zoom link: <https://us02web.zoom.us/j/5449580745?pwd=eEFtZUJxelIwOHpIMVkrV3NlYUdCdz09#success>

tanterem:

<https://classroom.google.com/c/NjMxNTE4NTAwOTE4>

felvételek:

<https://drive.google.com/drive/folders/1E8ZfmwKcUjuk4MOmgv00Kc8fX85aECXO>

diagram készítő oldal:

<https://app.diagrams.net/>

# 0. modul

## Basic of Basics

## Personal Computer

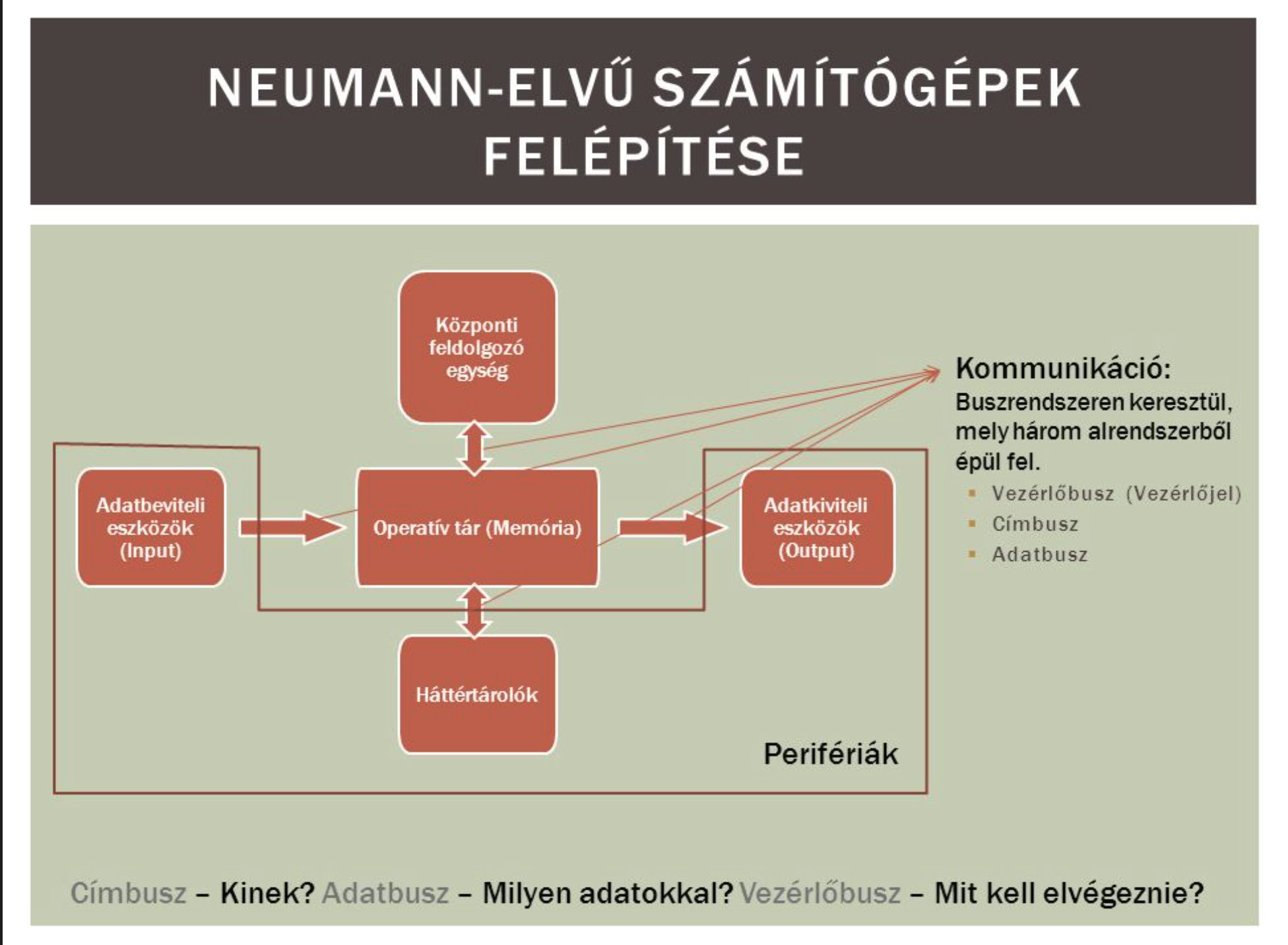
Bevezetés az IT-ba

## Mi az a PC?

Olyan számítógép, amely nem egy központi számítógép terminálja (munkaállomása), hanem önálló, egyetlen személy (az ún. végfelhasználó) által kezelt, kisebb méretű gép saját billentyűzettel, monitorral és minden egyéb mással ami működtetéséhez szükséges.

Az első elektronikusan működő számítógép, az ENIAC *( Electronic Numerical Integrator And Computer)* építési tapasztalatai alapján a számítógép építéséhez nélkülözhetetlen alapelveket Neumann János matematikus dolgozta ki, aki az ENIAC-nál gyorsabb, megbízhatóbb, egyszerűbb és könnyebben kezelhető gépet szeretett volna megépíteni.

* Soros utasítás-végrehajtás, az utasítások végrehajtása időben egymás után történjen *(ennek*
* Teljesen elektronikus működés *(ez Neumann idejében elektroncsöves felépítést jelentett, amit később a tranzisztoros, majd az integrált áramkörös felépítés követett)*
* Kettes számrendszer használata
* Adatok és programok ugyanabban a belső memóriában tárolódnak (tárolt program elv)
* Univerzális felhasználhatóság, Turing-gép *(programozhatóság; a különböző feladatok programokkal legyenek megoldva, nem pedig erre a célra épített hardverrel)*
* Szerkezet: öt funkcionális egység



Az 5 egység közötti kommunikáció buszrendszeren keresztül történik, melynek három alrendszere:  
- vezérlőbusz

- címbusz

- adatbusz

### Felépítése

#### Processzor - Központi feldolgozó egység (CPU - CENTRAL PROCESSING UNIT)

A számítógép "agya", ahol az összes munka végrehajtódik. Egy elektronikus eszköz, amely adatokat fogad és azokon különböző műveletek hajt végre. Ezek lehetnek számítási, vagy vezérlési műveletek.

#### Memória

A memória a számítógép emlékezete. Ide kerülnek a futó programok és az általuk használt adatok.

RAM: írható-olvasható memória. A számítógép kikapcsolása után törlődik a tartalma.

ROM: csak olvasható memória, tartalma nem változik a gép kikapcsolása után sem.

#### Perifériák

A számítógéphez csatlakoztatható olyan eszköz, ami nem egy további eszköz illesztését oldja meg. A perifériát a külvilággal történő kapcsolattartásra használja a számítógép.

**Bemeneti eszközök:** Ezeken keresztül vihetünk be adatokat a számítógépbe, adhatunk parancsot neki.

**Kimeneti eszközök:** Ezek segítségével tud „visszaüzenni” a felhasználónak a gép, ezeken keresztül jeleníthetjük meg adatainkat.

#### Háttértárolók

A háttértárak segítségével tárolhatjuk huzamosabb ideig az adatainkat, így azok a gép kikapcsolása után is megőrződnek. Optikai és mágneses.

0,1 bit -8x- bájt (B) -1024x kilobájt (kB) -1024x Megabájt (MB) -1024x Gigabájt (GB) -1024x Terrabájt (TB)

Ha veszek egy 1 TB-os merevlemezt akkor miért csak ~930GB szabad hely van rajta?

## Fogalmak

Hardver: A számítógép mechanikus, elektronikus és elektromechanikus alkotóelemeinek összessége. (Minden amit kézzel meg lehet fogni)

Szoftver: A hardvert vezérlő utasítások, programok, dokumentációk (adatok) összessége.

## Hálózatok

### Mi az a hálózat?

A számítógép-hálózat a számítógépek közötti kommunikációs rendszer.

### Kérdezz-feleltek

Miért érdemes több számítógépet összekapcsolni? Milyen érvek szólnak a hálózat kiépítése mellett?

### Hálózat kiépítéséhez szükséges eszközök

* hálózati kártya
* átviteli közeg - vezetékes (koax, UTP, optikai) vagy vezeték nélküli (mikrohullámú)
* hálózati kiegészítő eszközök
* hálózati szoftver

## Kiegészítő eszközök

Repeater: Nagy távolságok esetén jelismétlőkkel erősítik a jeleket, hogy azok ne haljanak el, ne vesszen el információ. Csak két portja van.

Hub: Több porttal rendelkező repeater. A jel bemegy az egyiket és kidobja az összesre. Többszereplős CB rádió. Ha egyszerre beszélnek ütközés --> véletlen ideig alvás majd újra próba

Switch: Intelligens HUB. Megnézi, kibontja a csomagot és megnézi ki a címzett. *(Csak neki küldi - MAC-address)* Folyamatosan tanul. Jancsi-Juliskával és Mariska-Józskával tud egyszerre kokettálni.

Bridge: Számítógépeket, számítógép hálózatokat köt össze. MAC-address alapján azonosít.

Router: Ugyanaz mint a bridge, csak IP cím alapján azonosít. *( Eggyel fentebbi OSI-szinten dolgozik.)* Google-> OSI model

## Hálózatok sebessége

A számítógép-hálózatok adatátviteli sebességének a mértékegysége bit/sec (vagy bps), ami az egy másodperc alatt átvitt bitek számát határozza meg.

