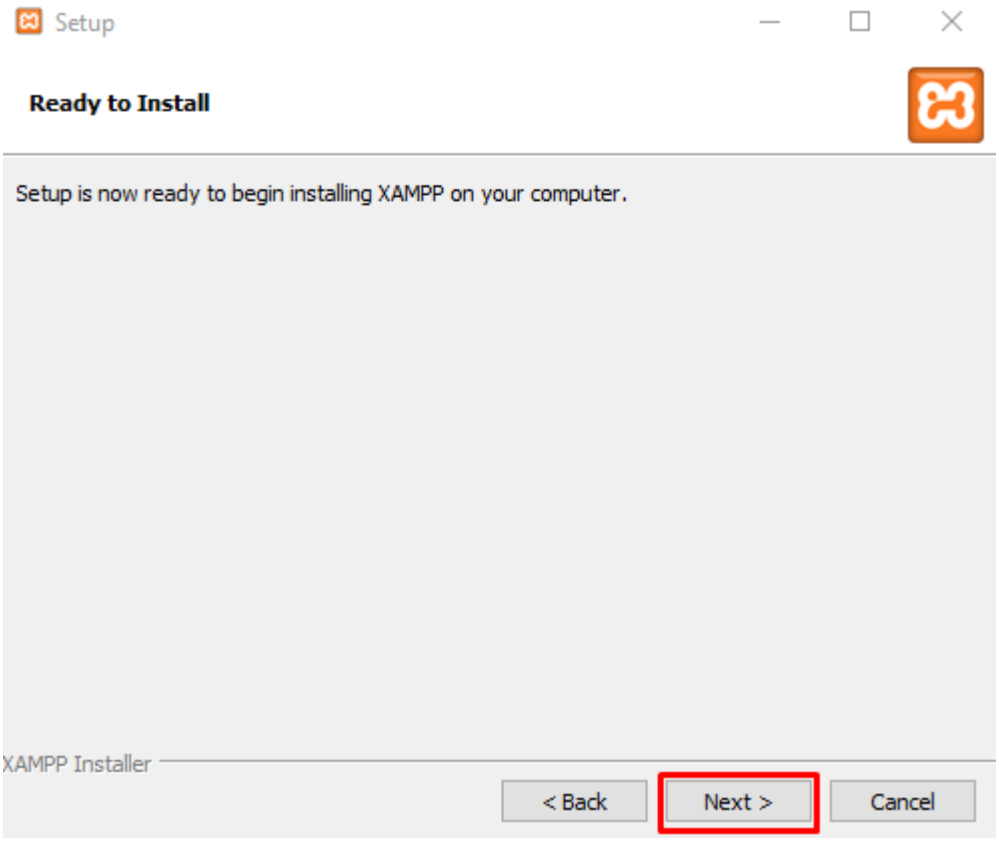
5. Jika tujuan menggunakan XAMPP adalah untuk menjalankan WordPress di localhost, pastikan untuk MySQL dan phpMyAdmin sudah tercentang seperti gambar di atas.
6. Selanjutnya, silakan pilih folder dimana file instalasi disimpan. Pada langkah ini kami menginstalnya langsung di folder **C:\XAMPP**. Hal ini karena untuk UAC tidak kami nonaktifkan.



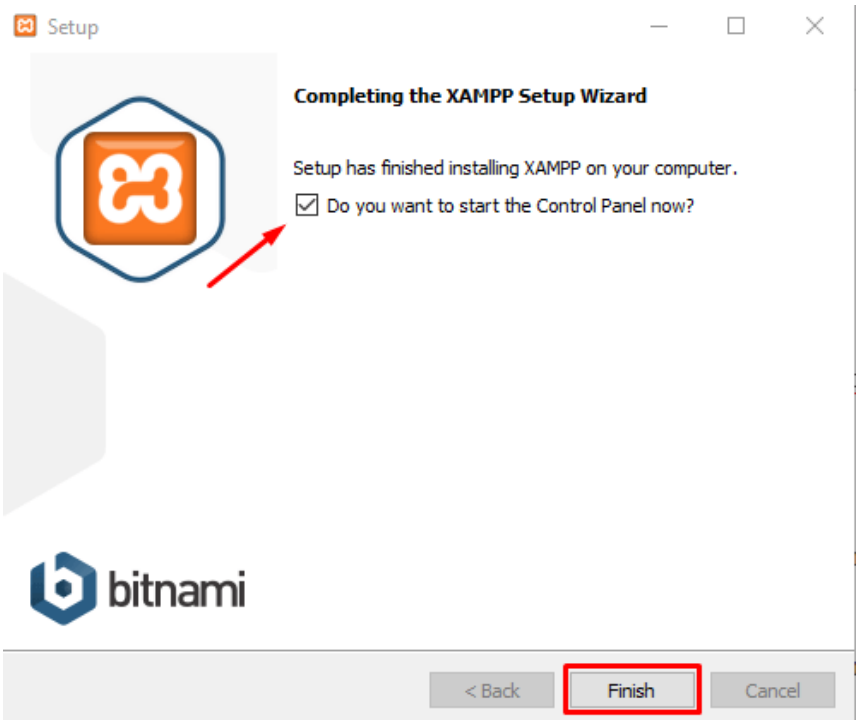
7. Akan tampil halaman yang menanyakan apakah Anda ingin menginstal Bitnami untuk XAMPP, yang akan berguna untuk melakukan instalasi beberapa CMS seperti WordPress, Drupal, dan Joomla.



8. Pada langkah ini XAMPP sudah siap untuk proses instalasi, silakan klik tombol **Next**.

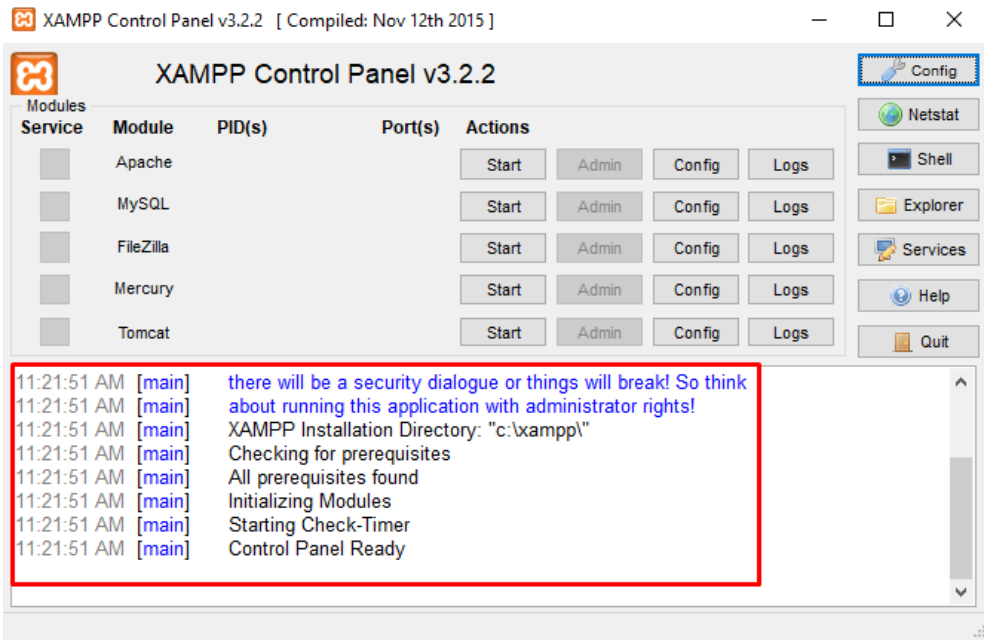


9. Jika sudah terinstal, akan tampil pertanyaan mengenai apakah Anda ingin langsung menjalankan kontrol panel. Pastikan pilihan tersebut sudah tercentang, kemudian klik tombol **Finish**.



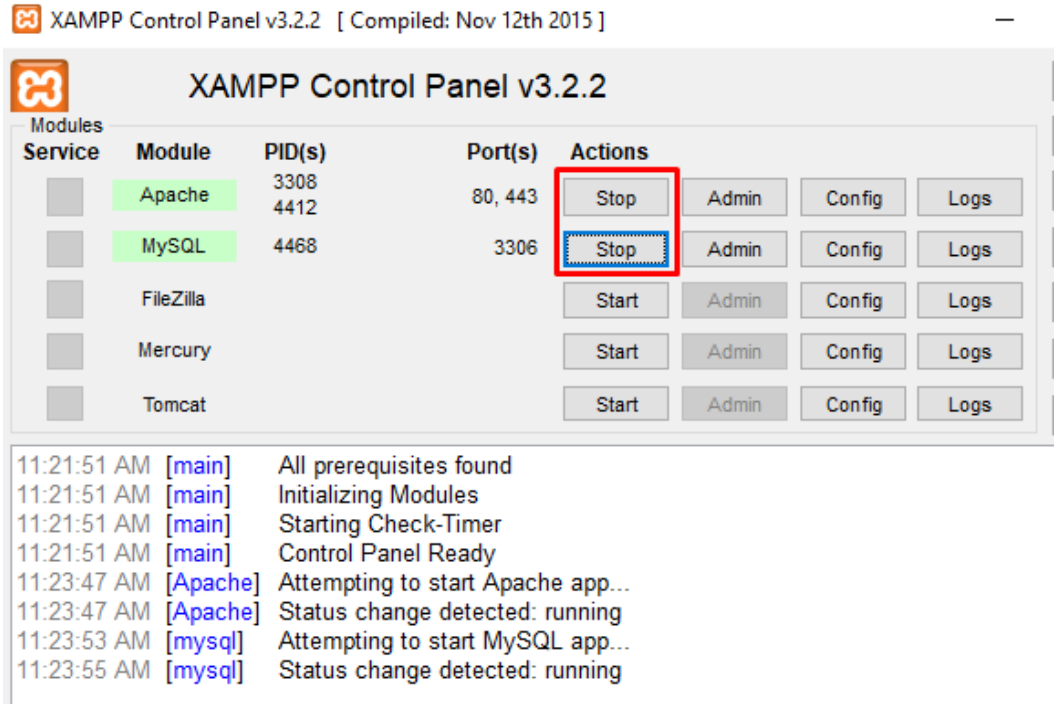
10. Kontrol panel akan otomatis muncul, tapi jika Anda tidak mencentang pilihan di halaman sebelumnya, maka Anda perlu membuka langsung kontrol panel melalui start menu atau folder XAMPP di komputer Anda.

Jika proses instalasi berjalan dengan baik, maka akan tampil tulisan berwarna biru dan hitam pada bagian kolom bawah di kontrol panel.

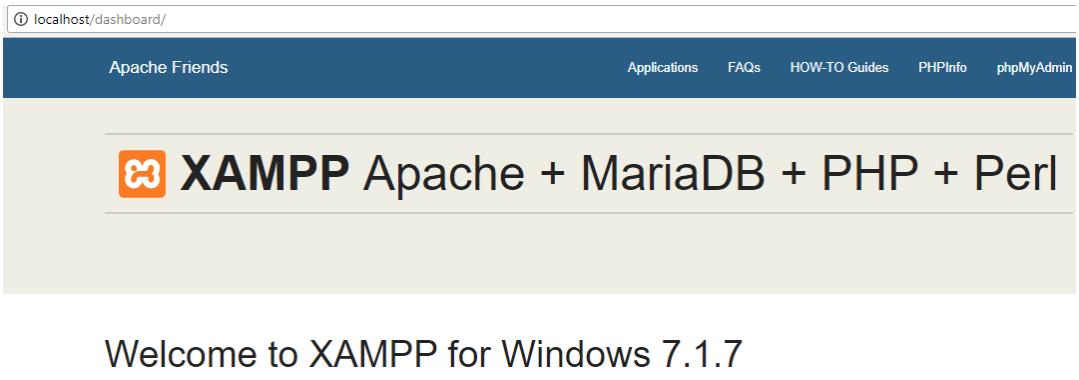


b. Jalankan XAMPP

Setelah Anda berhasil menginstal, kemudian klik **Start** untuk **Apache** dan **MySQL**.



Setelah keduanya sudah berjalan tanpa error, silakan akses localhost melalui **http://localhost** atau **127.0.0.1** pada browser.



C. GitHub

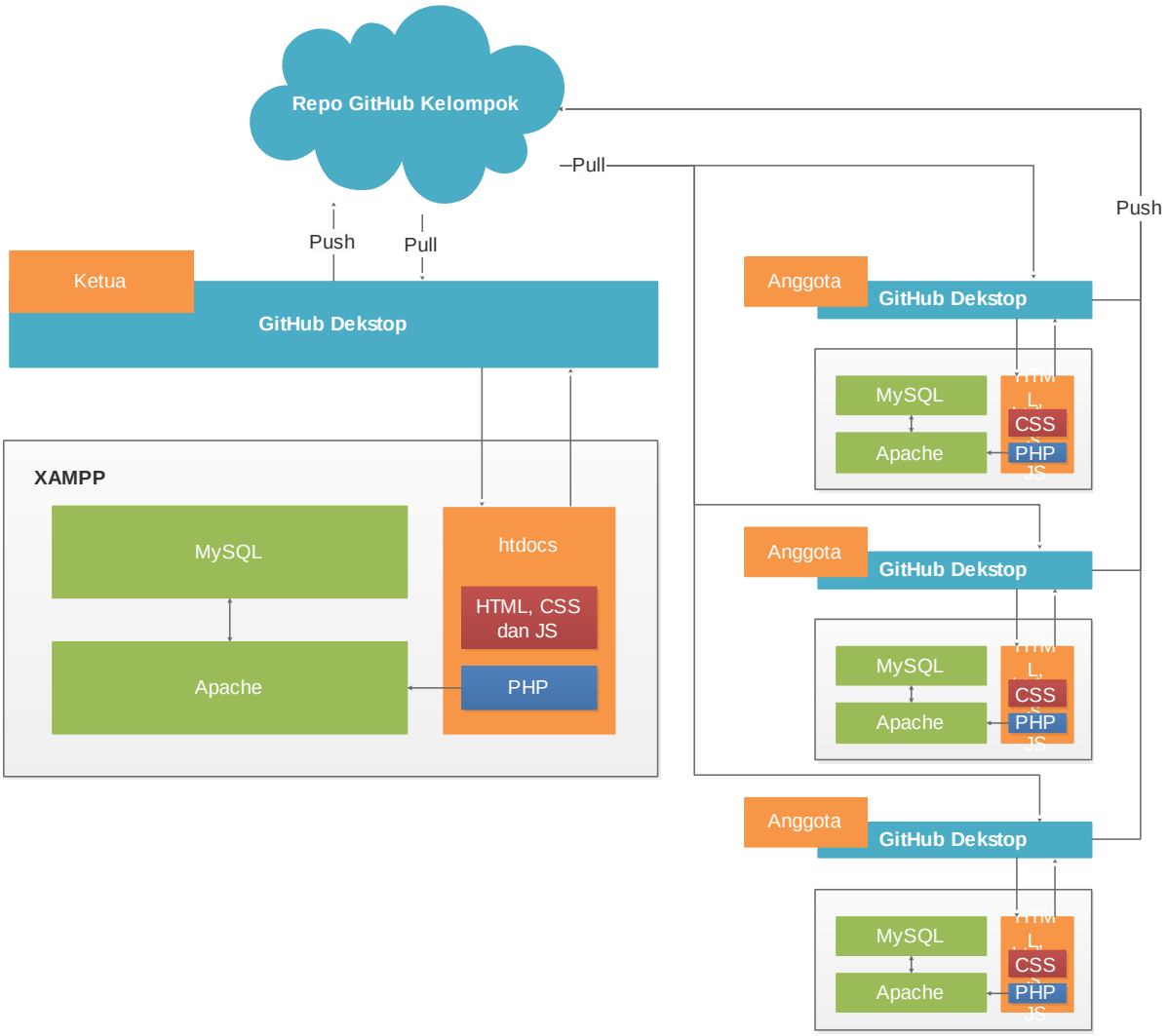
- http://rogerdudler.github.io/git-guide/index.id.html
- https://github.com/

GitHub is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

GitHub vs Bitbucket

GiHub	**Bitbucket**
First of all, Github only hosts projects that use the Git VCS. That's it. Nothing else. But Git is far and away the most commonly used VCS, so Github is still the largest code host of them all, with some 13.7 million-plus repositories of code. Git was originally started in 2008, and was written in both Ruby and Erlang. Github is designed to encourage close communication and collaboration within development teams. To this end it includes features like highlighted code comments and collaborative code review.	Right off the bat, Bitbucket's advantage over Github is that it supports the Mercurial VCS in addition to Git. But it also doesn't support SVN, yet. Bitbucket is written in Python and uses the Django web framework. Bitbucket was also launched in 2008 in Australia and was originally an independent startup offering hosting only for Mercurial projects. It was acquired in 2010 by fellow Australian company Atlassian, and about a year later added support for Git repos

Fitur: 1. An integrated issue tracker right within your project 2. Milestones and labels within projects 3. Branch comparison views 4. Native applications for Windows and Mac desktops, and also an Android app 5. Support for over 200 programming languages and data formats 6. Github pages, a feature for publishing and hosting within Github 7. Security such as use of SSL, SSH and/or https for all data transmission, and two-factor authentication for login 8. API integration for easy integration of 3rd-party tools, and integration with a large number of other tools and platforms. Some examples are Asana and Zendesk for issue/ bug tracking; CloudBees, Travis and CodeClimate for Continuous Integration (CI); AWS, Windows Azure, Google Cloud, and Heroku cloud hosting. 9. The Github guys also recognize that SVN is also a widely used alternative to Git, so they provide a tool to import SVN repos into Git and host them on Github, although reports are that it's at best a clunky, somewhat awkward solution. And they shrewdly made sure that Github repos are fully accessible on the SVN client. 10. Syntax highlighting. Github users will be used to this as a standard, indispensable feature, but Bitbucket notably continues to lack it.	Fitur: 1. Pull requests and code reviews 2. Unlimited private repos 3. Branch comparison and commit history 4. Bitbucket Mac and Windows client called SourceTree; Android app called BitBeaker 5. Bitbucket for Enterprises, called Stash 6. Integration with tools like Jira, Crucible, Bamboo, Jenkins



Gambar 2. Mekanisme Pull dan Push pada Repositori GitHub

Setiap kelompok akan memiliki satu buah repositori GitHub yang akan digunakan sebagai tempat pengembangan aplikasi dari awal sampai akhir semester, aktivitas (commit) akan digunakan sebagai salah satu parameter penilaian individu. Repositori akan dibuat oleh ketua kelompok, setelah repositori dibuat makan ketua akan membuat repositori lokal (htdocs) dan kemudian di publish pada repositori GitHub. Setelah repositori lokal ketua sinkron dengan repositori awan, maka ketua berkewajiban untuk mengundang teman satu kelompok dan dosen ke dalam repo sebagai kontributor.

3. Alat dan Bahan

A. Kertas A4 (disediakan Politeknik Negeri Jember)
B. Kertas F4 (disediakan Politeknik Negeri Jember)
C. Folio Bergaris (disediakan Politeknik Negeri Jember)
D. BoardMarker (disediakan Politeknik Negeri Jember)
E. Komputer (disediakan Politeknik Negeri Jember dengan jumlah sesuai kapasitas lab)
F. XAMPP

G. Sublime Text 3
H. GitHub Desktop

4. Kegiatan Praktik

A. Mahasiswa mebentuk kelompok degan ketetuan masing-masing kelopok terdiri dari 4-5 orang
B. Mahasiswa berkumpul dengan anggota kelompoknya
C. Masing-masing mahasiswa menginstal XAMPP dan Sublime Text 3
D. Masing-masing anggota meng-Instal GiHub Desktop
   - Ketua membuat Repo. baru untuk project, disarankan langsung mengambil dari direktori lokal ketua (**htdocs**).
   - Masing-masing anggota kemudian dimaukan ke dalam Repo
   - Mengundang dosen pengampu dalam Repo
E. Masing-masing anggota me-refresh GitHub Desktop
F. Anggota kelompok melakukan Pull terhadap Repo yang telah dibuat ketua
G. Masing-masing anggota membuat folder di dalam Repo dengan ketentuan sebagai berikut: **NIM_Nama**

5. Refrensi *Website*

[1] https://www.codeigniter.com/
[2] https://codeigniter-id.github.io/user-guide/
[3] https://www.w3schools.com/
[4] https://www.tutorialspoint.com/codeigniter/
[5] https://www.malasngoding.com/category/codeigniter/
[6] https://www.codepolitan.com/

PRAKTIK 2: Mengimplementasikan User Iterface (HTML, CSS dan JS)

Matakuliah : Workshop SI berbasis WEB
Minggu Ke : 2 dan 3
Waktu : 2 x 100 menit
Tema : Mengimplemetasikan User Interface (HTML, CSS dan JS)

1. **Kompetensi Dasar**
   A. Mahasiswa mampu memahami dan mengimplmentasikan HTML
   B. Mahasiswa mampu memahami dan mengimplmentasikan CSS

2. **Dasar Teori**
   A. HTML
   a. Pengertian HTML

HTML adalah singkatan dari **Hypertext Markup Language**. Disebut **hypertext** karena di dalam HTML sebuah text biasa dapat berfungsi lain, kita dapat membuatnya menjadi **link** yang dapat berpindah dari satu halaman ke halaman lainnya hanya dengan meng-*klik* text tersebut. Kemampuan text inilah yang dinamakan **hypertext**, walaupun pada implementasinya nanti tidak hanya text yang dapat dijadikan **link**.

Disebut **Markup Language** karena bahasa **HTML** menggunakan tanda (*mark*), untuk menandai bagian-bagian dari text. Misalnya, text yang berada di antara tanda tertentu akan menjadi tebal, dan jika berada di antara tanda lainnya akan tampak besar. Tanda ini di kenal sebagai HTML **tag**.

Jika anda ingin melihat bagaimana sebenarnya HTML, silahkan klik kanan halaman ini, lalu pilih *View Page Source* (di *Browser Mozilla Firefox* atau *Google Chrome*). Akan tampil sebuah halaman baru yang merupakan kode **HTML** dari halaman ini.

HTML merupakan bahasa dasar pembuatan web. Disebut dasar karena dalam membuat web, jika hanya menggunakan **HTML** tampilan web terasa hambar. Terdapat banyak bahasa pemograman web yang ditujukan untuk memanipulasi kode **HTML**, seperti **JavaScript** dan **PHP**. Akan tetapi sebelum anda belajar **JavaScript** maupun **PHP**, memahami HTML merupakan hal yang paling awal.

HTML bukanlah bahasa pemograman (*programming language*), tetapi bahasa **markup** (*markup language*), hal ini terdengar sedikit aneh, tapi jika anda telah mengenal bahasa pemograman lain, dalam HTML tidak akan ditemukan struktur yang biasa di temukan dalam bahasa pemograman seperti *IF*, *LOOP*, maupun *variabel*. **HTML** hanya sebuah bahasa struktur yang fungsinya untuk menandai bagian-bagian dari sebuah halaman.

Selain **HTML**, dikenal juga **xHTML** yang merupakan singkatan dari *eXtensible Hypertext Markup Language*. **xHTML** merupakan versi lama dari **HTML** (sebelum era HTML5 seperti saat ini). xHTML menggunakan aturan penulisan yang lebih ketat. Jika anda menemukan artikel yang membahas xHTML, bisa disamakan dengan HTML, karena perbedaannya tidak terlalu banyak.

b. Pengertian Tag dalam HTML

Sebagai sebuah bahasa markup, **HTML** membutuhkan cara untuk memberitahu web browser untuk apa fungsi sebuah text. Apakah text itu ditulis sebagai sebuah *paragraf, list*, atau sebagai *link*? Dalam **HTML**, tanda ini dikenal dengan istilah **tag**.

Hampir semua tag di dalam **HTML** ditulis secara berpasangan, yakni **tag pembuka** dan **tag penutup**, dimana objek yang dikenai perintah tag berada di ***antara*** tag pembuka dan tag penutup ini. Objek disini dapat berupa text, gambar, maupun video. Penulisan tag berada di antara *dua kurung siku: "<"* dan *">"*.

Berikut adalah format dasar penulisan tag HTML:

```
<tag_pembuka>objek yang dikenai perintah tag</tag_penutup>
```

```
Sebagai contoh, perhatikan kode HTML berikut :
```

```
<p> Ini adalah sebuah paragraf </p>
```

- ** ** adalah tag pembuka, dalam contoh ini ***p*** adalah tag untuk *paragraf*.
- ** ** adalah tag penutup paragraf. Perbedaannya dengan tag pembuka terletak dari tanda *forward slash* (/)

Jika lupa memberikan penutup tag, umumnya browser akan "*memaafkan*" kesalahan ini dan tetap menampilkan hasilnya seolah-olah kita menuliskan tag penutup. Walaupun ini sepertinya memudahkan, tidak jarang malah bikin bingung.

Sebagai contoha lain, jika ingin membuat suatu text dalam sebuah paragraf yang di tulis tebal atau miring, di dalam **HTML** dapat ditulis sebagai berikut:

```
<p>Ini adalah sebuah paragraf. <i>Hanya kumpulan beberapa
kalimat</i>.
Paragraf ini terdiri dari <b>3 kalimat</b></p>.
```

Hasil dari kode **HTML** diatas, diterjemahkan oleh browser menjadi:
"Ini adalah sebuah paragraf. *Tidak lain dari kumpulan beberapa kalimat*. Paragraf ini terdiri dari **3 kalimat**."

Tag **<i>** pada kode **HTML** diatas memberikan perintah kepada browser untuk menampilkan text secara garis miring (i, singkatan dari ***italic***), dan tag **<b>** untuk menebalkan tulisan (b, singkatan dari ***bold***).

Terdapat pengecualian beberapa tag yang tidak berpasangan, seperti **
** untuk *break* (pindah baris) atau **<hr>** untuk *horizontal line* (garis horizontal). Tag ini dikenal juga dengan sebutan *self closing tag* atau *void tag*, untuk penulisannya bisa ditulis dengan **
**, maupun **
**.

HTML tidak case-sensitif, dalam artian penulisan ** ** dianggap sama dengan **<P>**. Pada awal kemunculan **HTML**, programmer web umumnya menggunakan huruf besar untuk seluruh tag agar membedakan dengan text yang berupa isi dari web. Namun varian **HTML**, **xHTML** mewajibkan huruf kecil untuk semua tag.

Dalam HTML5, aturan ini kembali tidak diharuskan. Akan tetapi kebiasaan web programmer saat ini adalah menggunakan huruf kecil untuk seluruh tag.

Pengertian Element dalam HTML

Element adalah isi dari tag yang berada diantara tag pembuka dan tag penutup, termasuk tag itu sendiri dan atribut yang dimikinya (jika ada). Sebagai contoh perhatikan kode HTML berikut:

```
<p> Ini adalah sebuah paragraf </p>
```

Pada contoh diatas, " Ini adalah sebuah paragraf " merupakan element p. Element tidak hanya berisi text, namun juga bisa tag lain.

Contoh:

```
<p> Ini adalah sebuah <em>paragraf</em> </p>
```

Dari contoh diatas,* Ini adalah sebuah paragraf * merupakan elemen **p**. Dalam pembahasan atau tutorial tentang HTML, tidak jarang istilah "tag" dan "element" saling dipertukarkan.

c. Pengertian Atribut dalam HTML

Atribut adalah informasi tambahan yang diberikan kepada *tag*. Informasi ini bisa berupa instruksi untuk warna dari text, besar huruf dari text, dll. Setiap atribut memiliki pasangan **nama** dan **nilai** (*value*), dan ditulis dengan **name="value"**. Value diapit tanda kutip, boleh menggunakan tanda kutip satu (') atau dua (").

Contoh kode HTML:

```
<a href="http://www.jti.polije.com">ini adalah sebuah link</a>
```

Pada kode HTML diatas, ***href="http://www.jti.polije.com"*** adalah **atribut. href** merupakan *nama dari atribut*, dan *http://www.jti.polije.com* adalah *value* atau nilai dari atribut tersebut.

Tidak semua tag membutuhkan atribut, tapi anda akan sering melihat sebuah tag dengan atribut, terutama atribut **id** dan **class** yang sering digunakan untuk manipulasi halaman web menggunakan CSS maupun JavaScript.

HTML memiliki banyak atribut yang beberapa diantaranya hanya cocok untuk tag tertentu saja. Sebagai contoh, atribut "***href***" diatas hanya digunakan untuk tag **<*a*>** saja (dan beberapa tag lain).

d. Struktur Dasar HTML

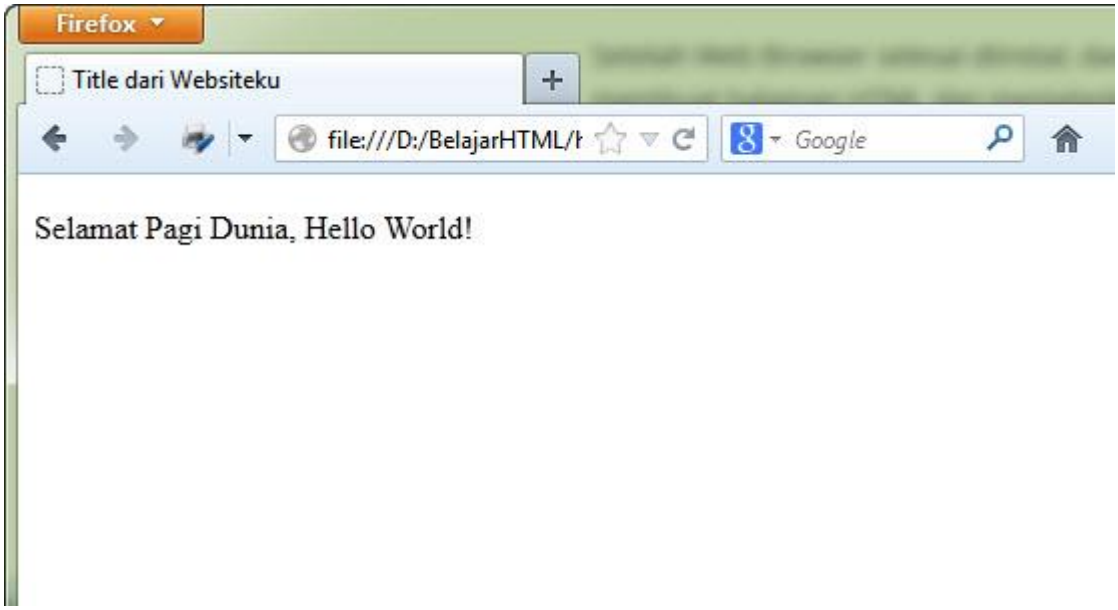
Setiap halaman **HTML** setidaknya memiliki struktur dasar yang terdiri dari : Tag **DTD** atau **DOCTYPE,** tag **html,** tag **head**, dan tag **body**. Inilah yang merupakan struktur paling dasar dari HTML, walaupun HTML tidak hanya berisi struktur tersebut.

Agar lebih mudah memahaminya, silahkan buka text editor (**Notepad++**), lalu ketikkan kode berikut ini:

Contoh struktur dasar HTML:

```
<!DOCTYPE html>
  <html>
  <head>
    <title>Title dari Websiteku</title>
  </head>
<body>
   <p>Selamat Pagi Dunia, Hello World!</p>
</body>
</html>
```

Save sebagai *halaman.html* dan jalankan file dengan cara double klik file tersebut, atau **klik kanan –> Open With –> Firefox.** Kita akan membahas tag-tag yang ditulis tersebut pada tutorial kali ini.



Pengertian DTD atau DOCTYPE

Tag paling awal dari contoh **HTML** diatas adalah **DTD** atau **DOCTYPE**. **DTD** adalah singkatan dari **Document Type Declaration**. Yang berfungsi untuk memberi tahu kepada web browser bahwa dokumen yang akan diproses adalah **HTML**.

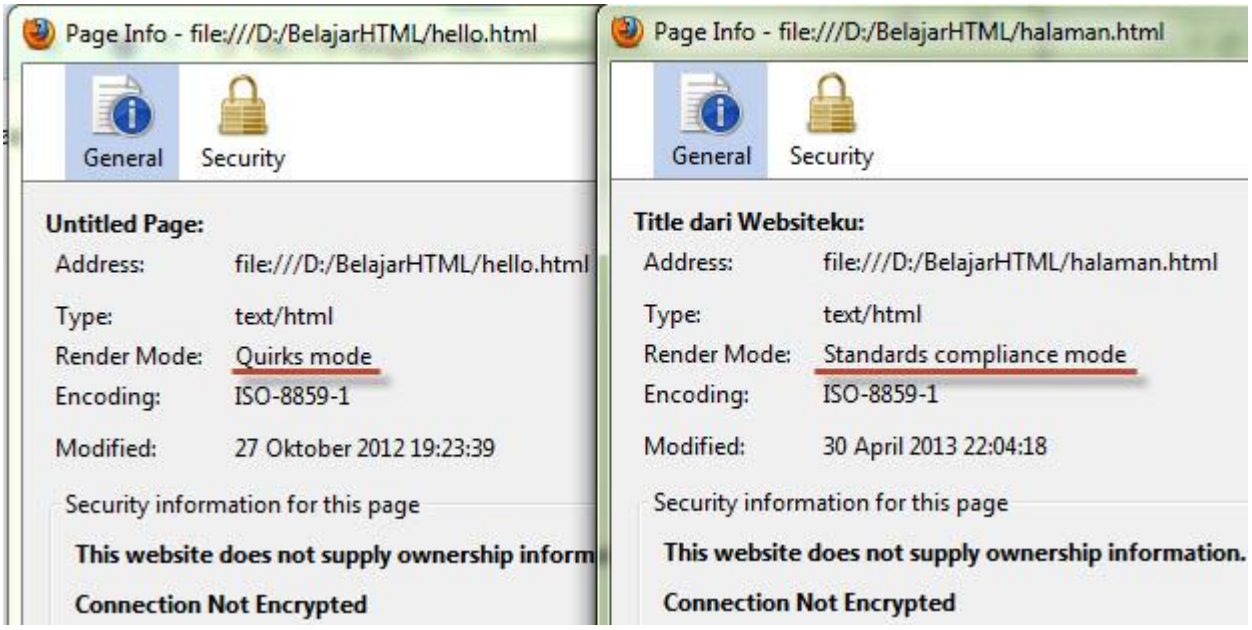
DTD memiliki banyak versi tergantung kepada versi **HTML** yang digunakan. Pada contoh diatas, saya menggunakan DTD versi **HTML 5**. Sebelum **HTML** 5, DTD terdiri dari text panjang yang hampir mustahil dihafal. Contohnya, DTD untuk **xHTML 1.0** adalah:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Jika kita tidak menuliskan DTD, browser akan tetap menampilkan dan memproses halaman web kita seperti tidak terjadi apa-apa. Namun browser sebenarnya menjalankan halaman **HTML** tersebut pada mode khusus yang disebut **quirk mode**.

Pada *quirk mode*, web browser menerjemahkan halaman web (terutama kode CSS) sedikit berbeda dari seharusnya. Ini karena web browser menganggap bahwa ketika DTD tidak ditemuka, halaman tersebut kemungkinan besar merupakan halaman web usang. Agar halaman ‘usang’ ini tetap tampil baik, web browser perlu menggunakan aturan-aturan yang berbeda, yakni: *quirk mode*.

Cara untuk mengetahui apakah web browser berjalan pada quirk mode atau standard mode lebih mudah jika menggunakan web browser Mozilla Firefox. Pada **Firefox**, klik kanan pada halaman web, lalu pilih **Page Info**. Pada bagian **Render Mode** akan terlihat apakah *quirk mode*, atau *standard mode*.



Tag <html>

Setelah DTD, tag berikutnya adalah tag <html>**.** Ini adalah tag pembuka dari keseluruhan halaman web. Semua kode **HTML** harus berada di dalam tag ini. Tag html dimulai dengan <html> dan diakhiri dengan </html>

Tag <head>

Elemen pada tag <head> umumnya akan berisi berbagai definisi halaman, seperti kode **CSS**, **JavaScript**, dan kode-kode lainnya yang tidak tampil di browser.

Tag <title> dalam contoh kita sebelumnya digunakan untuk menampilkan title dari sebuah halaman web. Title ini biasanya ditampilkan pada bagian paling atas web browser. Contohnya pada tampilan halaman.html, '*Title dari Websiteku*' akan ditampilkan pada tab browser.

Tag <body>

Tag <body> akan berisi semua elemen yang akan tampil dalam halaman web, seperti paragraf, tabel, link, gambar, dll. Tag body ini ditutup dengan </body>**.** Sebagian besar waktu kita dalam merancang web akan dihabiskan di dalam tag <body> ini.

Perhatikan bahwa setiap tag akan diakhiri dengan penutup tag. Termasuk <html> yang merupakan tag paling awal dari sebuah halaman web.

e. HTML5

HTML5 merupakan kelanjutan dari *HTML 4* yang tidak diupdate sejak tahun 1999. Dimana W3C sebagai badan yang membuat standar HTML lebih berminat mengembangkan ***XHTML*** daripada melanjutkan *HTML*.

Setelah beberapa tahun pengembangan, *XHTML* ternyata tidak sesuai dengan harapan. ***XHTML 2.0*** tidak kunjung selesai, sehingga beberapa programmer web

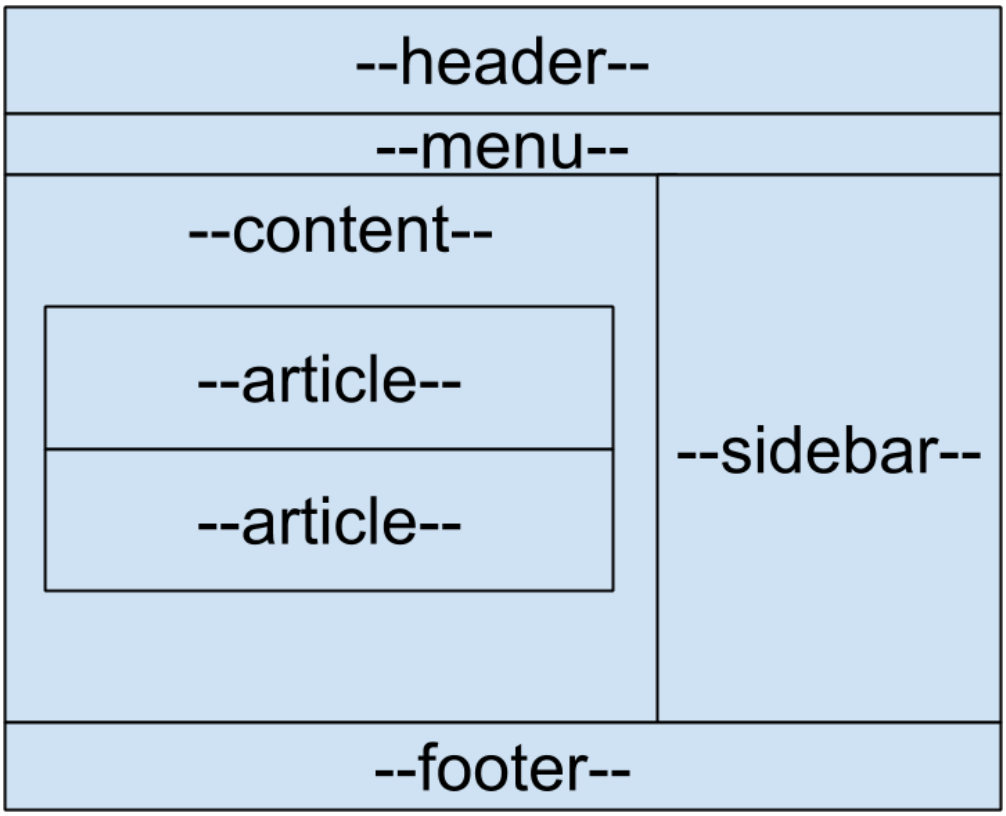
bergabung membentuk badan tersendiri: **WHATWG**. *WHATWG* kemudian mengembangkan format yang akan menjadi cikal bakal ***HTML5***. Singkat cerita, *W3C* meninggalkan *XHTML 2.0* dan ikut mengembangkan HTML5 bersama *WHATWG*.

Tepat pada tanggal 28 Oktober 2014 lalu, **W3C** telah resmi merampungkan standar ***HTML5*** (sumber resminya dapat dibaca pada http://www.w3.org/blog/news/archives/4167). Walaupun demikian, saat ini *HTML5* telah didukung hampir oleh semua web browser modern, sehingga sudah saatnya kita mulai membuat kode HTML menggunakan standar *HTML5*.

**Membuat Struktur Halaman HTML dengan Tag **

Sebelum era HTML5 yang memiliki tag untuk membuat struktur halaman yang lengkap, web developer umumnya menggunakan *tag * dengan atribut id atau class untuk memisahkan bagian-bagian struktur dalam halaman HTML, seperti *header*, *footer*, dan *sidebar*.

Sebagai contoh, berikut adalah gambar sederhana struktur sebuah website dengan 2 kolom:



Untuk membuat struktur diatas, berikut adalah kode HTMLnya:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Belajar HTML5</title>
</head>
```

```
<body>
  <div id="header">
    <h1>Judul Website</h1>
    <img src="logo_website.jpg" />
  </div>

  <div id="menu">
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
       <li><a href="#">Contact</a></li>
    </ul>
  </div>

  <div id="content">
    <div id="article_1">
      <div id="article_header_1">
        <h1>Judul Artikel 1</h1>
        <h2>Sub Judul Artikel 1</h2>
      </div>
        <p>...Ini adalah isi dari artikel 1...</p>
    </div>
     <div id="article_2">
      <div id="article_header_2">
        <h1>Judul Artikel 2</h1>
        <h2>Sub Judul Artikel 2</h2>
      </div>
        <p>...Ini adalah isi dari artikel 2...</p>
    </div>
  </div>

  <div id="sidebar">
    <h1>Artikel Terbaru</h1>
      <ul>
        <li><a href="#">Link 1</a></li>
        <li><a href="#">Link 2</a></li>
        <li><a href="#">Link 3</a></li>
      </ul>
    </div>

  <div id="footer">
    <p>Footer - Copyright Jti.polije 2014</p>
  </div>
</body>
</html>
```

Kode diatas sepenuhnya valid dan sangat sering digunakan hingga saat ini. Akan tetapi, **HTML 5** mencoba mengganti semua *tag * menjadi *semantic tag* yang lebih bermakna.

Semantic Tag untuk membuat Struktur web dengan HTML5

Seperti yang telah kita bahas pada tutorial sebelumnya tentang semantic tag, HTML5 mencoba menggantikan tag ‘*tanpa arti*‘ untuk sering digunakan untuk membuat struktur halaman web. Tag-tag yang bisa kita gunakan untuk keperluan ini adalah <header>, <nav>, <section>, <article>, <aside> dan <footer>.

Berikut adalah pembahasan beberapa tag HTML5 yang ditujukan untuk membuat struktur halaman:

Tag <header>

Tag <header> digunakan untuk bagian halaman web yang merupakan *header*. Tag ini bisa muncul lebih dari 1 kali, tergantung kebutuhan. Bagian atas web dimana kita meletakkan logo dan judul situs adalah tempat terbaik untuk tag *<header>*. Namun di bagian atas artikel dinama terdapat judul dan sub judul artikel juga bisa di ‘*bungkus*‘ dengan tag <header>.

Contoh penggunaan tag <header>:

```
<header>
  <h1>Judul Website</h1>
  <img src=”logo_website.png” />
</header>
```

Tag <nav>

Tag <nav> digunakan sebagai ‘*container*‘ dari menu *navigasi*. Sebaiknya digunakan untuk menu utama yang dirasa penting seperti pada bagian *header*.

```
Contoh penggunaan tag <nav>:
<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</nav>
```

Tag <section>

Tag <section> digunakan untuk memisahkan bagian-bagian dari struktur web. Tag ini bisa digunakan sebagai container untuk kumpulan artikel, gallery, atau bagian lain dari halaman web yang perlu pemisahan. Walaupun tag <section> terkesan ‘generik’, tetapi jika yang kita butuhkan hanya kontainer tanpa makna apa-apa, sebaiknya tetap menggunakan tag .
Bagian utama dimana kontent berada bisa ‘dibungkus’ menggunakan tag <section>. Dan jika halaman tersebut memiliki banyak bagian yang secara logika bisa dipisah, bisa menggunakan beberapa tag <section>.

Contoh penggunaan tag <section>:

```
<section>
  <h1>Judul 1</h1>
   <p>...Kumpulan dari konten...</p>
</section>
<section>
  <h1>Judul 1</h1>
  <p>...Kumpulan dari konten...</p>
</section>
```

Tag <main>

Tag <main> cocok digunakan untuk menandakan bagian utama dari sebuah halaman. Berbeda dari *tag* <section>, *tag* <main> umumnya hanya digunakan 1 kali untuk bagian paling penting, yang biasanya berupa konten/artikel utama.

Contoh penggunaan tag <main>:

```
<main>
  <h1>Judul Utama</h1>
  <p>...penjelasan...</p>
  <article>
    <h2>Judul Artikel 1</h2>
    <p>...penjelasan artikel 1...</p>
    <p>... </p>
    <p>... </p>
  </article>
  <article>
    <h2>Judul Artikel 2</h2>
    <p>...penjelasan artikel 2...</p>
    <p>... </p>
    <p>... </p>
  </article>
</main>
```

Tag <article>

Tag <article> bertujuan untuk menampung konten web yang merupakan artikel. Umumnya tag ini berada di dalam *tag* <section> atau <main>. Tag ini cocok sebagai container untuk artikel dalam sebuah blog.

Contoh penggunaan tag <article>:

```
<article>
 <h2>Judul Artikel</h2>
 <p>...penjelasan artikel...</p>
 <p>... </p>
 <p>... </p>
</article>
```

Tag <aside>

Tag <aside> bertujuan untuk menandai bagian web yang bukan berisi konten utama, tetapi memiliki kaitan dengan artikel yang saat ini ditampilkan. Bagian paling pas untuk tag <aside> adalah *sidebar*. Karena pada *sidebar* bisa terdiri dari berbagai konten yang tidak langsung berkaitan dengan konten utama seperti ’10

artikel terbaru', atau '5 komentar terbaru'. Selain untuk sidebar, tag <aside> juga bisa digunakan di dalam artikel untuk menandai bagian tambahan.

Contoh penggunaan tag <aside>:

```
<article>
  <h2>Judul Artikel 1</h2>
  <p>...penjelasan artikel 1...</p>
  <p>... </p>
  <p>... </p>
</article>
<aside>
  <p>...penjelasan tambahan...</p>
</aside>
```

Tag <footer>

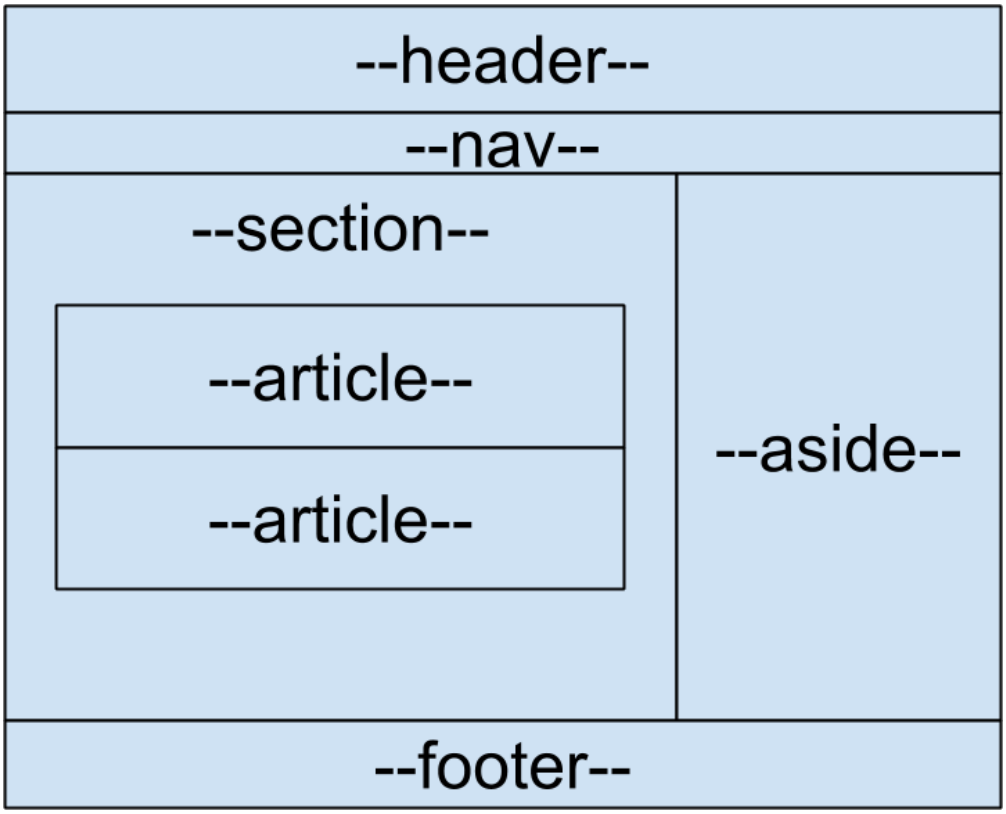
Tag <footer> biasannya digunakan pada bagian bawah halaman, dimana kita menampilkan beberapa informasi mengenai website. Walapun penggunaan paling jelas adalah untuk bagian *footer* halaman (meletakkan *copyright, about us*, dll), tag ini juga cocok digunakan pada bagian bawah artikel untuk menampung informasi tambahan seperti *'tentang penulis*' maupun link untuk share ke sosial media.

Contoh penggunaan tag <footer>:

```
<main>
  <h2>Judul Artikel 1</h2>
  <p>...penjelasan artikel 1...</p>
  <p>... </p>
  <p>... </p>
</main>
<footer>
  <p>...copyright 2014 jti.polije...</p>
</footer>
```

Membuat Struktur Halaman HTML dengan tag HTML5

Dengan menggunakan tag-tag HTML5 diatas, kita akan merevisi struktur web sebelumnya dengan menggunakan HTML5. Struktur yang akan kita buat sama dengan contoh pertama:



Dan berikut adalah kode HTML5 yang digunakan untuk membuat struktur tersebut:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Belajar HTML5</title>
</head>

<body>
  <header>
    <h1>Judul Website</h1>
    <img src="logo_website.jpg" />
  </header>

  <nav>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </nav>

  <section>
    <article>
      <header>
        <h1>Judul Artikel 1</h1>
        <h2>Sub Judul Artikel 1</h2>
      </header>
```

```
        <p>...Ini adalah isi dari artikel 1...</p>
    </article>
    <article>
      <header>
        <h1>Judul Artikel 2</h1>
        <h2>Sub Judul Artikel 2</h2>
      </header>
        <p>...Ini adalah isi dari artikel 2...</p>
    </article>
  </section>

  <aside>
    <h1>Artikel Terbaru</h1>
    <ul>
      <li><a href="#">Link 1</a></li>
      <li><a href="#">Link 2</a></li>
      <li><a href="#">Link 3</a></li>
    </ul>
  </aside>

  <footer>
    <p>Footer - Copyright Jti.polije 2014</p>
  </footer>
</body>
</html>
```

Perhatikan bahwa pada kode diatas saya menggunakan 6 semantic tag dari HTML5, yakni `<header>`, `<nav>`, `<section>`, `<artikel>`, `<aside>` dan `<footer>`.

f. Form pada HTML (tag form)

Pengertian tag `<form>`

Form biasanya digunakan untuk mengumpulkan data dari pengunjung web. Mulai dari form untuk login, form kontak, form untuk pendaftaran user, bahkan untuk mengirimkan data antar halaman web.Penggunaan form hanya menggunakan HTML saja tidak akan terlalu berguna. Form biasanya hanya berupa *interface* yang disediakan untuk mengumpulkan data dari user, dan akan diproses dengan bahasa pemograman web seperti **JavaScript** atau **PHP**, dan disimpan di dalam tabel **MySQL**.

