# Internet:

bits – binary – on/off – byte - atoms

electric, light, radio wave

protocol

addresses

ipv6 – 128 bits

DNS – domain name server

spoofing

packet of info

different routes, routing, options

pieces, packets

reliability

tcp control

scalable

url – uniform resource locator

computer – server

hypertext transfer protocol

html

„post” request

cookie

security ssl tls

http&dns // tcp/ip and routing //wires, cables & wirless

encryption

256 bit keys

symmetric/asymmetric

hackers

cyber-army

virus-dos-phishing

human mistake

# Command line

<https://www.codecademy.com/articles/command-line-commands>

* pwd outputs the name of the current working directory.
* ls lists all files and directories in the working directory.
* cd switches you into the directory you specify.
* mkdir creates a new directory in the working directory.
* touch creates a new file inside the working directory
* Options modify the behavior of commands:
  + ls -a lists all contents of a directory, including hidden files and directories
  + ls -l lists all contents in long format
  + ls -t orders files and directories by the time they were last modified
  + Multiple options can be used together, like ls -alt
* From the command line, you can also copy, move, and remove files and directories:
  + cp copies files
  + mv moves and renames files
  + rm removes files
  + rm -r removes directories
* Wildcards are useful for selecting groups of files and directories
* *Redirection* reroutes standard input, standard output, and standard error.
* The common redirection commands are:
  + > redirects standard output of a command to a file, overwriting previous content.
  + >> redirects standard output of a command to a file, appending new content to old content.
  + < redirects standard input to a command.
  + | redirects standard output of a command to another command.
* A number of other commands are powerful when combined with redirection commands:
  + sort: sorts lines of text alphabetically.
  + uniq: filters duplicate, adjacent lines of text.
  + grep: searches for a text pattern and outputs it.
  + sed : searches for a text pattern, modifies it, and outputs it.
* The *environment* refers to the preferences and settings of the current user.
* The *nano* editor is a command line text editor used to configure the environment.
* **~/.bash\_profile** is where environment settings are stored. You can edit this file with nano.
* *environment variables* are variables that can be used across commands and programs and hold information about the environment.
  + export VARIABLE="Value" sets and exports an environment variable.
  + USER is the name of the current user.
  + PS1 is the command prompt.
  + HOME is the home directory. It is usually not customized.
  + PATH returns a colon separated list of file paths. It is customized in advanced cases.
  + env returns a list of environment variables.

# Git

## basics

verziókövető rendszer

* Git is the industry-standard version control system for web developers
* Use Git commands to help keep track of changes made to a project:
  + git init creates a new Git repository
  + git status inspects the contents of the working directory and staging area
  + git add adds files from the working directory to the staging area
  + git diff shows the difference between the working directory and the staging area
  + git commit permanently stores file changes from the staging area in the repository
  + git log shows a list of all previous commits
* git checkout HEAD filename: Discards changes in the working directory.
* git reset HEAD filename: Unstages file changes in the staging area.
* git reset SHA: Can be used to reset to a previous commit in your commit history.

Additionally, you learned a way to add multiple files to the staging area with a single command: git add filename\_1 filename\_2

* Git *branching* allows users to experiment with different versions of a project by checking out separate *branches* to work on.

The following commands are useful in the Git branch workflow.

* git branch: Lists all a Git project's branches.
* git branch branch\_name: Creates a new branch.
* git checkout branch\_name: Used to switch from one branch to another.
* git merge branch\_name: Used to join file changes from one branch to another.
* git branch -d branch\_name: Deletes the branch specified.
* A *remote* is a Git repository that lives *outside* your Git project folder. Remotes can live on the web, on a shared network or even in a separate folder on your local computer.
* The *Git Collaborative Workflow* are steps that enable smooth project development when multiple collaborators are working on the same Git project.

We also learned the following commands

* git clone: Creates a local copy of a remote.
* git remote -v: Lists a Git project's remotes.
* git fetch: Fetches work from the remote into the local copy.
* git merge origin/master: Merges origin/master into your local branch.
* git push origin <branch\_name>: Pushes a local branch to the originremote.

Git projects are usually managed on Github, a website that hosts Git projects for millions of users. With Github you can access your projects from anywhere in the world by using the basic workflow you learned here.

**Git – working with remotes**

[**https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes**](https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes)

git remote command. It lists the shortnames of each remote handle you’ve specified

command line-ban „man” paranccsal meg lehet nézni hogy kell használni a parancsokat

git: jól követi a sok változtatást, sokféle verziót, sok felhasználó párhuzamos változtatását

git remote: push parancs, és megadva a remote, ahová felrakjuk

klónozás: leszedi az egészet

pull: be van álltva remote. ha van szerveren olyan commit ami nekem nincs meg, akkor azokat lehúzza

clone-nal kezdünk, és frissítünk a pull-lal

git gui a felhaszn felület

git bash command line

ctrl+c – vel ki lehet lépni a cat-ből

hogy ne kelljen mindifg megadni a jelszót push-olásnál

ssh-keygen

public/private keygen pair

ell.: ls ~/.ssh/ és ott lesz egy pub végű

**git stash**

elmenti és törli a dolgokat a staging-ből

git stash list: megmutaja miket stasheltem

visszahozás: git stash apply

git stash drop meg pop

## branching

<http://learngitbranching.js.org/>

### branches:

pointers to a specific commit

create: git branch new\_branch

git checkout [name] - This will put us on the new branch before committing our changes

if you want to create a new branch AND check it out at the same time, you can simply type git checkout -b [yourbranchname].

### Branches and Merging:

combining the work from two different branches together

git merge (branch-name) – merge-eli a masterbe, ha a master-en állsz

git checkout bugFix; git merge master

### rebasing:

second way of combining work between branches

(That way it would look like these two features were developed sequentially, when in reality they were developed in parallel)

Checkout a new branch named bugFix, Commit once, Go back to master and commit again

Check out bugFix again and rebase onto master (ennél a branch-en állsz)

### Moving around in Git

HEAD always points to the most recent commit

Detaching HEAD: attaching it to a commit instead of a branch.

### Relative Refs

use git log to see hashes (commit names)

With relative refs, you can start somewhere memorable (like the branch bugFix or HEAD) and work from there.

Moving upwards one commit at a time with ^

Moving upwards a number of times with ~<num>

Each time you append ^ to a ref name, you are telling Git to find the parent of the specified commit.

vagy HEAD^

tilde operator (optionally) takes in a trailing number that specifies the number of parents you would like to ascend

You can directly reassign a branch to a commit with the -f option: git branch -f master HEAD~3

git branch –f master C6

git checkout HEAD~1

git branch –f bugFix HEAD ~1

### Reversing Changes

has both a low-level component (staging individual files or chunks) and a high-level component (how the changes are actually reversed)

git reset / git revert

git reset: moving a branch reference backwards in time to an older commit. git reset will move a branch backwards as if the commit had never been made in the first place.

In order to reverse changes and share those reversed changes with others, we need to use git revert.

git reset HEAD~1

git checkout pushed

git revert HEAD

### Moving Work Around

git cherry-pick <Commit1> <Commit2> <...>

to copy a series of commits below your current location (HEAD)

Git Interactive Rebase: using the rebase command with the -i option.

You can reorder commits, completely omit some commits (by pick), you can squash commits (combine)

git rebase -i HEAD~4

**Locally stacked commits:** We need to tell git to copy only one of the commits over

### Juggling Commits

You have some changes (newImage) and another set of changes (caption) that are related, so they are stacked on top of each other in your repository

we used rebase -i to reorder the commits. Once the commit we wanted to change was on top, we could easily --amend it and re-order back to our preferred order.

git commit --amend

### Git Tags

a way to permanently mark historical points in your project's history

they never move as more commits are created

git tag v1 c1 – c1-re rak egy ’v1’ tag-et

If you leave the commit off, git will just use whatever HEAD is at

you go into detached HEAD state -- this is because you can't commit directly onto the v1 tag.

### Git Describe

git has a command to describe where you are relative to the closest "anchor" (aka tag)

git describe <ref> (Where <ref> is anything git can resolve into a commit)

The output of the command looks like:

<tag>\_<numCommits>\_g<hash>

Where tag is the closest ancestor tag in history, numCommits is how many commits away that tag is, and <hash> is the hash of the commit being described.

### Rebasing Multiple Branches

git rebase master bugfix

git rebase bugfix side

git rebase side another

git rebase another master

### Specifying Parents

the ^ modifier also accepts an optional number after it

### Git Remotes

they are actually just copies of your repository on another computer

### Git Remote Branches

a new branch appeared in our local repository called o/master. This type of branch is called a remote branch

Remote branches reflect the state of remote repositories

when you check them out, you are put into **detached HEAD** mode. Git does this on purpose because you can't work on these branches directly

they are displayed in the format of: <remote name>/<branch name>

git sets up your remote to be named origin when you git clone a repository

### Git Fetch

to fetch data from a remote repository

it downloads the commits that the remote has but are missing from our local repository, and updates where our remote branches point (for instance, o/master)

It will not update your master branch

git fetch is just a download step

### Git Pull

git pull is essentially a shorthand for a git fetch followed by a merge of whatever branch was just fetched.

### Git Push

a command to "publish" your work

### Diverged Work

git doesn't allow you to push your changes. It forces you to incorporate the latest state of the remote before being able to share your work.

most straightforward is to move your work via rebasing:

fetch, rebase o/master, push

or with merge: fetch, merge o/master, push

git pull --rebase is shorthand for a fetch and a rebase

### Merging feature branches

It's common for developers on big projects to do all their work on feature branches (off of master) and then integrate that work only once it's ready

Some developers only push and pull when on the master branch -- that way master always stays updated to what is on the remote (o/master).

we combine two things: integrating feature branch work onto master and push/pull from the remote

merge or rebase: Rebasing makes your commit tree look very clean since everything is in a straight line, but modifies the (apparent) history of the commit tree

### Remote-Tracking branches

git checkout -b totallyNotMaster o/master creates a new branch named totallyNotMaster and sets it to track o/master.

git checkout -b foo o/master; git pull

git checkout -b foo o/master; git commit; git push

another way:

git branch -u o/master foo; git commit; git push

### Push arguments

git push <remote> <place>

git push origin master: *„Go to the branch named "master" in my repository, grab all the commits, and then go to the branch "master" on the remote named "origin." Place whatever commits are missing on that branch and then tell me when you're done.”*

it’s not important where we are checked out!

**<place> argument details:**

git push origin <source>:<destination>

git push origin master:newBranch

### fetch arguments

git fetch origin foo: it will go to the foo branch on the remote, grab all the commits that aren't present locally, and then plop them down onto the o/foo branch locally

### delete

we can delete the foo branch on remote by pushing the concept of "nothing" to it

(git push origin :foo)

fetching "nothing" to a place locally actually makes a new branch

# HTML&CSS

## html

kacsacsőrrel indul és végződik

tag

attribútum

tag-ek: dobozok

szelektor – bajuszzárójel – szabály - bajusz.

meta: ékezetes betűk miatt fontos (utf8)

heading: h1-h6

div: divízió

p: paragraph

tulajdonság: érték

css ->

betűvastagság: font-weight – normál v. vastag

color: pl. rgba(0,0,0,.8)

méret: font size : pl. 16px

rendezés: text-align - center

font family

webfontok mindenhol működnek – pl. google fonts

ad egy linket, be kell másolni a html-be (header) és aztán a css-ben le kell hivatkozni a betűtípust

**html**: structure

**css**: design

javascript: interactive

Tags

img: no closing bracket

**<> </>**

tags can have attributes

nested tags: pl em, strong

doc type

html szendvicsben a kenyér

először head-tag

meta charset=”utf-8” self contained tag!

title tag!

aztán body (csak egy van, ebben van a tartalom)

head: extra , body: visual

had: meta tags: charset, keywords, description, for search engines

css-link: <link rel=”stylesheet” type=”text/css” href=”css/main.css”>

body: h1-h5

p embeds: small, mark, sub, sup, pre

browser styles

images: <img src=”path”>

width=”100%”

de jobb css-ben megadni

links: internal, external, dl, anchor

<a href=”source”> contact us </a>

go up: ../

linkek <http://-vel>

href=„#top”: felmegy

target=”\_blank” : új ablakban nyílik meg a link

lists:

* unordered <ul> + <li>
* ordered <ol> + <li>
* definition <dl> <dd> <dt>

div: divide content into specific sections

id’s & classes

id egyszer van egy oldalon, egyedi

class többször van

<div id=”header”> vagy navigation, stb

több szó kötőjellel

adding CSS: inline styling, <style> tags, stylesheet in head

inline: <a style „color: red”>

header-be <style>

#navigation a{

color: vmi;

}

</style>

stylesheet: headerbe rel=”stylesheet” + type és href

Javascript:

<script> tag, linking js.file, inline

<script src=”scripts/main.js”> </script>

bottom of body

## CSS

most css3

browser styles

collection of rules – specific tag-re vonatkozik

2 parts: selector {declaration; } (font-size: 10px)

#header p, .comment div, li

targeting elements: p {}, a {}

#: id-symbol (overwrites)

.: class names (overwrites)

css cascade conflict: a későbbi érvényes

inheritance

more specific rule wins - id’s > classes > elements (specificity star wars)

benne van a strong: strong nyer

targeting multiple: vesszővel

descendent: nested things

#parent #child grandchild – hard to overwrite

box model: margin, padding, element (border, width, height)

width és height nélkül a tartalomhoz igazodik a függőleges méret, vízszintesen kitölti az oldalt

100%-on túlnyúlhat, hozzáaadja a pixeleket

margin: top és bottom nem adódik össze (vertical margin collapse)

margin valami + auto: középre teszi

% is lehet

padding: 4db. % is lehet

longhand method: pl. margin-top, de a shorthand ajánlott

borders:

border top width-style-coor

margin collapse: lehet h csak aza alján legyen mindig, és akkor nincs para

html5 és css3 most – szabványhivatalok ellenőrzik

**caniuse.com** – html/css/js-feature-ök – megmutatja melyik böngészőn mi működik. ügyfél megmondja milyen böngészőt támogatsz

meta-tag-ek: mellékes dolgok, nem az oldal értelmezi, hanem pl. keresőmotor

article: tartalmilag összefüggő részeket tart össze (szemantikus elem)

strong: kiemelt szöveg

**span**: bármi szövegrészlet amit külön akarunk kezeleni

ugyanez a **div**, csak blokk-szinten. ezek a ***joker***-ek

aside: a main ellentéte, nincs benne fő tartalom. pl. reklám, menü, stb

section: tartalomrészleteket választ el, berakható header, footer, stb

dl-ben dd és dt elemek váltják egymást

id-t ritkán használunk, sose lehet tudni h tényleg 1 lesz abból a dologból

descendant selectors 🡪 space: ilyenekben lévő ilyenek

ne használjuk ki a conflict-ot

## advanced selectors

inline styling: only for very specialized rules (div-be beleírom)

embedded: for specific pages – head-be beírom

external stylesheets: ahogy eddig csináltuk

comments: css-ben. /\* .... \*/ html-ben: <!-- --!>

Important Declaration: can’t be overwritten - space after value !important

child selectors: direct children are moodified, nothing else ’>’ jellel: „#main content > p”

Adjacent Selectors: for an element directly after an element: + sign

Attribute selectors: span[class] -> all spans that have a class

or a[title = search engine] or span[class ~= „deck”] or [href$=”pdf”]

Pseudo Selectors: special keywords

dynamic: hover, button, tick

.class:keyword \_ pl. a:hover {color: red}

pseudo classes: behavioural, structural

selector: keyword {declaration} pl. hover, active, visited

first and last child: article p:first-child{ ue. last-child

first & last of type: article p:first-of-type{ ue. last-of-type

nth child: li:nth-child(x), li:nth-child(y){ vagy nth-child(even) v. odd v. 2n+1

nth of Type Selectors: nth-of-type(1) stb.

combining selectors: article.x -> all articles with class „x” - tojás szerint felülírható. arra jó, h mindent kiherélünk

universal selector: \*{}

## Text styling

font size: absolute, relative (em, %) em: szorzás

font family: x, y, z (ha 1ik nincs) – 2-nél több font ne legyene egy oldalon, lassú

text decoration: pl. text-decoration: none v. inherit, stb.

font weight: pl. bold, normal, bolder, stb

transforming text: change Casing {text-transform}

Text Colour: foreground, background

styling links: a:hover{ color: x}

letter/word spacing & line height: letter-spacing: 10px v. 0.2em

paragraph spacing: margin-bottom: 32px

## Blocks

Block-level Elements: egy folyadék, kitöltöi szélességében a rendelkezésre álló teret (beállítás: display block)

inline elements: egymás mellé mennek, olyan szélesek, mint a tartalmuk. nem lehet a szélességet állítani

inline-block: inline elemnek nem triviális a magassága, ez rá a megoldás. whitespace megjelenik közötte. kódban nem lehet space v. enter: csúnya. inkább olyan blokkra kell, ami olyan széles mint a szöveg

width & height: pl. 70% - jó mobilra

rounded corners: border-radius: 10px. vagy 4 adat a 4 sarokra

Backgrounds: background-color, background-image, background-repeat, background-position (pl. center v. 20% 30% - from left, from top background-size: 200px)

Background Shorthand: background, és a fentiek sorban szóközzel elválasztva, de a color és a size külön alá

Multiple Backgrounds: background-image: url(x), url(y) – top to bottom

background-repeat: no-repeat

**blockquote**: indicates that the enclosed text is an extended quotation – pl. a teljes bekezdés egy idézet

**hgroup**: (*HTML Headings Group Element*) represents the heading of a section. It defines a single title that participates in [the outline of the document](https://developer.mozilla.org/en-US/docs/Sections_and_Outlines_of_an_HTML5_document) as the heading of the implicit or explicit section that it belongs to. – főcímek csoportosítása pl. cikkek

**nav**: represents a section of a page that links to other pages or to parts within the page: a section with navigation links

**q**: indicates that the enclosed text is a short inline quotation. This element is intended for short quotations that don't require paragraph breaks; for long quotations use **blockquote** – a bekezdésnek csak egy része

.alma, .körte – pl. ua. classokat ugyanúgy formázza

.alma.körte – mindkét class-sal rendelkező element

.alma .körte – az a körte ami része az almának

komment: csúnya, nem menő, magyarázkodásnak tűnik, a kód önmagáért beszéljen

inkább arra kell, h megmagyarázzuk ha a logikus megoldás helyett máshogy csináltuk

## box sizing

box-sizing has three possible values (content-box, padding-box, and border-box)

the most popular value is border-box

**Universal Box Sizing with Inheritance**

html {

box-sizing: border-box;

}

\*, \*:before, \*:after {

box-sizing: inherit;

}

## Positioning

box model: blocks, inline

normal doc flow: block level elements

floating: image {float: left}

Clearing Floats: clear: left/right – de így nem lehet margin-t rakni az alatta lévő szövegre

kell egy plusz div a floatotlt blokkok és az alatta lévő szöveg közé

de jobb, ha van egy „:after” pseudo class, ami üres (content:””) clear: both

content columns: section, aside, float:left, width 46%, megint jön az after

text columns w/o floats:

-webkit-column-count: x+ webkit column gap

-webkit-column-rule: 1px solid blue

text-align: justify

relative position: „position: relative” „left: 50px”, top, bottom, stb – csak kis kiigazításokra

absolute position: „position: absolute” „top: 0” „left: 0”, vmihez képest pl. kép is lehet abszolút (üveglap-effektus)

fixed positioning: „position: fixed” mindig ott marad pl. a tetején egy csík, pl. egy menüsor

z-index & stacking order: minél lejjebb van, az lesz a felső réteg

default z-index nulla ---- z-index: 10

clipping content: max-height-et ad, túllógnak a dobozok.. „overflow: auto” v. hidden v. scroll

Float's sister property is **clear**.

the footer can be cleared to ensure it stays beneath both floated columns.

Both is most commonly used, which clears floats coming from either direction. Left and Right can be used to only clear the float from one direction respectively.

If the parent element contained nothing but floated elements, the height of it would literally collapse to nothing.

We fix it by clearing the float after the floated elements in the container but before the close of the container.

**Techniques for Clearing Floats**

clear: both; value to the next element

*Empty Div Method* is, quite literally, an empty div. <div style="clear: both;"></div>.

*easy clearing:*

you apply an additional class like "clearfix" to it. Then apply this CSS:

.clearfix:after {

content: ".";

visibility: hidden;

display: block;

height: 0;

clear: both;

## Flexbox

sokkal jobb mint az előző fejezetben lévők

flex containers: pl. div class=”flex-container”

display: flex

flex grow: flex-grow: 1

flex shrink: ha csökken a terület, lecsökkennek

flex wrap: min. width miatt ne ne legyen scroll, ha csökkentem a területet. köv. sorba kerül (flex-wrap: wrap) lehet reverse is, akkor feljebb megy

flex basis: starting width (like min-width)csak jobb

flex: 1 0 200 px -🡪 growth, shrink, basis

Creating a Menu: display. flex (volt még justify contenttel vmi)

Creating Nested Menu's: 2 menüsor külön kezelve. fb- és twitter ikonokat rakott

flow & axis: alapból vízszintes. „flex-flow” paranccsal módosítható (row helyett column)

(ha kivesszük a flow-ból, körbefollya a szöveg, magasság eltaartás megszűnik)

módosul a main és cross axis, helyet cserélnek (main a vízszintes alapból)

justify content csak a main axis-re vonatkozik, ha column-ra állítjuk nem működik

van row reverse is, meg column reverse

align items: cross axis-re vonatkozik

grid & stack layout: van egy ilyen, h box-sizing: border-box, itt justify-content: space between van

csinált egy transitiont

element order: .one{order: x}

*Flexbox a cikkből*

* **main axis** - The main axis of a flex container is the primary axis along which flex items are laid out. Beware, it is not necessarily horizontal; it depends on the flex-direction property (see below).
* **main-start | main-end** - The flex items are placed within the container starting from main-start and going to main-end.
* **main size** - A flex item's width or height, whichever is in the main dimension, is the item's main size. The flex item's main size property is either the ‘width’ or ‘height’ property, whichever is in the main dimension.
* **cross axis** - The axis perpendicular to the main axis is called the cross axis. Its direction depends on the main axis direction.
* **cross-start | cross-end** - Flex lines are filled with items and placed into the container starting on the cross-start side of the flex container and going toward the cross-end side.
* **cross size** - The width or height of a flex item, whichever is in the cross dimension, is the item's cross size. The cross size property is whichever of ‘width’ or ‘height’ that is in the cross dimension.

.container {

display: flex; /\* or inline-flex \*/

}

.container {

flex-direction: row | row-reverse | column | column-reverse;

}

Think of flex items as primarily laying out either in horizontal rows or vertical columns.

**flex-wrap:**

By default, flex items will all try to fit onto one line. You can change that and allow the items to wrap as needed with this property. Direction also plays a role here, determining the direction new lines are stacked in.

.container{

flex-wrap: nowrap | wrap | wrap-reverse;

}

**flex-basis**

the default size of an element before the remaining space is distributed.

It can be a length (e.g. 20%, 5rem, etc.) or a keyword

.item {

flex-basis: <length> | auto; /\* default auto \*/

}

flex-start (default): items are packed toward the start line

flex-end: items are packed toward to end line

center: items are centered along the line

space-between: items are evenly distributed in the line; first item is on the start line, last item on the end line

space-around: items are evenly distributed in the line with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies.

**align-self:** This allows the default alignment (or the one specified by align-items) to be overridden for individual flex items.

perfect centering

.parent {

display: flex;

height: 300px; /\* Or whatever \*/

}

.child {

width: 100px; /\* Or whatever \*/

height: 100px; /\* Or whatever \*/

margin: auto; /\* Magic! \*/

}

a list of 6 items, all with a fixed dimensions in a matter of aesthetics but they could be auto-sized. We want them to be evenly and nicely distributed on the horizontal axis so that when we resize the browser, everything is fine

.flex-container {

display: flex;

flex-flow: row wrap;

justify-content: space-around;

}

div ~ p: ezután következő összes

border box: méret

display: none – majd javascript-nél lesz jó

block típusú elem mellé nem fér semmi, hiába 50% a szélessége. kivéve ha float. de körülfolyja. erre jó a clear. leraksz egy vonalzót, csak odáig pöndörödjön fel a papír.

after: nem a kacsa után jön, hanem a kacsán belül minden elem után

position: static 🡪 semmi, ez a default

float a jövőben csak arra lesz, h körülfolyjon vmit a szöveg

## Form

HTML Forms are one of the main points of interaction between a user and a web site or application.

made of one or more widgets (select boxes, buttons, checkboxes, or radio buttons)

ask only for what you absolutely need

following HTML elements: <form>, <label>, <input>, <textarea>, and <button>.

**form:**

<form action="/my-handling-form-page" method="post">

</form>

always set at least the action attribute and the method attribute.

* The action attribute defines the location (an URL) where the form's collected data should be sent.
* The method attribute defines which HTTP method to send the data with (it can be "get" or "post").

