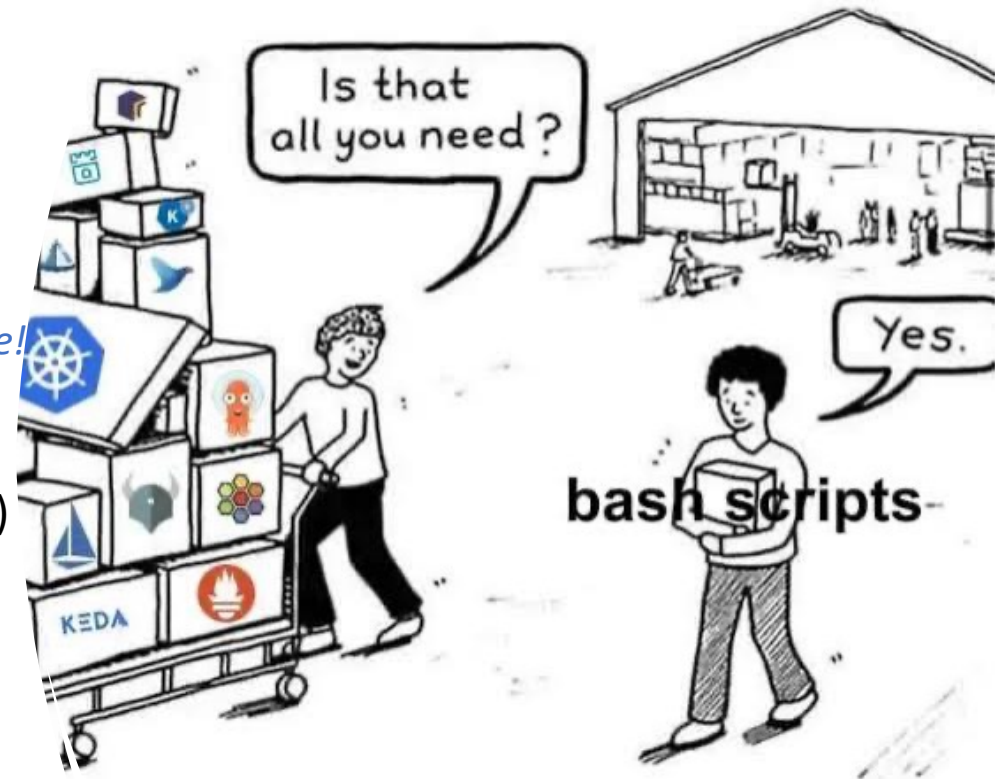


Lecture 10

408/508 *Computational Techniques for Linguists*

Today's Topics

- Some (8) Bash cheat sheets
 - *really a complete programming language!*
- New topic:
 - the *browser* is really a complete environment (*inside your computer*)
 - all browsers run html5:
 - html (for content)
 - css (for layout)
 - javascript (programming)
 - **goal:** building your own webserver



<https://devhints.io/bash>

Bash cheat sheet for parameter expansions

(what we discussed last time)

Parameter expansions

Basics

```
name="John"
echo ${name}
echo ${name:3/3}      #=> "john" (substitution)
echo ${name:0:2}      #=> "Jo" (slicing)
echo ${name::2}        #=> "Jo" (slicing)
echo ${name::-1}       #=> "Joh" (slicing)
echo ${name:~-1}       #=> "n" (slicing from right)
echo ${name:~-2:1}     #=> "h" (slicing from right)
echo ${food:-Cake}     #=> $food or "Cake"
```

```
length=2
echo ${name:0:length} #=> "Jo"
```

See: Parameter expansion

```
STR="/path/to/foo.cpp"
echo ${STR%.cpp}      # /path/to/foo
echo ${STR%.cpp}.o    # /path/to/foo.o
echo ${STR%/*}        # /path/to
```

```
echo ${STR##*.}       # cpp (extension)
echo ${STR##*/}       # foo.cpp (basepath)
```

```
echo ${STR#*/}        # path/to/foo.cpp
echo ${STR#*/*}       # foo.cpp
```

```
echo ${STR/foo/bar}   # /path/to/bar.cpp
```

```
STR="Hello world"
echo ${STR:0:5}       # "world"
echo ${STR: -5:5}     # "world"
```

```
SRC="/path/to/foo.cpp"
BASE=${SRC%*/}        #=> "foo.cpp" (basepath)
DIR=${SRC%$BASE}      #=> "/path/to/" (dirpath)
```

Substitution

\${FOO%suffix}	Remove suffix
\${FOO#prefix}	Remove prefix
\${FOO%%suffix}	Remove long suffix
\${FOO##prefix}	Remove long prefix
\${FOO/from/to}	Replace first match
\${FOO//from/to}	Replace all
\${FOO/%from/to}	Replace suffix
\${FOO/#from/to}	Replace prefix

Length

\${#FOO}	Length of \$FOO
----------	-----------------

Default values

\${FOO:-val}	\$FOO, or val if unset (or null)
\${FOO:=val}	Set \$FOO to val if unset (or null)
\${FOO:+val}	val if \$FOO is set (and not null)
\${FOO:?message}	Show error message and exit if \$FOO is unset (or null)

Omitting the : removes the (non)nullity checks, e.g. \${FOO-val} expands to val if unset otherwise \$FOO.

Comments

```
# Single line comment
```

```
: '
This is a
multi line
comment
'
```

Substrings

\${FOO:0:3}	Substring (position, length)
\${FOO:~-3:3}	Substring from the right

Manipulation

```
STR="HELLO WORLD!"
echo ${STR,,}          #=> "hello world!" (lowercase 1st letter)
echo ${STR,,}          #=> "hello world!" (all lowercase)
```

```
STR="hello world!"
echo ${STR^}           #=> "Hello world!" (uppercase 1st letter)
echo ${STR^^}          #=> "HELLO WORLD!" (all uppercase)
```

<https://devhints.io/bash>

Bash cheat sheet for `[[...]]`

Conditionals

Conditions

Note that `[[` is actually a command/program that returns either 0 (true) or 1 (false). Any program that obeys the same logic (like all base utils, such as `grep(1)` or `ping(1)`) can be used as condition, see examples.

<code>[[-z STRING]]</code>	Empty string
<code>[[-n STRING]]</code>	Not empty string
<code>[[STRING == STRING]]</code>	Equal
<code>[[STRING != STRING]]</code>	Not Equal
<code>[[NUM -eq NUM]]</code>	Equal
<code>[[NUM -ne NUM]]</code>	Not equal
<code>[[NUM -lt NUM]]</code>	Less than
<code>[[NUM -le NUM]]</code>	Less than or equal
<code>[[NUM -gt NUM]]</code>	Greater than
<code>[[NUM -ge NUM]]</code>	Greater than or equal
<code>[[STRING =~ STRING]]</code>	Regex
<code>((NUM < NUM))</code>	Numeric conditions
More conditions	
<code>[[-o noclobber]]</code>	If <code>OPTIONNAME</code> is enabled
<code>[[! EXPR]]</code>	Not
<code>[[X && Y]]</code>	And
<code>[[X Y]]</code>	Or

File conditions

<code>[[-e FILE]]</code>	Exists
<code>[[-r FILE]]</code>	Readable
<code>[[-h FILE]]</code>	Symlink
<code>[[-d FILE]]</code>	Directory
<code>[[-w FILE]]</code>	Writable
<code>[[-s FILE]]</code>	Size is > 0 bytes
<code>[[-f FILE]]</code>	File
<code>[[-x FILE]]</code>	Executable
<code>[[FILE1 -nt FILE2]]</code>	1 is more recent than 2
<code>[[FILE1 -ot FILE2]]</code>	2 is more recent than 1
<code>[[FILE1 -ef FILE2]]</code>	Same files

Example

```
# String
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
  echo "String is not empty"
else
  echo "This never happens"
fi

# Combinations
if [[ X && Y ]]; then
  ...
fi

# Equal
if [[ "$a" == "$b" ]]; then
  ...
fi

# Regex
if [[ "A" =~ . ]]; then
  ...
fi

if (( $a < $b )); then
  echo "$a is smaller than $b"
fi

if [[ -e "file.txt" ]]; then
  echo "file exists"
fi
```

Bash cheat sheet (scripting)

Variables

This is a quick reference to getting started with Bash scripting.