A ping az az idő, amely során a testpacket visszatér a kiküldéstől számitva. [www.speedtest.net](http://www.speedtest.net)

## Hálózatok csoportosítása

### Földrajzi kiterjedés szerint

**PAN**: személyi hálózat, amely egyes embereknek szántak. Például egy vezeték nélküli hálózat, amely az egeret összeköti a számítógéppel.

**LAN:**  helyi hálózat, amely lehet egy intézmény vagy vállalat saját hálózata, **WLAN** - wireless+LAN

**MAN:** városi hálózatok -, ami összeköti a kis helyi hálózatokat

**WAN:** nagy kiterjedésű hálózatokról -, amely egy egész ország területétől egészen a Föld teljes egészéig terjedhet. Idetartozik az internet.

**intranet** - általában vállalaton belüli számítógépek zárt, hálózatos rendszere.

**extranet** - az intranet kibővített változata, amely korlátozott mértékben lehetőséget biztosít külső felhasználók számára a rendszer használatához (pl. üzleti partnerek).

### Topológia szerint

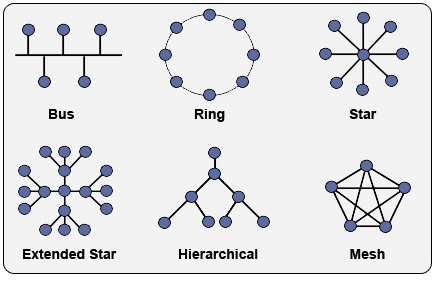
**Sín**: kevés kábel így olcsó viszont könnyen meghibásodik. Pl.: karácsonyfa izzó

**Gyűrű**: körbe épített sín. Ugyanaz mint a sín

**Csillag**: minden eszköz külön kábellel kapcsolódik a központi kiszolgáló géphez. Üzembiztosabb, egy kábel hibásodás miatt csak egy gép hal el. Viszont drágább.

**Fa**: hierarchikus

**Teljes**: minden eszköz minden eszközzel össze van kötve



## Protokoll

**A protokoll a hálózati kommunikációt leíró szabályok rendszere.** Protokollokat használnak a hálózatokban egymással kommunikáló számítógépek és programok is. Szabályok összessége a kommunikációra.

**TCP/IP**: az internet szabványosított, leggyakrabban használt kommunikációs protokolljainak az összessége. *(MAC address cím: egyedi azonosítója minden hálózati eszköznek)*

**SMTP**: kimenő levelezési protokoll

**POP3**: bejövő levelezési protokoll

**FTP**: hálózaton történő adatátviteli (fájl) protokoll

**HTTP**: információ átviteli protokoll

**SFTP** és **HTTPS**: a fenti két protokoll titkosított, autentikált változata

### Portszámok

Másnéven csatlakozó számok. A portszámok segítségével tudják megkülönböztetni a hálózaton egy időben folyó "beszélgetéseket". Egyidejűleg e-mail és böngészés.

FTP: 21, SFTP: 22, SMTP: 25, POP3: 110, HTTP: 80, HTTPS: 443,

## Szerver-kliens

**A szerver funkciója a hálózaton lévő számítógépek kiszolgálása.** Ez magába foglalhatja az adatok központi tárolását egyéni vagy közös felhasználás céljából, továbbá különféle szolgáltatások nyújtását a hálózati felhasználók számára. A számítógépes hálózatra csatlakoztatott minden számítógépet – a szerverek kivételével – munkaállomásnak, **kliens**nek nevezünk.

# 1. modul

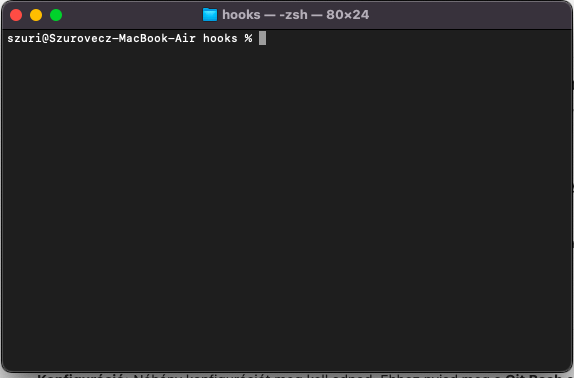
## Introduction to Basics

## Command line

Parancsok, tippek, trükkök

### Mi az a command line?

A parancssor *(command line, vagy command-line interface(CLI))* nevű ablak egy szöveges alapú alkalmazás, amit arra használhatsz, hogy fájlokat tekinthess meg, szerkeszthess, és különböző műveleteket végezhess rajtuk grafikus felület nélkül.



A sor elején lévő rész, a $, % vagy > karakterig *(azt is beleértve)* a parancssori prompt vagy röviden prompt a neve. Azt jelzi, hogy a számítógép a te utasításodra vár.

### Parancssor használat alapinfok

* A parancs kiadását követően az ENTER leütésével lehet a parancsot futtatni.
* A balra, jobbra kurzor navigációs billentyűkkel lehet beírt parancson belül balra jobbra vinni a kurzort
* A fel, le kurzor navigáció billentyűkkel lehet a legutóbb használt parancsok között lépkedni
* A tab billentyű használatával lehet a kiegészítési javaslatokat kérni

## Parancssor utasítások

**pwd** – aktuális könyvtár

**ls** – a könyvtár tartalma

**ls -l** : részletes lista, **ls -al** (vagy **ls –la**) – részletes lista a rejtett fájlokat is beleértve. (A megjelenő lista ’drwxr’ karaktereinek jelentése: ’d’ - directoy, a többi 4 a linux jogosultság karakterei. Azaz, megmutatja, hogy adott user tud e olvasni (r), írni (w), vagy végrehajtani (w) az adott folderben, vagy fájlban.)

**cd mappanév** – navigálás könyvtárba

**cd ..** – navigálás a szülő könyvtárba (azaz, egyel feljebb), (a cd . parancs az aktuális könyvtárat jelöli)

**mkdir mappanév** – könytár létrehozás (többet is létre lehet hozni, szóközökkel elválasztva)

**rm -r** **mappanév** – könytár törlése (a benne lévő tartalommal együtt)

**cp -r mappanév** **./célmappa/mappanév** – könyvtár másolása

**mv mappanév ./célmappa/mappanév** – könyvtár áthelyezése

**echo > fáljnév.kiterjesztés** – fájl létrehozása

**touch** **fáljnév.kiterjesztés** – fájl létrehozása (akár több is egyszerre: **touch** **a.txt b.txt c.txt**)

**echo szöveg > fáljnév.kiterjesztés** – fájl létrehozása, szöveges tartalommal

**rm fájlnév** – fájl törlése

**cp ./mappa/fáljnév.kiterjesztés ./célmappa/fáljnév.kiterjesztés** –fájl másolása

**mv ./mappa/fáljnév.kiterjesztés ./célmappa/fáljnév.kiterjesztés** –fájl áthelyezése

**cat** – fájl tartalmának megjelenítése

**|** – parancsok sorbafűzése (A | szimbólum előtti parancs kimenetét a | szimbólum utáni parancs bemenetére irányítja.)

**grep keresendő szöveg** – szöveg szűrése, keresése (önmagában nem működik, kell elé egy kimenet, hogy miben keressen – pl.: **ls |** **grep keresendő szöveg**

> – parancsok kimenetének fájlba irányítása (példa rá a fenti echo paranccsal)

>> – ugyanaz, de nem felülír, hanem hozzáfűz (példa rá a fenti echo paranccsal: nem felülírja a

szöveget, hanem hozzáadja a meglévőhöz)

**clear** – képernyő törlése

**history** – előzmények

**exit** – kilépés

**ctrl+c** – futó parancs megszakítása

**vim fájlnév.kiterjesztés** – a fálj szerkesztése a bash szövegszerkesztőjével (mentés és kilépés – **:wq**)

**whoami** - A felhasználói azonosítóhoz társított felhasználói nevet jeleníti meg

**tasklist** vagy **ps aux** – futó folyamatok (process lista) (le tudjuk ellenőrizni, hogy a saját alkalmazásunk fut-e, mennyi processzoridőt használ, stb.)

**ipconfig** és **ipconfig /all** vagy **ipconfig getifaddr en0** – hálózati kártya adatok lekérése

**ping szervernév** – szerver elérhetőségének vizsgálata (ip cím is elegenedő)

**nslookup szervernév** – szerver névfeloldása

**netstat** – gépünk hálózati forgalma (amit további parancsokkal lehet szűrni, pontosítani)

**open -a alkalmazásnév** – alkalmazás elindítása (nem működik, de simán csak az alkalmazásnévvel működhet)

Az összes parancs az összes oprendszerre: <https://ss64.com/>

Feladat: Navigáljunk a C (Helyi lemez) gyökerébe és a kilistázott tartalmat írassuk ki a képernyőre. Ha ez megy, akkor most a képernyő helyett az egész tartalmat írassuk ki egy fájlba. Ha ez is megvan, akkor csak azokat a tartalmakat írjuk bele egy másik fájlba, amiben szerepel az a kifejezés, hogy “Program”. Mindkét esetben, írassuk ki a fájlok tartalmát, hogy leellenőrizzük tényleg csak az szerepel benne amit mi szeretnénk.

Megoldás:

echo | ls > a.txt

cat a.txt

echo | ls | grep Program > b.txt

cat b.txt

## GIT

git, git bash

A git a legnépszerűbb verziókezelő szoftver.