**Add widgets with the <label>, <input>, and <textarea> elements**

<form action="/my-handling-form-page" method="post">

<div>

<label for="name">Name:</label>

<input type="text" id="name" />

</div>

<div>

<label for="mail">E-mail:</label>

<input type="email" id="mail" />

</div>

<div>

<label for="msg">Message:</label>

<textarea id="msg"></textarea>

</div>

</form>

for attribute on all [<label>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/label)elements; it's a formal way to link a label to a form widget

On the [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) element, the most important attribute is the type attribute

<input> tag is an auto-closing element

To define the default value of an [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) element you have to use the value attribute like this:

<input type="text" value="by default this element is filled with this text" />

On the contrary, if you want to define the default value of a [<textarea>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/textarea), you just have to put that default value between the starting and ending tag of the[<textarea>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/textarea) element, like this:

<textarea>by default this element is filled with this text</textarea>

And a [<button>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button) to finish

<div class="button">

<button type="submit">Send your message</button>

</div>

Three types: submit, reset, or button.

aztán formázás!!

to send the data We also need to give a name to our data

se the name attribute on each form widget that will collect a specific piece of data

**Overview of HTML5 Forms Types, Attributes, and Elements**

<input type=email> vagy tel, url, date, color, range

<input placeholder=”hi guys” type= „text”> hints as to what info is useful

autofocus?

maxlength

min, max, step: <input type=number min=0 max=100 step=5>

datalist: like a dropdown menu

required fields

pattern = x y

no validate, form no validate

spellcheck

meter

progress

output

keygen

**web-form-usability**

Forms can make a website usable

Forms need to be usable

Six Components Of Web Forms:

* Labels: what the corresponding input fields mean
* Input Fields: enable users to provide feedback
* Actions: links or buttons that perform an action
* Help
* Messages: give feedback to the user based on their input
* Validation: ensure that the data submitted by the user conforms to acceptable parameters

all forms have three main aspects: Relationship, Conversation, Appearance:

**Aspect 1: The Relationship**: based on trust, Every relationship has a goal, Base the name of the form on its purpose, getting to know the other person is essential, choose appropriate language and remove superfluous text, Do not ask questions beyond the scope of the form, no sudden changes in behavior or appearance

**Aspect 2: The Conversation**: a conversation, not an interrogation, Order the labels logically, Group related information, such as personal details, address one topic at a time, natural pauses in a conversation will indicate where to introduce white space, remove clutter such as banners and unnecessary navigation

**Aspect 3: The Appearance:**

*labels*: Individual words vs. sentences, Sentence case vs. title case, Colons at the end of labels, Alignment of labels: top vs. left vs. right,

*input fields*: Provide the appropriate type of input field, not too much Restricting of format, Clearly distinguish mandatory vs. optional fields

*actions*: Primary vs. secondary, Naming conventions (not submit but join xy pl.)

*help*: Text to accompany forms (only where needed, such as to explain why credit card data is being requested), User-triggered and dynamic help (only where required)

*messages*: Error message, Success message

*validation*: Only where needed (such as the availability of a user name), Smart defaults to make the user’s completion of the form faster and more accurate

label-nek kell ID, plusz name is kell, ha szervernek felküldjük

## CSS colors

CSS Color: #hex 0: darkest

rgb(x,y,z) 0-255

opacity: 0 to 1 (transparent to solid), 3 rgb mögé is megadható

gradients: jó mellé egy background ha a böngésző nem támogatja

background: linear-gradient(top, color1 0%, color2 100%)

vendor prefix: -moz-, -webkit-(chrome)

reverse-elte hoverelésnél

colorzilla.com/gradient-editor

box shadow: 2px 2px 4px 4px rgb(5,5,5, 0.6) - (right, bottom, blurred, spread, shadow color)

**outline:**

draws a line around the outside of an element - you can't specify particular sides, won't effect the position of the element

it isn't always rectangular

pl. to emphasize a link when tabbed to without affecting positioning and in a differnet way than hover

a:active {

outline: 1px dashed red;

}

shorthand: outline: [ <outline-width> || <outline-style> || <outline-color> ]

long:

a:active {

outline-width: 1px;

outline-style: dashed;

outline-color: red;

}

## ****Table****

(<table>)

permitted content:

1. an optional [<caption>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/caption) element,
2. zero or more [<colgroup>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/colgroup) elements,
3. an optional [<thead>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/thead) element,
4. one of the two alternatives:
   * one [<tfoot>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/tfoot) element, followed by:
     + either zero or more [<tbody>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/tbody) elements,
     + or one or more [<tr>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/tr) elements,
   * a second alternative followed by an optional [<tfoot>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/tfoot)element:
     + either zero or more [<tbody>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/tbody) elements,
     + or one or more [<tr>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/tr) elements

Attributes:

*align*

*bgcolor*

*border*

*cellpadding*

*cellspacing*

*frame*

*rules*

*summary*

*width*

**tr-table row**

**th-table header**

**tc-table column**

**td-table data?**

## responsive design

mediaquery: lefut ha teljesül

@media screen and (max-width: 700px) {

width: 92%;

}

ebbe bele lehet még tenni selectorokat

ha a szélesség 700 alá csökken akkor lesz érvényes (break point)

tablet pl. kb 768px. mobil 480 px

fluid layout: %-okkal adom meg

@media – pl. width x pixel – váltja a színeket

## transform:

translateX – balra mozgat Y-lefelé sima-mind2

scaleX Y mind2

rotate x-nél 3d z-2d

## transitions

2 állapot, pl. hovernél változik

eredeti elembe: transition time, delay és p. linear

bezier görbék

minden más a másik elembe

különböző jellemzőkhöz lehet más időtartam

## How To Organize CSS Files

* 1. Around Your HTML Structure
  2. Around Aspects Of Design

top down approach makes it easier to work on a specific section of the site

### BEM

Block, Element, Modifier methodology

*block* is a top-level abstraction of a new component, for example a button: .btn { }. This block should be thought of as a parent.

*elements*, can be placed inside and these are denoted by two underscores following the name of the block like .btn\_\_price { }

*modifiers* can manipulate the block so that we can theme or style that particular component. This is done by appending two hyphens to the name of the block just like btn--orange.

<a class="btn btn--big btn--orange" href="http://css-tricks.com">

<span class="btn\_\_price">$9.99</span>

<span class="btn\_\_text">Subscribe</span>

</a>

## meta tags:

metadata: important is the ability to encourage your content to make the best first impression possible

title and meta description are still the most important meta tags

Page Title – make page titles as keyword-relevant as possible and up to 70 characters

Meta Description - If your webpage were a commercial, this would be its slogan - 155 characters

Authorship Markup – Google is going to rank content that is connected to authors that they deem to be reliable sources

Social Meta Tags – Facebook’s OpenGraph allows you to specify metadata to optimize how your content appears in a user’s timeline, Twitter Cards, Schema.Org (for Google+)

duplicate content issues?

# Python

## basics

most a python 3 van

print „”

boolean: True False

indentation error – 4 spaces

# and „”” „””: comments

\*\* = ^

% = maradékos osztás maradéka

maradékos osztás: //

reassign: meal = meal + tax

strings

’ 🡪 \’

r 🡪 put before strings to print it out as it is

c = "cats"[0]

String methods: len() lower() upper() str() turns non-strings into strings

*slice*: user = „Tuna McFish” - User[2:7] = „na Mc”

Dot Notation: only works with strings. - print ministry.upper()

**concatenation:** print "Life " + "of " + "Brian" – prints without spaces

print "The value of pi is around " + str(3.14)

String Formatting with %:

**string\_1 = "Camelot"**

**string\_2 = "place"**

**print "Let's not go to %s. 'Tis a silly %s." % (string\_1, string\_2)**

comparators

booelan operators: and - or – not

sorrend: not and or

**if** is a conditional statement that executes some specified code after checking if its expression is True.

def, if, else után „:”!!

elif: „else if”

float: tört szám

command line: py –i – kiírja a python verziómat, belemegy a python-ba, mehet a kérdezz felelek

exit()-tel kilép

futtatás: belemegyek a megfelelő könyvtárba, és beírom, hogy py\_filenév

nem mindig pontos!!!

kapunk error-t ha vmit elbaszunk (pl.nullával osztás)

1e3 = 1000

nem lehet space-t rakni a számok közé

lehet zárójelezni

ha hiba van, csak az elsőért kiabál, mert ott leáll

változó: valami = 3

betűvel kell kezdődnie, nem lehet benne space, nagy és kisbetűt megkülönbözteti

van python styleguide

valami változó növekedjen 1-gyel: +=1 vagy lehet /= stb.

comment: # valami

jön a két egyenlőségjel, nagyon vigyázni kell, keveredhet a szimpla és a dupla

kasztolás: ha betűt számokká alakítunk pl. int(True)=1

type() – megmondja hogy milyen type

string: str – összeadás-összefűzés

kivonni nem lehet, stringhez számot adni nem lehet, a számot először stringgé kell alakítani

stringet be lehet szorozni számmal

benne van –e vmi: „a” in „nagyarpi” = True

len() – milyen hosszú

ki lehet szedni egyes betűket: funny\_word[5] – 6. betű

[2:4] – 2. és a 3.

[2:4:2] – 2. és 3. kettesével. ha a step negatív, visszafelé számol

lista: []

fruits = [„alma”, „korte”, „banon”]

fruits[2]=”banon”

van üres lista is

listát lehet concatenálni

szorzható, akkor ismétlődik az egész

itt is lehet „in” parancs

len: itt az elemek száma

össze is lehet hasonlítani őket

át is lehet írni a lista elemeit = jellel

törölni is lehet, ha vmelyik elem = []

**None**: semmit se tud, kiértékelődik és nem csinál semmit

pass: lefut, de nem csinál semmit

## data structures

*http://pymbook.readthedocs.io/en/latest/datastructure.html*

**lists**

append: beteszi a végére

insert: beteszi vhányadiknak - a.insert(0, 111) – beteszi a 111-et a 0. helyen

count: megszámolja az elemeket

remove: kiveszi

reverse

extend: beteszi elemenként a hozzáadott listát

stack LIFO – a.pop() kieszi az utolasót a. pop(i)-kiveszi az i-edik elemet

queue FIFO – a.pop(0)

List comprehensions: Each list comprehension consists of an expression followed by a for clause, then zero or more for or if clauses.

**>>>** a = [1, 2, 3]

**>>>** [x \*\* 2 **for** x **in** a]

[1, 4, 9]

tuples: olyan minta lista, csak sima zárójelek közé. de nem lehet módosítani. (véletlen változtatgatások ellen lehet jó)

sets: another type of data structure with no duplicate items . nincs duplikáció

set operations:

**>>>** a = set('abracadabra')

**>>>** b = set('alacazam')

**>>>** a *# unique letters in a*

{'a', 'r', 'b', 'c', 'd'}

**>>>** a - b *# letters in a but not in b*

{'r', 'd', 'b'}

**>>>** a | b *# letters in either a or b*

{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

**>>>** a & b *# letters in both a and b*

{'a', 'c'}

**>>>** a ^ b *# letters in a or b but not both*

{'r', 'd', 'b', 'm', 'z', 'l'}

a.add(’p’)

dictionaries

add data: data['key'] = 'value'

If you want to loop through a dict use items() method.

**for** x, y **in** data.items():

**...**  **print**("*%s* uses *%s*" % (x, y))

refaktorolás: rövidítjük a kódot, de uazt csinálja. minden mentés után érdemes – utána teszt futtatás

while: addíg amíg

for: ahányszor megadjuk annyiszor fut

break: kiugrik

continue: léptetés után célszerű használni, különben inf loop lesz. szóval kimegy az elejére az a lényeg

## functions:

def x():

call it: type name

variable a zárójelben

storing: return

adunk egy változót, és azt printeljük

default values: fv(x=”y”), ha nincs input ezt adja ki

variable scope: ha a variable benne van a fv-ben, csak az fér hozzá

keyword arguments: sorban veszi az inputokat. ha kevesebb van vagy más sorrendet akarunk. az x-szel együtt adjuk meg

flexible number of variables in a function: (\* args)

unpacking args: \*lista neve

dictionary: x = {„key”:”value”, „key”:”value”, stb.}

print x(item)

vagy for k, v in x.items():

print (k+v)

def test(expected, actual):

if expected == actual:

print(„check”)

else:

print(„jaj”)

## classes & objects:

class Capital:

indented dolgok a részei lesznek

self: self.variable – how you access variables

to access a class: object = class

object.def()

init: def\_\_init\_\_(self): ezt mindig behozza ha a class-ből behívsz vmi programot

class vs instance variable: mindegyiknél ugyanaz vs mindegyiknél egyedi

inheritance: class X(y) inherits from y

overwrite: child can overwrite inherited function

**Object Orientation**

Python is an object-oriented language – methods: operations that can be done on some particular type of object

pl. strings-nek az upper

syntax: object.method( ) v. object.method(parameters)

**>>>** tale = 'This is the best of times.'

**>>>** tale.count('i')

the dot between the object and the method name is an operator

***codeacademy:***

class is just a way of organizing and producing objects with similar attributes and methods

method: function of objects

**dictionary**: vminek a jellemzőit tárolja

kulcs-érték párok, kulccsal lehet keresni az értékeket

máshol map-nek hívják

nemcsak a key beírásával szedhetünk ki értéket, hanem változóval is (aminek az értéke a key), így pl.

for-ral ki lehet venni az összeset

variables: global, member, instance

variable scope: melyik fv-ben látjuk az adott változót. van globális, mindenhol látszik. csak kívülről befelé megy, bentről kifelé nem

global scope > function scope

bent felül lehet írni, de attól kint még változatlan marad

globálist kb sose használunk

inheritance: class DerivedClass(BaseClass):

ha felülírtam, de az eredeti kell: super!

attribute: variable a class-ben

objektum: az osztályon belül egy tag. pl. kutya osztályon belül gömbi egy objektum

konstruktor: meghívódik, 1. paraméter

## modules

modules: files with Python definitions and statements

\_\_name\_\_ variable (global)

**import** (filenév)

benne lévő függvények elérése: **filenév.fv-név(arg)**

lehet direktben importálni fv-eket is: from bars import simplebar, starbar

submodules

A directory with a \_\_init\_\_.py can also be used as a module and all .py files inside it become submodules.

You can create an empty \_\_init\_\_.py using touch command.

$ touch mymodule/\_\_init\_\_.py

If *\_\_init\_\_.py* file contains a list called *\_\_all\_\_*, then only the names listed there will be public. So if the mymodule’s *\_\_init\_\_.py* file contains the following

**from** mymodule.bars import simplebar

\_\_all\_\_ = [simplebar, ]

Then from mymodule only simplebar will be available.

Default modules:

in python 3.5: To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics".

help> modules

Module **os**: provides operating system dependent functionality

etuid() function returns the current process’s effective user’s id.

uname() returns different information identifying the operating system (sysname, nodename, release, version, machine)

*getcwd()\*returns the current working directory. \*chdir(path)* changes the current working directory to path.

Requests Module: helps you to do HTTP GET or POST calls

You can use the get method to fetch any website:

>>> import requests

>>> req = requests.get('http://google.com')

>>> req.status\_code  
a command which can download a given file :

**import** **os**

**import** **os.path**

**import** **requests**

**def** download(url):

*'''Download the given url and saves it to the current directory.*

*:arg url: URL of the file to be downloaded.*

*'''*

req = requests.get(url)

*# First let us check non existing files.*

**if** req.status\_code == 404:

**print**('No such file found at *%s*' % url)

**return**

filename = url.split('/')[-1]

**with** open(filename, 'wb') **as** fobj:

fobj.write(req.content)

**print**("Download over.")

**if** \_\_name\_\_ == '\_\_main\_\_':

url = input('Enter a URL:')

download(url)

(when the module name is \_\_main\_\_, then only ask for a user input and then download the given URL.)

## file handlilng

we divide files in two categories, text file and binary file

File opening: open() - It requires two arguments, first the file path or file name, second which mode it should open. Modes:

“r” -> open read only, you can read the file but can not edit / delete anything inside

“w” -> open with write power, means if the file exists then delete all content and open it to write

“a” -> open in append mode

After opening a file one should always close the opened file. We use method close() for this.

To read the whole file at once use the read()

readline() can help you to read one line each time from the file.

You can loop through the lines:

for x in f:

print(x, end=' ')

Writing in a file: write()

copyfile.py: copy a given text file to another file

sys module: contains all command line parameters

enumerate(iterableobject): returns the index number and the value from the iterable object

Count spaces, tabs and new lines

with statement: will take care of closing the file

**videó: How to read & write:**

fw = open(’sample.txt’, ’w’)

fw.write(’kjkj’)

fw.close()

\n – new line

fr = open(’sample.txt’, ’r’)

**Regex**: python-ban „import re” – pl. email cím formátuma megfeleő –e egy formban, vagy telszám

**rstrip** – leszedi a new line-t a sor végéről

## tkinter

**GUI with Tkinter**

rfom tkinter import \*

root (*vagy bármilyen változó*) = Tk # creates blank window

theLabel = Label(root, „text vmi”)

theLabel.pack() – elhelyezés

mainloop – x ideig ott lesz

**Shapes and Graphics**

canvas = Canvas(root, width = x, height = y)

canvas.pack()

alapból minden fekete

blackLine = canvas.create\_line(x-starting point, y-starting point, x-ending, y-ending, fill = z)

**canvas.delete(blackLine) v. (ALL)**

tehát:

w = Canvas ( master, option=value, ... )

|  |  |
| --- | --- |
| **Option** | **Description** |
| bd | Border width in pixels. Default is 2. |
| bg | Normal background color. |
| confine | If true (the default), the canvas cannot be scrolled outside of the scrollregion. |
| cursor | Cursor used in the canvas like *arrow, circle, dot etc.* |
| height | Size of the canvas in the Y dimension. |
| highlightcolor | Color shown in the focus highlight. |
| relief | Relief specifies the type of the border. Some of the values are SUNKEN, RAISED, GROOVE, and RIDGE. |
| scrollregion | A tuple (w, n, e, s) that defines over how large an area the canvas can be scrolled, where w is the left side, n the top, e the right side, and s the bottom. |
| width | Size of the canvas in the X dimension. |
| xscrollincrement | If you set this option to some positive dimension, the canvas can be positioned only on multiples of that distance, and the value will be used for scrolling by scrolling units, such as when the user clicks on the arrows at the ends of a scrollbar. |
| xscrollcommand | If the canvas is scrollable, this attribute should be the .set() method of the horizontal scrollbar. |
| yscrollincrement | Works like xscrollincrement, but governs vertical movement. |
| yscrollcommand | If the canvas is scrollable, this attribute should be the .set() method of the vertical scrollbar. |

standard items:

**arc .** Creates an arc item, which can be a chord, a pieslice or a simple arc.

coord = 10, 50, 240, 210

arc = canvas.create\_arc(coord, start=0, extent=150, fill="blue")

**image .** Creates an image item, which can be an instance of either the BitmapImage or the PhotoImage classes.

filename = PhotoImage(file = "sunshine.gif")

image = canvas.create\_image(50, 50, anchor=NE, image=filename)

**line .** Creates a line item.

line = canvas.create\_line(x0, y0, x1, y1, ..., xn, yn, options)

**oval .** Creates a circle or an ellipse at the given coordinates. It takes two pairs of coordinates; the top left and bottom right corners of the bounding rectangle for the oval.

oval = canvas.create\_oval(x0, y0, x1, y1, options)

**polygon .**Creates a polygon item that must have at least three vertices.

oval = canvas.create\_polygon(x0, y0, x1, y1,...xn, yn, options)

cheatsheet: <http://www.python-course.eu/tkinter_canvas.php>

## recursion

**Factorial**

number = eval(input(„gimme a number”))

for i in range(number):

number = number \* (1 + i)

recursive: referring to itself

def factorial(number):

if number <= 1:

return 1

else:

return number \* factorial(number - 1)

**fibonacci:**

if n < 2 return n

else: return fib(n-1) + fib(n-2)

**fractal:**

koch snowflake: 3 \* 4n sides

infinite perimeter, finite area

fractal antenna can pick up more signals & take up less space

menger sponge – 3d

**Factorial with loop**

def factorial\_iterative(number):

product = 1

for i in range(number):

product \*= i+1

return product

print('5! is', factorial\_iterative(5))

**Factorial with recursion**

def factorial(number):

if number <= 1: #base case

return 1

else:

return number \* factorial(number-1)

print('5! is', factorial(5))

fraktál: többször is meghívhatja magát

segédfüggvényt írt, utána már csak azt hívta meg a fraktálosban

canvas.create\_polygon(x0, y0, x1, y1, ...., fill=’green’, outline=’black’

## unit testing

Unit testing, specifically tests a single "unit" of code **in isolation**. A unit could be an entire module, a single class or function, or almost anything in between

we may write test code in a file named test\_primes.py

**import** **unittest**

**from** **primes** **import** is\_prime

**class** **PrimesTestCase**(unittest.TestCase):

*"""Tests for `primes.py`."""*

**def** test\_is\_five\_prime(self):

*"""Is five successfully determined to be prime?"""*

self.assertTrue(is\_prime(5))

**if** \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

if it was successful, a "." would have been printed

helpful when refactoring code

Good testing requires modular, decoupled code

A unit test consists of one or more assertions (statements that assert that some property of the code being tested is true).

**self.assertTrue** asserts that the argument passed to it evaluates to True

method names should explicitly tell what is being asserted by the test

Each test should test a single, specific property of the code

"OK" is only reported when no tests actually ran!

changing the line in is\_prime to for element in range(2, number):

Edge Case: unusual or unexpected input

the Python unittest framework does not print out the expected and actual values

Third-Party Test Frameworks: py.test and nose

...

unittest módszerek pl.: assertTrue, assertEqual, assertRaises

„.” test ran successful

F – failure

**general rules of testing**

Each test unit must be fully independent. **setUp()** and **tearDown()** methods.

make tests that run fast

Always run the full test suite before a coding session, and run it again after. This will give you more confidence that you did not break anything in the rest of the code.

implement a hook that runs all tests before pushing code to a shared repository.

The first step when you are debugging your code is to write a new test pinpointing the bug.

**unittest:**

**import** unittest

**def** fun**(**x**):**

**return** x + 1

**class** MyTest**(**unittest.TestCase**):**

**def** test**(**self**):**

self.assertEqual**(**fun**(**3**),** 4**)**

doctest:

searches for pieces of text that look like interactive Python sessions in docstrings, and then executes those sessions to verify that they work

**def** square**(**x**):**

*"""Return the square of x.*

*>>> square(2)- 4*

*>>> square(-2) - 4 """*

**return** x \* x

**if** \_\_name\_\_ == '\_\_main\_\_'**:**

**import** doctest

doctest.testmod**()**

**Tools**

py.test, mock, nose, unitest2, tox

...

**Test-driven development (TDD)**

The process can be defined as such:

Write a failing unit test

Make the unit test pass

Refactor

Agile develeopment-tel összeillik

először megírjuk a teszteket, aztán a fejlesztést

The main methods that we make use of in unit testing for Python are:

* assert: base assert allowing you to write your own assertions
* assertEqual(a, b): check a and b are equal
* assertNotEqual(a, b): check a and b are not equal
* assertIn(a, b): check that a is in the item b
* assertNotIn(a, b): check that a is not in the item b
* assertFalse(a): check that the value of a is False
* assertTrue(a): check the value of a is True
* assertIsInstance(a, TYPE): check that a is of type "TYPE"
* assertRaises(ERROR, a, args): check that when a is called with args that it raises ERROR

**Nose**

## exceptions

not a syntax error, a value error

példa:

while True:

try:

kód

except ValueError:

print(„vmi”)

vagy ZeroDivisionError

vagy simán except: - de akkor eltakarhat problémákat

finally:

execute no matter what

...

Any error which happens during the execution of the code is an exception.

NameError: someone tries to access a variable which is not defined

TypeError: incompatible data types

try:

statements to be inside try clause

statement2

statement3

...

except ExceptionName:

statements to evaluated in case of ExceptionName happens

One can raise an exception using *raise* statement.

If we want to have some statements which must be executed under all circumstances, we can use finally clause, it will be always executed before finishing try statements.

nem feltétlenül error-ra használjuk az exception-t.

raise: pl. eltalának egy számot, és vége a programnak

try-on kívüli exception. leáll a program!

if number == 2

raise ValueError(’ketto’)

print(ize)

**JSON**: javascriptr object notation, xml kiváltására

strukturális alappillérek a nyelvekben: dictionary, list

nehezebb különleges adatok tárolása

de gyorsabb

## command line arguments

getopt module

sys module provides access to any command-line arguments via the sys.argv:

* sys.argv is the list of command-line arguments.
* len(sys.argv) is the number of command-line arguments.

getopt.getopt method: parses command line options and parameter list

## csv

1. plain text
2. 1 rekord egy soron
3. vesszővel, pontosvesszővel van elválasztva
4. every record has the same sequence

SAP mondjuk ezt adja át vmi másik rendszernek

meg lehet jeleníteni táblázatos formában is

python: import.csv

for ciklussal végigmegy soronként

<https://docs.python.org/2/library/csv.html#writer-objects>

<http://www.tutorialspoint.com/python/python_command_line_arguments.htm>

python cuccok még: map, lambda, filter, list comprehension

# Javascript

## basics

<body> -ban egy <script>

benne: alert(„valami”);

small script esetén body aljára kell tenni, különben nem írja ki a többit

vagy elmentjük egy js-file-ba és lehivatkozzuk a html-ben: <script src=”vmi.js”></script>

Google Chrome Developer Tools: console lap

case sensitive

always end statements with’;’

whitespace not sensitive

comments: /\* \*/ vagy // (one-line)

fentről lefelé megy

## variables:

to store information

nem kezdődhet számmal

var myVariable = 40 vagy „sdfsd2”, stb.

5 + „hello” = „5hello”

nem lehet szorozni egy string-et

NaN = not a number

++: egy shorthand

document.write(változó) kiírja az oldalra

console.log(változó) kiírja a konzolra

boolean: kisbetűvel

**if statements:**

if + feltétel + {

vmi;

}

else {

}

else if { }

**comparison operators:**

=== : value and type ellenőrzése

negation: ’!’