✱

```
git commit && git push
git commit || echo "Commit failed"
```

```
set -euo pipefail
IFS=$'\n\t'
```

See: [Unofficial bash strict mode](#)

```
#!/usr/bin/env bash

NAME="John"
echo "Hello $NAME!"
```

```
NAME="John"
echo "Hi $NAME"    ==> Hi John
echo 'Hi $NAME'    ==> Hi $NAME
```

```
get_name() {  
    echo "John"  
}  
  
echo "You are $(get_name)"
```

See: [Functions](#)

```
echo {A,B}.js
```

<code>{A,B}</code>	Same as A B
<code>{A,B}.js</code>	Same as A.js B.js
<code>{1..5}</code>	Same as 1 2 3 4 5

See: Brace expansion

```
NAME="John"
echo $NAME
echo "$NAME"
echo "${NAME}!"
```

```
echo "I'm in $(pwd)"
echo "I'm in `pwd`"
# Same
```

See [Command substitution](#)

```
if [[ -z "$string" ]]; then
    echo "String is empty"
elif [[ -n "$string" ]]; then
    echo "String is not empty"
fi
```

See: [Conditionals](#)

<https://devhints.io/bash>

Bash cheat sheet (loops and functions)

Loops

Basic for loop

```
for i in /etc/rc.*; do
  echo $i
done
```

C-like for loop

```
for ((i = 0 ; i < 100 ; i++)); do
  echo $i
done
```

Ranges

```
for i in {1..5}; do
  echo "Welcome $i"
done
```

With step size

```
for i in {5..50..5}; do
  echo "Welcome $i"
done
```

Reading lines

```
cat file.txt | while read line; do
  echo $line
done
```

Forever

```
while true; do
  ...
done
```

Functions

Defining functions

```
myfunc() {
  echo "hello $1"
}
```

```
# Same as above (alternate syntax)
function myfunc() {
  echo "hello $1"
}
```

```
myfunc "John"
```

Returning values

```
myfunc() {
  local myresult='some value'
  echo $myresult
}
```

```
result="$(myfunc)"
```

Raising errors

```
myfunc() {
  return 1
}
```

```
if myfunc; then
  echo "success"
else
  echo "failure"
fi
```

Arguments

\$#	Number of arguments
\$*	All arguments
@	All arguments, starting from first
\$1	First argument

<https://devhints.io/bash>

Bash cheat sheet: (arrays and dictionaries)

Arrays

Defining arrays

```
Fruits=('Apple' 'Banana' 'Orange')
```

```
Fruits[0]="Apple"  
Fruits[1]="Banana"  
Fruits[2]="Orange"
```

Operations

```
Fruits=("${Fruits[@]}" "Watermelon") # Push  
Fruits+=("Watermelon") # Also Push  
Fruits=( ${Fruits[@]/Ap*/} ) # Remove by regex match  
unset Fruits[2] # Remove one item  
Fruits=("${Fruits[@]}") # Duplicate  
Fruits=("${Fruits[@]}" "${Veggies[@]}") # Concatenate  
lines=(`cat "logfile"`) # Read from file
```

Working with arrays

```
echo ${Fruits[0]} # Element #0  
echo ${Fruits[-1]} # Last element  
echo ${Fruits[@]} # All elements, space-separated  
echo ${#Fruits[@]} # Number of elements  
echo ${Fruits[0]} # String length of the 1st element  
echo ${#Fruits[3]} # String length of the Nth element  
echo ${Fruits[@]:3:2} # Range (from position 3, length 2)  
echo ${!Fruits[@]} # Keys of all elements, space-separated
```

Iteration

```
for i in "${arrayName[@]"; do  
  echo $i  
done
```

Dictionaries

Defining

```
declare -A sounds
```

```
sounds[dog]="bark"  
sounds[cow]="moo"  
sounds[bird]="tweet"  
sounds[wolf]="howl"
```

Declares sound as a Dictionary object (aka associative array).

Working with dictionaries

```
echo ${sounds[dog]} # Dog's sound  
echo ${sounds[@]} # All values  
echo ${!sounds[@]} # All keys  
echo ${#sounds[@]} # Number of elements  
unset sounds[dog] # Delete dog
```

Iteration

Iterate over values

```
for val in "${sounds[@]"; do  
  echo $val  
done
```

Iterate over keys

```
for key in "${!sounds[@]"; do  
  echo $key  
done
```

<https://devhints.io/bash>

Bash cheat sheet

Options

Options

```
set -o noclobber # Avoid overlay files (echo "hi" > foo)
set -o errexit   # Used to exit upon error, avoiding cascading errors
set -o pipefail  # Unveils hidden failures
set -o nounset   # Exposes unset variables
```

Glob options

```
shopt -s nullglob # Non-matching globs are removed ('*.foo' => '')
shopt -s failglob # Non-matching globs throw errors
shopt -s nocaseglob # Case insensitive globs
shopt -s dotglob # Wildcards match dotfiles ("*.sh" => ".foo.sh")
shopt -s globstar # Allow ** for recursive matches ('lib/**/*.rb' => 'lib/a/b/c.rb')
```

Set GLOBIGNORE as a colon-separated list of patterns to be removed from glob matches.

History

Commands

history	Show history
shopt -s histverify	Don't execute expanded result immediately

Operations

!!	Execute last command again
!!:s/<FROM>/<TO>/	Replace first occurrence of <FROM> to <TO> in most recent command
!!:gs/<FROM>/<TO>/	Replace all occurrences of <FROM> to <TO> in most recent command
!\$:t	Expand only basename from last parameter of most recent command
!\$:h	Expand only directory from last parameter of most recent

Expansions

!\$	Expand last parameter of most recent command
!*	Expand all parameters of most recent command
!-n	Expand nth most recent command
!n	Expand nth command in history
!<command>	Expand most recent invocation of command <command>

Slices

!!:n	Expand only nth token from most recent command (command is 0; first argument is 1)
!^	Expand first argument from most recent command
!\$	Expand last token from most recent command

<https://devhints.io/bash>

Bash cheat sheet

Miscellaneous

Numeric calculations

```
$(a + 200) # Add 200 to $a
```

```
$(($RANDOM%200)) # Random number 0..199
```

Inspecting commands

```
command -V cd  
#=> "cd is a function/alias/whatever"
```

Trap errors

```
trap 'echo Error at about $LINENO' ERR
```

or

```
traperr() {  
  echo "ERROR: ${BASH_SOURCE[1]} at about ${BASH_LINENO[0]}"  
}
```

```
set -o erretrace  
trap traperr ERR
```

Source relative

```
source "${0%/*}/../share/foo.sh"
```

Directory of script

```
DIR="${0%/*}"
```

Subshells

```
(cd somedir; echo "I'm now in $PWD")  
pwd # still in first directory
```