Sebuah **form** dalam HTML harus berada di dalam **tag form**, yang diawali dengan `<form>` dan diakhiri dengan `</form>`. Tag form akan membutuhkan beberapa atribut untuk dapat berfungsi dengan seharusnya.

Atribut pertama adalah **action**, yang berfungsi untuk menjelaskan kemana data form akan dikirimkan. Biasanya nilai dari atribut **action** ini adalah alamat dari sebuah halaman **PHP** yang digunakan untuk memproses isi data form.

Atribut kedua adalah **method**, yang berfungsi untuk menjelaskan bagaimana data isian form akan dikirim oleh web browser. Nilai dari atribut **method** ini bisa berupa **get** atau **post**.

Perbedaan ***method get*** dan ***method post*** adalah, jika kita mengisi atribut **method** dengan **get** (dimana ini adalah nilai *default* seandainya atribut *method* tidak ditulis) maka isian form akan terlihat pada *url browser*. **Method get** ini biasanya digunakan untuk *query* pencarian. **Method post** biasanya digunakan untuk data yang lebih sensitif seperti yang berisi *password*, atau *registrasi user*. Data hasil form tidak akan terlihat pada browser.

Struktur dasar form akan terlihat sebagai berikut:

```
<form action="prosesdata.php" method="post">
   ...isi form...
</form>
```

Mengenal tag `<input>`

Tag input merupakan tag paling banyak digunakan di dalam **form** dan memiliki banyak bentuk, mulai dari isian text biasa, text *password*, *checkbox*, *radio*, sampai dengan tombol *submit*, semuanya dalam bentuk tag `<input>`.

Bentuk-bentuk dari keluarga tag input ini dibedakan berdasarkan **atribut type**:

- **<input type=”text” />** atau bisa juga **<input />** adalah *textbox* inputan biasa yang menerima input berupa text, contohnya digunakan untuk inputan *nama*, *username*, dan inputan yang berupa text pendek. **Input type text** ini juga bisa memiliki atribut **value** yang bisa diisi nilai tampilan awal dari text
- **<input type=”password” />** dalam tampilannya sama dengan **type text**, namun teks yang diinput tidak akan terlihat, akan berupa bintang atau bulatan. Biasanya hanya digunakan untuk inputan yang sensitif seperti *password*.
- **<input type=”checkbox” />** adalah inputan berupa *checkbox* yang dapat diceklist atau di centang oleh user. User dapat memilih atau tidak memilih checkbox ini. **Type checkbox** memiliki atribut **checked** yang jika ditulis atau diisi dengan nilai **checked**, akan membuat chexkbox langsung terpilih pada saat pertama kali halaman ditampilkan. Contoh inputan **checkbox** berupa *hobi*, yang oleh user dapat dipilih beberapa hobi.
- **<input type=”radio” />** mirip dengan **checkbox**, namun user hanya bisa memilih satu diantara pilihan group radio. **Type radio** ini berada dalam suatu grup dan user hanya bisa memilih salah satunya. Contoh inputan type radio adalah jenis kelamin.
- **<input type=”submit” />** akan menampilkan tombol untuk memproses form. Biasanya diletakkan pada baris terakhir dari form. **Atribut value** jika diisi akan membuat text tombol submit berubah sesuai inputan nilai **value**.

Perhatikan juga bahwa seperti tag ***<img>*** dan ***
, tag ***<input> juga merupakan tag yang berdiri sendiri dan tidak membutuhkan penutup tag.

Mengenal tag `<textarea>`

Tag textarea pada dasarnya sama dengan **input type text**, namun lebih besar dan dapat berisi banyak baris. Panjang dan banyak baris untuk text area di atur melalui **atribut rows** dan **cols**, atau melalui **CSS**.

Contoh penggunaan ***textarea*** adalah sebagai berikut:

```
<textarea rows="5" cols="20">
    Text yang diisi dapat mencapai banyak baris
</textarea>
```

Elemen yang berada diantara **tag textarea** akan ditampilkan sebagai text awal dari form.

Mengenal tag `<select>`

Tag select digunakan untuk inputan yang telah tersedia nilainya, dan user hanya dapat memilih dari nilai yang ada. **Tag select** digunakan bersama-sama dengan **tag option** untuk membuat box pilihan.

Contoh penggunaan ***tag select*** adalah sebagai berikut:

```
<select>
    <option>Pilihan 1</option>
    <option>Pilihan 2</option>
    <option value="pilihan ketiga">Pilihan 3</option>
</select>
```

Ketika form dikirim untuk diproses, nilai dari tag <***option***> akan dikirimkan. Nilai ini adalah berupa text diantara tag option, kecuali jika kita memberikan atribut ***value***. Jika atribut ***value*** berisi nilai, maka nilai *value*-lah yang akan dikirim. Ada atau tidaknya atribut ***value*** ini tidak akan tampak dalam tampilan form.

T**ag select** memiliki atribut **selected** yang dapat ditambahkan agar tag select berisi nilai awal. Contoh penggunaanya adalah sebagai berikut:

```
<select>
    <option>Pilihan 1</option>
    <option>Pilihan 2</option>
    <option value="pilihan ketiga" selected>Pilihan 3</option>
</select>
```

Mengenal Atribut: Name

Setiap tag inputan di dalam form harus ditambahkan *atribut* **name** agar dapat diproses oleh web server nantinya. Di dalam halaman proses (yang biasanya berupa bahasa **PHP** atau **ASP**), nilai dari atribut ***name*** inilah yang akan menjadi *variabel form*.

Contoh pemakaiannya adalah sebagai berikut:

```
<input type="text" name="username">
<input type="text" name="email">
```

Kedua input diatas akan tampak sama persis, namun pada saat pemrosesan data, masing-masing akan dibedakan menurut *atribut* **name**.

Akhirnya, Sebuah Form Utuh

Merangkum seluruh tag form HTML yang telah kita bahas diatas, maka saatnya untuk membuat sebuah **form HTML**. Silahkan buka *text editor*, dan tuliskan kode HTML berikut, lalu save sebagai **formulir.html**

Contoh penggunaan tag form:

```
<!DOCTYPE html>
<html>
<head>
   <title>Belajar Membuat Form </title>
</head>
<body>
<form action=" formulir.html" method="get">

Nama: <input type="text" name="nama" value="Nama Kamu" />
<br />

Password: <input type="password" name="password" />
<br />

Jenis Kelamin :
<input type="radio" name="jenis_kelamin" value="laki-laki"
checked />
Laki - Laki
<input type="radio" name="jenis_kelamin" value="perempuan" />
Perempuan
<br />

Hobi: <input type="checkbox" name="hobi_baca" /> Membaca Buku
      <input type="checkbox" name="hobi_nulis" checked />
Menulis
      <input type="checkbox" name="hobi_mancing" /> Memancing
<br />

Asal Kota:
 <select name="asal_kota" >
     <option value="Kota Jakarta"> Jakarta</option>
     <option>Bandung</option>
    <option value="Kota Semarang" selected>Semarang</option>
 </select>
<br />

Komentar Anda:
```

```
<textarea name="komentar" rows="5" cols="20">
Silahkan katakan isi hati anda
</textarea>
<br />

<input type="submit" value="Mulai Proses!" >

</form>
</body>
</html>
```

Terlepas dari tampilan yang kurang rapi, kita telah membuat sebuah form utuh (tampilan form dapat diubah dengan mudah menggunakan CSS). Perhatikan bahwa untuk ***atribut* target** saya mengisinya dengan: **formulir.html** dan ***atribut* method** *dengan nilai* ***get***, sehingga pada saat anda klik kombol ***mulai proses,*** perhatikan perubahan pada alamat *address bar browser*, akan tampil tambahan seperti berikut:

```
file:///D:/BelajarHTML/formulir.html?nama=Andi&password=rahasi
a&jenis_kelamin=laki-laki&hobi_nulis=on&asal_kota=Bandung
&komentar=Sudah+mahir+html
```

Jika diperhatikan dengan lebih lanjut, semua isian form tampak terlihat dari baris ini (karena **method form** kita set menjadi **get**) setiap nilai dibatasi dengan karakter **dan (&)** sedangkan spasi di ubah menjadi karakter **tambah (+)**

g. Cara Membuat link di HTML

Tujuan kata **Hypertext** dari **HTML** adalah membuat sebuah text yang ketika di-klik akan pindah ke halaman lainnya. **HTML** menggunakan tag **<a>** untuk keperluan ini.

Link ditulis dengan **<a>** yang merupakan singkatan cari *anchor* (jangkar). Setiap tag <a> setidaknya memiliki sebuah atribut **href**. Dimana **href** berisi alamat yang dituju (*href* adalah singkatan dari ***hypertext reference***).

Agar lebih jelas, kita akan lihat menggunakan contoh. Silahkan buka text editor dan buat kode seperti dibawah ini.

Contoh penggunaan tag link **<a>:**

```
<!DOCTYPE html>
<html>
<head>
  <title>Penggunaan Tag Link </title>
</head>
<body>
  <h1>Belajar Tag Link</h1>
  <p>Saya sedang belajar HTML dari
  <a href="http://www.jti.polije.com">Jti.polije</a></p>
</body>
</html>
```

Alamat Absolute dan Alamat Relatif

Pada contoh diatas kita menuliskan alamat link secara lengkap dengan bagian ***"http://www."***. Penulisan seperti ini disebut juga dengan ***external link***, yang berarti kita membuat link ke alamat lain di internet, atau lebih dikenal dengan: **alamat absolut**.

Alamat absolut adalah penulisan alamat file dengan menyertakan nama website, seperti: ***href="http://www.jti.polije.com/belajar_html.html"***, atau ***href="http://www.wikipedia.org"***.

Namun jika kita ingin membuat link di dalam situs yang sama, maka tidak perlu menyebutkan alamat lengkap tersebut. Tetapi cukup mencantumkan alamat file yang dituju ***relatif*** kepada file saat ini. Jenis alamat ini disebut **alamat relatif**.

Sebagai contoh *alamat relatif*, apabila kita ingin membuat link kepada file ***hello.html*** pada folder yang sama dengan file saat ini, maka atribut hrefnya, berisi: ***href=”hello.html”***. Berikut adalah kode HTMLnya:

Contoh penggunaan tag link <***a***> untuk alamat relatif:

```
<!DOCTYPE html>
<html>
<head>
  <title>Penggunaan Tag Link </title>
</head>
<body>
  <h1>Belajar Tag Link</h1>
  <p>Halaman HTML pertama saya adalah <a
href="hello.html">Hello</a></p>
</body>
</html>
```

Alamat absolute ditulis dengan lengkap, seperti **http://www.jti.polije.com**, atau jika kita merujuk kepada halaman tertentu, menjadi **http://www.jti.polije.com/nama_halaman.html**.

Alamat relatif maksudnya adalah relatif kepada file tempat kita memanggil link ini. Seandainya nama file kita adalah **belajar_link.html**, dan dalam folder yang sama terdapat halaman **belajar_list.html,** kita dapat menggunakan **href=”belajar_list.html”** untuk membuat link ke halaman **belajar_list.html** .

Jika file tersebut berada di dalam folder **lagi_belajar**, maka alamat relatifnya menjadi **href=”lagi_belajar/belajar_list.html”**. Namun bagaimana jika halaman tersebut berada 2 tingkat di luar folder saat ini? Kita dapat menggunakan **href=”../../belajar_list.html”**, untuk naik 2 folder diatasnya.

Atribut tag `<a>: target`

Atribut penting lainnya dari tag <a> adalah **target**. Atribut ini digunakan untuk menentukan apakah link yang dituju terbuka di jendela browser saat ini, atau terbuka di jendela baru.

Secara *default*, setiap link yang kita tulis akan terbuka pada jendela yang sama (menimpa halaman web saat ini). Tetapi apabila kita ingin halaman tersebut terbuka pada tab baru, gunakan atribut **target="_blank".**

Contoh penggunaan tag link <**a**> dengan atribut **target**:

```
<!DOCTYPE html>
<html>
<head>
<title>Penggunaan Tag Link </title>
</head>
<body>
<h1>Belajar Tag Link</h1>
<p>Saya sedang belajar html dari <a
href="http://www.jti.polije.com"
target="_blank">jti.polije</a> dan akan terbuka pada tab
baru</p>
</body>
</html>
```

Jika kita men-klik link tersebut, maka halaman jti.polije.com akan terbuka di tab baru, dan tidak menimpa tab saat ini.

B. CSS

a. Pengertian CSS

Dalam bahasa bakunya, seperti di kutip dari wikipedia, **CSS adalah** "*kumpulan kode yang digunakan untuk mendefenisikan desain dari bahasa markup" ,* dimana bahasa markup ini salah satunya adalah **HTML**.

Untuk pengertian bebasnya, **CSS adalah** kumpulan kode program yang digunakan untuk mendesain atau mempercantik tampilan halaman HTML. Dengan CSS kita bisa mengubah desain dari text, warna, gambar dan latar belakang dari (hampir) semua kode **tag HTML**.

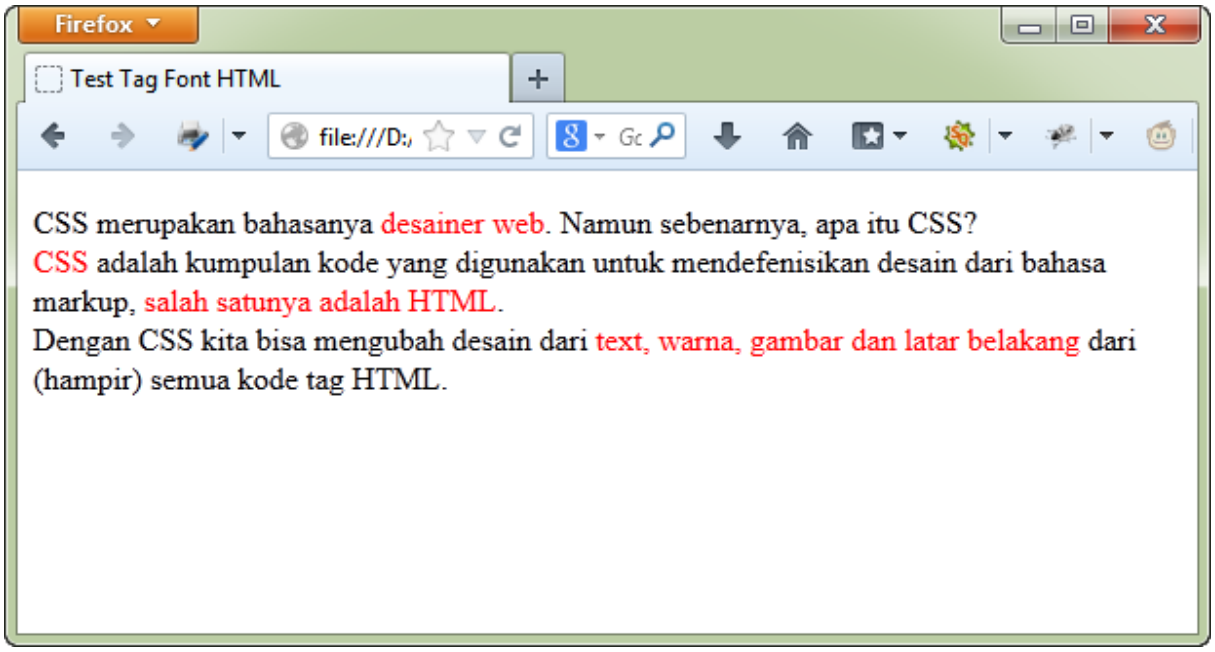
CSS biasanya selalu dikaitkan dengan HTML, karena keduanya memang saling melengkapi. HTML ditujukan untuk membuat **struktur**, atau konten dari halaman web. Sedangkan CSS digunakan untuk **tampilan** dari halaman web tersebut. Istilahnya, "*HTML for content, CSS for Presentation"*.

Fungsi dan Kegunaan CSS

Awal mula diperlukannya **CSS** dikarenakan kebutuhan akan halaman web yang semakin kompleks. Pada awal kemunculan HTML, kita bisa membuat suatu paragraf bewarna merah dengan menulis langsung kode tersebut didalam tag HTML, atau membuat latar belakang sebuah halaman dengan warna biru. Contoh kode HTML untuk hal itu adalah sebagai berikut:

```
<!DOCTYPE html>
<html>
```

```
<head>
      <title>Test Tag Font HTML</title>
</head>

<body>
  <p>
    CSS merupakan bahasanya <font color="red">desainer
web</font>.
    Namun sebenarnya, apa itu CSS?
  <br />
    <font color="red">CSS </font> adalah kumpulan kode yang
digunakan
    untuk mendefenisikan desain dari bahasa markup,
    <font color="red">salah satunya adalah HTML</font>.
  <br />
    Dengan CSS kita bisa mengubah desain dari
    <font color="red">text, warna, gambar dan latar
belakang</font>
    dari (hampir) semua kode tag HTML.
  </p>
</body>
</html>
```

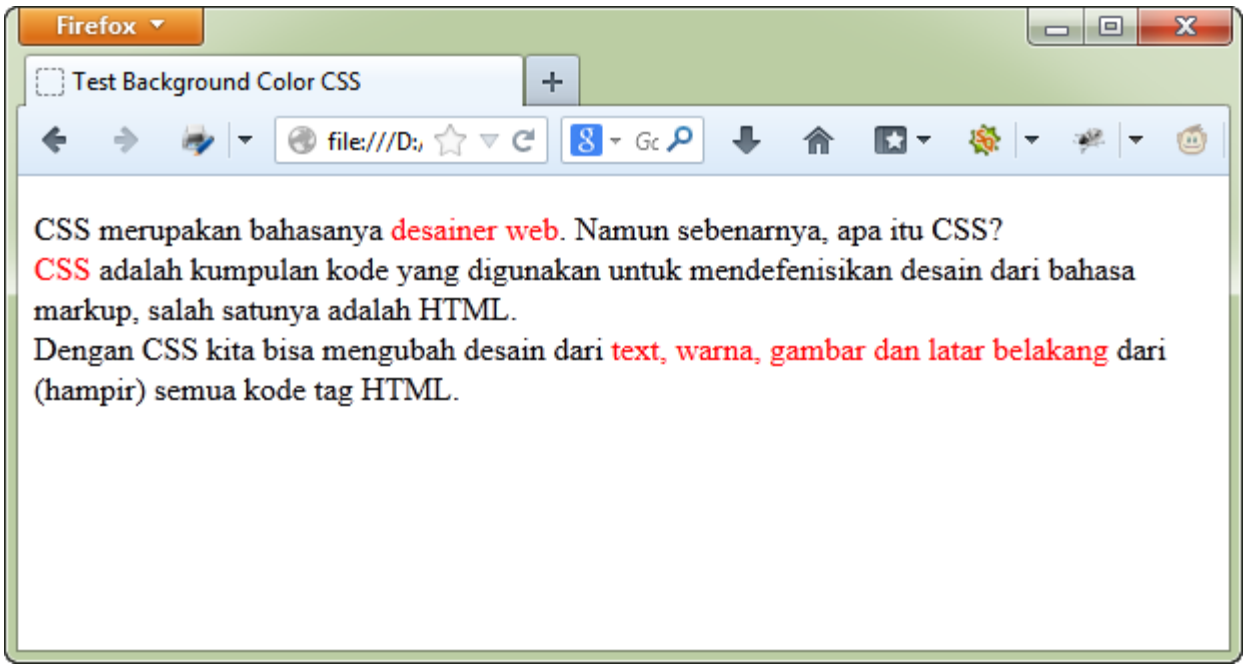


Saya menggunakan **tag <font>** untuk membuat beberapa kata di dalam paragraf tersebut berwarna merah. Hal ini tidak salah, dan semuanya berjalan sesuai keinginan. Untuk sebuah artikel yang memiliki 5 paragraf, kita tinggal copy-paste tag **<font color=”red”>** pada kata-kata tertentu.

Namun setelah website tersebut memiliki katakanlah 50 artikel seperti diatas, dan karena sesuatu hal anda ingin merubah seluruh text merah tadi menjadi biru, maka akan dibutuhkan waktu yang lama untuk mengubahnya satu persatu, halaman per halaman.

Dalam kondisi inilah CSS mencoba *'memisahkan'* **tampilan** dari **konten**. Untuk paragraf yang sama, berikut kode HTML bila ditambahkan kode CSS:

```
<!DOCTYPE html>
<html>
<head>
   <title>Test Background Color CSS</title>
   <style type="text/css">
     .warna {
       color: red;
     }
   </style>
</head>

<body>
    <p>
      CSS merupakan bahasanya <span class=warna>desainer
web</span>.
      Namun sebenarnya, apa itu CSS?
    <br />
      <span class=warna>CSS </span>adalah kumpulan kode
      yang digunakan untuk mendefenisikan desain dari bahasa
markup,
      <span class=warna>salah satunya adalah HTML</span>.
    <br />
      Dengan CSS kita bisa mengubah desain dari
      <span class=warna>text, warna, gambar dan latar
belakang</span>
      dari (hampir) semua kode tag HTML.
    </p>
</body>
</html>
```



Dalam contoh **CSS** diatas, saya mengubah **tag <font>** menjadi **tag **. **Tag ** sendiri merupakan tag yang *tidak bermakna*, namun bisa di kostumasi menggunakan CSS. **Tag span** saya tambahkan dengan atribut atribut **class="warna".** Atribut **class** berguna untuk memasukkan kode CSS pada **tag <style>** di bagian head.

Jika kita ingin merubah seluruh warna menjadi biru, maka tinggal mengubah isi dari CSS **color: red** menjadi **color:blue,** dan seluruh tag yang memiliki **class="warna"** akan otomatis berubah menjadi biru.CSS memungkinkan kita merubah tampilan dari halaman, tanpa mengubah isi dari halaman.

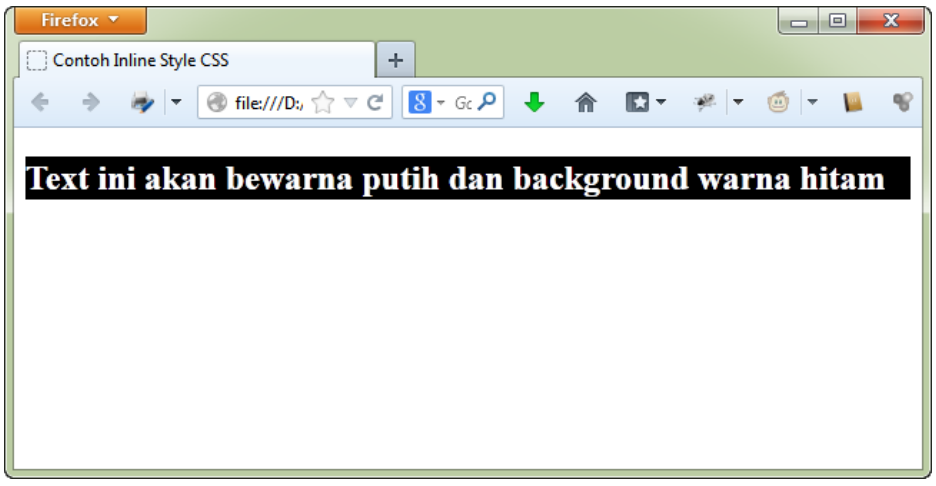
b. Integrasi CSS pada HTML

HTML pada dasarnya adalah kumpulan dari tag-tag yang disusun sehingga memiliki bagian-bagian tertentu, seperti *paragraf*, *list*, *tabel* dan sebagainya. CSS digunakan untuk mendesain tag-tag HTML ini. Secara garis besar, terdapat 3 cara menginput kode CSS ke dalam HTML, yaitu metode **Inline Style**, **Internal Style Sheets**, dan **External Style Sheets**.

Metode Inline Style

Metode Inline Style adalah cara menginput kode CSS langsung ke dalam tag HTML dengan menggunakan atribut **style**, contoh penggunaan *Metode Inline Style CSS* adalah sebagai berikut:

```
<!DOCTYPE html>
<html>
<head>
     <title>Contoh Inline Style CSS</title>
</head>
   <body>
      <h2 style="background-color:black; color:white" >
         Text ini akan bewarna putih dan background warna
hitam
      </h2>
   </body>
</html>
```



Dalam kode diatas, saya menyisipkan atribut **style** pada tag **<h2>**, nilai dari atribut **style** ini adalah kode CSS yang ingin diterapkan.

Penggunaan tag CSS seperti ini walaupun praktis, namun tidak disarankan, karena kode CSS langsung tergabung dengan HTML, dan tidak memenuhi tujuan dibuatnya CSS agar desain terpisah dengan konten.

Metode Internal Style Sheets

Metode Internal Style Sheets, atau disebut juga **Embedded Style Sheets** digunakan untuk memisahkan **kode CSS** dari **tag HTML** namun tetap dalam satu halaman HTML. Atribut **style** yang sebelumnya berada di dalam tag, dikumpulkan pada pada sebuah **tag <style>.** Tag style ini harus berada pada bagian <head> dari halaman HTML.

Contoh penggunaan **motode internal style sheets CSS:**

```
<!DOCTYPE html>
<html>
<head>
    <title>Contoh Inline Style CSS</title>
    <style type="text/css">
        h2 {
        background-color:black;
        color:white;
        }
    </style>
</head>
<body>
    <h2>
        Text ini akan bewarna putih dan background warna hitam
    </h2>
</body>
</html>
```

Contoh metode **internal style sheets** diatas sudah jauh lebih baik daripada **inline style**, karena kita sudah memisahkan CSS dari HTML. Seluruh kode CSS akan berada pada tag head dari HTML.

Namun kekurangan menggunakan **internal style sheets,** jika kita memiliki beberapa halaman dengan style yang sama, maka kita harus membuat kode CSS pada masing-masing halaman tersebut. Hal ini dapat diatasi dengan menggunakan metode **external style sheets**.

Metode External Style Sheets

Kekurangan dari metode **internal style sheets** sebelumnya adalah jika ingin membuat beberapa halaman dengan tampilan yang sama, maka setiap halaman akan memiliki kode CSS yang sama.

Metode External Style Sheets digunakan untuk '*mengangkat'* kode CSS tersebut kedalam sebuah file tersendiri yang terpisah sepenuhnya dari halaman HTML. Setiap halaman yang membutuhkan kode CSS, tinggal '*memanggil'* file CSS tersebut.

Masih menggunakan contoh yang sama dengan **internal style sheets**, tahap pertama kita akan pindahkan isi dari tag **<style>** ke sebuah halaman baru, dan savelah sebagai **belajar.css**

Isi dari file **belajar.css :**

```
h2 {
background-color:black;
color:white;
}
```

Pastikan bahwa akhiran dari file tersebut adalah **.css** dan untuk keperluan contoh kali ini, savelah pada folder yang sama dengan halaman HTML kita.
Kembali kehalaman HTML, CSS menyediakan 2 cara untuk menginput Kode CSS tersebut ke halaman HTML, yang pertama adalah menggunakan **@import**

Contoh penggunaan `@import CSS`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Contoh Inline Style CSS</title>
    <style type="text/css">
        @import url(belajar.css);
    </style>
</head>
<body>
    <h2>
        Text ini akan bewarna putih dan background warna hitam
    </h2>
</body>
</html>
```

Untuk **metode @import** external style sheets ini, kita menyisipkan **@import url(belajar.css);** pada **tag <style>**. Alamat pada bagian **url** bisa berupa alaman *relatif* (seperti: folderku/belajar.css) maupun *absolut* (seperti www.jti.polije.com/belajar.css).
Cara input kedua **external style sheets**, adalah menggunakan **tag <link>**. Berikut contohnya:

```
<!DOCTYPE html>
<html>
<head>
    <title>Contoh Inline Style CSS</title>
    <link rel="stylesheet" type="text/css" href="belajar.css">
```

```
</head>
<body>
    <h2>
        Text ini akan bewarna putih dan background warna hitam
    </h2>
</body>
</html>
```

Pada metode link **external style sheets** ini, kita menggunakan atribut **href** pada **tag <link>**, yang akan berisi alamat dari halaman CSS, dalam hal ini **belajar.css**

Dari ketiga jenis cara input CSS ke dalam halaman HTML, yang **paling direkomendasikan** adalah *metode external style sheets*, baik menggunakan **@import** maupun dengan **tag <link>**. Karena dengan menggunakan kode CSS yang dipisahkan, seluruh halaman web dapat menggunakan file CSS yang sama, dan jika kita ingin mengubah seluruh tampilan halaman website, kita hanya butuh mengubah 1 file CSS saja.

c. Pengertian Selector CSS

Karena kode CSS digunakan untuk mengubah/memanipulasi tampilan dari tag HTML, CSS membutuhkan suatu cara untuk '*mengaitkan'* atau **menghubungkan** kode CSS dengan tag HTML yang sesuai. Hal inilah yang dimaksud dengan **Selector** dalam CSS.

Sesuai dengan namanya, selector digunakan untuk mencari bagian web yang ingin dimanipulasi atau yang ingin di-style. Misalnya : "*cari seluruh tag <p>",* atau "*cari seluruh tag HTML yang memiliki atribut class="warning""* atau *"cari seluruh link yang ada di dalam tag <p>"*.

Selector paling dasar dari CSS adalah tag dari HTML itu sendiri, misalnya: tag *p, i, h1, li,* dll. Selector didalam CSS dapat menjadi kompleks tergantung kebutuhannya. Mengenai selector, akan kita bahas secara lebih detail dalam tutorial-turorial CSS selanjutnya.

Pengertian Property CSS

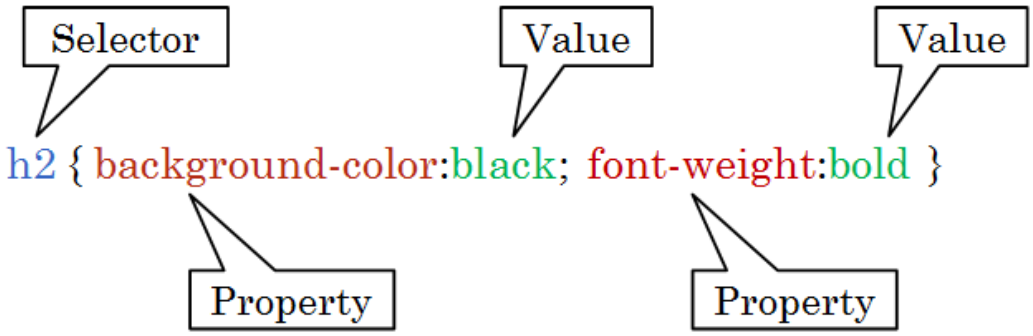
Property CSS adalah *jenis style*, atau elemen apa yang akan diubah dari sebuah tag HTML. CSS memiliki puluhan **property** yang dapat digunakan agar menampilkan hasil akhir yang kita inginkan. Hampir semua **property** dalam CSS dapat dipakai untuk seluruh **selector**.

Jika selector digunakan misalnya untuk "*mencari seluruh tag <p>*", maka property adalah *"efek apa yang ingin dimanipulasi dari tag p tersebut*", seperti ukuran text, warna text, jenis fontnya, dll.

Pengertian Value CSS

Value CSS adalah nilai dari **property**. Misalkan untuk property **background-color** yang digunakan untuk mengubah warna latar belakang dari sebuah selector, **value** atau nilainya dapat berupa **red, blue, black**, atau **white**.

Untuk lebih jelasnya tentang *selector*, *property* dan *value* pada **CSS**, dapat dilihat pada gambar dibawah ini:



Selector, Property dan Value ini adalah elemen inti dari CSS, 90% tutorial yang akan kita pelajari mengenai CSS akan membahas tentang ketiga aspek ini.

d. Aturan dan Cara Penulisan Kode CSS

Untuk lebih mudah memahami aturan dan cara penulisan kode CSS, dibawah ini adalah contoh kode HTML + CSS sederhana yang akan kita bahas secara lebih mendalam:

```
<!DOCTYPE html>
<html>
<head>
<title>Belajar Penulisan CSS</title>
<style type="text/css">
body {
     font-size: 14px;
     color: navy;
     }
p {
    font-family: calibri, helvetica, sans-serif;
  }
h1, h2 {
        text-decoration: underline;
        font-size: 23px;
        color: green;
       }
</style>

</head>
<body>
   <h1>Belajar Aturan dan Cara Penulisan Kode CSS</h1>

   <h2>Lorem ipsum dolor sit amet</h2>
   <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
   Nulla erat dolor, ullamcorper in ultricies eget,
   fermentum rhoncus leo. Curabitur eu mi vitae metus
   posuere laoreet.</p>
```

```
    <p>Eam facilis omittantur at, usu efficiantur neglegentur
    delicatissimi et, in per vero splendide persequeris.
    Semper persius his te, ea mea omnium aliquip alienum,
    in gloriatur vituperata has. An per dicat clita oporteat,
    explicari deterruisset ex per. Liber tibique ullamcorper
    ei duo, at hinc civibus oporteat his.</p>

    <p>Nam verear constituto an, eu latine bonorum euripidis
    vim. Quidam maiorum interesset pri id, usu vero saepe
    graeci ea. Prompta nominati oportere te usu.
    Ad eam nulla accusamus.</p>

    <h2>Lorem ipsum dolor sit amet</h2>
    <p>Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Nulla erat dolor, ullamcorper in ultricies
eget,
    fermentum rhoncus leo. Curabitur eu mi vitae
    metus posuere laoreet.</p>

</body>
</html>
```

Kode HTML di atas memiliki kode CSS pada bagian head yang diinput menggunakan motode *Metode Internal Style Sheets*. Kita akan fokus pada bagian CSS saja:

```
body {
     font-size: 14px;
     color: navy;
     }
p {
    font-family: calibri, helvetica, sans-serif;
  }
h1, h2 {
        text-decoration: underline;
        font-size: 23px;
        color: green;
       }
```

Seperti yang dapat dilihat dari contoh diatas, aturan penulisan CSS adalah sebagai berikut:

- **Selector** di tempatkan pada awal penulisan CSS. Dalam contoh diatas, **body, p, h1** dan **h2** adalah selector. Khusus untuk selector yang lebih dari 1 (seperti pada contoh selector terakhir: h1,h2) untuk memisahkan kedua selector tersebut, digunakan tanda koma (,)
- Setelah penulisan selector, seluruh isi dari property dan valuenya (nilainya), berada di antara kurung kurawal "**{**" dan "**}**".
- Diantara property satu dengan yang lainnya, dipisahkan dengan tanda titik koma (**;**)
- Diantara property dengan value (nilai), dipisahkan dengan anda titik dua (**:**).
- Untuk property yang memiliki 2 kata , spasi diantaranya digantikan dengan tanda hubung (-), contohnya: **background-color** dan **border-left**.
- Untuk properti yang berada pada baris terakhir, kita boleh mengabaikan tanda '**;**' sebagai tanda tutup, tetapi disarankan agar tetap menggunakan tanda '**;**', karena bisa saja kita lupa menuliskan tanda titik koma pada saat menambahkan properti lainnya. Contoh property tanpa tanda '**;**' dapat di lihat pada contoh "**text-decoration: underline**".

e. Selector Dasar CSS

CSS memiliki jenis selector yang bervariasi, bahkan sangat beragam tergantung kebutuhan kita untuk mendesain halaman web. Dalam tutorial ini akan membahas 5 jenis selector dasar di dalam CSS. Selector CSS tidak hanya 5 jenis ini, namun dalam kebanyakan kasus sudah mencukupi untuk membuat sebuah halaman web HTML+CSS.

Selector adalah sebuah pola (***pattern***) yang digunakan untuk '*mencari'* suatu tag di dalam HTML. Analogi untuk *selector*, misalnya: *mencari semua tag p*, atau *mencari seluruh tag h1 yang memiliki atribut class=judul*. CSS memiliki banyak selector, kita akan membahasnya satu persatu:

Universal Selector

Universal selector hanya ada 1 di dalam CSS, yaitu tanda bintang "*". Selector ini bertujuan untuk '*mencari*' semua tag yang ada.

Contoh **Universal Selector** CSS:

```
* {
   color: blue;
   background-color: white;
  }
```

Kode CSS diatas bermaksud untuk membuat seluruh tag HTML berwarna biru, dan background berwarna putih.

Element Type Selector

Element Type Selector atau **Tag Selector** adalah istilah untuk selector yang nilainya merupakan tag HTML itu sendiri. Setiap tag HTML bisa digunakan sebagai selector, dan seluruh tag tersebut akan *ditangkap* oleh selector ini.

Contoh Element Type Selector CSS:

```
h1 {
   text-decoration: underline;
   }

p {
  font-size:14px;
  }
```

Contoh kode CSS diatas akan membuat semua tag **<h1>** akan bergaris bawah, dan seluruh tag ** ** akan berukuran 14pixel.

Efek dari element type selector adalah dari awal tag, sampai akhir tag. Jika didalam tag terdapat tag <i>, maka tag tersebut juga akan berukuran 14 pixel, sampai ditemui tag penutup .

Class Selector

Class Selector merupakan salah satu selector yang paling umum dan paling sering digunakan. *Class Selector* akan '*mencari'* seluruh tag yang memiliki atribut *class* dengan *nilai* yang sesuai.

Untuk penggunaan **Class Selector**, kita harus memiliki tag HTML yang mempunyai atribut *class*. Contohnya:

```
<p class="paragraf_pertama"> Ini adalah sebuah paragraf
pertama</p>
<h1 class="judul">Judul Artikel</h1>
<h2 class="judul penting berwarna">Sub Judul Artikel<h2>
```

Perhatikan bahwa untuk semua tag diatas, kita menambahkan atribut *class* dengan nilainya adalah nama dari kelas itu sendiri. **Sebuah nama class dapat dimiliki oleh lebih dari 1 tag, dan dalam sebuah tag dapat memiliki lebih dari 1 class**.

Contohnya dalam baris terakhir pada contoh diatas,tag **h2** memiliki atribut *class=”judul penting berwarna”.* Tag ini teridiri dari 3 class, yaitu *judul*, *penting*, dan class *berwarna.*

Sedangkan untuk kode CSS Class Selector adalah sebagai berikut:

```
.paragraf_pertama {
                  color: red;
                  }

.judul {
       font-size:20px;
       }
.penting {
       color:red;
       font-size: 1em;
       }
```

Untuk menggunakan **class selector**, di dalam CSS kita menggunakan **tanda titik** sebelum nama dari **class**.

Untuk contoh kita, seluruh class yang memiliki nilai “*paragraf_pertama*”, warna text akan menjadi merah. Dan seluruh class *judul* akan memiliki font 20 pixel.

ID Selector

ID Selector bersama-sama dengan **class selector** merupakan **selector** paling umum dan juga sering dipakai (walau tidak sesering *class selector*). Penggunaan *ID selector* hampir sama dengan class selector, dengan perbedaan jika pada Class Selector kita menggunakan atribut **class** untuk tag HTML, untuk **ID selector**, kita menggunakan **atribut id**.

Contoh penggunaan **atribut id** pada tag HTML

```
<p id=”paragraf_pembuka”> Ini adalah sebuah paragraf
pembuka</p>

<h1 id=”judul_utama”>Judul Artikel</h1>

<h2 id=”sub_judul”>Sub Judul Artikel<h2>
```

Atribut id selain untuk selector CSS, juga berperan sebagai kode unik untuk masing-masing tag (terutama dipakai untuk kode JavaScript). Karena hal tersebut, id

yang digunakan harus unik dan tidak boleh sama. Dengan kata lain, **id hanya bisa digunakan satu kali dalam sebuah halaman web dan tidak boleh sama**.

Contoh penggunaan id selector kode CSS Class Selector adalah sebagai berikut:

```
#paragraf_ pembuka {
                   color: red;
                   }

#judul utama {
             font-size:20px
             }
```

Di dalam kode CSS, kita menggunakan **tanda pagar** "#" sebagai penanda bahwa kita mencari tag yang memiliki **id** tersebut.

Attribute Selector

Selector dasar terakhir kita adalah **attribute selector.** Selector ini sedikit lebih *advanced* dibandingkan dengan selector-selector sebelumnya. Atribut Selector ini digunakan untuk mencari seluruh tag yang memiliki atribut yang dituliskan.

Contoh penggunaan Attribute Selector kode CSS adalah sebagai berikut:

```
[href] {
       font-size:20px ;
       }

[type="submit"] {
                width:30px;
                }
```

Seperti yang dapat dilihat dari contoh diatas, setiap **atribut selector** harus berada diantara tanda kurung siku "[" dan "]".

[href] akan cocok dengan seluruh tag yang memiliki atribut **href**, apapun nilai dari **href** (href biasanya terdapat pada tag **<a>**). Untuk contoh **[type="submit"]** akan cocok dengan tag yang memiliki *atribut type* dengan nilai *submit*, yang dalam hal ini adalah tombol *submit* dalam *form*.

Walaupun memiliki kemampuan mencari tag yang sangat spesifik, namun **atribut selector** ini tidak terlalu sering digunakan.

Selain kelima selector dasar diatas, CSS masih memiliki selector yang lebih '*jauh*' dalam memilih tag yang akan dimanipulasi, salah satu contohnya, seperti **pseudo selector** yang digunakan untuk tiap event dari link, atau dikenal dengan efek **mouseover**, yaitu kita mencari kondisi pada saat mouse berada di atas tag tertentu. **Pseudo Selector** ini akan kita bahas pada lain kesempatan.

Itulah 5 selector dasar dalam CSS, yang selain berdiri sendiri, dapat juga digabungkan dengan selector lainnya untuk keperluan yang lebih khusus.

f. Cara Penggunaan Selector CSS

Selector tersebut tidak hanya berdiri sendiri, namun dapat digabung menjadi sebuah selector yang lebih spesifik. CSS memungkinkan kita untuk menggabungkan beberapa jenis **selector** menjadi sebuah selector gabungan, misalnya kita butuh selector untuk mencari *seluruh* ***tag*** **<*em*>** *yang berapa di dalam* ***tag*** **<h1>**, atau *seluruh* ***tag*** **<*h1*>** *yang didalamnya terdapat* ***tag*** **<*a*>** *dengan* ***class "judul"***.

Agar mudah memahami cara penggabungan selector CSS ini kita akan mempersiapkan sebuah halaman HTML *dummy* untuk bahan pembelajaran. Silahkan ketik kode HTML berikut, dan save sebagai **belajar_css.html**

```
<!DOCTYPE html>
<html>
    <head>
    <title>Belajar CSS</title>

    <style type="text/css">
    </style>

</head>
<body>
  <h2 class="judul">Belajar CSS</h2>
    <p> Ini <strong>adalah</strong> sebuah <em>paragraf</em>
</p>

    <p id="belajar_html" class="paragraf" >Saya sedang belajar
HTML dari
    <a href="http://www.jti.polije.com"
target="_blank">Jti.polije</a>
    akan terbuka pada tab baru</p>

    <p id="belajar_css" class="paragraf">
    Saya sedang belajar CSS, mohon jangan diganggu</p>

    <br />

    <h3 class="judul subheader">Daftar Belanjaan</h3>
    <ol>
          <li><em>Minyak Goreng</em></li>
          <li>Sabun Mandi</li>
          <li>Deterjen</li>
          <li>Shampoo</li>
          <li>Obat Nyamuk</li>
    </ol>
 </body>
</html>
```

Halaman HTML diatas merupakan halaman HTML sederhana yang terdiri dari tag header (<**h2**>), paragraf(<**p**>), dan list (<**ol**>). Jika anda belum memahami tag-tag HTML diatas..

Perhatikan juga pada bagian <head> dari *belajar_css.html* tersebut saya juga menyediakan tag **<style>** yang sebentar lagi akan kita isi dengan CSS. Beberapa dari tag HTML yang ada juga telah memiliki atribut seperti "*class*" dan "*id*" yang siap di-"*style*" menggunakan CSS.
Tujuan pertama kita, adalah:

a. Membuat seluruh tag header <h2> berwarna biru
Karena kita menginginkan seluruh tag berwarna biru, maka tinggal menggunakan *CSS Tag Selector*. Inputkan kode CSS berikut diantara tag <*style*> dan <*/style*> pada bagian head:

```
h2 {
   color: blue;
   }
```

b. Membuat seluruh tag header <h3> juga berwarna biru
Untuk keperluan ini, kita tinggal menambahkan kode CSS seperti contoh **<h2>** sebelumnya, sebagai berikut:

```
h2 {
   color: blue;
   }
h3 {
   color: blue;
   }
```

Namun CSS memiliki cara yang lebih efisien jika yang kita inginkan adalah *style* yang sama untuk kedua tag. Kita bisa menggabungkan kode CSS diatas menjadi:

```
h2 , h3 {
        color: blue;
        }
```

Tanda koma (",") sebagai pemisah menginstruksikan kepada browser bahwa kedua tag akan berwarna biru. Tidak ada batasan seberapa banyak selector yang bisa digunakan. Katakan kita ingin seluruh tag header dari **<h1>** sampai **<h6>** bewarna biru, maka kode CSSnya adalah:

```
h1 , h2, h3 , h4, h5 , h6{
    color: blue;
    }
```

c. Seluruh tag yang berada di dalam tag <p> berwarna merah
Jika yang kita maksud adalah seluruh tag **** dimanapun tag tersebut berada, maka kode CSSnya cukup sebagai berikut:

```
strong {
       color:red;
       }
```

Namun kode CSS tersebut akan membuat seluruh tag <**strong**> dimanapun, termasuk tag <**strong**> yang berada di dalam tag <**h1**> akan berwarna merah. Sehingga, jika ditambahkan syarat hanya untuk tag <**strong**>yang berada di dalam tag **<p>**, maka kita harus menulis selector CSS menjadi:

```
p strong {
     color:red;
     }
```

Maksud dari kode "**p strong**" adalah seluruh tag **** yang berada di dalam tag **<p>**. perhatikan bahwa kedua tag dipisahkan dengan tanda spasi. (Selector: **p strong**, berbeda pengertiannya dengan selector: **p, strong**)

Untuk contoh *extreme*, katakan kita memiliki halaman HTML yang cukup rumit, sehingga kita menginginkan warna merah untuk "*seluruh tag <i> yang berada di dalam tag , dimana tag tersebut harus berada di dalam tag <h2>*", maka kode CSSnya adalah:

```
h2 span i {
      color:red;
      }
```

Yang perlu diingat dalam urutan ini adalah: tag yang dikenai style hanya tag yang paling terakhir, dimana dalam contoh diatas, hanya tag **<i>** saja yang akan bewarna merah.

d. Seluruh tag <h2> yang memiliki class judul, dan seluruh tag yang memiliki class paragraf menjadi *italic (huruf miring)*

Prinsip dari penggabungan ini sama dengan tujuan 1 sampai 3 sebelumnya, cuma kali ini kita harus menggunakan class selector. Kode CSS untuk keperluan itu adalah:

```
h2.judul , .paragraf {
      font-style: italic;
      }
```

h2.judul diperlukan untuk mencari seluruh tag **h2** dengan **class judul**, sedangkan **.paragraf** digunakan untuk mencari seluruh tag yang memiliki class paragraf, apapun tag tersebut.

e. Ubah ukuran text menjadi 14pt untuk tag yang memiliki id “belajar_html”, tag p dengan id “belajar_css” dan seluruh tag h3 yang memiliki class “subheader”

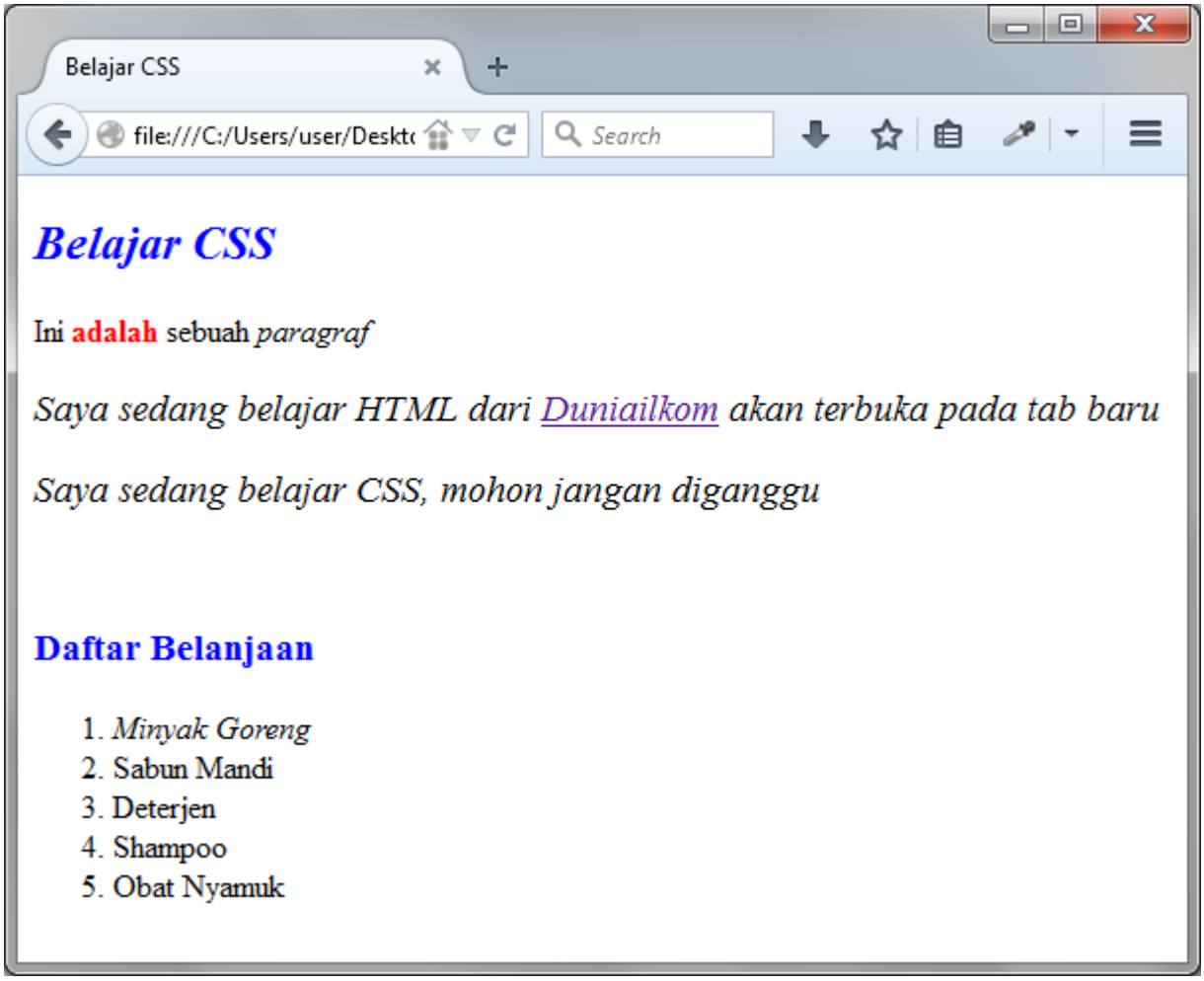
Tujuan diatas terkesan rumit, namun jika anda telah paham tujuan 1 sampai dengan 4 sebelumnya, maka kode CSS berikut akan menjelaskannya:

```
#belajar_html, p#belajar_css, h3.subheader {
       font-size:14pt;
       }
```

Sebagai penutup, hasil akhir dari halaman HTML **belajar_css.html** kita adalah sebagai berikut:

```
<!DOCTYPE html>
<html>
    <head>
    <title>Belajar CSS</title>

    <style type="text/css">
        h2 , h3{
          color: blue;
          }
        p strong {
          color:red;
          }
        h2.judul , .paragraf {
          font-style: italic;
          }
        #belajar_html, p#belajar_css, h3.subheader {
          font-size:14pt;
          }
     </style>
</head>
<body>
  <h2 class="judul">Belajar CSS</h2>
    <p> Ini <strong>adalah</strong> sebuah <em>paragraf</em>
</p>

    <p id="belajar_html" class="paragraf" >Saya sedang belajar
HTML dari
    <a href="http://www.jti.polije.com"
target="_blank">Jti.polije</a>
    akan terbuka pada tab baru</p>

    <p id="belajar_css" class="paragraf">
    Saya sedang belajar CSS, mohon jangan diganggu</p>

    <br />

    <h3 class="judul subheader">Daftar Belanjaan</h3>
    <ol>
```

```
            <li><em>Minyak Goreng</em></li>
            <li>Sabun Mandi</li>
            <li>Deterjen</li>
            <li>Shampoo</li>
            <li>Obat Nyamuk</li>
    </ol>
 </body>
</html>
```



Sampai disini setidaknya anda sudah bisa membaca maksud dari sebagian besar selector dalam CSS. Sebagai contoh, anda tentunya tidak bingung membaca kode CSS berikut:

```
table#daftar_hadir, table#daftar_kuliah {
    font-size:14pt;
    }

div.header, div.footer, div#main_menu {
    width:960px;
    }
```

g. Urutan Prioritas Selector CSS (Cascading)

Di dalam CSS, sebuah tag bisa memiliki lebih dari dari satu kode CSS. Dalam tutorial mengenal Urutan Prioritas Selector CSS ini kita akan membahas *urutan* atau *prioritas* dari kode CSS yang akan dipakai oleh tag HTML, atau dikenal dengan istilah ***Cascading***.

Pengertian Cascading dari CSS

CSS adalah singkatan dari **Cascading Style Sheet**, dimana *cascade* dalam bahasa inggris dapat berarti *air terjun kecil yang berjatuhan dari atas ke bawah*. Di dalam CSS, maksud dari **cascading** ini adalah *style* yang dapat ditimpa atau menimpa style lain sesuai urutannya, atau kita sebut saja sebagai prioritas CSS.

Mengenai kata "**prioritas**" untuk kode CSS, saya akan bagi menjadi 2 bagian, pada artikel ini hanya akan membahas tentang prioritas atau urutan dari kode CSS jika dilihat dari **"sumber"** kode tersebut. Kita akan menguji prioritas dari **external style sheet, internal style sheet,** dan **inline style CSS.** Pada tutorial berikutnya kita akan membahas tentang prioritas CSS dilihat dari kespesifikannya.

Efek Cascading berdasarkan sumber kode CSS

Misalkan kita memiliki sebuah tag header **<h2>,** lalu ingin membuat kode CSS menggunakan *external style sheet* untuk merubah tag header tersebut menjadi **biru**. Namun pada saat yang sama kita juga membuat *internel style sheet* untuk mengubahnya menjadi warna **merah**, maka warna apakah yang akan tampil di browser?

Untuk mengujinya, marilah kita mencobanya secara langsung, langkah pertama, buatlah sebuah file CSS yang akan *diimport*, misalkan **prioritas.css**, ketikkan kode CSS berikut:

```
h2 {
    color:blue;
   }
```

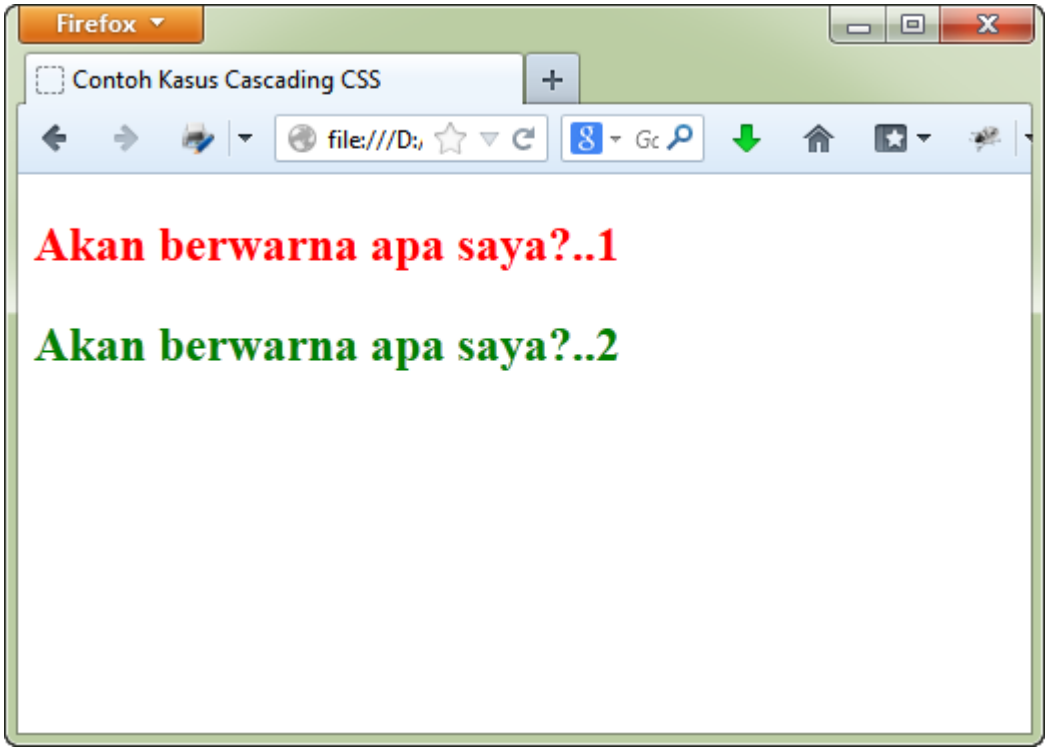
Lalu sebagai sample HTML, saya menggunakan **prioritas.html,** savelah kedua file pada folder yang sama:

```
<!DOCTYPE html>
<html>
<head>
   <title>Contoh Kasus Cascading CSS</title>
   <link rel="stylesheet" type="text/css"
href="prioritas.css">
   <style type="text/css">
            h2 {
               color:red;
               }
    </style>
```

```
</head>

<body>
    <h2>
        Akan berwarna apa saya?..1
    </h2>

    <h2 style="color:green">
        Akan berwarna apa saya?..2
    </h2>
</body>
</html>
```

Perhatikan bahwa sebelum tag <*style*>, saya "*memanggil*" file **prioritas.css** terlebih dahulu. Lalu pada tag **<h2>** yang kedua saya menambahkan atribut "**color:green**" pada tag **<h2>.**

Jika kita menjalankan kode HTML diatas, warna text pada kedua tag <h2> akan bewarna merah dan biru, dan tidak ada yang bewarna biru.



Dari contoh sederhana diatas, tag **<h2>** sebenarnya "*dikenakan"* atau *"diubah"* oleh 3 *property* CSS yang sama secara bersamaan, yakni ketiga selector tersebut ingin mengubah warna text dari tag <h2>, namun hanya ada satu kode yang akan "**menang**".

Dalam masalah '*timpa-menimpa'* ini, CSS memiliki aturan prioritas tersendiri. Jika terdapat property CSS yang saling '*bentrok'*, maka urutan prioritasnya adalah sebagai berikut (dari yang paling kuat):

- **Inline style**, yakni style yang langsung melekat pada tag.
- **Internal style**, yakni style yang dideklarasikan pada awal halaman (tag <*style*>)
- **Eksternal style**, yakni style yang dideklarasikan pada sebuah file .css , dan dipanggil melalui tag <*link*> atau *@import*

Maka jika melihat sekali lagi kode HTML diatas, text "*Akan berwana apa saya?..1*" akan berwarna **merah** karena **internal style** *color:red* lebih mendapat prioritas lebih tinggi daripada **external style** *color:red*.

Sedangkan text "*Akan berwana apa saya?..2*" berwarna **hijau** karena **inline style** *color:green* lebih mendapat prioritas daripada **external style** *color:blue* maupun **internal style** *color:red*.

Selain prioritas antar "*sumber*" kode CSS tersebut, proses *cascading* atau prioritas CSS juga masih berlanjut untuk kode CSS dalam file yang sama.

h. Urutan Prioritas Selector CSS (Specificity)

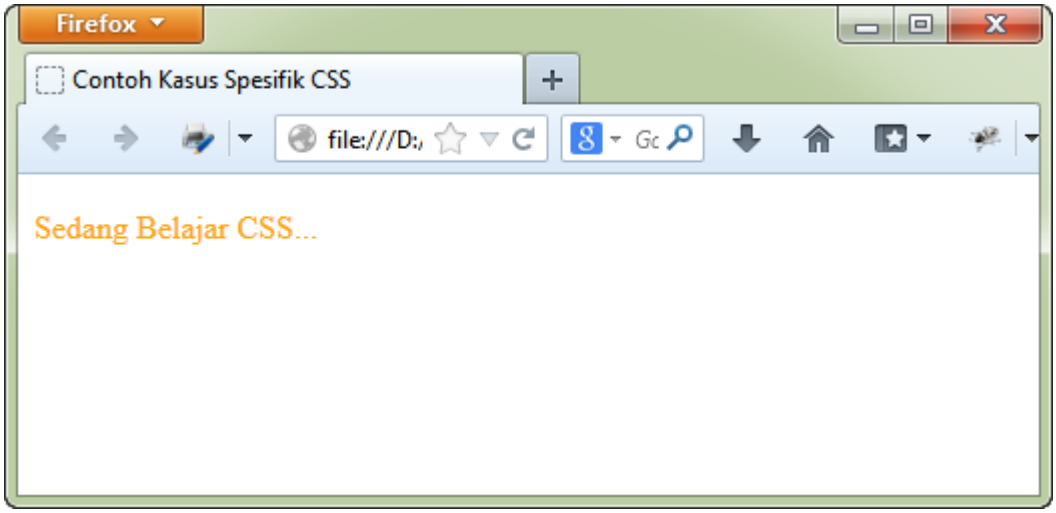
Pengertian ke-*spesifik*-an selector CSS

Pada bagian ini kita akan membahas urutan prioritas kode CSS jika seluruh kode CSS tersebut berada dalam file yang sama. Dalam kasus ini kita akan mempelajari apa yang akan terjadi jika beberapa kode CSS yang dibuat saling menimpa.
Agar lebih mudah dipahami, saya akan langsung membuat file sample kita, yaitu **spesifik.html** :

```
<!DOCTYPE html>
<html>
<head>
   <title>Contoh Kasus Spesifik CSS</title>
   <style type="text/css">
            p  {
               color:red;
               }
            div p {
               color:green;
               }
            #aaa{
               color:orange;
               }
            body div p {
               color:yellow;
               }
            div p.kalimat {
               color:silver;
               }
    </style>
</head>

<body>
   <div>
```

```
    <p id="aaa">
        Sedang Belajar CSS...
    </p>
   </div>
</body>
</html>
```

Sebelum anda menjalankan file **spesifik.html** tersebut, silahkan perhatikan terlebih dahulu defenisi CSS diatas. Terdapat 5 jenis selector yang '*mencoba*' untuk merubah warna sebuah paragraf. Dan silahkan tebak akan bewarna apa paragraf "*Sedang Belajar CSS...*".



Dan, seperti yang kita lihat, paragraf tersebut tampak berwarna **orange**, dimana selector CSS "**#aaa**" adalah tempat pendefenisisan warna "*color:orange*", namun bagaimana ini bisa terjadi? bagaimana dengan ke-4 defenisi lainnya?

Jawabannya: karena selector **#aaa** dianggap paling spesifik atau paling *detail* dari ke-5 selector lainnya.

Cara Menghitung ke-*spesifik*-an Selector CSS

CSS memiliki aturan tertentu tentang ke-spesifik-an selector, yaitu ketika sebuah property dari CSS, saling menimpa satu sama lain, ***selector yang paling spesifik lah yang akan menang***.

Aturan ke-spesifik-an **CSS** dihitung menggunakan formula 4 digit, contohnya: 0,1,0,0. Perhitungannya sama seperti hitung-hitungan kita sehari-hari, dimana 0,1,0,0 lebih besar dari 0,0,1,5, atau agar lebih mudah nilai 0,1,0,0 bisa ditulis menjadi 0100, dan nilai 0,0,1,5 menjadi 0015.

Berikut adalah nilai **CSS Specificity** untuk selector di dalam CSS:

- Setiap element/tag selector bernilai 0,0,0,1
- Setiap class selector, attribut selector bernilai 0,0,1,0
- Setiap ID selector bernilai 0,1,0,0

- Setiap inline style bernilai 1,0,0,0

Dengan menggunakan aturan tersebut, mari kita hitung angka *ke-spesifik-an* dalam contoh sebelumnya.

- Selector **p**, hanya terdiri dari 1 tag selector, maka nilainya: 0,0,0,1
- Selector **div p**, terdiri dari 2 tag selector, maka nilainya: 0,0,0,2
- Selector **#aaa**, terdiri dari 1 ID selector, maka nilainya: 0,1,0,0
- Selector **body div p**, terdiri dari 3 tag selector, maka nilainya: 0,0,0,3
- Selector **div p.kalimat**, terdiri dari 1 class selector dan 2 tag selector, maka nilainya: 0,0,1,2

Dari hasil yang kita peroleh, maka selector **#aaa** memiliki nilai paling tinggi, yaitu 0,1,0,0. Maka, karena itulah paragraf dalam contoh **spesifik.html** akan bewarna *orange*.

Sering kali dalam merancang sebuah website yang cukup kompleks, kita akan sering dibuat pusing dan bertanya-tanya kenapa style yang telah ditulis tidak merubah kode HTML yang ada, atau tidak berefek apa-apa. Mungkin jawabannya adalah nilai selector tersebut tertimpa oleh nilai selector lain yang lebih spesifik. Memahami aturan **prioritas** dan **ke-spesifik-an** CSS ini sangat penting untuk menghindari hal tersebut.

Mengenal perintah !important

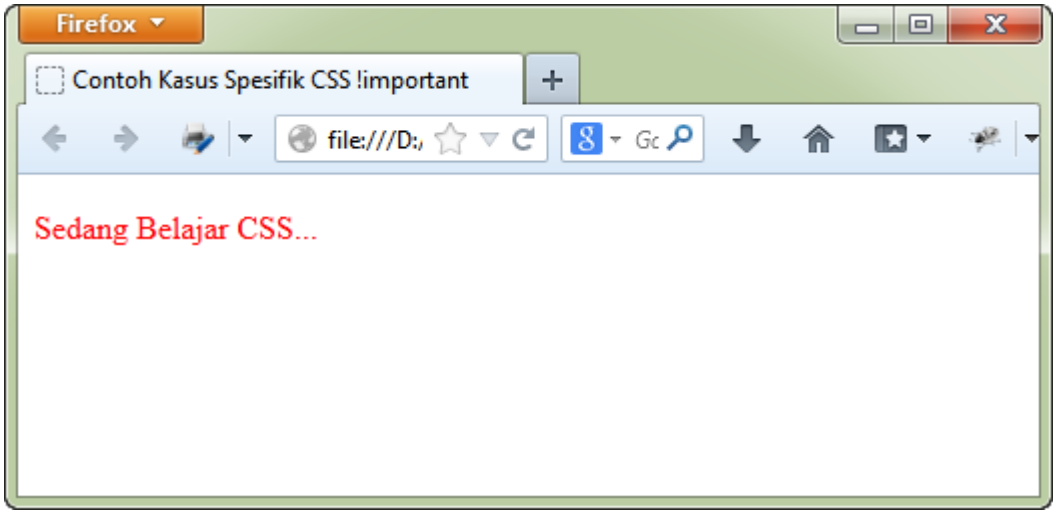
Jika aturan prioritas diatas tidak cukup, CSS menyediakan "***senjata***" pamungkas, yaitu menggunakan perintah **!important**.

Perintah **!important** diletakkan di belakang *propery* dari CSS, dan perintah **!important** ini akan mengambil alih urutan prioritas CSS yang kita pelajari diatas. Mari kita ubah **spesifik.html** dengan menambahkan perintah **!important**.

```
<!DOCTYPE html>
<html>
<head>
   <title>Contoh Kasus Spesifik CSS !important</title>
   <style type="text/css">
            p  {
               color:red !important;
               }
            div p {
               color:green;
               }
            #aaa{
               color:orange;
               }
            body div p {
               color:yellow;
```

```
            }
        div p.kalimat {
            color:silver;
            }
    </style>
</head>

<body>
    <div>
     <p id="aaa" class="kalimat">
         Sedang Belajar CSS...
     </p>
    </div>
</body>
</html>
```

Perhatikan akhir baris pada selector **p**, saya menambahkan perintah **!important** diakhir property, dan seperti yang bisa ditebak, paragraf kita sekarang akan berwarna merah.



Penggunaan perintah **!important** sebaiknya digunakan jika memang sudah terdesak dan sedapat mungkin dihindari,. Hal ini karena perintah **!important** akan mengubah urutan prioritas dan akan menyulitkan perancangan CSS.

Dalam **tutorial CSS** kali ini, bisa dikatakan bahwa perancangan kode **CSS** sebenarnya lebih condong ke *seni* (**art**) daripada pemograman. Karena selain memikirkan bagaimana tampilan web, kita juga harus memperhatikan aturan *prioritas* dan *ke-spesifik-an* CSS, dan mengombinasikannya untuk membuat tampilan website yang indah.

i. Mengenal Sifat Penurunan Dalam CSS (Inheritance)

Dalam tutorial belajar CSS kali ini kita akan membahas tentang efek penurunan property di dalam CSS, atau lebih populer dengan kata inggrisnya "**inheritance**" dalam CSS. Tutorial ini mungkin terkesan agak 'rumit', namun merupakan salah satu poin penting dalam memahami CSS.

Pengertian Inheritance dalam CSS

Inheritance dalam CSS adalah sifat penurunan efek **CSS** dari sebuah tag HTML kepada tag HTML lainnya. Syarat untuk **inheritance** adalah**:** tag tersebut harus berada di dalam tag lainnya.

Di dalam HTML, setiap tag umumnya akan berada di dalam tag lain. Untuk seluruh tag HTML, akan berada di dalam tag `<html>`, dan untuk seluruh tag yang tampil di web browser akan berada di dalam tag `<body>`.

Sebagai contoh, perhatikan potongan HTML berikut ini:

```
<div><p>Saya sedang belajar <em>inheritance</em>
CSS</p></div>
```

Dari contoh diatas, tag **<p>** berada di dalam tag ** ,** sehingga dapat dikatakan tag ** ** adalah induk (*parent*) dari tag **<p>.** Sedangkan tag **** yang berada di dalam tag **<p>** merupakan anak (*child*) dari tag **<p>** dan *grandchild* dari tag ** .**

Jika kita membuat kode CSS sebagai berikut:

```
div {
color:green;
}
```

Maka efek dari kode **CSS** tersebut akan membuat seluruh text di dalam tag <**div**> akan berwarna hijau, walapun di dalam tag <**div**> juga terdapat tag-tag lainnya. Dapat dikatakan bahwa efek CSS tersebut diturunkan (***inherit***) dari tag induk <**div**> kepada tag anak <**p**> dan <**em**>.

Namun perlu diperhatikan bahwa tidak semua *property* CSS akan diturunkan dari tag induknya. Misalnya *property* **border**, hanya berlaku untuk satu tag induk saja dan tidak akan diturunkan kepada tag anak.

Property seperti *background-color* yang digunakan untuk mengubah warna latar belakang sebenarnya juga tidak diturunkan, namun '*seolah-olah*' diturunkan. Hal ini terjadi karena nilai '*default*' dari property *background-color* adalah '*transparent*', dimana warna latar belakang tag induk akan dilewatkan, sehingga seolah-olah juga bewarna seperti induknya.

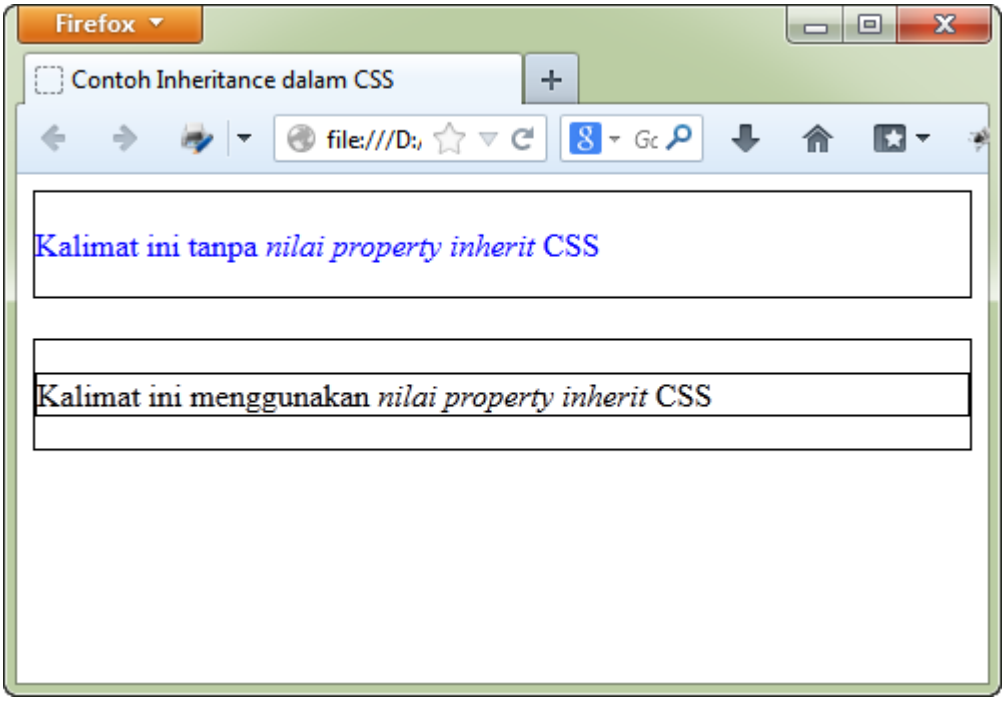
Mengenal Nilai Property: inherit

Untuk '*memaksa'* proses penurunan, CSS memiliki nilai *property:* **inherit**. Jika sebuah kode **CSS** memiliki nilai property **inherit**, maka property tersebut akan mencopy nilai dari tag induk (parent).

Sebagai contoh, karena *property* **border** secara *default* tidak diturunkan, maka kita akan mencoba nilai **inherit** pada *property* **border**.

Contoh HTML yang akan kita gunakan adalah inherit.html:

```
<!DOCTYPE html>
<html>
<head>
   <title>Contoh Inheritance dalam CSS</title>
   <style type="text/css">
      div.satu {
            border: 1px solid black;
            color: blue;
      }

      div.dua {
            border: 1px solid black;
      }
      p.dua {
            border: inherit;
      }
    </style>
</head>

<body>
   <div class="satu">
     <p class="satu">
      Kalimat ini tanpa <em class="satu">nilai property
inherit</em> CSS
     </p>
   </div>
   <br />
   <div class="dua">
     <p class="dua">
       Kalimat ini menggunakan <em class="dua">nilai property
inherit</em> CSS
     </p>
   </div>
</body>
</html>
```



Dalam contoh kode **HTML** diatas, saya menggunakan dua kalimat yang memiliki nama *class* yang berbeda, yakni class **satu** dan class **dua**.

Pada **class="satu"**, saya hanya membuat *property border* dan *color* pada tag **div.satu**. Karena prinsip inheritance dalam CSS, seluruh kalimat pada tag <**div**> yang memiliki **class="satu"** akan berwarna biru (sesuai dengan sifat *inheritance*-nya), namun efek border tidak diturunkan kepada tag <**p**>.

Pada **class="dua"**, saya menambahkan nilai inherit untuk property border pada tag **<p>.** Hasil dari nilai inherit ini akan membuat tag <p> juga memiliki border yang sama dengan yang didefenisikan pada tag induknya.

Sifat *inheritance* **CSS** ini akan memudahkan dalam desain web, karena jika kita ingin membuat seluruh tulisan menjadi warna dan ukuran tertentu, cukup dengan membuat style untuk tag induk, maka seluruh tag yang berada di dalamnya akan ikut berubah.

Misalkan kita ingin seluruh font berukuran 14px, maka kode CSSnya, cukup dengan:

```
body {
font-size: 14px;
}
```

Dan seluruh tag yang berada di dalam tag **body** (hampir semua tag merupakan child dari tag **body**), ukuran fontnya akan berubah menjadi 14px.

j. Mengenal Model Warna RGB

Dalam desain di media visual seperti televisi maupun monitor komputer, model warna yang digunakan adalah model warna **RGB (Red Green Blue)**. Disebut

RGB, karena warna *merah*, *hijau* dan *biru* inilah dasar dari warna lainnya. Warna seperti kuning, coklat, pink, hitam dan putih berasal dari perpaduan ketiga warna ini.

Didalam desain web, CSS mendukung hingga **16.777.216** kombinasi warna. Warna ini berasal dari perpaduan warna merah, hijau dan biru tersebut. Masing-masing warna dasar ini dapat bernilai 0 sampai 255, dimana 0 adalah tanpa warna, 255 adalah warna maksimal. Dalam dunia ilmu komputer, kombinasi warna ini dikenal dengan **24bit warna**, atau disebut juga **true color**. Perpaduan ketiga warna ini akan memberi kita 256 kombinasi merah x 256 kombinasi hijau x 256 kombinasi biru = 16.777.216 kombinasi warna.

Misalkan, untuk mendapatkan warna kuning, sebenarnya warna kuning adalah perpaduan warna merah dan hijau. Untuk mendapatkan warna kuning terang, maka kita harus mencampurkan merah sebanyak 255, hijau sebanyak 255, dan biru sebanyak 0.

Format #RRGGBB

Format penulisan warna **#RRGGBB** adalah yang paling populer digunakan untuk CSS. Dimana **RR** adalah nilai untuk warna *merah*, **GG** untuk *hijau*, dan **BB** untuk warna *biru*. Masing-masing nilai ini berisi angka **00** sampai dengan **FF**. Kenapa FF? Karena CSS menggunakan angka *hexadesimal*. FF adalah nilai maksimal, equivalen dengan 255 dalam nilai desimal

Contohnya, **#FF0000** adalah warna merah '*murni*', **#00FF00** adalah warna hijau murni, sedangkan **#777777** adalah kode untuk warna silver.

Format #RGB

Selain menggunakan format warna 6 digit, CSS juga mendukung penulisan 3 digit. Format penulisan 3 digit ini merupakan penulisan singkat dari 6 digit. Contohnya **#RGB** merupakan penulisan singkat dari #RRGGBB, #F0F adalah singkatan dari #FF00FF, #09A sama dengan #0099AA (merah=00, hijau=99, biru=AA).

Penulisan Desimal

Selain penulisan hexadesimal, CSS juga menyediakan penulisan warna menggunakan angka desimal. Format penulisannya adalah: **rgb(0, 160, 255)** atau **rgb(0%, 63%, 100%)**, dimana urutan warna adalah merah, hijau, dan biru. Menggunakan format pertama, 255 sama dengan 100%. Jika kita menuliskan nilai diluar dari angka yang seharusnya, yakni 0 – 255 atau 0% – 100% maka CSS secara otomatis akan menyamakannya dengan nilai yang terdekat. Format penulisan warna seperti ini tidak terlalu sering digunakan.

Kata Warna (Keyword)

Selain menggunakan kode warna dengan format RGB, CSS juga mendukung 17 kata warna dalam bahasa inggris, yakni: ***aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, dan yellow***. Kata-

kata ini berasal dari warna dasar VGA pada Windows dan dikenal juga dengan istilah *classic internet color*. Berikut tabel lengkap kata warna CSS:

Tabel Kata Warna dalam CSS

Keyword Warna	Nilai RGB
aqua	#00ffff
black	#000000
blue	#0000ff
fuchsia	#ff00ff
gray	#808080
green	#008000
lime	#00ff00
maroon	#800000
navy	#000080
olive	#808000
orange	#ffa500
purple	#800080
red	#ff0000
silver	#c0c0c0
teal	#008080
white	#ffffff
yellow	#ffff00

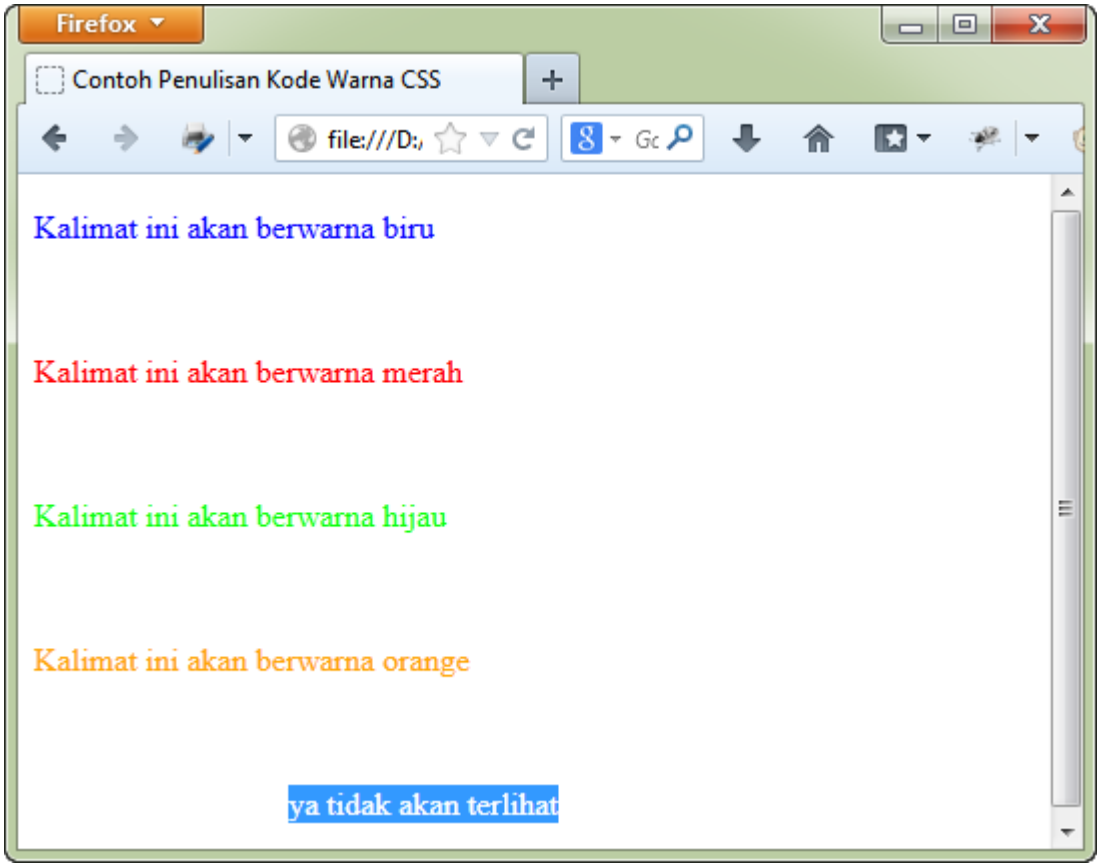
Selain tabel warna standar di atas, beberapa modern web browser juga mendukung keyword warna yang disebut 147 SVG colors (atau X11 colors). List lengkap untuk keyword warna tambahan ini dapat dilihat di : https://developer.mozilla.org/en-US/docs/CSS/color_value.

Nilai property: Transparent

Pada kasus tertentu, kita ingin sebuah tag dalam CSS menjadi transparan, untuk keperluan ini CSS menyediakan keyword *transparent* sebagai nilai. Dengan memberikan nilai transparan, sebuah tag akan 'melewatkan' warna dari tag dibelakangnya.

Sebagai contoh penggunaan kode warna diatas, saya akan menampilkannya dalam file **warna.html** sebagai berikut:

```
<!DOCTYPE html>
<html>
<head>
   <title>Contoh Penulisan Kode Warna CSS</title>
   <style type="text/css">
            p.satu {
            color:#0000FF;
            }
            p.dua {
            color:#F00;
            }
            p.tiga {
            color:rgb(10, 255, 10);
            }
            p.empat {
            color:orange;
            }
            p.lima{
            color:transparent;
            }
    </style>
</head>

<body>
       <p class="satu">
          Kalimat ini akan berwarna biru
       </p>
   <br />
       <p class="dua">
          Kalimat ini akan berwarna merah
       </p>
   <br />
       <p class="tiga">
          Kalimat ini akan berwarna hijau
       </p>
   <br />
       <p class="empat">
          Kalimat ini akan berwarna orange
       </p>
   <br />
       <p class="lima">
          Kalimat ini sepertinya tidak akan terlihat
       </p>
</body>
</html>
```



Seperti yang terlihat dari kode HTML di atas, setiap kalimat diset menggunakan kode warna yang berbeda-beda. Yang menjadi menarik, nilai **transparent** yang diberikan kepada selector **class=”lima”** menjadi tidak terlihat, dan hanya terlihat jika di blok.

k. Mengenal Satuan Nilai (Value) dalam CSS

Nilai Absolut

Nilai absolut adalah sebuah nilai di dalam CSS, dimana nilai tersebut tidak dipengaruhi kondisi tag induknya (*parent*), dan didasarkan pada ukuran sebenarnya di media *printing*. Di dalam CSS, nilai absolut ini adalah: **milimeter (mm), centimeter (cm), inchi (in), point (pt),** dan **pica (pc).**

Kelima nilai absolut ini berasal dari media percetakan. Kita tentunya sudah familiar dengan ukuran seperti **milimeter**, dan **centimeter**. Sedangkan 1 **inchi** adalah sekitar 25,4 mm.

Ukuran **point (pt)** sering digunakan dalam media cetak offline seperti *Microsoft word* dalam menentukan ukuran dari **font**. 1 inchi setara dengan 72pt, sehingga ukuran font normal sebesar 12pt setara dengan seperdelapan inchi. Contoh penggunaannya dalam CSS: **p {font-size: 18pt;}** sama dengan **p {fontsize: 0.25in;}.**

Ukuran **pica (pc)** mungkin tidak terlalu familiar. 1pc setara dengan 12pt. contoh penulisannya dalam CSS adalah **p {font-size: 1.5pc;}.**

Kelima nilai absolut tersebut, selain **pt** tidak terlalu sering digunakan, karena konversi dari ukuran dunia nyata ke dalam monitor dipengaruhi banyak hal, 1cm di satu monitor bisa jadi berbeda dengan 1cm pada monitor lainnya, sehinga agak menyulitkan dalam mendesain kode CSS yang "pas".

Nilai Relatif

Berbeda dengan nilai absolut, nilai relatif didalam CSS dipengaruhi oleh tag disekitarnya. CSS mendukung banyak nilai relatif, beberapa yang kita bahas adalah: **pixel (px), Em-height (em),** dan **X-height (ex).**

Ukuran **pixel (px),** merupakan ukuran yang paling sering digunakan dalam CSS. Di dalam CSS, pixel termasuk nilai relatif karena ia tidak terikat dengan perbandingan media tertentu. Didalam spesifikasi CSS2. 1, 1 pixel disamakan dengan 0.26mm atau 1/96 inci, namun banyak browser yang mengabaikan spesifikasi ini dan menyamakan 1 pixel dengan 1 pixel sesungguhnya yang terdapat di monitor.

1 unit **Em-height (em)** sama dengan besar ukuran asal yang diturunkan tag tersebut. Ketika kita mendefenisikan sebuah paragraf sebesar 12pt, maka sebuah tag **** yang berada di dalam paragraf tersebut disebut secara **inheritance** akan berukuran 1em. Jika kita merubahnya menjadi **em {font-size: 1.2em;}** maka tag **** tersebut akan berukuran 1,2 kali lebih besar dari ukuran tag p.

Ukuran relatif terakhir yaitu **X-height (ex).** ex berasal dari tinggi sebuah karakter "x" (huruf x kecil) dari sebuah font. Namun, banyak font yang tidak mendukung hal ini, sehingga browser kadang kala menyamakan **1ex** dengan **0,5em**.

Nilai Persentasi

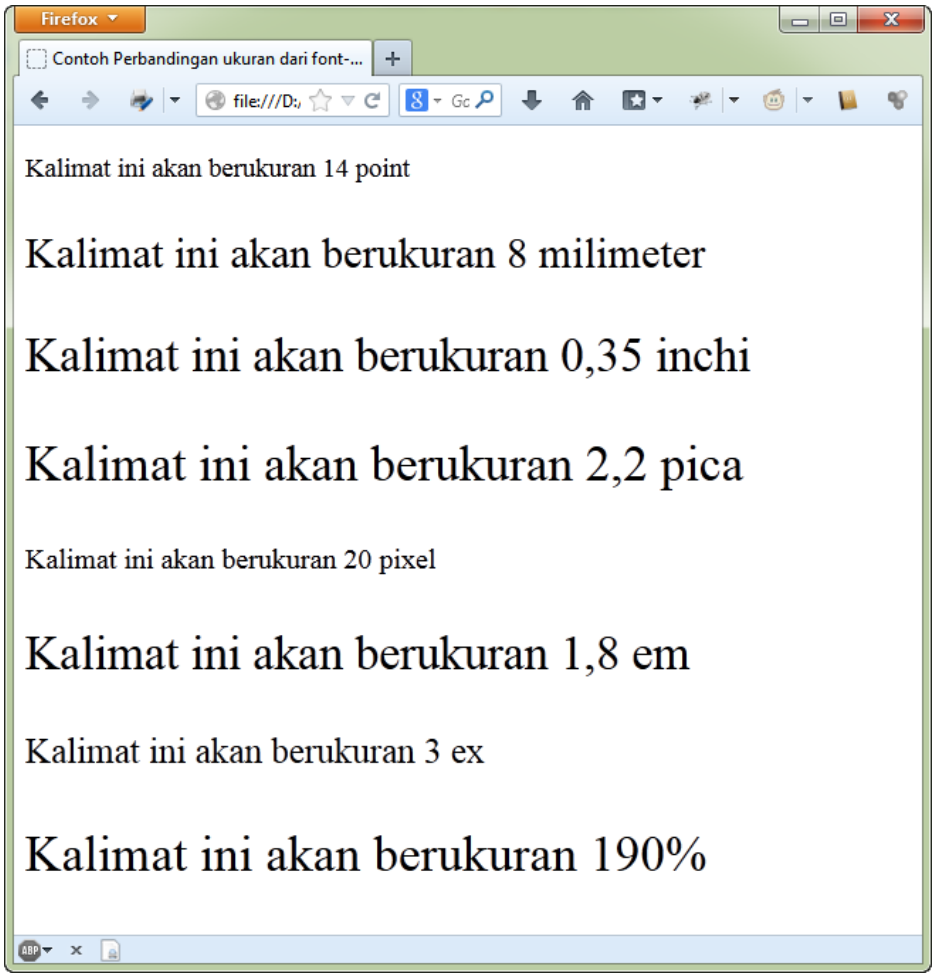
Nilai persentasi adalah sebuah nilai per seratus (%) dari suatu property yang relatif terhadap induk (*parent*) dari tag tersebut. Misalkan kita membuat sebuah defenisi CSS sebagai **p {font-size: 10pt**;}, dan **em {font-size: 120%;}.** Maka jika terdapat tag **** didalam tag **<p>,** maka ukuran font dari tag **** akan menjadi 120% dari 10pt, atau sekitar 12pt.

Dengan membuat nilai font lainnya menjadi persen, hal ini akan memudahkan jika suatu saat kita ingin merubah seluruh tampilan web. Hanya merubah ukuran tag **<p>**, maka secara langsung tag **** akan menyesuaikan tampilannya.

Berikut adalah contoh **perbandingan_fontsize.html**, dimana saya menampilkan ukuran property **font-size** dengn berbagai value:

```
<!DOCTYPE html>
<html>
<head>
   <title>Contoh Perbandingan Ukuran dari font-size</title>
   <style type="text/css">
            body    { font-size:14pt; }
            p.satu  { font-size:8mm; }
            p.dua   { font-size:0.35in; }
            p.tiga  { font-size:2.2pc; }
            p.empat { font-size:20px; }
            p.lima  { font-size:1.8em }
```

```
            p.enam  { font-size:3ex; }
            p.tujuh { font-size:190%; }
    </style>
</head>
 <body>
       <p>Kalimat ini akan berukuran 14 point</p>
       <p class="satu">Kalimat ini akan berukuran 8
milimeter</p>
       <p class="dua">Kalimat ini akan berukuran 0,35
inchi</p>
       <p class="tiga">Kalimat ini akan berukuran 2,2 pica</p>
       <p class="empat">Kalimat ini akan berukuran 20
pixel</p>
       <p class="lima">Kalimat ini akan berukuran 1,8 em</p>
       <p class="enam">Kalimat ini akan berukuran 3 ex</p>
       <p class="tujuh">Kalimat ini akan berukuran 190%</p>
</body>
</html>
```



Jika diperhatikan lagi, saya membuat sebuah selector **body {font-size:14pt;}** pada bagian awal definisi CSS. Selector body akan menurunkan (**inherit**) ukuran font ini ke seluruh tag **<p>** yang berada di dalam tag <body>**,** namun pada masing-masing

tag **<p>,** saya menambahkan property HTML "**class**", sehingga ukuran font akan ditimpa (**override**) oleh nilai lainnya.

Khusus untuk **class=lima, class=enam,** dan **class=tujuh**, saya menggunakan nilai relatif **em, ex,** dan **%**, sehingga nilainya bergantung dengan nilai induknya (parent), yakni **body {font-size:14pt;}.** Cobalah mengubah ukuran font-size pada **body** ini, lalu jalankan kembali

l. Pengertian CSS Vendor Prefix dan Cara Penulisannya

Pengertian CSS Vendor Prefix

Vendor prefix adalah sebutan untuk penambahan beberapa karakter khusus di awal penulisan property, terutama untuk property CSS3 terbaru. Sebagai contoh, untuk property **column-count**, jika menggunakan vendor prefix ditulis menjadi: **-webkit-column-count**.

Kenapa harus menggunakan CSS Vendor Prefix?

Seiring dengan cepatnya perkembangan CSS3, web browser seolah-olah berlomba untuk mengimplementasikan berbagai property baru yang belum resmi disetujui badan standarisasi web: **W3C**. Dalam istilah W3C, property yang belum standar ini bisanya berada dalam status **draft**, dan bisa berubah sewaktu-waktu.

Property yang belum 'selesai' ini diimplementasikan oleh web browser menggunakan **vendor prefix**. Dengan tujuan, property ini bisa diuji coba oleh programmer web di seluruh dunia.

Pada awalnya, property dengan **vendor prefix** tidak ditujukan untuk website live, tapi hanya untuk uji coba. Ketika spesifikasi **W3C** masuk ke tahap **rekomendasi** (yaitu ketika property tersebut sudah dianggap selesai), tambahan **vendor prefix** juga akan dihapus.

Masalahnya, banyak web developer yang tidak sabar menunggu property ini resmi dirilis dan memilih untuk langsung menggunakannya. Oleh karena itu, jika kita ingin menggunakan property **CSS3** yang masih baru harus dibuat menggunakan **vendor prefix**.

Cara Penulisan Vendor Prefix

Vendor prefix ditambahkan di awal penulisan property, sesuai dengan inisial web browser. Berikut adalah awalan vendor prefix pada web browser populer:

- **-webkit-** (Chrome, dan versi terbaru dari Opera)
- **-moz-** (Firefox)
- **-o-** (Opera versi lama)
- **-ms-** (Internet Explorer)

Berikut contoh cara penulisan **CSS Vendor Prefix**:

```
div {
```

```
  -webkit-column-count: 3;
  -moz-column-count: 3;
  -o-column-count: 3;
  -ms-column-count: 3;
  column-count: 3;
}
```

Property **column-count** digunakan untuk membuat kolom koran (multiple column). Sayangnya property ini belum menjadi standar resmi, sehingga belum didukung penuh oleh web browser (pada saat artikel ini ditulis). Karena itu kita harus menulisnya dengan penambahan **vendor prefix**.

Pada baris terakhir terdapat penulisan property 'resmi', yakni **tanpa vendor prefix**. Ini dipersiapkan agar ketika property tersebut sudah dinyatakan stabil (sudah didukung penuh), nilai dari property ini akan menimpa efek sebelumnya (yang menggunakan vendor prefix). Dengan demikian, hasil yang di dapat akan seragam pada setiap web browser.

Penambahan property dengan **vendor prefix** memang sedikit merepotkan. Kode CSS kita menjadi 5 kali lebih panjang, namun ini hanya digunakan untuk property CSS3 yang relatif baru. Jika property tersebut sudah selesai, kita bisa 'membuang' bagian vendor prefix dan menggunakan nama property resmi.

C. JS

a. Pengertian JavaScript

JavaScript adalah bahasa pemrograman web yang bersifat *Client Side Programming Language*. **Client Side Programming Language** adalah tipe bahasa pemrograman yang pemrosesannya dilakukan oleh *client*. Aplikasi *client* yang dimaksud merujuk kepada *web browser* seperti **Google Chrome dan Mozilla Firefox**.

Bahasa pemrograman *Client Side* berbeda dengan bahasa pemrograman *Server Side* seperti PHP, dimana untuk *server side* seluruh kode program dijalankan di sisi server.

Untuk menjalankan **JavaScript**, kita hanya membutuhkan aplikasi *text editor* dan *web browser*. **JavaScript** memiliki fitur: *high-level programming language, client-side, loosely tiped* dan berorientasi objek.

Fungsi JavaScript Dalam Pemograman Web

JavaScript pada awal perkembangannya berfungsi untuk membuat interaksi antara user dengan situs web menjadi lebih cepat tanpa harus menunggu pemrosesan di *web server*. Sebelum *javascript*, setiap interaksi dari user harus diproses oleh *web server*.

Bayangkan ketika kita mengisi *form registrasi* untuk pendaftaran sebuah situs web, lalu men-klik tombol *submit*, menunggu sekitar 20 detik untuk website memproses isian form tersebut, dan mendapati halaman yang menyatakan bahwa terdapat kolom form yang masih belum diisi.

Untuk keperluan seperti inilah **JavaScript** dikembangkan. Pemrosesan untuk mengecek apakah seluruh form telah terisi atau tidak, bisa dipindahkan dari *web server* ke dalam *web browser*.

Dalam perkembangan selanjutnya, *JavaScript* tidak hanya berguna untuk *validasi form*, namun untuk berbagai keperluan yang lebih modern. Berbagai animasi untuk mempercantik halaman web, fitur chatting, efek-efek modern, games, semuanya bisa dibuat menggunakan *JavaScript*.

Akan tetapi karena sifatnya yang dijalankan di sisi client yakni di dalam web browser yang digunakan oleh pengunjung situs, user sepenuhnya dapat mengontrol eksekusi JavaScript. Hampir semua web browser menyediakan fasilitas untuk mematikan JavaScript, atau bahkan mengubah kode JavaScript yang ada. Sehingga kita tidak bisa bergantung sepenuhnya kepada JavaScript.

Perkembangan JavaScript Saat Ini

Dalam perkembangannya, **JavaScript** mengalami permasalahan yang sama seperti kode pemograman web yang bersifat *client side* seperti **CSS**, yakni bergantung kepada implementasi web browser.

Kode *JavaScript* yang kita buat, bisa saja tidak bekerja di *Internet Explorer*, karena web browser tersebut tidak mendukungnya. Sehingga programmer harus bekerja extra untuk membuat kode program agar bisa "mengakali" dukungan dari web browser.

Karena hal tersebut, **JavaScript** pada awalnya termasuk bahasa pemograman yang rumit, karena harus membuat beberapa kode program untuk berbagai web browser.

Namun, beberapa tahun belakangan ini, **JavaScript** kembali bersinar berkat kemudahan yang ditawari oleh komunitas programmer yang membuat library **JavaScript** seperti **jQuery**. Library ini memudahkan kita membuat program **JavaScript** untuk semua web browser, dan membuat fitur-fitur canggih yang sebelumnya membutuhkan ribuan baris kode program menjadi sederhana.

Kedepannya, **JavaScript** akan tetap menjadi kebutuhan programmer, apalagi untuk situs saat ini yang mengharuskan punya banyak fitur modern sebagai standar.

b. Sejarah dan Perkembangan Versi JavaScript

Nama **JavaScript** sebenarnya sedikit membingungkan. Selain kemiripan cara penulisan dengan bahasa pemograman **Java**, *JavaScript* sepenuhnya berbeda dengan bahasa pemograman desktop: **Java**.

Sejarah Penamaan JavaScript

Sejarah **JavaScript** dimulai sekitar tahun 1994, ketika internet dan website sedang mengalami perkembangan yang pesat. Website pada saat itu umumnya dibuat menggunakan bahasa pemograman **PERL** yang pemrosesannya hanya bisa dilakukan di sisi web server.

Kelemahan pemrosesan di sisi *web server* adalah, setiap instruksi dari user harus dikirim terlebih dahulu kepada *web server*, baru kemudian ditampilkan lagi di dalam *web browser*. Karena kecepatan rata-rata koneksi internet yang terbatas, hal ini dipandang tidak efisien. Programmer web membutuhkan bahasa pemograman *client-side* yang bisa berjalan di *web browser* tanpa harus dikirim ke server.

Pada tahun 1995, **Brendan Eich** seorang programmer dari **Netscape** mulai mengembangkan sebuah bahasa pemograman script yang dinamakan **Mocha**. *Netscape* pada saat itu merupakan perusahaan software ternama yang memiliki web broser **Netscape Navigator**.

Brendan Eich (source: wikipedia)

Bahasa script *Mocha* ini ditujukan untuk *client-side* dan juga *server-side*. Dalam perkembangan selanjutnya, nama *Mocha* diubah menjadi **LiveScript** untuk versi *client-side*, dan **LiveWire** untuk versi *server-side*.

Pada saat bahasa pemograman tersebut akan dirilis, *Netscape* mengadakan kerjasama dengan **Sun Microsystems** untuk mengembangkan *LiveScript*, dan tepat ketika **Netscape Navigator 2** dirilis, *Netscape* merubah nama *LiveScript* menjadi **JavaScript** dengan tujuan bahasa baru ini akan populer seperti bahasa *Java* yang saat itu sedang booming di kalangan programmer. Versi *JavaScript* ini dinamakan dengan **JavaScript 1.0**.

Pesaing JavaScript: Kemunculan JScript dari Microsoft

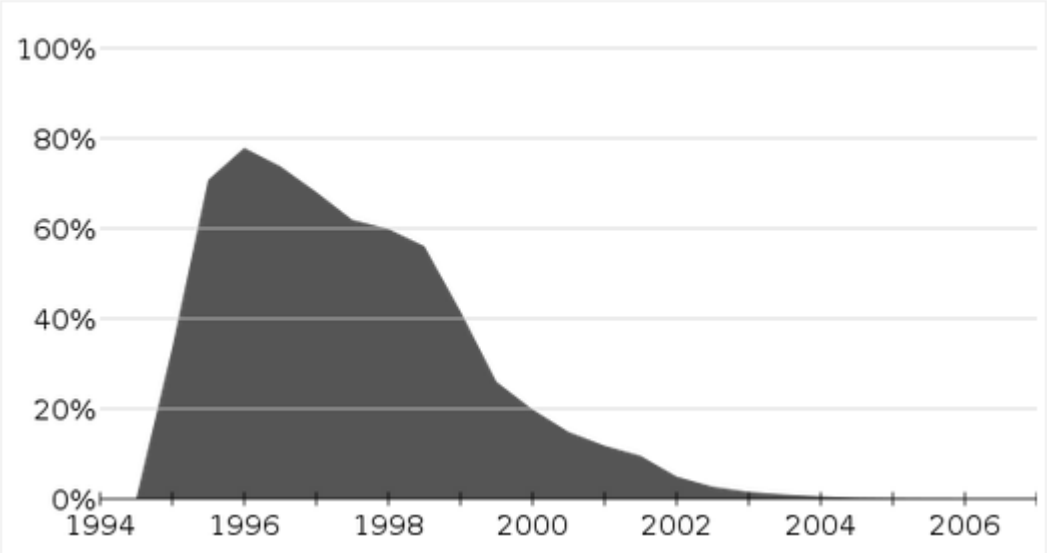
Karena kesuksesan *JavaScript 1.0*, *Netscape* selanjutnya mengembangkan JavaScript versi 1.1 pada Netscape Navigator 3, dan mengantarkan *Netscape Navigator* menjadi pemimpin pasar web browser saat itu.

Selang beberapa bulan kemudian, **Microsoft** yang melihat kepopuleran *JavaScript*, memperkenalkan web browser **Internet Explorer 3** dengan **JScript**. *JScript* adalah penamaan lain dari *JavaScript*. Hal ini dilakukan *Microsoft* karena

JavaScript merupakan merk dagang yang dimiliki oleh *Sun* dan *Netscape*. Sehingga *Microsoft* terpaksa mencari nama lain untuk versi *JavaScript* mereka.

Selain memiliki fitur yang mirip, *JScript* juga menambahkan beberapa fitur tersendiri, sehingga *JavaScript* dan *JScript* tidak sepenuhnya kompitable.

Web Browser *Internet Explorer 3* yang dirilis microsoft pada tahun 1996 ini adalah awal dari kemunduran *Netscape Navigator*, karena microsoft merilis Internet Explorer 3 secara gratis dan sebagai software bawaan dari Sistem Operasi *Windows*. Akan tetapi, keputusan *Microsoft* menambahkan fitur *JScript* merupakan langkah besar dalam perkembangan *JavaScript*.



Perkembangan Penggunaan Web Browser Netscape Navigator (source: wikipedia)

Perubahan Menjadi Nama Standar: ECMAScript

Implementasi *JScript* di dalam Internet Explorer membuat kalangan programmer bingung, karena terdapat 2 versi JavaScript: *JavaScript* di *Netscape Navigator* and *JScript* pada *Internet Explorer*. Versi *JavaScript* juga memiliki 2 versi, yakni versi 1.0 dan 1.1. Hal ini semakin menambah kerumitan dalam pembuatan program. Permasalahan terjadi karena ketiga versi JavaScript tersebut memiliki perbedaan fitur.

Kejadian ini sama seperti yang dialami oleh HTML dan CSS, dan kalangan programmer sepakat bahwa diperlukan sebuah standarisasi untuk *JavaScript*.

Pada pertengahan tahun 1997, *JavaScript 1.1* diajukan ke badan standarisasi Eropa: **European Computer Manufacturers Association (ECMA)** untuk membuat sebuah standar bahasa pemograman script web browser. Atas dasar ini, dibentuklah sebuah komite dengan anggota yang terdiri dari programmer dari berbagai perusahaan internet pada saat itu, seperti *Netscape, Sun, Microsoft, Borland, NOMBAS* serta beberapa perusahaan lain yang tertarik dengan perkembangan *JavaScript*.

Komite standarisasi ini menghasilkan bahasa pemograman yang disebut **ECMAScript**, atau secara formal disebut **ECMAScript -262**. 1 tahun berikutnya,

badan standarisasi **ISO (International Organization for Standardization)** juga mengadopsi *ECMAScript* sebagai standar. Sejak saat itu, semua web browser menjadikan *ECMAScript* sebagai standar acuan untuk *JavaScript*.

ECMAScript terus dikembangkan hingga mencapai versi 3 pada tahun 1999. Berita baiknya, hampir semua web browser saat itu, terutama *Microsoft Internet Explorer 5.5* dan *Netscape 6* telah mendukung *ECMAScript-262 versi 3*. Namun berita buruknya, masing-masing web browser menerapkan standar dengan sedikit berbeda, sehingga masih terdapat kemungkinan tidak kompitable.

ECMAScript versi 5

ECMA-262 versi 5 dirilis pada 2009. *ECMAScript versi 4* sengaja dilompati karena beberapa alasan ketidakcocokan proposal yang diajukan. ECMA-262 versi 5 inilah yang saat ini menjadi versi paling stabil dan terdapat di mayoritas web browser modern seperti *Internet Explorer, Google Chrome, Firefox, Opera, dan Safari.*

Akan tetapi, perbedaan implementasi *ECMAScript* tetap ada di dalam web browser. Biasanya perbedaan ini terkait dengan fitur-fitur tambahan. Salah satu cara programmer untuk mengatasi masalah ini yaitu dengan mendeteksi web browser yang digunakan user, lalu menjalankan fungsi yang dirancang secara spesifik untuk web browser tersebut. Proses ini dikenal sebagai **browser sniffing**, dan bukan sesuatu yang menyenangkan.

Kabar baiknya, sekarang banyak terdapat library *JavaScript* yang dirancang untuk melapisi perbedaan *ECMAScript* ini, salah satunya adalah **jQuery**. *jQuery* menyediakan fungsi otomatis dalam mengatasi perbedaan implementasi *ECMAScript* di dalam web browser.

ECMAScript 6

ECMAScript 6 atau **ES6** atau **ECMAScript 2015** dirilis pada bulan Juni 2015. Cukup banyak penambahan baru pada versi ini, sebagian besar merupakan fitur lanjutan untuk membuat aplikasi yang memiliki kompleksitas tinggi, seperti penggunaan JavaScript di server menggunakan **node.js**.

Diantara fitur tersebut adalah: iterator baru, *python-style generator*, *arrow function*, binary data, *typed arrays*, *collections* (maps, sets and weak maps), *promises* untuk membuat *asynchronous* programming, penambahan function untuk tipe data number dan math, reflection, serta *proxies* (metaprogramming untuk virtual objects dan wrappers).

Mulai dari ECMAScript 6 dan selanjutnya, penamaan ECMAScript akan menggunakan nama tahun saat standar tersebut dirilis, seperti ECMAScript 2015, ECMAScript 2016, dst. Banyak perdebatan mengenai pilihan nama ini, sehingga masih sering disebut sebagai ECMAScript 6.

ECMAScript 7

ECMAScript 7 atau nama resminya: **ECMAScript 2016**, diselesaikan pada Juni 2016. Fitur baru termasuk exponentiation operator (**) dan

Array.prototype.includes. Karena relatif baru, belum banyak web browser yang menerapkan ECMAScript 7.

ECMAScript atau JavaScript?

ECMAScript adalah versi standar dari *JavaScript*. Namun karena kepopuleran JavaScript, Hampir semua kalangan dan programmer menyebut *ECMAScript* dengan sebutan umum: *JavaScript*.

Merk dagang *JavaScript* saat ini dimiliki oleh perusahaan **Oracle** (yang mengakuisisi *Sun Microsystem* beberapa tahun lalu), Namun anda juga akan mendengar versi **JavaScript 1.5** atau **JavaScript 1.8**. Versi *JavaScript* ini adalah versi yang diadopsi oleh **Mozilla Firefox** (yang merupakan '*reingkarnasi'* dari *Netscape*). *JavaScript 1.5* sebenarnya adalah *ECMAScript 3*. Dan JavaScript 1.8 merupakan versi *ECMAScript* dengan beberapa penambahan internal oleh Mozilla.

Apa yang dimaksud dengan ECMAScript Engine (JavaScript Engine)?

Jika anda membaca perkembangan tentang *JavaScript*, maka selain versi *ECMAScript* yang digunakan, terdapat istilah **JavaScript Engine** atau dalam bahasa standarnya: **ECMAScript Engine**.

JavaScript Engine adalah mekanisme internal yang dimiliki oleh web browser. *JavaScript Engine* dapat diumpamakan dengan *compiler* dalam bahasa pemograman lain, yakni algoritma yang digunakan untuk menjalankan *JavaScript*. Semakin cepat sebuah web browser menjalankan *JavaScript* akan semakin baik.

Biasanya disetiap rilis baru web browser seperti *Google Chrome, Internet Explorer*, maupun Mozilla Firefox, juga diikuti rilis terbaru *JavaScript Engine* yang menawarkan kecepatan lebih baik.

V8 adalah nama JavaScript Engine untuk Google Chrome, **SpiderMonkey** untuk Mozilla Firefox, dan Chakra untuk Internet Explorer. Daftar lengkap tentang JavaScript engine ini dapat dilihat di
http://en.wikipedia.org/wiki/List_of_ECMAScript_engines

c. Cara Menjalankan Kode Program JavaScript

Aplikasi Untuk Menjalankan JavaScript

JavaScript merupakan *bahasa script* yang berjalan pada *web browser*, sehingga program yang dibutuhkan untuk menjalankan *JavaScript* hanyalah sebuah aplikasi **text editor** dan sebuah **web browser** seperti *Google Chrome* atau *Mozilla Firefox*.

Untuk aplikasi *text editor*, anda bisa menggunakan aplikasi notepad bawaan Windows, atau menggunakan aplikasi khusus *text editor*. Salah satu aplikasi text editor sederhana yang saya gunakan adalah **Notepad++** yang ringan dan bersifat gratis. Pilihan *web browser* bukan sesuatu yang mutlak. Anda bebas menggunakan aplikasi web browser kesukaan. Terdapat beberapa jenis aplikasi web browser populer yang bisa diinstall dengan gratis, anda bahkan bisa menginstall seluruhnya

Cara Menjalankan kode JavaScript

Jika aplikasi *Notepad++* dan web browser telah tersedia, saatnya mencoba menjalankan aplikasi **JavaScript** pertama anda.

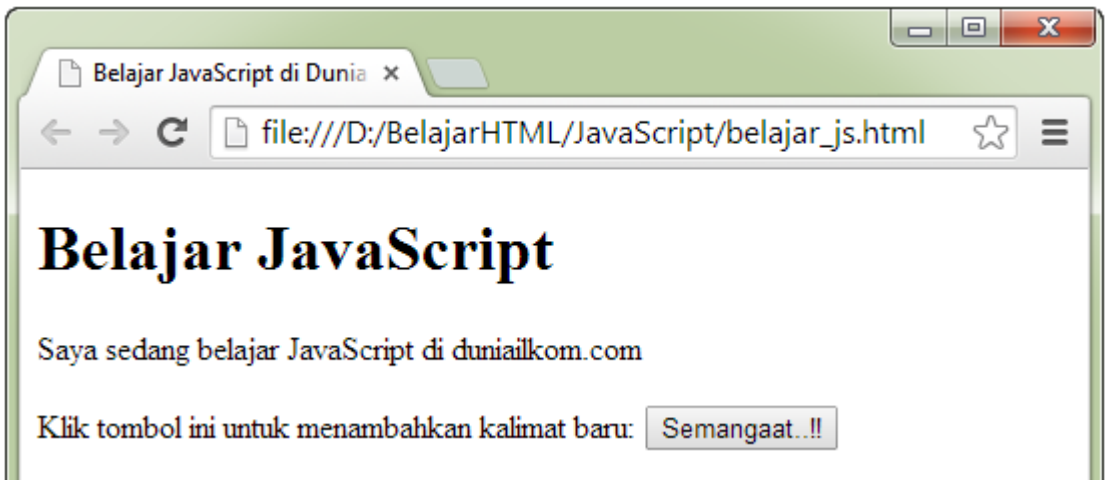
Cara penulisan *JavaScript* mirip dengan penulisan bahasa pemograman web lainnya seperti **PHP** dan **CSS**, yakni dengan menyisipkan kode JavaScript di dalam HTML.

Silahkan buka aplikasi text editor, lalu ketikkan kode berikut:

```
<!DOCTYPE html>
<html>
<head>
<title>Belajar JavaScript di Jti.polije</title>

<script type="text/javascript">
function tambah_semangat()
{
   var a=document.getElementById("div_semangat");
   a.innerHTML+="<p>Sedang Belajar JavaScript,
Semangat...!!!</p>";
}
</script>

</head>

<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di jti.polije.com </p>
Klik tombol ini untuk menambahkan kalimat baru:

<button id="tambah"
onclick="tambah_semangat()">Semangaat..!!</button>

<div id="div_semangat"></div>
</body>
</html>
```

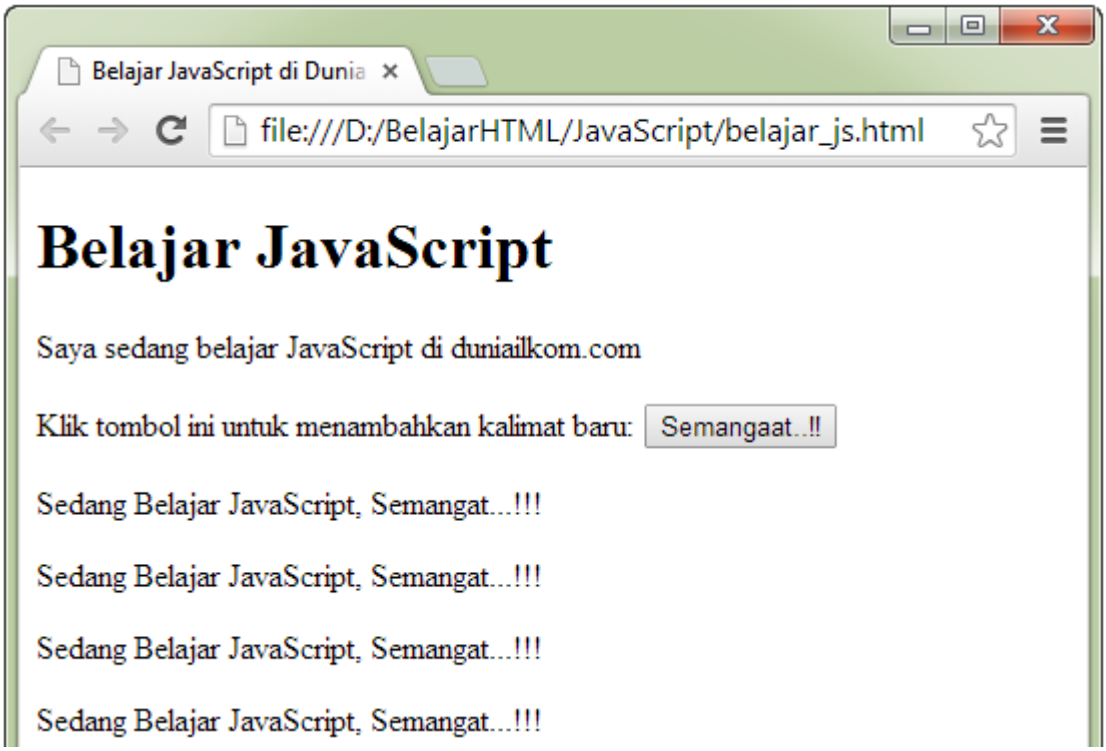
Simpanlah kode HTML diatas dengan nama: ***belajar_js.html***. Folder tempat anda menyimpan file HTML ini tidak menjadi masalah, karena kita tidak perlu meletakkannya dalam folder web server seperti file **PHP**. Saya membuat sebuah folder baru di ***D:\BelajarHTML\Javascript***. Savelah di folder tersebut.

Perhatikan bahwa nama file dari contoh JavaScript kita berakhiran **.html**, karena pada dasarnya kode tersebut adalah kode HTML yang '*disiipkan'* dengan **JavaScript**.



Untuk menjalankan file tersebut, sama seperti HTML biasa, kita tinggal double klik ***belajar_js.html*** dan hasilnya akan tampil di dalam web browser.

Untuk menguji kode *JavaScript* yang telah dibuat, silahkan klik tombol *"Semangaat..!!"* beberapa kali, dan kalimat baru akan ditambahkan di akhir halaman web kita.



Terlepas dari kode ***JavaScript*** yang saya tulis diatas (kita akan mempelajarinya dalam tutorial-tutorial selanjutnya), kode tersebut pada dasarnya berfungsi untuk menambahkan sebuah kalimat kedalam halaman web setelah halaman web tampil di *web browser*.

Fitur inilah yang membuat ***JavaScript*** menawarkan kelebihannya, dimana kita bisa merubah apapun yang terdapat dalam halaman web saat web telah dikirim ke *web*

browser. Bahkan dengan men-klik sebuah tombol, kita bisa mengganti seluruh isi halaman web tanpa harus berpindah halaman.

d. Cara Menampilkan Error JavaScript

Kemudahan untuk menjalankan **JavaScript** hanya dengan *web browser*, memberikan permasalah tersendiri untuk programmer. Biasanya dalam membuat program adakalanya kita melakukan kesalahan penulisan program, seperti salah penulisan fungsi, atau lupa menambahkan tanda ";" sebagai penutup baris. Biasanya pesan kesalahan akan langsung ditampilkan, dan kita tinggal melakukan koreksi.

Akan tetapi, pesan kesalahan (*error)* untuk *JavaScript* tidak langsung ditampilkan *web browser*. **Web browser** pada dasarnya adalah aplikasi untuk menampilkan halaman web, dan secara *default* web browser "*menyembunyikan*" permasalahan *coding* halaman web yang ditampilkan. Hal ini berguna untuk pengguna awam yang pasti akan bingung melihat pesan-pesan error dari sebuah halaman web.

Khusus untuk programmer, kita butuh hal sebaliknya, yaitu agar *web browser* dapat menampilkan pesan kesalahan dari program yang kita buat.

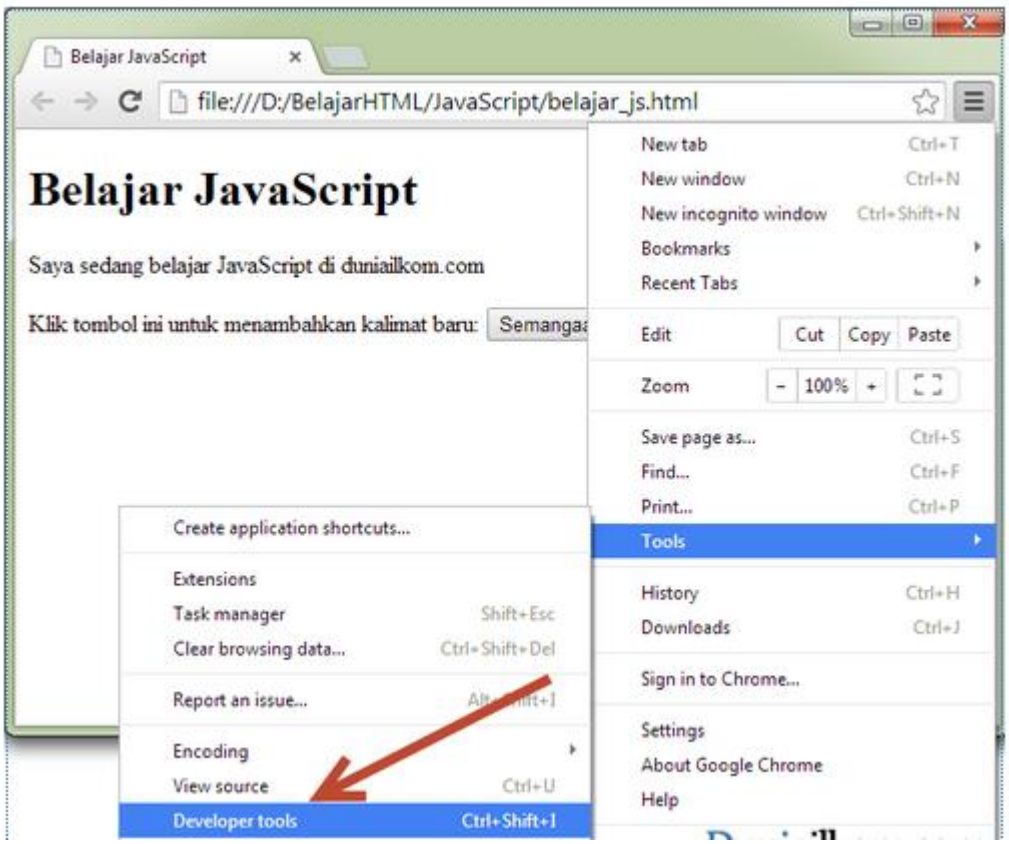
Beberapa tahun yang lalu, untuk keperluan debugging ini kita harus menggunakan plugin tambahan yang diinstall ke dalam web browser. Namun saat ini, web browser telah menyediakan fitur bawaan untuk menampilkan kesalahan dan proses debugging untuk programmer web, terutama **JavaScript**.

Menu Developer Tools dalam Google Chrome

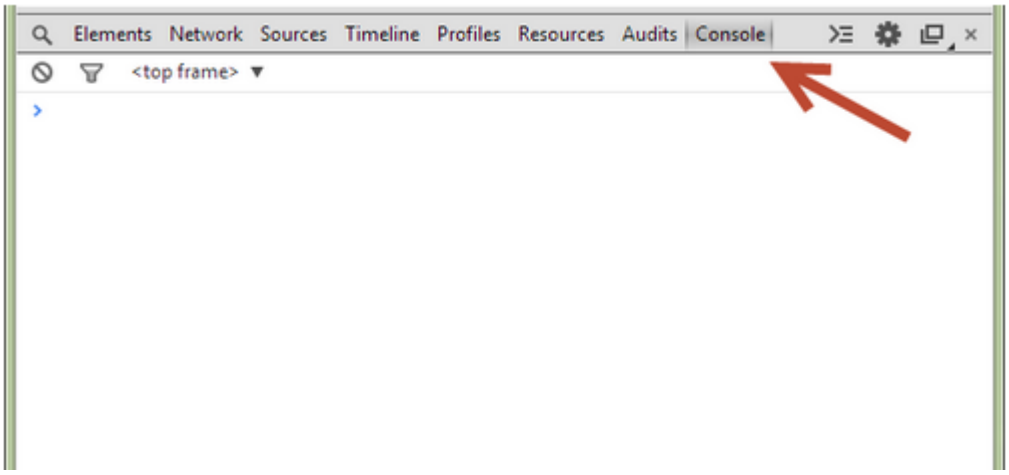
Jika anda menggunakan **Google Chrome**, pada web browser ini terdapat fitur yang dinamakan **Developer Tools** dan **JavaScript Console**. Sesuai dengan namanya, **Developer Tools** adalah fasilitas yang dirancang untuk pengembangan web. Dengan fitur ini, kita bisa menampilkan pesan kesalahan **JavaScript, HTML, CSS** dan melihat efeknya secara *real time*.

Untuk mengakses fitur ini, silahkan klik tombol setting *Google Chrome* yang terletak di kanan atas, cari menu **Tools**, lalu pilih salah satu *Developer Tools* atau *JavaScript Console*. **Developer Tools** bisa juga dibuka dengan shortcut tombol keyboard: **ctrl+shift+i**. Untuk kenyamanan, anda dapat menghapalkan tombol shortcut tersebut karena kita akan sering menggunakan fitur *Developer Tools* ini.

Berikut adalah tampilan menu pada Google Chrome:



Saat menu *Developer Tools* ditampilkan, anda akan melihat banyak tab-tab yang disediakan untuk keperluan *debugging*, namun untuk saat ini kita hanya fokus pada **tab console**. Pada *tab console* inilah pesan error *JavaScript* akan ditampilkan.



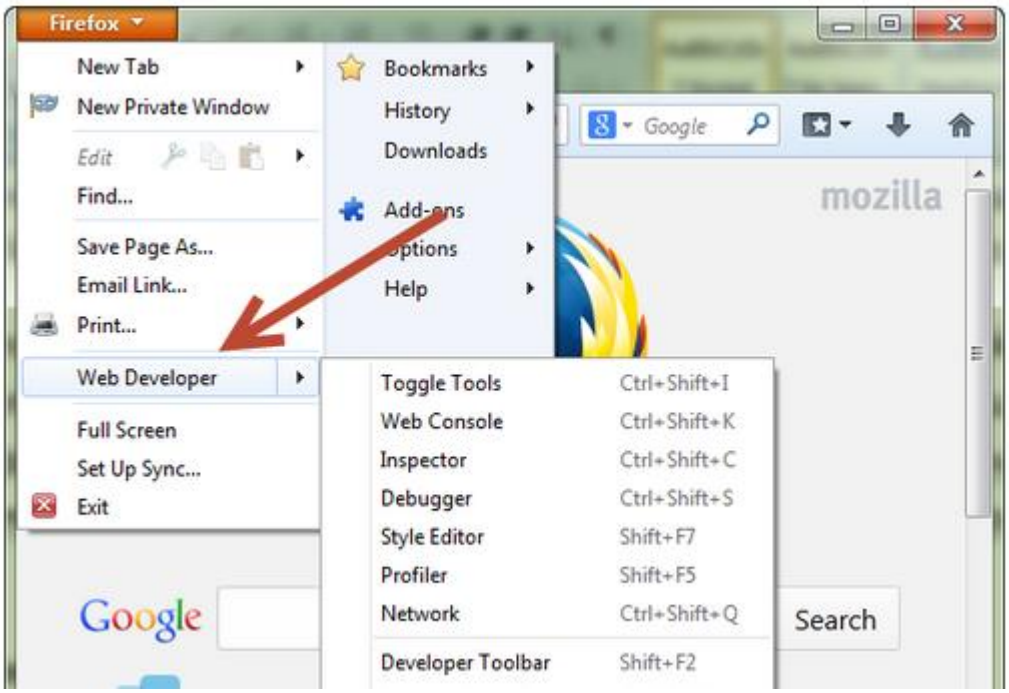
Untuk mengujinya, silahkan edit contoh halaman ***belajar_js.html*** kita sebelumnya. Hapus beberapa huruf dari kode *JavaScript* yang ada, lalu jalankan dengan jendela *Developer Tools* dalam keadaan terbuka, maka anda bisa melihat *error* program ditampilkan di tab console, beserta baris dimana kode tersebut error.

Menu Web Developer dalam Mozilla Firefox

Selain *Google Chrome*, **Mozilla Firefox** merupakan salah satu web browser populer lainnya. Sama seperti *Google Chrome*, secara default bawaan *Firefox* juga tidak menampilkan pesan *error JavaScript*. Untuk menampilkannya, kita harus mengakses jendela **Web Developer**.

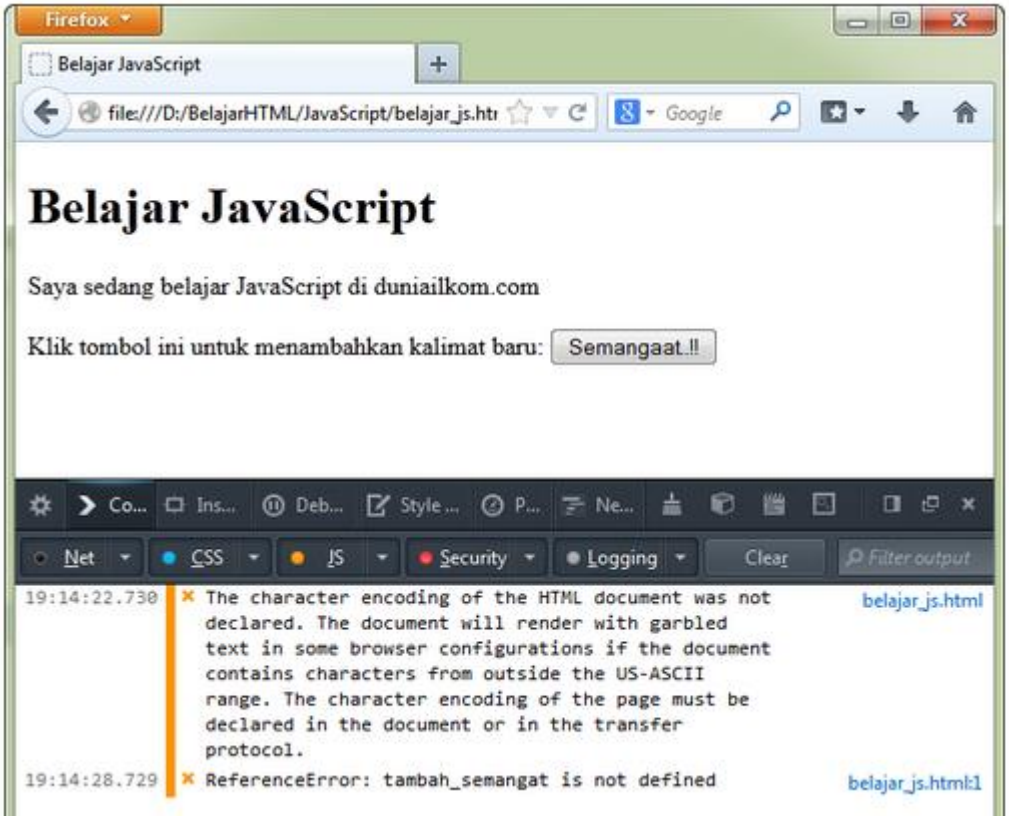
Untuk memunculkan jendela **Web Developer**, klik menu *Firefox* di kiri atas web browser, lalu pilih menu **Web Developer**.

Anda juga bisa menggunakan shortcut yang sama dengan *Google Chrome*, yakni kombinasi tombol keyboard: **ctrl+shift+i**. Berikut adalah tampilan pilihan menu dalam Firefox:



Menu pada **Web Developer** lumayan lengkap, namun kita hanya akan fokus pada pilihan **console**, karena pada menu inilah error *JavaScript* akan ditampilkan.

Sama seperti pada menu *Developer Tools* bawaan **Google Chrome**, cobalah menghapus beberapa baris kode program, dan jalankan dengan Firefox untuk melihat tampilan kode error.



Setiap aplikasi web developer menyediakan fitur untuk menampilkan *error JavaScript*. Namun masing-masing memiliki settingan "*error*" yang ditampilkan. Misalnya untuk *Firefox Web Developer*, akan menampilkan *error* jika *character set* tidak didefenisikan pada header HTML, namun pesan error ini tidak tampil di *Google Chrome*.

Untuk menghilangkan pesan error character set dari *Firefox Web Developer*, kita tinggal menambahkan baris **meta tag** dalam bagian header HTML seperti berikut ini:

```
1  <html>
2  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /
3  <head>
```

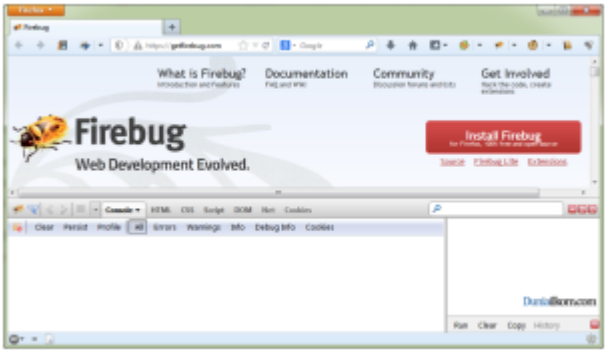
Beberapa *error* juga tidak berdampak "*serius*" dalam menjalankan *JavaScript*. Jika anda memeriksa kode error yang ada dalam situs **duniailkom**, anda akan menemukan banyak error, namun kode **JavaScript** yang ada tetap bisa berjalan.

Akan tetapi, khusus dalam tahap pembelajaran, sedapat mungkin kita menghindari error kode program ini.

Web Developer "must-have" Plugin: Firebug

Sebelum web browser menyediakan menu web developer, **Plugin Firebug** merupakan aplikasi plugin yang harus diinstall oleh *web programmer*. Plugin ini menawarkan fasilitas lengkap untuk menulis program, *debugging*, dan mengedit *JavaScript* pada saat yang bersamaan.

Plugin **Firebug** tersedia untuk *Mozilla Firefox* dan *Google Chrome*. Anda boleh menginstall plugin ini untuk mengetahui fitur-fitur apa saja yang disediakan oleh **firebug**. *Plugin Firebug* merupakan alternatif pilihan untuk menggantikan menu web developer bawaan web browser.



Di dalam tutorial kali ini, kita telah berkenalan dengan **aplikasi web developer** untuk memeriksa dan menampilkan error kode *JavaScript*. Walaupun saya hanya membahas aplikasi *web developer* untuk *Google Chrome* dan *Mozilla Firefox* saja, secara umum setiap web browser modern saat ini telah menyediakan menu yang

serupa. Selain untuk menampilkan error, bagian **console** pada semua fitur *web developer* ini juga berfungsi untuk menampilkan hasil dari kode **JavaScript**.

e. Cara Memasukkan kode JavaScript ke dalam HTML

JavaScript termasuk jenis bahasa *script*, yang digunakan di dalam file **HTML**. Untuk menginput, atau memasukkan kode **JavaScript** ke dalam HTML, *JavaScript* menyediakan 4 alternatif, yaitu:

- Menggunakan tag <script> (internal JavaScript)
- Menggunakan tag <script scr=””> (external JavaScript)
- Menggunakan Event Handler (Inline JavaScript)
- Menggunakan URL (href:=”javascript:”)

Cara Memasukkan JavaScript menggunakan tag <script> (internal JavaScript)

Cara pertama untuk menginput kode *JavaScript* ke dalam halaman HTML adalah dengan menggunakan **tag <script>** secara *internal*. **Internal** disini berarti bahwa kode *JavaScript* ditulis pada halaman yang sama dengan HTML, atau di dalam satu file HTML.

Cara ini merupakan cara yang paling sering digunakan, jika kode JavaScript tidak begitu panjang, dan hanya digunakan di 1 halaman saja. Kode JavaScript yang akan diinput diletakkan diantara *tag pembuka* **<script>** dan *tag penutup* **</script>** seperti berikut ini:

```
<script>
//kode javascript diletakkan disini
</script>
```

Tag **<script>** akan memberitahu web browser bahwa kode diantara tag tersebut bukanlah HTML, tetapi **JavaScript**.

Dalam beberapa buku atau tutorial **JavaScript**, mungkin anda akan menemukan penggunaan tag **<script>** seperti berikut ini:

```
<script type="text/javascript">
//kode javascript diletakkan disini
</script>
```

Penggunaan atribut **type="text/javascript"** digunakan untuk membedakan *javascript* dengan bahasa script lain seperti **VBScript** yang ditulis sebagai **type="text/vbscript".** Namun karena **VBScript** sudah jarang digunakan, hampir semua web browser modern menjadikan **JavaScript** sebagai bahasa *default*, sehingga anda tidak perlu menulis **type="text/javascript".** Tetapi juga tidak salah jika anda ingin menegaskan penggunaan JavaScript dengan menuliskannya secara langsung.

Pada halaman web yang lama, kadang anda juga akan menemukan penggunaan atribut **language** sebagai pengganti atribut **type** seperti berikut ini:

```
<script language="text/javascript">
//kode javascript diletakkan disini
</script>
```

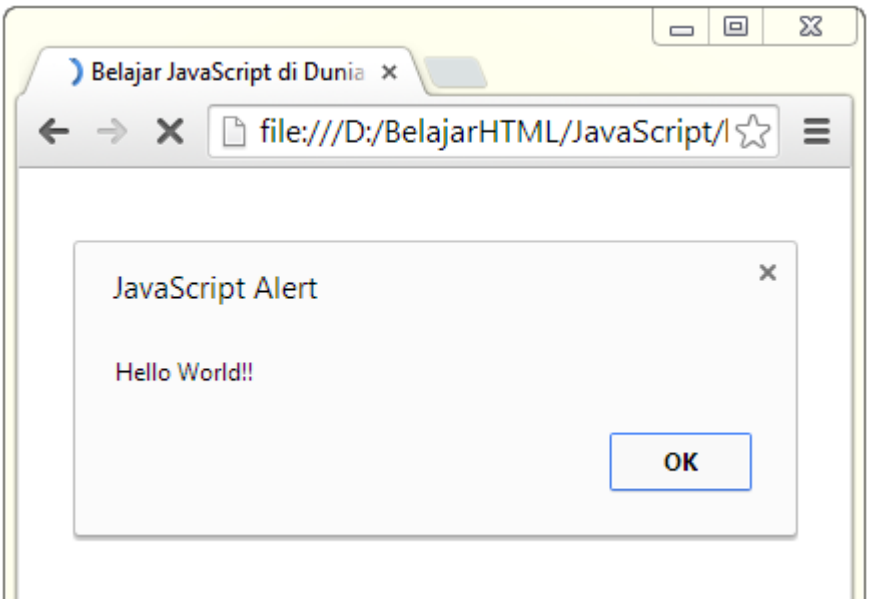
Atribut *language* sudah dianggap usang (*deprecated*), dan disarankan untuk tidak digunakan lagi.

Sebagai contoh cara penginputan JavaScript dengan **tag <script>**, berikut adalah kode HTMLnya:

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<head>
<title>Belajar JavaScript di Jti.polije</title>

<script>
alert("Hello World!!");
</script>

</head>

<body>
<h1>Belajar JavaScript</h1>
<p>Saya sedang belajar JavaScript di jti.polije.com</p>
<p>Belajar Web Programming di Jti.polije.</p>
</body>
</html>
```

Dalam contoh diatas, saya meletakkan **tag <script>** di dalam **tag** <head> dari HTML (pada baris ke 7). **Tag <script>** tersebut berisi kode JavaScript: **alert("Hello World!!");.** *alert()* adalah fungsi dalam JavaScript yang akan menampilkan pesan ke dalam web browser. Fungsi ini sering digunakan dalam proses pembuatan program *JavaScript* untuk menampilkan output sederhana. Fungsi *alert* membutuhkan 1 inputan (argumen) bertipe **String**. Kita akan membahas cara penulisan fungsi dan tipe-tipe data JavaScript pada tutorial-tutorial selanjutnya.

Jika anda menjalankan program diatas, maka di dalam web browser akan tampil hasil seperti berikut ini:



Cara Memasukkan JavaScript Menggunakan tag <script src=" "> (external JavaScript)

Cara atau metode kedua untuk menginput kode **JavaScript** ke dalam halaman **HTML** adalah dengan memindahkan kode *JavaScript* ke dalam sebuah file terpisah, lalu '*memanggilnya'* dari **HTML**. Cara ini sangat disarankan karena akan memberikan banyak keuntungan dan fleksibilitas dalam membuat program *JavaScript*.

Sebuah file **JavaScript** disimpan dalam ekstensi **.js**, seperti: **kode.js, register.js,** atau **kodeku.js.** Dari halaman HTML, kita memanggilnya menggunakan **tag <script>** dengan *atribut* ***src***. Atribut **src** berisi alamat dari file *javascript* tersebut, seperti berikut ini:

```
<script src="kode_javascript.js"></script>
```

Perhatikan bahwa **tag <script>** tetap ditutup dengan tag penutup **</script>,** atau anda bisa membuatnya menjadi *self closing tag* seperti berikut ini:

```
<script src="kode_javascript.js" />
```

Penamaan file **JavaScript** dengan akhiran **.js** hanyalah sebuah kesepakatan di kalangan programmer. Anda bisa membuat akhiran atau extension apapun, seperti: *kode_javascript.aku*, atau *kode_javascript.duniailkom*, asal pada saat pemanggilan dalam **tag <script>** harus sesuai dengan nama file tersebut, seperti **<script src="kode_javascript.duniailkom"></script>**

Namun agar lebih nyaman dan tidak membuat bingung, sebaiknya tetap mengikuti kesepakatan dengan menggunakan akhiran **.js**.

Sebagai contoh program, saya akan menampilkan **alert "Hello World!!"** seperti kode program sebelumnya, namun kali ini saya akan memisahkannya menjadi sebuah file tersendiri. Kode JavaScript tersebut akan dipindahkan kedalam file **kode_javascript.js** dengan isi file sebagai berikut:

```
alert("Hello World!!")
```

Ya, hanya 1 baris itu saja, dan savelah pada folder yang sama dengan tempat kode HTML akan dijalankan dengan nama file **kode_javascript.js.** Lalu pada kode program HTML, kita akan menjalankan file javascript tersebut sebagai berikut:

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<head>
<title>Belajar JavaScript di Jti.polije</title>

<script src="kode_javascript.js"></script>

</head>

<body>
<h1>Belajar JavaScript</h1>
<p>Saya sedang belajar JavaScript di jti.polije.com</p>
<p>Belajar Web Programming di Jti.polije.</p>
</body>
</html>
```

Perhatikan bahwa di dalam file **kode_javascript.js** saya langsung menuliskan perintah **alert**, dan dipanggil oleh tag <**script**> pada baris ke 7 contoh file HTML diatas.

Alamat dari file javascript bisa berupa *alamat relatif* seperti contoh diatas, atau bisa juga *alamat absolut* seperti www.duniailkom.com/kode_javascript.js. Aturan penulisan alamat ini sama seperti atribut **src** HTML lainnya. Perbedaan antara alamat *relatif* dan *absolut* telah dibahas pada Tutorial HTML cara membuat link HTML.

Cara Memasukkan JavaScript Menggunakan Event Handler (Inline JavaScript)

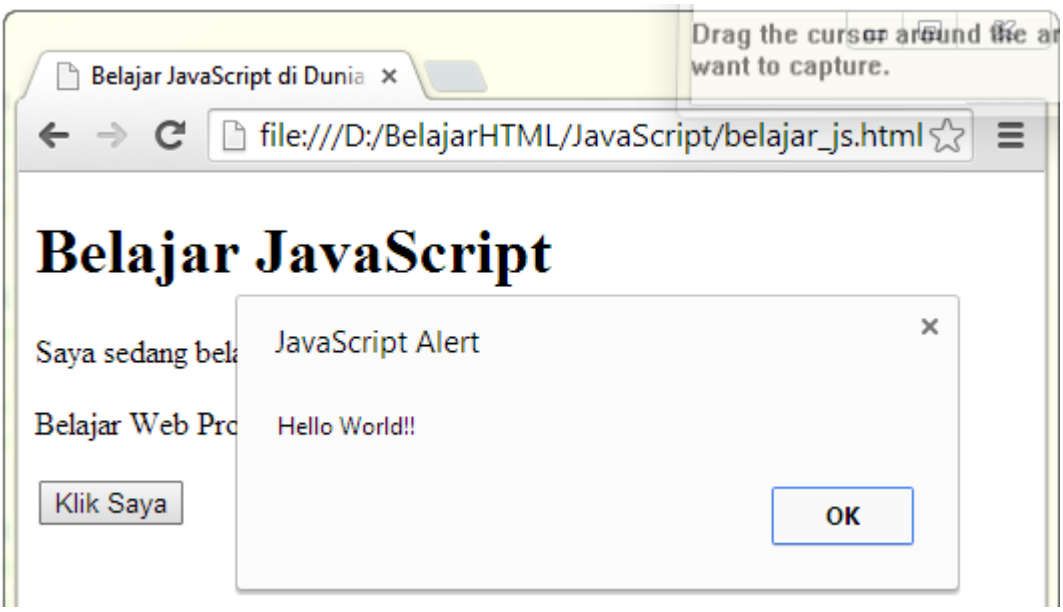
Cara ketiga untuk menjalankan JavaScript adalah dengan memanggilnya menggunakan **Event Handler** dari dalam *tag HTML*.

Konsep *Event Handler* akan kita pelajari secara khusus pada tutorial terpisah, namun secara sederhananya, **event handler** adalah pemanggilan kode javascript ketika 'sesuatu' terjadi dalam **tag HTML**.

Sesuatu disini maksudnya ketika sebuah element dalam HTML di *klik*, di *klik kanan*, di arahkan *mouse*, dan lain-lain. *Event handler* di dalam **JavaScript** ditulis dengan penambahan kata **on**. Sehingga jika sebuah tombol di-klik, maka disebut sebagai **onclick**, jika mouse berada diatas element disebut sebagai **onmouseover**, dan lain-lain.

Sebagai contoh, ketika sebuah tombol di-klik, maka kita bisa menampilkan **alert("Hello World!!").** Berikut adalah contoh kode programnya:

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<head>
<title>Belajar JavaScript di Jti.polije</title>
</head>

<body>
<h1>Belajar JavaScript</h1>
<p>Saya sedang belajar JavaScript di jti.polije.com</p>
<p>Belajar Web Programming di Jti.polije.</p>

<button onclick="alert('Hello World!!')">Klik Saya

</body>
</html>
```



Perhatikan pada baris ke-13 dari contoh kode diatas, yaitu pada penulisan **tag <button>.** Di dalam tag tersebut, saya menambahkan **onclick=”alert(‘Hello World!!’)”**, ini adalah kode *JavaScript* yang diinput melalui metode *event handler*.