**logical operators:**

&& : and

|| : or

**while**: a teendő után variable++ h ne legyen infinite loop

**for**: for (age = 5; age < 10; age++) { }

break – kijön a loop-ból

continue – erre a körre nem csinálja meg az utasítást

Math object: pl. Math.round(7.8) v. floor(lefelé kerekít), ceil felfelé, van még max

**NaN** – not a number

if(isNaN(a)) – ha ’a’ nem egy szám

**strings:**

„ ’ probléma megoldható \ – vel

myString.length

.toUpperCase() .toLowerCase, .indexOf(„vmi”) – hányadik helyen van a vmi – ha nincs benne akkor -1 az eredmény

**slice & split:**

str.slice(x, y) – x.-től y.- betűig

tags.split(„,”)

**arrays**

myArray = [x, y, z]

vagy = new Array(5) (5 slots)

van hozzá .length, .sort, .reverse

**Objects**

creating

var myArray = new Array (); belerak 2 elemet

majd

var myCar = new Object();

myCar.maxSpeed = 50; stb.

myCar.drive = function() { vmi }

meghívás

myCar.drive()

shorthand:

var myCar2 = {maxSpeed = 70, drive: function() { vmi}}

**THIS**

ha beírom a fv-be, pl. hogy test: function() { console.log(this); }

aztán meghívbom – myCar2.test() – kiírja az object mutatóit

it’s a timesaver

utána rakhatunk mást pl. ’console.log(this.driver)’

**Constructor functions:**

creates an object

var myArray = new Array();

így megy:

var Car = function(maxSpeed, stb); {

this.maxSpeed = maxSpeed;

stb

}

aztán lehet vele új objecteket csinálni::

var myCar = new Car(50; „ninja”);

és egy csomó myCar

**Date object**

var myDate = new Date();

.getFullYear, .getDay, stb

.getTime – milliseconds since 1970/1/1

## DOM

document object model

ineract with html

every html element is an object

Node: everything we can change in a document (elements, text, attributes)

**Traversing the DOM**

reach a node, and then traverse from there

document.getElementsByClassName(„content”)

kiszedi a „content” class nevű elemeket

elmentjük egy változóba (pl. myContentDivs)

ezekből a h2-t:

var myH2 = myContentDivs[1].getElement**s**ByTagline(„h2”);

get elements by ID: .getElementById(„vmi”)

**Changing Page Contents**

.innerHTML – kiadja az egész html-t az adott tag-ben vagy bármiben (formátummal együtt)

lehet cserélni vele html-tageket is

vagy

.textContent

megváltoztatás: „= vmi más”

**Changing Attributes**

.getAttribute(„vmi pl. class”)

.setAttribute(„vmi pl. class”, „mire módosítsa”) – módosítja

.className – kihozza a class nevet

**changing styles**

title.setAttribute(„style”, „position: relative;”);

vagy „left: 10px;”

aztán

title.style.left = 20px; vagy top, color, backgroundColor, stb

**adding elements to DOM**

create in JS, push it into HTML

var newLi = document.createElement(„li”)

var newA = document.createElement(„a”)

kiválasztunk egy ul-t:

var menu = document.GetElementById(„vmi”).getElementsByTagName(„ul”)[0];

aztán:

menu.appendChild(newLi);

newLi.appendChild(newA);

newA.innerHTML = „Blog”; - beírja h blog

vagy

menu.insertBefore(newLi, menu.getElementsByTagName(„li”)[0]); -2. tag h mi elé rakja

**Removing Elements**

kiválasztunk vmit, egy parentet meg egy child-ot

aztán: parent.removeChild(child);

változóba is rakható:

var removed = parent.removeChild(child);

visszatehető:

parent.appendChild(removed)

**Events:**

kiválasztjuk a title-t

title.onclick (vagy on-akármi)= function () {

alert(„valami”);

};

**Onclick event**

van egy show more-gomb

kiválasztjuk content-et és button-t a html-ből

if-fel írjuk (vagy nyitva van, vagy csukva)

button.onclick = function () {

if(content.className == „open”) {

content.className = „”; *// összecsukja*

button.innerHTML = „Show more”;

} else {

content.className = „open”; *// kinyitja*

button.innerHTML = „Show less”;

};

};

**Window onload event**

html-ben a hivatkozást tartalmazó script a végén van

ehelyett lehet:

window.onload = function() {}; és mindent berak ebbe a function-be

vagy:

úgy kezdi a js-t, hogy function setUpEvents() {

ebbe rak mindent

és aztán

window.onload = function() {

setUpEvents(); };

**Timers**

meghatározott idő után történik vmi

kiválasztunk egy message-t

function showMessage () {

myMessage.className = „show”;

}

setTimeout(showmessage, 3000); *//milliseconds-ben , egyszer jelenik meg*

többször:

setInterval(changeColour, 3000);

clearInterval(változó neve, amibe elmentettük a setInterval-t) – vége

JavaScript can manipulate the DOM

The Document Object Model (DOM) is a programming interface for HTML and XML documents.

DOM provides a representation of the document as a structured group of nodes and objects that have properties and methods.

**Important Data Types**

document, element, nodeList, attribute, namedNodeMap

**DOM interfaces**

The following is a brief list of common APIs in web and XML page scripting using the DOM.

* [document.getElementById](https://developer.mozilla.org/en-US/docs/DOM/document.getElementById)(id)
* document.[getElementsByTagName](https://developer.mozilla.org/en-US/docs/Web/API/Element.getElementsByTagName)(name)
* [document.createElement](https://developer.mozilla.org/en-US/docs/DOM/document.createElement)(name)
* parentNode.[appendChild](https://developer.mozilla.org/en-US/docs/DOM/Node.appendChild)(node)
* element.[innerHTML](https://developer.mozilla.org/en-US/docs/DOM/element.innerHTML)
* element.[style](https://developer.mozilla.org/en-US/docs/DOM/element.style).left
* element.[setAttribute](https://developer.mozilla.org/en-US/docs/DOM/element.setAttribute)
* element.[getAttribute](https://developer.mozilla.org/en-US/docs/DOM/element.getAttribute)
* element.[addEventListener](https://developer.mozilla.org/en-US/docs/DOM/element.addEventListener)
* [window.content](https://developer.mozilla.org/en-US/docs/DOM/window.content)
* [window.onload](https://developer.mozilla.org/en-US/docs/DOM/window.onload)
* [window.dump](https://developer.mozilla.org/en-US/docs/DOM/window.dump)
* [window.scrollTo](https://developer.mozilla.org/en-US/docs/DOM/window.scrollTo)

Although the DOM is often accessed using JavaScript, it is not a part of the JavaScript language. It can also be accessed by other languages.

querySelector (p ’cica’)– első cicát adja vissza

querySelectorAll – összes cicát

jobb mint a tagname-es Anikó szerint

classList.add(„cica”)

remove vmi

toggle vmi

események: van ugye onload, meg onclick

de most már van olyan, hogy addEventListener(’click’, meghívandó függvény neve) – *ha lehet ne itt hozzuk létre a fv-t*

## Forms

neve MyForm a html-ben

js:

var myForm = document.forms.myForm;

kicsit még variálja a kinézetet:

myForm.name.onfocus = function ()(

myForm.name.style.border = „4px solid pink”;

};

.onblur is van – ellentéte az onfocusnak

.onsubmit

**Form validation**

myForm.onsubmit = function() {

if myForm.name.value == „”){

message.innerHTML = „please enter sg”;

return false;

} else {

message.innerHTML = „”;

return true;

}

## Libraries

jquery

$ - jellel indul

var myPara = document.getElementById(„content”).getElementByTagName(„p”)[5];

ugyanaz mint

var myPara = $(„#content p:last-child”);

**what next?**

Libraries: jquery, MooTools, Modernizr

ajax – communication w server w/o leaving the web page

json

## functions:

function getAverage (a, b) {

var average = (a+b)/2;

console.log(average);

}

hívása: getAverage(x, y);

**variable scope:**

local: egy function-ben van

numbers are blue on console, strings are black

típus meghatározása: typeof (vmi)

**Function Expressions**

var SayBye = function(){ *an anonymous function*

console.log(’bye’);

};

*meghívjuk*

sayBye();

most egy fv-t berakunk egy másik fv-be (?)

function callFunction(fun){

fun();

}

*aztán meghívjuk*

callFunction(sayBye);

**Functions: declarations and expressions**

**basic**

|  |  |
| --- | --- |
| 1 | function sayHi(name) { |

|  |  |
| --- | --- |
| 2 | alert("Hi, "+name) |

|  |  |
| --- | --- |
| 3 | } |

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 5 | sayHi('John') |

**return**

|  |  |
| --- | --- |
| 1 | function sum(a, b) { |

|  |  |
| --- | --- |
| 2 | **return a+b** |

|  |  |
| --- | --- |
| 3 | } |

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 5 | var result = sum(2,5) |

|  |  |
| --- | --- |
| 6 | alert(result) |

An empty return gives undefined

**Local variables**

|  |  |
| --- | --- |
| 1 | function sum(a, b) { |

|  |  |
| --- | --- |
| 2 | var sum = a + b |

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 4 | return sum |

|  |  |
| --- | --- |
| 5 | } |

**Function Declaration**

can be called both after and before the definition

|  |  |
| --- | --- |
| 1 | function sayHi(name) { |

|  |  |
| --- | --- |
| 2 | alert("Hi, "+name) |

|  |  |
| --- | --- |
| 3 | } |

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 5 | sayHi("John") |

**Function Expression**

|  |  |
| --- | --- |
| 1 | var f = function(name) { |

|  |  |
| --- | --- |
| 2 | alert("Hi, " + name + "!"); |

|  |  |
| --- | --- |
| 3 | } |

can be used only after they are executed.

***Use declarations, please***

Function is a value

One function can accept another function as an argument.

**Running at place**

It is possible to create and run a function created with Function Expression at once

when we want to do the job involving local variables. We don’t want our local variables to become global, so wrap the code into a function.

Why function is in brackets? That’s because JavaScript only allows Function Expressions to be called in-place.

**Named function expressions (NFE)**

A function expression may have a name:

var f = function sayHi(name) {

alert("Hi, "+name)

}

NFEs exist to allow recursive calls from anonymous functions:

setTimeout(function factorial(n) {

return n == 1 ? n : n\*factorial(n-1)

}, 100)

**Function naming**

A function is an action. So it’s name should be a verb

## Modules & require

csinálunk egy fv-t

egy másik file-ban behívjuk:

require(’./filenév’)

de még az eredeti file-ban oda kell írni, hogy:

module.exports = fv neve;

## Array

create: var fruits = ["Apple", "Banana"];

index: var first = fruits[0];

loop over:

fruits.forEach(function (item, index, array) {

console.log(item, index);

});

Add to the end: var newLength = fruits.push("Orange");

Remove from the end: var last = fruits.pop();

Remove from front: var first = fruits.shift();

Add to the front: var newLength = fruits.unshift("Strawberry")

Find the index of an item: var pos = fruits.indexOf("Banana");

Remove an item by Index Position: var removedItem = fruits.splice(pos, 1);

Copy an Array: var shallowCopy = fruits.slice(); 2 szám közti részt kiveszi

**Accessing array elements**

console.log(arr[0]); // logs 'this is the first element'

**concat**

var new\_array = old\_array.concat(value1[, value2[, ...[, valueN]]])

Elements of the original arrays are copied into the new array

var alpha = ['a', 'b', 'c'],

numeric = [1, 2, 3];

var alphaNumeric = alpha.concat(numeric);

console.log(alphaNumeric); // Result: ['a', 'b', 'c', 1, 2, 3]

több array esetén:

var nums = num1.concat(num2, num3);

**Filter:**

creates a new array with all elements that pass the test implemented by the provided function.

var arr = [

{"name":"apple", "count": 2},

{"name":"orange", "count": 5},

{"name":"pear", "count": 3},

{"name":"orange", "count": 16},

];

var newArr = arr.filter(function(item){

return item.name === "orange";

});

**forEach:**

executes a provided function once per array element

arr.forEach(function(item,index){

console.log(item);

});

**Map():**

creates a new array with the results of calling a provided function on every element in this array.

var oldArr = [{first\_name:"Colin",last\_name:"Toh"},{first\_name:"Addy",last\_name:"Osmani"},{first\_name:"Yehuda",last\_name:"Katz"}];

function getNewArr(){

return oldArr.map(function(item,index){

item.full\_name = [item.first\_name,item.last\_name].join(" ");

return item;

});

Parse and return a array of objects that contains a additional new property, full\_name

**reduce:**

applies a function against an accumulator and each value of the array (from left-to-right) has to reduce it to a single value

pl. Parse the array and return an object that contains the number of times each string occured in the array:

var arr = ["apple","orange","apple","orange","pear","orange"];

function getWordCnt(){

return arr.reduce(function(prev,next){

prev[next] = (prev[next] + 1) || 1;

return prev;

},{});

accumulator: olyan mint a counter, de nem csak 1-gyel nőhet

function accumulate(arr) – a listában lévő elemekkel növeli a vmi-t

paraméterek: callback item, initial value

**every()**

tests whether all elements in the array pass the test implemented by the provided function.

*arr*.every(*callback*[, *thisArg*])

**callback**

Function to test for each element, taking three arguments:

**currentValue (required)**

The current element being processed in the array.

**index (optional)**

The index of the current element being processed in the array.

**array (optional)**

The array every was called upon.

**thisArg**

Optional. Value to use as this when executing callback.

executes the provided callback function once for each element present in the array until it finds one where callback returns a falsy value

function isBigEnough(element, index, array) {

return element >= 10;

}

[12, 5, 8, 130, 44].every(isBigEnough); // false

[12, 54, 18, 130, 44].every(isBigEnough); // true

**Some**

tests whether some element in the array passes the test implemented by the provided function.

*arr*.some(*callback*[, *thisArg*])

**Parameters**

**callback**

Function to test for each element, taking three arguments:

**currentValue**

The current element being processed in the array.

**index**

The index of the current element being processed in the array.

**array**

The array some() was called upon.

**thisArg**

Optional. Value to use as this when executing callback.

returns true if the callback function returns a truthy value for any array element; otherwise, false.

function isBiggerThan10(element, index, array) {

return element > 10;

}

[2, 5, 8, 1, 4].some(isBiggerThan10); // false

[12, 5, 8, 1, 4].some(isBiggerThan10); // true

var fruits = ['apple', 'banana', 'mango', 'guava'];

function checkAvailability(arr, val) {

return arr.some(function(arrVal) {

return val === arrVal;

});

}

checkAvailability(fruits, 'kela'); //false

checkAvailability(fruits, 'banana'); //true

## Switch

evaluates an expression, matching the expression's value to a case clause, and executes statements associated with that case

first evaluates its expression. It then looks for the first case clause whose expression evaluates to the same value as the result of the input expression (using strict comparison, ===) and transfers control to that clause, executing the associated statements.

In the following example, if expr evaluates to "Bananas", the program matches the value with case "Bananas" and executes the associated statement. When break is encountered, the program breaks out of switch and executes the statement following switch. If break were omitted, the statement for case "Cherries" would also be executed.

switch (expr) {

case "Oranges":

console.log("Oranges are $0.59 a pound.");

break;

case "Apples":

console.log("Apples are $0.32 a pound.");

break;

case "Bananas":

console.log("Bananas are $0.48 a pound.");

break;

case "Cherries":

console.log("Cherries are $3.00 a pound.");

break;

case "Mangoes":

case "Papayas":

console.log("Mangoes and papayas are $2.79 a pound.");

break;

default:

console.log("Sorry, we are out of " + expr + ".");

}

console.log("Is there anything else you'd like?");

## Objects

objectName.propertyName

var myCar = new Object();

myCar.make = "Ford";

myCar.model = "Mustang";

myCar.year = 1969;

**Enumerating all properties**

* [for...in](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...in) loops  
  This method traverses all enumerable properties of an object and its prototype chain
* [Object.keys(o)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys)  
  This method returns an array with all the own (not in the prototype chain) enumerable properties' names ("keys") of an object o.
* [Object.getOwnPropertyNames(o)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyNames)  
  This method returns an array containing all own properties' names (enumerable or not) of an object o.

**Creating new objects**

object initializers

var obj = { property\_1: value\_1, // property\_# may be an identifier...

2: value\_2, // or a number...

// ...,

"property n": value\_n }; // or a string

The following statement creates an object and assigns it to the variable x if and only if the expressioncond is true:

if (cond) var x = {greeting: "hi there"};

**Using a constructor function**

Create an instance of the object with new.

function Car(make, model, year) {

this.make = make;

this.model = model;

this.year = year;

}

var mycar = new Car("Eagle", "Talon TSi", 1993);

An object can have a property that is itself another object.

var rand = new Person("Rand McKinnon", 33, "M");

var car1 = new Car("Eagle", "Talon TSi", 1993, rand);

**Object.create method**

var animal1 = Object.create(Animal);

**Indexing object properties**

You can refer to a property of an object either by its property name or by its ordinal index

**Defining properties for an object type**

You can add a property to a previously defined object type by using the prototype property. This defines a property that is shared by all objects of the specified type

Car.prototype.color = null;

car1.color = "black";

**Defining methods**

a property of an object that is a function

they have to be assigned as the property of an object

objectName.methodname = function\_name;

var myObj = {

myMethod: function(params) {

// ...do something

}

};

You can then call the method in the context of the object as follows:

object.methodname(params);

You can make this function a method of car by adding the statement

this.displayCar = displayCar;

So, the full definition of car would now look like

function Car(make, model, year, owner) {

this.make = make;

this.model = model;

this.year = year;

this.owner = owner;

this.displayCar = displayCar;

}

**this**

suppose you have a function called validate that validates an object's value property, given the object and the high and low values:

function validate(obj, lowval, hival) {

if ((obj.value < lowval) || (obj.value > hival)) {

alert("Invalid Value!");

}

}

Then, you could call validate in each form element's onchange event handler, using this to pass it the element, as in the following example:

<input type="text" name="age" size="3"

onChange="validate(this, 18, 99)">

**egyéb**

simán a fv magában = egy változó, amiben benne van a fv

hozzárendelhetem máshoz

function runFunction(function, a, b) {

console.log(fun(a, b));

}

runFunction(substract, 4, 8) – ki fogja vonni

behelyettesítődik a substract a fun helyére

névtelen fv-t is be lehet tenni egy változóba

runFunction(function(a, b) {

return a/b;

}, 4, 8);

a paraméterek közé beírtuk a fv-t, és egyből le is futtatta

forEach is ugyanez

map: visszatér egy új listával

filter: true-nél beteszi, false-nál nem

foreach első 2 paramétere: mi az elem és hogy hányadik

rövidítve a map: [1, 2, 3].map(e => e \* 2)

**object példa:**

var car = {

km: 120000

ride: function(km)

this.km += km;

}

}

car.ride(120) – akkor a km 120120 lesz

**konstruktor példa:**

function Car(km) {

this.km = km

this.ride = function(km) {

this.km += km

}

}

*//nincs benne return*

## Prototypes

**\*food\* is the prototype of \*waffles\* and \*carrots\*.**

const food = {

init: function (type) {

this.type = type

},

eat: function () {

console.log(’you ate the ’ + this.type)

}

}

const waffle = Object.create(food)

waffle.init(’waffle’)

**ellenőrzése:**

food.isPrototypeOf(waffle)

**prototype**

egy új cucc mindig tárolja, h ő minek a prototípusa

konstruktorral együtt használjuk

ha nem talál meg vmit saját magán, akkor megkeresi a prototype-jában

egy konstruktorhoz egy prototype van

**Date**

new Date();

new Date(*value*);

new Date(*dateString*);

new Date(*year*, *month*[, *day*[, *hour*[, *minutes*[, *seconds*[, *milliseconds*]]]]]);

Integer value representing the number of milliseconds since 1 January 1970 00:00:00 UTC

## Higher-order functions

functions are values

composition is possible – fv-ek egymásba rakása

**filter**

pl. a **filter**-nek egy argumentje van – egy másik fv

var dogs = animals.filter(function(animal) {

return animal.species === ’dog’

})

a benti cucc a callback function

ha kivesszük az isDog-ot egy külön fv-be, akkor meg csak simán:

var dogs = animals.filter(isDog)

otherAnimals = animals.reject(isDog)

**reject**

filter inverze a **reject** függvény

**map**

van egy lista

animals. mindegyiknek van name-je és species-e

var names = animals.map(function(animal) {

return animal.name + ’ is a ’ + animal.species

})

itt a callback nem egy true/false dolog, hanem hogy hogyan módosítsa az elemeket

arrow functions-szel: var names = animals.map( (x) => x.name)

**find**

ez is egy list transformation eszköz

**reduce**

ez egy szuper list transformation eszköz

egy functiont és egy starting értéket vesz

van egy lista amountokkal, össze akarjuk adni az amount-ot

var totalAmount = orders.reduce (function(sum, order) {

return sum + order.amount

}, 0)

az utolsó 0 a kiinduló érték (lehet szám vagy üres object). sum mindig nő az amount-tal

*advanced:*

van egy lista, 2 személy, van nekik 3-3 fegyverük, azoknak 3 tulajdonságuk

vmi file-t megnyit (data.txt)

import fs from ’fs’

var output = fs.readFileSync(’data.txt’, ’utf8’)

.trim()

.split(’ \n’)

.map(line => line.split(’\t’))

.reduce( (customers, line) => {}, )

customers[line[0]] = customers[line[0]] || []

customers[line[0]].push({

name: line[1],

price: line[2],

quantity: line[3]

})

return customers

}, {})

## THIS

**Object.prototype**

All objects in JavaScript are descended from [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object); all objects inherit methods and properties fromObject.prototype,

|  |
| --- |
| function go(a,b) { |

|  |  |
| --- | --- |
| 2 | alert("a="+a+", b="+b) |

|  |  |
| --- | --- |
| 3 | } |

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 5 | go(1)     // a=1, b=undefined |

|  |  |
| --- | --- |
| 6 | go(1,2)   // a=1, b=2 |

|  |  |
| --- | --- |
| 7 | go(1,2,3) // a=1, b=2, extra argument is not listed |

ha var-t lát bárhol, akkor hoist-olja, nem lesz error, viszont fölötte nem kap értéket, csak tudja h van ilyen

let-nél viszont erroros lesz

**let**: 2 dologra jó -> hogy block scope (nem csak fv, hanem for is, meg ilyenek) és nem hoistol!

**const**: utána már nem módosítsható az értéke

&& és || operátorok:

||-gyal adhatunk default értéket egy fv-nek, pl.

function printName(n) { console.log(n ||’nobody’)}

## Unittest

### Tape

var tape = require(’tape’);

falsy / truthy

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

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

reduce: más típusú dolgot csinál belőle, leredukálja

sum és az every is ilyen

reduce-példa:

var numbers = [4, 5, 2, 15, 9];

var sum = numbers.reduce(function(acc, e, i, arr) {

return acc + e

});

első körben az acc = 4, a 2. körben 9, stb, az utolsó értékét az acc-nak adja vissza a fv

### mocking

dependence injection pattern

van egy meal.js

hogyan teszteljük?

be kell adnunk neki a connectiont, de ne legyen valós hívás

tape(function(t) {= {

var mockConnection

query: function(sql, cb) {

cb(null, [{}, {}, {}])

}

},

var meal = create meal(mockConnection);

meal.getAll(function(err, meals) {

t.equal(meals.length, 3);

t.end();

});

{);

hamis connection!!

kell egy createApp függyvény, aminek a paramétere a connection. így igazi meg hamis connection-nel is létrehozhatjuk

ha van expect, kell flush

injektálás fv argumentumokon keresztül történik

videók: mocking, dependency injection pattern-re keresni

**$httpBackend**

<https://docs.angularjs.org/api/ngMock/service/$httpBackend>

When an Angular application needs some data from a server, it calls the $http service, which sends the request to a real server using $httpBackend service.

With dependency injection, it is easy to inject $httpBackend mock (which has the same API as $httpBackend) and use it to verify the requests and respond with some testing data without sending a request to a real server.

two ways to specify what test data should be returned:

* $httpBackend.expect - specifies a request expectation
* $httpBackend.when - specifies a backend definition

Request expectations: .expect(...).respond(...)

Backend definitions: .when(...).respond(...)

**Flushing**

allows the test to explicitly flush pending requests. This preserves the async api of the backend, while allowing the test to execute synchronously.