Redirection

```
python hello.py > output.txt # stdout to (file)  
python hello.py >> output.txt # stdout to (file), append  
python hello.py 2> error.log # stderr to (file)  
python hello.py 2>&1 # stderr to stdout  
python hello.py 2>/dev/null # stderr to (null)  
python hello.py &>/dev/null # stdout and stderr to (null)
```

```
python hello.py < foo.txt # feed foo.txt to stdin for python
```

Case/switch

```
case "$1" in  
  start | up)  
    vagrant up  
    ;;  
  *)  
    echo "Usage: $0 {start|stop|ssh}"  
    ;;  
esac
```

printf

```
printf "Hello %s, I'm %s" Sven Olga  
#=> "Hello Sven, I'm Olga"
```

```
printf "1 + 1 = %d" 2  
#=> "1 + 1 = 2"
```

<https://devhints.io/bash>

Bash cheat sheet

Getting options

```
while [[ "$1" =~ ^- && ! "$1" == "--" ]]; do case $1 in
-V | --version )
    echo $Version
    exit
;;
-s | --string )
    shift; string=$1
;;
-f | --flag )
    flag=1
;;
esac; shift; done
if [[ "$1" == "--" ]]; then shift; fi
```

Special variables

\$?	Exit status of last task
\$!	PID of last background task
\$\$	PID of shell
\$0	Filename of the shell script
See Special parameters.	

Grep check

```
if grep -q 'foo' ~/.bash_history; then
    echo "You appear to have typed 'foo' in the past"
fi
```

```
##=> "this is now you print a float: 2.000000"
```

Heredoc

```
cat <<END
hello world
END
```

Reading input

```
echo -n "Proceed? [y/n]: "
read ans
echo $ans
```

```
read -n 1 ans    # Just one character
```

Go to previous directory

```
pwd # /home/user/foo
cd bar/
pwd # /home/user/foo/bar
cd -
pwd # /home/user/foo
```

Check for command's result

```
if ping -c 1 google.com; then
    echo "It appears you have a working internet connection"
fi
```

Browser

- Nowadays **browsers** are very powerful in their own right (can compute locally, not just display pages from a webserver)



Frontend
stuff

CSS (Cascading Style Sheets)

SVG (Scalable Vector Graphics)

- cf. HTML5 canvas

Javascript

- *programming language*

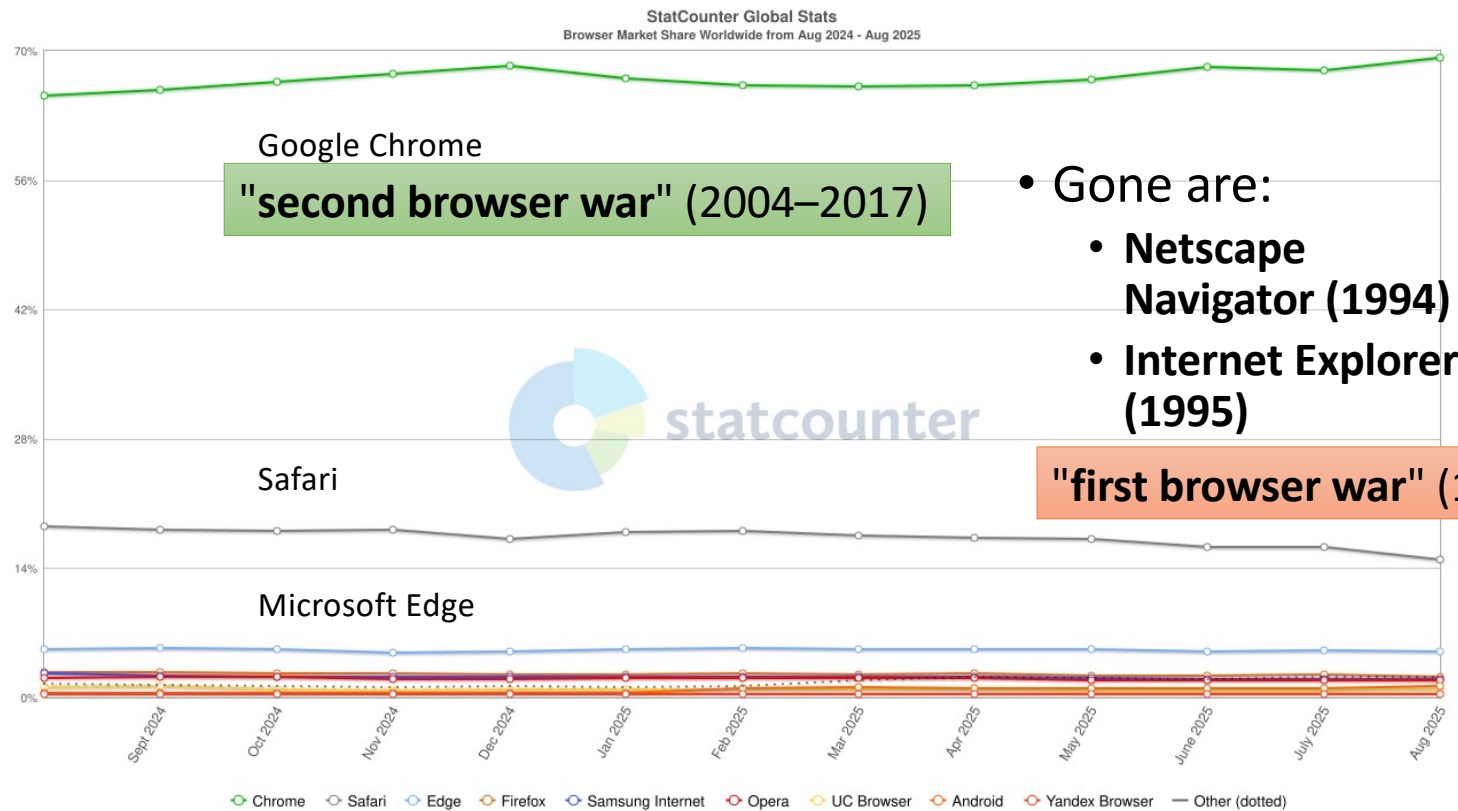
DOM (Domain Object Model)

- *programmatically access to documents*

Websockets

- An API to interact with regular programs

Browsers



• Gone are:

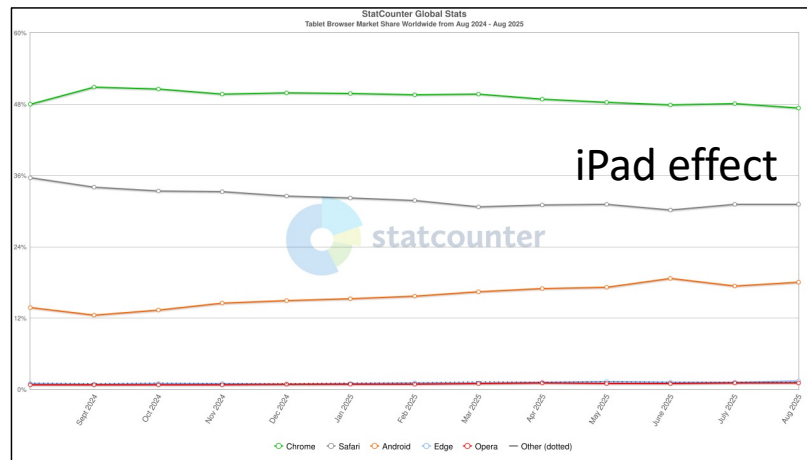
- Netscape Navigator (1994)
- Internet Explorer (1995)