Cara penginputan kode JavaScript dengan menggunakan **Event Handler** seperti ini, walaupun praktis namun tidak disarankan, karena kita mencampurkan *JavaScript* dengan *HTML*. Dan jika kode **JavaScript** agak panjang, akan menyulitkan untuk memisahkan kode HTML dengan JavaScript.

Hasil yang didapat menggunakan *event handler* diatas, sebaiknya dipindahkan ke dalam **tag *<script>***.

Di dalam pemograman **Javascript**, ada istilah yang disebut ***Unobtrusive JavaScript***. *Unobtrusive JavaScript* adalah filosofi atau paradigma dalam pemograman yang berpendapat bahwa *content* (**HTML**) sedapat mungkin harus terpisah dari *behavior* (**JavaScript**) agar mudah dalam pemeliharaan dan lebih *fleksibel*. Konsep ini dapat dibaca pada http://en.wikipedia.org/wiki/Unobtrusive_JavaScript.

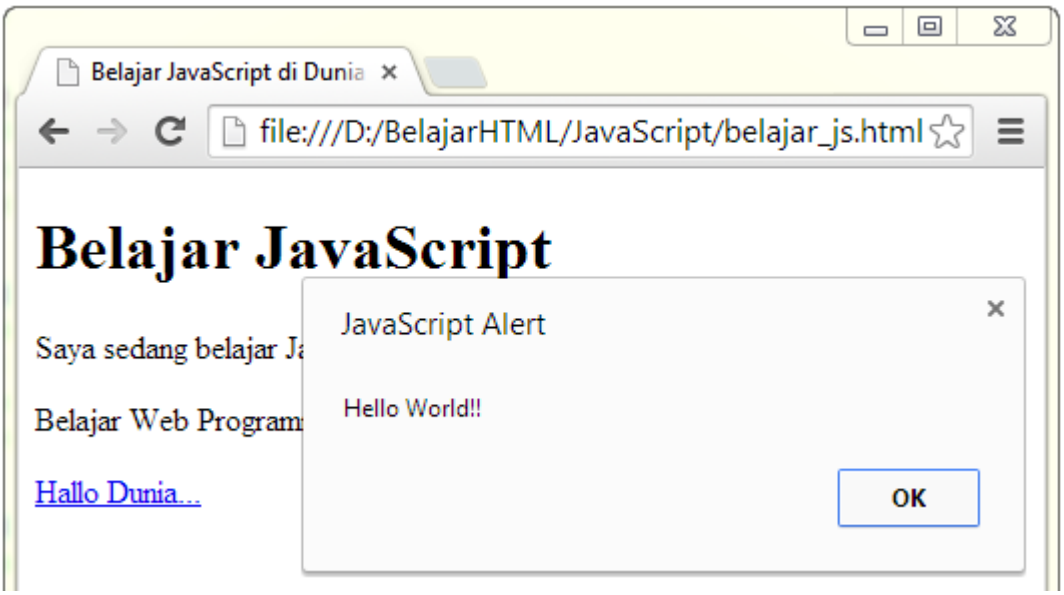
Cara Memasukkan JavaScript Menggunakan URL (href:=”javascript:”)

Cara terakhir (dan juga paling jarang digunakan saat ini) adalah dengan menyisipkan JavaScript ke dalam alamat **href** dari tag *HTML*. Cara ini disebut juga dengan **protocol javascript**. Disebut demikian, karena kita mengganti alamat link dari yang biasa menggunakan protocol **http//:** menjadi **javascript:**

Sebagai contoh penggunaannya, perhatikan kode berikut ini:

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<head>
<title>Belajar JavaScript di Jti.polije</title>
</head>

<body>
<h1>Belajar JavaScript</h1>
<p>Saya sedang belajar JavaScript di jti.polije.com</p>
<p>Belajar Web Programming di Jti.polije.</p>

<a href="javascript:alert('Hello World!!')">Hallo Dunia...</a>

</body>
</html>
```

Jika anda menjalankan kode diatas, dan men-klik link **Hallo Dunia…** akan tampil alert **Hello World!!**, yang berasal dari JavaScript**.** Disini kita telah menjalankan *JavaScript* menggunakan ***protocol javascript***.



Cara menjalankan *JavaScript* seperti ini berasal dari awal kemunculan javascript, dan sudah banyak ditinggalkan.

Anda juga bisa menggunakan konsep **protocol javascript** ini untuk menjalankan perintah JavaScript tanpa membuat halaman HTML. Caranya adalah dengan menuliskan perintah JavaScript langsung di bagian address bar dari web browser, seperti contoh berikut:

Konsep ini bisa digunakan untuk membuat sebuah aplikasi *javascript* yang bisa disimpan dalam web browser, dan dieksekusi pada saat diperlukan, atau dikenal dengan istilah ***bookmarklet***. Pembahasan tentang bookmarklet akan kita bahas pada tutorial JavaScript lanjutan.

Best Practice JavaScript: gunakan tag <script src="">

Dari ke-4 cara menginput kode **JavaScript**, memisahkan kode *JavaScript* kedalam sebuah file tersendiri (menggunakan **metode <script src="">**) adalah yang paling disarankan. Beberapa keuntungan menggunakan metoda <script src> bila dibandingkan dengan moteda cara memasukkan **JavaScript** lainnya adalah:

- Menyederhanakan halaman HTML dengan memindahkan seluruh kode **JavaScript**, sehingga halaman **HTML** hanya berisi konten saja.
- Sebuah file **JavaScript external** bisa digunakan untuk beberapa halaman HTML, sehingga jika diperlukan perubahan, kita hanya perlu mengedit sebuah file daripada mengubah secara satu persatu halaman HTML tempat *JavaScript* ditulis secara internal.
- Jika file *JavaScript* external digunakan oleh beberapa halaman, file tersebut hanya perlu di *download* oleh web browser pada saat pertama kali saja. Pada saat loading halaman lainnya, web browser cukup mengambilnya dari *browser cache*, sehingga mempercepat loading halaman.

f. Posisi Terbaik Meletakkan kode JavaScipt di dalam HTML

Posisi JavaScript pada di awal HTML (di dalam tag <head>**)**

Pada dasarnya, anda bebas meletakkan kode program **JavaScript** dibagian manapun dalam HTML, selama berada di dalam **tag <script>** (atau melalui *event handler*, walaupun tidak disarankan)**.**

Tag <script> bisa diletakan pada awal HTML (di dalam **tag** <head>**),** di tengah HTML (di dalam **tag** <body>**),** maupun diakhir HTML (sebelum tag penutup </html>). Posisi peletakan ini akan mempengaruhi bagaimana urutan kode JavaScript dijalankan.

Halaman HTML diproses oleh web browser dari atas ke bawah secara berurutan. Hal ini berlaku juga untuk **HTML**, **CSS**, dan **JavaScript**. Kode **JavaScript** yang diletakkan pada awal dokumen (didalam **tag** <head>**),** akan diproses terlebih dahulu sebelum web browser memproses isi dokumen HTML yang terdapat di dalam **tag** <body>.

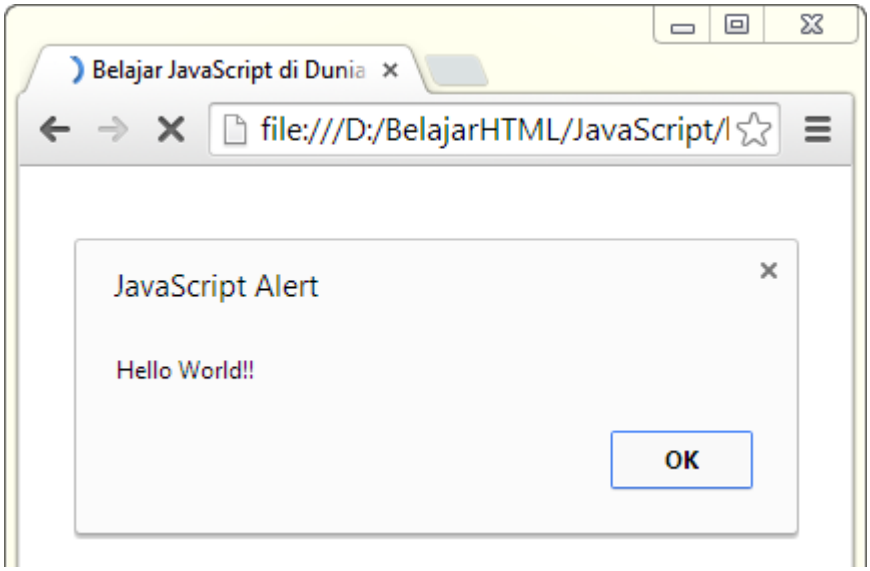
Untuk melihat efeknya,mari kita coba contoh kode HTML dan JavaScript berikut ini:

```
<!DOCTYPE html>
<html>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<head>
<title>Belajar Tag Label HTML</title>

<script>
   alert("Hello World!!");
</script>

</head>
<body>
```

```
<p>Saya sedang belajar JavaScript di Jti.polije</p>
<p>Belajar Web Programming di Jti.polije</p>
</body>

</html>
```

Contoh kode diatas sama persis dengan contoh yang saya jalankan ketika membahas tentang cara men-input kode javascript pada tutorial sebelumnya. Namun kali ini saya akan fokus bagaimana web browser mengeksekusi halaman tersebut.

Dalam contoh diatas, **tag <script>** berisi kode **JavaScript** yang diletakkan pada awal halaman (di dalam **tag** <head>). Jika anda menjalankan program diatas, maka di dalam web browser akan tampil hasil seperti berikut ini:



Sebelum anda men-klik tombol **OK** dari kotak dialog **alert**, perhatikan tampilan text HTML dari halaman tersebut. Pada halaman web tidak terlihat tulisan apapun, padahal di dalam kode HTML, saya telah menambahkan 2 kalimat di dalam **tag <p>.** Dan hanya jika anda men-klik tombol OK, baru setelahnya akan tampil 2 kalimat tersebut.

Hal ini berarti bahwa jika kode **JavaScript** diletakkan pada posisi **tag** <head>**,** maka kode tersebut akan dijalankan (diproses) sebelum '*isi'* website (**tag** <body>**)** ditampilkan. Sehingga disimpulkan bahwa pemrosesan program dimulai dari atas ke bawah halaman web.

Posisi peletakan **JavaScript** diawal HTML ini , bisa menjadi permasalahan tersendiri dalam membuat kode **JavaScript**.

Pada saat kita menulis program **JavaScript**, sebagian besar kode tersebut digunakan untuk memanipulasi halaman web. Permasalahan muncul jika saat kode JavaScript dieksekusi, halaman atau tag HTML yang akan diproses belum tersedia. Sebagai contoh lain, silahkan jalankan kode program HTML berikut ini:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   var a=document.getElementById("div_semangat");
   a.innerHTML="<p>Sedang Belajar JavaScript,
Semangat...!!!</p>";
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di jti.polije.com </p>
<div id="div_semangat"></div>
</body>
</html>
```

Dalam file HTML diatas, saya membuat beberapa baris kode JavaScript, anda tidak perlu memahami kode tersebut, karena kita akan mempelajarinya nanti. Namun inti dari kode tersebut adalah menambahkan 1 kalimat ke dalam tag HTML yang memiliki **id=”div semangat”**.

Tag HTML dengan **id=”div_semangat”** berada di dalam sebuah **tag div** pada baris ke. Pada tag inilah saya akan mengisi kalimat yang dibuat dengan **JavaScript**.

Jika anda menjalankan halaman tersebut, hasil tampilan pada web browser tidak akan menampilkan kalimat yang saya buat dari JavaScript. Apa yang terjadi?

Untuk melihat kenapa kalimat tersebut tidak tampil, anda bisa melihatnya dari error **JavaScript**. Dari tab **console**, anda bisa melihat 1 buah error, yakni :

```
Uncaught TypeError: Cannot set property 'innerHTML' of null
(pada Google Chrome)
```

atau

```
TypeError: a is null (pada Mozilla Firefox)
```

Kedua web browser tersebut menampilkan ***error*** yang menyatakan bahwa variabel **a** yang digunakan di dalam JavaScript tidak bisa menemukan tag dengan **id="div semangat".**

Hal ini terjadi karena pada saat JavaScript di eksekusi, **tag div** tersebut belum tersedia (belum sempat diproses), karena **JavaScript** di jalankan sebelum **tag** `<body>`. *Tag div* baru tersedia ketika web browser telah selesai menjalankan **JavaScript**.

Salah satu solusi untuk permasalahan ini adalah dengan memindahkan kode JavaScript ke akhir halaman HTML.

Posisi JavaScript pada di akhir HTML (sebelum tag `</html>`)

Dari contoh sebelumnya, kita dapat melihat permasalahan yang terjadi jika kode **JavaScript** diposisikan pada awal halaman.

Untuk mengatasi permasalahan tersebut, beberapa programmer menyarankan untuk meletakkan kode JavaScript di akhir halaman (yakni sebelum **tag penutup** `</html>`). Peletakan **tag <script>** diakhir halaman akan memastikan bahwa seluruh tag HTML telah tampil pada web browser sebelum kode **JavaScript**.

Untuk membuktikannya, mari kita ubah kode program sebelumnya, dan memindahkan kode **JavaScript** ke posisi akhir halaman:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>
```

```
</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di jti.polije.com </p>
<div id="div_semangat"></div>
</body>

<script>
   var a=document.getElementById("div_semangat");
   a.innerHTML="<p>Sedang Belajar JavaScript,
Semangat...!!!</p>";
</script>

</html>
```

Dan, seperti yang terlihat, kalimat *"Sedang Belajar JavaScript, Semangat...!!!"* sukses ditampilkan ke dalam web browser hanya dengan memindahkan kode **JavaScript** ke akhir halaman.

Namun sebagian programmer tidak menyukai peletakan JavaScript di akhir halaman seperti ini, karena terkesan kurang '*elegan*'. Untungnya JavaScript menyediakan solusi untuk hal ini.

Solusi Elegan JavaScript : Fungsi Event onload

JavaScript menyediakan solusi untuk programmer untuk tetap meletakkan JavaScript di awal halaman, yakni menggunakan **Event Onload**.

Walaupun saya belum membahas tentang konsep **event**, **Event** di dalam *JavaScript* dapat dibayangkan dengan sebuah '***kejadian***' yang terjadi pada suatu

elemen. Pada pembahasan tentang cara menginput kode **JavaScript**, sekilas saya telah membahas tentang '*kejadian'* ini, yaitu ketika sebuah tombol **di-klik**, atau **di-klik kanan**. *Klik* dan *klik kanan* adalah kejadian (atau **event**) yang terjadi pada sebuah tombol.

Sebuah **event** di dalam javascript tidak harus '*berwujud'* seperti **klik** atau **klik kanan**,namun bisa juga yang *'tidak berwujud'*. Salah satu event yang '*tidak berwujud'* ini adalah event pada saat halaman di tampilkan, atau di dalam bahasa inggris, pada saat halaman **loading**.

Event pada saat halaman ditampilkan (**load**) inilah yang bisa dimanfaatkan untuk mengeksekusi kode **JavaScript**. Kita ingin kode javascript di jalankan ketika seluruh dokumen telah tampil, tanpa memperhatikan pada posisi mana kode **JavaScript** di letakkan. Untuk hal ini, JavaScript menyediakan **event onload**.

Menjalankan **JavaScript** pada **event onload**, akan membuat javascript '*menunggu'* seluruh tag HTML tampil sebelum memprosesnya, walaupun **tag** <**script**> berada di awal halaman (bahkan di posisi manapun).

Berikut adalah kode JavaScript yang diletakkan di awal halaman HTML, dan menggunakan event onload:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
window.onload = function()
{
   var a=document.getElementById("div_semangat");
   a.innerHTML="<p>Sedang Belajar JavaScript,
Semangat...!!!</p>";
}
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di jti.polije.com </p>
<div id="div_semangat"></div>
</body>

</html>
```

Perhatikan cara penulisan event onload pada baris ke 7. **Event onload** adalah salah satu ***method*** (atau fungsi) dari objek ***window*** dalam ***JavaScript*** (anda tidak perlu memahaminya maksudnya, saya akan membahasnya secara lebih dalam pada tutorial khusus *JavaScript* mengenai **event**).

Dengan menggunakan **event onload**, seluruh kode *JavaScript* dapat diletakkan di awal halaman, tanpa perlu khawatir dengan urutan pengeksekusian oleh web browser.

g. Cara Menampilkan Hasil Program JavaScript

Tidak seperti bahasa pemograman **PHP** yang memiliki perintah ***echo*** atau ***print*** untuk menampilkan hasil program ke dalam web browser, **JavaScript** tidak menyediakan perintah sederhana untuk menampilkan hasil program ke dalam web browser. Dalam **JavaScript**, kita membutuhkan beberapa langkah yang agak panjang jika ingin menampilkan hasil ke dalam web browser.

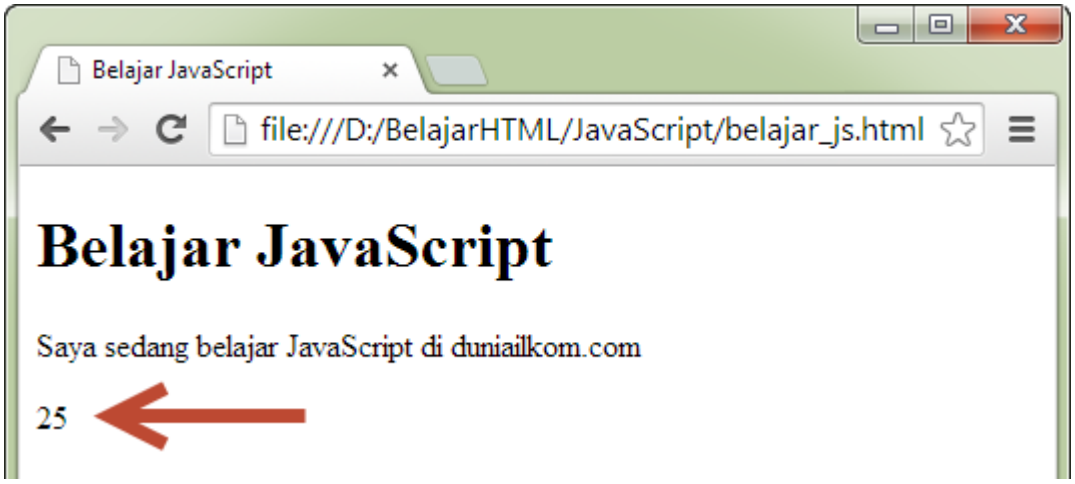
Pertama, kita harus membuat sebuah **tag '*container*'**, atau tag penampung untuk hasil program JavaScript. *Tag container* ini bisa berupa tag HTML apapun, seperti tag paragraf **<p>** atau tag ** .**

Kedua, kita harus mencari elemen '*container'* ini dari JavaScript. JavaScript menyediakan beberapa cara untuk mengakses elemen dalam HTML. Salah satu caranya adalah dengan menggunakan **fungsi** (atau lebih tepatnya: **method**): **document.getElementById("id_continer")**. Fungsi **getElementById** akan mencari elemen HTML yang memiliki atribut **id** yang diinputkan di dalam tanda kurung.

Langkah ***ketiga***, adalah *menginputkan* hasil program kedalam tag '*container'* dengan menggunakan properti **innerHTML**.

Untuk mempermudah pemahaman cara menampilkan hasil program JavaScript ke dalam web browser, berikut adalah contoh program untuk menampilkan hasil penjumlahan:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
window.onload = function()
{
   var hasil;
   hasil = 1+3+5+7+9;
   document.getElementById("tempat_hasil").innerHTML=hasil;
}
</script>

</head>

<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di jti.polije.com </p>
<div id="tempat_hasil">
</div>
</body>
</html>
```



Pada baris ke-7, saya masuk kedalam JavaScript menggunakan **tag <script>.** Pada baris pertama, saya menggunakan metoda **event onload** untuk memastikan bahwa kode JavaScript dijalankan setelah seluruh HTML telah selesai di proses

Pada baris berikutnya, saya membuat *variabel* **hasil** yang akan menampung hasil penjumlahan. Pada baris ke-9 saya melakukan beberapa penjumlahan sederhana, dan menyimpannya ke dalam variabel **hasil**. Pada baris ke-10 inilah saya menampilkan nilai akhir penjumlahan ke dalam tag HTML yang memiliki *id* **tempat_hasil**.

Proses untuk menampilkan hasil program yang agak panjang ini akan kurang praktis dalam pembuatan program **JavaScript** dimana kita akan sering untuk menguji hasil program sebelum masuk ke bagian program berikutnya.

Untuk keperluan ini, kita akan melihat alternatif cara untuk menampilkan hasil program dari JavaScript, yakni dengan **fungsi alert** dan **fungsi console.log**.

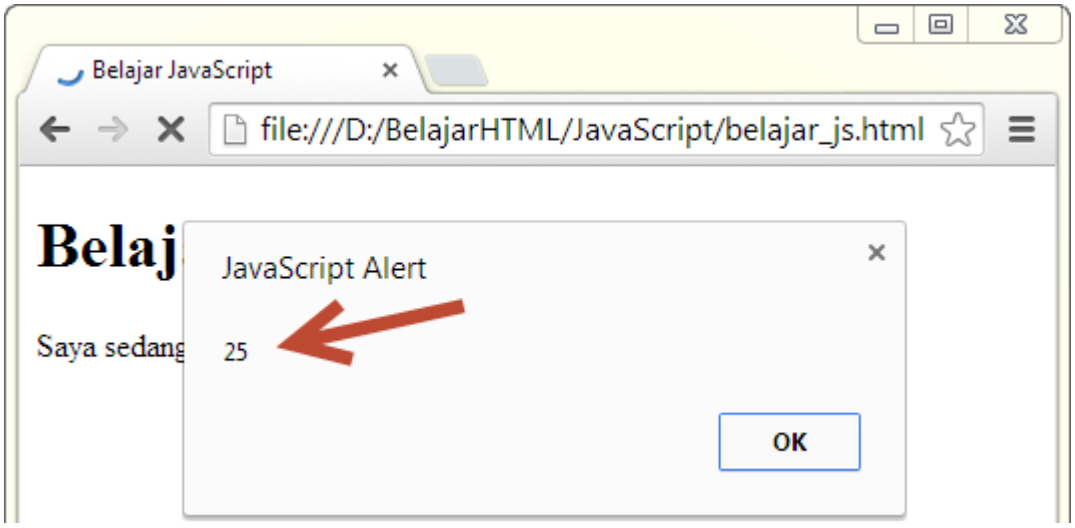
Fungsi Alert untuk Menampilkan Hasil Program JavaScript

Fungsi alert yang telah beberapa kali saya gunakan dalam beberapa tutorial sebelum ini, dan merupakan cara paling sederhana untuk menampilkan hasil program **JavaScript**.

Fungsi alert akan menampilkan '*apapun*' yang diberikan sebagai argumen ke dalam fungsi ini. Setiap output yang ditampilkan akan dikonversi menjadi bentuk text (tipe data **String**). Fungsi alert akan menampilkan hasil **JavaScript** dengan cepat. Berikut adalah contoh kode **JavaScript** sebelumnya jika menggunakan fungsi alert:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
window.onload = function()
```

```
{
   var hasil;
   hasil = 1+3+5+7+9;
   alert(hasil);
}
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di jti.polije.com </p>
</div>
</body>
</html>
```



Fungsi Console.Log untuk Menampilkan Hasil Program JavaScript

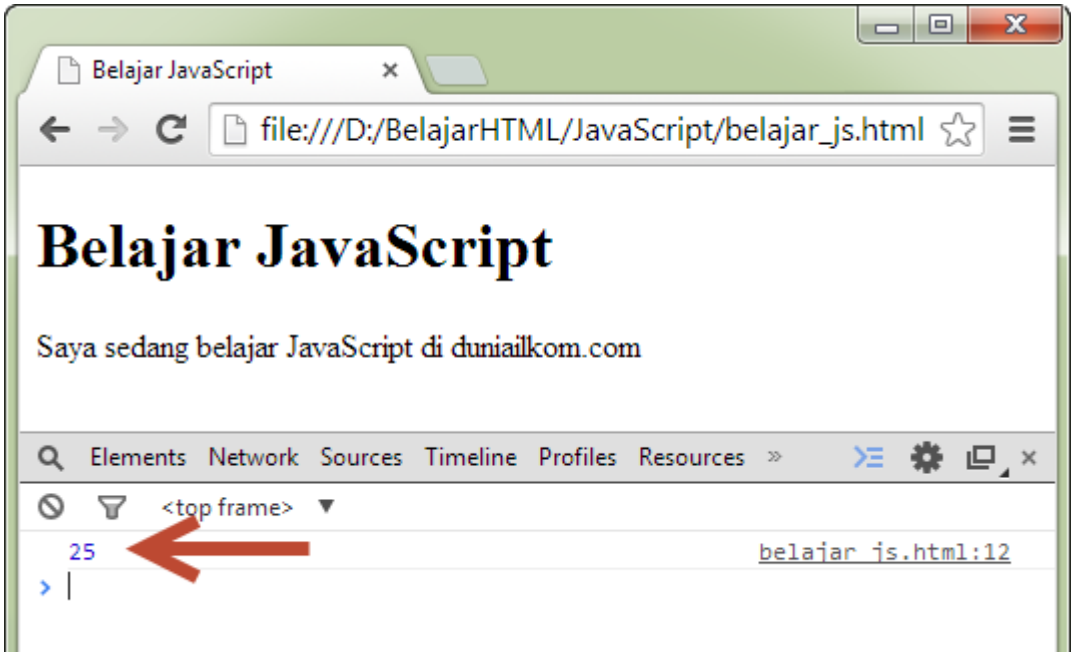
Web browser saat ini telah menyediakan menu khusus untuk programming web, yakni melalui menu **Web Developer**. Menu *web developer* ini menyediakan cara lain (yang lebih '*modern*') untuk menampilkan hasil program **JavaScript** dibandingkan menggunakan **fungsi alert**, yakni dengan fungsi **console.log**.

Fungsi **console.log** akan menampilkan hasil program ke dalam tab *console* pada menu **Web Developer**. Cara penggunaan fungsi ini sama seperti *fungsi alert*, dimana kita hanya butuh menginput hasil yang ingin ditampilkan kedalam argumen fungsi ini. Jika fungsi itu bukan bertipe **String**, maka akan dikonversi menjadi String.

Berikut adalah contoh penggunaan fungsi console.log dalam menampilkan hasil program **JavaScript**:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>
```

```
<script>
window.onload = function()
{
   var hasil;
   hasil = 1+3+5+7+9;
   console.log(hasil);
}
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di jti.polije.com </p>
</div>
</body>
</html>
```



Pilihan apakah menggunakan **fungsi alert** dan fungsi **console.log** tergantung kepada kenyamanan kita sebagai programmer. Dalam *tutorial JavaScript* ini, saya akan lebih banyak menggunakan fungsi **console.log**, karena dibandingkan fungsi **alert**, fungsi **console.log** bisa digunakan untuk menampilkan banyak hasil program sekaligus dan tampilan hasil program akan lebih rapi.

Kedua fungsi ini digunakan hanya dalam proses pembuatan program, atau proses pencarian kesalahan (*debugging*). Akan sangat jarang kita menggunakan fungsi ini dalam situs live. Untuk menampilkan output kepada user di dalam halaman HTML, biasanya menggunakan fungsi lain seperti **document.getElementById** dan **InnerHTML** seperti cara pertama pada awal artikel ini

Cara lain untuk menampilkan hasil dari **JavaScript** adalah menggunakan fungsi **document.write()**. Berikut adalah cara penggunaannya:

```
<!DOCTYPE html>
<html>
<head>
<title>Belajar JavaScript</title>

<script>
window.onload = function()
{
   document.write("Hello duniailkom");
}
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di duniailkom.com </p>
</div>
</body>
</html>
```

Fungsi **document.write** akan menampilkan hasil **JavaScript** ke dalam web browser, tetapi dengan satu ketentuan: fungsi ini akan menghapus seluruh tag HTML yang telah ditulis, dan menggantinya dengan isi dari fungsi ini. Fungsi **document.write** hanya akan berguna jika anda ingin menuliskan seluruh kode HTML hanya melalui **JavaScript**, termasuk tag pembuka HTML seperti **document.write("<!DOCTYPE html>")**, dan seterusnya.

h. Pengertian Core JavaScript dan DOM (Document Object Model)

Pada tutorial JavaScript part 1 – part 7, kita belum membahas cara penulisan kode program JavaScript. Di dalam tutorial-tutorial tersebut, kita mempersiapkan '*lingkungan'* yang harus diketahui untuk dapat dengan mudah memahami dan manjalankan JavaScript.

Mempelajari **JavaScript** pada dasarnya terdiri dari 2 bagian besar, yakni **JavaScript Inti** (sering disebut dengan istilah: **Core JavaScript**), serta **API** yang disediakan oleh *web browser* (yang dikenal dengan istilah **DOM,** singkatan dari **Document Object Model**)..

Pengertian Core JavaScript

Core JavaScript atau **JavaScript inti** adalah istilah yang merujuk kepada '*Bahasa Pemograman JavaScript*'. Pada bagian **Core JavaScript** inilah kita akan belajar tentang aturan pemograman yang umumnya dipelajari, seperti cara

pendefenisian variabel, perbedaan tipe-tipe data, cara pembuatan array, cara penulisan struktur IF, serta cara pembuatan Objek.

Bagian **Core JavaScript** membahas tentang "*bahasa*" (atau **syntax**) dari **JavaScript**. Jika anda pernah menggunakan bahasa pemograman seperti **C++** atau **PHP**, tidak akan terlalu sulit untuk mempelajari aturan penulisan dalam ***JavaScript***.

Perbedaan mendasar antara *JavaScript* dengan dengan bahasa pemograman lain, adalah sifat JavaScript yang lebih berorientasi ke **Objek**. Namun JavaScript '*tidak mengharuskan'* menggunakan objek. Saya menyebutnya '*tidak harus'* karena kita bisa membuat beberapa kode program yang *seolah-olah* tanpa menggunakan objek. Contohnya adalah **fungsi alert**.

Core JavaScript juga merupakan istilah untuk ***JavaScript*** yang tidak terikat dengan '*lingkungannya'* yaitu **web browser**. Selama ini JavaScript hanya dikenal sebagai bahasa tipe ***client-side JavaScript*** yang dijalankan di dalam *web browser*.

Akan tetapi, perkembangan ***JavaScript*** saat ini juga mendukung penggunaanya di sisi server (salah satunya dengan menggunakan **node.js**).

Pengertian DOM (Document Object Model)

Bagian kedua yang akan kita pelajari dalam menguasai *Client-Side* **JavaScript** adalah **DOM** (singkatan dari **Document Object Model**). **DOM** adalah **API** (**Application Programming Interface**) yang disediakan web browser kepada programmer.

Secara sederhananya, **DOM** adalah kumpulan aturan atau cara bagi programmer untuk '*memanipulasi'* apapun yang tampil dalam halaman web. **DOM** tidak terikat dengan *JavaScript*, dan sepenuhnya bukan bagian dari ***JavaScript***. **DOM** yang sama bisa juga diakses dengan bahasa *client-side* lain seperti **JScript**.

Tag atau ***element*** yang ada di dalam HTML diatur di dalam **DOM**. Dengan menggunakan JavaScript, kita bisa memanipulasi seluruh tag HTML ini. Salah satu contoh DOM yang telah kita gunakan adalah fungsi **document.getElementById**.

Fungsi **document.get ElementById** berfungsi untuk mencari sebuah tag HTML berdasarkan **id**. Selain *document.getElementById*, dalam DOM juga disediakan fungsi lain seperti *document.getElementByName*, *document.getElementByClass*, dan lain-lain. Fitur-fitur seperti inilah yang akan kita pelajari pada bagian **DOM**.

Karena sifatnya yang berbeda, tutorial JavaScript ini akan memisahkan **Core JavaScript** dengan **DOM**. Untuk tahap awal kita akan membahas tentang **Core JavaScript**, dan setelah itu baru mempelajari struktur dan aturan **DOM** dari *web browser*.

i. Aturan Dasar Penulisan Kode Program JavaScript

Aturan yang akan kita bahas adalah tentang *case sensitivity*, *whitespace*, cara penulisan komentar, aturan penulisan *identifier*, *reserved keyword*, dan penulisan *semicolon*.

Perbedaan Penulisan Huruf (Case Sensitivity)

Di dalam **JavaScript**, penulisan huruf besar dan huruf kecil dibedakan, atau dalam istilah pemograman bersifat **Case Sensitif**. Hal ini berarti penulisan *variabel* , *keyword*, maupun nama *fungsi* di dalam JavaScript harus konsisten. Variabel **nama**, **Nama**, dan **NAMA** merupakan 3 variabel berbeda. Sedangkan untuk penulisan *keyword* **while**, harus ditulis dengan '**while'**, bukan '**While'** atau '**WHILE'**.

Namun karena **HTML** sendiri tidak bersifat *case-sensitif*, kadang hal ini bisa mendatangkan permasalahan. Contohnya, jika menggunakan **event handler**, di dalam HTML kadang bisa ditulis: **onclick**, **onClick**, atau **OnClick**. Ketiga penulisan ini dibolehkan di dalam **HTML**. Akan tetapi untuk menghindari permasalahan, sebaiknya kita membuat kesepakatan untuk menggunakan huruf kecil untuk semua penulisan *keyword* dan *variabel* di dalam JavaScript.

Penggunaan Karakter Spasi, Enter, dan Tab (Whitespace)

Karakter-karakter *spasi*, *enter*, *tab* dan karakter lain yang '*tidak kelihatan'* (sering dikenal dengan istilah **whitespace**) akan diabaikan pada saat pemrosesan **JavaScript**. Karakter-karakter ini bisa digunakan untuk '*menjorokkan'* (**indent**) kode program agar lebih mudah dibaca.

Cara Penulisan Komentar dalam JavaScript

JavaScript mendukung 2 jenis **cara penulisan komentar**, yakni menggunakan karakater **//** untuk komentar dalam 1 baris, dan karakter pembuka komentar **/*** dan penutup ***/** untuk komentar yang mencakup beberapa baris.
Berikut adalah contoh penulisan komentar di dalam JavaScript:

```
<script>
   // ini adalah komentar dalam 1 baris
   /* Baris ini juga merupakan komentar */ //ini juga komentar

   /* Komentar ini
     mencakup beberapa
     baris
   */

   /*
   * Beberapa programmer
   * menambahkan tanda bintang
   * agar penulisan komentar
   * lebih rapi
   */
</script>
```

Aturan Penulisan Identifier (Variabel dan Nama Fungsi) JavaScript

Di dalam **JavaScript**, *identifier* adalah sebutan untuk *nama*. *Nama* ini bisa terdiri dari nama **variabel**, atau nama dari **fungsi**. Aturan penulisan *identifier* dalam JavaScript adalah :

- Karakter pertama harus diawali dengan **huruf**, **underscore** (_) atau **tanda dollar** ($)
- Karakter kedua dan seterusnya bisa ditambahkan dengan **huruf**, **angka**, **underscore** (_) atau **tanda dollar** ($).

Dari aturan tersebut dapat dilihat bahwa kita tidak bisa menggunakan angka sebagai karakter pertama dari sebuah **variabel** atau nama **fungsi**.

Berikut adalah contoh penulisan nama variabel yang dibolehkan:

```
<script>
   jti.polije
   $ilkom
   v12
   _karakter
   b3l4j4r
</script>
```

Namun karakter berikut tidak bisa digunakan sebagai identifier:

```
<script>
   %jti.polije   // terdapat karakter %
   Suab andi   // terdapat karakter spasi
   4ngka         // diawali dengan angka
   suka#suka     // terdapat karakter #
</script>
```

Kata Kunci (Reserved Keyword) JavaScript

Seperti bahasa pemograman lain, **JavaScript** juga memiliki beberapa kata kunci atau *keyword* yang tidak bisa digunakan sebagai nama *variabel* atau nama dari sebuah *fungsi*. Istilah ini sering disebut sebagai **reserved keyword**.

Reserved keyword merupakan kata kunci yang digunakan **JavaScript** dalam menjalankan fungsinya. **Keyword** di dalam JavaScript adalah sebagai berikut:

```
arguments           encodeURI           Infinity  Number          RegExp
Array               encodeURIComponent  isFinite  Object          String
Boolean             Error               isNaN     parseFloat      SyntaxError
Date                eval                JSON      parseInt        TypeError
decodeURI           EvalError           Math      RangeError      undefined
decodeURIComponent  Function            NaN       ReferenceError  URIError
```

```
abstract      double        goto          native        static
boolean       enum          implements    package       super
byte          export        import        private       synchronized
char          extends       int           protected     throws
class         final         interface     public        transient
const         float         long          short         volatile
```

```
arguments       eval

implements      let             private         public          yield
interface       package         protected       static

class       const       enum        export      extends     import      super

break           delete          function        return          typeof
case            do              if              switch          var
catch           else            in              this            void
continue        false           instanceof      throw           while
debugger        finally         new             true            with
default         for             null            try
```

Sebagian dari nama keyword diatas bisa digunakan dalam situasi khusus (seperti nama dari **method** untuk **objek**), namun sedapat mungkin kita tidak menggunakannya agar tidak menimbulkan masalah di kemudian hari.

Aturan Penulisan Tanda Semicolon pada Akhir Baris

Berbeda dari kebanyakan bahasa pemograman, di dalam **JavaScript** karakter titik-koma (bahasa inggris: **semicolon**) sifatnya **opsional** untuk digunakan sebagai penanda akhir dari baris program, dan boleh tidak ditulis.

JavaScript 'mendeteksi' baris baru (karakter '*break'*) sebagai penanda akhir baris program. Kode perintah berikut berjalan sebagaimana mestinya di dalam JavaScript:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   a=13
   b=a+2
   console.log(b)
   alert(b)
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di jti.polije.com </p>
```

```
</div>
</body>
</html>
```

Perhatikan bahwa di dalam **tag <script>** saya tidak menggunakan tanda *semicolon* untuk menutup baris perintah **JavaScript**, dan kode tersebut berjalan sukses tanpa menghasilkan *error*.

Namun *javascript* tidak akan '*mendeteksi'* setiap baris baru sebagai penanda akhir baris program, perhatikan kode berikut ini:

```
<script>
   var a
   a
   =
   15
   console.log(a)
</script>
```

Kode tersebut akan diproses oleh JavaScript menjadi:

```
<script>
   var a; a = 15; console.log(a)
</script>
```

Akan tetapi, penulisan berikut ini:

```
<script>
   a = 5
   b = 3 + a
   (b + 10).toString()
</script>
```

Akan diproses menjadi:

```
<script>
   a = 5 ; b = 3 + a(b + 10).toString();
</script>
```

Hal ini terjadi karena **JavaScript** akan menganggap baris baru sebagai tanda *semicolon* (pemisah baris program) apabila pada saat memproses baris berikutnya sudah tidak mungkin '*bersambung'*. Penggunaan karakter pembuka kurung "(" di awal baris baru, akan dianggap sebagai bagian sambungan dari baris sebelumnya.

Penggunaan tanda semicolon ";" walau bersifat *opsional*, namun sangat dianjurkan digunakan. Tanda **semicolon** akan membuat program lebih mudah dibaca dan tidak membuat ambigu seperti contoh kita diatas. Di dalam tutorial JavaScript di jti.polije ini, saya juga akan menggunakan tanda ";" pada setiap akhir baris.

j. Aturan Penamaan Variabel JavaScript

Aturan penamaan variabel pada JavaScript sama dengan aturan pembuatan **identifier**:

- Karakter pertama harus diawali dengan **huruf**, **underscore** (_) atau **tanda dollar** ($)
- Karakter kedua dan seterusnya bisa ditambahkan dengan **huruf**, **angka**, **underscore** (_) atau **tanda dollar** ($).

Cara Membuat Variabel JavaScript

Walaupun kita tidak perlu menyebutkan jenis tipe data dari suatu variabel, namun kita tetap harus mendeklarasikan **variabel** di dalam **JavaScript**. Cara membuat variabel di dalam **JavaScript** di bedakan menjadi 2, yakni dengan menggunakan *keyword* **var**, dan **tanpa var**.

Jika menggunakan kata kunci **var**, berikut adalah contoh penulisannya:

```
<script>
  var a;
  var b, c, d;
  var e; var f;
  var g=12;
  var h="Saya Sedang Belajar JavaScript di Jti.polije";
</script>
```

Pada 2 baris terakhir saya membuat *variabel*, sekaligus memberikan nilai ke dalam variabel tersebut.

Cara kedua untuk membuat variabel adalah tanpa menggunakan keyword **var**, seperti berikut ini:

```
<script>
  a;
  b=12;
  c=" Saya Sedang Belajar JavaScript di Jti.polije";
</script>
```

Pembuatan variabel tanpa menggunakan *keyword* **var** memang lebih cepat, akan tetapi tidak disarankan. Walaupun variabel yang dideklarasikan tanpa keyword **var** akan tetap berfungsi sebagaimana mestinya seperti variabel dengan **var**, namun **JavaScript** ‘menyimpan’ variabel tersebut dengan cara yang berbeda. Salah satu perbedaannya adalah tentang **variabel scope** (yang akan kita bahas setelah ini). Sedapat mungkin kita selalu membuat variabel menggunakan keyword **var**.

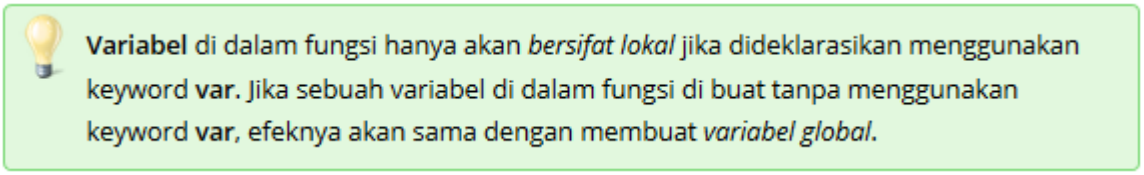
Jangkauan Variabel (Variabel Scope) dalam JavaScript

Jangkauan Variabel (atau **Variabel Scope**) adalah konsep tentang pembatasan akses dari sebuah variabel. Yaitu pada bagian mana sebuah variabel masih bisa diakses.

Sebuah **variabel** jika *dideklarasikan* (baik dengan *keyword* **var** ataupun *tanpa* **var**), akan bersifat **global**, atau dikenal dengan istilah **global variable**. Sebuah variabel akan menjadi global variabel sepanjang variabel tersebut di deklarasikan di luar fungsi.

Jika sebuah variabel di deklarasikan di dalam fungsi, maka variabel tersebut hanya akan bisa diakses di dalam fungsi tersebut, atau bersifat lokal (dikenal juga dengan **local variable**).

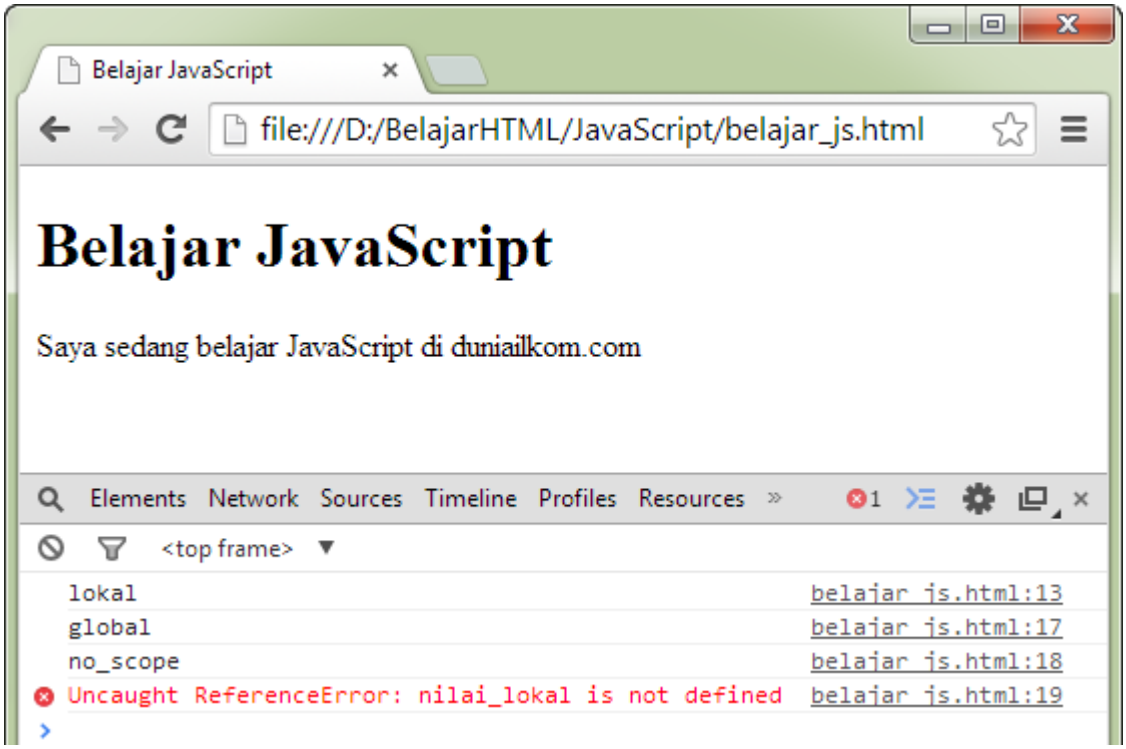
Apabila kita membuat 2 variabel dengan nama yang sama sebagai **global variabel**, dan juga **local variable** di dalam sebuah fungsi, maka *local variable* akan memiliki prioritas yang lebih tinggi dibandingkan **global variabel**.