**mock $httpBackend**

describe('MyController', function() {

var $httpBackend, $rootScope, createController, authRequestHandler;

beforeEach(module('MyApp'));

beforeEach(inject(function($injector) {

// Set up the mock http service responses

$httpBackend = $injector.get('$httpBackend');

// backend definition common for all tests

authRequestHandler = $httpBackend.when('GET', localhost könyvtár)

.respond({meals: tulajdonságok});

// Get hold of a scope (i.e. the root scope)

$rootScope = $injector.get('$rootScope');

// The $controller service is used to create instances of controllers

var $controller = $injector.get('$controller');

createController = function() {

return $controller('MyController', {'$scope' : $rootScope });

};

}));

**Methods**

when(method, url, [data], [headers], [keys]); *- creates a new backend definition.*

whenGET(url, [headers], [keys]); - *Creates a new backend definition for GET requests.*

expectGET(url, [headers], [keys]); - *Creates a new request expectation for GET requests.*

flush([count], [skip]); *- Flushes pending requests using the trained responses. Requests are flushed in the order they were made. useful for simulating scenarios where responses arrive from the server in any order.*

verifyNoOutstandingExpectation(); - *Verifies that all of the requests defined via the expect api were made. If any of the requests were not made, verifyNoOutstandingExpectation throws an exception.*

verifyNoOutstandingRequest(); *- Verifies that there are no outstanding requests that need to be flushed.*

resetExpectations(); - *Resets all request expectations, but preserves all backend definitions. Typically, you would call resetExpectations during a multiple-phase test when you want to reuse the same instance of $httpBackend mock.*

## AJAX Requests

asynchronous JavaScript and XML- kérdezünk a szervertől vmit. kezdi a fetch kiváltani

communicate with a server

async.js

window.onload = function()

var http = new XMLHttpRequest();

http.open(„GET”, „data/tweets.json”, true) – utolsó elem h aszinkron –e vagy sem

http.send();

different ready states: not initialized, has been set up, sent, in process, retrieved

ezek között váltogat ez:

http.onreadystatechange = function () {

console.log(http);

}

van http.status is

## callback

paraméterként megadott function

szinkronos: calls it at the same time, not later

aszinkron esetén: későbbi időpontban történik

## promise

to organize callbacks

an object that represents sg that’s not finished yet

a placeholder for an async operation, like a httprequest

in the promise we put callbacks that run when the request completes

var promise = get(„valami.json”);

promise.**then**(function(valami) {

console.log(valami);

}).**catch**(function(error) {

console.log(error)

});

ha több callback van, akkor összeköthetőek

a promise-on belül berakhatunk egy újabb get-et



## generators

function\* gen() {

var x = yield 10;

}

yield-del lehet pause-olni, és kiadja azt a változót, ami a yield után jön

var myGen = gen(); - nem futtatja, csak előkészíti

futtatás: myGen.next()

ez egy object-et ad vissza: {value: 10, done: false}

false, mert nem ment végig a generator



## reading writing files

egy modul kell hozzá

var fs = require(’fs’)

fs.readFileSync(’filename’, ’utf8’)

blocking the code until we leave

writeFileSync

sync nélkül is lehet

## http request

xml egy leíró nyelv

az onloadot hívja meg ha elkészült

az xhr.response lesz az amivel válsszolt a szerver

meg kell openelni

meg kell mondani milyen típus (pl. get) és az url-t meg kell adni és elküldeni a kérést

ha visszaérkezett, meghívódik a fv

var xhr = new XMLHttpRequest();

xhr.onload = function() {

console.log(xhr.response);

};

xhr.open('GET', 'http://calapi.inadiutorium.cz/api/v0/en/calendars/default/2015/6/27')

xhr.send();

valami egyházi API

példa:

<script>

var xhr = new XMLHttprequest();

xhr.onload = function() {

console.log(xhr.response);

};

xhr.open(’GET’, url)

xhr.send();

</script>

## API

application programming interface

request – response

restaurant – order to kitchen API -> waiter

vagy online repjegyfoglalásnál

so, API is a waiter ruinning between applications, databases & devices

delivering data

**REST API**

pl. facebook graph api

graph.facebook.com/youtube?fields=id,name,likes

post/get

**RESTful**

web services: soap rest

representation state transfer

client – server

resource url-jét használják a kliensek

request – response (representation of resource) sokféle formátumban

restful – client can directly access thru url

self-descriptive messages

uniform interfaces

http stateless

get: retrieve

post: create

put: update

delete: delete

**Postman**

API Debugging chrome app

var xhr = new XMLHttpRequest();

xhr.open(„GET”, „textfile.txt””, true) *– aszinkronos lesz*

xhr.onreadystatechange = function {

if (xhr.readystate === 4) { - *elkészült*

var status = xhr.status;

if (status >= 200 && status < 300 || status == 304) {

alert(xhr.responseText);

} else {

alert(’sg bad happened’);

}

};

xhr.send(null);

ha a statusproperty 200 és 299 között van, akkor minden oké, meg a 304

**AJAX:**

1. csinálunk egy xhr objectet
2. megnyitjuk,
3. választunk egy methodot (pl. get)
4. megadjuk url-t,
5. szinkront
6. on readystate eventet megadjuk
7. request status ellenőrizzük
8. küldjüök a request-et

POST esetén:

<body>-ba berakott egy form-ot

végére: var getRequestBody és elkapjuk benne a html-ből a form-ot

form elelmeit változókba rakja

encodeURIcomponent method – encodes all spec chars

values.push(complete) és values.join(’&’)

elejére eventutility.addEvent(form „submit” function execute) stb

## Clients & Servers

Request/response

Certain protocols – agreed set of rules

Socket van client és server között, packet-et küld TCP protocolban (the way it’s sent)

Ip-cím után be kell írni a port numbert – ebből tudja, hogy a node.js-nek küldtük és nem más programnak – olyan mint egy gate a reptéren

## Creating a Server

Kell a http modul:

var http = require(’http’)

var server = http.createServer(function(req, res) { //request & response objects

res.writeHead(200, {’Content-Type’: ’text/plain’});

res.end(’HEY’);

});

Server.listen(portnr, ’ipadress’);

*Request & response headers: ebben van, h mi a content type, status, stb*

*200 - status*

Böngészőben: után írjuk be a portnr-t

## NPM

node package manager

Csomagokat lehet installálni, pl. express

**Package.json**

Dependencies: apllication-ünk függ tőle, be lehet jelölni, uninstall-nál is fent marad

## Express

egy library – mások által írt kódcsomag, szerver íráshoz

route: a / jel után minden az elérési útvonalban

A node package

Npm install express –save

Aztán, behívjuk a modult, berakjuk egy változóba a fv-t:

Var express = require(’express’);

Var app = express();

App.listen(3000); *- adunk neki egy port nr-t*

Requestek az alábbiak szerint működnek:

GET – app.get(’route’, fn)

POST – app.post(’route’, fn)

DELETE – app.delete(’route’, fn)

Pl.

app.get(’/’, function(req, res)) {

res.send(’this is the homepage’)

});

Utána be kell írnunk a böngészőbe, hogy localhost 3000 / és kiírja azt a szöveget

Content type-ot nem adtuk meg

## Route Parameters

app.get(’/profile/:id’, function(req, res)) {

res.send(’you see a profile with an id of ’ + req.params.id)

});

localhost 3000 /profile/egy számon látjuk ezt

## Template Engines

Fileküldés: res.sendFile(filenév elérési úttal + url (pl. /index.html)

Install EJS as a package (embeddedejs.com) – templating language

Npm install ejs –save

Tell express that we want to use ejs as a view engine: app.set(’view engine’, ’ejs’);

Most nem sendfile-t hanem render method-ot alkalmazunk, ahhoz h a view-t (vmi html) elküldjük

app.get(’/profile/:name, function(req, res)) {

res.render (’profile’, {person: req.params.name}); *- ez a profile utáni név*

});

És így lett egy html oldalunk (ejs kiterjesztéssel)

Html-be is berakhatjuk ezt a nevet: a h1 szövegében **<%= person %>**

Berakott még egy olyat a fv-be-hogy:

Var data = {age: 29, job: ’ninja’ }

Data:data

Html-be: data.age

Lehet olyat is, h egy listát raksz be a data-ba, és a html-ben végigmész rajta for-ral

Így: fv-be berakott egy hobbies-t

<ul>

<% data.hobbies.forEach(function(item) { %>

<li> <%= item %> </li>

<% }); %>

</ul>

## static files & middleware

Deal with static file requests

A honlap is lássa css file-unkat pl.

Middleware – between request and response

app.use(’/directory name’, function(req, res, next) { *- go to the next middleware*

next();

});

Vagy

app.use(’/directory name’, express.static(’dir name’)) {

});

## Query strings

req.query

app.get(’/contact, function(req, res) {

res.render (’contact’, {qs: req.query});

});

Van egy form a html-ében

Beír egy <p>-t:

<p><%= qs.dept </p> - *a querystring dept tulajdonsága (volt benne dept és person)*

Form-ok input mezőibe beírhatjuk: value = „<%=qs.person %y

## POST requests

Pl. ha formot submittelünk

Asks the server to accept/store data in the body of the request

Html-ben: <form name=vmi method=”POST”></form> - *a name benne lesz a küldött info-ban*

Van ilyen middleware, hogy **body parser**

$ npm install body-parser

Js-be: var bodyParser = require(’body-parser’);

var urlencodedParser = bodyParser.urlencoded({ extended(false) }); - *ez egy middleware*

app.post(’/contact, urlencodedParser, function(req, res) {

res.render (’contact’, {qs: req.query});

});

Ha benyomjuk a submit-buttont, beindul ez a post request fv

Html-ben így írhatjuk ki:

<p> you have contacted <%= data.who %> in the <%= data.dept %> department<p> *- who a form neve pl.*

## To-do App

npm init

npm install express

var express = require(’express’);

var app = express();

app.set(’view engine’, ’ejs’);

app.use(’assets/’, express.static(’/public’))

localhost:3000/assets/styles.css

app.listen(3000);

split it up into modules!

MVC structure: model(data), view(template files), controller(controls)

Create a controller: külön folderbe egy új js file

module.exports = function(app) {

app.get (’/todo’, function (req, res)) {

render vmi

})

};

Lesz egy post is, meg egy delete is

Máshol meg behívjuk require-rel egy változóba, aztán majd elindítjuk azt a function-t

**stringify**, **parse**

json.parse(response)

var.obj = json.parse(„macska”123)

stringify – ugyanez fordítva, hogy le tudjuk menteni

send(null) – send() – a get-nél van, amikor nem küldünk semmit

DONE? legyen inkáb 1,2,3,4

client – küldi a request-et

server.listen(3000, '127.0.0.1');

localhost: 3000-ot írunk böngészőbe

**todo feladat**

todoapp backendjének megírása

ne a heroq-hoz csatlakozzon hanem a localhostot

jason-t vár és jason-nel tér vissza

curl-parancs:

$ curl <http://index.hu> –s

karakteresen visszadja a http választ

-s | python –m json.tool

post-nál meg lehet adni az adatot amit beküldünk neki, content type-ot is meg kell adni

get todo-s egész listát

get todo id alapján

post todos

put todos id felülírja

deletet kitörli id alapján

error ha nincs olyan id (404)

? completetd true – adja vissza a completed-eket

/todos/id

most memóriába tárolja, később majd adatbázisba kell (file-ba)

## Connect to MySQL

<https://github.com/mysqljs/mysql>

var mysql = require (’mysql’)

var connection = mysql.createcConnection({

host: ’localhost’,

user: ’root’

password

database

});

connection.connect();

csinált egy schema.sql file-t

var article = {

author: ’AB’

}

var query = connection.query(’insetr into ?’, article, function (err, result) {

if (err) {

console.error(err);

return;

}

console.error(result)

});

**Installing node-mysql**

mkdir sp-node-mysql

cd sp-node-mysql

npm install mysql

jelszót érdemes environment variable-ben tárolni

## Closures

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

Closures are functions that refer to independent (free) variables (variables that are used locally, but defined in an enclosing scope). In other words, these functions 'remember' the environment in which they were created.

lexical scoping: the scope of a variable is defined by its location within the source code (it is apparent lexically) and nested functions have access to variables declared in their outer scope.

A closure is a special kind of object that combines two things: a function, and the environment in which that function was created. The environment consists of any local variables that were in-scope at the time that the closure was created.

**Closures:** functions are also closures. functions have access to variables that are defined outside the function.

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

videóból:

a class:

class Dog {  
 constructor() {  
 this.sound = 'woof'  
 }  
 talk() {  
 console.log(this.sound)  
 }  
}  
const sniffles = new Dog()  
sniffles.talk() // Outputs: "woof"

This is the same dog as a factory function:

const dog = () => {  
 const sound = 'woof'  
 return {  
 talk: () => console.log(sound)  
 }  
}  
const sniffles = dog()  
sniffles.talk() // Outputs: "woof"

factory functions are flexible, prototype is better when creating a large amount of objects

factory: doesn’t use ’this’

**pattern**: absztraktabb mint egy sablon, nem annyira konkrét, nem csak egy megoldást enged

vanilla js – ha nincs framework – ez ritka

Namespacing patterns – van a könyvben

hogy mindenhol hozzáférhető legyen: különleges névbe egy objektum, aztán annak minden egyes kulcsa egy metódus vagy miafasz

todomvc.com – mindeféle módon megírják a todo-t

<https://github.com/tastejs/todomvc/blob/master/examples/vanillajs/js/model.js>

modulokba tenni: hibakezelési dolgok nem kelleneke a reveal-be scak a request methodok

# MySQL

database management – **DBMS** – programmes to access, manipulate

major types:

- hierarchical (parent child)

- network (many to many)

- relational (tables) - popular mysql, oracle

- object oriented

SQL – structured query language

## Relational Database Concepts

pl.

students ( id, name, os)

classes (id, name, instructor)

The ID is unique, ezekre koncetnrálunk, összekötjük

hozzáadhatunk egy 3. csoportot: enrollments (enrollment id, student id, class id, pl. grade)

*1 vs. many*

**organization & normalization**

oszlopok

norm: új tábla

* atomizálni kell (tovább nem bontható részek) pl. nem cím,hanem város
* új táblák kellenek, primary key útján kapcsolódnak
* hogy kapcsolódnak egymáshoz az adatok

## Tutorial

CREATE TABLE student (

* first\_name VARCHAR(30) NOT NULL,
* state CHAR(2) NOT NULL DEFAULT „PA”,
* sex ENUM (’M’, ’F’) NOT NULL,
* date\_entered TIMESTAMP,
* student\_id INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY); *– ez unique lesz*

primary key uniquely idetifies a record or row, must be given a value, which can’t be changed

Atomic tables

tinyint <128

char, varchar, blob, bigint, float, double, enum, set

INSERT INTO tablename VALUE

SELECT \* FROM student

aztán csinál egy másik table-t

CREATE TABLE class

Foreign key – hivatkozikegy másik primary key-re, doesn’t have to be unique

SELECT x, y FROM table;

RENAME TABLE table

SELECT x, y FROM table WHERE state=”xy” OR vmi

vagy IS NULL

ORDER BY state DESC

LIMIT 5

CONCAT

WHERE first\_name LIKE ’D%’ – d-vel kezdődik

IN bele lehet tenni egy másik select utasítást

LIKE ’\_\_\_y’ – 4 betű és y-ra végződik

SELECT DISTINCT state - mindent csak egyszer sorol fel

hány fiú van:

SELECT COUNT(\*)

FROM students

WHERE sex=’M’;

vagy GROUP BY sex

narrow results: HAVING Amount > 1

math things: MIN(vmi)

DESCRIBE

join: SELECT attributes from tests, scores WHERE date = vmi AND tets.test\_id = scores.test\_id

LEFT JOIN

SELECT column, another\_column, …

FROM mytable;

SELECT \*

BETWEEN … AND … NOT BETWEEN … AND …

IN (…)NOT IN (…)

All strings must be quoted

DISTINCT keyword will remove duplicate rows,

Limiting results to a subset: commonly used with the ORDER BY clause are the LIMIT and OFFSET

## JOIN

**INNER JOIN**.

Select query with INNER JOIN on multiple tables

SELECT column, another\_table\_column, …

FROM mytable

**INNER JOIN another\_table**

**ON mytable.id = another\_table.id**

WHERE *condition(s)*

ORDER BY column, … ASC/DESC

LIMIT num\_limit OFFSET num\_offset;

If the two tables have asymmetric data, which can easily happen when data is entered in different stages, then we would have to use a **LEFT JOIN**, **RIGHT JOIN** or **FULL JOIN**

**OUTER JOIN**.

SELECT DISTINCT building\_name, role

FROM buildings

LEFT JOIN employees

ON building\_name = building;

When joining table A to table B, a **LEFT JOIN** simply includes rows from A regardless of whether a matching row is found in B. The **RIGHT JOIN** is the same, but reversed, keeping rows in B regardless of whether a match is found in A. Finally, a **FULL JOIN** simply means that rows from both tables are kept, regardless of whether a matching row exists in the other table.

## expressions

SELECT **particle\_speed / 2.0** AS half\_particle\_speed

FROM physics\_data

WHERE **ABS(particle\_position) \* 10.0 > 500**;

## aggregate functions

**SELECT AGG\_FUNC(*column\_or\_expression*) AS aggregate\_description**, …

FROM mytable

WHERE *constraint\_expression*;

**Common aggregate**

|  |  |
| --- | --- |
| **Function** | Description |
| **COUNT(**\***)**,**COUNT(***column***)** | A common function used to counts the number of rows in the group if no column name is specified. Otherwise, count the number of rows in the group with non-NULL values in the specified column. |
| **MIN(***column***)** | Finds the smallest numerical value in the specified column for all rows in the group. |
| **MAX(***column***)** |  |
| **AVG(***column*) |  |
| **SUM(***column***)** |  |

## Grouped aggregate functions

SELECT AGG\_FUNC(*column\_or\_expression*) AS aggregate\_description, …

FROM mytable

WHERE *constraint\_expression*

**GROUP BY column**;

**AS** is a keyword in SQL that allows you to rename a column or table using an alias. The new name can be anything you want as long as you put it inside of single quotes.

the columns will not be renamed in either table. The aliases only appear in the result set.

**HAVING** clause is used specifically with the **GROUP BY** clause to allow us to filter grouped rows from the result set.

SELECT group\_by\_column, AGG\_FUNC(*column\_expression*) AS aggregate\_result\_alias, …

FROM mytable

WHERE *condition*

GROUP BY column

**HAVING *group\_condition***;

If you aren't using the `GROUP BY` clause, a simple `WHERE` clause will suffice.

## Order of execution

Complete SELECT query

SELECT DISTINCT column, AGG\_FUNC(*column\_or\_expression*), …

FROM mytable

JOIN another\_table

ON mytable.column = another\_table.column

WHERE *constraint\_expression*

GROUP BY column

HAVING *constraint\_expression*

ORDER BY *column* ASC/DESC

LIMIT *count* OFFSET *COUNT*;

## Insert

schema: the database schema is what describes the structure of each table, and the datatypes that each column of the table can contain: the values in the Year column must be an Integer, and the values in the Title column must be a String.

**INSERT** statement

Insert statement with values for all columns

INSERT INTO mytable

VALUES (value\_or\_expr, another\_value\_or\_expr, …),

(value\_or\_expr\_2, another\_value\_or\_expr\_2, …),

…;

Insert statement with specific columns

INSERT INTO mytable

**(column, another\_column, …)**

VALUES (value\_or\_expr, another\_value\_or\_expr, …),

(value\_or\_expr\_2, another\_value\_or\_expr\_2, …),

…;

## Updating

Update statement with values

UPDATE mytable

SET column = value\_or\_expr,

other\_column = another\_value\_or\_expr,

…

WHERE condition;

always write the constraint first and test it in a **SELECT** query

## Deleting

DELETE FROM mytable

WHERE condition;

## Creating tables

CREATE TABLE IF NOT EXISTS mytable (

column *DataType* *TableConstraint* DEFAULT *default\_value*,

another\_column *DataType* *TableConstraint* DEFAULT *default\_value*,

…

);

CREATE TABLE IF NOT EXISTS todos (id int auto\_increment, todo text, completed enum(’true’, ’false’) DEFAULT(’false’), destroyed(’true’, ’false’) DEFAULT(’false’));

CREATE TABLE todos (

todo\_id int NOT NULL AUTO\_INCREMENT,

todo\_text TEXT,

completed ENUM('true', 'false') DEFAULT 'false',

destroyed ENUM('true', 'false') DEFAULT 'false',

PRIMARY KEY (todo\_id)  
);

CREATE TABLE meals (

id INT NOT NULL AUTO\_INCREMENT,

name TEXT,

calories INT,

date DATETIME,

PRIMARY KEY (id)  
);

|  |  |
| --- | --- |
| **Constraint** | Description |
| **PRIMARY KEY** | This means that the values in this column are unique, and each value can be used to identify a single row in this table. |
| **AUTOINCREMENT** | For integer values, this means that the value is automatically filled in and incremented with each row insertion. Not supported in all databases. |
| **UNIQUE** | This means that the values in this column have to be unique, so you can't insert another row with the same value in this column as another row in the table. Differs from the `PRIMARY KEY` in that it doesn't have to be a key for a row in the table. |
| **NOT NULL** | This means that the inserted value can not be `NULL`. |
| **CHECK (expression)** | This is allows you to run a more complex expression to test whether the values inserted are value. For example, you can check that values are positive, or greater than a specific size, or start with a certain prefix, etc. |
| **FOREIGN KEY** | This is a consistency check which ensures that each value in this column corresponds to another value in a column in another table.  For example, if there are two tables, one listing all Employees by ID, and another listing their payroll information, the `FOREIGN KEY` can ensure that every row in the payroll table corresponds to a valid employee in the master Employee list. |

Movies table schema

CREATE TABLE movies (

id INTEGER PRIMARY KEY,

title TEXT,

director TEXT,

year INTEGER,

length\_minutes INTEGER

);

## Altering tables

Adding columns

ALTER TABLE mytable

ADD column *DataType* *OptionalTableConstraint*

DEFAULT default\_value;

Removing columns

ALTER TABLE mytable

DROP column\_to\_be\_deleted;

Renaming the table

ALTER TABLE mytable

RENAME TO new\_table\_name;

## Dropping tables

DROP TABLE IF EXISTS mytable;

## subquery

<https://sqlbolt.com/topic/subqueries>

Subqueries are used to complete an SQL transformation by nesting one query within another query.

- A **non-correlated** subquery can be run independently of the outer query and can be used to complete a multi-step transformation.

- A **correlated** subquery cannot be run independently of the outer query. The order of operations in a correlated subquery is as follows:

* + - 1. A row is processed in the outer query.
      2. Then, for that particular row in the outer query, the subquery is executed.

*SELECT \**

*FROM flights*

*WHERE origin in (*

*SELECT code*

*FROM airports*

*WHERE elevation > 2000);*

másik:

*SELECT a.dep\_month,*

*a.dep\_day\_of\_week,*

*AVG(a.flight\_distance) AS average\_distance*

*FROM (*

*SELECT dep\_month,*

*dep\_day\_of\_week,*

*dep\_date,*

*sum(distance) AS flight\_distance*

*FROM flights*

*GROUP BY 1,2,3*

*) a*

*GROUP BY 1,2*

*ORDER BY 1,2;*

*SELECT origin, id,*

*(SELECT COUNT(\*)*

*FROM flights f*

*WHERE f.id < flights.id*

*AND f.origin=flights.origin) + 1*

*AS flight\_sequence\_number*

*FROM flights;*

## union

merging columns of different tables

Each SELECT statement within the **UNION** must have the same number of columns with similar data types. The columns in each SELECT statement must be in the same order. By default, the UNION operator selects only distinct values.

*SELECT id, avg(a.sale\_price) FROM (*

*SELECT id, sale\_price FROM order\_items*

*UNION ALL*

*SELECT id, sale\_price FROM order\_items\_historic) AS a*

*GROUP BY 1;*

**UNION ALL** allows us to utilize information from multiple tables in our queries, including **duplicate** values.

**INTERSECT** returns only common rows returned by the two SELECT statements.

**EXCEPT** returns distinct rows from the first SELECT statement that aren’t output by the second SELECT statement.

categories that are in the new\_products table that aren't in the legacy\_products table:

SELECT category FROM new\_products

EXCEPT

SELECT category FROM legacy\_products;

## Conditional Aggregates

aggregate functions that compute a result set based on a given set of conditions.

What do we do when we need to test whether a value is or is not null? We use the special keywords IS NULL or IS NOT NULL in the WHERE clause

**CASE**

a way to represent "if, then, else", or conditional logic

**case {condition}**

**when {value1} then {result1}**

**when {value2} then {result2}**

**else {result3}**

**end**

SELECT

CASE

WHEN elevation < 500 THEN 'Low'

WHEN elevation BETWEEN 500 AND 1999 THEN 'Medium'

WHEN elevation >= 2000 THEN 'High'

ELSE 'Unknown'

END AS elevation\_tier

, COUNT(\*)

FROM airports

GROUP BY 1;

If ELSE is not included, the result will be NULL.

**CASE WHEN**

to identify the total amount of airports as well as the total amount of airports with high elevation in the same result set.

SELECT state,

COUNT(CASE WHEN elevation >= 2000 THEN 1 ELSE NULL END) as count\_high\_elevation\_aiports

FROM airports

GROUP BY state;

the total flight distance as and flight distance by origin for Delta (carrier = 'DL'):

SELECT origin, sum(distance) as total\_flight\_distance,

sum(CASE WHEN carrier = 'DL' THEN distance ELSE 0 END) as total\_delta\_flight\_distance

FROM flights

GROUP BY origin;

CASE statements can be used inside aggregates (like SUM() and COUNT()) to provide filtered measures

## Table transformation

DATETIME(vmi), DATE(vmi), TIME(vmi)

**DATETIME**; Returns the date and time of the column specified. This can be modified to return only the date or only the time.

**DATETIME**(time1, '+3 hours', '40 minutes', '2 days');

returns a time 3 hours, 20 minutes, and 2 days after time1.

SELECT (number1 + number2);: Returns the sum of two numbers.

SELECT **CAST**(number1 AS REAL) / number3;: Returns the result as a real number

SELECT ROUND(number, precision);: Returns the numeric value rounded off

SELECT **ROUND**(distance, 2) as distance\_from\_market

FROM bakeries;

**MAX**(n1,n2,n3,...): returns the greatest value in the set of the input numeric expressions **MIN**(n1,n2,n3,...): returns the least value in the set of the input numeric expressions

**concatenation** of strings: SELECT string1 || ' ' || string2;

**REPLACE**(string,from\_string,to\_string) - returns the string string with all occurrences of the string from\_string replaced by the string to\_string

## Analyzing Business Metrics

order by 1 statement is a shortcut for order by date(ordered\_at). The 1 refers to the first column.

round(1.0 \* sum(amount\_paid) - a shortcut to ensure the database represents the percent as a decimal.

KPIs: key performance indicators

how to calculate DAU (Daily Active Users) for Mineblocks per-platform:

select

date(created\_at),

platform,

count (distinct user\_id) as dau

from gameplays

group by 1, 2

order by 1, 2;

Daily ARPPU - Average Revenue Per Purchasing User:

CTEs, Common Table Expressions, **with clauses**

with {subquery\_name} as (

{subquery\_body}

)

select ...

from {subquery\_name}

where ...

Retention: for all players on Day N, we'll consider them retained if they came back to play again on Day N+1.

self-join: joining every row to every other row.

select

date(g1.created\_at) as dt,

round(100 \* count(distinct g2.user\_id) /

count(distinct g1.user\_id)) as retention

from gameplays as g1

left join gameplays as g2 on

g1.user\_id = g2.user\_id

and date(g1.created\_at) = date(datetime(g2.created\_at, '-1 day'))

group by 1

order by 1

limit 100;

This means "only join rows where the date in g1 is one less than the date in g2", which makes it possible to see if users have returned!