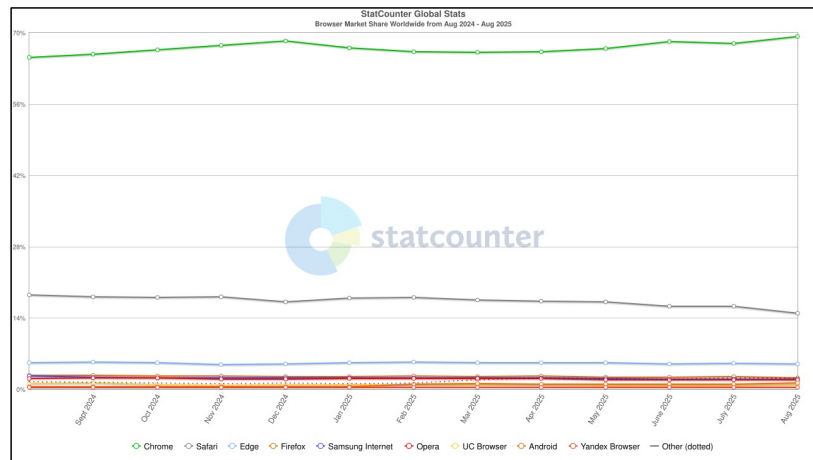
"first browser war" (1995–2001)

Tablets vs Overall

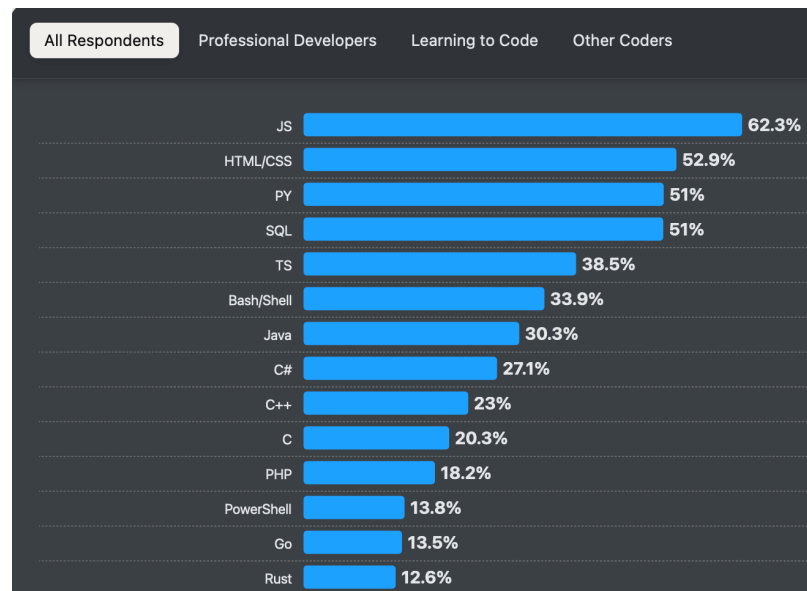
Tablets



Overall



Programming Language



- **stackoverflow:**

- The most popular programming language has been JavaScript every year we have done the survey except for 2013 and 2014, when SQL was the most popular language.

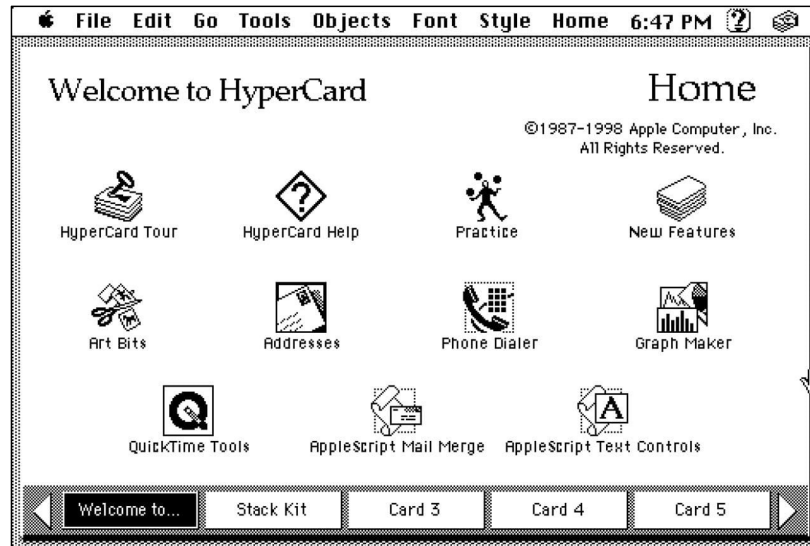
- Question asked:

- *Which **programming, scripting, and markup languages** have you done extensive development work in over the past year, and which do you want to work in over the next year?*

HTML

- HTML: Hypertext Markup Language
 - Web browser: can read and render pages written in HTML
 - **currently:** HTML5
- What is "hypertext"?
 - linked content (Nelson, 1963)
 - nowadays: selected text/images/video can have arbitrary associated actions
- Hypercard for the Macintosh (1987)
- World Wide Web (WWW) (1992)
 - World Wide Web Consortium (W3C)

Hypercard



- Before WWW
 - 1987
 - card stack with links
- Myst
 - 1993
 - a computer game
 - developed using Hypercard

World's first webpage

World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#) , [Policy](#) , November's [W3 news](#) , [Frequently Asked Questions](#) .

[What's out there?](#)

Pointers to the world's online information, [subjects](#) , [W3 servers](#), etc.

[Help](#)

on the browser you are using

[Software Products](#)

A list of W3 project components and their current state. (e.g. [Line Mode](#) ,X11 [Viola](#) , [NeXTStep](#) , [Servers](#) , [Tools](#) , [Mail robot](#) , [Library](#))

[Technical](#)

Details of protocols, formats, program internals etc

[Bibliography](#)

Paper documentation on W3 and references.

[People](#)

A list of some people involved in the project.

[History](#)

A summary of the history of the project.

[How can I help ?](#)

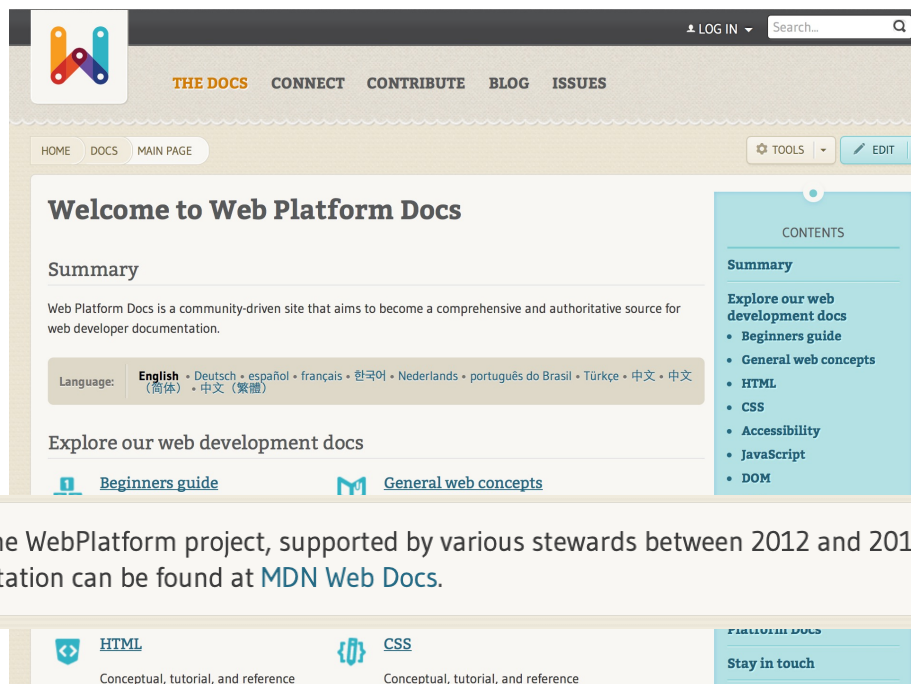
If you would like to support the web..