Berikut adalah contoh efek konsep **global variable** dan **local variable** dalam JavaScript:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
var nilai = "global";
function test() {
   var nilai = "lokal";
   var nilai_lokal = "jti.polije";
   tanpa_var = "no_scope"; //akan menjadi global variabel!!
   console.log(nilai);
   }

test(); // print: lokal
console.log(nilai); // print: global
console.log(tanpa_var); //print: no_scope (bisa diakses)
console.log(nilai_lokal); //error, karena nilai_lokal adalah
lokal variabel
</script>

</head>
```

```
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript di jti.polije.com </p>
</body>
</html>
```



Dalam contoh kode diatas, saya membuat dan mendeklarasikan beberapa variabel secara **global** maupun **lokal**.

Pada baris ke-8, saya membuat sebuah variabel global, dengan nama **nilai**, dan memberikan nilai string "**global**" ke dalam variabel ini.

Selanjutnya, saya membuat **fungsi test()** dan mendeklarasikan kembali variabel **nilai**, namun kali ini memberikan nilai string "**lokal**". Di dalam fungsi ini saya juga membuat variabel baru, yakni **nilai_lokal** dan **tanpa_var**.

Khusus untuk variabel **tanpa_var**, sesuai dengan namanya, saya mendeklarisakannya tanpa keyword var. sehingga efeknya, variabel **tanpa_var** akan menjadi variabel **global**.

Dengan membuat beberapa perintah **console.log**, kita dapat mencari tahu apa yang terjadi dari contoh kode diatas.

Variabel **nilai** apabila diakses dari luar fungsi akan bernilai "**global**", sedangkan jika diakses dari dalam fungsi akan bernilai "**lokal**". Variabel **tanpa_var** juga sukses ditampilkan, karena variabel ini telah menjadi **variabel global**.

Saat mencoba mengakses variabel **nilai_lokal**, web browser akan menampilkan *error*. Hal ini terjadi karen variabel **nilai_lokal** bersifat lokal di dalam **fungsi test**, sehingga apabila diakses dari luar fungsi, JavaScript '*tidak melihat*' variabel ini, dan menampilkan *error*.

k. Jenis dan Pengertian Tipe Data Dalam JavaScript

Sebagai bahasa pemograman yang **berbasis objek**, konsep tipe data di dalam **JavaScript** sedikit berbeda jika dibandingkan dengan bahasa pemograman yang berbasis prosedural seperti PHP (walaupun PHP tersedia dalam bentuk **prosedural** maupun **objek**).

Tipe data sederhana seperti Angka (**Float**), Text (**String**) '*seolah-olah'* bersifat sebagai **objek** di dalam **JavaScript**.

Jenis dan Pengertian Tipe data dalam JavaScript

Tipe data dalam JavaScript dibedakan menjadi 2 kelompok, yakni **tipe data dasar** (*primitif*) dan **tipe data objek**.

Tipe data dasar terdiri dari tipe data **angka**, tipe data text (**string**), dan tipe data **boolean**. Tipe data **null** dan **undefined** juga merupakan tipe data dasar, namun memiliki jenis tersendiri.

Selain ke-5 tipe data dasar tersebut, tipe data lain yang ada di dalam **JavaScript** adalah **tipe data objek**. Contoh tipe data **objek** adalah tipe data tanggal (**date**), **array**, dan **fungsi**. Saya akan membahas masing-masing jenis tipe data ini secara mendetail dalam beberapa tutorial berikutnya.

Konsep Objek pada Tipe Data Dasar JavaScript

Walaupun disebut tipe data dasar, tipe data *angka*, text (**string**), dan **boolean** di dalam **JavaScript** berprilaku 'seolah-olah' sebagai **objek**. Dimana setiap variabel yang berisikan tipe data, akan memiliki method (atau fungsi) yang '*melekat'* kepada variabel tersebut.

Dalam bahasa pemograman jenis **prosedural**, setiap tipe data dapat disebut '*berdiri'* sendiri. Untuk menjalankan beberapa perintah, kita menggunakan fungsi yang menjadikan variabel tipe data sebagai inputan untuk fungsi tersebut.

Contohnya, untuk mencari panjang sebuah string di dalam PHP, kita menggunakan fungsi **strlen**. Jika **$a** adalah sebuah *variabel* bertipe **String**, maka untuk mencari panjang dari variabel **$a** adalah dengan **strlen($a)**. Dan untuk membuat karakter string menjadi huruf besar, kita menggunakan fungsi **strtoupper ($a)**.

Berikut adalah contoh penulisan fungsi **strlen** dan **strtoupper** dalam **PHP**:

```
<?php
$a = "Saya Sedang Belajar JavaScript di jti.polije.com";
echo strlen($a); //48
echo strtoupper($a); //SAYA SEDANG BELAJAR
?>
```

Contoh diatas adalah penggunaan fungsi di dalam pemograman *prosedural* **PHP**. Tetapi karena **JavaScript** berbasis objek, konsepnya akan sedikit berbeda.

Di dalam **JavaScript**, walaupun juga memiliki fungsi bawaan untuk membantu kita dalam membuat kode program, namun kebanyakan fungsi dasar ‘melekat’ kedalam variabel, atau di dalam istilah pemograman objek: setiap *variabel* akan memiliki ‘**method**’.

Cara pemanggilan **method** dari **variabel** di dalam JavaScript adalah dengan menambahkan tanda ‘titik’. Misalkan **a** variabel dengan tipe data **string** yang berisi sebuah kalimat. Untuk mencari panjang dari **sting a,** kita memanggil **method** dengan cara: **a. length**. Dan untuk menampilkan karakter dalam variabel **a** menjadi huruf besar, ditulis dengan **a.toUpperCase()**

Berikut adalah contoh penggunaan method **length** dan **toUpperCase** dalam JavaScript:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   a = "Saya Sedang Belajar JavaScript di Jti.polije";
   console.log(a.length);
   console.log(a.toUpperCase());
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar JavaScript</p>
</body>
</html>
```

Dari contoh tersebut terlihat bahwa dalam JavaScript, *fungsi* bawaan untuk pemrosesan string adalah **method** dari **object String** itu sendiri. Penjelasan dan method-method yang ada di dalam **JavaScript** akan kita bahas secara detail bagian selanjutnya.

Di dalam konsep *pemograman berbasis objek*, **method** adalah sebutan untuk **fungsi** yang '*melekat*' kepada sebuah **objek** (dalam contoh kita, adalah *fungsi* yang melekat kepada variabel ber-*tipe data* **String**).

Cara pemanggilan **method** dilakukan dengan menambahkan tanda titik setelah penulisan **variabel**. Misalkan variabel **a** bertipe data **String**, dan string di dalam JavaScript memiliki method untuk memotong *string*, yakni method **slice**. Maka untuk pemanggilan fungsi **slice** adalah sebagai berikut: **a.slice()**

Jika ini adalah kali pertama anda mendengar istilah **objek** dan **method**, mungkin penjelasan di dalam tutorial kali ini terkesan '*susah untuk dipahami*'. Saya akan membahas konsep pemograman objek pada tutorial khusus nantinya. Namun paling tidak di dalam tutorial kali ini dapat diambil kesimpulan, bahwa fungsi di dalam JavaScript bisa dipanggil dari variabelnya.

Di dalam **JavaScript** terdapat beberapa method yang melekat pada hampir seluruh tipe data, misalnya method **toString()**. Method ini berfungsi untuk mengkonversi nilai variabel, dan menampilkannya menjadi **String**.

l. Mengenal Tipe Data dan Operator Angka (Number) JavaScript

Tipe data Angka dalam JavaScript

Tidak seperti bahasa pemograman lain pada umumnya, **JavaScript** tidak membedakan tipe data angka (**number**) antara angka bulat dengan angka desimal, atau tidak membedakan antara bilangan integer dengan float. Seluruh tipe data angka di dalam JavaScript disimpan dalam bentuk desimal (**float**).

Jangkauan tipe data angka di dalam **JavaScript** merujuk kepada aturan standar *IEEE 754* yang mencakup angka paling kecil ±5 × 10^-324 dan angka paling tinggi ±1.7976931348623157 × 10^308.

Angka di dalam **JavaScript** bisa dengan tepat merepresentasikan nilai antara −2^53 sampai dengan 2^53, atau dari −9007199254740992 sampai 9007199254740992. Jika anda menggunakan nilai yang lebih besar dari jangkauan ini, maka akan mengorbankan tingkat ketelitian.

Cara Membuat Tipe Data Angka

Karena di dalam **JavaScript** sebuah variabel tidak perlu di deklarasikan akan bertipe apa, maka jika sebuah variabel diberikan nilai angka, maka variabel tersebut telah menjadi variabel dengan tipe angka. Berikut adalah contohnya:

```
<script>
var a=12; // deklarasi dengan var
b=6; // deklarasi tanpa keyword var
</script>
```

Operator Aritmatika di dalam JavaScript

Operasi yang sering digunakan untuk tipe data angka adalah operasi artimatika seperti **penambahan** (+), **pengurangan** (-), **pembagian** (/), **perkalian** (*), dan **modulus** atau **sisa hasil bagi** (%).

Urutan Prioritas Operator Aritmatika

Sama seperti pelajaran matematika yang kita pelajari, di dalam **JavaScript** operator *perkalian* dan *pembagian* lebih kuat dari pada operator *penambahan* dan *pengurangan*. Perintah operasi 1 + 3 * 5 akan proses oleh JavaScript menjadi 1 + (3 * 5). Operasi **modulus** (sisa hasil bagi) juga sama kuat dengan operasi *perkalian* dan *pembagian*.

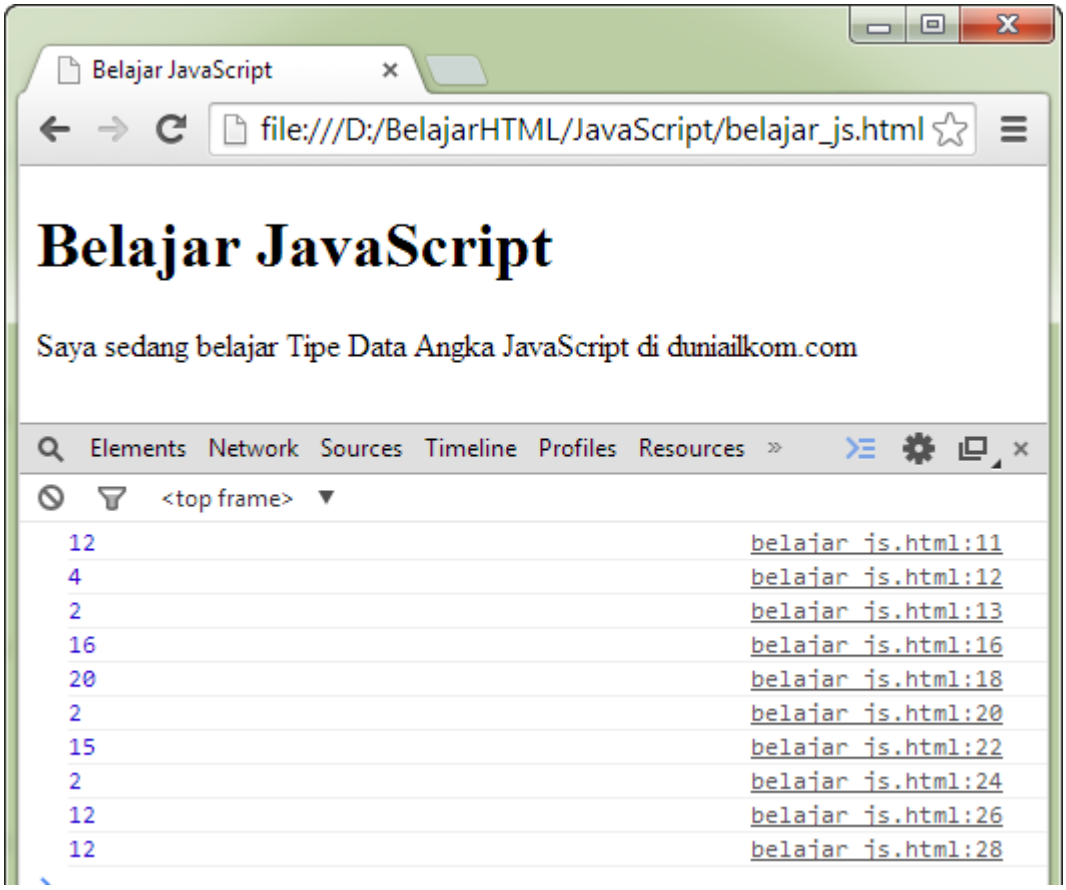
Jika operasi matematika yang akan diproses memiliki lebih dari 1 operator yang sama kuat dalam sebuah perhitungan, **JavaScript** akan memprioritaskan urutan perhitungan dari kiri ke kanan. Operasi 3 * 4 / 5 *5 akan dikerjakan menjadi: (((3 * 4) / 5) * 5).

Untuk membuat **prioritas** suatu operasi menjadi lebih tinggi, kita bisa menambahkan **tanda kurung** didalam program. Penggunaan tanda kurung juga akan mempermudah pembacaan program. Walaupun kita bisa menebak bagaimana operasi 5 + 3 * 2 akan diproses, namun akan lebih mudah di baca jika operasi tersebut tetap dibuat dengan menambahkan tanda kurung menjadi 5 + (3 * 2).

Berikut adalah contoh kode program cara penggunaan tipe data angka dan operasi matematis dalam JavaScript:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   var a = 12;
   var b = 4;
   var c = 2;
   console.log(a);       // hasil: 12
   console.log(b);       // hasil: 4
   console.log(c);       // hasil: 2

   var d = a + b;
   console.log(d);       // hasil: 16
   var e = b * 5;
   console.log(e);       // hasil: 20
   var f = a % 5;
   console.log(f);       // hasil: 2
   var g = 3 + 6 * 2;
   console.log(g);       // hasil: 15
   var h = c + b * a % 3;
   console.log(h);       // hasil: 2
```

```
    var i = 3 * 4 / 5 * 5;
    console.log(i);       // hasil: 12
    var j = (((3 * 4) / 5) * 5);
    console.log(j);       // hasil: 12
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar Tipe Data Angka JavaScript di </p>
</body>
</html>
```



Pada awal kode **JavaScript** diatas (yang dimulai dari baris ke-8), saya membuat 3 buah variabel, yakni variabel **a**, **b** dan **c**. Pada saat pendefenisian variabel ini, saya juga langsung memberikan nilai awal. Perintah **console.log** selanjutnya akan menampilkan isi dari ketiga variabel ini.

Karena variabel di dalam JavaScript tidak memiliki tipe dasar, dengan memberikan nilai awal berupa angka (***number***), maka variabel tersebut akan **bertipe data number**.

Setelah mendefenisikan beberapa variabel, selanjutnya saya melakukan operasi-operasi sederhana seperti ***penambahan***, ***pengurangan***, dan juga ***modulus***.

Dalam 3 contoh operasi terakhir, kita bisa melihat bagaimana **JavaScript** menangani perintah matematis berdasarkan urutan '*kekuatan*' operator.

Dalam menjalankan operasi *matematis*, adakalanya perhitungan yang dilakukan akan menghasilkan nilai yang *tidak lazim*, seperti pembagian sebuah bilangan dengan 0, atau akar kuadrat dari bilangan negatif. Untuk menangani hal ini, **JavaScript** memiliki nilai angka **NaN** dan **Infinity**.

m. Pengertian Hasil Operasi Matematis NaN dan Infinity

JavaScript memiliki 2 hasil operasi matematis yang '*bukan*' berupa angka, yakni **NaN** dan **Infinity**. Di dalam tutorial **JavaScript** kali ini kita akan membahas tentang keduanya dan apa saja operasi yang menghasilkan kedua nilai ini.

Pengertian NaN dan Infinity dalam JavaScript

NaN dan **Infinity** adalah dua hasil perhitungan matematis yang digunakan JavaScript untuk menampung nilai yang bukan angka.

Operasi matematika yang hasilnya '*tidak terdefinisi*', tidak akan dianggap **error** di dalam **JavaScript**. Misalnya operasi pembagian sebuah angka dengan angka 0, atau hasil dari akar kuadrat nilai negatif. Di dalam JavaScript, operasi seperti ini diselesaikan dengan nilai **NaN** dan **Infinity**.

Perhitungan yang menghasilkan nilai Infinity

Hasil **Infinity** akan di dapat jika kita melakukan salah satu operasi di bawah ini:

- Jika sebuah angka melewati nilai maksimum angka yang bisa ditampung di dalam **JavaScript** (disebut juga dengan istilah: ***overflow***).
- Jika sebuah angka lebih besar dari angka *negatif* yang bisa ditampung (***negative overflow***).
- Melakukan operasi aritmatika dengan **infinity** (misal: a=1 + infinity)
- Melakukan operasi pembagian dengan nilai 0 (*division by zero*)

  Nilai **Infinity** di dalam **JavaScript** dibedakan menjadi 2, yakni **infinity positif** dan **infinity negatif**. **Infinity positif** di dapat jika angka yang dihasilkan lebih besar dari angka positif yang dapat ditampung. **Infinity negatif** di dapat jika angka yang dihasilkan lebih negatif dari angka negatif yang dapat ditampung (bukan nilai paling kecil).

  Jika angka hasil operasi sangat kecil, atau disebut dengan **underflow**, JavaScript akan menampilkannya dengan angka 0.

Perhitungan yang menghasilkan nilai NaN

Nilai **NaN** (singkatan dari *Not a Number*) akan di dapat jika melakukan operasi dibawah ini:

- Pembagian 0 dengan 0
- Pembagian **infinity** dengan **infinity**
- Akar kuadrat dari nilai negatif

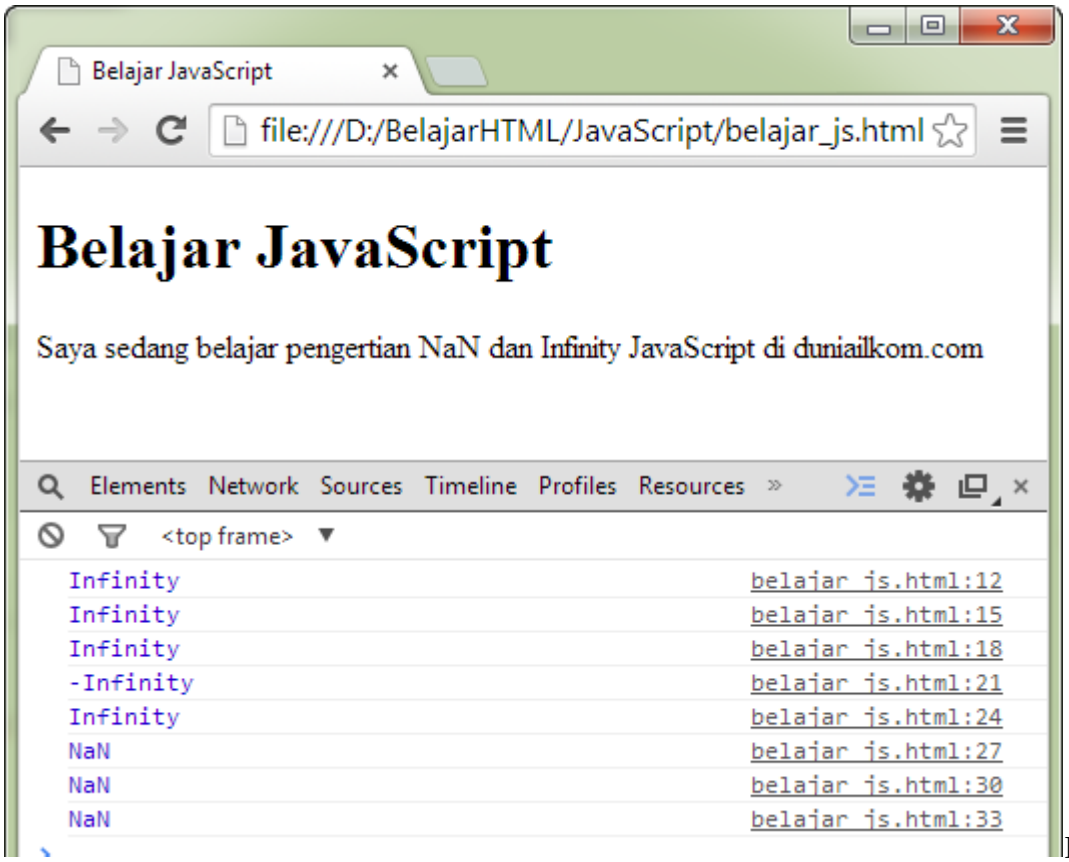
- Operasi aritmatika dengan nilai yang bukan angka (dan tidak bisa dikonversi menjadi angka).

Contoh Perhitungan NaN dan Infinity

Berikut adalah contoh program JavaScript yang akan menghasilkan nilai **Infinity** dan **NaN**:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   a = 5;
   b = 7;

   c = b / 0;
   console.log(c);      // Hasil: Infinity (division by zero)

   d = 12 / 0;
   console.log(d);      // Hasil: Infinity (division by zero)

   e = 1 + Math.pow(10,309);
   console.log(e);      // Hasil: Infinity (overflow)

   f = -Math.pow(10,309) - 1;
   console.log(f);      // Hasil: -Infinity (negative
infinity)

   g = b + e;
   console.log(g);      // Hasil: Infinity (karena e dan b
infinity)

   h = 0/0;
   console.log(h);      // Hasil: NaN

   i = Math.sqrt(-25);
   console.log(i);      // Hasil: NaN (akar kuadrat negatif)

   j = c / d
   console.log(j);   // Hasil: NaN (c dan d sama-sama bernilai
infinity)

</script>
</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar pengertian NaN dan Infinity JavaScript
</p>
</div>
```

```
</body>
</html>
```



Di

Dalam contoh kode **JavaScript** diatas, saya melakukan beberapa perhitungan matematis yang akan menghasilkan nilai **NaN** dan **Infinity**. Untuk perhitungan variabel **e** dan **i**, saya menggunakan objek **Math**.

Salah satu sifat dari nilai **NaN** yang agak unik adalah nilai **NaN** tidak akan menghasilkan **TRUE** jika dibandingkan dengan apapun, termasuk dengan **NaN** sendiri. **a == NaN** tidak akan pernah menghasilkan **TRUE**, berapapun nilai dari variabel a. Jika anda ingin membandingkan nilai **NaN**, bisa menggunakan persamaan **a != a.** Persamaan **a!= a** hanya akan bernilai TRUE jika (dan hanya jika) variabel **a** bernilai **NaN.** Atau bisa juga dengan menggunakan fungsi isNaN().

Seperti yang telah dibahas bahwa **JavaScript** memiliki fungsi yang bisa digunakan untuk proses perhitungan atau untuk menampilkan tipe data, termasuk tipe data angka.

n. Mengenal Method Objek Angka (Number) dalam JavaScript

Pengertian Objek Angka (Number) dalam JavaScipt

Dengan memberikan nilai angka ke dalam sebuah **variabel**, secara otomatis variabel tersebut akan bertipe *angka* (**number**). **Variabel** dengan tipe angka, akan memiliki **method** yang bisa digunakan untuk memproses data yang ada didalam varibel tersebut.

Di dalam **JavaScript**, *method* untuk tipe *angka* akan lebih banyak berfungsi untuk mengontrol bagaimana angka tersebut ditampilkan ke dalam web browser, dan umumnya akan menghasilkan nilai dengan tipe data **String**. Kita akan membahas **method** untuk tipe data number ini secara satu-persatu.
Method ini diurutkan menurut abjad.

Method JavaScript: Number.toExponential()

Method **toExponential** digunakan untuk menampilkan angka menjadi tampilan *scientific notation*. **Scientific Notation** adalah tampilan angka dengan 1 digit sebelum tanda desimal, dan diikuti dengan tanda pangkat. Misalnya angka 123.45, jika ditulis kedalam bentuk *scientific notation* akan menjadi 1.23 x 10^2. Akan tetapi, bahasa pemograman umumnya mengganti penyebutan pangkat sepuluh ini menjadi karakter **e** atau **E**. Sehingga 1.23 x 10^2 ditulis menjadi 1.23e+2.

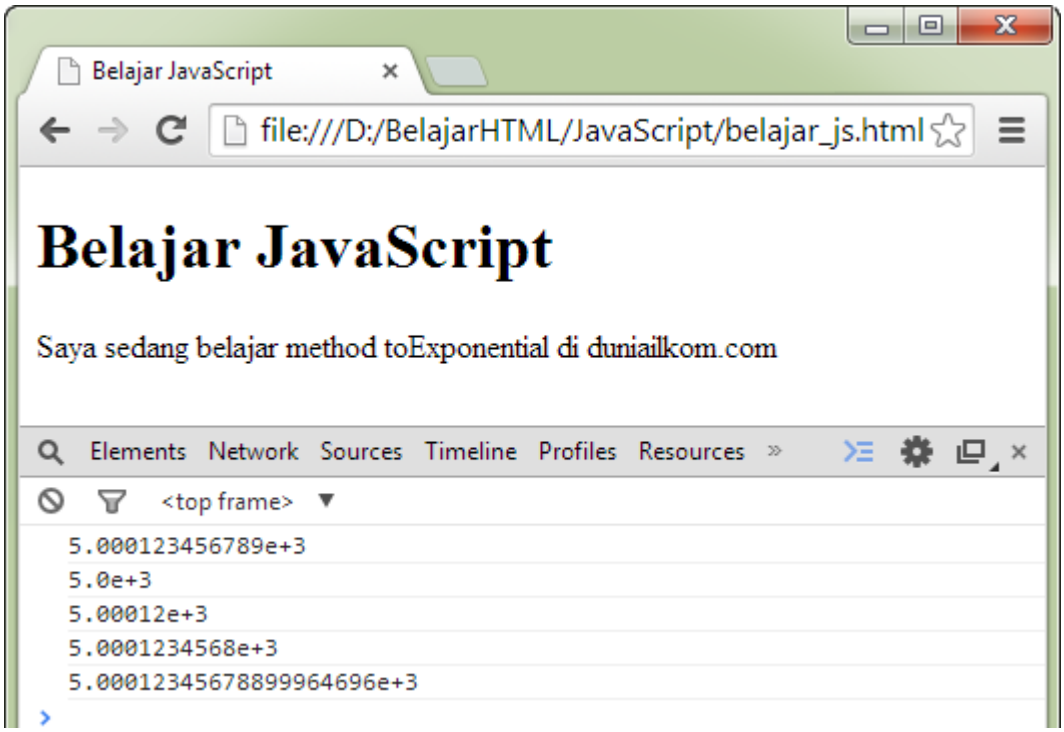
Method **Number.toExponential** menyediakan 1 argumen *opsional* yang jika diinput inputan akan menentukan 'panjang' digit secara keseluruhan. Argumen ini bisa diisi dengan angka 0 – 20. Apabila tidak menggunakan argumen, maka method **toExponential** akan menggunakan panjang paling maksimal sesuai jumlah digit dalam variabel asal.

Tipe data kembalian method `Number.toExponential()` bertipe `String.`

Berikut adalah contoh penggunaan method `toExponential`:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   a = 5000.123456789;
   b = a.toExponential();
   c = a.toExponential(1);
   console.log(b);                        // hasil:
5.000123456789e+3
   console.log(c);                        // hasil: 5.0e+3
   console.log(a.toExponential(5));       // hasil: 5.00012e+3
   console.log(a.toExponential(10));      // hasil:
5.0001234568e+3
```

```
   console.log(a.toExponential(20));    // hasil:
5.00012345678899964696e+3
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar method toExponential</p>
</div>
</body>
</html>
```



Method JavaScript: Number.toFixed()

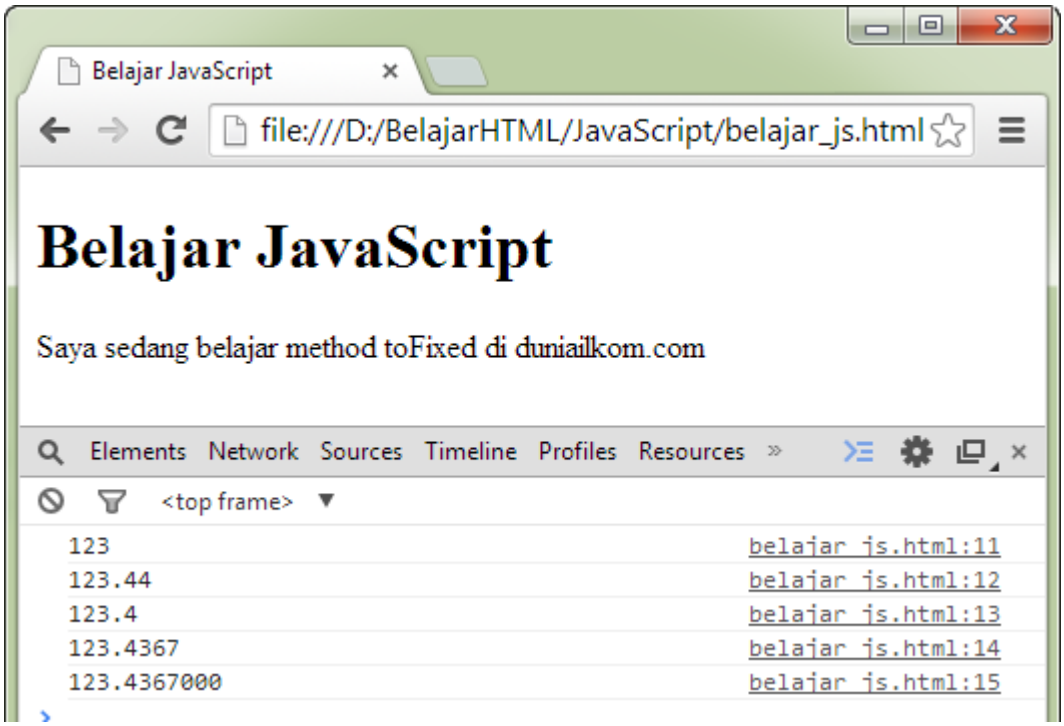
Method **toFixed** digunakan untuk membuat tampilan angka dengan jumlah desimal yang tetap. Method ini membutuhkan satu parameter *opsional* yang jika diinput akan menentukan jumlah digit setelah tanda desimal.

Jika angka yang di tampilkan mengurangi digit asal, method ini akan membulatkan ke bilangan terdekat (0,5 akan menjadi 1). Dan apabila angka yang ditampilkan melebihi digit asal, method ini akan menambahkan angka 0 dibelakang hasil agar tampilan sesuai dengan yang diinginkan.

Misalkan angka 123,222222, akan ditampilkan menjadi 123,22 jika menggunakan method **toFixed(2),** dan akan ditampilkan menjadi 123,222 jika menggunakan **toFixed(3).** Hasil pemanggilan method ini bertipe **String**.

Berikut adalah contoh penggunaan method **toFixed** untuk tipe data angka dalam JavaScript:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   a = 123.4367;
   b = a.toFixed();
   c = a.toFixed(2);
   console.log(b);                 // hasil: 123;
   console.log(c);                 // hasil: 123.44;
   console.log(a.toFixed(1));      // hasil: 123.4;
   console.log(a.toFixed(4));      // hasil: 123.4367;
   console.log(a.toFixed(7));      // hasil: 123.4567000;
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar method toFixed</p>
</div>
</body>
</html>
```



Seperti hasil dari contoh, method **toFixed** akan membulatkan angka desimal jika ke bilangan terdekat, dan menambahkan angka 0 pada akhir angka untuk mencukupi bilangan **fixed** yang diminta. Fitur ini akan sangat berguna untuk

merapikan tampilan, misalnya untuk tampilan nominal mata uang yang biasanya memiliki 2 tempat desimal.

Method JavaScript: Number. toPrecision()

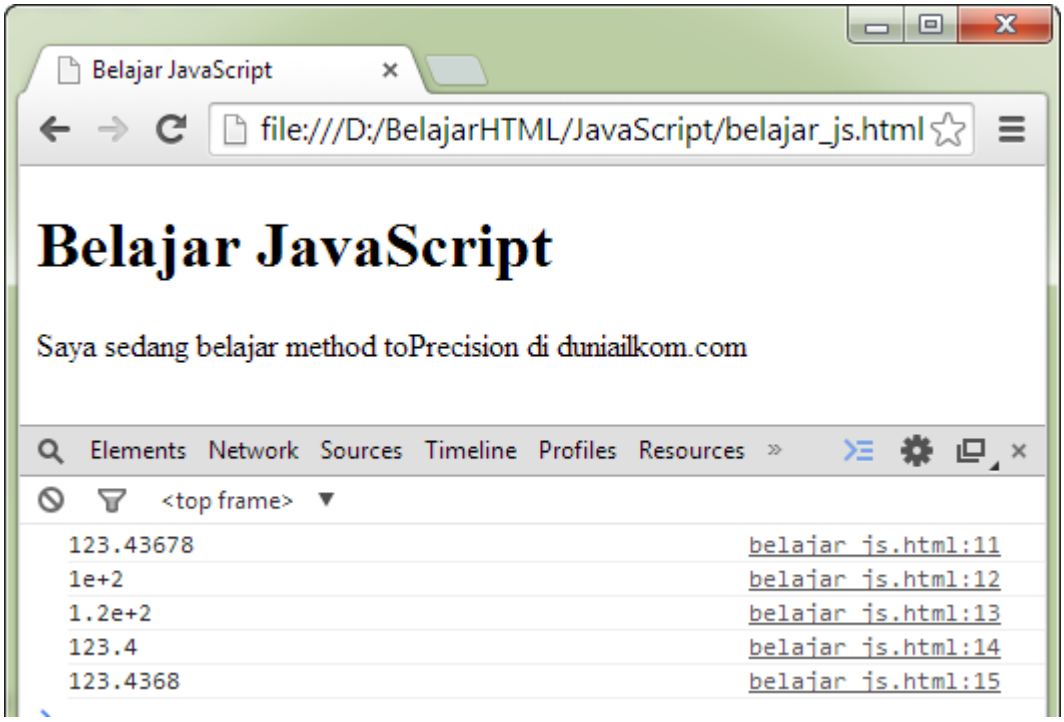
Method **toPrecision** digunakan untuk menampilkan angka dengan jumlah digit angka yang '*tetap'* tergantung nilai angka yang dijadikan sebagai argumen. Misalkan jika angka yang akan ditampilkan adalah 12.2234 maka hasil dari toPrecision(2) adalah 12, dan toPrecision(3) adalah 12.2.

Berbeda dengan 2 method sebelumnya, nilai tetap untuk method **toPrecision** adalah jumlah digit sebelum dan sesudah desimal.

Jika argumen tidak ditulis, method ini akan mengembalikan nilai awal variabel, tanpa '*menformatnya'*. Hasil kembalian **toPrecision()** bertipe **String**.

Sebagai contoh program, berikut adalah hasil tampilan penggunaan method **toPrecision** di dalam JavaScript:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   a = 123.43678;
   b = a.toPrecision();
   c = a.toPrecision(1);
   console.log(b);                    // hasil: 123.4367
   console.log(c);                    // hasil: 1e+2
   console.log(a.toPrecision(2));     // hasil: 1.2e+2
   console.log(a.toPrecision(4));     // hasil: 123.4;
   console.log(a.toPrecision(7));     // hasil: 123.4368;
</script>
</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar method toPrecision</p>
</div>
</body>
</html>
```



Perhatikan bahwa jika angka asal memiliki digit yang lebih dari argumen method to precision, seperti contoh pemanggilan a.toPrecision(2) dari123.43678 , hasilnya akan ditampilkan menggunakan scientific notation.

Method JavaScript: Number.toString()

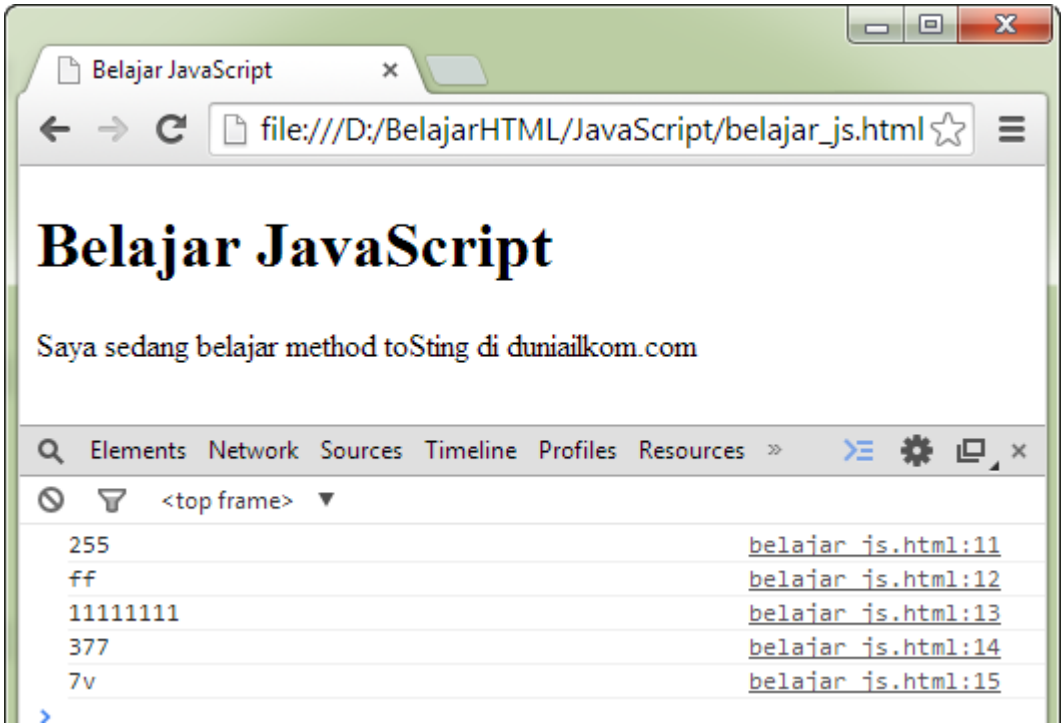
Method **toString** dimiliki hampir semua tipe data. **Method** ini berfungsi untuk menkonversi tipe data menjadi **string**.

Jika digunakan untuk tipe data number, method ini bisa diberikan sebuah **argumen** *opsional* yang akan menampilkan angka ke dalam bentuk basis lain selain 10. Misalkan angka asal 255, jika dipanggil dengan **toString(2)** akan ditampilkan menjadi 11111111. Karena **toString(2)** berarti menampilkan angka menjadi basis 2 (bilangan biner). Argumen yang didukung adalah dari 2 sampai 36.

Berikut adalah contoh tampilan penggunaan method **toSting** untuk tipe data Number di dalam JavaScript.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   a = 255;
   b = a.toString();
   c = a.toString(16);
   console.log(b);                    // hasil: 255
```

```
    console.log(c);                 // hasil: ff
    console.log(a.toString(2));     // hasil: 11111111
    console.log(a.toString(8));     // hasil: 377
    console.log(a.toString(32));    // hasil: 7v
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar method toSting</p>
</div>
</body>
</html>
```



Diluar dari ke empat method yang dijelaskan di sini, **JavaScript** juga memiliki 2 method lain untuk tipe data number, yakni **Number.toLocaleString()** untuk menampilkan angka sesuai aturan lokal web browser (mengganti tanda titik menjadi koma untuk pembeda tempat desimal), dan **Number.valueOf()** yang digunakan untuk memanggil tipe data '*primitif*' dari objek angka. Keduanya tidak terlalu sering digunakan.

Selain memiliki **method** (*fungsi*) tipe data number di dalam **JavaScript** juga memiliki ***read-only properti***, atau **konstanta** yang menyimpan angka '*khusus*' yang bisa digunakan di dalam kode program.

o. Mengenal Konstanta Objek Angka (Number) dalam JavaScript

Konstanta Objek Number dalam JavaScript

Selain memiliki **method** atau fungsi, objek number di dalam **JavaScript** juga memiliki beberapa **konstanta** (*constant*) yang bisa digunakan untuk proses matematika. **Konstanta** ini berisi beberapa nilai yang bisa digunakan dalam perhitungan matematis dan proses logika program.

Jika anda pernah menggunakan bahasa *pemograman berbasis objek*, istilah **konstanta** dalam **JavaScript** ini lebih tepatnya disebut dengan **read-only property.**

Namun berbeda dengan *method* yang kita bahas sebelumnya, *konstanta* ini melekat kepada objek **Number**, bukan variabelnya dan dipanggil dengan menuliskan **Number.nama_konstanta**.

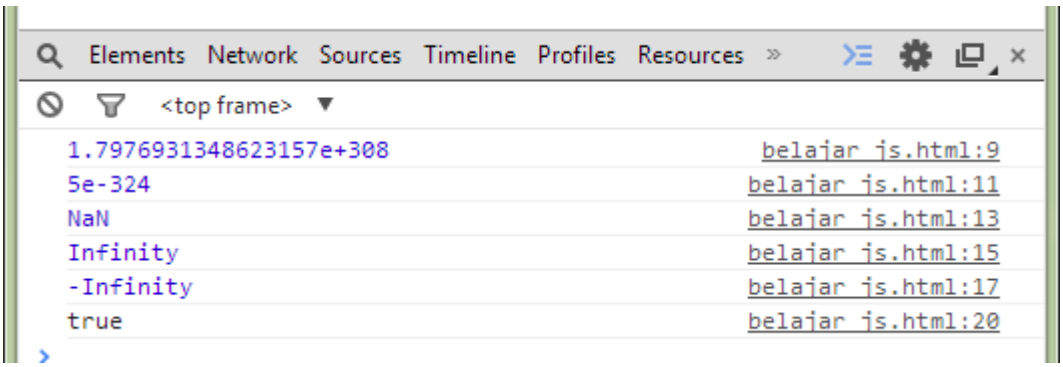
Berikut adalah beberapa konstanta untuk objek Number di dalam JavaScript:

- **Number.MAX_VALUE**: *Konstanta* ini menyimpan **angka tertinggi** yang bisa ditampung di dalam **JavaScript**. Nilainya adalah 1.79E+308, atau 1.79 x 10^308.
- **Number.MIN_VALUE**: *Konstanta* ini menyimpan **angka terkecil** yang bisa ditampung di dalam JavaScript. Nilai tersebut bukan nilai paling negatif, tetapi nilai yang hampir mendekati 0. Nilainya adalah 5E-324 atau 5 x 10^-324.
- **Number.NaN**: *Konstanta* ini menyimpan nilai khusus **NaN**, singkatan **Not a Number**. Nilai ini biasanya dihasilkan dari beberapa perhitungan yang ‘tidak biasa’ seperti hasil 0/0. **NaN** sudah kita bahas pada tutorial sebelumnya: Pengertian Hasil Operasi Matematis NaN dan Infinity.
- **Number.NEGATIVE_INFINITY**: *Konstanta* ini menyimpan nilai khusus **Negatif Infinity**. **Negatif Infinity** adalah nilai khusus yang di dapat jika membuat sebuah angka yang lebih negatif dari nilai maksimum yang bisa ditampung di dalam JavaScript, yakni lebih negatif dari –**Number.MAX_VALUE**, dan biasa di kenal dengan istilah **overflow**.
- **Number.POSITIVE_INFINITY**: *Konstanta* ini menyimpan nilai khusus **Positif Infinity**. **Positif Infinity** adalah nilai khusus yang di dapat jika membuat sebuah angka yang lebih besar dari nilai maksimum yang bisa ditampung di dalam **JavaScript**, yakni lebih besar dari **Number.MAX_VALUE**.

Berikut adalah contoh kode program JavaScript untuk penggunaan konstanta **Number**:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   a = Number.MAX_VALUE;
```

```
   console.log(a);                 // hasil:
1.7976931348623157e+308

   b = Number.MIN_VALUE;
   console.log(b);                 // hasil: 5e-324

   c = Number.NaN;
   console.log(c);                 // hasil: NaN

   d = Number.POSITIVE_INFINITY;
   console.log(d);                 // hasil: Infinity

   e = Number.NEGATIVE_INFINITY;
   console.log(e);                 // hasil: -Infinity

f = 1/0;
console.log ( d == f );            // hasil: true
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar Konstanta Number JavaScript di
jti.polije.com </p>
</div>
</body>
</html>
```



Di dalam baris terakhir contoh program, saya menguji apakah hasil 1/0 sama dengan **Number.POSITIVE_INFINITY**.

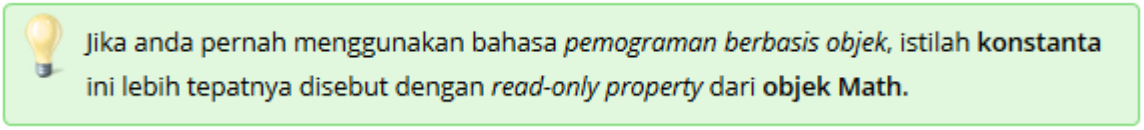
p. Konstanta dan Method (fungsi) Objek Math dalam JavaScript

Konstanta untuk Objek Math dalam JavaScript

Objek Math memiliki beberapa *konstanta* matematika yang bisa digunakan di dalam proses pembuatan program. Untuk menggunakan konstanta **objek Math**, kita tinggal menulis: **Math.nama_konstanta**.

Berikut adalah kosntanta untuk Objek Math di dalam JavaScript, diurutkan berdasarkan abjad:

- **Math.E**: Berisi nilai dari logaritma natural e, dengan nilai 2.718281828459045
- **Math.LN10** : Berisi nilai dari logaritma natural 10, dengan nilai 2.302585092994046
- **Math.LN2** : Berisi nilai dari logaritma natural 2, dengan nilai 0.6931471805599453
- **Math.LOG10E** : Berisi nilai dari logaritma natural e basis 10, dengan nilai 0.4342944819032518
- **Math.LOG2E** : Berisi nilai dari logaritma natural e basis 2, dengan nilai 1.4426950408889634
- **Math.PI** : Berisi nilai dari pi (π) dengan nilai 3.141592653589793
- **Math.SQRT1_2**: Berisi hasil dari 1 dibagi dengan akar kuadrat 2, dengan nilai 0.707106781186
- **Math.SQRT2**: Berisi hasil akar kuadrat dari 2, dengan nilai 1.4142135623730951



Berikut adalah contoh penggunaan konstanta **objek Math** di dalam JavaScript

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
   console.log(Math.E);
   console.log(Math.LN10);
   console.log(Math.LN2);
   console.log(Math.LOG10E );
   console.log(Math.LOG2E);
   console.log(Math.PI);
   console.log(Math.SQRT1_2);
   console.log(Math.SQRT2);

   // penggunaan di dalam operasi matematika
   var jari2 = 7
   var luas_lingkaran = Math.PI * jari2 * jari2;
   console.log(luas_lingkaran);
 </script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar konstanta objek Math JavaScript </p>
</div>
```

```
</body>
</html>
```

Pada baris terakhir kode **JavaScript** diatas, saya menghitung rumus *luas lingkaran* dengan menggunakan kosntanta **Math.PI**.

Fungsi untuk Objek Math dalam JavaScript

Selain *konstanta*, **Objek Math** juga menyediakan banyak fungsi *matematis*. Anda mungkin akan jarang menggunakan *fungsi* atau *method* ini, kecuali membuat aplikasi ilmiah atau aplikasi kalkulator dengan **JavaScript**.

Berikut adalah *method* yang disediakan oleh **objek Math JavaScript**, diurutkan berdasarkan abjad, dan akan kita bahas satu per satu:

- `Math.abs()`
- `Math.acos()`
- `Math.asin()`
- `Math.atan()`
- `Math.atan2()`
- `Math.ceil()`
- `Math.cos()`
- `Math.exp()`
- `Math.floor()`
- `Math.log()`
- `Math.max()`
- `Math.min()`
- `Math.pow()`
- `Math.random()`
- `Math.round()`
- `Math.sin()`
- `Math.sqrt()`
- `Math.tan()`

Method atau fungsi yang akan kita bahas ini lebih tepatnya disebut dengan *static function* dari **objek Math**. **Math** juga bukan *class* dari **objek** seperti **Number** atau **String**. Di dalam JavaScript tidak ada **Math()** *constructor*.

Method Objek Math JavaScript: Math.abs()

Method **Math.abs** berfungsi untuk menghasilkan **nilai absolut** (nilai *negatif* akan menjadi *positif*, sedangkan nilai *positif* akan tetap *positif*). Fungsi ini membutuhkan 1 argumen angka. Berikut adalah contoh pemanggilan fungsi **abs()**:

```
<script>
Math.abs(-5)      // hasilnya: 5
Math.abs(5)       // hasilnya: 5
Math.abs(-22.78)  // hasilnya: 22.78
</script>
```

Method Objek Math JavaScript: Math.acos()

Method **Math.acos()** berfungsi untuk menghitung nilai **arccosine**. Fungsi ini membutuhkan 1 argumen angka dengan nilai antara -1 sampai dengan 1. Nilai akhir fungsi adalah **0** sampai dengan **π** *radian*.

Method Objek Math JavaScript: Math.asin()

Method **Math.asin()** berfungsi untuk menghitung nilai **arcsine**. Fungsi ini membutuhkan 1 argumen angka dengan nilai antara -1 sampai dengan 1. Nilai akhir fungsi adalah **-π/2** sampai dengan **π/2** *radian*.

Method Objek Math JavaScript: Math.atan()

Method **Math.atan()** berfungsi untuk menghitung nilai **arctangent**. Fungsi ini membutuhkan 1 argumen angka dengan nilai apapun. Nilai akhir fungsi adalah **-π/2** sampai dengan **π/2** *radian*.

Method Objek Math JavaScript: Math.atan2()

Method **Math.atan2()** berfungsi untuk menghitung nilai **arctangent** dari rasio **y/x**. Fungsi ini membutuhkan 2 buah argumen untuk nilai **y** dan **x**. Nilai hasil fungsi adalah diantara **-π** dan **π** *radians*.

Method Objek Math JavaScript: Math.ceil()

Method **Math.ceil()** berfungsi untuk pembulatan keatas dari sebuah nilai desimal. Fungsi ini membutuhkan 1 argumen, yaitu angka yang akan dibulatkan. Berikut adalah hasil pemanggilan fungsi ceil:

```
<script>
Math.ceil(1.99);    // hasilnya: 2
Math.ceil(1.01);    // hasilnya: 2
Math.ceil(1.0);     // hasilnya: 1
Math.ceil(-1.99);   // hasilnya: -1
</script>
```

Method Objek Math JavaScript: Math.cos()

Method **Math.cos()** berfungsi untuk menghitung nilai **cosinus**. Fungsi ini membutuhkan 1 buah argumen dalam bentuk sudut dengan nilai *radian*. Untuk menkonversi derajat menjadi *radian*, kalikan besar sudut dengan 0.017453293 (2π/360). Nilai akhir fungsi ini berada antara −1.0 dan 1.0.

Method Objek Math JavaScript: Math.exp()

Method **Math.exp()** digunakan untuk menghitung hasil dari **e^x** dimana **x** adalah argumen yang diberikan. **e** merupakan logaritma natural dengan nilai 2.718.

Method Objek Math JavaScript: Math.floor()

Method **Math.floor()** berfungsi untuk pembulatan kebawah dari sebuah nilai desimal. Fungsi ini membutuhkan 1 argumen, yaitu angka yang akan dibulatkan. Berikut adalah hasil pemanggilan fungsi floor:

```
<script>
Math.floor(1.99);    // hasilnya: 1
Math.floor(1.01);    // hasilnya: 1
Math.floor(1.0);     // hasilnya: 1
Math.floor(-1.01);   // hasilnya: -2
</script>
```

Method Objek Math JavaScript: Math.log()

Method **Math.log()** berfungsi untuk menghitung nilai *logaritma natural*, yaitu nilai dari **log e x**. Fungsi ini membutuhkan 1 buah argumen angka.