# mongodb

<http://blog.modulus.io/mongodb-tutorial>

<http://www.newthinktank.com/2015/12/mongodb-tutorial/>

<https://www.cheatography.com/ovi-mihai/cheat-sheets/mongodb/>

<https://blog.codecentric.de/files/2012/12/MongoDB-CheatSheet-v1_0.pdf>

display all:

db.restaurants.find()

display the fields restaurant\_id, name, borough and cuisine:

db.restaurants.find({},{"restaurant\_id" : 1,"name":1,"borough":1,"cuisine" :1});

exclude the field \_id:

db.restaurants.find({},{"restaurant\_id" : 1,"name":1,"borough":1,"cuisine" :1,"\_id":0});

display all the restaurant which is in the borough Bronx:

db.restaurants.find({"borough": "Bronx"});

display the first 5 restaurant which is in the borough Bronx:

db.restaurants.find({"borough": "Bronx"}).limit(5);

display the next 5 restaurants:

db.restaurants.find({"borough": "Bronx"}).skip(5).limit(5);

restaurants who achieved a score more than 90:

db.restaurants.find({grades : { $elemMatch:{"score":{$gt : 90}}}});

restaurants that achieved a score, more than 80 but less than 100:

db.restaurants.find({grades : { $elemMatch:{"score":{$gt : 80 , $lt :100}}}});

restaurants which locates in latitude value less than -95.754168

db.restaurants.find({"address.coord" : {$lt : -95.754168}});

restaurants that does not prepare any cuisine of 'American' and their grade score more than 70 and lattitude less than -65.754168

db.restaurants.find(

{$and:

[

{"cuisine" : {$ne :"American "}},

{"grades.score" : {$gt : 70}},

{"address.coord" : {$lt : -65.754168}}

]

}

);

restaurants which does not prepare any cuisine of 'American ' and achieved a grade point 'A' not belongs to the borough Brooklyn. The document must be displayed according to the cuisine in descending order

db.restaurants.find(

{$query:

{

"cuisine" : {$ne : "American "},

"grades.grade" :"A",

"borough": "Brooklyn"

},

$orderby : {"cuisine":-1}

}

);

restaurant Id, name, borough and cuisine for those restaurants which contains 'Wil' as first three letters for its name

db.restaurants.find(

{name: /^Wil/},

{

"restaurant\_id" : 1,

"name":1,"borough":1,

"cuisine" :1

}

);

restaurant Id, name, borough and cuisine for those restaurants which contains 'ces' as last three letters

db.restaurants.find(

{name: /ces$/},

{

"restaurant\_id" : 1,

"name":1,"borough":1,

"cuisine" :1

}

);

restaurant Id, name, borough and cuisine for those restaurants which contains 'Reg' as three letters somewhere in its name

db.restaurants.find(

{"name": /.\*Reg.\*/},

{

"restaurant\_id" : 1,

"name":1,"borough":1,

"cuisine" :1

}

);

restaurants which belongs to the borough Bronx and prepared either American or Chinese:

db.restaurants.find(

{

"borough": "Bronx" ,

$or : [

{ "cuisine" : "American " },

{ "cuisine" : "Chinese" }

]

}

);

restaurant Id, name, borough and cuisine for those restaurants which belongs to the borough Staten Island or Queens or Bronxor Brooklyn

db.restaurants.find(

{"borough" :{$in :["Staten Island","Queens","Bronx","Brooklyn"]}},

{

"restaurant\_id" : 1,

"name":1,"borough":1,

"cuisine" :1

}

);

restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronxor Brooklyn

db.restaurants.find(

{"borough" :{$nin :["Staten Island","Queens","Bronx","Brooklyn"]}},

{

"restaurant\_id" : 1,

"name":1,"borough":1,

"cuisine" :1

}

);

restaurant Id, name, borough and cuisine for those restaurants which achieved a score which is not more than 10

db.restaurants.find(

{"grades.score" :

{ $not:

{$gt : 10}

}

},

{

"restaurant\_id" : 1,

"name":1,"borough":1,

"cuisine" :1

}

);

<http://www.w3resource.com/mongodb-exercises/>

# AngularJS

MVC style JS framework

2 way data binding

primarily for single page apps (without need to referesh), dynamical injection

MVC:

model: database

view: display

controller: functionality control, interaction between V & M, diff. C for diff. app areas

insert data in HTML: <html tag> {{data}} </html tag>

typical workflow when making an AngularJS app:

* Create a module, and use ng-app in the view to define the application scope.
* Create a controller, and use ng-controller in the view to define the controller scope.
* Add data to $scope in the controller so they can be displayed with expressions in the view.

## data binding

html-be beírom h ng-app (vagy body-tag-be)

csinált egy input mezőt, html tag-ben ng-model=”valami”

ha a html-ben kitöltöm az inputot, akkor elmenti azt a szót az ng-modelben megadott változóba

aztán egy p-be berakom a változót {{ }}-be és kiírja az oldalon, ez mindig frissül

{{ title }} is called an expression. Expressions are used to display values on the page.

## directives

Directives bind behavior to HTML elements. When the app runs, AngularJS walks through each HTML element looking for directives. When it finds one, AngularJS triggers that behavior (like attaching a scope or looping through an array).

[**https://docs.angularjs.org/api/ng/directive/ngApp**](https://docs.angularjs.org/api/ng/directive/ngApp)

we use the ng-app directive to define the application scope.

they can not only be attributes, they can be also html elements

**custom directives**

pl. bárminek nevezhetsz egy html elementet

we create a directive that’s gonna be used as an element

home-ban létrehozza <random-ninja> - space ne legyen benne

a routeprovider home-ba is berakjuk a ninjacontroller-t, így a home is hozzáfér az adatfile-hoz

app-ban csinálunk egy directive-et

myNinjaApp.directive('randomNinja') –

ha a htmlben látja a <random-ninja> elemet, ide fog ugrani

benne egy function, abban:

return {

restrict: ’EA’,

scope: {

valami

}

}

E: only use as an element, A: as an attribute

scope: create an own scope

we create an attribute in the ninja html tag

<random-ninja ninjas="ninjas"></random-ninja>

scope:

ninjas: ’=’ --- binding them together

title: ’=’

now we create a controller

we have to add a template

kész, random megjelenít egy képet

myNinjaApp.directive('randomNinja', [function() {

return {

restrict: 'E',

scope: {

ninjas: '=',

title: '='

},

template: '<img ng-src="{{ninjas[random].thumb}}">'

controller function($scope) {

$scope.random = Math.floor \* (Math.random() \* 4);

}

};

}])

vagy templateurl-t adunk meg a template helyett, és egy külön file-ban létrehozunk vmi komplexebbet

**codeacademy-példa:**

directives are readable and reusable

We write code to teach the browser a new HTML element <app-info>, and use it in the view to display each app's details.

the new directive named 'appInfo' returns an object with three options:

* restrict specifies how the directive will be used in the view: 'E' means it will be used as a new HTML element.
* scope specifies that we will pass information into this directive through an attribute named info. The = tells the directive to look for an attribute named info in the <app-info> element, like this: <app-info info="shutterbugg"></app-info>

The data in info becomes available to use in the template given by templateURL.

* templateUrl specifies the HTML to use in order to display the data in scope.info. Here we use the HTML in js/directives/appInfo.html.

*app.directive('appInfo', function() {*

*return {*

*restrict: 'E',*

*scope: {*

*info: '='*

*},*

*templateUrl: 'js/directives/appInfo.html'*

*};*

*});*

We use app.directive to create a new directive named 'installApp'.

The directive contains the three options restrict, scope, and templateUrl that we saw before in the 'appInfo' directive. It also contains a fourth option link. The link is used to create interactive directives that respond to user actions. The link function takes three inputs:

* scope refers to the directive's scope. Any new properties attached to $scope will become available to use in the directive's template.
* element refers to the directive's HTML element.
* attrs contains the element's attributes.

Inside the link function, there are two properties buttonText and installed, and the function download().

*app.directive('installApp', function() {*

*return {*

*restrict: 'E',*

*scope: {},*

*templateUrl: 'js/directives/installApp.html',*

*link: function(scope, element, attrs) {*

*scope.buttonText = "Install",*

*scope.installed = false,*

*scope.download = function() {*

*element.toggleClass('btn-active');*

*if(scope.installed) {*

*scope.buttonText = "Install";*

*scope.installed = false;*

*} else {*

*scope.buttonText = "Uninstall";*

*scope.installed = true;*

*}*

*};*

*}*

*};*

*});*

return-ben lehet egy div is, a funkció név lesz a html-tage neve, és akkor kiírja, amit beleírtunk a return div-jébe

html-ben lehet div + directive neve kötőjeles formában is

Directives are a way to make standalone UI components, like <app-info>

## basic behaviour

*app.directive(„enter”, function () {*

*return function (scope, element) {*

*element.bind()”mouseenter”, function () {*

*element.addClass(„panel”);*

html-ben pedig: <div enter>valami</div>

ha rámegyünk a szövegre egérrel, megkapja azt a class-t

van még directives talking to controllers

directives talking to each other

## services

* + - 1. we make a new service. We use app.factory to create a new service named forecast
      2. service needs to use AngularJS's built-in $http to fetch JSON from the server. Therefore, we add $http to the forecast service as a dependency. Now $http is available to use inside forecast.
      3. inside forecast, we use $http to construct an HTTP GET request for the weather data. If the request succeeds, the weather data is returned; otherwise the error info is returned.
      4. in the controller, we use the forecast service to fetch data from the server. First we add forecast into MainController as a dependency so that it's available to use. Then within the controller we use forecast to asynchronously fetch the weather data from the server and store it into $scope.fiveDay.

*app.factory('forecast', ['$http', function($http) {*

*return $http.get('https://s3.amazonaws.com/codecademy-content/courses/ltp4/forecast-api/forecast.json')*

*.success(function(data) {*

*return data;*

*})*

*.error(function(err) {*

*return err;*

*});*

*}]);*

Why are services useful? Instead of filling the controller with code to fetch weather data from a server, it's better to move this independent logic into a service so that it can be reused by other parts of the app.

Services are a way to make standalone communication logic, like forecast which fetches weather data from a server

## expressions

egy p-be berakom, hogy {{ 5 \* 5 }}, akkor kiírja hogy 25

<body ng-init=[1,2,3,4]> - initialize data when an app loads

p-be beírható, hogy numbers[0], kiírja hogy 1

ng-init-ben változónak is adhatunk értéket, aztán class névnek megadhatjuk ezt a változót, és az adott class-hez rendelhetünk vmi css-formázást

## ng-repeat

ng-init-be ír egy stringekből álló listát (ninjas)

li-kbe berakja őket

<li ng-repeat=„vmi in ninjas”> {{vmi}} </li>

useful when we have large amounts of data

## modules

csináltunk egy app.js file-t

var myNinjaApp = angular.module('myNinjaApp', []) – oda mennek az injected dependency-k

a html tag-ben megadott ng-app-et ide irányítjuk (modul nevét adjuk meg, nem a változóét)

myNinjaApp.config –before the app runs

myNinjaApp.run – as the app runs

myNinjaApp.controller

minden a modulban van

## Controllers

A controller manages the app's data.

head-be bírjuk az app.js-t: <script src="app/app.js"></script>

diff controllers control diff parts of the app

app.js-be: myNinjaApp.controller(’name’, function($cope)

a html-ben pedig a body-ban egy div: <div ng-controller="name"></div>

ez a div a controller scope-ja (scope: binding part between the html & the controller) - *ng-controller is a directive that defines the controller scope. This means that properties attached to $scope in MainController become available to use within <div class="main">.*

controller-be $scope.message=”vmi”

html-be: <p>{{message}}</p> és kiírja, hogy vmi” (html-be nem kell írni, hogy $scope)

$scope is a dependency in the controller

but we have to protect the variable names!

function-t []-be kell tenni, function elé, a [ mögé beírni a dependency-t (itt: $scope) (többit is, ha van még)

## filters

change the display of output data

array of objects, mindegyiknek van name és belt

li-ben csak ninja.name-t írunk pl.

filter (pl. abc-sorrend):

ninja in ninjas | orderBy: ’name’

’-name’: ford. sorrend

we can add more than just 1 filter with |

filter: search, és elé a html-ben egy input mező (ng-model=search), csak azt írja ki, amiben benne van az a betűkombináció

megadunk egy számot is az arrayben nevenként

aztán li-be {{ninja.name}} - {{ninja.rate | currency: Ł}}, ez a display-ben a számot gbp-ben adja meg

van még csomó filter (**filter components in ng**)

date filter is

## ng-include

can bring your html from another file

we can break up a complex html file into logical pieces, or for repetitions

most navigation bar-t csinálunk vele

new file: header.html

how to include:

* + - 1. body alá: <ng-include src=”’header.html’”> </ng-include> IDÉZŐJELEK!!!!!!!
      2. ng-include elé header-t is be lehet írni, úgy követhetőbb: <header ng-include”’header.html’”></header>

## ng-show

beraktuk az egészet egy content div-be

egy placeholder-t az inputba

új tulajdonság a listába: available: true/false

csak azt mutatja, amelyik true-ra értékelődik

| filter: search” ng-show=”ninja.available”

!-lel megfordítható

ng-hide-dal is lehet megfordítani

The **ngShow** directive shows or hides the given HTML element based on the expression provided to the ngShow attribute. The element is shown or hidden by removing or adding the ng-hide CSS class onto the element. The .ng-hide CSS class is predefined in AngularJS and sets the display style to none

When the ngShow expression evaluates to false then the ng-hide CSS class is added to the class attribute on the element causing it to become hidden. When true, the ng-hide CSS class is removed from the element causing the element not to appear hidden.

## ng-if

removes or recreates a portion of the DOM tree based on an expression. If the expression assigned to ngIf evaluates to a false value then the element is removed from the DOM, otherwise a clone of the element is reinserted into the DOM.

When an element is removed using **ngIf** its scope is destroyed and a new scope is created when the element is restored. The scope created within ngIf inherits from its parent scope using prototypal inheritance.

## ng-click

egy gombbal más tulajdonság alapján rendezzük sorba

<button ng-click="order = ’name’">Order by Name</button>

azt akarjuk hogy a filterben az orderby dinamikusan cserélődjön

orderBy: order (változó) click-re name lesz

add a cross to delete item!

<div class="remove" ng-click="removeNinja">x</div>

egy függvényt írunk a controller-be:

$scope.removeNinja = function(ninja) {

var removedNinja = $scope.ninjas.indexOf(ninja);

$scope.ninjas.splice(removedNinja, 1);

}

splice: megadott index-től számítva 1-et kivesz

egy ng-click-hez több function-t is kapcsolhatunk

## ng-submit

It's very much like ngClick, but instead of actually clicking the button, you can just hit enter and it will submit.

add a new ninja to the list

create a form, w 3 input fields

we will store it in a variable newNinja

a formelementhez kapcsoljuk az ng-submit-ot egy fv-re mutat az app.-js-ben

when we add into a a field, it will be stored as a string! parseInt(rate)!!

clear out the fields

$scope.addNinja = function() {

$scope.ninjas.push({

name: $scope.newninja.name,

colour: $scope.newninja.colour,

rate: parseInt($scope.newninja.rate),

available: true

});

$scope.newninja.name = "";

$scope.newninja.colour = "";

$scope.newninja.rate = "";

}

## ng-src

mindegyik kap egy képet

li-be berakunk egy img html-taget

error: %7B represents {{

ng-src: browser waits until angular pastes the source dinamically, no error in code (like when using just src)

## Views and Routes

not different pages, just different views

button requests the appropriate view and injects it into the html

views directory, új html file

directory-ba beraktuk a komplett content-részt

body-ba <main ng-view></main> ide rakja majd a kért view-t

app.js-be rakjuk a routing-ot, [] a dependenciest – itt: ’ngRoute’

angular.route.min-t letöltjük a honlapról, a lib-be

berakjuk egy config method-ba

*myApp.config([’$routeProvider, function($routeProvider){*

*$routeProvider*

*.when(’/home’, {*

*templateUrl: ’view6home.html’*

*.when( stb*

*.otherwise({*

*redirectTo: '/home'*

*});*

*}])*

a directory.html-ben már nem kell külön megadni a controller-t, mert a config-ban megadtuk

routing: pl. #/directory

## UI router

<https://scotch.io/tutorials/angular-routing-using-ui-router>

it changes your application views based on state of the application and not just the route URL

you can change the parts of your site using your routing even if the URL does not change.

all states, routing, and views are handled in one .config()

in the HTML we also load up ui-router in addition to loading Angular.

When creating a link with UI-Router, you will use ui-sref. The href will be generated from this and you want this to point to a certain state of your application. These are created in your app.js.

also use <div ui-view></div> instead of ngRoute’s <div ng-view></div>.

in app.js:

routerApp.config(function($stateProvider, $urlRouterProvider) {

$urlRouterProvider.otherwise('/home');

$stateProvider

.state('home', {

url: '/home',

templateUrl: 'partial-home.html'

})

.state('about', {

});

});

Let’s fill out our partial-home.html page

**Nested Views:** We’ll add two buttons to our home page. add our buttons to partial-home.html

When linking to a nested view, we use dot denotation: ui-sref=".list" and ui-sref=".paragraph".

These will be defined in our Angular file and once we set it up there, we will inject into our new <div ui-view></div>.

In our app.js file, we create the nested states.

Pl.:

.state('home.list', {

url: '/list',

templateUrl: 'partial-home-list.html',

controller: function($scope) {

Now the ui-sref we defined in home.html are linked to an actual state. With home.list and home.paragraph created, those links will now take the template provided and inject it into ui-view.

we have to also define the partial-home-list.html file

Now when we click List, it will inject our list of dogs into the template. Or if we click Paragraph, it will inject the string we gave.

**Multiple Views About Page**

Each will have its own controller and template file so our app stays clean.

**RELATIVE VS ABSOLUTE NAMING**

UI-Router assigns every view to an absolute name. The structure for this is viewName@stateName

Since our main ui-view inside our about state, we gave it a blank name. The other two views because columnOne@about and columnTwo@about.

Having the naming scheme this way let’s us define multiple views inside a single state

<https://github.com/angular-ui/ui-router/wiki/Multiple-Named-Views#view-names---relative-vs-absolute-names>

## JSON and $http

külső adatbázis

app.js-ből kivesszük

convert $scope.ninjas -> console.log(angular.toJson($scope.ninjas));

key-ek is idézőjelben!!!

jsonlint.cont – validate

beraktuk egy külön json fileba, app.js-ből töröltük

controller dependency-kbe beírjuk hogy **$http**

controller-be:

$http.get('data/ninjas.json').success(function(data)) {

$scope.ninjas = data;

});

get-teli, aztán aszinkron: siker esetén az lesz a data amit betöltött

## transclude & replace

pass things thru the prev return object

a home-ban mindent kitöröl a random.ninja directive-en belül, és beilleszti a random.html-ben lévőt

de mondhatjuk h ne törölje

transclude: true (a controllerben)

az új html-ben csinálunk egy <div ng-transclude> tag-et és abban lesz benne minden a régiből

replace: true (controllerben) – random ninja tag-ből div lesz

(replaces random-ninja tag with the outermost tag in the html)

## Animations

we have to dl a separate module

**angular.animate-min.js**

set as a dependency in our module (ngAnimate)

ng-enter, ng-fade dokumentációban le van írva

ng-leave-stagger – fokozatosan tűnnek el a lista tagjai

## Form Validation

contact.html-t csináltunk, config-ba beraktuk

from classes: ng-pristine/ng-dirty, ng-untouched/ng-touched, ng-valid/ng-invalid

form tag-ben novalidate -> angular-ra bízzuk a validálást html helyett

input tag-ekbe: ng-required=”true”

kell mindegyikgbe egy ng-model iss

now the fields have diff properties

email field class-e „ng-invalid” lesz ha nem jó amit beírtunk, stb

ng-invalidra beállíthatunk más css-t, pl. piros keret (de ne legyen piros, ha untouched)

vannak property-k: pl. ng-pristine class property-je: $pristine: true

input mező után a html-be egy <div>, ebbe írjuk hogy mikor jöjjön fel az üzenet h nem valid amit beírt a user

pl.

<div ng-show=”contactForm.name.$touched && contactForm.name.$invalid”>Enter a valid name</div>

a végére a submit gombhoz ng-disabled, amíg nem oké minden

ng-disabled="contactForm.$invalid"

css-t is rátehetünk, halványítás, cursor: not-allowed, stb

## Location Service

html-ben a form name-ben lehivatkozunk egy function-t: ng-submit="sendMessage()"

csinálunk neki egy controllert, app.js.-ben lehivatkozzuk a config-ben

dependency-kbe $location

$scope.sendMessage = function() {

$location.path('contact-success')

};

kell még neki egy route:

.when('/contact-success', {

templateUrl: 'views/contact-success.html',

contact: 'ContactController'

## URL-s:

# helyett

location provider method-ot a config-ba, dependency-be $locationProvider, és alá: $locationProvider.html5mode(true);

az index htmle head-be kell még: <base href="/" />

most már a header-ben sem kell a #

de ha beírjuk a böngészőbe, úgy nem működik

## ng-apply, ng-watch

aszinkron

Changing scope values asynchronously - updates don't propagate without .apply().

<https://docs.angularjs.org/api/ng/type/$rootScope.Scope>

## ng-class

to dynamically attach an active class (ha **igaz**, amit hozzárendelünk)

ng-class=”{’active’: isCurrentCategory(category}”

## angular material

a UI Component framework and a reference implementation of Google's Material Design Specification

npm install angular angular-material

scriptbe animate és aria (for accessibity)

Md-toolbar – ide jöhet a tolbar

md-content, md-sidebar, stb

<https://material.angularjs.org/latest/api/directive/mdToolbar>

## egghead

one controller per state is recommended

defining a method: don’t reference $scope inside a function

ng-repeat-ben van olyan hogy „limitTo”, csak x eredményt mutat

# SASS

extension to css

split files into modules

need to be translated into css, written in ruby

install ruby, **prepros**

if we drag sass files into prepros it will translate them into css

a styles scss-t a styles.css-be fogja fordítani a program

a html-t továbbra is a css-hez kötjük

## variables

pl. egy color kódot egy változóba rakunk

minden változó elé $ jel

$deepBlue: #032f3e;

pl. megadhatunk méretet változóba: $sectionHeading: 24px; utána csak itt kell módosítani

## nesting

könnyebb csoportosítani a style-okat így

pl. most belemegyünk a main-nav-ba

de ne legyen túl sok szint

quicker & more efficient

## mixins

chunk of reusable css that we can inject

pl. így néz ki:

@mixin banner {

width: 100%;

position: relative;

color: white;

.banner-content {

position: absolute;

top: 50px;

width: 100%;

}

aztán hozzárendeljük az adott class-hez, kiegészítjük azzal amit csak arra a class-ra akarunk

.lead-banner {

@include banner;

li {

text-transform: uppercase;

font-size: 20px;

max-width: 500px;

margin: 60px 0;

}

}

## importing files

szétszedjük a sass file-t, aztán importálni kell

pl. csak variables az egyikben

nem mindegyikből akarunk css fordítást: auto-comply-ból ki kell venni a pipát a prepros-ban, csak a styles-ból

mindent importálunk a styles-ba, a tetején

@import "reset";

## pseudo classes

extra effects

be nesteljük, és kell elé egy &-jel

a {

color: $offWhite;

text-decoration: none;

padding: 16px;

display: block;

text-align: center;

**&:hover {**

**color: #333;**

**}**

}

vagy berakhatjuk ezt egy mixinbe (clearfix lett a neve), aztán az ul-be beírjuk h @include clearfix

## math operators

pl. width: (40px \* 2); vagy (100% / 6) h egyenlő részekre ossza

## making a grid

3 sorba rakta a képeket így:

#projects li {

float: left;

width: 23%;

margin-right: 2%;

img {

width: 100%;

}

}

de lehet máshogy is. egy új mixint csinálunk, amiben változóként megadható a margin és a column száma (az utolsónak már nincs margin-ja)

@mixin grid ($cols, $mgn) {

float: right;

margin-right: $mgn;

margin-bottom: $mgn;

width: ((100% - (($cols - 1) \* $mgn)) / $cols );

&:nth-child(#{$cols}n) {

margin-right: 0;

}

}

*a styles-ban:*

#projects li {

@include grid(4, 2%);

img {

width: 100%;

}

}

## colour functions

<http://sass-lang.com/documentation/Sass/Script/Functions.html>

pl. a lighten function:

&:hover {

background-color: lighten($color, $amount);

}

első a base color, másik h mennyire világosítsa ki

complement($color)

## content

új mixint csinálunk, mediaQuery

@mixin mQ($arg) {

@media screen and (max-width: $arg) {

@content;

}

}

a styleban pedig ezt írjuk a listaelemekhez

li {

float: left;

width: (100% / 6);

@include mQ(600px) {

width: 100%;

}

ez, ha összehúzzuk a képernyőt, egymásra rakja a listaelemeket

## if statements

az alábbi attól függően fog működni, hogy az include-ban hány paramétert adunk meg

@mixin mQ($arg...) {

@if length($arg) == 1 {

@media screen and (max-width: nth($arg, 1)) {

@content;

}

@if length($arg) == 2 {

@media screen and (max-width: nth($arg, 1)) and (min-width: nth($arg, 2)) {

@content;

}

}

rakhatunk a span-re is, hogy ha nagyobb a screen size, akkor nagyobb szöveg legyen

@include mQ(3000px, 1200px) {

font-size: 68px;

}

# Bootstrap

mobile friendly

CDN or download

head-be: meta name=viewport --- responsive lesz

bemásolom a cdn linkeket

bs has a dependency to jquery!