[Getting code](#)

Getting the code by [anonymous FTP](#) , etc.

- At CERN:
 - August 6 1991
 - Tim Berners-Lee on a **NeXT** computer
- Note:
 - all text, no graphics
 - no fancy fonts
 - no sidebars
 - **no advertising!**
 - no Javascript

Reference



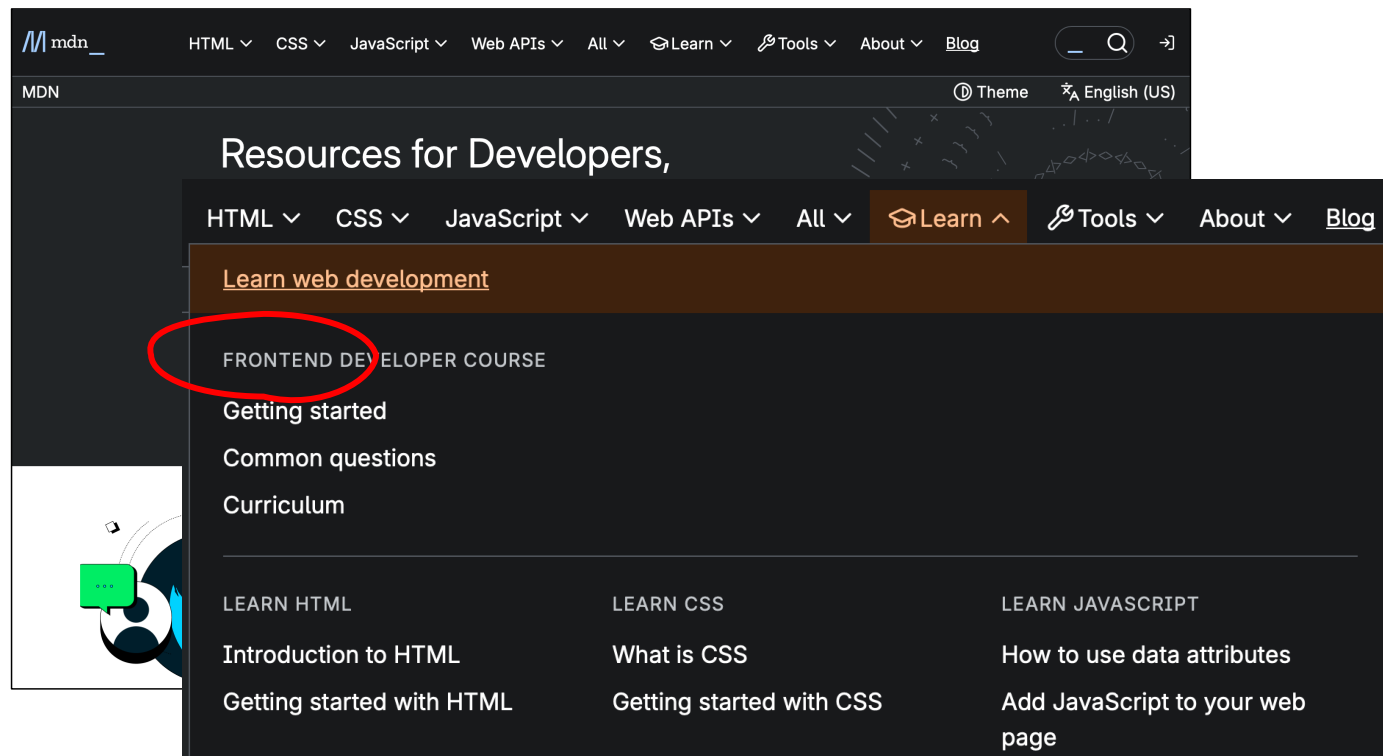
<https://webplatform.github.io>

Now deprecated...

Notice: The WebPlatform project, supported by various stewards between 2012 and 2015, has been **discontinued**. This site is now available on [github](#). New documentation can be found at [MDN Web Docs](#).

MDN

- <https://developer.mozilla.org/en-US/>



Reference

<https://developer.mozilla.org/en-US/docs/Web/Tutorials>

 mdn web docs

References > Tutorials

English (US) ▾

In this article

For complete beginners to the Web

HTML Tutorials

CSS Tutorials

JavaScript Tutorials

For complete beginners to the Web

[Getting started with the Web](#)

Getting started with the Web is a concise series introducing you to the practicalities of web development. You'll set up the tools you need to construct a simple webpage and publish your own simple code.

HTML Tutorials

Introductory level

[Introduction to HTML](#)

This module sets the stage, getting you used to important

 mdn web docs

References > Tutorials

English (US) ▾

In this article

For complete beginners to the Web

HTML Tutorials

CSS Tutorials

JavaScript Tutorials

CSS Tutorials

Introductory level

[CSS basics](#)

CSS (Cascading Style Sheets) is the code you use to style your webpage. CSS Basics takes you through what you need to get started. We'll answer questions like: How do I make my text black or red? How do I make my content show up in such-and-such a place on the screen? How do I decorate my webpage with background images and colors?

[CSS first steps](#)

CSS (Cascading Style Sheets) is used to style and lay out web pages — for example, to alter the font, color, size, and spacing of your content, split it into multiple columns, or add animations and other decorative features. This module

Reference

<https://developer.mozilla.org/en-US/docs/Web/Tutorials>

 mdn web docs

References > Tutorials

English (US)

In this article

- For complete beginners to the Web
- HTML Tutorials
- CSS Tutorials**
- JavaScript Tutorials

CSS Challenges

Flex your CSS skills, and see where you need more practice.

Advanced level

Using CSS transforms

Apply rotation, skewing, scaling, and translation using CSS.

CSS transitions

CSS transitions provide a way to animate changes to CSS properties, instead of having the changes take effect instantly.

Canvas tutorial

Learn how to draw graphics using scripting using the canvas element.

 mdn web docs

References > Tutorials

English (US)

JavaScript Tutorials

In this article

- For complete beginners to the Web
- HTML Tutorials
- CSS Tutorials
- JavaScript Tutorials**

Introductory level

JavaScript first steps

In our first JavaScript module, we first answer some fundamental questions such as "what is JavaScript?", "what does it look like?", and "what can it do?", before moving on to taking you through your first practical experience of writing JavaScript. After that, we discuss some key JavaScript features in detail, such as variables, strings, numbers and arrays.