### Tulajdonságai:

* több verzió tárolása
* visszaállás korábbi verziókra
* kód könnyű megosztása
* közös munka
* változások nyomon követése
* backup *(biztonsági mentés)*
* conflict kezelés *(ugyanazon sort módosították többen egyszerre a fájlban)*

#### Fogalmak:

**repository**: Az adatbázis neve, amely a verziókezelt fájlokat tárolja. Lényegében egy mappa a fájlokkal, amiben van egy speciális rejtett . git mappa, ami a kezeléshez szükséges adatbázist tárolja.

* **local repository:** saját gépünkről elért repository
* **remote repository:** távoli helyről (pl. GitHub) elért repository

**commit**: fájl vagy fájlok egy elmentett állapota, verziója

**branch**: commitok összessége

**push**: lokális változtatások, új dolgok bevezetésének feltöltése a szerverre

**pull**: szerveren tárolt fájlok letöltése a lokális repository-ba

**status**: megmutatja, mi az állapota a lokális repository-ban lévő fájloknak

**staging area/staged files**: lokális fájlok, amelyek ki vannak választva a következő commithoz



### Alap parancsok

**git config --global user.name 'NEVED'**

**git config --global user.email 'EMAIL\_CÍMED'** – ezzel a két sorral azonosít be a GitHub

(csak új gépen kell végrehajtani)

**git init** –Inicializál egy új üres local repository-t, megjelöli verziókövetésre a mappát,

vagy git klónt hozunk létre, pl.:

**git clone https://gitlab.com/FELHASZNALO/A\_REPOSITORYM.git** (klónozásnál nem kell még mappát sem létrehozni, csak onnan kell kiadjuk a parancsot, ahová szeretnénk a klónt, ill. inicializálásra sincs szükség)

**git branch -M main** – a fő branch átnevezése ’main’-re

**git add filenév** : verziókövetve lesz az adott file (zöld színű, előtte piros)

**git add .** : minden fájl verziókövetetté válik (git add . = git add -A (elvileg megyegyeznek,

Tamás szerint))

**git commit -m ’kommentár’** – a verziókövetett fájlokat (zöldek) commitálja (hozzáadja a

repositoryhoz) és kísérőszöveget (kommentárt) írunk mellé,

ami általában a változtatást írja le röviden

**git commit -a -m ’kommentár’** – a git add és a git commit utasítások egyben, de érdemes a

kettőt külön megtenni

**git status** – az adott állapot monitorozása (nem árt sűrűn alkalmazni)

**git remote** – a távoli repository neve

**git remote -v** – megmondja, hogy milyen connectstringen keresztül kapcsolódunk a

repositoryhoz

### A folyamat

- új remote repository létrehozása a GitHub-on

- egy mappa létrehozása a gépen, amely a local repository helye lesz

- a remote repositoryban is megtalálható parancsok lefuttatása:

**echo "# First-project" >> README.md**

**git init** (inicializálunk)

**git add README.md (**íratlan szabályként, szokásként a README.md szokta tartalmazni a

repository-ban a hasznos információkat, tudnivalókat.**)**

**git commit -m "first commit"**

**git branch -M main** (ez az utasítás azért kell, hogy a kódunk valamennyi változata ne a

„master”, hanem a „main” branchen legyen, mert a Githubon „main”

az alapértelmezett branch)

**git remote add origin https://github.com/Szabolcs888/First-project.git** (a local repo

összekapcsolása a remote repoval)

**git push -u origin main** (a local rep feltöltése a remote rep-ba)

És amint van mit lehívni:

**git pull origin main** (a remote rep-ban történt változások letöltése)

Egy local repositorynak CSAK EGY remote repositoryja lehet!

### Branchek

Meglévő repositoryba az alap branch (main) mellé új branch létrehozása:

**git branch feature/név utasítással kell kezdeni!**

**git branch feature/adam** (branch létrehozása)

**git checkout feature/adam** (a branchre való átváltás)

**git checkout -b feature/adam** (egy menetben megtörténik a fenti két utasítás)

**echo szöveg a fájlba > adam.md**

**git add .**

**git commit -m ’created adam fajl’ (vagy bármilyen leírás a fájlról)**

**git push -u origin feature/adam**

**git branch** (kilistázza a brancheket)

**git branch -a** (kilistázza a távoli brancheket is)

**git log** – naplót jelenít meg

### Merge és Rebase – a branchek összefésülése, összevonása

#### Merge

Ezzel a módszerrel egy meglévő branch változtatásait egy másik branchre vihetjük át. Amennyiben voltak a két ágon eltérő commit-ok, úgy egy merge commit fog létrejönni, ha nincs ütközés. (A branchről való mergelés előtt le kell töltenöünk a main legfrissebb állapotát!)

**git checkout main** (a main-re való visszaváltás)

**git merge --no-ff feature/adam -m 'merged feature/adam branch'** (a két branch

összevonása és commitolása (a mainből történik!))

**git push -u origin main** (ezzel felkerül a remote rep-ba és egy állapotba kerül a kettő. Ezután vagy kitörlik a leágazó branchet, vagy megtartják (cégfüggő).)

#### Rebase

A merge-hez képest itt az irányok lesznek az eltérőek (céges hozzáállás kérdése, hogy ki melyiket részesíti előnyben). A rebase esetében ugyanis azt mondjuk meg, hogy melyik branch változtatásait, commit-jait játssza újra az aktuális branch-en. Eközben az eredeti commit-ok eltűnnek és azok másolatai játszódnak újra esetünkben a main branch-en.

**git checkout feature/adam**

**git rebase main**

**git checkout main**

**git merge feature/adam -m 'merged feature/adam branch with rebase'**

Az történt,hogy az adam branchből (a **git rebase main** utasítással) rebaiseltem a maint (ahogy a példában is van) és a main változtatásai megjelentek adam-nál is, de a mainben változatlan maradt minden. (A **git rebase main** utasítással) gyakorlatilag frissítettük a bob branchet a main újabb adataival, de fordítva ez nem történt meg. Azután visszalépve a mainbe és kiadva a **git merge adam -m 'merged feature/adam branch with rebase'** parancsot már megegyezik az adam és a main branch. (A leírás lehet, hogy nem pontos, de idővel a gyakorlat alapján letisztul.)

### Ignorált fájlok

A **.gitignore** fájlba felsorolhatjuk/eltárolhatjuk azoknak a fájloknak, mappáknak a neveit, amelyeket a projekt során használunk, de nem szeretnénk, hogy verziókezelve legyenek, és hogy a remote rep-ba kerüljenek. A git figyelmen kívül fogja hagyni őket. Ilyen fájlok lehetnek például bizonyos kódszerkesztő programok működéséhez szükséges fájlok. Tipikusan egy IntelliJ esetében például a .idea/ mappa és a project-name.iml fájl ilyenek. Más programok esetében más fájlok is megjelennek. Bármilyen programot is használunk a fejlesztéshez, nagyon fontos, hogy normálisan felvegyük az ignorálni kívánt fájlokat, hiszen ha ezt elfelejtjük, azzal csapattársainknak sok fájdalmat okozhatunk.

Pár példa a .gitignore fájl tartalmára:

● settings.xml,

● minden a gyökér könyvtárban lévő .iml kiterjesztésü fájl,

● target/, out/, bin/, .idea/, .settings/ mappák és bennük lévő fájlok a gyökér mappában,

● bármelyik mappában elhelyezkedő orig/ mappa,

● bármelyik mappában elhelyezkedő .bad kiterjesztésű fájl

setting.xml

\*.iml

target/

out/

bin/

.idea/

.settings/

\*\*/orig/

\*\*/\*.bad

Ha nem tudjuk pontosan, hogy hogyan fogjuk nevezni a fájlt, csak a kiterjesztése szerint tudjuk, hogy ignorálni kell, akkor \*-ot használunk a fájl neve helyén, nem tudjuk, hogy melyik almappában lesz, akkor a mappa nevét \*\*-al helyettesítjük.

### Egyéb

A local és a remote repository összekapcsolása ssh-n keresztül:

1. új ssh key létrehozása Githubon
2. Git Bashban ssh-keygen utasítás
3. felajánl egy helyet a kulcsfájl mentésére
4. a kulcsnál továbbnyomni, mert már eleve titkos csatornán megy az információ
5. be kell menni az adott mappába, ahová a kulcsot mentette
6. a .pub kiterjesztésű fájlt kiíratni a cat paranccsal
7. ki kell jelölni a többsoros szöveget és ki kell másolni és beilleszteni a GitHub Key mezőjébe
8. hozzá kell adni a keyt

(GMT20231019-145101\_Recording\_2560x1440.mp4\_1.21-től)

---

A Githubon lehet egy mappába tenni több repositoryt is, organization néven.

(GMT20231028-105459\_Recording\_1920x1080 \_1ó11től)

---

Restore szituációkhoz részletes leírások:

<https://www.nobledesktop.com/learn/git/undo-changes>

**git checkout filename.html** – ezzel az utasítással visszaáll a fájl az előző commit állapotába

<https://www.git-tower.com/learn/git/faq/git-discard-changes>

## SDLC

Csapatmunka, fejlesztési módszertanok

### Bevezetés

Miért kell az IT-ban csapatban dolgozni, amikor egyedül is lehet? Azért, mert akkor gyorsabban elkészülünk. Egy kellően senior fejlesztő bármilyen alkalmazást meg tud egyedül is csinálni, de egy viszonylag komplex webes alkalmazás esetén ez 5-8 évet is igénybe vehet még ha főállásként is dolgozik rajta. Viszonylag nehéz addig bevétel nélkül létezni… Tehát kulcsfontosságú megtanulni csapatban dolgozni és hogy hogyan kell jól együttműködni. Az IT nem egy irígy közeg, ne féljünk segítséget kérni és ne érezzük rosszul magunkat miatta, illetve mi is érezzük kötelességnek segíteni a másikat.