bemásoljuk a body-ba scriptbe az url-jét (<http://code.jquery.com/jquery-1.12.4.min.js>)

## grid system

12 columns

class names

we add content into rows and columns

content: large screens: 8 columns wide, medium 6, small 12

sidebar: 4, 6, 12

responsive

## containers

you put all your rows and columns in it

container: has fixed width, changes at certain breakpoints, built in padding 15px

container-fluid: fluid width, stretches 100% of its parent element, 15% padding

## rows & columns

to use a row we have to put it into a container

column: to indicate how much horizontal space we want to take up with an element

all columns have to be within a row, by using bootstrap classes

col-{size}-{number}

size of targeted viewport, number of columns you want the element to span

**column classes:**

sizes: xs, sm, md, lg ( 768, 992, 1200 pxs)

number 1 – 12

pl. col-xs-12 vagy col-md-6

<div class="col-md-2 thumb">thumbnail</div>

ha eléri az 1200px-et a képernyő szélessége, akkor lesz 2 col széles, egyébként teljes képernyős lesz

<div class="col-md-2 col-sm-4 col-xs-6 thumb">thumbnail</div>

extra small (<768px) széles képernyőn 6 széles lesz

## Column Offsetting

<div class="col-xs-offset-2 col-xs-12 col-md-4 col-md-offset-0 thumb">

csak kis képernyőnél legyen offset, amúgy ne

## Push & Pull columns

col-lg-push-4 - nagy képernyőn 4 oszloppal jobbra tolja

col-lg-pull-8 – 8-cal balra

de ha mondjuk md-re adjuk meg, akkor utána lg-nél ki kell nullázni, különben arra is érvényes lesz

## clearfix

div class=”clearfix”

így float left-nél nem akad bele a fölötte lévőbe

de magadhatjuk a head-ben is, hogy style --- .thumb:nth-child(odd): clear both;

## Text Styles

headings: h1 – h6

just change body font-family

bárhol <span class=h1>

class=lead is van

## list styles

pl. ul és ol lehet class=”list-inline”

dl lehet clas=”dl-horizontal”

## button styles

class="btn btn-default btn-lg"

van btn-primary, -info, -warning, -danger, -success

btn-xs –sm –lg

active disabled

## image styles

class=”img-responsive” – no overlapping

ha berakjuk egy column-ba

img-circle – köralakú lesz, van még img-rounded + img-thumbnail

de xs-12-nél nem lesz szélesebb a kép eredeti méreténél: ha beírjuk, hogy center-block, akkor ilyenkor középre teszi

## visibility

to hide: „hidden-xs” / „hidden-md” (hidden at xs size, stb)

to show: „visible-xs” / „visible-md-block” (only visible at xs size, stb)

vagy visible—md-inline (eredetileg block) vagy md-inlike-block (egy sorban lesz de lesz szélessége)

## Drop-down Menu's

van egy button (class=”btn btn-default”) és egy lista

benne szöveg és class=”caret” – ettől lesz benne egy nyíl

az egészet berakjuk egy div-be class=”dropdown”

button—nek classbe dropdown-toggle

ul-nek classbe dropdown-menu

button classbe még: data-toggle=”dropdown”

tehetünk még bele egy li-taget, class=”dropdown-header”

bármelyik li lehet class=”disabled”

## Tabs & pills

ul-nek class=”nav”

aztán ul class-be még nav-tabs – horizontal lesz

li-be class=”active” tabs lesz, mint a wikipedia

nav-tabs helyett nav-pills: kicsit másfajta lesz

nav-stacked – egymás alá teszi

nav-justified

## Navbars

ul marad class=”nav”

classbe még navbar-nav

berakjuk az egész ul-t egy <nav> element-be

nav class=”navbar navbar-default”

navbar-fixed-bottom – lent lesz és sosem tűnik el

vagy fixed-top

*logo a navbar elejére linkkel*: az ul elé rakunk egy div-et <div clas=”navbar-header”>

a div-en belül egy anchor tag: <a href=”/index.html” class=”navbar-brand”>VMI SZÖVEG</a>

ez egy link lesz a homepage-re

buttons in the navbar: ul alá rakunk egy button-t: class=”btn btn-default navbar-btn”

ha jobbra akarjuk akkor navbar-right

**mobil-képernyőn:**

ul-t és button-öket berakjuk egy div-be

class=”collapse navbar-collapse” id=”vmi”

a logo-s cucc button-jébe azt írjuk, hogy: class=”navbar-toggle collapsed” data-toggle=”collapse” data-target=”#vmi”

3db csík a gombra: 3-szor <span clas=”icon-bar”>

erre kattintva nyílik majd le a menü

## jumbotrons:

sog that stands out

div class=”jumbotron”

van benne mondjuk egy h1 és egy p

ha a h1-et és p-t egy containerbe tesszük, akkor a szöveg nem fog a képernyő széléig érni

## forms

pl. input class=”form-control”

vagy a label-t és input-ot berkahtajuk egy div-be, és a div-nek class=”form-group”

checkboxos részeket, radio button-öket is formgroup-okba tehetünk, úgy jobba nelkülönülnek

chkboxnál lehet class=”checkbox-inline”

van radio-inline is

ha mindkét radio-nak uazt a name-et adod, akkor csak az egyiket lehet kiválasztani

## badges & labels

a button-on eblül csinálok egy span-t és class=”badge”

illetve ugyanez label-lel, az nem köralakú lesz

## panels

egy dobozba rakunk dolgokat

belerakjuk egy div-be a dolgokat, class=”panel” panel-default (vagy panel-primary stb)

azon belül lesz egy heading (class=”panel-heading”)

címnek lehet class=”panel-title”, contentnek lehet class=”panel-body”

van panel-footer is

## tooltips

rámész vmire és feljön egy buborék

pl egy anchor tagen belül beírjuk hogy data-toggle=”tooltip” data-placement=”top” title=”vmi” (ez jön fel)

bármin lehet, nemcsak a-n

## carousels

váltogatja a megadott képeket

kell egy div, class=”carousel” id=”vmi” data-ride=”carousel”

ezen belül egy div class=”carousel-inner”

ezen belül egy 3-szor div class=”item”

ezekbe berakunk 1-1 képet

de ez így még nem látszik, mert nem aktív

az első item-hez be kell írni, hogy class=”item active”

a külső divhez írhatjuk, hogy „carousel slide”, akkor slide-olnak a képek

az itemeken belül a képeknek adhatunk szöveget egy „carousel caption” class-ű div-vel

**nyilak**

”carousel-inner”-en kívül egy a-tag, class=”right carousel-control” href=”#vmi” (carosuel id-je) data-slide=”next”

ebbe egy span class=”glyphicon glyphicon-chevron-right” – ez egy jobbra nyíl lesz

balra data-slide: prev

**circle-tags**

”carousel-inner”-en kívül egy ol-tag

class=”carousel-indicators”

ebbe li data-target=”#vmi” (id) data-slide-to=”0” – első képre ugrik

ebből annyit ahány kép van

első li legyen class=”active”

jquery-vel adta meg a sebességet, szünetet, stb

## modals

egy kis doboz ami feljön

div class=”modal” id=”vmi”

ezen belül div class=”modal-dialog”

ezen belül div class=”modal-content”

ezen belül div class=”modal-header” és „modal-body”

headerbe h2 class=”modal-title”

body-ba néhány p

ok. amihez hozzárendeljük (pl. volt egy anchor-tag a panel-footerben), abban data-toggle=”modal”

a modal-headerbe mehet egy button class=”close” data-dismiss=”modal”, szöveg: x

ebből lesz egy becsukó gomb

## Accordions

group of panels bundled together

div class=”panel-group” id=”vmi”

ebbe div class=”panel panel-default”

ebbe egy „panel-heading”, amibe egy h3, class=”panel-title” data-target=”#panel1” data-toggle=”collapse”

meg egy „panel-collapse collapse” id=”panel1”, ebbe egy panel-body, abba egy p

klikkelésre eltűnik/megjelenik a p

h3-nak lehet még egy olyan attribute, hogy data-parent=”#pnelgroup id-je” – akkor csak egy van nyitva egyszerre

## Themes

másol által készített css file-ok

free pl. bootswatch.com

vagy letöltjük a file-t, vagy head-be a link:

<link href=”link” rel=”stylesheet” type=”text/css”>

az még mindig módosítható

# Gulp

<https://css-tricks.com/gulp-for-beginners/>

<https://github.com/zellwk/gulp-starter-csstricks>

It's often used to do front end tasks like:

* Spins up a web server
* Compiles Sass to CSS
* Refreshes the browser automatically whenever you save a file
* Optimizes all assets (CSS, JS, fonts, and images) for production

chain together different tasks into simple commands that are easy to understand and execute.

*installing: npm install gulp –g*

aztán: npm install gulp --save-dev

Determining Folder Structure:

|- app/

|- css/

|- fonts/

|- images/

|- index.html

|- js/

|- scss/

|- dist/

|- gulpfile.js

|- node\_modules/

|- package.json

In this structure, we'll use the app folder for development purposes, while the dist (as in "distribution") folder is used to contain optimized files for the production site.

Since app is used for development purposes, all our code will be placed in app.

create a **gulpfile.js**, which stores all Gulp configurations.

var gulp = require('gulp');

gulp.task('task-name', function() {

// Stuff here

});

to run it: $ gulp task-name

*gulp.task('task-name', function () {*

*return gulp.src('source-files') // Get source files with gulp.src*

*.pipe(aGulpPlugin()) // Sends it through a gulp plugin*

*.pipe(gulp.dest('destination')) // Outputs the file in the destination folder*

*})*

a real task takes in two additional gulp methods — gulp.src and gulp.dest.

**gulp.src** tells the Gulp task what files to use for the task

**gulp.dest** tells Gulp where to output the files once the task is completed.

## sass-css preprocessing

We can compile Sass to CSS in Gulp with the help of a plugin called gulp-sass.

npm install gulp-sass --save-dev

a js file-ban:

var sass = require('gulp-sass');

gulp.task('sass', function(){

return gulp.src('app/scss/styles.scss')

.pipe(sass())

.pipe(gulp.dest('app/css'))

});

lefuttatod command line-ban (gulp sass), és létrejön egy css file a fordításával

## Globbing

több scss file estén: Node globs

Globs are matching patterns for files that allow you to add more than one file into gulp.src. It's like regular expressions, but specifically for file paths.

Most workflows with Gulp tend to require 4 different globbing patterns:

* \*.scss: matching any files ending with .scss in the root folder (project).
* \*\*/\*.scss: - matches any file ending with .scss in the root folder and any child directories.
* !not-me.scss: The ! indicates that Gulp should exclude the pattern from its matches
* \*.+(scss|sass): The plus + and parentheses () allows Gulp to match multiple patterns, with ifferent patterns separated by the pipe | character.

gulp.task('sass', function() {

return gulp.src('app/scss/\*\*/\*.scss')

.pipe(sass())

.pipe(gulp.dest('app/css'))

})

compiles all Sass files into CSS files with a single command

## Watching

we can tell Gulp to automatically run the sass task whenever a file is saved through a process called "watching".

gulp.watch('files-to-watch', ['tasks', 'to', 'run']);

PL.

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

we can group together multiple watch processes:

gulp.task('watch', function(){

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

// Other watchers

})

automatically runs the sass task whenever you save a .scss-file

## BrowserSync

helps make web development easier by spinning up a web server that helps us do live-reloading

npm install browser-sync --save-dev

in the js-file:

var browserSync = require('browser-sync').create();

We need to create a browserSync task to enable Gulp to spin up a server. we need to let Browser Sync know where the root of the server should be. In our case, it's the `app` folder:

gulp.task('browserSync', function() {

browserSync.init({

server: {

baseDir: 'app'

},

})

})

*de nálam csak ez működött:*

gulp.task('browserSync', function() {

browserSync.init({

proxy: 'http://localhost:3000',

port: 7000

});

});

We also have to change our sass task so Browser Sync can update the CSS in the browser whenever the sass task is ran.

gulp.task('sass', function() {

return gulp.src('app/scss/\*\*/\*.scss')

.pipe(sass())

.pipe(gulp.dest('app/css'))

.pipe(browserSync.reload({

stream: true

}))

});

Now, we have to run both the watch and browserSync tasks at the same time, by adding a second argument to the watch task:

gulp.task('watch', ['browserSync'], function (){

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

// Other watchers

})

We'll also want to make sure sass runs before watch so the CSS will already be the latest whenever we run a Gulp command.

gulp.task('watch', ['browserSync', 'sass'], function (){

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

// Other watchers

});

Now, if you run gulp watch in the command line, Gulp should start both the sass and browserSync tasks concurrently. When both tasks are completed, watch will run.

At the same time, a browser window that points to app/index.html would also pop up.

we can also reload the browser if any HTML or JavaScript file gets saved by calling the browserSync.reload function when a file gets saved:

gulp.task('watch', ['browserSync', 'sass'], function (){

gulp.watch('app/scss/\*\*/\*.scss', ['sass']);

gulp.watch('app/\*.html', browserSync.reload);

gulp.watch('app/js/\*\*/\*.js', browserSync.reload);

});

## Combining Gulp tasks

we need to use an extra plugin called Run Sequence.

$ npm install run-sequence --save-dev

js file-ba

var runSequence = require('run-sequence');

gulp.task('task-name', function(callback) {

runSequence('task-one', 'task-two', 'task-three', callback);

});

sorba megy végig a task-okon

Run Sequence also allows you to run tasks simultaneously if you place them in an array

we can now create a task that ensures that clean:dist runs first, followed by all the other tasks:

gulp.task('default', function (callback) {

runSequence(['sass','browserSync', 'watch'],

callback

)

})

(when you have a task named default, you can run it simply by typing the gulp command)

## gulp karma

<https://github.com/karma-runner/gulp-karma>

## gulp nodemon

$ npm install --save-dev gulp-nodemon

## gulp mocha

npm install gulp gulp-mocha

## gulp-tape

npm install --save-dev gulp-tape tape

'use strict';

 var gulp **=** require('gulp');

var tape **=** require('gulp-tape');

var tapColorize **=** require('tap-colorize');

gulp.task('test', function() {

**return** gulp.src('test/\*.js')

    .pipe(tape({

      reporter**:** tapColorize()

    }));

});

# Heroku

heroku login

heroku create

git push heroku master

procfile: abba kell beírni, hogy miket indítson el, mi hol van

az lesz benne „web: node vmi.js”

a server file-unkra kíváncsi

process.env.PORT || 3000

***master váltás:***

git remote –v 🡪 kiírja a herokun lévő remote-okat

git remote remove heroku „aktuális master címed”

git remote add heroku „cím”

# Browserify

With Browserify you can write code that uses require in the same way that you would use it in Node.

*npm install uniq*

var unique = require('uniq');

Now recursively bundle up all the required modules starting at main.js into a single file called bundle.js with the browserify command:

browserify main.js -o bundle.js

a html-be még kell egy script:

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

<https://github.com/substack/browserify-handbook>

# Codeship

codeshipre megy, amit pusholunk, onna tovább heroku-ra

csss menjen fel herokura: codeship-en pipeline / custom script-be:

* rm -rf .git
* rm -rf .gitignore
* rm -rf node\_modules/
* git config --global user.name "CODESHIP deploy script"
* git config --global user.email "codeship@example.com"
* git init .
* git remote add heroku git@heroku.com:student-mngmt-tool.git
* git add --all && git commit -am 'CODESHIP deploye'
* git push -f heroku master

# Karma

## basics

<https://www.youtube.com/watch?v=ivwY-nGOHiA>

var testingAngularApp = angular.module(’testingAngularApp’, []);

testingAngularApp.controller(’testingAngularCtrl’, function ($rootScope, $scope) {

$scope.title = „testing AngularJS”;

});

**installációk**

npm install karma-cli phantomjs –g

npm install karma jasmine –save-dev

ez berakja a jason-file-ba a dev dependencies-be

van egy „karma init”-parancs, ami létrehoz egy karma file-t. be kell írni, hogy jasmine.

do you want to capture any browsers automatically? PhantomJS

location of test files? üres

any files excluded? üres

karma watch all the files? yes

létrejön a karma.conf.js

list of files-t manuálisan adjuk meg: angular.js, angular-mocks.js, app.js, unit/\*.js

indítás: karma start karma.conf.js

## First Unit Test

describe(’name’, function() {

beforeEach(module(’testingAngularApp’)

describe(’name of controller’, function(

var scope, ctrl;

beforeEach(inject(function($controller, $rootScope)) { – *to inject angular components into our tests*

scope = $rootscope.$new();

ctrl = $controller(’name’, {$scope: scope} ) *– assign the angular controller to this variable*

*a fent definiált scope variable minden var-hoz hozzáfér ami a controllerben van*

afterEach (function)

*cleanup codes*

it (’name of unit test’, function)

expect(scope.title).toBeDefined();

expect(scope.title).toBe(’name’ );

*scope.title = value to check*

*toBeDefined = sg to check it against*

*ha az it block-ban vmi nem teljesül, akkor a test failed*

a failed testben benne van a describe block meg az it block neve, hogy be lehessen azonosítani

## testing controllers & scope

**csinál egy holiday planning app-et**

var testingAngularApp = angular.module(’testingAngularApp’, []);

testingAngularApp.controller(’testingAngularCtrl’, function ($rootScope, $scope) {

$scope.title = „testing AngularJS”;

$scope.destinations = [];

$scope.newDestination = {

city: undefined,

country: undefined

}

$scope.addDestination = function() {

$scope.destination.push(

{

city: $scope.newDestination.city,

country.$scope.newDestination.country

)

}

});

**így néz ki a tesztfile:**

describe(’name’, function() {

beforeEach(module(’testingAngularApp’)

describe(’test of controller’, function(

var scope, ctrl;

beforeEach(inject(function($controller, $rootScope)) {

scope = $rootscope.$new();

ctrl = $controller(’controller name’, {$scope: scope} )

it (’name of unit test’, function() {

expect(scope.title).toBeDefined();

expect(scope.title).toBe(’name’ );

it (’adjon hozzá 2 új desztincáiót’, function () {

expect(scope.destinations).toBeDefined();

expect(scope.destinations.length).toBe(0);

scope.newDestination = {

city=”vmi”,

country = „vmi”

}

scope.addDestination;

expect(scope.destinations.length).toBe(1);

expect(scope.destinations[0].city)toBe(’vmi’);

expect(scope.destinations[0].country)toBe(’vmi’);

supertest

it(’vmi’, function(done) {

supertest(app)

.get(’vmi’)

.expect(200)

.end(function(err, res) {

res.status.should.equal(200);

done();

})

});

indítáas: mocha ...js

# Node passport

passportjs.org/docs

authentication middleware for Node to authenticate requests

Passport delegates all other functionality to the application

single sign-on using an **OAuth** provider such as Facebook or Twitter has become a popular authentication method. Services that expose an API often require token-based credentials to protect access.

Authentication mechanisms, known as strategies, are packaged as individual modules

app.post('/login', passport.authenticate('local', { successRedirect: '/', failureRedirect: '/login' }));

installing: npm install passport

## Authenticate

passport.authenticate() and specifying which strategy to employ.

app.post('/login',

passport.authenticate('local'),

function(req, res) {

// If this function gets called, authentication was successful.

// `req.user` contains the authenticated user.

res.redirect('/users/' + req.user.username);

});

if authentication fails, Passport will respond with a 401 Unauthorized status

**Redirect**: A redirect is issued after authenticating a request. Upon successful authentication, the user will be redirected to the home page. If authentication fails, the user will be redirected back to the login page for another attempt.

Redirects are often combined with **flash messages** in order to display status information to the user (failureFlash: true) - instructs Passport to flash an error message using the message given by the strategy's verify callback, if any.

flash message can also be set specifically: failureFlash: 'Invalid username or password.'

Using flash messages requires a req.flash() function. (connect-flash middleware)

**Disable Sessions**

After successful authentication, Passport will establish a persistent login session. for users accessing a web application via a browser.

However, in some cases, session support is not necessary. For example, API servers typically require credentials to be supplied with each request. When this is the case, session support can be safely disabled by setting the *session option to false*.

app.get('/api/users/me',

passport.authenticate('basic', { session: false }),

function(req, res) {

res.json({ id: req.user.id, username: req.user.username });

});

**Custom Callback**

can be provided to allow the application to handle success or failure.

app.get('/login', function(req, res, next) {

passport.authenticate('local', function(err, user, info) {

if (err) { return next(err); }

if (!user) { return res.redirect('/login'); }

req.logIn(user, function(err) {

if (err) { return next(err); }

return res.redirect('/users/' + user.username);

});

})(req, res, next);

});

If authentication failed, user will be set to false. If an exception occurred, err will be set. An optional info argument will be passed, containing additional details provided by the strategy's verify callback.

## Configure

Three pieces need to be configured to use Passport for authentication:

**authentication strategies**

pl. verifying a username and password vagy delegated authentication using OAuth vagy federated authentication using OpenID

via the use() function

Strategies require a **Verify Callback** to find the user that possesses a set of credentials

If the credentials are valid, the verify callback invokes done to supply Passport with the user that authenticated

return done(null, user);

if not valid: return done(null, false);

additional info message: return done(null, false, { message: 'Incorrect password.' });

if an exception occurred: return done(err);

**application middleware**

In a Connect or Express-based application, passport.initialize() middleware is required to initialize Passport. If your application uses persistent login sessions, passport.session() middleware must also be used.

app.configure(function() {

app.use(express.static('public'));

app.use(express.cookieParser());

app.use(express.bodyParser());

app.use(express.session({ secret: 'keyboard cat' }));

app.use(passport.initialize());

app.use(passport.session());

app.use(app.router);

});

**sessions (optional)**

If authentication succeeds, a session will be established and maintained via a cookie set in the user's browser.

Each subsequent request will not contain credentials, but rather the unique cookie that identifies the session.

In order to support login sessions, Passport will serialize and deserialize user instances to and from the session.

passport.serializeUser(function(user, done) {

done(null, user.id);

});

passport.deserializeUser(function(id, done) {

User.findById(id, function(err, user) {

done(err, user);

});

});

When subsequent requests are received, this serialized ID is used to find the user, which will be restored to req.user.

## Username & Password

most widely used way for websites to authenticate users

passport-local module:

**$ npm install passport-local**

var passport = require('passport')

, LocalStrategy = require('passport-local').Strategy;

passport.use(new LocalStrategy(

function(username, password, done) {

User.findOne({ username: username }, function(err, user) {

if (err) { return done(err); }

if (!user) {

return done(null, false, { message: 'Incorrect username.' });

}

if (!user.validPassword(password)) {

return done(null, false, { message: 'Incorrect password.' });

}

return done(null, user);

});

}

));

username and password arguments are submitted to the application via a login form.

<form action="/login" method="post">

<div>

<label>Username:</label>

<input type="text" name="username"/>

</div>

<div>

<label>Password:</label>

<input type="password" name="password"/>

</div>

<div>

<input type="submit" value="Log In"/>

</div>

</form>

**Route:** login form is submitted to the server via the POST method. Using authenticate() with the local strategy

app.post('/login',

passport.authenticate('local', { successRedirect: '/',

failureRedirect: '/login',

failureFlash: true })

);

**Parameters**

passport.use(new LocalStrategy({

usernameField: 'email',

passwordField: 'passwd'

},

function(username, password, done) {

// ...

}

));

# Local storage

Using [local storage](http://hacks.mozilla.org/2009/06/localstorage/) in modern browsers is ridiculously easy. All you have to do is modify the localStorage object in JavaScript. You can do that directly or (and this is probably cleaner) use the setItem() and getItem() method:

localStorage.setItem('favoriteflavor','vanilla');

If you read out the favoriteflavor key, you will get back “vanilla”:

var taste = localStorage.getItem('favoriteflavor');

// -> "vanilla"

To remove the item, you can use the removeItem() method:

localStorage.removeItem('favoriteflavor');

var taste = localStorage.getItem('favoriteflavor');

// -> null

That’s it! You can also use sessionStorage instead of localStorage if you want the data to be maintained only until the browser window closes.

 you can only store strings in the different keys!

localStorage.setItem( 'car', JSON.stringify(car) );

<https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API>

# Green Fox final exam

<https://docs.google.com/presentation/d/1SSnRtFDvfTei56dUs6bwkxXTAT8lLcyQ36s1M5vW2NU/edit?usp=sharing>

The student, on their own, is able to:

* Follow indentation and coding styleguides in their code
* Use and set up linters

**$ npm i --save-dev eslint [eslint-plugins]**

eslint --init

* Create variables, functions, objects in JavaScript / Java based on a given specification

**When the JavaScript parser sees a function in the main code flow, it assumes Function Declaration.**

**When a function comes as a part of a statement, it is a Function Expression.**

**Function Expressions can be used only after they are executed.**

* Use conditions and loops:
  + Filtering

var evens = [1, 2, 3, 4, 5].filter(function(e) {

return e % 2 === 0;

});

* + Finding elements or index

console.log(['apple', 'pear', 'melon'].indexOf('pear')); // 1

var triples = [1, 2, 3, 4, 5].map(function(e) {

return e \* 3;

});

console.log(triples); // [3, 6, 9, 12, 15]

['apple', 'pear', 'melon'].forEach(function(e) {

console.log(e);

});

* + Generating collections

arrays: concat, join, push, pop, shift, unshift, splice, reverse, sort

* + Checking mutiple values and counting

var isAllEven = [2, 6, 14, 5, 4].every(function(e) {

return e % 2 === 0;

});

console.log(isAllEven); // false

var isAnyEven = [2, 6, 14, 5, 4].some(function(e) {

return e % 2 === 0;

});

console.log(isAnyEven); // true

* Determine the output of expressions
* Explain their own code
* Install packages via a package manager

npm init, npm install

* Write unit tests for functions and methods without mocking dependencies

tape, karma

falsy:

false

0 (zero)

"" (empty string)

null

undefined

NaN (a special Number value meaning Not-a-Number!)