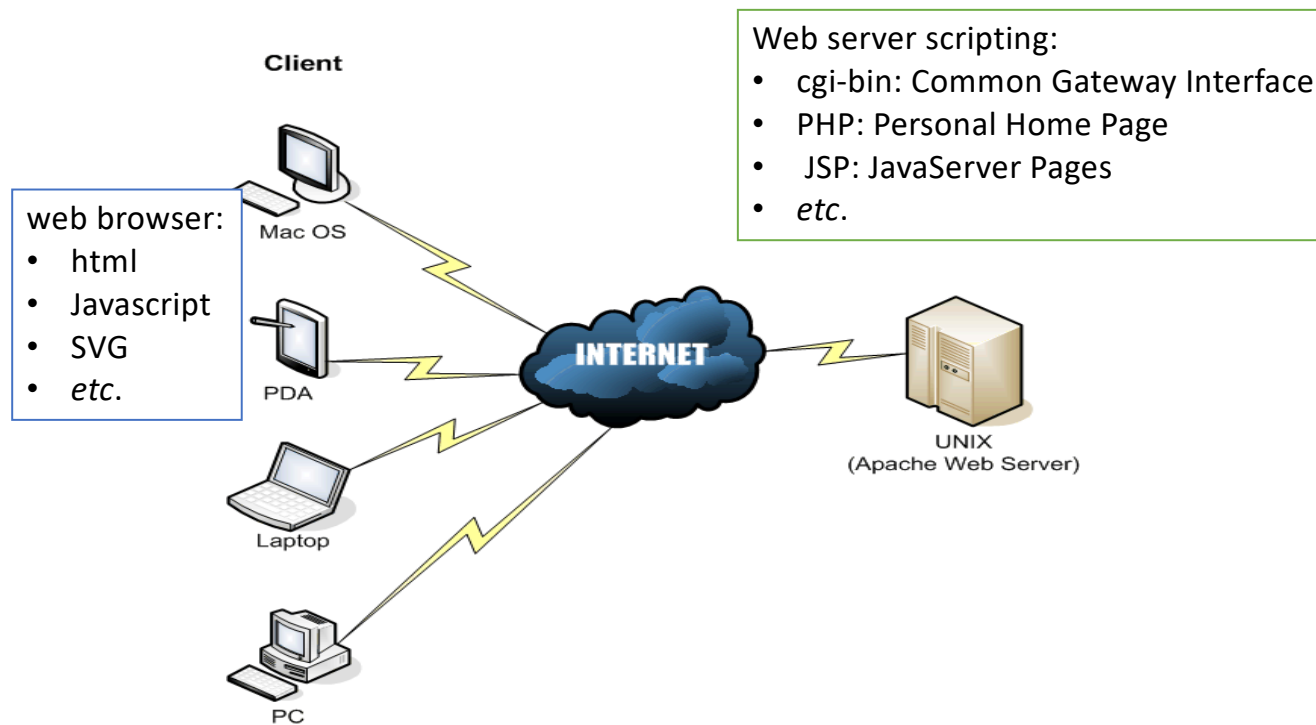
JavaScript building blocks

In this module, we continue our coverage of all JavaScript's key fundamental features, turning our attention to commonly-encountered types of code blocks such as

Frontend web development

- **HTML:**
 - structure of content
- CSS (cascading style sheets):
 - presentation
- Javascript
 - scripting language
- DOM (document object model):
 - hierarchical representation of webpage
- SVG (scalable vector graphics):
 - 2D graphical objects and methods

Client/server model



<http://www.visualbuilder.com/jsp/tutorial/introduction-to-jsp/>

HTML

boilerplate inserted by my Aquamacs editor:

```
1 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
2 <html> <head>
3 <title></title>
4 </head>
5
6 <body>
7 <h1></h1>
8
9
10
11 <hr>
12 <address></address>
13 <!-- hhmts start --><!-- hhmts end -->
14 </body> </html>
```

- First line:
 - <!DOCTYPE HTML>
 - signifies HTML5
- Tags:
 - <tag> ... </tag>
 - html
 - head: title, style, javascript definitions etc.
 - body: body of the document
 - h1: heading level 1 (1-6)
 - address: contact information
- "self-closing" tags:
 - hr: horizontal rule
 - br: line break optional:

- optionally paired:
 - <p> .. </p>: paragraph
- comment:
 - <!-- ... -->

HTML

- hypertext (link):
 - `<a href=URL>text</a>` (text presented in blue)
- URL: uniform resource locator
 - Examples:
 - <http://sandiway.arizona.edu/>
 - <http://nlp.stanford.edu:8080/parser/>
 - <https://netid.arizona.edu/newid.php> (PHP)
 - <http://localhost/perl/test.pl> (mod_perl program)
 - Format:
 - protocol://host(:port)/path
 - protocol://host(:port)/path?query
 - protocol = http (hypertext transfer protocol)
 - port = TCP/IP port number

HTML

- Images:
 - `<img src=URL>`
 - attribute: src (required)
 - value: URL (or filename etc.) (jpg, gif, png supported, see note below)
 - attribute: alt (supposed to be required)
 - value: text
 - attribute: height
 - value: pixels
 - attribute: width
 - value: pixels
 - attribute: align (not in HTML5)
 - value: top|bottom|middle|left|right
- Can embed:
 - `<a href=URL><img src=URL></a>`
- **Note:** http://en.wikipedia.org/wiki/Comparison_of_web_browsers#Image_format_support

HTML

- Images can be embedded inside the file via base64 encoding:

```

```



Last modified: Mon Sep 25 19:58:54 MST 2023

HTML

- On Ubuntu:
 - `base64 imagefile > encoded.txt`
- macOS:
 - `base64 -i imagefile -o encoded.txt`
- base64 in Windows:
 - `certutil -encode imagefile encoded.txt`
- HTML file:
 - ``



put contents of
encoded.txt
in this gap!

HTML

- Example (macOS):

```
$ ls -l sandiway.png
```

```
-rw-r--r--@ 1 sandiway  staff  158266 Sep 25 19:45 sandiway.png
```

```
$ base64 -i sandiway.png -o encoded.txt
```

```
$ ls -l encoded.txt
```

```
-rw-r--r--  1 sandiway  staff  211025 Sep 25 19:47 encoded.txt
```

HTML

sandiway@sandiway-XPS-15-9570: ~/Desktop

```
GNU nano 6.2 test.html *
<html>
<head></head>
<body>
 ... </b>`
- `<i> ... </i>`

**bold**

*italics*

- block-level:

- `<pre> ... </pre>`

preformatted

The **HTML** `<pre>` **element** represents preformatted text which is to be presented exactly as written in the HTML file. The text is typically rendered using a non-proportional ("**monospace**") font. Whitespace inside this element is displayed as written.

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/pre>

# HTML

HTML

CSS