Method Objek Math JavaScript: Math.max()

Method **Math.max()** berfungsi untuk mencari angka paling besar diantara argumen yang diberikan. Fungsi ini membutuhkan 1 atau lebih argumen. Berikut adalah contoh penggunaan fungsi **Math.max():**

```
<script>
Math.max(1,2,3,4,5,6);     // hasilnya: 6
Math.max(10,20,30,90,5);   // hasilnya: 90
Math.max(50);              // hasilnya: 50
</script>
```

Method Objek Math JavaScript: Math.min()

Method **Math.min()** berfungsi untuk mencari angka paling kecil diantara argumen yang diberikan. Fungsi ini membutuhkan 1 atau lebih argumen. Berikut adalah contoh penggunaan fungsi **Math.min():**

```
<script>
Math.min(1,2,3,4,5,6);       // hasilnya: 1
Math.mmin(10,20,30,90,5);    // hasilnya: 5
Math.max(50);                // hasilnya: 50
</script>
```

Method Objek Math JavaScript: Math.pow()

Method **Math.pow()** berfungsi untuk untuk mencari hasil pemangkatan. Fungsi ini membutuhkan 2 buah argumen. *Argumen pertama* adalah angka asal, dan

argumen kedua adalah nilai pangkat. Berikut adalah contoh penggunaan fungsi **Math. pow():**

```
<script>
Math.pow(1,100);      // 1^100, hasilnya: 1
Math.pow(5,3);        // 5^3, hasilnya: 125
Math.pow(50);         // hasilnya: 50
</script>
```

Method Objek Math JavaScript: Math.random()

Method **Math.random()** berfungsi untuk menghasilkan angka acak dalam setiap pemanggilan. Fungsi ini tidak membutuhkan argumen apapun. Nilai akhir berada dalam rentang 0 dan 1. Untuk hasil angka acak 1-100, kita tinggal mengalikan hasil fungsi ini dengan 100. Berikut adalah contoh penggunaan fungsi **Math.random():**

```
<script>
Math.random();    // hasilnya: 0.2605599395465106
Math.random();    // hasilnya: 0.6355015402659774
Math.random();    // hasilnya: 0.5217791700270027
</script>
```

Method Objek Math JavaScript: Math.round()

Method **Math.round()** berfungsi untuk membulatkan nilai angka ke bilangan terdekat. Jika nilai desimal dibawah 0.5 maka akan dibulatkan ke bawah, namun jika nilai desimal bernilai 0.5 atau lebih, akan dibulatkan keatas. Fungsi ini membutuhkan 1 argumen, yaitu angka yang akan dibulatkan. Berikut adalah hasil pemanggilan fungsi floor:

```
<script>
 Math.round(1.56);    // hasilnya: 2
 Math.round(1.44);    // hasilnya: 1
 Math.round(1.0);     // hasilnya: 1
 Math.round(-1.6);    // hasilnya: -2
 </script>
```

Method Objek Math JavaScript: Math.sin()

Method **Math.sin()** berfungsi untuk menghitung hasil **sinus**. Fungsi ini membutuhkan 1 buah argumen dalam bentuk sudut dengan nilai radian. Untuk menkonversi derajat menjadi radian, kalikan besar sudut dengan 0.017453293 (2π/360). Nilai akhir fungsi ini berada antara −1.0 dan 1.0.

Method Objek Math JavaScript: Math.sqrt()

Method Math.sqrt() digunakan untuk mencari hasil dari akar kuadrat sebuah angka. Fungsi ini membutuhkan 1 argumen, yaitu angka yang akan dihitung. Berikut adalah contoh penggunaan fungsi Math. sqrt();

```
<script>
Math.sqrt(25);   // hasilnya: 5
Math.sqrt(81);   // hasilnya: 9
Math.sqrt(-3);   // hasilnya: NaN
</script>
```

Method Objek Math JavaScript: Math.tan()

Method **Math.tan()** berfungsi untuk menghitung hasil **tangen**. Fungsi ini membutuhkan 1 buah argumen dalam bentuk sudut dengan nilai *radian*. Untuk menkonversi derajat menjadi *radian*, kalikan besar sudut dengan 0.017453293 (2π/360).

q. Mengenal Tipe Data String dan Operator String JavaScript

Pengertian Tipe Data String di Dalam JavaScript

Tipe data String di dalam **JavaScript** adalah tipe data yang terdiri dari kumpulan karakter yang berurutan. Atau di dalam penggunaan sehari-hari string adalah tipe data yang menampung nilai **text** atau kalimat.

Untuk membuat sebuah tipe data **string**, kita hanya tinggal menambahkan tanda kutip (bahasa inggris: **'quotes'**) pada awal dan akhir dari text. **JavaScript** mendukung penggunaan *tanda kutip satu* (') manupun *tanda kutip ganda* (''). Didalam sumber bahasa inggris sering disebut sebagai **single quote** dan **double quote**.

Di dalam **JavaScript**, kedua tanda kutip ini bisa digunakan secara terpisah, maupun secara bersamaan. Perhatikan contoh berikut ini:

```
<script>
var nama = "Sylvia";
var situs = 'jti.polije.com';
var pesan = 'dia berkata:"hello World!"';
var pesan2 = "Hari ini hari jum'at";
</script>
```

Jika sebuah **string** diinput dengan menggunakan karakter awal *tanda kutip satu*, maka juga harus diakhiri dengan *tanda kutip satu* juga, walaupun di dalam kalimat tersebut terdapat *tanda kutip dua*, dan begitu juga sebaliknya.

JavaScript menggunakan karakter set **Unicode** yang membutuhkan memory penyimpanan **16 bit** untuk 1 karakter. Hal ini berbeda jika dibandingkan dengan bahasa pemograman lain, seperti **PHP** yang tidak mendukung **Unicode** dan hanya membutuhkan memory **8 bit** untuk karakter-karakternya. Walaupun membutuhkan ruang memory 2 kali dari **PHP**, tetapi **JavaScript** dengan dengan karakter **Unicode**-nya, mendukung hampir seluruh *aksara* komputer seperti huruf *jepang* maupun karakter latin.

Kita bisa mendefenisikan variabel sebagai berikut ini di dalam JavaScript:

```
var p = "π";
var e = 'μ';
```

Penggunaan Karakter Khusus String: Escape Sequences

JavaScript memiliki cara penanganan untuk penggunaan karakter-karakter khusus (atau dikenal dengan istilah **Escape Sequences**). Karakter khusus ini termasuk karakter **new line** untuk baris baru, karakter **tab**, **tanda kutip**, dan lain-lain.

Untuk menggunakannya, kita harus men-'**escape**' karakter khusus ini dengan menggunakan tanda garis miring (\), atau disebut dengan karakter **backslash**.

Berikut adalah **Escape Sequences** di dalam JavaScript:

- **0**: Karakter NUL
- **b**: Backspace
- **t**: Horizontal tab
- **n**: Newline
- **v**: Vertical tab
- **f**: Form feed
- **r**: Carriage return
- **"**: Tanda kutip dua (double quote)
- \': Tanda kutip satu (apostrophe atau single quote)
- \\: Garis miring backslash
- **xXX**: Karakter Latin-1 dengan menggunakan dua digit heksa desimal XX
- **uXXXX**: Karakter Unicode dengan menggunakan empat digit heksa XXXX

Berikut adalah contoh penggunaan **Escape Sequences** di dalam JavaScript:

```
<script>
var nama = 'Juma\'in';
//hasil: Juma'in

var situs = 'http:\\\\www.jti.polije.com';
```

```
//hasil: http:\\www.jti.polije.com

var pesan = "dia berkata:I\"ll be back";
//hasil: dia berkata:I'll be back

var pesan2 = "Kalimat ini terdiri dari\n 2 baris";
// hasil: Kalimat ini terdiri dari
//        2 baris

var santai = "Saya sedang ngopi di caf\u00e9";
//hasil: Saya sedang ngopi di café

</script>
```

Di dalam contoh diatas, pada bagian variabel situs, saya membuat 4 buah karakter *backslash* karena *backslash* sendiri adalah karakter **escape**. Untuk variabel **pesan2**, kalimat tersebut akan menjadi 2 baris karena karakter escape **\n** berfungsi untuk '*pindah baris'*. Sedangkan variabel **santai** akan ditampilkan menjadi *"Saya sedang ngopi di café".* Karakter *é* ditulis menggunakan kode Unicode **\u00e9**.

Operator untuk operasi String di dalam JavaScript

Operasi yang sering dilakukan untuk tipe data **String** adalah operasi **penyambungan string**, atau dikenal dengan istilah '**concatenate string'**. Untuk operasi ini, **JavaScript** menggunakan operator tambah (+). Berikut contoh penggunaannya :

```
<script>
var a="Sub";
var b="Ilkom";
var situs = a + b;    // Jti.polije
</script>
```

JavaScript akan '*mendeteksi'* operasi tipe data pada saat menggunakan operator +. Jika kedua tipe data adalah **angka** (**number**), maka operasi yang akan dilakukan adalah *penjumlahan*, namun jika salah satu atau kedua variabel bertipe **String**, akan dilakukan operasi **penyambungan String**.

Berikut adalah contoh '*perubahan prilaku'* operasi + :

```
<script>
var a="Sub";
var b="Ilkom";
var c="14";
var d=12;
var e=3;

console.log(a+b);     // Jti.polije
console.log(a+c);     // Sub14
```

```
console.log(c+d);      // 1412
console.log(d+e);      // 15
</script>
```

Perhatikan bahwa variabel **c** adalah **String** karena kita menggunakan tanda kutip. Sehingga “14” + 12 di dalam JavaScript akan menghasilkan String “1412”, bukan 26.

String sebagai Array dari Karakter

Di dalam **JavaScript**, string bisa dianggap sebagai **array** dari karakter, dan kita bisa mengambil nilai sebuah karakter dari **String** dengan mengaksesnya seperti **array**.

Walaupun kita belum membahas tentang **array**, konsepnya untuk **String** sebagai array, bisa dipahami sebagai berikut: sebuah **string** dimulai dari array dengan index 0 untuk karakter pertama, index 1 untuk karakter kedua, dan seterusnya. Jika variabel **a=”jti.polije”**, maka hasil dari **a[0]** adalah **d**, dan **a[5]** adalah **i**.

Berikut contoh pengaksesan karakter dari sebuah string dalam **JavaScript**:

```
<script>
var situs = "jti.polije";
console.log(situs[0]);    // d
console.log(situs[1]);    // u
console.log(situs[2]);    // n
console.log(situs[3]);    // i
console.log(situs[4]);    // a
</script>
```

Contoh Penggunaan String di dalam JavaScript

Sebagai penutup dari perkenalan kita dengan tipe data String dalam JavaScript, berikut adalah contoh kode program untuk *pendefenisian String*, *penggunaan karakter escape*, *operasi penyambungan string*, dan *pengaksesan string sebagai array* dalam JavaScript:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Belajar JavaScript</title>

<script>
var belajar = 'Saya sedang belajar "JavaScript"';
console.log(belajar);         // Saya sedang belajar
"JavaScript"

var situs = 'http:\\\\www.jti.polije.com';
console.log(situs);           // http:\\www.jti.polije.com
```

```
var pesan = "dia berkata:I\"ll be back";
console.log(pesan);            // dia berkata:I"ll be back

var baris2 = "Kalimat ini terdiri dari \n 2 baris";
console.log(baris2);          // Kalimat ini terdiri dari
                              // 2 baris

var santai = "Saya sedang ngopi di caf\u00e9";
console.log(santai);          //Saya sedang ngopi di café

var a="Sub";
var b="Ilkom";
var c="14";
var d=12;
var e=3;
console.log(a+b);    // Jti.polije
console.log(a+c);    // Sub14
console.log(c+d);    // 1412
console.log(d+e);    // 15

var situs = "jti.polije";
console.log(situs[0]);      // d
console.log(situs[1]);      // u
console.log(situs[2]);      // n
console.log(situs[3]);      // i
console.log(situs[4]);      // a
</script>

</head>
<body>
<h1>Belajar JavaScript</h1>
<p> Saya sedang belajar Tipe Data String JavaScript di
jti.polije.com </p>
</div>
</body>
</html>
```

Di dalam tutorial kali ini, kita telah membahas tentang pendefenisian String, penggunaan karakter escape, operasi penyambungan string, dan pengaksesan string sebagai array di dalam **JavaScript**.

Namun seperti pembahasan tentang tipe data **number**, di dalam **JavaScript** tipe data **String** juga memiliki **method** dan **properti** yang bisa kita gunakan dalam manipulasi tipe data **String**.

r. Property dan Method (fungsi) Objek String JavaScript

Pengertian Propery dan Method (fungsi) dari Objek String

Walaupun tipe data **string** bukan di defenisikan menjadi **objek**, namun JavaScript '*memperlakukan'* tipe dasar **String** ini sebagai **Objek String**, sehingga memiliki **property** dan **method** yang dapat di gunakan.

Property dan **method** dari **objek String** semuanya mengembalikan nilai baru, dan tidak bisa mengubah nilai dalam variabel asal. **Variabel** asal String tetap bernilai seperti semula. Dalam pemograman sifat ini disebut dengan **immutable variable**.

Berikut adalah **Property** dan **Method** yang dimiliki **objek String** di dalam **JavaScript**. Kita akan membahasnya satu per satu. *Property* dan *Method* ini diurutkan secara abjad:

Property Objek String JavaScript:

- **string.length**

Method Objek String JavaScript:

- **string.charAt()**
- **string.charCodeAt()**
- **string.concat()**
- **string.indexOf()**
- **string.lastIndexOf()**
- **string.localeCompare()**
- **string.match()**
- **string.replace()**
- **string.search()**
- **string.slice()**
- **string.split()**
- **string.substr()**
- **string.substring()**
- **string.toLowerCase()**
- **string.toString()**
- **string.toUpperCase()**
- **string.trim()**
- **string.valueOf()**

Property Objek String: String.length

String.length merupakan property satu-satunya untuk **objek String**. **Property** ini akan mengembalikan nilai panjang karakter dari sebuah **String**. Berikut contoh penggunaannya:

```
<script>
var situs = "jti.polije";
console.log(situs.length);          // 10
var belajar = "belajar javascript";
console.log(belajar.length);        // 18
</script>
```

Method Objek String: String.charAt()

Method **String.charAt()** berfungsi untuk mengambil sebuah karakter dari **String**. Method ini membutuhkan 1 buah argumen angka yang diisi dengan posisi karakter yang akan diambil, dimulai dari urutan 0 untuk karakter pertama, urutan 1 untuk karakter kedua, dan seterusnya. Berikut contoh penggunaannya:

```
<script>
var situs = "jti.polije";
console.log(situs.charAt(0));     // d
console.log(situs.charAt(5));     // i
console.log(situs.charAt(7));     // k
console.log(situs.charAt(11));    // string kosong
</script>
```

Cara ini mirip dengan pengaksesan karakter String sebagai array, seperti yang pernah kita bahas pada tutorial: Mengenal Tipe Data String dan Operator String JavaScript.

Method Objek String: String.charCodeAt()

Method **String.charCodeAt()** berfungsi untuk mengambil nilai **Unicode** karakter dari **String**. Method ini membutuhkan 1 buah argumen angka yang diisi dengan posisi karakter yang akan diambil, dimulai dari urutan 0 untuk karakter pertama, urutan 1 untuk karakter kedua, dan seterusnya. Berikut contoh penggunaannya:

```
<script>
var situs = "jti.polije";
console.log(situs.charCodeAt(0));     // 100
console.log(situs.charCodeAt(5));     // 105
console.log(situs.charCodeAt(7));     // 107
console.log(situs.charCodeAt(11));    // NaN
</script>
```

Method Objek String: String.concat()

Method **String.concat()** berfungsi untuk operasi *penyambungan* String. Method ini membutuhkan 1 atau lebih argumen bertipe **String** untuk disambung. Berikut contoh penggunaannya:

```
<script>
var situs = "jti.polije";
```

```
console.log(situs.concat(" oke"));
// jti.polije oke

console.log(situs.concat(" situs", " belajar", "
javaScript"));
// jti.polije situs belajar javaScript
</script>
```

Operasi penyambungan String biasanya akan lebih mudah jika menggunakan operator "+".

Method Objek String: String.indexOf()

Method **String.indexOf()** berfungsi untuk mencari karakter atau kata dalam sebuah **String**. Method ini membutuhkan 2 argumen: 1 argumen bertipe **String** yang diisi dengan karakter yang akan dicari, serta 1 buah argumen opsional yang berfungsi untuk menandai posisi awal pencarian. Jika argumen kedua tidak diisi, maka pencarian akan dimulai dari awal **String**.

Hasil kembalian method **String.indexOf()** adalah posisi dari karakter yang ditemukan, dimulai dari 0 untuk posisi karakter pertama. Method ini akan mengembalikan nilai -1 jika karakter tidak ditemukan.

Berikut contoh penggunaannya:

```
<script>
var situs = "jti.polije";
console.log(situs.indexOf("k"));   // 7
console.log(situs.indexOf("i"));   // 3
console.log(situs.indexOf("i",4));
// 5 (pencarian dimulai dari karakter ke-5)

var belajar = "Saya sedang belajar javaScript di
jti.polije.com";
console.log(belajar.indexOf("sedang"));    // 4

//  (pencarian dimulai dari karakter ke-18)
console.log(belajar.indexOf("ja",17));     // 20

console.log(belajar.indexOf("php"));       // -1 (tidak
ditemukan)
</script>
```

Method Objek String: String.localeCompare()

Method **String.localeCompare()** berfungsi untuk membandingkan 2 buah string. Method ini membutuhkan 1 buah argumen bertipe string yang berisi string yang akan dibandingkan. Jika **String** asal '*kurang dari'* string **argumen**, method ini akan mengembalikan nilai <0, Jika String asal '*lebih dari'* string argumen, method ini akan mengembalikan nilai >0, dan akan mengembalikan 0 jika **string** tersebut sama.

Berikut contoh penggunaannya:

```
<script>
var c = "c";
console.log(c.localeCompare("c"));    // 0
console.log(c.localeCompare("a"));    // 1
console.log(c.localeCompare("d"));    // -1

var kota = "Jakarta";
console.log(kota.localeCompare("jakarta"));   // 1 (case
sensitif)
console.log(kota.localeCompare("padang"));    // -1
console.log(kota.localeCompare("bandung"));   // 1
</script>
```

Method Objek String: String.match()

Method **String.match()** berfungsi untuk menemukan string menggunakan **regular expression**. Method ini membutuhkan 1 buah argumen yang berisi format **regular expression** yang akan dicari. Hasil akhir method ini adalah **array** yang berisi data seluruh string yang ditemukan.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "1 tambah 2 sama dengan 3";
var hasil = kalimat.match(/\d+/g); // cari semua angka
console.log(hasil);     // hasil: ["1", "2", "3"]
</script>
```

Method Objek String: String.replace()

Method **String.replace()** berfungsi untuk pencarian string menggunakan **regular expression**, dan mengganti kata yang dicari. Method ini merupakan fungsi **find and replace** di dalam **JavaScript**. Method **String.replace()** membutuhkan 2 buah argumen: argumen pertama adalah format **regular expression** yang akan dicari, dan argumen kedua adalah String (atau bisa juga berupa fungsi) yang akan menggantikan text hasil pencarian. Hasil akhir method ini adalah sebuah **String** akhir yang telah diubah.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "Saya sedang belajar PHP di Tutorial PHP
jti.polije.com";

// cari semua kata "PHP", ganti menjadi "JavaScript";
var hasil = kalimat.replace(/PHP/g,"JavaScript");

// hasil: Saya sedang belajar JavaScript di Tutorial
// JavaScript jti.polije.com
console.log(hasil);
</script>
```

Method Objek String: String.search()

Method **String. search()** berfungsi untuk menemukan string menggunakan **regular expression**. Method ini membutuhkan 1 buah argumen yang berisi format **regular expression** yang akan dicari. Hasil akhir method ini adalah angka yang menunjukkan posisi kata yang dicari (huruf pertama dimulai dari index 0). Method ini mirip dengan **String.indexOf(),** namun perbedaan terletak pada argumen yang merupakan **regular expression**.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "Saya sedang belajar JavaScript di Tutorial
JavaScript jti.polije.com ";
var hasil = kalimat.search(/JavaScript/g);    // cari kata
"JavaScript"
console.log(hasil);     // 20 (JavaScript berada di karakter
ke-21)
</script>
```

Method Objek String: String.slice()

Method **String.slice()** berfungsi untuk '*memotong'* string menjadi string baru. Method ini membutuhkan 2 buah argumen angka yang berisi **posisi awal** dan **posisi akhir** string yang akan 'dipotong'. Posisi ini dimulai dari 0 untuk karakter pertama. Jika argumen bernilai *negatif*, maka hal itu berarti perhitungan dimulai dari akhir string.

Hasil akhir method ini adalah String baru hasil pemotongan. Method **String.slice()** tidak akan mengubah string asal, namun mengembalikan string baru.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "Belajar JavaScript";

// potong dari index ke 3 sampai dengan ke 7.
var hasil = kalimat.slice(3,7)
console.log(hasil);    // ajar

// potong dari index ke 9 dari awal sampai dengan index ke 6
dari akhir.
var hasil2 = kalimat.slice(8,-6)
console.log(hasil2);    // Java
</script>
```

Method Objek String: String.split()

Method **String.split()** berfungsi untuk memisahkan string menjadi array. Method ini membutuhkan 2 buah argumen: argumen pertama berupa karakter '*pembatas'* untuk memisahkan string, dan argumen kedua bersifat *opsional* yang berisi seberapa banyak **array** yang akan dihasilkan. Jika argumen kedua tidak ditulis, maka seluruh hasil akan dikembalikan berapapun jumlahnya.

Hasil akhir method ini adalah sebuah **array** yang terdiri dari kata-kata yang terpisahkan dari **String asal**.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "Satu,Dua,Tiga,Empat,Lima";
// pisahkan String "kalimat" menjadi array dengan pemisah
tanda koma
var hasil = kalimat.split(",")
console.log(hasil);    // ["Satu"," Dua", "Tiga", "Empat",
"Lima"]

var kalimat2 = "Satu;Dua;Tiga;Empat;Lima";
// pisahkan String "kalimat" menjadi array dengan pemisah
tanda ";"
// batasi array hanya 3 isian
var hasil2 = kalimat2.split(";",3)
console.log(hasil2);    // ["Satu", "Dua", "Tiga"]
</script>
```

Method Objek String: String.substr()

Method **String.substr()** berfungsi untuk '*mengambil'* bagian string asal untuk menjadi string baru. Method ini membutuhkan 2 buah argumen: argumen pertama berupa angka yang berisi posisi awal dari string yang akan 'diambil', dan argumen kedua berisi berapa karakter yang akan diambil. Posisi string asal dimulai dengan angka 0 untuk karakter pertama. Jika argumen pertama bernilai negatif, berarti perhitungan dimulai dari akhir string.

Hasil akhir method ini adalah String baru hasil pengambilan. Method **String. substr()** tidak akan mengubah string asal, namun mengembalikan string baru.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "Belajar JavaScript";

// ambil mulai dari index ke 3, sebanyak 9 karakter.
var hasil = kalimat.substr(3,9)
console.log(hasil);    // ajar Java

// ambil mulai dari index ke 6 dari akhir string
var hasil2 = kalimat.substr(-6)
console.log(hasil2);    // Script
</script>
```

Method Objek String: String.substring()

Method **String. substring()** berfungsi untuk mengambil string menjadi potongan string baru. Method ini membutuhkan 2 buah argumen angka yang berisi posisi awal dan akhir string yang akan diambil. Posisi ini dimulai dari 0 untuk

karakter pertama. Berbeda dengan method **String.slice(),** argumen kedua dari method ini tidak bisa diisi dengan nilai *negatif*.

Hasil akhir method ini adalah String baru hasil pemotongan. Method **String.slice()** tidak akan mengubah string asal, namun mengembalikan string baru.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "Belajar JavaScript";

// potong dari index ke 3 sampai dengan ke 7.
var hasil = kalimat.substring(3,7)
console.log(hasil);    // ajar

// potong dari index ke 8 dari awal sampai dengan akhir
String.
var hasil2 = kalimat.substring(8)
console.log(hasil2);   // JavaScript
</script>
```

Method Objek String: String.toLowerCase() dan String.toLocaleLowerCase()

Method **String.toLowerCase()** dan **String.toLocaleLowerCase()** berfungsi untuk mengubah String asal menjadi huruf kecil. Perbedaan antara **toLowerCase()** dengan **toLocaleLowerCase()**, bahwa pada method **toLocaleLowerCase()** JavaScript menkonversinya dengan aturan lokal browser. Namun pada penggunaan normal dengan *bahasa Indonesia* atau *bahasa Inggris*, kedua fungsi ini akan mengembalikan nilai yang sama

Hasil akhir method ini adalah String baru hasil pengubahan. Method **String.toLowerCase()** dan **String.toLocaleLowerCase()** tidak akan mengubah string asal, namun mengembalikan string baru.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "Belajar JavaScript";
console.log(kalimat.toLocaleLowerCase());  // hasil: belajar
javascript
console.log(kalimat.toLowerCase());        // hasil: belajar
javascript

var kalimat2 = "bELAJAR JaVaScrIPT";
console.log(kalimat2.toLocaleLowerCase()); // hasil: belajar
javascript
console.log(kalimat2.toLowerCase());       // hasil: belajar
javascript
</script>
```

Method Objek String: String.toUpperCase() dan String.toLocaleUpperCase()

Method **String.toUpperCase()** dan **String.toLocaleUpperCase()** berfungsi untuk mengubah String asal menjadi huruf besar. Perbedaan antara **toUpperCase()**

dengan **toLocaleUpperCase(),** bahwa pada method **toLocaleUpperCase()** JavaScript menkonversinya dengan aturan lokal browser. Namun pada penggunaan normal dengan *bahasa Indonesia* atau *bahasa Inggris*, kedua fungsi ini akan mengembalikan nilai yang sama

Hasil akhir method ini adalah String baru hasil pengubahan. Method **String.toUpperCase()** dan **String.toLocaleUpperCase()** tidak akan mengubah string asal, namun mengembalikan string baru.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "Belajar JavaScript";
console.log(kalimat.toLocaleUpperCase());  // hasil: BELAJAR
JAVASCRIPT
console.log(kalimat.toUpperCase());        // hasil: BELAJAR
JAVASCRIPT

var kalimat2 = "bELAJAR JaVaScrIPT";
console.log(kalimat2.toLocaleUpperCase()); // hasil: BELAJAR
JAVASCRIPT
console.log(kalimat2.toUpperCase());       // hasil: BELAJAR
JAVASCRIPT
</script>
```

Method Objek String: String.toString() dan String.valueOf()

Method **String.toString()** dan **String.valueOf()** berfungsi untuk memanggil tipe data '**primitif'** *String* dari **Objek String**. Kita akan jarang menggunakan kedua method ini. Dan jika digunakan, akan mengembalikan nilai String asal.

Berikut contoh penggunaannya:

```
<script>
var kalimat = "Belajar JavaScript";
console.log(kalimat.toString());      // hasil: Belajar
JavaScript
console.log(kalimat.valueOf());       // hasil: Belajar
JavaScript

var kalimat2 = "bELAJAR JaVaScrIPT";
console.log(kalimat2.toString());     // hasil: bELAJAR
JaVaScrIPT
console.log(kalimat2.valueOf());      // hasil: Belajar
JavaScript
</script>
```

Method Objek String: String.trim()

Method **String.trim()** berfungsi untuk '*membuang'* bagian spasi atau string kosong di awal dan diakhir String. Biasanya method ini dipanggil untuk memastikan input dari user tidak mengandung spasi di awal dan akhir inputan.

Hasil akhir method ini adalah String baru hasil pemrosesan. Method **String.trim()** tidak akan mengubah string asal, namun mengembalikan string baru.

Berikut contoh penggunaannya:

```
<script>
var nama = " Chelsea Susanti   ";
console.log(nama);            // hasil:" Chelsea Susanti   "
console.log(nama.trim());     // hasil: "Chelsea Susanti"
</script>
```

s. Mengenal Tipe Data Boolean di dalam JavaScript

Pengertian Tipe Data Boolean dalam JavaScript

Tipe data **Boolean** adalah tipe data yang hanya mempunyai dua nilai, yakni benar (**True**) atau salah (**False**). Tipe data boolean sering digunakan untuk membuat alur logika program. Struktur logika seperti **if, else, while,** dan **do while**, membutuhkan nilai **boolean** sebagai '*pengontrol'* alur program.

Tipe data **boolean** juga merupakan hasil yang didapat dari *operasi perbandingan*. Misalkan apakah variabel **a** sama dengan **b**, atau apakah **a** lebih besar dari **b**.

Cara Mendefenisikan Tipe Data Boolean

Untuk membuat tipe data **boolean** di dalam **JavaScript**, kita cukup memberikan nilai **true**, atau **false** ke dalam sebuah ***variabel***. Berikut adalah contoh pembuatan tipe data **boolean** di dalam **JavaScript**:

```
<script>
var a = true;
var benar = true;
var salah = false;
</script>
```

Konversi Tipe Data Menjadi Boolean

Di dalam **JavaScript**, sebuah tipe data akan '*berubah'* sifatnya tergantung kapan tipe data itu digunakan. Misalkan kita membuat variabel **a = 12**. Variabel **a** dalam contoh ini akan menjadi tipe data number. Namun jika digunakan di dalam struktir logika seperti **if (a)**, maka **a** akan '*bersifat'* menjadi **boolean** dengan nilai **true**.

Aturan konversi tipe data menjadi boolean ini sering menjadi sumber 'error' jika tidak dipahami. Di dalam **JavaScript**, terdapat 6 nilai yang akan menghasilkan boolean **false**, atau disebut juga dengan **falsy value**.

Nilai-nilai berikut ini akan dianggap **false** di dalam JavaScript:

- **0**
- **-0**
- **NaN**

- **“” (string kosong)**
- **undefined**
- **null**

Selain nilai yang dicantumkan diatas, nilai lain akan dianggap sebagai **true** di dalam **JavaScript**, termasuk **array** dan **objek** ‘*kosong’*.

Method untuk Objek Boolean

Sama seperti tipe data **number** dan **string**, tipe data **boolean** juga memiliki **method** atau fungsi yang bisa ‘*dipanggil’*. Namun tidak seperti tipe data **number** dan **string** yang memiliki banyak *method*, tipe data boolean hanya memiliki 2 buah **method**, yakni **toString()** dan **valueOf().**

Method **toString()** akan menkonversi nilai boolean menjadi **string**. Nilai boolean **true** akan menjadi **“true”,** dan nilai boolean **false** akan menjadi **“false”.**

Method **valueOf()** akan menghasilkan nilai ‘*primitif’* dari boolean, kita akan jarang memanggil method ini, namun jika digunakan akan menghasilkan nilai yang sama dengan method **toString().**

Berikut adalah contoh penggunaannya:

```
<script>
var a = true;
var b = false;

console.log(a.toString()); // true
console.log(b.toString()); // false

console.log(a.valueOf()); // true
console.log(b.valueOf()); // false
</script>
```

t. Cara Penggunaan Operasi Perbandingan dan Logika

Operator Perbandingan dalam JavaScript

Di dalam **JavaScript** (dan juga bahasa pemograman lain) *operator perbandingan* adalah operator yang digunakan untuk membandingkan sebuah nilai atau **variabel** dengan **variabel** lainnya. Hasil dari operasi perbandingan ini akan menghasilkan nilai **boolean**.

Operator perbandingan di dalam **JavaScript** adalah sebagai berikut:

Operator sama dengan (==)

Operator **sama dengan** adalah operator yang akan membandingkan 2 buah nilai atau *variabel* dan menghasilkan nilai **true** jika variabel tersebut *bernilai sama*. Berikut adalah contoh programnya:

```
<script>
var a = true;
var benar = true;
console.log(a==benar); // true

var a = 12;
var b = 4;
console.log(a==b); // false

var a = 7;
var b = "7";
console.log(a==b); // true !
</script>
```

Perhatikan persamaan pada baris terakhir. Operasi == tidak melihat tipe data dari *variabel* yang akan dibandingkan, sehingga 7 (tipe data **number**) akan dianggap sama dengan "7" (tipe data **string**). Jika anda ingin membandingkan kedua variabel ini, dan memasukkan jenis tipe data sebagai salah satu penilaian sama atau tidaknya 2 buah variabel, maka harus menggunakan operator **identikal** (===).

Operator identik dengan (===)

Operator identikal === hampir sama dengan operator **==,** yaitu membandingkan apakah 2 buah variabel atau hasil operasi program sama atau tidak. Perbedaannya, operator === lebih '*ketat aturan'* daripada operator ==. Operasi 7 == "7" akan dianggap sama dan menghasilkan nilai **true**, namun operasi 7 === "7" akan dianggap **false**, karena tipe data kedua nilai ini berbeda.

Berikut adalah contoh penggunaannya:

```
<script>
var a = true;
var benar = true;
console.log(a===benar); // true

var a = 12;
var b = 4;
console.log(a===b); // false

var a = 7;
var b = "7";
console.log(a===b); // false !

var a = "7";
var b = "7";
console.log(a===b); // true
</script>
```

Operator tidak sama dengan (!=)

Operator **!=** adalah kebalikan dari operator ==, dan akan menghasilkan nilai **true** jika hasil operasi 2 buah variabel yang dibandingkan tidak memiliki nilai yang sama. Berikut adalah contoh penggunannya:

```
<script>
var a = true;
var benar = true;
console.log(a!=benar); // false

var a = 12;
var b = 4;
console.log(a!=b); // true

var a = 7;
var b = "7";
console.log(a!=b); // false !
</script>
```

Perhatikan juga untuk persamaan baris terakhir, operator **!=** tidak mempertimbangkan tipe data variabel, sama seperti operator ==. Jika anda ingin jenis tipe data juga merupakan kriteria perbandingan, maka gunakan operator **!==**.

Operator tidak identik dengan (!==)

Jika operator **!=** tidak mempertimbangkan tipe data, maka operator **!==** hanya akan **false** jika operator yang dibandingkan memiliki nilai yang sama dan juga tipe data yang sama. Berikut adalah contoh penggunaannya:

```
<script>
var a = true;
var benar = true;
console.log(a!==benar); // false

var a = 12;
var b = 4;
console.log(a!==b); // true

var a = 7;
var b = "7";
console.log(a!==b); // true !

var a = "7";
var b = "7";
console.log(a!==b); // false
</script>
```

Operator Kurang dari (<) dan Kurang sama dengan dari: (<=)

Operator < dan <= hanya akan bernilai **true** jika **variabel** di sisi kiri operator memiliki nilai yang kurang dari **variabel** di sisi kanan. Perbedaan antara < dan <=

adalah jika kedua nilai yang dibandingkan sama, maka operator < akan menghasilkan **false**, namun operator <= akan menghasilkan **true**. Berikut adalah contoh programnya:

```
<script>
var a = 3;
var b = 4;
console.log(a<b); // true
console.log(a<=b); // true

var a = 5;
var b = 5;
console.log(a<b); // false
console.log(a<=b); // true
</script>
```

Operator Besar dari (>) dan Besar sama dengan dari (>=)

Operator > dan >= hanya akan bernilai **true** jika variabel di sisi **kiri** operator memiliki nilai yang lebih besar dari variabel di sisi **kanan**. Perbedaan antara > dan >= adalah jika kedua nilai yang dibandingkan sama, maka operator > akan menghasilkan **false**, namun operator >= akan menghasilkan true. Berikut adalah contoh programnya:

```
<script>
var a = 3;
var b = 4;
console.log(a>b); // false
console.log(a>=b); // false

var a = 5;
var b = 5;
console.log(a>b); // false
console.log(a>=b); // true
</script>
```

Walaupun di dalam contoh yang kita jalankan, operasi perbandingan menggunakan tipe data angka (**number**), namun operasi perbandingan di dalam **JavaScript** juga bisa dilakukan untuk tipe data **String**. Operasi perbandingan string di dalam JavaScript dilakukan secara bit per bit atau dengan melihat urutan dari kode *Unicode*nya.

Urutan dari karakter **unicode** dapat dicari di google, atau kunjungi *unicode-table.com*

Dari tabel **Unicode** ini kita dapat melihat urutan karakter dari 0000 sampai dengan FFFF. Karakter dengan kode unicode **0022** akan lebih kecil dari karakter dengan kode unicode **0023**.

Jika string yang dibandingkan terdiri dari beberapa karakter, hasil perbandingan akan ditentukan dari karakter paling kiri ke kanan. Jika *karakter pertama* untuk kedua variabel sama, maka dilanjutkan ke *karakter kedua*, dan demikian seterusnya sampai ditemukan karakter yang memiliki kode **unicode** yang berbeda.

Berikut adalah contoh kode program perbandingan **String** di dalam JavaScript:

```
<script>
var a = "a";
var b = "b";
console.log(a<b); // true

var a = "apple";
var b = "applu";
console.log(a==b); // false

var a = "@ku";
var b = "?ku";
console.log(a>b); // true
</script>
```

Operasi Logika dalam JavaScript

Selain *operasi perbandingan*, **operasi logika** juga sangat berkaitan dengan tipe data **boolean**. Operasi logika ini juga sering digunakan untuk pembuatan alur program.

Berikut adalah operator logika dan cara penulisan operator logika di dalam JavaScript:

- Operator "**dan**" (**and**), ditulis dengan **&&**. Operator **and** hanya akan menghasilkan nilai **true** jika kedua nilai yang dibandingkan juga bernilai **true**, dan menghasilkan nilai **false** jika salah satu atau kedua nilai yang dibandingkan adalah **false**.
- Operator "**atau**" (**or**), ditulis dengan ||. Operator **or** akan menghasilkan nilai **true** jika salah satu atau kedua nilai yang dibandingkan adalah **true**. Jika kedua nilai yang dibandingkan **false**, maka hasilnya adalah **false**.

- Operator **negasi**, ditulis dengan **!**. Operator ini digunakan untuk *membalik* nilai logika. Jika ditulis a adalah true maka !a adalah false dan begitu juga sebaliknya.

Berikut adalah contoh kode program penggunaan operator logika di dalam JavaScript:

```
<script>
var a = true;
var b = false;

var hasil1 = a && b;
console.log(hasil1); //false

var hasil2 = a && a;
console.log(hasil2); //true

var hasil3 = a || b;
console.log(hasil3); //true

var hasil4 = !a;
console.log(hasil4); //false
</script>
```

Di dalam **JavaScript** tidak dikenal operator logika **XOR**, yakni operator yang akan menghasilkan nilai true jika salah satu bernilai true, tetapi akan menghasilkan false jika kedua nilai true atau kedua nilai false. Namun hal ini bisa *'diakali'* dengan menggunakan kode seperti berikut ini: **!a != !b**.

Berikut contoh penggunaannya:

```
<script>
var a = true;
var b = false;

var xor1 = !a != !b;
console.log(xor1); // true

var xor2 = !a != !a;
console.log(xor2); // false
</script>
```

D. Bootstrap

– https://getbootstrap.com/
– https://www.malasngoding.com/pengertian-dan-cara-menggunakan-bootstrap/

Bootstrap adalah sebuah library framework CSS yang di buat khusus untuk bagian pengembangan front-end website. bootstrap merupakan salah satu framework HTML, CSS dan javascript yang paling populer di kalangan web developer. pada saat ini

hampir semua web developer telah menggunakan bootstrap untuk membuat tampilan front-end menjadi lebih mudah dan sangat cepat. karena anda hanya perlu menambahkan class-class tertentu untuk misalnya membuat tombol, grid, navigasi dan lainnya.

Bootstrap telah menyediakan kumpulan komponen class interface dasar yang telah di rancang sedemikian rupa untuk menciptakan tampilan yang menarik, bersih dan ringan. selain komponen class interface, bootstrap juga memiliki fitur grid yang berfungsi untuk mengatur layout pada halaman website yang bisa digunakan dengan sangat mudah dan cepat. dengan menggunakan bootstrap kita juga di beri keleluasaan dalam mengembangkan tampilan website yang menggunakan bootstrap yaitu dengan cara mengubah tampilan bootstrap dengan menambahkan class dan CSS sendiri.

Tentu anda bertanya-tanya kenapa sangat banyak yang telah menggunakan bootstrap dalam pengembangan website. berikut ini akan di jelaskan beberapa kegunaan yang telah menjadi kelebihan pada bootsrap. adapun beberapa kelebihan bootstrap adalah sebagai berikut:

- Penggunaan bootstrap sangat menghemat waktu.
- Tampilan bootstrap yang sudah cukup terlihat modern.
- Mobile Friendly yang maksudnya tampilan bootstrap sudah sangat responsive, yaitu tampilan bootstrap sudah mendukung segala jenis resolusi, baik itu pc, laptop, tablet dan smartphone.
- Sangat ringan karena bootstrap di buat dengan sangat terstruktur.
- Dan masih banyak lagi kelebihan dan kegunaan dari bootstrap yang akan anda temukan sendiri setelah mencoba membangun sebuah aplikasi berbasis web dengan menggunakan bantuan framework css bootstrap.

3. Alat dan Bahan

A. Kertas A4 (disediakan Politeknik Negeri Jember)
B. Kertas F4 (disediakan Politeknik Negeri Jember)
C. Folio Bergaris (disediakan Politeknik Negeri Jember)
D. BoardMarker (disediakan Politeknik Negeri Jember)
E. Komputer (disediakan Politeknik Negeri Jember dengan jumlah sesuai kapasitas lab)
F. XAMPP
G. Sublime Text 3
H. GitHub Desktop
I. Bootstrap

4. Kegiatan Praktik

A. Mahasiswa berkumpul dengan anggota kelompoknya
B. Masing-masing mahasiswa membuat merefresh GitHub Dekstop (pull)
C. Masing-masing mahasiswa mengikuti langkah praktikum sebagai berikut:

- https://www.malasngoding.com/belajar-php-tipe-data-pada-php/
- https://www.malasngoding.com/belajar-html-membuat-paragraf-pada-html/
- https://www.malasngoding.com/belajar-html-membuat-tabel-pada-html/
- https://www.malasngoding.com/belajar-html-membuat-hyperlink-pada-html/
- https://www.malasngoding.com/belajar-html-membuat-list-pada-html/
- https://www.malasngoding.com/belajar-html-atribut-form-pada-html/
- https://www.malasngoding.com/menampilkan-gambar-pada-html/
- https://www.malasngoding.com/belajar-html-menghubungkan-html-dengan-css/
- https://www.malasngoding.com/belajar-html-mengenal-class-dan-id-pada-html/

D. Masing-masing mahasiswa melakuka pull dan push untuk menggungah pekerjaan pada repositori

5. Refrensi *Website*

[1] https://www.w3schools.com/
[2] https://www.malasngoding.com/
[3] https://www.codepolitan.com/
[4] http://www.duniailkom.com/

PRAKTIK 3: Membuat Antar Muka WEB (HTML, CSS, JS dan Bootstrap) dan Menerapkan Perintah Esekusi Bahasa Pemrgraman (PHP)

Matakuliah : Workshop SI berbasis WEB
Minggu Ke : 4 dan 5
Waktu : 2 x 100 menit
Tema : Membuat Antar Muka WEB (HTML, CSS, JS dan Bootstrap) dan Menerapkan Perintah Esekusi Bahasa Pemrgraman (PHP)

1. **Kompetensi Dasar**
   A. Mahasiswa mampu membuat antar muka web dengan mengimplementasikan: HTML, CSS, JS dan Bootsrap
   B. Mahasiswa mampu menerapkan perintah esekusi bahasa pemrograman PHP

2. **Dasar Teori**

A. Implementasi Tampilan dengan Bootstrap

– https://getbootstrap.com/
– https://www.malasngoding.com/pengertian-dan-cara-menggunakan-bootstrap/

Bootstrap adalah sebuah library framework CSS yang di buat khusus untuk bagian pengembangan front-end website. bootstrap merupakan salah satu framework HTML, CSS dan javascript yang paling populer di kalangan web developer. pada saat ini hampir semua web developer telah menggunakan bootstrap untuk membuat tampilan front-end menjadi lebih mudah dan sangat cepat. karena anda hanya perlu menambahkan class-class tertentu untuk misalnya membuat tombol, grid, navigasi dan lainnya.

B. PHP

PHP (Akronim dari PHP adalah: Hypertext Preprocessor) merupakan widely-used open source general-purpose scripting language yang secara special didesain untuk web development dan dapat di-embedded/di-gabungkan dengan HTML [1].