**var tape = require('tape');**

**var calculateDueDate = require('./duedate');**

**tape('dueDateOnSameDay', function(t) {**

**t.deepEqual(calculateDueDate('2016-08-22 09:30', 2), '2016-08-22 11:30');**

**t.end();**

**});**

* Include and call 3rd party libraries in their code

**var moment = require('moment');**

* Use frameworks and know its components and processes
* Use async structures:
  + Writing functions that are taking and calling callbacks
  + Determining the calling order in nested callbacks
* Use git:
  + Commit, add, push, pull
  + Resolving a simple merge conflict

the number of planets are

<<<<<<< HEAD

nine

=======

eight

>>>>>>> branch-a

* + Creating a pull request
* Use databases:
  + Implement database calls with JavaScript / Java
  + Write a simple SQL query for a single table:
    - Listing, Deleting, Updating, Creating rows with specific values

INSERT INTO mytable

(column, another\_column, …)

VALUES (value\_or\_expr, another\_value\_or\_expr, …),

(value\_or\_expr\_2, another\_value\_or\_expr\_2, …),

…;

UPDATE mytable

SET column = value\_or\_expr,

other\_column = another\_value\_or\_expr,

…

WHERE condition;

DELETE FROM mytable

WHERE condition;

**Order of execution**

Complete SELECT query

SELECT DISTINCT column, AGG\_FUNC(column\_or\_expression), …

FROM mytable

JOIN another\_table

ON mytable.column = another\_table.column

WHERE constraint\_expression

GROUP BY column

HAVING constraint\_expression

ORDER BY column ASC/DESC

LIMIT count OFFSET COUNT;

* Use the command line:
  + Traversing directories
  + Moving, copying, creating, deleting files

cp, mv, rm, rm –r

* + Passing command line arguments to commands

ezzel vittük be a todo-kat

sys module provides access to any command-line arguments via the sys.argv:

* sys.argv is the list of command-line arguments.
* len(sys.argv) is the number of command-line arguments.

getopt.getopt method: parses command line options and parameter list

* Write backend code:
  + Implement an HTTP endpoint

An endpoint is a service that can listen for requests.

app.**post**('/meals', function(req, res) {

myMeals.addMeal(req.body, function(result) {

res.send(result);

});

});

* Write frontend code:
  + Create simple layout in CSS and HTML based on a given specification
  + Select any HTML element using CSS selectors
  + Handle events and change the DOM

**querySelector, createElement, appendChild, removeChild, innerHTML, getAttribute, setAttribute**

**button.addEventListener('click', alertGreenFox);**

**var mealName = document.createElement('td');**

**mealName.innerHTML = item.name;**

**meal.appendChild(mealName);**

**mealName.classList.add('name\_col');**

**c.classList.add('x');**

* + Change attributes and properties on HTML elements using JavaScript
* Deploy application to hosting service

<https://devcenter.heroku.com/articles/getting-started-with-nodejs#set-up>

<https://devcenter.heroku.com/articles/deploying-nodejs>

* Differentiate between backend and frontend functionality
* Refactor code:
  + Create meaningful names for functions, classes and variables
  + Split bigger functions to smaller ones
  + Separate functions to different files
  + Point out code smells

prototype:

const waffle = Object.create(food)

waffle.init(’waffle’)

# jquery

for interacting with html

<https://jquery.com/download/>

$(document).ready(function() {

alert(„jquery loaded”)

});

element kiválasztás pl.: $(„#main-heading”)

de ez egy jquery object lesz (úgy néz ki mint egy array)

has access to all jquery methods and properties

## selectors

$(„#lead-banner p”)

$(„h3”)

$(„.wrapper”) – class ponttal

módosítás:

$(„h3”)**.css**({border: „3px solid red”})

## filters

to refine selectors (like pseudo classes)

„:”-tal elválasztva, pl. ($(„header nav li:first-child”)

:even, :odd

section:not(’#contact’) – minden a section-ből kivéve az

lt(3) – less than – az első 3-at adja vissza

$(„div[class]).css({valami}) – every div that has a class (attribute filter)

## traversing DOM

$(„valami”).next() vagy .prev() – azonos szinten lévő köv vagy előző elem

.parent

.children

.find(„.facebook”) – ha van ilyen class, megkeresi. performance szempontból lehet jó

.closest(.valami)

## chaining

$(„valami”).css(({color: vmi}).next.css(vmi)

végére ;

## adding content

script.js-be:

egy változóba beladott egy tag-et

aztán megfog egy elementet:

$.(„#tweets”).append(változó) – adds to the bottom of element

.prepend() – top of element

.before, .after, .html – changes the whole html, .text() – changes text

## wrap & unwrap

wrapped – it has parents

wrap() – all elements individually

unwrap()

wrapAll() – mindet egybe

$(„section”).wrap(„<div>”) – becsomagolja egy div-be

$(„section”).unwrap();

**másik példa:**

var wrapper = „<div class=’wrapper’>”;

var button = $(„.button”);

var wrapped = true;

button[0].oncklick = function() {

if(wrapped) {

$(„section”).unwrap();

wrapped = false;

button.text(„wrap”);

} else {

$(„section”).wrapAll(wrapper)

wrapped = true;

button.text(„unwrap”);

}

}

## removing

.empty() – empties inner HTML

.remove() – removes element

pl. $(„.button”).empty();

## change attributes

.removeAttr() – removes an attribute

.attr() – reads or sets an attribute

pl. $(„.vmi”).(„alt”, „location”) – első tag az attribute, 2. az érték amit kap

ha nincs 2., akkor csak beolvassa a jelenlegi értéket

## ccs with jQuery

$(„.valami”).css(„position”); - megnézi mi a position-je az elemnek

$(„.valami”).css(„top”, „-200px”); - módosítja

chain-elni is lehet

objektummal is lehet, átláthatóbb:

$(„.valami”).css({

„top” :, „-200px”,

„left” : „30px”

});

## add & remove classes

removeClass()

addClass()

toggleClass

$(„header .wrapper”).removeClass(„wrapper”);

$(„header > div”) – direct child

## event binding

on() – binds event to matched elements

off() – unbinds event from matched elements

var myList = $(„vmi”)

myList.on(„click”, function() {

$(this).css({ „color” : „pink” })

myList.off(„click”);

});

## event helpers

element.click(function() {vmi} );

## doc ready & window load events

hogyha head-ben van a script, akko js-ben be kell írni, hogy:

$(document).on(„ready”, function() {

ugyanez helyett elég az is, hogy $(function(

másik, az egész webpage betöltését megvárja: $(window).on(„load”, function() vagy $(window).load(function,

## event object

.stopPropagation() – stop bubbling up

.target() – brings back the element i clicked

## animations

click-re más attribute-ot kap

.animate(vmi, 4000) – sebesség

csak olyat lehet animálni ami számként van meghatározva

fading in & out: (animate(opacity) helyett): .fadeOut() – a végén eltűnik

.fadeOut().fadeIn() – eltűnik, megjelenik

.fadeTo(time, opacity)

## hide / show

hide(), show(), toggle()

$(this).hide(1000) – 1 mp-re eltűnik

## sliding

modifies height of element

$(„.valami”).slideUp(); - felcsúszik addig amit megadtunk – speed, duration megadható

.slideDown – vissza

## plugins

responsiveslides.com

# Functional programming

higher order functions: take functions as parameters or return a function

Functional code is characterised by one thing: the absence of side effects.

It doesn’t rely on data outside the current function, and it doesn’t change data that exists outside the current function.

simple map: name\_lengths = map(len, ["Mary", "Isla", "Sam"]) *// [4, 4, 3]*

simple lambda: squares = map(lambda x: x \* x, [0, 1, 2, 3, 4]) *// [0, 1, 4, 9, 16]*

simple reduce: sum = reduce(lambda a, x: a + x, [0, 1, 2, 3, 4]) *// 10*

sam\_count = reduce(lambda a, x: a + x.count('Sam'),

sentences,

0)

other ones: filter, all, any and find.

# Clean code

## Naming

Use Intention-Revealing Names

Beware of using names which vary in small ways

Avoid Disinformation

If names must be different, then they should also mean something different. Distinguish names in such a way that the reader knows what the differences offer.

make your names pronounceable.

Use Searchable Names - The length of a name should correspond to the size of its scope!

Avoid Mental Mapping

Classes and objects should have noun or noun phrase names like Customer, WikiPage, Account, and AddressParser. Avoid words like Manager, Processor, Data, or Info in the name of a class. A class name should not be a verb.

Methods should have verb or verb phrase names like postPayment, deletePage, or save.

## Functions

they should be small.

FUNCTIONS SHOULD DO ONE THING. THEY SHOULD DO IT WELL. THEY SHOULD DO IT ONLY.

One Level of Abstraction per Function

**stepdown rule:** We want the code to read like a top-down narrative. We want every function to be followed by those at the next level of abstraction so that we can read the program, descending one level of abstraction at a time as we read down the list of functions

Pl.:

*To include the setups and teardowns, we include setups, then we include the test page content,*

*and then we include the teardowns.*

*To include the setups, we include the suite setup if this is a suite, then we include the*

*regular setup.*

*To include the suite setup, we search the parent hierarchy for the “SuiteSetUp” page*

*and add an include statement with the path of that page.*

*To search the parent. . .*

Use Descriptive Names

a function shouldn’t have more than 3 arguments

Prefer Exceptions to Returning Error Codes

## Comments

Comments Do Not Make Up for Bad Code

todo comments

A comment may be used to amplify the importance of sg that may seem inconsequential.

## Formatting

the team should agree to a single set of formatting rules and all members should comply

it’s possible to build significant systems out of files that are typically 200 lines long

The Newspaper Metaphor

**Blank lines:** each blank line is a visual cue that identifies a new and separate concept. but lines of code that are tightly related should appear vertically dense.

**Variables** should be declared as close to their usage as possible. Instance variables should be declared at the top of the class

**Dependent Functions.** If one function calls another, they should be vertically close, and the caller should be above the callee

no wider lines than 120 characters

## Objects and Data Structures

the difference between objects and data structures: objects hide their data behind abstractions and expose functions that operate on that data. Data structure expose their data and have no meaningful functions.

Law of Demeter: method f of a class C should only call the methods of these: C, an object created by f, an object passed as an argument to f, an object held in an instance variable of C

Data Transfer Objects

Active Records are data structures with public variables

**conclusion**:

*Objects* expose behavior and hide data. This makes it easy to add new kinds of objects without changing existing behaviors. It also makes it hard to add new behaviors to existing objects.

*Data structures* expose data and have no significant behavior. This makes it easy to add new behaviors to existing data structures but makes it hard to add new data structures to existing functions.

## Error Handling

Use Exceptions Rather Than Return Codes

Try-Catch-Finally Statement: When you execute code in the try portion of a try-catch-finally statement, you are stating that execution can abort at any point and then resume at the catch.

When we’ve caught the exception, we can refactor. We can narrow the type of the exception we catch to match the type that is actually thrown

Each exception that you throw should provide enough context to determine the source and location of an error

wrapping third-party APIs is a best practice

## Boundaries

third party code

we could write some tests to explore our understanding of the third-party code (learning tests)

## Unit tests

The Three Laws of TDD:

1. You may not write production code until you have written a failing unit test.
2. You may not write more of a unit test than is sufficient to fail, and not compiling
3. You may not write more production code than is sufficient to pass the currently failing test.

keep tests clean (readible), so you can easily change them

the number of asserts in a test ought to be minimized AND we want to test a single concept in each test function

five other rules **(F.I.R.S.T.):** fast, independent (Tests should not depend on each other), repeatable, self-validating (tests should have a boolean output), timely (tests should be written just before the production code that makes them pass)

## Classes

should be small & we should be able to write a brief description of the class in about 25 words, without using the words “if,” “and,” “or,” or “but.”

*Single Responsibility Principle*: a class or module should have only one reason to change

Cohesion! A class in which each variable is used by each method is maximally cohesive. Maintaining Cohesion Results in Many Small Classes.

## Systems

Separate Constructing a System from Using It

Dependency Injection: an object should not take responsibility for instantiating dependencies itself. Instead, it should pass this responsibility to another “authoritative” mechanism

The class takes no direct steps to resolve its dependencies; it is completely passive. Instead, it provides setter methods or constructor arguments (or both) that are used to inject the dependencies.

## Emergence

a design is “simple” if it follows these rules:

* + Runs all the tests
  + Contains no duplication
  + Expresses the intent of the programmer
  + Minimizes the number of classes and methods

## Concurrency

is a decoupling strategy. It helps us decouple what gets done from when it gets done

the application looks like many little collaborating computers rather than one big main loop

Single Responsibility Principle: a given method/class/component should have a single reason to change.

# React

<https://scotch.io/tutorials/learning-react-getting-started-and-concepts>

performs on the client side & can be rendered server side, and they can work together inter-operably.

**Virtual DOM:** selectively renders subtrees of nodes based upon state changes, runs a “diffing” algorithm, which identifies what has changed, and updates the DOM with the results of diff.

## PAGE SETUP

include react.js, react-dom.js and JSXTransformer.js, and then write your component in a script node with type set to text/jsx

<html>

<head>

<script src="build/react.js"></script>

<script src="build/react-dom.js"></script>

<script src="build/JSXTransformer.js"></script>

</head>

<body>

<div id="mount-point"></div>

<script type="text/jsx">

// React Code Goes Here

</script>

</body>

</html>

## basics

React’s basic building blocks are called components

<script type="text/jsx">

ReactDOM.render(

<h1>Hello, world!</h1>,

document.getElementById('myDiv')

);

</script>

JSX: Javascript XML syntax transform - lets you write HTML-ish tags in your Javascript

For regular html tags, the class attribute is **className** and the for attribute is **htmlFor** in JSX

syntax without JSX:

React.render(

React.createElement('h1', null, 'Hello!'),

document.getElementById('myDiv')

);

## COMPONENTS

our first argument is the component we want to render, and the second is the DOM node it should mount to

We can use the createClass method to create custom component classes:

var MyComponent = React.createClass({

render: function(){

return (

<h1>Hello, world!</h1>

);

}

});

After creating a class we can render it to our document:

ReactDOM.render(

<MyComponent/>,

document.getElementById('myDiv')

);

## PROPS

When we use our defined components, we can add attributes called props.

These attributes are available in our component as this.props and can be used in our render method to render dynamic data

var MyComponent = React.createClass({

render: function(){

return (

<h1>Hello, {this.props.name}!</h1>

);

}

});

ReactDOM.render(<MyComponent name="Handsome" />, document.getElementById('myDiv'));

rendering only works with a single node

## Specs, lifecycle & state

there are several lifecycle methods & specs we can use to make the component do sg

LIFECYCLE METHODS

* componentWillMount – Invoked once, on both client & server before rendering occurs.
* componentDidMount – Invoked once, only on the client, after rendering occurs.
* shouldComponentUpdate – Return value determines whether component should update.
* componentWillUnmount – Invoked prior to unmounting component.

SPECS

* getInitialState – Return value is the initial value for state.
* getDefaultProps – Sets fallback props values if props aren’t supplied.
* mixins – An array of objects, used to extend the current component’s functionality.

STATE

Every component has a state object and a props object.

State is set using the setState method.

Calling setState triggers UI updates and is the bread and butter of React’s interactivity.

If we want to set an initial state before any interaction occurs we can use the getInitialState method.

Below, see how we can set our component’s state:

var MyComponent = React.createClass({

getInitialState: function(){

return {

count: 5

}

},

render: function(){

return (

<h1>{this.state.count}</h1>

)

}

});

## Events

events are attached as properties of components and can trigger methods

var Counter = React.createClass({

incrementCount: function(){

this.setState({

count: this.state.count + 1

});

},

getInitialState: function(){

return {

count: 0

}

},

render: function(){

return (

<div class="my-component">

<h1>Count: {this.state.count}</h1>

<button type="button" onClick={this.incrementCount}>Increment</button>

</div>

);

}

});

ReactDOM.render(<Counter/>, document.getElementById('mount-point'));

## Unidirectional Data Flow

application data flows unidirectionally via the state and props objects, as opposed to the two-way binding of libraries like Angular.

a common parent component manages the state and pass it down the chain via props.

state should be updated using setState method to ensure that a UI refresh will occur

The resulting values are passed down to child components using attributes that are accessible in said children via this.props.

## Egghead

install babel, webpack & webpack dev server

npm install react react-dom –save

npm install babel-loader babel-core babel-preset-es2015 babel-preset-react

kellenek ezek a file-ok: index.html, main.js, App.js, webpack.config.js

Webpack config is the file that's going to compile all of our JavaScript and our JSX down to JavaScript, and it's also going to launch our development server.

a script called, "start," is simply going to run Webpack dev server - run **npm start**

### components

App.js: in the extension of React.component, we are always going to have a render method. A render method is expected to return other elements or components.

we're going to use JSX, which is an HTML-like syntax. Here, inside this H1, I'm just going to say, "Hello world." Then export, by default, our app component.

Now the app component is rendering an H1 with an innerHTML of "Hello world." the H1 ends up getting passed to this function, called "React.createElement."

The first argument there is the element we want to create. In our case, it's an H1. The second argument is for props, which we'll talk about later. The last argument can either be another element, or it can be just a string that's meant to be the innerHTML of our element.

another way we can create a component: a stateless function component: a constant variable called "app." That's going to be equal to a function that simply returns our JSX.

The render method of our React component is only allowed to return a single node!

### props

we can do rendering directly from app.js. We can pass values into our component using **props** or properties

App.propTypes = {

txt: React.PropTypes.string,

cat: React.PropTypes.number

}

aztán:

render() {

let txt = this.props.txt

return <h1>{txt}</h1>

}

curly brackets, and this.props

We can also define the property types that we're expecting by adding a property to our component called proptypes, and that's just going to be an object where each key of the object is the name of our property, and then we pass in the type from React that we're expecting.

on any of these proptypes we can tack on isRequired.

We can also define default properties:

App.defaultProps = {

txt: 'this is the default txt'

}

the property that we passed in is actually overriding that, if there’s nothing, we’ll get our default property value.

### state

Unlike props, which are meant to be passed into our component as static values or methods, state is a collection of values that's meant to be managed by our component itself.

To initialize that, we're going to create a constructor method:

constructor() {

super();

this.state = {txt: 'this is the state txt'}

}

how to access:

render() {

return (

<div>

<input type="text" onChange={this.update.bind(this)}/>

<h1**>{this.state.txt}**</h1>

</div>

To manage that state, we create a custom method called update. It's going to take in an event. We're going to call this.setState, which is how we set our new state. We pass in the value that we care about. In this case, it's going to be e.target.value.

update(e) {

this.setState({txt: e.target.value})

}

To trigger that update, we wrap this whole thing in a div so that we only have one node.

Our state is being updated.

### owner/ownee

When one component renders another component, this is the owner-ownee relationship, where the parent component is also called a composite component.

we create a new component, this is going to be stateless. It's going to be called widget. it's going to take in props.

Instead of this.update, it's going to be props.update, and instead of this.state.text, it's going to be props.text.

in our app component, we're going to create a widget using the JSX syntax. We're going to set our prop of text to this.state(text). We're going to have a value here called "update" equal to this.update.

we've got the exact same functionality, except now, if we want, we can go ahead and drop a bunch of these on the page. Now, as we type in each one, it's updating the state.text value of our parent component.

### refs

Refs are a way to reference an instance of a component from within our React application.

refs won't work with stateless function components. We're going to create a new component: a slider

we could do create an update red, update green, update blue and then just track those in our multiple update statements, but we want to keep tracking for our entire state in one update method: we're going to be updating red to something, green to something, and blue to something.

we're going to give one of these a ref value. This one is going to be red, this one will be green, and this one will be blue. To get at those values, we're going to need to reach out to the DOM. We'll import React DOM from React-DOM.

we're going to say React DOM.findDOMnode. We're going to pass in our refs, so this.refs.red.

### Accessing Child Properties

When we want to access the innerHTML or nested components of another component, we can use this.props.children

we create a button component. In here, we're going to return a regular button. To get at this innerHTML of our button tag, between the regular button tags we put **this.props.children**.

we can also access nested components through this.props.children.

### Component Lifecycle - Mounting Basics

when a component is added or removed from the DOM, this is called mounting and unmounting the component.

start with a simple component, we're going to have a constructor, we'll call super to get our context, we're going to set our state to a val of 0, we're also going to have an update method

update method is going to set our state of val to this.State.Val +1, and here in our render method we're simply going to return a button with an onClick of this.Update, then its innerHTML will be this.State.Val

when we click on it, it is increasing and the render method is being called.

class App extends React.Component {

constructor() {

super();

this.state = {val: 0};

this.update = this.update.bind(this);

}

update(e) {

this.setState({val: this.state.val + 1})

}

**componentWillMount**() {

console.log(’mounting!!’)

}

render() {

console.log('rendering!');

return <button onClick={this.update}> {this.state.val} </button>

}

}

first mounting lifecycle:

**componentWillMount**: when our component is fully prepped and guaranteed to make it into the DOM.

it will only get called once.

**componentDidMount**: after our component has been placed into the DOM.

the render method is continuously called, while the componentDidMount is only called once.

**componentWillUnmount**: when we are about to remove our component from the DOM

to illustrate this we're going to need to create a wrapper component to work with the DOM.

we're going to have two buttons one for mounting, one for unmounting.

class Wrapper extends React.Component {

constructor() {

super();

}

mount() {

ReactDOM.render(<App />, document.getElementById('a'))

}

unmount () {

ReactDOM.unmountComponentAtNode(document.getElementById('a'))

}

render () {

return (

<div>

<button onClick={this.mount.bind(this)}>Mount</button>

<button onClick={this.unmount.bind(this)}>Unmount</button>

<div id="a"></div>

</div>

)

}

}

### mounting usage

how we can use the component mounting lifecycle phases of our component.

start with componentWillMount. At this state, our component is ready to go into the DOM. We don't have access to the DOM, but we do have access to our state, as well as our props.

set a new value called "m," for multiplier, in our state, and then in our render method take our val and multiply it by our multiplier.

when I click on this it is multiplying by two. We were able to change the state before the component mounted.

componentWillMount() {

this.setState({m: 2})

}

render() {

console.log('rendering!');

return (

<button onClick={this.update}>

{this.state.val \* this.state.m}

</button>

)

}

In componentDidMount, we have access to our component in the actual DOM, so we could console.log ReactDOM.findDOMNode. When we mount here, we can see our actual HTML in the DOM.

componentDidMount() {

console.log(ReactDOM.findDOMNode(this));

}

In componentWillUnmount, we can clean up any processes we have running.

### updating

itt 5-ösével növekszik a gombon az érték, és kiírja, hogy előző körben mennyi volt

a state folyton változik, de nincs mindig renderelve

class App extends React.Component {

constructor() {

super();

this.update = this.update.bind(this);

this.state = {increasing: false}

}

update() {

ReactDOM.render(

<App val={this.props.val+1} />,

document.getElementById('app')

);

}

componentWillReceiveProps(nextProps) {

this.setState({increasing: nextProps.val > this.props.val})

}

**shouldComponentUpdate**(nextProps, nextState) {

return nextProps.val % 5 === 0;

}

render() {

console.log(this.state.increasing);

return (

<button onClick={this.update}>

{this.props.val}

</button>

)

}

**componentDidUpdate**(prevProps, prevState) {

console.log('prevProps', prevProps);

}

}

### Higher Order Components

I've got a really simple component: a button. When we click on in, it increments value

also a couple of life-cycle methods here. we want to give the same functionality to more than one component if we'd like to. create a function here "mixin."

It's going to take in an inner component. It's going to return class extends react.component. We're taking in a component. We're returning a new component.

create a new render method which is going to return our inner component. As a prop, we're going to pass in update as this.update. We're going to use the spread operator to pass in this.state

create a stateless component, call it, "button." It's going to take in props and out of button.

on the click event, we'll call props.update. Its inner HTML will be props.text-props.val.

Now, to mix in the functionality from our mixin function into this button, we're simply going to say, "let button mixed equal mixin" and pass in our button component.

in our app component, we'll wrap everything up in a div.

We'll pass in our button mixed component. We're going to add a text prop of button.

create another one. This will be our label component. Rather than the onClick, we'll say, "on MouseMove."

Now when we click on our button, the value increases. When we move over a label, the value also increases. They both have the exact same state and the exact same functionality.

let Mixin = InnerComponent => class extends React.Component {

constructor() {

super();

this.update = this.update.bind(this);

this.state = {val: 0}

}

update() {

this.setState({val: this.state.val + 1})

}

componentWillMount() {

console.log('will mount');

}

render() {

return <InnerComponent

update={this.update}

{...this.state}

{...this.props} />

}

componentDidMount() {

console.log('mounted');

}

}

const Button = (props) => <button

onClick={props.update}>

{props.txt} - {props.val}

</button>

const Label = (props) => <label

onMouseMove={props.update}>

{props.txt} - {props.val}

</label>

let ButtonMixed = Mixin(Button)

let LabelMixed = Mixin(Label)

class App extends React.Component {

render() {

return (

<div>

<ButtonMixed txt="Button"/>

<LabelMixed txt="Label"/>

</div>

);

}

### Composable Components

React components should be reusable as well as composable.

going to create some prop types

We've got min set to a number, max set to a number, step and value set to a number. We've got a property of label set to a string, and that's going to allow us to take this label right here out of the control of the user and put it into the control of the component.

We've got our update method which is our only required method and that has a type of function, and then finally we've got a type property

We've got min and max both set to a default of 0, step set to 1, val set to 0, label set to an empty string, and for a default we're going to use range.

some initial settings: our type, min, max, step, and default value are all set to the props that are passed into our component

Set our min = 0, our max = 255, our step = 1, our value = this.state.red

If we want to mess around and say this our step of 0.01 we can do that. That will step by .01.