```
1 <pre>
2 L TE
3 A A
4 C V
5 R A
6 DOU
7 LOU
8 REUSE
9 QUE TU
10 PORTES
11 ET QUI T'
12 ORNE O CI
13 VILISÉ
14 OTE- TU VEUX
```

Output

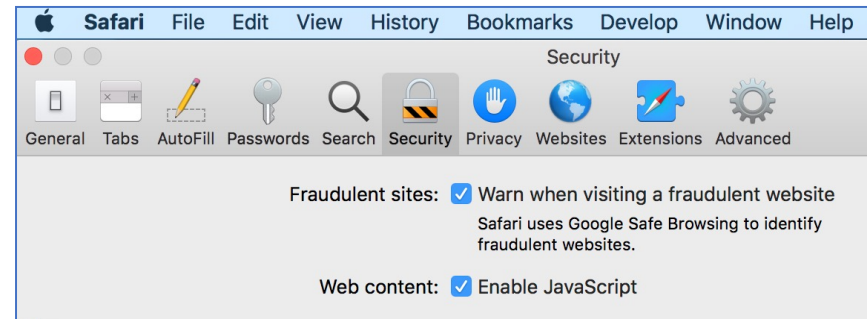
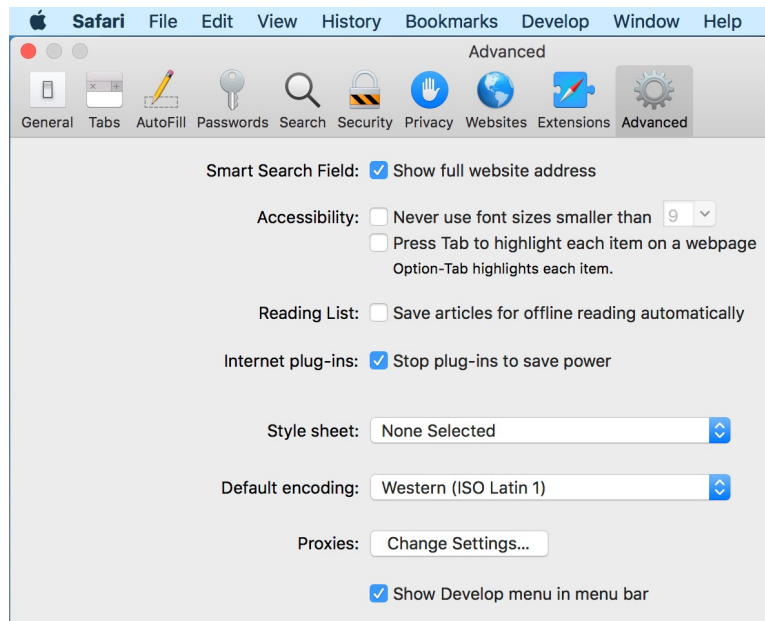
```
L TE
A A
C V
R A
DOU
LOU
REUSE
QUE TU
PORTES
ET QUI T'
ORNE O CI
VILISÉ
OTE- TU VEUX
LA BIEN
SI RESPI
RER - Apollinaire
```

# HTML

- Lists:
  - list item: `<li> ... </li>`
  - ordered lists: `<ol> ... </ol>`
  - unordered lists: `<ul> ... </ul>`
  - **note**: can be nested arbitrarily deep

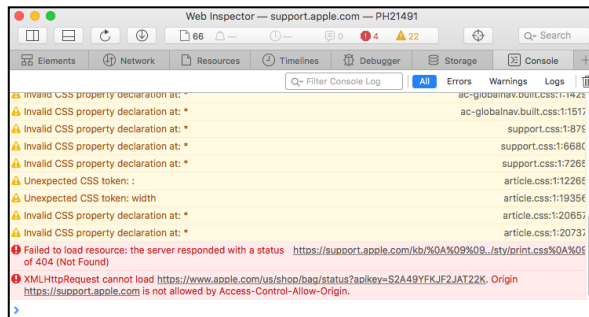
# Debugging

- In Safari:
  - Preferences > Advanced > **Show Develop menu in menu bar**



# Debugging

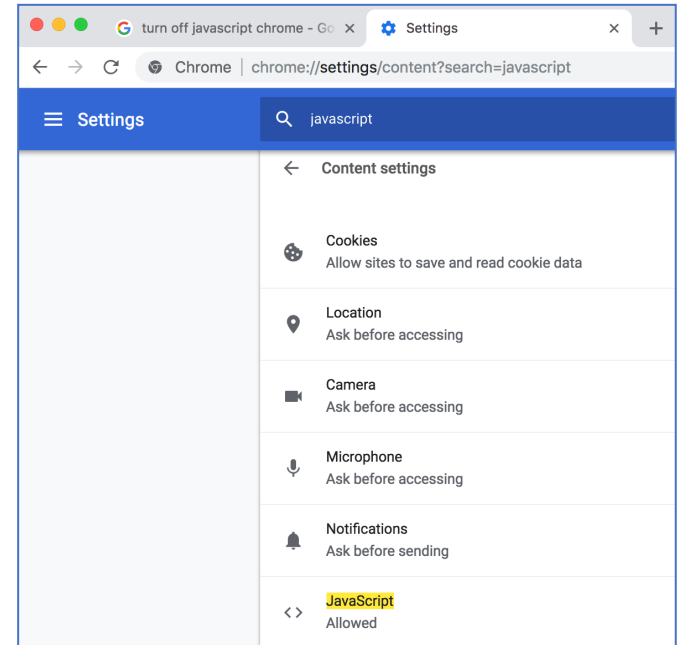
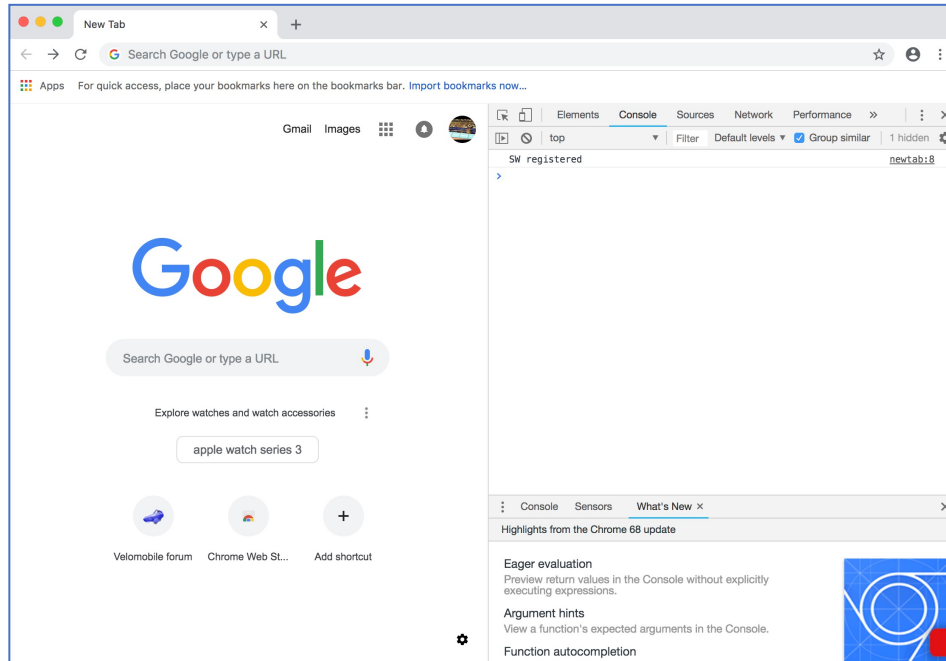
- In the Develop menu:
  - *Allow JavaScript from Smart Search Field*
  - **javascript:** ok in address bar
  - Show Page Source
  - Show Web Inspector



| Develop                                  | Window | Help |
|------------------------------------------|--------|------|
| Open Page With User Agent                |        |      |
| Hide Web Inspector                       |        |      |
| Show Error Console                       |        |      |