### Menete

1. **leírás/analízis**: a feladat megfogalmazása és megértése
2. **tervezés**: a fő feladat felosztása kisebb feladatokra, a munkamenet felvázolása és **felosztása**, vagyis a kisebb feladatok megvalósításának kiosztása
3. **megvalósítás**: a kisebb feladatok megvalósítása (egyénileg vagy közösen), kommunikáció a csapattagokkal, feladatok újratervezése (szükség szerint), a felmerülő hibák kezelése
4. **kiadás:** elérhetővé tenni a felhasználóknak
5. **utómunka**: a program minden funkciójának tesztelése, a hibák javítása, refaktorálás, szükség szerint újratervezés vagy módosítás

#### Leírás/Analízis

Az első és legfontosabb feladat az, hogy magát a feladatot megértsük

* minden csapattagnak értenie kell, hogy mi a feladat, mi a cél
* a legfőbb kérdések, amelyekre válaszolni kell:
  + Mi a feladat? (Egy mondatban összefoglalva.)
  + Mit várunk? Mi lesz a végeredmény?
  + Mit kell feltétlenül tudnia a programnak?
  + Mi az, amivel esetleg fejleszthetjük még a programot? (Plusz funkciók.)
  + Milyen eszközökre van szükségünk a megvalósításhoz? (Például: valamilyen szöveges nyersanyag, IDE license, API-k stb.)

Megjegyzés: nyilván a programozási nyelv a mi esetünkben egyértelmű...

#### Tervezés

A fő feladat megfogalmazása után fel kell osztani azt kisebb feladatokra. Érdemes olyan kis feladatokra felosztani, amelyek könnyen megfogalmazhatóak és elvégezhetőek, lépésről lépésre.

A kisebb feladatokat további, még kisebb feladatokra lehet osztani. Nincs szabály arra, hogy hány (és mekkora) feladatokra kell osztani a fő feladatot - ez szokás, megállapodás, “érzés” kérdése.

A megfogalmazott kisebb feladatokat sorrendbe kell állítani. Melyik a legfontosabb? Melyik kevésbé fontos? Mi a feladatok közötti összefüggés? A kisebb feladatok sorrendbe állítása lényegében a munkamenet maga.

Nem szabad elfelejteni, hogy a feladatok és a munkamenet változhat, a projekt terve nincs kőbe vésve!

**Felosztás**

A tervezés része a felosztás. Mivel csapatmunkáról van szó, ezért a csapattagok mind részt vesznek a munkában tehát a kisebb feladatokat ki kell osztani a csapattagok között. A kiosztás azt jelenti, hogy a csapattagok választanak maguknak feladatot (vagy feladatokat), amit (vagy amiket) elvégeznek. Nem kötelező az összes kisebb feladatot már a munka elején elvállalni(kiosztani) - a lényeg, hogy előbb-utóbb mindegyik feladatot elvégezze valaki.

Egy feladaton több csapattag is dolgozhat együtt - de akkor tényleg együtt dolgozzanak, és ne egymás mellett párhuzamosan!

A feladatok kiválasztásánál érdemes követni a prioritási listát.

Több megközelítés is lehetséges a feladatok kiosztásánál, több tényezőt figyelembe véve. A csapattagok kompetenciáját, a feladat sürgősségét vagy tudásmegosztást szem előtt tartva.

#### Megvalósítás

A tervezés és feladatválasztás után jöhet a tényleges kódolás. A csapattagok önállóan vagy akár együtt is dolgozhatnak, a megvalósítás alatt folyamatos kommunikáció szükséges.

Tájékoztatni kell a csapatot arról, hogyan áll a feladat megvalósítása. Időről-időre meg kell osztani az elkészült kódot (kódrészletet) a csapattal.

A munka közben felmerülő kérdéseket, problémákat, esetleges tervezési hibákat meg kell vitatni (szükség szerint újra kell tervezni). Szabad segítséget kérni (nyilván), sőt!

Egy kisebb feladat befejezése után jöhet a következő, de csak azután ha a **tesztelés** és refaktorálás megtörtént és jó minőségű kódot adunk ki a kezünkből. A különböző funkciók mellett az egész programot (alkalmazást) is tesztelni kell.

#### Kiadás

Ha a (rész) fejlesztés és tesztelés rendben volt, akkor lehet kiadni a kódot, terméket, programot és mindenki számára elérhetővé tenni. (release)

#### Utómunka/Karbantartás

Már a megvalósítás alatt tesztelni kell a különböző funkciókat, valamint javítani az esetleges hibákat. Hibás, nem működő kódot soha nem adunk ki a kezünkből!Azonban bármikor előfordulhat, hogy az általunk már késznek nyilvánított feladatot, javítani vagy bővíteni kell. A fejlesztés, tesztelés és hibajavítás után át kell nézni a tervezett feladatokat.

### SDLC

A fent vázolt csapatmunka menetét, az IT-ban SDLC-nek, másnéve Software Development Life Cycle-nek nevezik.

Az SDLC nem egy egyszeri folyamatot mutat be, ez egy folytonosan megismétlődő vagyis iterációkon áteső feladatok halmaza.

### SDLC Modellek

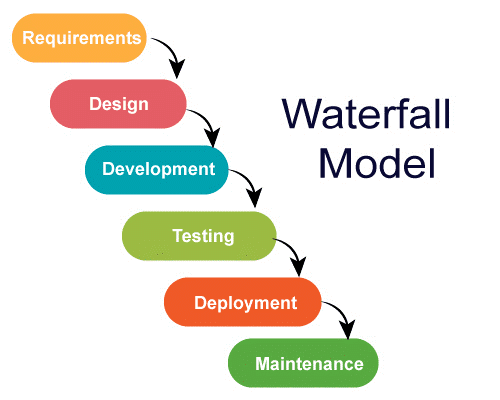
Népszerű vagy régebben népszerű modellek a következőek:

* Vízesés modell - Waterfall Model
* V-Model
* Iteratív modell - Iterative Model

#### Vízesés modell

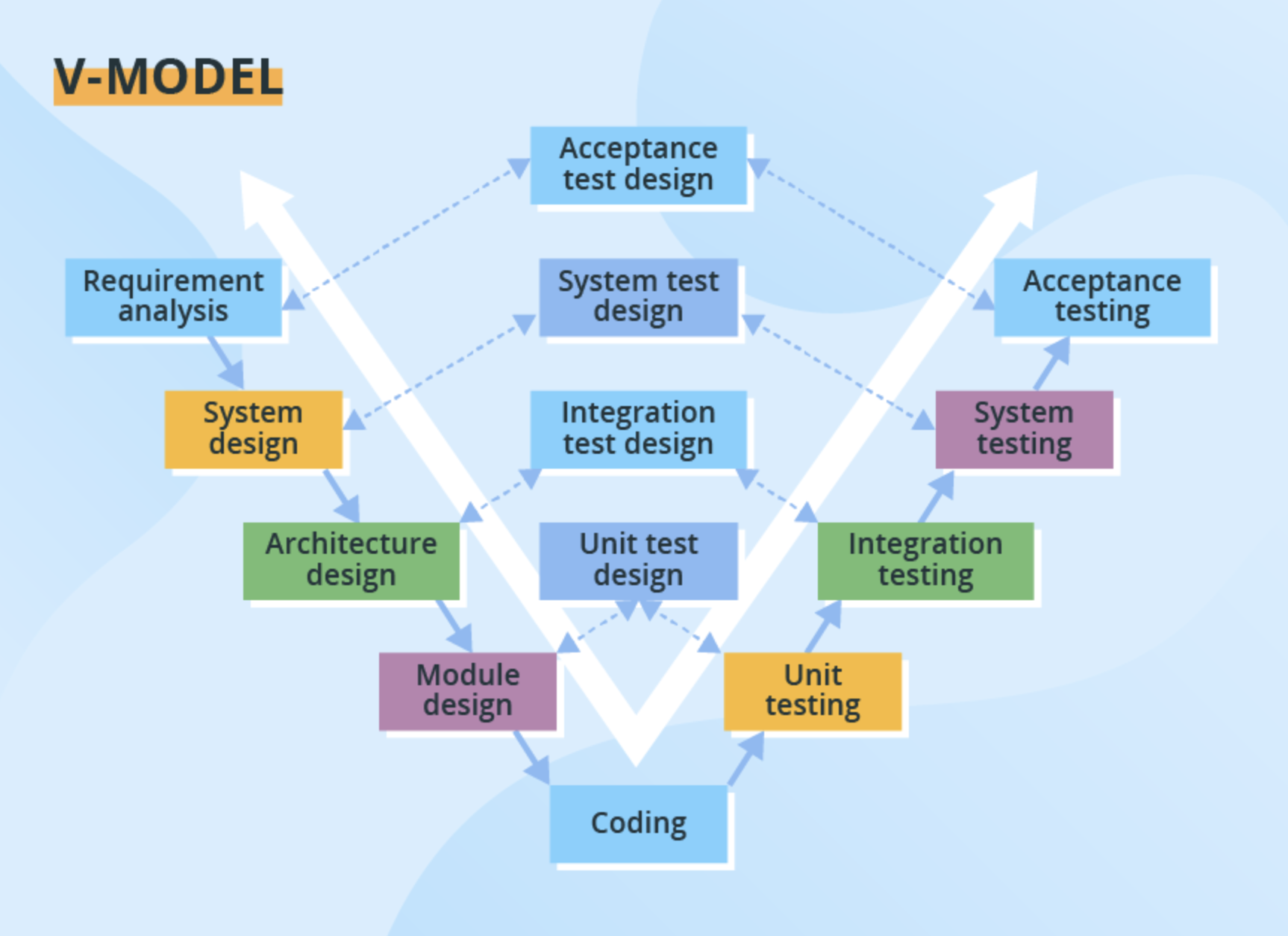
Itt a folyamatok egymás után hajtódnak végre csak és kizárólag akkor lépünk a következő lépcsőfokra, ha a fenti teljes egészében befejeződött. A tesztelés itt túl későn kezdődött meg, így nagyon költséges volt visszalépni az előző fázisok egyikére. Nagyszámú dokumentum mennyiség, fajsúlyos tervezés biztosítja a program megfelelő elkészültét. Ám a termék kis-mértékű utólagos módosítása is horribilis költségekkel járhat.