So now we have a component that's much more reusable and much more composable.

### Dynamically Generated Components

generating components dynamically from a data set.

an initial state that has a key of data.

in my render method, what I want to do is return a table for the tbody. Inside of that tbody, have a row for each person.

I'm going to create this rows variable. It's going to be equal to this.state.data.map.

We'll return a new component called person row. We'll set that to person

we're going to create that person component. This will be a stateless component called person row. It takes in its props and returns a table row with a table dialogue inside of it. This first one will contain props.data.id. The second one will be props.data.name.

we need a prop of key. It needs to be unique. We set that to person.id.

class App extends React.Component {

constructor() {

super();

this.state = {data: [

{id: 1, name:”vmi”},

{id: 2, name:”vmi”},

{id: 3, name:”vmi”}

]}

}

render(){

stb

}

visszadja a táblázat adatait egy táblában

# JS – the good parts

A block is a set of statements wrapped in curly braces. Unlike many other languages,

blocks in JavaScript do not create a new scope, so variables should be defined at the

top of the function, not in blocks.

falsy values:

* false
* null
* undefined
* The empty string ''
* The number 0
* The number NaN

All other values are truthy

**Statements:**

* switch
* while
* for
* do
* try
* throw
* return
* break

**Expressions:** prefix, infix, invocation, refinement

**Literals:** notation for specifying new objects

## objects:

can be nested.

var stooge = {

"first-name": "Jerome",

"last-name": "Howard"

};

var flight = {

airline: "Oceanic",

number: 815,

departure: {

IATA: "SYD",

time: "2004-09-22 14:55",

city: "Sydney"

},

arrival: {

IATA: "LAX",

time: "2004-09-23 10:42",

city: "Los Angeles"

}

};

Retrieval:

stooge["first-name"] // "Joe"

flight.departure.IATA // "SYD"

The || operator can be used to fill in default values:

var middle = stooge["middle-name"] || "(none)";

var status = flight.status || "unknown";

Attempting to retrieve values from undefined will throw a TypeError exception. This can be guarded against with the && operator:

flight.equipment && flight.equipment.model // undefined

A value in an object can be updated by assignment.

stooge['first-name'] = 'Jerome';

Objects are passed around by reference. They are never copied

**Prototype:** Every object is linked to a prototype object from which it can inherit properties.

The prototype link is used only in retrieval. If we try to retrieve a property value from an object, and if the object lacks the property name, then JavaScript attempts to retrieve the property value from the prototype object. And if that object is lacking the property, then it goes to its prototype, and so on until the process finally bottoms out with Object.prototype.

**Enumeration**: The for in statement can loop over all of the property names in an object.

**Delete**: The delete operator can be used to remove a property from an object.

One way to minimize the use of global variables is to create a single global variable

for your application:

var MYAPP = {};

That variable then becomes the container for your application:

MYAPP.stooge = {

"first-name": "Joe",

"last-name": "Howard"

};

## Functions

Functions in JavaScript are objects.

Function objects are linked to Function.prototype

every function receives two additional parameters: this and arguments.

There is no runtime error when the number of arguments and the number of parameters do not

match. If there are too many argument values, the extra argument values will be ignored. If there are too few argument values, the undefined value will be substituted for the missing values

**Method Invocation:**

var myObject = {

value: 0,

increment: function (inc) {

this.value += typeof inc === 'number' ? inc : 1;

}

};

myObject.increment( );

document.writeln(myObject.value); // 1

**Function Invocation:**

var sum = add(3, 4); // sum is 7

When a function is invoked with this pattern, **this** is bound to the global object. This was a mistake in the design of the language. Had the language been designed correctly, when the inner function is invoked, this would still be bound to the this variable of the outer function. A consequence of this error is that a method cannot employ an inner function to help it do its work because the inner function does not share the method’s access to the object as its this is bound to the wrong value.

Fortunately, there is an easy workaround. If the method defines a variable and assigns it the value of this, the inner function will have access to this through that variable. By convention, the name of that variable is **that**:

// Augment myObject with a double method.

myObject.double = function ( ) {

var that = this; // Workaround.

var helper = function ( ) {

that.value = add(that.value, that.value);

};

helper( ); // Invoke helper as a function.

};

// Invoke double as a method.

myObject.double( );

document.writeln(myObject.getValue( )); // 6

**Constructor Invocation:** If a function is invoked with the new prefix, then a new object will be created with a hidden link to the value of the function’s prototype member, and this will be bound to that new object.

Functions that are intended to be used with the new prefix are called constructors. By convention, they are kept in variables with a capitalized name.

**Apply Invocation:**

The apply method lets us construct an array of arguments to use to invoke a function. It also lets us choose the value of this. The apply method takes two parameters. The first is the value that should be bound to this. The second is an array of parameters.

**Return**: When return is executed, the function returns immediately without executing the remaining statements. A function always returns a value. If the return value is not specified, then undefined

is returned.

**Exceptions**: The throw statement interrupts execution of the function.

The exception object will be delivered to the catch clause of a try statement:

**Augmenting Types:** by augmenting Function.prototype, we can make a method available to

all functions

JavaScript does not have a separate integer type, so it is sometimes necessary to extract just the integer part of a number. The method JavaScript provides to do that is ugly. We can fix it by adding an integer method to Number.prototype. It uses either Math.ceiling or Math.floor, depending on the sign of the number:

Number.method('integer', function ( ) {

return Math[this < 0 ? 'ceiling' : 'floor'](this);

});

document.writeln((-10 / 3).integer( )); // -3

JavaScript lacks a method that removes spaces from the ends of a string. That is an easy to fix:

String.method('trim', function ( ) {

return this.replace(/^\s+|\s+$/g, '');

});

document.writeln('"' + " neat ".trim( ) + '"');

**Scope**: controls the visibility and lifetimes of variables and parameters

block scope: all variables defined in a block (a list of statements wrapped with curly braces) are not visible from outside of the block

JavaScript has function scope: parameters and variables defined in a function are not visible outside of the function, and a variable defined anywhere within a function is visible everywhere within the function

therefore it is best to declare all of the variables used in a function at the top of the function body.

**Closure**:

inner functions get access to the parameters and variables of the functions they are defined within

Earlier, we made a myObject that had a value and an increment method. Suppose we wanted to protect the value from unauthorized changes.

the function has access to the context in which it was created. This is called closure.

**Module**: s a function or object that presents an interface but that hides its state and implementation

By using functions to produce modules, we can almost completely eliminate our use of global variables

in the last line we immediately invoke the function we just made with the () operator.

The general pattern of a module is a function that defines private variables and functions; creates privileged functions which, through closure, will have access to the private variables and functions; and that returns the privileged functions or stores them in an accessible place.

**Cascade**: Some methods do not have a return value. For example, it is typical for methods that set or change the state of an object to return nothing. If we have those methods return this instead of undefined, we can enable cascades. In a cascade, we can call many methods on the same object in sequence in a single statement.

**Curry**: allows us to produce a new function by combining a function and an argument:

**Memoization**: Functions can use objects to remember the results of previous operations, making it possible to avoid unnecessary work. This optimization is called memoization.

We will keep our memoized results in a memo array that we can hide in a closure.

## Inheritance

JavaScript is a prototypal language, which means that objects inherit directly from other objects.

**Pseudoclassical**: Instead of having objects inherit directly from other objects, an unnecessary level of indirection is inserted such that objects are produced by constructor functions.

In classical languages, class inheritance is the only form of code reuse. JavaScript has more and better options.

**Object Specifiers**: It sometimes happens that a constructor is given a very large number of parameters. This can be troublesome because it can be difficult to remember the order of the

arguments. In such cases, it can be much friendlier if we write the constructor to accept a single object specifier instead. That object contains the specification of the object to be constructed. So, instead of:

var myObject = maker(f, l, m, c, s);

we can write:

var myObject = maker({

first: f,

last: l,

state: s,

city: c

});

The arguments can now be listed in any order, arguments can be left out if the constructor is smart about defaults, and the code is much easier to read.

**Prototypal**: Prototypal inheritance is conceptually simpler than classical inheritance: a new object can inherit the properties of an old object

Once we have an object that we like, we can make more instances with the Object.create method. We can then customize the new instances

This is differential inheritance. By customizing a new object, we specify the differences from the object on which it is based.

**Functional**: we get no privacy. All properties of an object are visible. but we can apply the module pattern. We start by making a function that will produce objects.

The function contains four steps:

1. It creates a new object. There are lots of ways to make an object. It can make an object literal, or it can call a constructor function with the new prefix, or it can use the Object.create method to make a new instance from an existing object, or it can call any function that returns an object.

2. It optionally defines private instance variables and methods. These are just ordinary vars of the function.

3. It augments that new object with methods. Those methods will have privileged access to the parameters and the vars defined in the second step.

4. It returns that new object.

The functional pattern has a great deal of flexibility. It requires less effort than the pseudoclassical pattern, and gives us better encapsulation and information hiding and access to super methods.

**Parts**: We can compose objects out of sets of parts. For example, we can make a function that can add simple event processing features to any object. In this way, a constructor could assemble objects from a set of parts.

## Arrays

JavaScript provides an object that has some array-like characteristics

Every array has a length property. JavaScript’s array length is not an upper bound. If you store an element with a subscript that is greater than or equal to the current length, the length will increase to contain the new element.

Since JavaScript’s arrays are really objects, the delete operator can be used to remove elements from an array:

delete numbers[2];

Unfortunately, that leaves a hole in the array.

Fortunately, JavaScript arrays have a splice method.

numbers.splice(2, 1);

A common error in JavaScript programs is to use an object when an array is required or an array when an object is required. The rule is simple: when the property names are small sequential integers, you should use an array. Otherwise, use an object.

**Methods**: JavaScript provides a set of methods for acting on arrays. The methods are functions stored in Array.prototype.

Array.method('reduce', function (f, value) {

var i;

for (i = 0; i < this.length; i += 1) {

value = f(this[i], value);

}

return value;

});

By adding a function to Array.prototype, every array inherits the method.

JavaScript arrays usually are not initialized. If you ask for a new array with [], it will be empty. If you access a missing element, you will get the undefined value.

if you are implementing algorithms that assume that every element starts with a known value (such as 0), then you must prep the array yourself. JavaScript should have provided some form of an Array.dim method to do this, but we can easily correct this oversight.

The cells of an empty matrix will initially have the value undefined. If you want them to have a different initial value, you must explicitly set them. Again, JavaScript should have provided better support for matrixes. We can correct that, too.

## Regular Expressions

is the specification of the syntax of a simple language.

methods that work with regular expressions are regexp.exec, regexp.test, string.match, string.replace, string.search, and string.split.

## Methods

JavaScript includes a small set of standard methods that are available on the standard types.

concat, join, pop, push, reverse, shift, slice, sort, splice

unshift: like the push method except that it shoves the items onto the front of this array instead of at the end.

apply: invokes a function, passing in the object that will be bound to this and an optional array of arguments.

number: The toExponential method converts this number to a string in the exponential form.

toFixed method converts this number to a string in the decimal form.

toPrecision method converts this number to a string in the decimal form

**string methods:**

string.charAt(pos): returns the character at position pos in this string.

string.charCodeAt(pos)

string.concat(string…)

string.indexOf(searchString, position)

string.lastIndexOf(searchString, position)

string.localeCompare(that): compares two strings

string.match(regexp): matches a string and a regular expression

string.replace(searchValue, replaceValue)

string.search(regexp)

string.slice(start, end)

string.split(separator, limit)

string.toLowerCase( )

string.toUpperCase( )

String.fromCharCode(char…): produces a string from a series of numbers.

## Awful Parts

## Global Variables

There are three ways to define global variables. The first is to place a var statement outside of any function:

var foo = value;

The second is to add a property directly to the global object.

window.foo = value;

The third is to use a variable without declaring it. This is called implied global:

foo = value;

**reserved words**

The following words are reserved in JavaScript:

abstract boolean break byte case catch char class const continue debugger defaultdelete do double else enum export extends false final finally float for function goto if implements import in instanceof int interface long native new null package private protected public return short static super switch synchronized this throw throws transient true try typeof var volatile void while with

They cannot be used to name variables or parameters. When reserved words are used as keys in object literals, they must be quoted.

**parseInt**: a function that converts a string into an integer. It stops when it sees a nondigit, so parseInt("16") and parseInt("16 tons") produce the same result.

**new**: JavaScript’s new operator creates a new object that inherits from the operand’s prototype member, and then calls the operand, binding the new object to this.

If youforget to use the new operator, youinstead get an ordinary function call, and this is bound to the global object instead of to a new object. That means that your function will be clobbering global variables when it attempts to initialize the new members.

By convention, functions that are intended to be used with new should be given names with initial capital letters, and names with initial capital letters should be used only with constructor functions that take the new prefix.

**JSLint** expects that every statement be followed by ; except for for, function, if, switch, try, and while. JSLint does not expect to see unnecessary semicolons or the empty statement.

# Design patterns

A pattern is a reusable solution

Anti-Patterns: represent a lesson that has been learned

Creational Design Patterns: handling object creation mechanisms:

* Factory: makes an instance of several derived classes based on interfaced data or events.
* Abstract factory: Creates an instance of several families of classes without detailing concrete classes.
* Prototype: A fully initialized instance used for copying or cloning.
* Singleton: A class with only a single instance with global access points
* Builder: Separates object construction from its representation, always creates the same type of object.

Structural Design Patterns: concerned with object composition:

* Decorator: Dynamically add alternate processing to objects.
* Facade: A single class that hides the complexity of an entire subsystem.
* Flyweight: for efficient sharing of information that is contained elsewhere.
* Adapter: Match interfaces of different classes therefore classes can work together despite incompatible interfaces
* Proxy: A place holder object representing the true object
* Bridge: Separates an object's interface from its implementation so the two can vary independently

Behavioral Design Patterns: focus on improving communication between objects:

* Iterator: Sequentially access the elements of a collection without knowing the inner workings of the collection.
* Mediator: Defines simplified communication between classes to prevent a group of classes from referring explicitly to each other.
* Observer: A way of notifying change to a number of classes to ensure consistency between the classes.
* Visitor: Adds a new operation to a class without changing the class
* Interpreter: A way to include language elements in an application to match the grammar of the intended language.
* Template Method: Creates the shell of an algorithm in a method, then defer the exact steps to a subclass.
* Chain of Responsibility: passing a request between a chain of objects to find the object that can handle the request.
* Command: Encapsulate a command request as an object to enable, logging and/or queuing of requests, and provides error-handling for unhandled requests.
* Memento: Capture an object's internal state to be able to restore it later.
* State: Alter an object's behavior when its state changes
* Strategy: Encapsulates an algorithm inside a class separating the selection from the implementation

how to define a class in JS: <http://www.phpied.com/3-ways-to-define-a-javascript-class/>

## Creational Pattern

forms the basis for a number of the other design patterns

Each of the following options will create a new empty object:

var newObject = {}; // or

var newObject = Object.create(null); // or

var newObject = new Object();

four ways in which keys and values can then be assigned to an object:

*1. Dot syntax*

newObject.someKey = 'Hello World'; *// Write properties*

**var** key = newObject.someKey; *// Access properties*

*2. Square bracket syntax*

newObject['someKey'] = 'Hello World'; *// Write properties*

**var** key = newObject['someKey']; *// Access properties*

*3. Object.defineProperty*

Object.defineProperty(newObject, "someKey", {

value: "for more control of the property's behavior",

writable: **true**,

enumerable: **true**,

configurable: **true**

});

*// If the above feels a little difficult to read, a short-hand could*

*// be written as follows:*

**var** defineProp = **function** ( obj, key, value ){

config.value = value;

Object.defineProperty(obj, key, config);

}

*// Create a new empty object*

**var** man = Object.create(**null**);

*// Populate the object with properties*

defineProp( man, 'car', 'Delorean' );

defineProp( man, 'dob', '1981' );

defineProp( man, 'beard', **false** );

*4. Object.defineProperties*

Object.defineProperties(newObject, {

"someKey": {

value: "Hello World",

writable: **true**

},

"anotherKey": {

value: "Foo bar",

writable: **false**

}

});

As we will see a little later in the book, these methods can even be used for inheritance, as follows:

**var** driver = Object.create(man);

defineProp(driver, 'topSpeed', '100mph');

driver.topSpeed *// 100mph*

## Constructor Pattern

Constructors are used to create specific types of objects - they both prepare the object for use and can also accept parameters which the constructor uses to set the values of member variables when the object is first created

By simply prefixing a call to a constructor function with the keyword **'new'**, you can tell JavaScript you would like function to behave like a constructor and instantiate a new object with the members defined by that function

a very basic constructor may be:

**function** Car(model, year, miles) {

**this**.model = model;

**this**.year = year;

**this**.miles = miles;

**this**.toString = **function** () {

**return this**.model + " has done " + **this**.miles + " miles";

};

}

**var** civic = **new** Car("Honda Civic", 2009, 20000);

**var** mondeo = **new** Car("Ford Mondeo", 2010, 5000);

The above is a simple version of the constructor pattern but it does suffer from some problems

**Constructors With Prototypes:** multiple Car objects can be created which access the same prototype.

Car.prototype.toString = **function** () {

**return this**.model + " has done " + **this**.miles + " miles";

};

## Singleton Pattern

can be implemented by creating a class with a method that creates a new instance of the class if one doesn't exist. In the event f an instance already existing, it simply returns a reference to that object.

With JavaScript, singletons serve as a namespace provider which isolate implementation code from the global namespace so-as to provide a single point of access for functions.

simplest form:

**var** mySingleton = {

property1: "something",

property2: "something else",

method1: **function** () {

console.log('hello world');

}

};

If you wished to extend this further, you could add your own private members and methods to the singleton by encapsulating variable and function declarations inside a closure. Exposing only those which you wish to make public

**var** mySingleton = **function** () {

*// here are our private methods and variables*

**var** privateVariable = 'something private';

**function** showPrivate() {

console.log(privateVariable);

}

*// public variables and methods (which can access*

*// private variables and methods )*

**return** {

publicMethod: **function** () {

showPrivate();

},

publicVar: 'the public can see this!'

};

};

**var** single = mySingleton();

single.publicMethod(); *// logs 'something private'*

console.log(single.publicVar); *// logs 'the public can see this!'*

To save on resources, you can place the instantiation code inside another constructor function

## Module Pattern

Modules help in keeping the units of code for a project both cleanly separated and organized.

the Module pattern is used to further emulate the concept of classes in such a way that we're able to include both public/private methods and variables inside a single object, thus shielding particular parts from the global scope.

Within the Module pattern, variables or methods declared are only available inside the module itself thanks to closure. Variables or methods defined within the returning object however are available to everyone.

**var** testModule = (**function** () {

**var** counter = 0;

**return** {

incrementCounter: **function** () {

**return** counter++;

},

resetCounter: **function** () {

console.log('counter value prior to reset:' + counter);

counter = 0;

}

};

})();

*// test*

testModule.incrementCounter();

testModule.resetCounter();

When working with the module pattern, you may find it useful to define a simple template that you use for getting started with it. Here's one that covers namespacing, public and private variables:

**var** myNamespace = (**function** () {

**var** myPrivateVar = 0;

**var** myPrivateMethod = **function** (someText) {

console.log(someText);

};

**return** {

myPublicVar: "foo",

myPublicFunction: **function** (bar) {

myPrivateVar++;

myPrivateMethod(bar);

}

};

})();

in the module pattern, public parts of your code are able to touch the private parts, however the outside world is unable to touch the class's private parts

**Disadvantages**: when you wish to change visibility, you actually have to make changes to each place the member was used

You also can't access private members in methods that are added to the object at a later point

inability to create automated unit tests for private members

## Revealing Module Pattern

allows the syntax of your script to be fairly consistent – it also makes it very clear at the end which of your functions and variables may be accessed publicly

**var** myRevealingModule = (**function**(){

**var** name = 'John Smith';

**var** age = 40;

**function** updatePerson(){

name = 'John Smith Updated';

}

**function** setPerson () {

name = 'John Smith Set';

}

**function** getPerson () {

**return** name;

}

**return** {

set: setPerson,

get: getPerson

};

}());

*// Sample usage:*

myRevealingModule.get();

we simply define all of our functions and variables in the private scope and return an anonymous object with pointers to the private functionality we wished to reveal as public.

## Observer Pattern

allows an object (known as a subscriber) to watch another object (the publisher)

Subscribers are able to register (subscribe) to receive topic notifications from the publisher when something interesting happens

## Mediator Pattern

a behavioral design pattern that allows us to expose a unified interface through which the different parts of a system may communicate

If a system may have too many direct relationships between modules, it may be time to have a central point of control

## Prototype Pattern

creates objects based on a template of an existing object through cloning.

// Use Object.create to instantiate a new car

var yourCar = Object.create( myCar );

one is a prototype of the other

## Command Pattern

aims to encapsulate method invocation, requests or operations into a single object and gives you the ability to both parameterize and pass method calls around that can be executed at your discretion

## Facade Pattern

simplifying the API being presented to other developers

## Factory Pattern

another creational pattern concerned with the notion of creating objects

the Factory Pattern suggests defining an interface for creating an object where you allow the subclasses to decide which class to instantiate

it doesn't explicitly require us use a constructor.

When we need to easily generate different instances of objects depending on the environment we are in

## Mixin Pattern

mixins are classes which provide the functionality to be inherited by a subclass. Inheriting from mixins are a means of collecting functionality and classes may inherit functionality from multiple mixins through multiple inheritance.

## Decorator Pattern

a structural design pattern that promotes code reuse and is a flexible alternative to subclassing

also

useful for modifying existing systems where you may wish to add additional features to objects without the need to change the underlying code that uses them

**Subclassing**: refers to inheriting properties for a new object from a base or 'superclass' object

In traditional OOP, a class B is able to extend another class A. Here we consider A a superclass and B a subclass of A. As such, all instances of B inherit the methods from A. B is however still able to define its own methods, including those that override methods originally defined by A.

**Decorators** are used when it's necessary to delegate responsibilities to an object where it doesn't make sense to subclass it. A common reason for this is that the number of features required demand for a very large quantity of subclasses

The decorator pattern isn't heavily tied to how objects are created but focuses on the problem of extending their functionality. Rather than just using inheritance, where we're used to extending objects linearly, we work with a single base object and progressively add decorator objects which provide the additional capabilities.

rather than subclassing, we add (decorate) properties or methods to a base object

## Flyweight

a useful classical solution for code that's repetitive, slow and inefficient – like a large number of simliar objects

There are two ways in which the Flyweight pattern can be applied:

* on the data-layer, where we deal with the concept of large quantities of similar objects stored in memory.
* on the DOM-layer where the flyweight can be used as a central event-manager to avoid attaching event handlers to every child element in a parent container you wish to have some similar behavior.

## MVC: Model-View-Controller

isolation of business data (Models) from user interfaces (Views), with a third component (Controllers) traditionally managing logic and user-input

**Models** manage the data for an application, represent unique forms of data that an application may require.

**Views** are a visual representation of models that present a filtered view of their current state. A view typically observes a model and is notified when the model changes, allowing the view to update itself accordingly

Templating: JavaScript templating solutions (such as Handlebars.js and Mustache) are often used to define templates for views as markup (either stored externally or within script tags with a custom type - e.g text/template) containing template variables

**Controllers**: intermediary between models and views which are classically responsible for two tasks: they both update the view when the model changes and update the model when the user manipulates the view.

miért jó:

* + Easier overall maintenance.
  + more straight-forward to write unit tests for business logic
  + Duplication of low-level model and controller code is eliminated
  + allows developers responsible for core logic and developers working on the userinterfaces to work simultaneously

**MVP: Model-View-Presenter**

derivative of the MVC design pattern which focuses on improving presentation logic

Unlike MVC, invocations from the view are delegated to the presenter, which are decoupled from the view and instead talk to it through an interface. This allows for all kinds of useful things such as being able to mock views in unit tests.

presenter acts as a mediator which talks to both the view and model, however both of these are isolated from each other.

for applications with very complex views and a great deal of user interaction

increases the testability of your application and provides a more clean separation between the view and the model, but lack of data binding support in the pattern can often mean having to take care of this task separately

**MVVM: Model-View-ViewModel**

attempts to more clearly separate the development of user-interfaces (UI) from that of the business logic and behavior in an application

This facilitates UI and development work occurring almost simultaneously within the same codebase. UI developers write bindings to the ViewModel within their document markup (HTML), where the Model and ViewModel are maintained by developers working on the logic for the application.

Model in MVVM represents domain-specific data or information that our application will be working with. A typical example of domain-specific data might be a user account (e.g name, avatar, e-mail) or a music track (e.g title, year, album).

As with MVC, the View is the only part of the application of users actually interact with. They are an interactive UI that represent the state of a ViewModel. In this sense, MVVM View is considered active rather than passive, but what does this mean?.

acilitates easier parallel development of a UI and the building blocks that power it

Models hold information, but typically don’t handle behavior.

View is the only part of the application that users actually interact with

The ViewModel can be considered a specialized Controller that acts as a data converter.

# Steve Krug – Don’t make me think!

egyértelmű elenevezések

egyértelmű, hogy vmire rá lehet –e kattintani

a felhasználók fejében nem bukkanhatnak fel kérdőjelek

egyértelmű vizuális hierarchia, webes konbvenciók, elkülöníthető részek, legyen világos mire lehet kattintani, minimalizáljuk a zajt

fölösleges szövegek elhagyása (jópofizás, használati utasítás)

állandó navigáció elemei: logo, kezdőlap, keresés, menüpontok, segédeszközök

minden oldalnak saját név kell, név a megfelelő helyen

„ön itt áll” legyen jól látható

breadcrumbs

fülek (pl. Amazon)

kezdőlapon a navigáció kismértékben eltérhet a többi oldalon lévőtől

legördülő menü helytakarékos de veszélyes