| Show Page Source                         |        |      |
| Show Page Resources                      |        |      |
| Show Snippet Editor                      |        |      |
| Show Extension Builder                   |        |      |
| Start Timeline Recording                 |        |      |
| Empty Caches                             |        |      |
| Disable Caches                           |        |      |
| Disable Images                           |        |      |
| Disable Styles                           |        |      |
| Disable JavaScript                       |        |      |
| Disable Site-specific Hacks              |        |      |
| Disable Local File Restrictions          |        |      |
| Enable WebGL                             |        |      |
| Allow JavaScript from Smart Search Field |        |      |

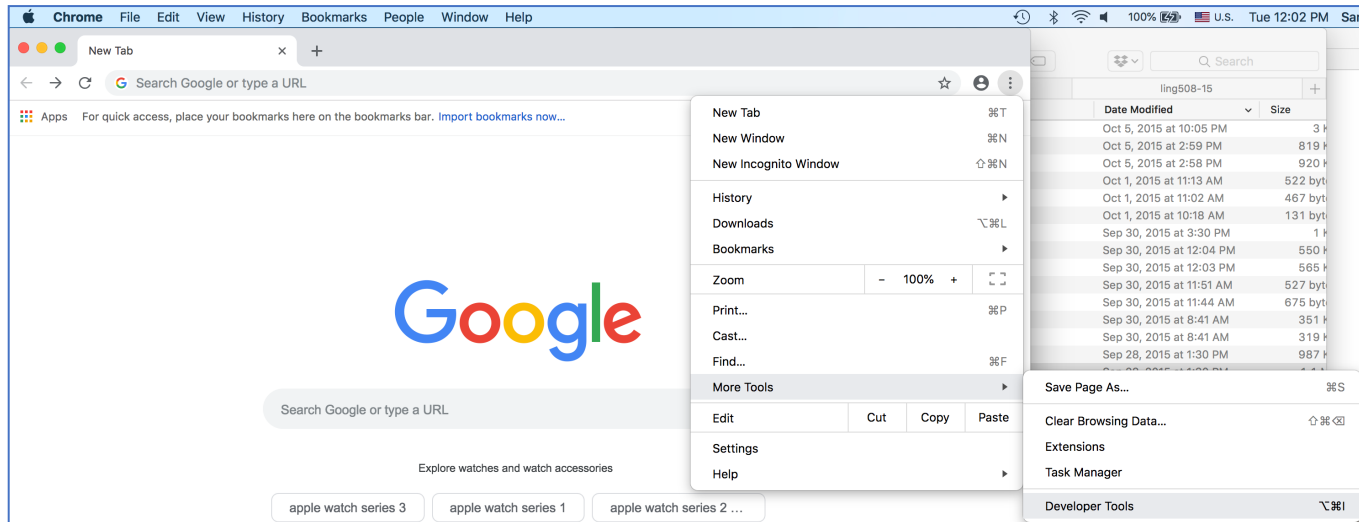
# Debugging

- Developer Tools and Javascript in Google Chrome:



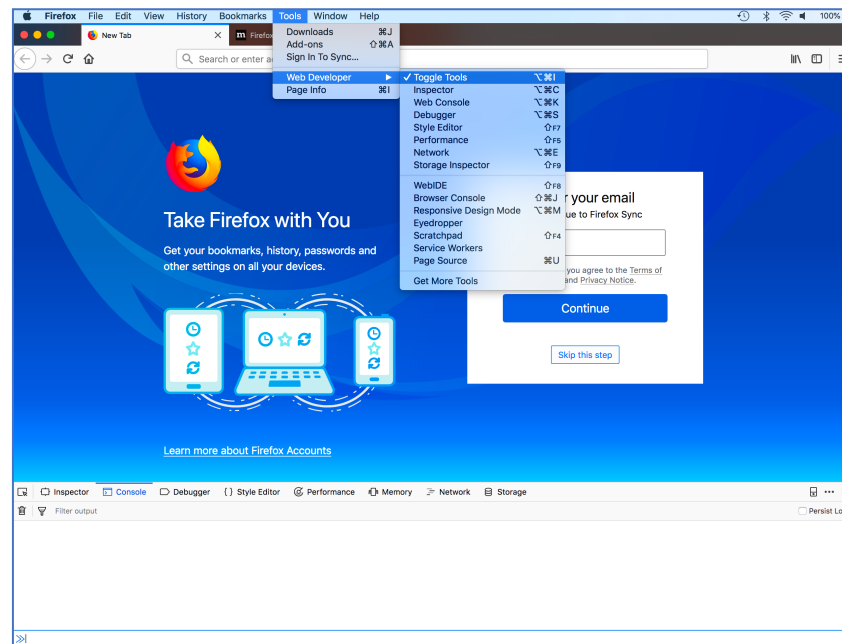
# Debugging

- In Google Chrome:



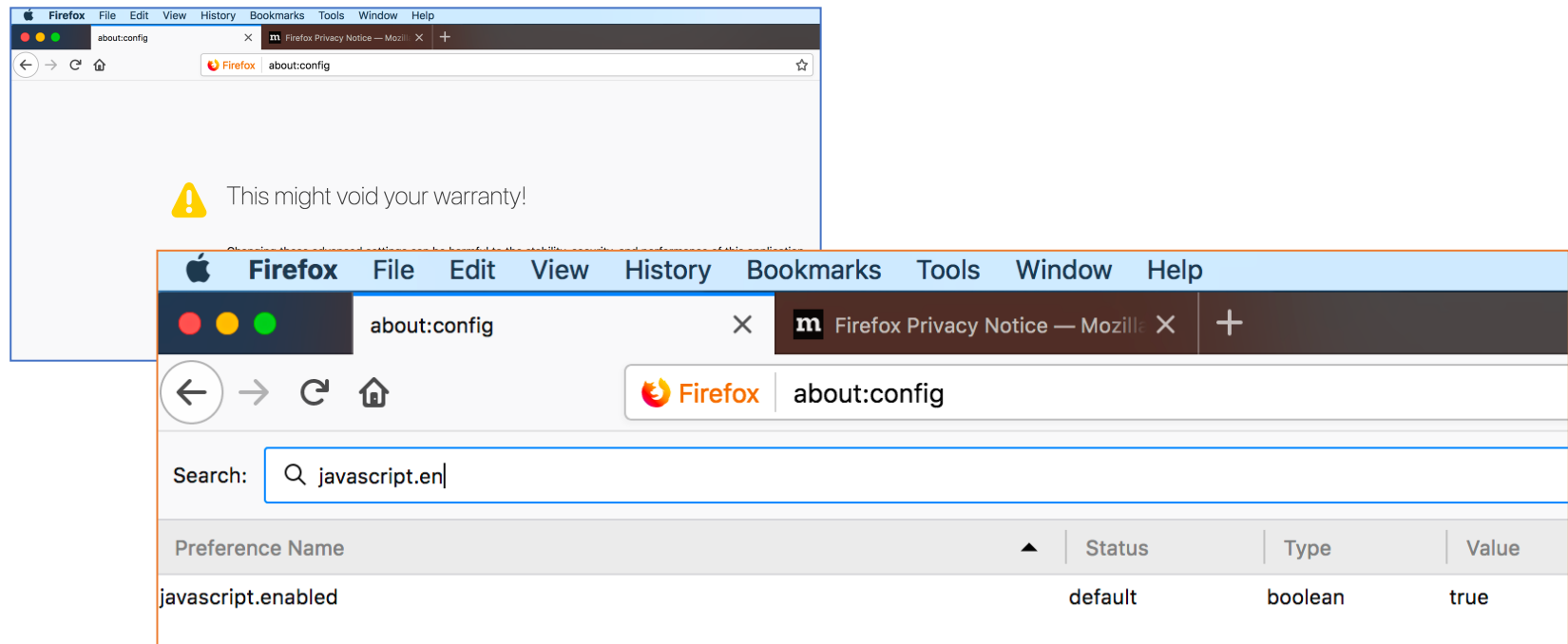
# Debugging

- Firefox:



# Debugging

- Firefox:



# Ungraded Homework Exercise

- Build your own html file.
- Play with the tags
- Embed a picture of yourself encoded in base64