Nagy hátránya lehet még, hogy lehet akkor derül ki a felhasználó számára, hogy “de nem is ezt kértem” vagy “legyen inkább máshogy”, amikor már átadásra került a teljes termék. A megrendelés és átvétel között pedig eltelt mondjuk pár év….



#### V-modell

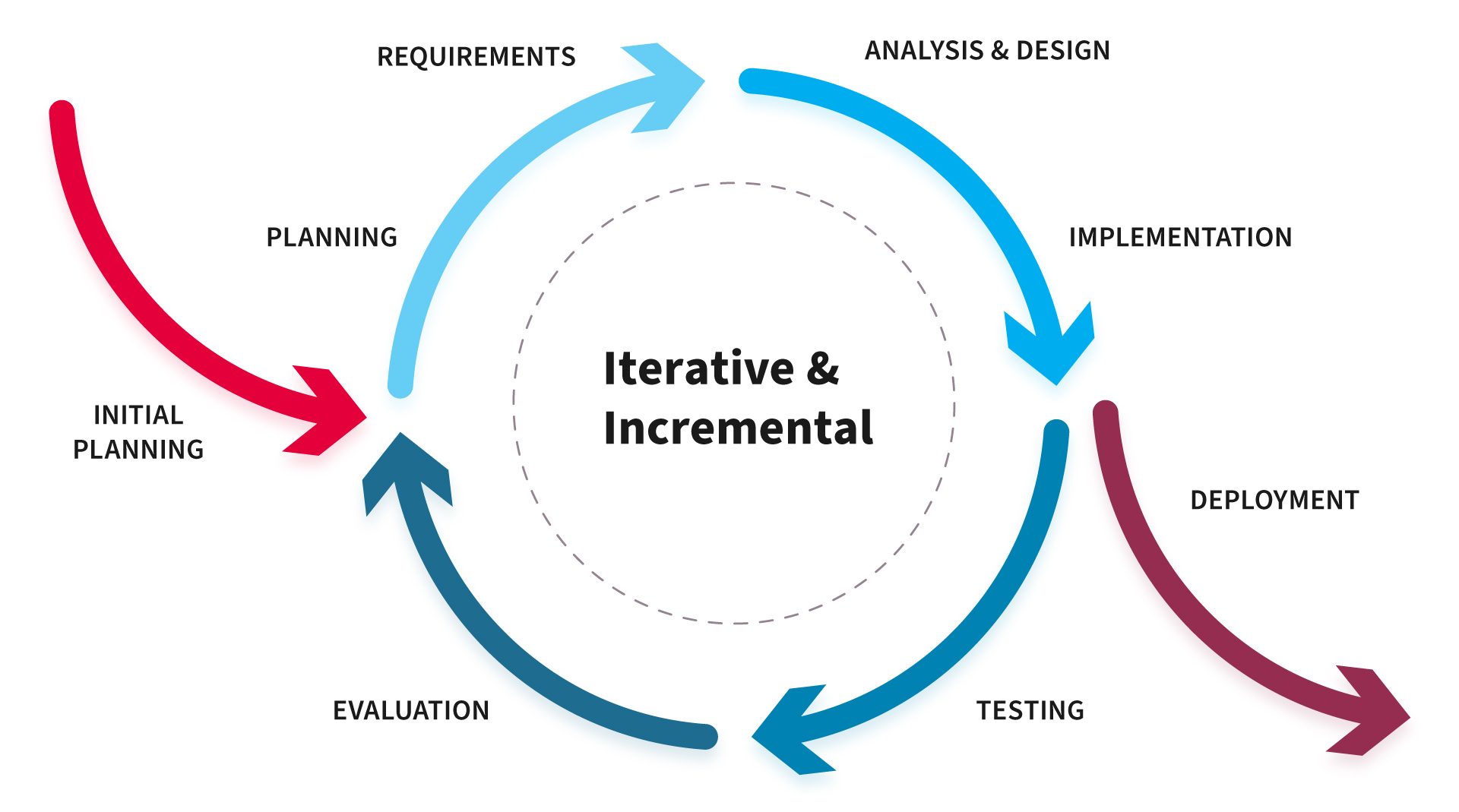
Itt vannak bizonyos folyamatok, amik jól végezhetőek egyidőben, ezzel gyorsítva a fejlesztés egészét. Minden lépcsőfoknak megvan a maga technikája, annak a helyességének a tesztelésére, tehát ez gyakorlatilag a vízesés modell egy kiegészítése. Gyakorlatilag, mire a fejlesztő elkezdi írni a kódot, már a helyességét ellenőrizni hivatott tesztek mind készen vannak.



#### Iteratív modell

Itt a kulcsszó az iteráció, ami ebben az esetben azt jelenti, hogy mihelyst van egy müködő kis önálló egység, azt ki is adom és elérhetővé teszem a felhasználónak, hogy minél hamarabb visszajelzést kapjunk. Ugyanúgy megvan minden, csak kisebb ciklusokban.

Iteráció : magát a ciklust, ami ismétlődhet, nevezik így.



***requirements*** - követelmény

***design***– tervezés

***implementation*** – implementáció (megvalósítás)

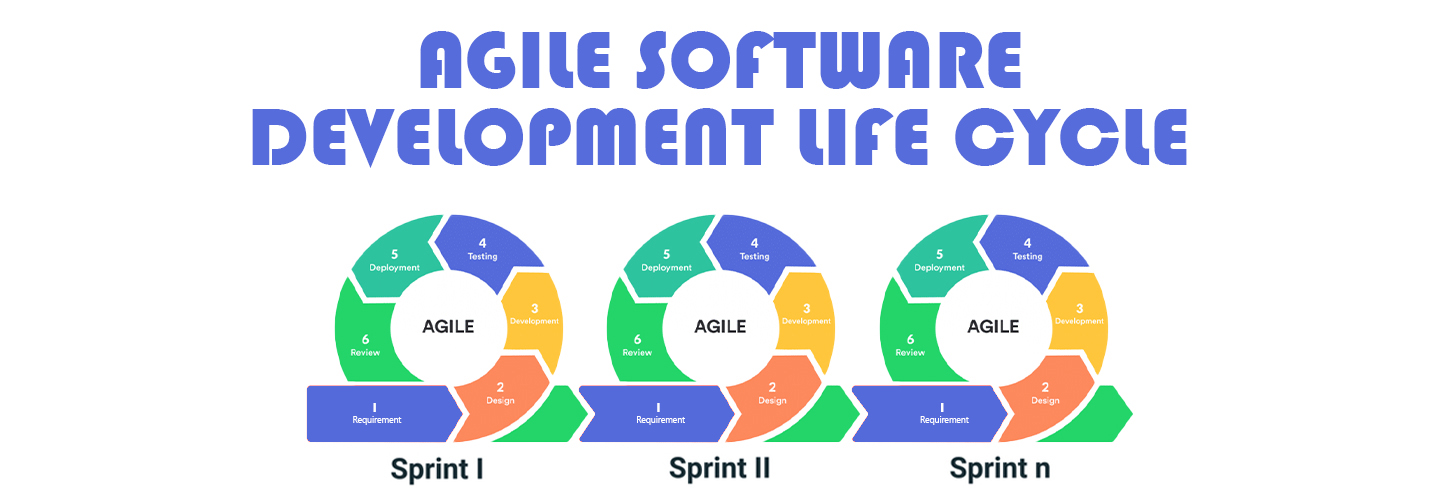
***deployment*** - kitelepítés, feltelepítés olyan szerverre, amiről az alkalmazást elérhetővé lehet tenni az ügyfeleknek

***testing*** - tesztelés

***evaluation*** – kiértékelés és végül kiadás

#### Agile model

Manapság a legdivatosabb SDLC az agile, ami az iteratív modellnek a továbbgondolt változata, amely az iterációkat összegyűjti sprintekké (fejlesztési ciklusok). Itt is az a legfontosabb, hogy amint van egy új működő verzió, miharmarabbi visszajelzést kajunk róla. Folyamatos az együttműködés és az alkalmazkodás is. Minden sprint végén a cél, egy **működő** termék vagy termékfrissítés kiadása. Egy sprint tipikusan 2-3 hét hosszú és a megrendelő vagy egy megbízottja folyamatosan a fejlesztőcsapat rendelkezésére áll a kérdések megválaszolására. Ideális csapatméret az 5-9 fős létszám. Mottó: csapat van nem egyén!



Hátránya lehet, hogy a bővíthetőség, karbantarthatóság nehézségekbe ütközhet. Illetve kell legyen valaki, akitől kapja a csapat a visszacsatolást folyamatosan. Viszont előnye, hogy flexibilis, kevesebb tervezést igényel és nem akkora baj ha menet közben változik a követelmény.