Contoh #1 Contoh Awal

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
```

```
        <?php
            echo "Hi, I'm a PHP script!";
        ?>

    </body>
</html>
```

Walaupun terdapat banyak commands yang digunakan untuk menampilkan dalam bentuk HTML (seperti yang dimukan pada C atau Perl), PHP pages berisi koponen HTML dengan peambahan code yang "something" (pada konteks ini, keluaran "Hi, I'm a PHP script!"). Kode PHP yang digunakan sebenarnya sangat pendek, dimana kode PHP hanya yang dimulai instructions <?php and ?> yang sekaligus digunakan untuk membedakan "PHP mode" atau HTML mode.

Pengguna PHP awam mungkin akan mengalami kebingugan, seperti pada penggunaan client-server JavaScript, dimana pada konteks ini aplikasi akan dijalankan pada server, kemudian setelah diproses, hasilnya diproses memnjadi HTML dan dikirimkan kepada client. Client/pengguna akan mendapatkan hasil dengan menjalankan script tersbut, namun tidak mengetahui bagaimana proses dari script tersebut. Dilain sisi, bisa juga melakukan konfigurasi pada server untuk memproses file HTML yang didalmnya terdapat kode PHP, dengan catatan pengguna tidak mampu melihat kode PHP apa yang anda tuliskan saat diterima oleh pengguna.

C. Penerapan Perintah Esekusi Bahasa Pemrograman (PHP)

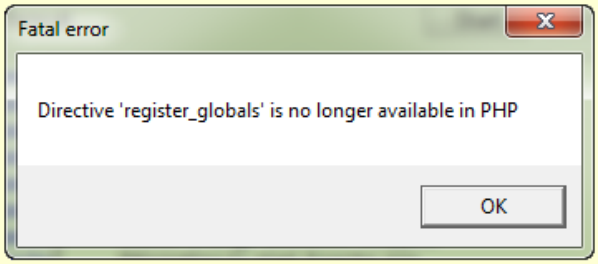
Pengertian Register Global dan Register Long Array

Register Global dan Register Long Array adalah metode lain dalam PHP yang bisa digunakan untuk menampilkan hasil form, namun kedua metode ini sekarang sudah menyandang status ‘deprecated’ yang berarti tidak disarakan digunakan dan kemungkinan tidak akan didukung oleh PHP pada masa mendatang. Kedua metode ini adalah metode ’legacy’ atau warisan yang digunakan pada versi PHP terdahulu.

Saya membahas Register Global dan Register Long Array hanya sebagai menambah pengetahuan, karena mungkin saja kita masih mendapati kode PHP lama (terutama versi PHP 4 kebawah) yang masih menggunakan kedua metode ini.

Baik Register Global dan Register Long Array sudah tidak disarankan untuk digunakan karena memiliki celah keamanan yang mudah dibobol. Anda disarankan untuk menggunakan variabel $_GET, $_POST atau $_REQUEST.

Update: **Register Global** dan **Register Long Array** sudah tidak didukung lagi dalam PHP. Jika anda menemukan pesan error pada saat mengaktifkan file *php.ini*, maka anda tidak bisa menjalankan contoh dalam tutorial ini, namun tidak perlu khawatir karena baik **Register Global** dan **Register Long Array** memang tidak disarankan digunakan lagi.



Cara Mengakses Nilai Form dengan Register Global

Register Global adalah sebuah metode dalam PHP yang '*memudahkan*' pengambilan nilai form dengan cara menjadikan atribut nama dari objek form secara otomatis menjadi variabel di dalam PHP.

Misalkan kita memiliki objek form sebuah inputan dengan kode HTML sebagai berikut:

```
<input type="text" name="user_name" />
```

Maka di dalam halaman **PHP**, akan langsung tersedia sebuah variabel **$user_name** yang berisi nilai dari objek form tersebut dan siap untuk diakses, terlepas dengan cara apa form tersebut dikirim (apakah **get** atau **post**)

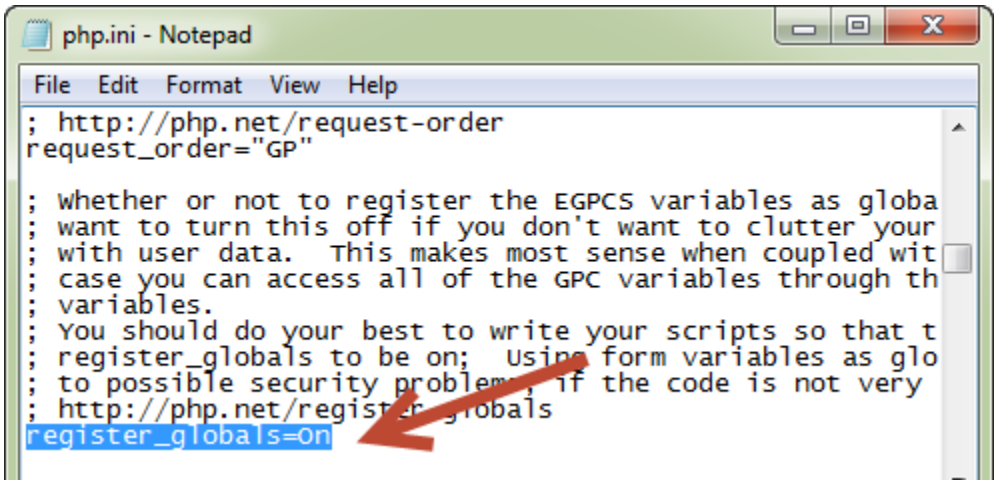
Dalam versi **PHP 5.5** bawaan **XAMPP** yang saya gunakan, **Register Global** sudah tidak didukung lagi. Namun jika anda menggunakan PHP versi lama dan ingin mencobanya, maka kita harus mengubah sebuah settingan PHP dalam file **php.ini**.

Bukalah file **php.ini**, lalu cari temukan baris setingan berikut:

```
register_globals=off
```

Kemudian aktifkan fasilitas ini menjadi:

```
register_globals=on
```



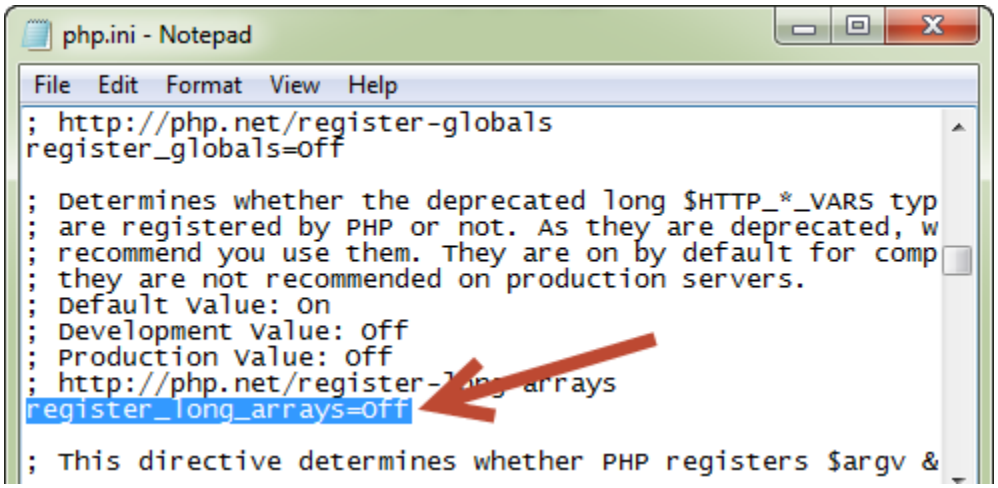
Setelah diubah, save file **php.ini**, dan matikan web server apache (dari **XAMPP Control panel**), lalu aktifkan lagi (*restart*). Hal ini diperlukan agar web server Apache dapat membaca perubahan setingan PHP yang baru saja kita lakukan.

Cara Mengakses Nilai Form dengan Register Long Array

Cara lain untuk mengambil nilai dari form adalah menggunakan **Register Long Array**. Sesuai dengan namanya, **Register Long Array** memiliki nama variabel yang lebih panjang jika dibandingkan variabel biasa (variabel **$_GET** dan **$_POST)**.

Untuk mengambil nilai form yang dikirim dengan **method=get**, kita menggunakan **$HTTP_GET_VARS**, dan untuk **method=post** menggunakan **$HTTP_POST_VARS**.

Sama seperti **Register Globals**, **Register Long Array** tidak didukung lagi untuk PHP versi terbaru. Jika anda memiliki PHP versi lama dan ingin mencobanya, lakukan perubahan settingan **php.ini** untuk pilihan **register_long_arrays=on**, seperti gambar berikut ini:



Setelah itu, restart *web server* **apache** menggunakan **XAMPP Control Panel**.

Menampilkan Keluaran

Mengenal Struktur Dasar Form HTML (atribut action dan method). Berikut adalah struktur dasar form sederhana dalam HTML:

```
<form action="proses.php" method="get">
   Nama: <input type="text" name="nama" />
   <br />
   E-Mail: <input type="text" name="email" />
   <br />
   <input type="submit" value="Proses Data" >
</form>
```

Jika anda menjalankan form HTML tersebut, akan ditampilkan form sederhana dengan 2 buah kotak inputan dan sebuah tombol "*Proses Data*" yang berfungsi untuk **submit** form. Dari struktur dasar tersebut, di dalam tag **<form>** terdapat 2 buah atribut. Yakni atribut **action** dan atribut **method**. Kita akan membahas kedua atribut ini secara lebih rinci.

Atribut pertama adalah **action**. Atribut **action** ini diisi dengan nilai berupa alamat halaman PHP dimana kita akan memproses isi form tersebut. Dalam contoh diatas, saya membuat nilai **action="proses.php"**, yang berarti saya harus menyediakan sebuah file dengan nama: **proses.php** untuk memproses form tersebut.

Isi atribut action sebenarnya adalah alamat dari halaman PHP. Karena atribut *action* pada contoh diatas ditulis **action="proses.php"**, maka file **proses.php** harus berada di dalam 1 folder dengan halaman HTML yang berisi form ini. Namun anda bisa dengan bebas mengubah alamat **proses.php** ini tergantung dimana file tersebut berada, misalnya menjadi alamat relatif seperti **action="file_php/proses.php"**, ataupun alamat absolut seperti **action="http://www.jti.polije.com/proses.php"**.

Atribut kedua yang berkaitan dengan pemrosesan form HTML adalah atribut **method**. Atribut inilah yang akan menentukan bagaimana cara form '*dikirim'* ke dalam halaman **proses.php**. Nilai dari atribut **method** hanya bisa diisi dengan1 dari 2 pilihan, yakni **get** atau **post**.

Jika seperti contoh diatas saya membuat nilai **method="get"**, maka nilai dari form akan dikirim melalui alamat *URL website*. Namun jika nilai method diubah menjadi **method="post",** maka nilai form tidak akan terlihat di dalam alamat URL. Perbedaan antara **method get** dan **post** akan kita bahas secara mendalam dalam tutorial selanjutnya. Nilai dari atribut **method** ini juga akan mempengaruhi cara kita memproses nilai dari form.

Setelah membuat tag pembuka form dengan atribut **action** dan **method**, isi form selanjutnya adalah 2 buah tag **<input type="text">** yang akan menampilkan kotak isian form. Hal yang paling penting diperhatikan adalah atribut **name** dari masing-masing tag **<input>.** Nilai dari **name** inilah yang menjadi penanda masing-masing objek form agar dapat diproses dengan PHP.

Setelah 2 buah *text input*, *objek form* terakhir adalah tombol **submit** yang apabila di klik akan mengirimkan data dari form ke halaman **proses.php** untuk

diproses. Atribut penting disini adalah atribut **type=”submit”,** yang akan otomatis mengirim isian form ketika tombol ini di klik.

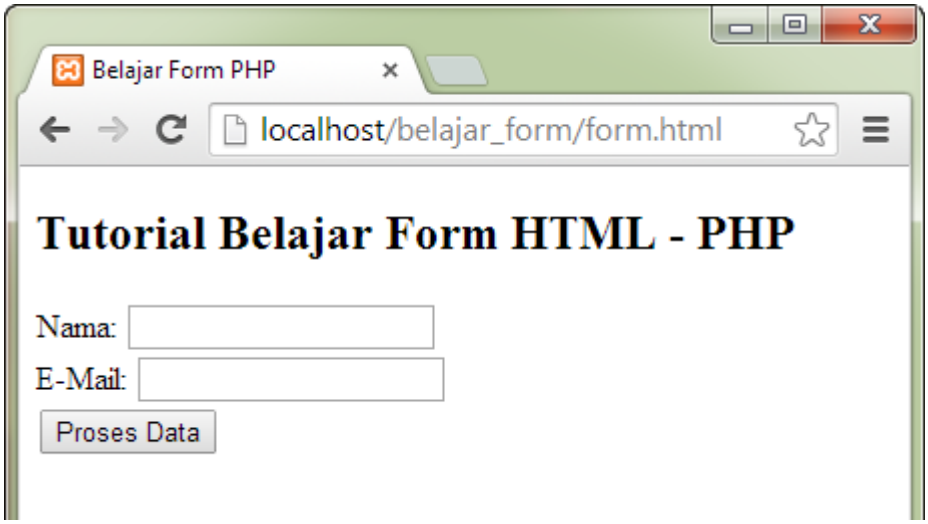
Cara Mengirimkan Nilai Form HTML ke dalam PHP

Untuk memahami Cara Mengirimkan Nilai Form HTML ke dalam PHP, kita akan langsung praktek dengan membuat 2 buah file, yakni halaman HTML yang berisi form dengan nama file **form.html**, dan halaman PHP yang akan berisi kode untuk menampilkan hasil form dengan nama file: **proses.php**.

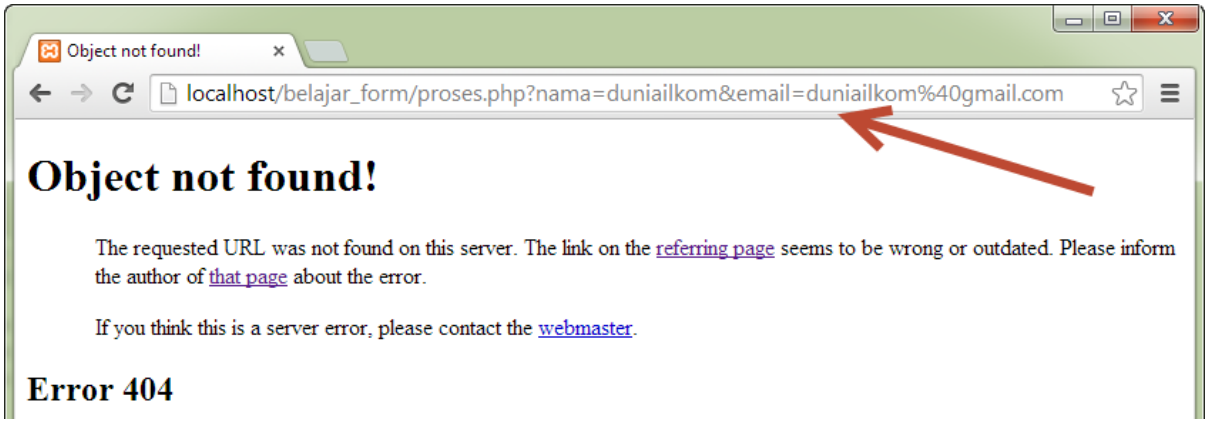
Karena kita akan mengeksekusi kode **PHP**, kedua file ini harus dijalankan dengan **XAMPP** dan berada di dalam folder **htdoc**. Untuk contoh kali ini saya akan membuat folder **belajar_form** di dalam folder **htdoc XAMPP**, sehingga untuk mengakses kedua halaman adalah dari alamat : http://localhost/belajar_form/form.html dan http://localhost/belajar_form/proses.php. Sebagai langkah pertama, kita akan membuat file **form.html** yang berisi kode HTML sebagai berikut:

```
<!DOCTYPE html>
<head>
   <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
   <title>Belajar Form PHP</title>
</head>

<body>
   <h2>Tutorial Belajar Form HTML - PHP </h2>
   <form action="proses.php" method="get">
      Nama: <input type="text" name="nama" />
      <br />
      E-Mail: <input type="text" name="email" />
      <br />
      <input type="submit" value="Proses Data" >
   </form>
</body>
</html>
```

Kode HTML diatas hanya berisi struktur kode HTML sederhana dengan 1 buah form yang berisi 2 text inputan untuk **nama** dan **e-mail**. Struktur form ini persis sama dengan form kita bahas pada bagian awal tutorial ini.



Sebelum membuat halaman **proses.php**, kita akan mencoba melakukan sedikit percobaan dengan form HTML ini. Silahkan coba input kedua kotak isian form ini dengan nilai apapun dan klik tombol **submit**. Ketika anda men-klik tombol submit, anda akan mendapati halaman error seperti berikut ini:



Halaman error tersebut memberitahu kita bahwa halaman **proses.php** tidak ditemukan (yang memang belum kita buat), namun perhatikan alamat **URL** pada *address bar* web browser, kita bisa melihat ada penambahan seperti berikut ini:

```
http://localhost/belajar_form/proses.php?nama=jti.polije&email
=jti.polije%40gmail.com
```

Perhatikan bahwa alamat URL sekarang telah berganti menjadi **proses.php**, yang kemudian diikuti dengan **?nama=jti.polije&email=jti.polije%40gmail.com**. Pesan inilah yang sebenarnya di kirim oleh halaman **form.html** sewaktu kita men-klik tombol *submit*.

Tanda ? menandakan awal dari data form, dan kemudian diikuti dengan nama dari objek form dan nilainya, dalam contoh ini, nilai yang akan dikirim adalah **nama=jti.polije.** Karakter "&" digunakan sebagai tanda pemisah nilai objek form yang

1 dengan yang lain, kemudian diikuti dengan nilai kedua **email=jti.polije%40gmail.com**. Tanda **%40** dalam kode ini adalah kode karakter **HTML** untuk karakter @ yang biasa digunakan di dalam email.

Jika anda menambahkan objek form ketiga, maka alamat URL akan semakin panjang. Kita bisa melihat data yang dikirim karena pada saat pembuatan form, saya menggunakan **method=get**. Namun jika anda merubah form HTML kita dengan menggunakan **method=post,** maka anda tidak akan melihat karakter-karakter ini di dalam URL.

Selanjutnya kita akan mencoba menampilkan nilai ini dengan PHP pada halaman **proses.php**.

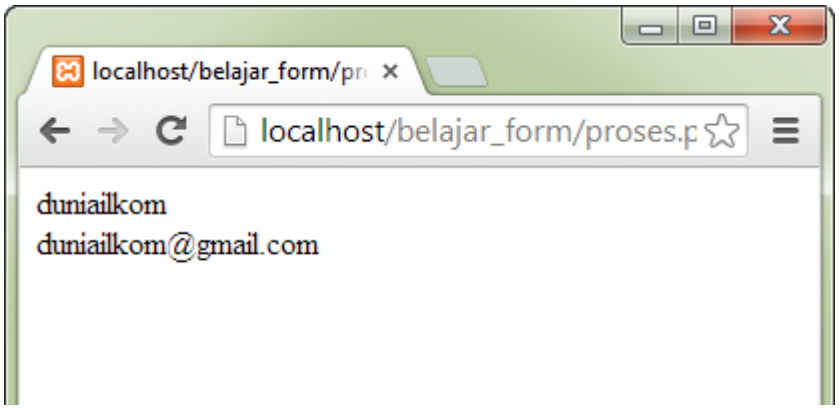
Cara Menampilkan nilai Form HTML dengan PHP ($_GET dan $_POST)

Setelah membuat halaman **form.html** yang berisi form HTML, kita akan membuat halaman **proses.php** yang berisi kode PHP untuk menangani nilai dari form ini.

Silahkan buat file **proses.php** dengan kode program sebagai berikut, dan savelah di dalam folder yang sama dengan **form.html** berada:

```
<?php
   echo $_GET['nama'];
   echo "<br />";
   echo $_GET['email'];
?>
```

Sebelum kita membahas kode program PHP tersebut, silahkan buka kembali halaman **form.html**, isi kotak input *nama* dan *email*, lalu klik tombol *Proses Data*. Apabila tidak ada error, maka akan tampil hasil berikut ini:



Tampilan diatas adalah hasil dari 3 baris kode program PHP yang kita buat di dalam halaman **proses.php.**

Untuk mengambil nilai form HTML, PHP menyediakan 2 buah variabel global yaitu variabel **$_GET** dan **$_POST**. Kita menggunakan variabel **$_GET** jika pada saat

pembuatan form menggunakan atribut **method=get**, dan menggunakan variabel **$_POST** jika form dibuat dengan **method=post**.

Kedua variabel ini sebenarnya adalah array, sehingga cara mengakses nilai dari form adalah dengan cara: **$_GET[‘nama_objek_form’].**

‘nama_objek_form’ adalah nilai dari atribut **name** di dalam **form**. Jika kita memiliki tag dengan kode HTML **<input type=”text” name=”nama” />**, maka untuk mengakses nilainya adalah dengan **$_GET[‘nama’]**, dan untuk tag **<input type=”text” name=”email” />** diakses dengan nilai **$_GET[’email’]**.

Sebagai latihan, silahkan anda mengganti atribut *method* dalam file **form.html** menjadi:

```
<form action="proses.php" method="post">
```

Lalu ubah juga file **proses.php** menjadi:

```
<?php
   echo $_POST['nama'];
   echo "<br />";
   echo $_POST['email'];
?>
```

Dan PHP akan menampilkan hasil yang sama, namun kali ini form dikirim menggunakan **method=post**.

Variabel dan Tipe Data

Pengertian Variabel dalam PHP

Dalam pemograman, **variabel** adalah suatu lokasi penyimpanan (di dalam memori komputer) yang berisikan nilai atau informasi yang nilainya tidak diketahui maupun telah diketahui (wikipedia).

Dalam defenisi bebasnya, variabel adalah *kode program yang digunakan untuk menampung nilai tertentu*. Nilai yang disimpan di dalam variabel selanjutnya dapat dipindahkan ke dalam database, atau ditampilkan kembali ke pengguna.

Nilai dari **variabel** dapat di isi dengan informasi yang diinginkan dan dapat dirubah nilainya pada saat kode program sedang berjalan. Sebuah **variabel** memiliki *nama* yang digunakan untuk mengakses nilai dari **variabel** itu. Jika anda memiliki pengetahuan dasar tentang bahasa pemograman, tentunya tidak asing dengan istilah **variabel**.

Sama seperti **variabel** dalam bahasa pemograman lainnya, variabel dalam PHP digunakan untuk menampung nilai inputan dari user, atau nilai yang kita definisikan sendiri. Namun PHP memiliki beberapa aturan tentang cara penggunaan dan penulisan **variabel.**

Aturan Penulisan Variabel dalam PHP

Penulisan variabel harus diawali dengan tanda $

Variabel di dalam PHP harus diawali dengan **dollar sign** atau **tanda dollar** ($).

Setelah tanda **$**, sebuah **variabel** PHP harus diikuti dengan karakter pertama berupa huruf atau *underscore* (_), kemudian untuk karakter kedua dan seterusny bisa menggunakan huruf, angka atau *underscore* (_). Dengan aturan tersebut, variabel di dalam PHP *tidak bisa* diawali dengan angka.

Minimal panjang variabel adalah 1 karakter setelah tanda $. Berikut adalah contoh penulisan variabel yang benar dalam PHP:

```
<?php
   $i;
   $nama;
   $Umur;
   $_lokasi_memori;
   $ANGKA_MAKSIMUM;
      ?>
```

Dan berikut adalah contoh penulisan variabel yang salah:

```
<?php
   $4ever; //variabel tidak boleh diawali dengan angka
   $_salah satu; //varibel tidak boleh mengandung spasi
   $nama*^; //variabel tidak boleh mengandung karakter khusus:
* dan ^
?>
```

Variabel dalam PHP bersifat case sensitif

PHP membedakan variabel yang ditulis dengan huruf besar dan kecil (bersifat **case sensitif**), sehingga **$belajar** tidak sama dengan **$Belajar** dan **$BELAJAR**, ketiganya akan dianggap sebagai variabel yang berbeda.

Untuk menghindari kesalahan program yang dikarenakan salah merujuk **variabel**, disarankan menggunakan huruf kecil untuk seluruh nama **variabel**.

```
<?php
   $andi="Andi";
   echo $Andi; // Notice: Undefined variable: Andi
?>
```

Dalam contoh diatas, PHP mengeluarkan error karena tidak menemukan variabel *$Andi*.

Cara Memberikan Nilai kepada Variabel

Sama seperti sebagian besar bahasa pemograman lainnya, untuk memberikan nilai kepada sebuah **variabel**, PHP menggunakan tanda **sama dengan** (=). Operator *'sama dengan'* ini dikenal dengan istilah ***Assignment Operators***.

Perintah pemberian nilai kepada sebuah variabel disebut dengan ***assignment.*** Jika *variabel* tersebut belum pernah digunakan, dan langsung diberikan nilai awal, maka disebut juga dengan proses ***inisialisasi***.

Berikut contoh cara memberikan nilai awal (*inisialisasi*) kepada variabel:

```
<?php
   $nama = "andi";
   $umur = “b17”;
   $pesan = "Saya sedang belajar PHP di jti.polije.com";
?>
```

Variabel dalam PHP tidak memerlukan deklarasi terlebih dahulu

Jika anda pernah mempelajari bahasa pemograman desktop seperti **Pascal**, **C**, **C++**, dan **Visual Basic**, di dalam bahasa pemograman tersebut, sebuah variabel harus dideklarasikan terlebih dahulu sebelum digunakan.

Namun di dalam PHP, **variabel** tidak perlu deklarasikan terlebih dahulu. Anda bebas membuat variabel baru di tengah-tengah kode program, dan langsung menggunakannya tanpa di deklarasikan terlebih dahulu.

```
<?php
   $andi="Andi";
   echo $andi;
?>
```

PHP memiliki keyword ***var*** untuk mendefenisikan variable, keyword ini digunakan untuk PHP versi 4 kebawah. PHP versi 5 tidak membutuhkan keyword ini, dan penggunaannya akan menghasilkan error, seperti contoh berikut ini:

```
<?php
   // kode program dibawah ini akan menghasilkan error
   // Parse error: syntax error, unexpected 'var' (T_VAR)
   var $andi="Andi";
   echo $andi;
?>
```

Variabel dalam PHP tidak bertipe

Dalam kelompok bahasa pemograman, PHP termasuk **Loosely Type Language**, yaitu jenis bahasa pemograman yang variabelnya tidak terikat pada sebuah tipe tertentu.

Hal ini berbeda jika dibandingkan dengan bahasa pemograman desktop seperti **Pascal** atau **C**, dimana jika anda membuat sebuah variabel bertipe **integer**, maka

variabel itu hanya bisa menampung nilai angka, dan anda tidak akan bisa mengisinya dengan huruf.

Di dalam PHP, setiap variabel bebas diisi dengan nilai apa saja, seperti contoh berikut:

```
<?php
   $a = 17; // nilai variabel a berisi angka (integer)
   $a = "aku"; // nilai variabel a diubah menjadi kata
(string)
   $a = 17.42; // nilai variabel a diubah menjadi desimal
(float)
?>
```

Variabel Sistem PHP (Predefined Variables)

Predefined Variables atau terjemahan bebasnya **Variabel Sistem PHP**, adalah beberapa variabel yang telah didefenisikan secara sistem oleh PHP, dan kita sebaiknya tidak membuat variabel dengan nama yang sama.

Beberapa contoh **Predefined Variables** dalam PHP adalah:

```
$GLOBALS , $_SERVER , $_GET , $_POST , $_FILES , $_COOKIE ,
$_SESSION , $_REQUEST , $_ENV, $php_errormsg,
$HTTP_RAW_POST_DATA, $http_response_header, $argc, $argv,
$this.
```

Cara Menampilkan Nilai Variabel

Untuk menampilkan nilai atau isi dari variabel, kita tinggal menampilkannya dengan perintah **echo** atau **print**, seperti berikut ini:

```
<?php
   $a='Saya Sedang belajar PHP';
   $b=5;

   print $a;
   echo $b;
?>
```

Hasil yang didapat adalah:

```
Saya Sedang belajar PHP5
```

Perhatikan bahwa kedua nilai variabel ditampilkan tanpa spasi diantaranya. Hal ini terjadi karena di dalam program PHP saya tidak menyisipkan spasi untuk pemisah diantara kedua variabel.

Walaupun kita akan membahasnya lebih lengkap pada tutorial mengenai string, kita juga bisa menampilkan variabel langsung di dalam string jika string tersebut berada di antara tanda kutip dua (“):

```
<?php
   $a=5;
   $b="Sedang belajar PHP $a";

   echo $b;
   // hasil: Saya Sedang belajar PHP 5
?>
```

Variabel dapat dikatakan sebagai inti dari sebuah bahasa pemograman. Karena melalui *variabel* inilah kita memanipulasi data inputan agar menjadi nilai yang diinginkan. Selain **variabel**, PHP juga menyediakan sebuah solusi lain untuk menampung nilai data dengan **konstanta** (**constant**).

Operator

Pengertian Operator Aritmatika dalam PHP

Operator Aritmatika adalah operator matematis yang terdiri dari operator **penambahan, pengurangan, perkalian, pembagian, modulus, plus,** dan **minus.**

Jenis Operator Aritmatika dalam PHP

Didalam PHP terdapat 7 jenis operator aritmatika, berikut ke tujuh operator tersebut:

Contoh	Nama Operator
+$a	Positif
-$a	Negatif
$a + $b	Penambahan
$a - $b	Pengurangan
$a * $b	Perkalian
$a / $b	Div/Pembagian
$a % $b	Mod/Sisa hasil bagi

Kebanyakan operator aritmatika dalam PHP bertipe **binary** yakni membutuhkan 2 **operand**, kecuali operator minus (-) dan plus (+) yang merupakan operator tipe **unary** (hanya membutuhkan 1 operand).

Dari ke 7 operator aritmatika dalam PHP tersebut, operator **modulus** ($a % $b) mungkin terdengar baru. Operator ini menghasilkan sisa hasil bagi dari hasil

pembagian. Misalkan 10 % 3, hasilnya adalah 1. Biasanya operator **modulus** ini digunakan bersama-sama dengan operator **pembagian** (/).

Cara Penggunaan Operator Aritmatika di dalam PHP

Penggunakan **operator aritmatika** di dalam PHP relatif mudah, karena kita telah terbiasa dengan operator ini.

Berikut adalah contoh kode program, cara penggunaan operator aritmatika dalam PHP:

```
<?php
$hasil1= -3;
$hasil2=3+5;
$hasil3=8-4.5;
$hasil4=2*5;
$hasil5=3+8/5-3;
$hasil6=10 % 4;

echo "\$hasil1:"; var_dump($hasil1); // $hasil1:int(-3)
echo "<br \>";
echo "\$hasil2:"; var_dump($hasil2); // $hasil2:int(8)
echo "<br \>";
echo "\$hasil3:"; var_dump($hasil3); // $hasil3:float(3.5)
echo "<br \>";
echo "\$hasil4:"; var_dump($hasil4); // $hasil4:int(10)
echo "<br \>";
echo "\$hasil5:"; var_dump($hasil5); // $hasil5:float(1.6)
echo "<br \>";
echo "\$hasil6:"; var_dump($hasil6); // $hasil6:int(2)
?>
```

Pada kode program diatas, saya menggunakan fungsi **var_dump()** untuk menampilkan hasil perhitungan, sehingga kita bisa melihat tipe data dari masing-masing variabel.

Dari hasil **var_dump(),** terlihat bahwa variabel **$hasil3** dan **$hasil5** bertipe **float**. Hal ini dikarenakan perhitungan aritmatika pada baris ke-4 dan ke-6 menghasilkan angka *desimal*, sehingga secara otomatis variabel tersebut tidak dapat ditampung sebagai **integer**, melainkan harus **float**.

Namun jika hasil operasi matematis tersebut menghasilkan bilangan bulat, PHP akan menyimpannya sebagai tipe data int (**integer**), seperti variabel **$hasil1, $hasil2, $hasil4** dan **$hasil6**.

Pada perhitungan baris ke-6 yaitu persamaan **$hasil5=3+8/5-3**, hasilnya adalah **1.6**. Hal ini karena operator pembagian memiliki prioritas lebih tinggi daripada operator tambah dan kurang. Operasi **3+8/5-3** dikerjakan oleh PHP sebagai **(3+(8/5))-3**. Namun untuk hal ini, disarankan menggunakan tanda kurung secara tertulis agar memudahkan dalam membaca alur program, dari pada bergantung kepada aturan prioritas operator PHP.

Namun jika tidak ditegaskan dengan menggunakan tanda kurung, **urutan prioritas** operator matematis dalam PHP mengikuti aturan tabel yang kita bahas

sebelumnya. Itulah operator-operator dasar aritmatika dalam PHP. Untuk keperluan yang lebih spesifik seperti pemangkatan, logaritma, exponensial dan sinus, PHP menyediakan fungsi matematika (**Mathematical Functions**).

PHP 5 Math Functions

Function	Description
abs()	Returns the absolute (positive) value of a number
acos()	Returns the arc cosine of a number
acosh()	Returns the inverse hyperbolic cosine of a number
asin()	Returns the arc sine of a number
asinh()	Returns the inverse hyperbolic sine of a number
atan()	Returns the arc tangent of a number in radians
atan2()	Returns the arc tangent of two variables x and y
atanh()	Returns the inverse hyperbolic tangent of a number
base_convert()	Converts a number from one number base to another
bindec()	Converts a binary number to a decimal number
ceil()	Rounds a number up to the nearest integer
cos()	Returns the cosine of a number
cosh()	Returns the hyperbolic cosine of a number
decbin()	Converts a decimal number to a binary number
dechex()	Converts a decimal number to a hexadecimal number
decoct()	Converts a decimal number to an octal number
deg2rad()	Converts a degree value to a radian value
exp()	Calculates the exponent of e
expm1()	Returns exp(x) - 1
floor()	Rounds a number down to the nearest integer
fmod()	Returns the remainder of x/y
getrandmax()	Returns the largest possible value returned by rand()
hexdec()	Converts a hexadecimal number to a decimal number
hypot()	Calculates the hypotenuse of a right-angle triangle
is_finite()	Checks whether a value is finite or not
is_infinite()	Checks whether a value is infinite or not
is_nan()	Checks whether a value is 'not-a-number'
lcg_value()	Returns a pseudo random number in a range between 0 and 1
log()	Returns the natural logarithm of a number
log10()	Returns the base-10 logarithm of a number

log1p()	Returns log(1+number)
max()	Returns the highest value in an array, or the highest value of several specified values
min()	Returns the lowest value in an array, or the lowest value of several specified values
mt_getrandmax()	Returns the largest possible value returned by mt_rand()
mt_rand()	Generates a random integer using Mersenne Twister algorithm
mt_srand()	Seeds the Mersenne Twister random number generator
octdec()	Converts an octal number to a decimal number
pi()	Returns the value of PI
pow()	Returns x raised to the power of y
rad2deg()	Converts a radian value to a degree value
rand()	Generates a random integer
round()	Rounds a floating-point number
sin()	Returns the sine of a number
sinh()	Returns the hyperbolic sine of a number
sqrt()	Returns the square root of a number
srand()	Seeds the random number generator
tan()	Returns the tangent of a number
tanh()	Returns the hyperbolic tangent of a number

PHP 5 Predefined Math Constants

Constant	Value	Description	PHP Version
INF	INF	The infinite	PHP 4
M_E	2.7182818284590452354	Returns e	PHP 4
M_EULER	0.57721566490153286061	Returns Euler constant	PHP 4
M_LNPI	1.14472988584940017414	Returns the natural logarithm of PI: log_e(pi)	PHP 5.2
M_LN2	0.69314718055994530942	Returns the natural logarithm of 2: log_e 2	PHP 4
M_LN10	2.30258509299404568402	Returns the natural logarithm of 10: log_e 10	PHP 4
M_LOG2E	1.4426950408889634074	Returns the base-2 logarithm of E: log_2 e	PHP 4

M_LOG10E	0.43429448190325182765	Returns the base-10 logarithm of E: log_10 e	PHP 4
M_PI	3.14159265358979323846	Returns Pi	PHP 4
M_PI_2	1.57079632679489661923	Returns Pi/2	PHP 4
M_PI_4	0.78539816339744830962	Returns Pi/4	PHP 4
M_1_PI	0.31830988618379067154	Returns 1/Pi	PHP 4
M_2_PI	0.63661977236758134308	Returns 2/Pi	PHP 4
M_SQRTPI	1.77245385090551602729	Returns the square root of PI: sqrt(pi)	PHP 5.2
M_2_SQRTPI	1.12837916709551257390	Returns 2/square root of PI: 2/sqrt(pi)	PHP 4
M_SQRT1_2	0.70710678118654752440	Returns the square root of 1/2: 1/sqrt(2)	PHP 4
M_SQRT2	1.41421356237309504880	Returns the square root of 2: sqrt(2)	PHP 4
M_SQRT3	1.73205080756887729352	Returns the square root of 3: sqrt(3)	PHP 5.2
NAN	NAN	Not A Number	PHP 4
PHP_ROUND_HALF_UP	1	Round halves up	PHP 5.3
PHP_ROUND_HALF_DOWN	2	Round halves down	PHP 5.3
PHP_ROUND_HALF_EVEN	3	Round halves to even numbers	PHP 5.3
PHP_ROUND_HALF_ODD	4	Round halves to odd numbers	PHP 5.3

3. Alat dan Bahan

A. Hosting dan Domain (disediakan Politeknik Negeri Jember)
B. Kertas A4 (disediakan Politeknik Negeri Jember)
C. Kertas F4 (disediakan Politeknik Negeri Jember)
D. Folio Bergaris (disediakan Politeknik Negeri Jember)
E. BoardMarker (disediakan Politeknik Negeri Jember)
F. Komputer (disediakan Politeknik Negeri Jember dengan jumlah sesuai kapasitas lab)
G. XAMPP
H. Sublime Text 3
I. GitHub Desktop
J. Bootstrap

4. Kegiatan Praktik

A. Mahasiswa berkumpul dengan anggota kelompoknya
B. Masing-masing anggota me-refresh GitHub Desktop
C. Mengikuti langkah-langkah pada link di bawah
   - https://www.malasngoding.com/pengertian-dan-cara-menggunakan-bootstrap/
   - https://www.malasngoding.com/membuat-navigation-bar-bootstrap/
   - https://www.malasngoding.com/membuat-form-dengan-bootstrap/
   - https://www.malasngoding.com/membuat-login-dengan-php-dan-mysqli-menggunakan-md5/
D. Anggota kelompok melakukan Pull terhadap Repo yang telah dibuat ketua

5. Refrensi *Website*

[1] https://www.w3schools.com/
[2] https://www.malasngoding.com/
[3] https://www.codepolitan.com/
[4] http://www.duniailkom.com/

PRAKTIK 4: Menerapkan Perintah Eksekusi Bahasa Pemrograman (PHP) dan MySQL

Matakuliah : Workshop SI berbasis WEB
Minggu Ke : 6 dan 7
Waktu : 2 x 100 menit
Tema : Menerapkan Perintah Eksekusi Bahasa Pemrograman (PHP) dan MySQL

1. Kompetensi Dasar

A. Mahasiswa mampu mengesekusi *Source Code* PHP
B. Mahasiswa mampu mengidentifikasi hasil *Source Code* PHP

2. Alat dan Bahan

A. SSD Internal (backup) dan RAM (backup) (disediakan Politeknik Negeri Jember)
B. Hosting dan Domain (disediakan Politeknik Negeri Jember)
C. Kertas A4 (disediakan Politeknik Negeri Jember)
D. Kertas F4 (disediakan Politeknik Negeri Jember)
E. Folio Bergaris (disediakan Politeknik Negeri Jember)
F. BoardMarker (disediakan Politeknik Negeri Jember)
G. Komputer (disediakan Politeknik Negeri Jember dengan jumlah sesuai kapasitas lab)
H. XAMPP
I. Sublime Text 3
J. GitHub Desktop
K. Bootstrap

3. Kegiatan Praktik

A. Mahasiswa berkumpul dengan anggota kelompoknya
B. Masing-masing anggota me-refresh GitHub Desktop
C. Mengikuti langkah-langkah pada link di bawah
    1. Setiap anggota membuat folder baru pada htdocs, dengan nama NIM_P4
    2. https://www.malasngoding.com/belajar-php-dasar-pengenalan-dan-kegunaan-php/
    3. https://www.malasngoding.com/belajar-php-dasar-untuk-pemula/
    4. https://www.malasngoding.com/belajar-php-kondisi-if-else-pada-php/
    5. https://www.malasngoding.com/belajar-php-tipe-data-pada-php/
D. Anggota kelompok melakukan Pull dan Push Repo pada folder masing-masing

4. Refrensi *Website*

[1] https://www.w3schools.com/
[2] https://www.malasngoding.com/

[3] https://www.codepolitan.com/
[4] http://www.duniailkom.com/

PRAKTIK 5: Menyusun Fungsi, File atau Sumber Daya Pemrograman yang Lain dalam Organisasi yang Rapi (HTML, CSS, JS, PHP, dan Resource) dan Menulis Kode dengan Prinsip sesuai Guidelines dan Best Practices (HTML dan PHP)

Matakuliah : Workshop SI berbasis WEB
Minggu Ke : 8 dan 9
Waktu : 2 x 100 menit
Tema : Menyusun fungsi, file atau sumber daya pemrograman yang lain dalam organisasi yang rapi (HTML, CSS, JS, PHP, dan Resource) dan Menulis kode dengan prinsip sesuai guidelines dan best practices (HTML dan PHP)

1. **Kompetensi Dasar**
   A. Mahasiswa mampu menyusun fungsi, file atau sumber daya pemrograman yang lain dalam organisasi yang rapi (HTML, CSS, JS, PHP, dan Resource)
   B. Mahasiswa mampu menulis kode dengan prinsip sesuai guidelines dan best practices (HTML dan PHP)

2. **Dasar Teori**

   A. Menyusun Fungsi, File atau Sumber Daya Pemrograman yang Lain dalam Organisasi yang Rapi (HTML, CSS, JS, PHP, dan Resource)

   REST (REpresentational State Transfer) merupakan standar arsitektur komunikasi berbasis web yang sering diterapkan dalam pengembangan layanan berbasis web. Umumnya menggunakan HTTP (Hypertext Transfer Protocol) sebagai protocol untuk komunikasi data. REST pertama kali diperkenalkan oleh Roy Fielding pada tahun 2000.

   B. Menulis Kode dengan Prinsip sesuai Guidelines dan Best Practices (HTML dan PHP)

   REST (REpresentational State Transfer) merupakan standar arsitektur komunikasi berbasis web yang sering diterapkan dalam pengembangan layanan berbasis web. Umumnya menggunakan HTTP (Hypertext Transfer Protocol) sebagai protocol untuk komunikasi data. REST pertama kali diperkenalkan oleh Roy Fielding pada tahun 2000.

3. Alat dan Bahan

A. SSD (backup) dan RAM (backup) (disediakan Politeknik Negeri Jember)
B. Hosting dan Domain (disediakan Politeknik Negeri Jember)
C. Kertas A4 (disediakan Politeknik Negeri Jember)
D. Kertas F4 (disediakan Politeknik Negeri Jember)
E. Folio Bergaris (disediakan Politeknik Negeri Jember)
F. BoardMarker (disediakan Politeknik Negeri Jember)
G. Komputer (disediakan Politeknik Negeri Jember dengan jumlah sesuai kapasitas lab)
H. Bolpoin (disediakan Politeknik Negeri Jember)
I. Spidol (disediakan Politeknik Negeri Jember)
J. Pensil (disediakan Politeknik Negeri Jember)
K. Penghapus (disediakan Politeknik Negeri Jember)
L. XAMPP
M. Sublime Text 3
N. GitHub Desktop

4. Kegiatan Praktik

A. Mahasiswa berkumpul dengan anggota kelompoknya
B. Masing-masing anggota me-refresh GitHub Desktop
C. Mengikuti langkah-langkah pada link di bawah
– https://www.malasngoding.com/belajar-php-mengenal-array-pada-php/
– https://www.malasngoding.com/belajar-php-perulangan-while-pada-php/
– https://www.malasngoding.com/belajar-php-operator-aritmatika-di-php/
– https://www.malasngoding.com/belajar-php-penanganan-form-pada-php/
– https://www.malasngoding.com/belajar-php-mengenal-include-dan-require-pada-php/
– https://www.malasngoding.com/cara-membuat-koneksi-php-dengan-database-mysql/
– https://www.malasngoding.com/membuat-crud-dengan-php-dan-mysql-menampilkan-data-dari-database/
D. Anggota kelompok melakukan Pull dan Push Repo pada folder masing-masing

5. Refrensi *Website*

[1] https://www.w3schools.com/
[2] https://www.malasngoding.com/
[3] https://www.codepolitan.com/
[4] http://www.duniailkom.com/

PRAKTIK 6: Mengimplementasikan Pemrograman Terstruktur (menggunakan tipe data dan control program, membuat program sederhana dan membuat program menggunakan prosedur dan fungsi) dan Membuat Data Entri Website

Matakuliah : Workshop SI berbasis WEB
Minggu Ke : 10 dan 11
Waktu : 2 x 100 menit
Tema : Mengimplementasikan Pemrograman Terstruktur (menggunakan tipe data dan control program, membuat program sederhana dan membuat program menggunakan prosedur dan fungsi) dan Membuat Data Entri Website

1. **Kompetensi Dasar**
   A. Mahasiswa Mampu Mengimplementasikan Pemrograman Terstruktur (menggunakan tipe data dan control program, membuat program sederhana dan membuat program menggunakan prosedur dan fungsi)
   B. Membuat Data Entri Website

2. **Alat dan Bahan**
   A. SSD (backup) dan RAM (backup) (disediakan Politeknik Negeri Jember)
   B. Hosting dan Domain (disediakan Politeknik Negeri Jember)
   C. Kertas A4 (disediakan Politeknik Negeri Jember)
   D. Kertas F4 (disediakan Politeknik Negeri Jember)
   E. Folio Bergaris (disediakan Politeknik Negeri Jember)
   F. BoardMarker (disediakan Politeknik Negeri Jember)
   G. Komputer (disediakan Politeknik Negeri Jember dengan jumlah sesuai kapasitas lab)
   H. XAMPP
   I. Sublime Text 3
   J. GitHub Desktop

3. **Kegiatan Praktik**
   A. Mahasiswa berkumpul dengan anggota kelompoknya
   B. Masing-masing anggota me-refresh GitHub Desktop
   C. Mengikuti langkah-langkah pada link di bawah
   – https://www.malasngoding.com/cara-membuat-form-validasi-dengan-php/
   – https://www.malasngoding.com/membuat-crud-dengan-php-dan-mysql-hapus-data/
   – https://www.malasngoding.com/membuat-crud-dengan-php-dan-mysql-edit-data/
   – https://www.malasngoding.com/membuat-crud-dengan-php-dan-mysql-input-data/

–

D. Anggota kelompok melakukan Pull dan Push Repo pada folder masing-masing

4. Refrensi *Website*

[1] https://www.w3schools.com/
[2] https://www.malasngoding.com/
[3] https://www.codepolitan.com/
[4] http://www.duniailkom.com/

PRAKTIK 7: Mengimplementasikan Pemrograman Terstruktur (membuat program menggunakan array, membuat program untuk akses file dan database dan mengkompilasi Program) dan Menggunakan Library atau Komponen Pre- Existing

Matakuliah : Workshop SI berbasis WEB
Minggu Ke : 12 dan 13
Waktu : 2 x 100 menit
Tema : Mengimplementasikan Pemrograman Terstruktur (membuat program menggunakan array, membuat program untuk akses file dan database dan mengkompilasi Program) dan Menggunakan Library atau Komponen Pre- Existing

1. **Kompetensi Dasar**
   A. Mahasiswa Mampu Mengimplementasikan Pemrograman Terstruktur (membuat program mengguna-kan array, membuat program untuk akses file dan database dan mengkompilasi Program) dan Menggunakan Library atau Komponen Pre- Existing
   B. Mahasiswa Mampu Menggunakan Library atau Komponen Pre- Existing

2. **Alat dan Bahan**
   A. SSD (backup) dan RAM (backup) (disediakan Politeknik Negeri Jember)
   B. Hosting dan Domain (disediakan Politeknik Negeri Jember)
   C. Kertas A4 (disediakan Politeknik Negeri Jember)
   D. Kertas F4 (disediakan Politeknik Negeri Jember)
   E. Folio Bergaris (disediakan Politeknik Negeri Jember)
   F. BoardMarker (disediakan Politeknik Negeri Jember)
   G. Komputer (disediakan Politeknik Negeri Jember dengan jumlah sesuai kapasitas lab)
   H. XAMPP
   I. Sublime Text 3
   J. GitHub Desktop

3. **Kegiatan Praktik**
   A. Mahasiswa berkumpul dengan anggota kelompoknya
   B. Masing-masing anggota me-refresh GitHub Desktop
   C. Masing-masing mahasiswa mengikuti langkah sesuai link berikut:
   – http://www.kursuswebsite.org/cara-install-bootstrap-4-dengan-composer/
   – https://www.warungbelajar.com/cara-menggunakan-composer-untuk-project-php.html

- https://www.warungbelajar.com/tutorial-php-part-56-debugging-php-menggunakan-xdebug-dan-visual-studio-code.html
- https://www.warungbelajar.com/tutorial-php-part-55-membuat-grafik-dari-database-mysql-dan-php-dengan-chart-js.html
- https://www.warungbelajar.com/import-data-dari-excel-ke-mysql-dengan-php-menggunakan-phpspreadsheet.html
- https://www.warungbelajar.com/cara-membuat-report-excel-di-php-menggunakan-phpspreadsheet.html

D. Anggota kelompok melakukan Pull dan Push Repo pada folder masing-masing

4. Refrensi *Website*

[1] https://www.w3schools.com/
[2] https://www.malasngoding.com/
[3] https://www.codepolitan.com/
[4] http://www.duniailkom.com/

PRAKTIK 8: Implementasi Project

Matakuliah : Workshop SI berbasis WEB
Minggu Ke : 14 dan 15
Waktu : 2 x 100 menit
Tema : Implementasi Project

1. **Kompetensi Dasar**
   A. Mahasiswa mampu memahami konsep kolaborasi
   B. Mahasiswa mampu membuat proyek yang merupakan pengaplikasian hasil dari perkuliahan:
      - TIF3603 Manajemen Basis Data
      - TIF3604 Interaksi Manusia dan Komputer
      - TIF3605 Pemodelan Sistem Informasi
      - TIF3606 Workshop Pengembangan Perangkat Lunak
      - TIF3607 Workshop Sistem Informasi Berbasis Web

   C. Mahasiswa mampu mengevaluasi kesesuaian pekerjaan dengan perencanaan yang telah dibuat

2. **Alat dan Bahan**
   A. SSD (backup) dan RAM (backup) (disediakan Politeknik Negeri Jember)
   B. Hosting dan Domain (disediakan Politeknik Negeri Jember)
   C. Kertas A4 (disediakan Politeknik Negeri Jember)
   D. Kertas F4 (disediakan Politeknik Negeri Jember)
   E. Folio Bergaris (disediakan Politeknik Negeri Jember)
   F. BoardMarker (disediakan Politeknik Negeri Jember)
   G. Komputer (disediakan Politeknik Negeri Jember dengan jumlah sesuai kapasitas lab)
   H. XAMPP
   I. Sublime Text 3
   J. GitHub Desktop

3. **Kegiatan Praktik**
   A. Mahasiswa berkumpul dengan anggota kelompoknya
   B. Masing-masing anggota me-refresh GitHub Desktop
   C. Anggota kelompok masing-masing mengerjakan bagiannya dalam proyek
   D. Anggota kelompok melakukan Pull dan Push terhadap Repo yang telah dibuat ketua

4. **Refrensi *Website***

[1] https://www.w3schools.com/
[2] https://www.malasngoding.com/
[3] https://www.codepolitan.com/

[4] http://www.duniailkom.com/

PRAKTIK 9: Implementasi dan Presentasi Project

Matakuliah : Workshop SI berbasis WEB
Minggu Ke : 16
Waktu : 2 x 100 menit
Tema : Prensentasi Project

1. **Kompetensi Dasar**
   A. Mahasiswa mampu memahami konsep kolaborasi
   B. Mahasiswa mampu mengevaluasi kesesuaian pekerjaan dengan perencanaan yang telah dibuat
   C. Mahasiswa mampu menjelaskan kontribusinya pada proyek yang dikerjakan

2. **Alat dan Bahan**
   A. SSD (backup) dan RAM (backup) (disediakan Politeknik Negeri Jember)
   B. Hosting dan Domain (disediakan Politeknik Negeri Jember)
   C. Kertas A4 (disediakan Politeknik Negeri Jember)
   D. Kertas F4 (disediakan Politeknik Negeri Jember)
   E. Folio Bergaris (disediakan Politeknik Negeri Jember)
   F. BoardMarker (disediakan Politeknik Negeri Jember)
   G. Komputer (disediakan Politeknik Negeri Jember dengan jumlah sesuai kapasitas lab)
   H. XAMPP
   I. Sublime Text 3
   J. GitHub Desktop

3. **Kegiatan Praktik**
   A. Mahasiswa berkumpul dengan anggota kelompoknya
   B. Masing-masing anggota me-refresh GitHub Desktop
   C. Mahasiswa akan dipanggil bisa berkelompok/satu per satu untuk menjelaskan proyek yang telah dikerjakan
   D. Komponen penilaian yang digunakan:
      a. Presensi Perkuliahan/Attitude
      b. Jumlah Commit Masing-Masing Contributor
      c. Melihat Kontrobusi dari Masing-Masing Anggota Kelompok
      d. Kesesuaian Perencaan dengan Produk Akhir
         - TIF4503 Kewirausahaan
         - TIF4503 Administrasi Basis Data
         - TIF4504 Perawatan Perangkat Lunak
         - TIF4506 Sistem Informasi Jasa Berbasis Layanan
         - TIF4507 Workshop Aplikasi Mobile
      e. Penilaian Demo Aplikasi di Akhir Semester

f. Penguasaan Masing-Masing Anggota

E. Team penilai dari Worshop WEB FrameWork: Team Dosen Pengampu

F. Waktu dan Tempat Presentasi/Demo Proyek: Menunggu pengumuman di akhir semester

4. Refrensi *Website*

[1] https://www.w3schools.com/

[2] https://www.malasngoding.com/

[3] https://www.codepolitan.com/

[4] http://www.duniailkom.com/

DAFTAR PUSTAKA

[1] https://www.codeigniter.com/
[2] https://codeigniter-id.github.io/user-guide/
[3] https://www.w3schools.com/
[4] https://www.tutorialspoint.com/codeigniter/
[5] https://www.malasngoding.com/category/codeigniter/
[6] https://www.codepolitan.com/