##### SCRUM

A Scrum az Agile SDLC-nek egy megvalósítása (az agile egy szempontrendszer, hogy iterációkban történjen a fejlesztés)

***grooming*** – A sprinten belüli feladatokat a követelmények alapján ki kell dolgozni, pontosítani kell. Meg kell tervezni, hogy hogyan kell megvalósítani. Előretekintés, a következő spint teendőinek a megbeszélése, amiből kiderül, hogy mindenkinek minden érthető e (2-4 óra kéthetente). A fejlesztőcsapatból legalább egy-egy fejlesztő, tesztelő, csapatvezető és a scrum master, project manager vagy product owner.

***sprint planning*** – A sprint első momentuma, a sprint megtervezése, mindenki megtudja, hogy ki mivel fog foglalkozni, mit fog commitálni, és mi a fontossági sorrend. Egy olyan megbeszélés, amin az egész fejlesztőcsapat részt vesz (sokszor a megrendelő, képviselője is valamint csapatvezető, architech…). (1-2 óra 2hetente)

***daily standup*** - A fejlesztő csapat vesz részt rajta. (10-15p naponta)

***sprint review/demo*** - Be kell mutatni az eredményt, hogy mi történt az elmúlt sprinten, ezt többnyire egy vagy több ember adja elő, akár a megrendelőnek (1x 30p)

***retrospective*** - a sprint végén visszatekintés, kiértékelés, észrevételek, a hatékonyság növelésére. mindenki hoz valamilyen témát. A fejlesztési csapat vesz részt rajta és jellemzően 1-2 óra hosszú.

### Feladatok

1. Miért jobb csapatban dolgozni?
2. Mik az SDLC fázisai?
3. Milyen modellekről tanultunk?
4. Mi az előnye az Agile-nak a Vízesés-sel szemben?
5. Milyen SCRUM meeting-eket ismertek? Melyiket mire használják?
6. Rajzolós feladat - Waterfall
7. Rajzolós feladat - Agile

**1.** **Miért jobb csapatban dolgozni?** - Mert gyorsabban elkészül a munka. Hatékonyabb.

**2.** **Mik az SDLC fázisai?**

1. leírás/analízis: a feladat megfogalmazása és megértése

2. tervezés: a fő feladat felosztása kisebb feladatokra, a munkamenet felvázolása és felosztása,

vagyis a kisebb feladatok megvalósításának kiosztása

3. megvalósítás: a kisebb feladatok megvalósítása (egyénileg vagy közösen), kommunikáció a

csapattagokkal, feladatok újratervezése (szükség szerint), a felmerülő hibák kezelése

4. kiadás: elérhetővé tenni a felhasználóknak

5. utómunka: a program minden funkciójának tesztelése, a hibák javítása, refaktorálás, szükség

szerint újratervezés vagy módosítás

**3.** **Milyen modellekről tanultunk?**

● Vízesés modell - Waterfall Model

● V-Model

● Iteratív modell - Iterative Model

● Agile model

**4. Mi az előnye az Agile-nak a Vízesés-sel szemben?**

A vízesésben a folyamatok egymás után hajtódnak végre és csak akkor van továbblépés a

következőpre, ha az előző maradéktalanul befejeződött. Míg az agileben sok röidebb ciklus

van, melyek ismétlődnek. Így rendszeresen van visszacsatolás a megrendelőtől, nem csak a

teljes folyamat végén.

**5. Milyen SCRUM meeting-eket ismertek? Melyiket mire használják?**

***1. grooming*** – A sprinten belüli feladatokat a követelmények alapján ki kell dolgozni,

pontosítani kell. Meg kell tervezni, hogy hogyan kell megvalósítani. Előretekintés, a következő

spint teendőinek a megbeszélése, amiből kiderül, hogy mindenkinek minden érthető e (2-4

óra kéthetente). A fejlesztőcsapatból legalább egy-egy fejlesztő, tesztelő, csapatvezető és a

scrum master, project manager vagy product owner.

***2. sprint planning*** – A sprint első momentuma, a sprint megtervezése, mindenki megtudja, hogy

ki mivel fog foglalkozni, mit fog commitálni, és mi a fontossági sorrend. Egy olyan

megbeszélés, amin az egész fejlesztőcsapat részt vesz (sokszor a megrendelő, képviselője is

valamint csapatvezető, architech…). (1-2 óra 2hetente)

***3. daily standup*** - A fejlesztő csapat vesz részt rajta. (10-15p naponta)

***4. sprint review/demo*** - Be kell mutatni az eredményt, hogy mi történt az elmúlt sprinten, ezt

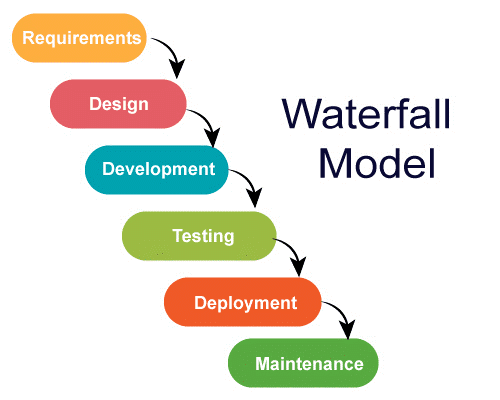
többnyire egy vagy több ember adja elő, akár a megrendelőnek (1x 30p)

***5. retrospective*** - a sprint végén visszatekintés, kiértékelés, észrevételek, a hatékonyság

növelésére. mindenki hoz valamilyen témát. A fejlesztési csapat vesz részt rajta és jellemzően

1-2 óra hosszú.

**6. Rajzolós feladat - Waterfall**



***1.requirements*** - követelmény

***2.design***– tervezés

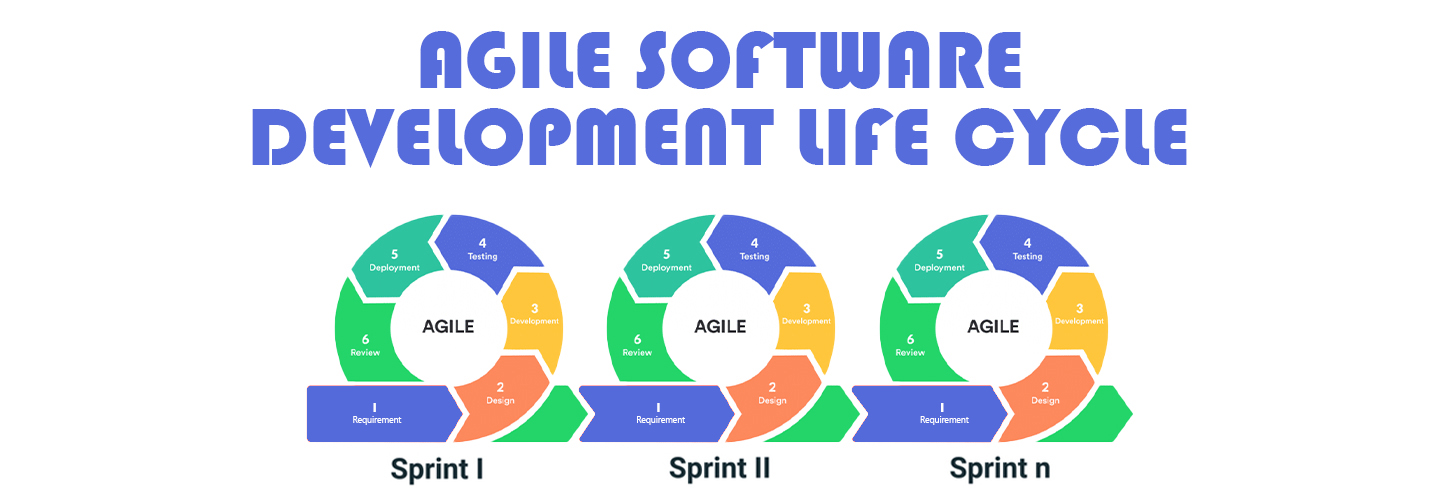
***3.development*** - fejlesztés

***4.testing*** - tesztelés

***5.deployment*** - kitelepítés, feltelepítés olyan szerverre, amiről az alkalmazást elérhetővé lehet tenni az ügyfeleknek

***6. maintenance*** – fenntartás, karbantartás, gondozás

**7. Rajzolós feladat – Agile**



***1.requirements*** - követelmény

***2.design***– tervezés

***3.development*** - fejlesztés

***4.testing*** - tesztelés

***5.deployment*** - kitelepítés, feltelepítés olyan szerverre, amiről az alkalmazást elérhetővé lehet tenni az ügyfeleknek

***6. review*** – áttekintés, felülvizsgálat

## Web basics

HTML, CSS, Javascript

### HTML

A weboldal leírója a HTML (HyperText Markup Language) nyelv, egy szabványosított programozási nyelv. A HTML fájl egy egyszerű szövegállomány, mely úgynevezett jelölő tag-eket tartalmaz. A tag-ek alapján tudja megjeleníteni az oldalt a böngésző.

A fájlok kiterjesztési nyelve .html, szerkesztéséhez elég lehet egy egyszerű szövegszerkesztő.

#### Példák HTML-re kommentárokkal („feladat0” fájl – azóta már bővítve)

<html> (ezen belül helyezkedik el a teljes tartalom)

   <head> (ez tartalmazza a metainformációkat)

      <title>proba honlap</title> (ez a szöveg látszik a böngésző tabon, azaz

a cím)

      <meta name="description" content="progmatic gyakorlat"> (a

keresőmotoroknak szól, amelyek sokkal

hatékonyabban találnak rá ezzel a taggel)

      <meta charset="UTF-8"> (karakterkódolás szabványosítása, ezzel explicit

mindenhol így fog megjelenni)

      <body bgcolor="white"> (meghatározza a háttér (azaz a body) színét)

      <style>body{color: blue;}</style> (meghatározza a szöveg színét)

<style> (css stílusok)

/\*CSS comment out \*/

</style>

      <script>

alert('Üdvözöllek') (felugró popup ablak)

//javascript comment out//

     </script>

       <!-- html comment out -->

(ha ezzel a taggel (<!-- -->) közrefogunk valamit (pl. az alert

parancsot), akkor "kikommenteljük" azt és nem fog lefutni)

   </head>

   <body> (ezen belül helyezkedik el a tényleges tartalom)

        Ez a honlapom (egyszerű szöveg)

        <h1>Cím1</h1> (címsorok 1-6-ig méretben)

        <h2>Cím2</h2>

        <h3>Cím3</h3>

        <p>Ez egy bekezdés.</p> (egy sorral elválasztja az adott szöveget az

előtte és az utána lévőtől)

        <p>Ez egy másik bekezdés.</p>

        <p>És mégegy bekezdés,<br> megtört sorral.</p> (A <br> egy önmagában

álló tag, ami új sort eredményez)

        <hr> (egy vízszintes vonal)

        <div>A HTML div elemét arra használják, hogy a weboldalon egy jól

elkülöníthető egységet elkülönítsenek.

           Általában egymáshoz szorosan kapcsolódó elemek mennek bele. Úgy

kell ezt érteni, hogy ha van a weboldalunkon

           jobb oldalt egy menü, akkor a menü összes elemét belehelyezhetjük.

Míg a tartalom egy másik div-be kerül.

           Így egymástól elkülönítettük a tartalmat és a menüt, amelyet

könnyedén egységesen formázhatunk stílusokkal.</div> (a ’div’-vel

már nem csak szöveges bekezdést, hanem egy

jól elülöníthető egységet is létrehoztunk)

        <br>

        <b>ez egy félkövér szöveg </b> (betűk formázása)

        <strong>ez is félkövér szöveg</strong>

        <big>nagyobb szöveg</big> <br>

        <em>szövegkiemelés</em>

        <i>dőlt szöveg</i>

        <small>kisebb szöveg</small>

        <br><br>

        <a href="https://www.facebook.com">Ez egy link a Facebookra.</a>

(link beágyazása)

        <br>

        <a href="https://www.facebook.com" target="\_blank">Link megnyitása új

lapon.</a> (link megnyitása új lapon)

        <br><br>

        <button type="button">Kattanj rám.</button> (gomb létrehozása)

        <br><br>

        <img src="hatter.jpg"> (kép beillesztése)

        <br><br>

        <img src="hatte.jpg" alt="ez egy kép egy hídról"> (kép beillesztése a

kép címével, ha esetleg nem lenne elérhető,

ill. a keresők is hatékonyabban találnak rá)

        <br><br>

        <a href="https://www.facebook.com">

            <img src="lovaglo\_suni.jpg" alt="vissza a főoldalra" width="40" height="40">

        </a> (kép linkkel. ez a link lehet akár a főoldalra mutató link is)

<button onclick="document.location='koszonom.html'">Örökbe fogadom</button> (gomb linkkel)

        <br><br>

        <table> (táblázat létrehozása)

            <tr>

               <td>1</td>

               <td>2</td>

               <td>3</td>

            </tr>

            <tr>

               <td>4</td>

               <td>5</td>

               <td>6</td>

            </tr>

        </table>

        <br><br>

        <table border="1"> (táblázat rácsozással)

            <tr>

               <td>1</td>

               <td>2</td>

               <td>3</td>

            </tr>

            <tr>

               <td>4</td>

               <td>5</td>

               <td>6</td>

            </tr>

        </table>

        <br><br>

        <table border="1">

            <th> (a 'th' tag-gel megadhatóak a tag-ek között fejlécként az

oszlopok nevei)

               <td>oszlop 1</td>

               <td>oszlop 2</td>

            </th>

            <tr>

               <td>4</td>

               <td>5</td>

               <td>6</td>

            </tr>

        </table>

        <br><br>

        <ul> (lista pontokba szedve (unordered))

            <li>kiskutya</li>

            <li>kismacska</li>

            <li>kiskacsacsőrűemlős</li>

        </ul>

        <br><br>

        <ol> (lista számokkal (ordered))

            <li>kiskutya</li>

            <li>kismacska</li>

            <li>kiskacsacsőrűemlős</li>

        </ol>

        <br><br>

        <dl> (definíciós lista)

            <dt>kiskutya</dt>

               <dd>puli,roti</dd>

            <dt>kismacska</dt>

               <dd>balinéz,bengáli</dd>

        </dl>

   </body>

</html>

space beszúrása: &nbsp

kép beillesztése méret megadásával (mindkét utasításnak ua. az eredménye):

<img src="img\_girl.jpg" alt="Girl in a jacket" style="width:500px;height:600px;">

<img src="img\_girl.jpg" alt="Girl in a jacket" width="500" height="600">

Egyéb

A Visual Studio Code-ban egy ! beírásával és megnyomásával egy alap template HTML jelenik meg. A format on save beállítás is hasznos! És a Live server plugin is.

### CSS

CSS, azaz Cascading Style Sheet. Egy olyan “adatlap”, ami meghatározza a HTML elemek megjelenését. Ezeket a stílusokat általában külön, egy .css kiterjesztésű fájlban tároljuk. Ezzel időt spórolhatunk meg, könnyebb karbantarthatóság.

A CSS stílusok többféle módon helyezkedhetnek el a html-ben.

tulajdonság megadása magában az adott html tagben (inline):

 <table>

        <tr>

            <td></td>

            <td style="font-size: 200px">&nbsp;42&nbsp;</td>

            <td></td>

        </tr>

</table>

tulajdonság megadása globálisan a Head tagbe ágyazva a Style taggel, azon belül pedig az adott html tag elnevezéssel (td, table, body, h5, stb.):

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Feladat 04</title>

    <style>

        td {

            font-size:1000%;

        }

        table {

            margin: auto;

            border-style: solid;

            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

        }

button {

        width: 100px;

    }

    </style>

</head>

<body>

    <table>

        <tr>

            <td></td>

            <td>42</td>

            <td></td>

        </tr>

    </table>

<button>Ok</button>

    <button>Rendben</button>

</body>

tulajdonság megadása „class”-al, ami csak a megjelölt html tagekre vonatkozik:

<head>

<style>

        .rounded\_img {

            border-radius: 20px;

        }

        .circle\_img {

            border-radius: 1000px;

        }

        .shadow\_img {

            padding-top: 10px;

            padding-left: 10px;

            padding-right: 10px;

            padding-bottom: 10px;

            border-radius: 20%;

            box-shadow: 10px 5px 15px red;

        }

    </style>

</head>

<body>

    <img class="rounded\_img" src="feladat5b/cat.avif" alt="cat image">

<img class="circle\_img" src="feladat5b/cat.avif" alt="cat image">

    <img class="shadow\_img" src="feladat5b/cat.avif" alt="cat image">

</body>

mégegy példa: div-be ágyazva és azon belül is van még két osztály:

<head>

    <style>

        .profile {

            border-style: double;

            width: 250px;

            display: inline-block;

            box-shadow: 10px 5px 15px red;

        }

        .catname {

            color: azure;

            font-size: 30px;

            text-align: center;

            margin-top: 5px;

            margin-bottom: 5px;

        }

        .catage {

            color: rgb(155, 194, 187);

            font-size: 20px;

            text-align: center;

            margin-top: 5px;

            margin-bottom: 5px;

        }

    </style>

</head>

<body>

    <div class="profile">

        <img src="feladat5b/cat.avif" alt="cat image">

        <p class="catname">Cilike</p>

        <p class="catage">fél éves</p>

        <button>Örökbe fogadom</button>

    </div>

</body>

tulajdonság megadása „id”-val, ami csak a megjelölt html tagre vonatkozik (csak egyet jelöljünk meg így!):

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Feladat 7</title>

</head>

<style>

#button-exit{

        border-radius: 20px;

        background-color: rgb(255, 0, 170);

    }

</style>

<body>

    <button id="button-exit">Kilépés</button>

</body>

egy másik fájlban, head tag-ek közé belinkelve:

<head>

      <title>0. feladat</title>

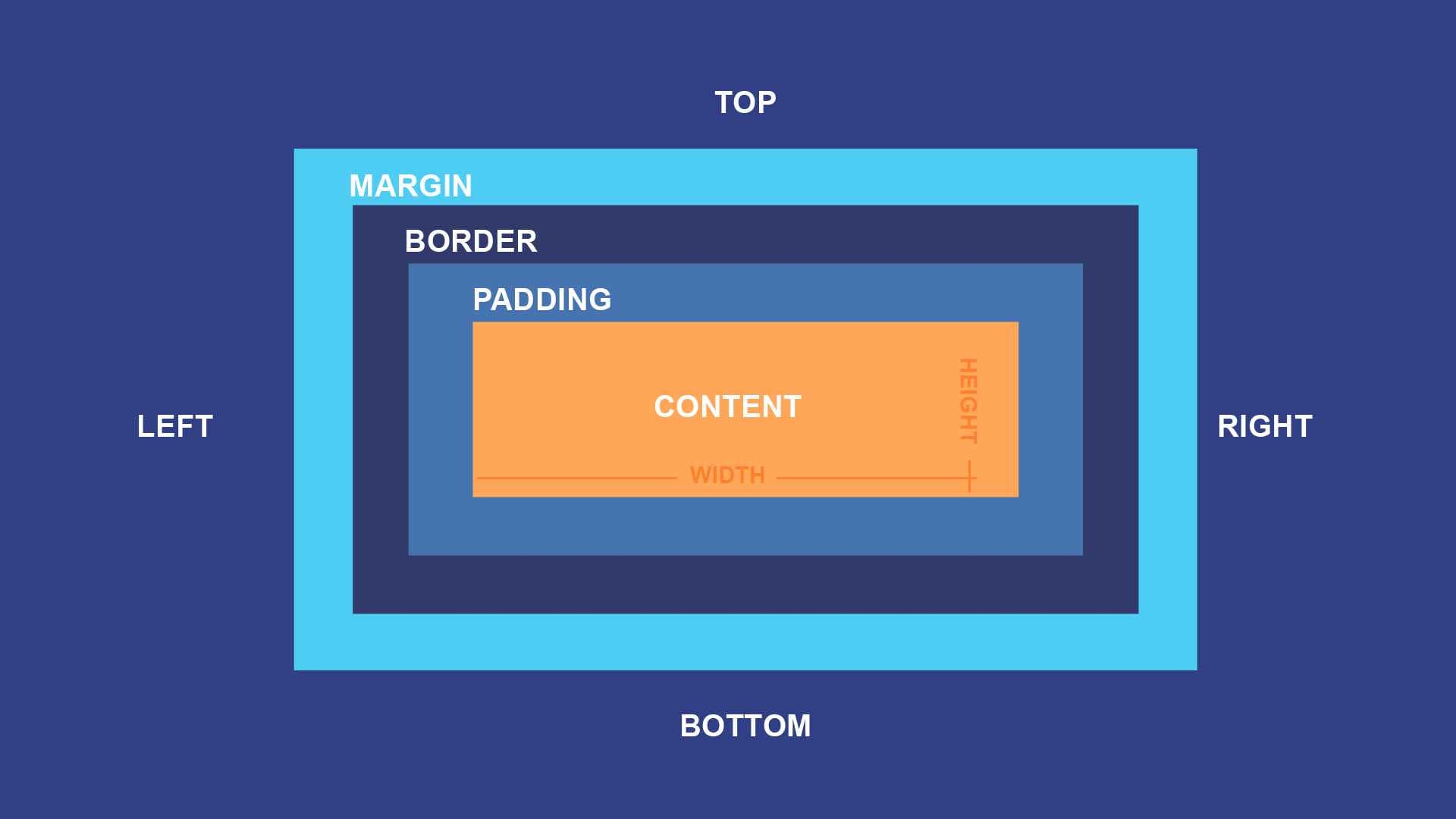
      <meta name="description" content="progmatic gyakorlat">

      <meta charset="UTF-8">

<link rel="stylesheet" href="Aboutme\_darkmode.css" />

   </head>

egy elem elhelyezkedésének és méretének tulajdonságainak hierarchiája:



#### Példák CSS-re

háttér

|  |
| --- |
| htmlelem{  background-color: érték; */\*háttér színe \*/*  background-image: url(a háttérkép elérése);   background-repeat: érték; */\*háttérkép ismétlődése \*/*  background-position: érték; */\*háttérkép pozíciója \*/*  background-attachment: érték; */\*háttérkép ragasztása\*/* } |

szöveg

|  |
| --- |
| htmlelem{  color: érték;  text-align: érték; */\*igazítás\*/*  text-decoration: érték; */\* szöveg megj: aláhúzott, áthúzott, stb.\*/*  font-family: érték; */\*betűtípus \*/*  font-size: érték; */\*betűméret \*/*  font-style: érték; */\*stílus \*/*  font-weight: érték; */\*betű vastagsága \*/* } |

méret

|  |
| --- |
| htmlelem{  width: szélesség; /\*csak blokkszintű elemeknél: <p>,<div>,<h1>,<ul>,<ol>,<table>\*/  height: magasság; /\*csak blokkszintű elemeknél\*/ } |

szegély

|  |
| --- |
| htmlelem{  border-style: érték; */\*solid, dotted, dashed, groove, double, stb. \*/*  border-color: érték; */\*keret színe\*/*  border-width: érték; */\*px-ben megadva\*/*  border-radius: érték; */\*adott html elem sarkainak lekerekítése\*/*  } |

térköz

|  |
| --- |
| htmlelem{  margin: értékek; */\*külső margó px, cm, em-ben megadva, sorrend: fel, jobbra, le, balra \*/*  margin:0 auto; */\*középre igazítás\*/*  padding: értékek; */\*belső margó px, cm, em-ben megadva, sorrend: fel, jobbra, le, balra \*/* } |

lista

|  |
| --- |
| htmlelem{  list-style-type: érték; */\*lista jelének típusa: disc, square, circle, none számozott listák esetén: decimal, lower-roman, upper-roman, lower-alpha \*/*  list-style-image: url(elérési út); */\* "listajel" helyett kép berakása\*/*  list-style-position: érték; */\*lehet: inside, outside \*/*  list-style: értékek; */\*előbbiek közös megadása\*/* } |

megjelenítés

|  |
| --- |
| htmlelem{  display: érték; */\*lehet block, inline, none, flex, stb\*/*  float: érték; */\*left, right, stb., körbefuttatás szöveggel vagy hasábos\*/* } |

szöveg igazítása:

<p style="text-align:center;">Ez a szöveg középre igazított.</p>

<p style="text-align:left;">Ez a szöveg balra igazított.</p>

<p style="text-align:right;">Ez a szöveg jobbra igazított.</p>

<p style="text-align:justify;">Ez a szöveg sorkizárt, de csak akkor látszik, ha jó hosszú a mondat.</p>

különböző pozíciók egymáshoz képest:

képek egymás mellé, ha máshogy nem menne

<https://www.w3schools.com/howto/howto_css_images_side_by_side.asp>

flexbox

<https://www.w3schools.com/css/tryit.asp?filename=trycss3_flexbox_flex-wrap_nowrap8>

linkek

A linkek is különféleképpen viselkedhetnek:

|  |
| --- |
| a:link { color:érték; }  a:visited { color:érték; } a:hover { color:érték; } a:active { color:érték; } |

**link**: link alapértelmezett tulajdonsága  
**visited**: már látogatott link

**hover**: felé vittük az egeret

**active**: rákattintunk

### Formok, form elemek (HTML elemek)

A weboldalak egyik nagyon fontos lehetséges területe a formok, másnéven űrlapok és az ezekhez kapcsolódó HTML elemek, melyek a felhasználói bevitelt biztosítják.

Ezeket az elemeket a form-ok esetén form tag-ek közé szokás rakni.

|  |
| --- |
| <form> . form elements . </form> |

Input

A HTML elemnek többféle típusa van. Nézzük őket sorba:

Egyszerű beviteli mező

|  |
| --- |
| <input type=”text”> |

Radio button

|  |
| --- |
| <input type=”radio”> |

Checkbox

|  |
| --- |
| <input type=”checkbox”> |

Submit/elküldés gomb

|  |
| --- |
| <input type=”submit”> |

Reset/visszaállítás gomb

|  |
| --- |
| <input type=”reset”> |

Egyszerű gomb

|  |
| --- |
| <input type=”button”> |

Jelszó mező

|  |
| --- |
| <input type=”password”> |

Dátum mező

|  |
| --- |
| <input type=”date”> |

Mennyiség választó mező

|  |
| --- |
| <input type=”number”> |

Mennyiség csúszka mező

|  |
| --- |
| <input type=”range”> |

Label / felirat

|  |
| --- |
| <label></label> |

Dropdown / lenyíló lista

|  |
| --- |
| <select id="workdays" name="workdays">  <option value="monday">Monday</option>  <option value="tuesday">Tuesday</option>  <option value="wednesday">Wednesday</option>  <option value="thursday">Thursday</option>  <option value="friday">Friday</option>  </select> |

Textarea / szövegdoboz

|  |
| --- |
| <textarea rows="20" cols="70"></textarea> |

Fieldset / Mezőhalmaz

|  |
| --- |
| <fieldset></fieldset> |

### Javascript

A Javascript ad dinamikát az oldalnak.

Ha a html-be írjuk a függvényeket, akkor a <head> tag-ek közé szokás tenni, de valójában bárhol lehetnek. Általánosságban viszont tanácsos külön fájlba tenni.

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport"

        content="width=device-width, initial-scale=1.0">

    <title>11. feladat\_darkmode</title>

    <script src="feladat11\_darkmode.js"></script>

</head>

#### Példák Javascriptre-re (néhány a js\_probak.html-ben is szerepel)

Ebben a példában a *document.querySelector(„”)* – megkeresi az adott dokumentumban a zárójelben lévő taget és eltárolja a *const* után található névben (változóban). Azaz megkeresi a h1 taget és eltárolja a myHeading-ben. És a myHeading.textContent utasításra felülírja a h1 szövegét.

|  |
| --- |
| const myHeading = document.querySelector("h1"); myHeading.textContent = "Hello world!"; |

<body>

    <h1>Hello</h1> <!-- felülírja a "myHeading" változó -->

</body>

<script>

    const myHeading = document.querySelector("h1");

    myHeading.textContent = "Hello world!";

 </script>

Id szerint is lehet keresni:

<body>

<p id="demo">Valami</p>

</body>

<script>

const myParagraph = document.getElementById('demo');

myParagraph.textContent = "ez is felülíródott";

</script>

A gomb megnyomására indul el a script (a gombnak van egy **onclick** tulajdonsága). Felülírásra kerül a paragrafus. A getEl..-re megkeresi a demo2 id-t és a textcont-tal felülírja az eredeti szöveget a gomb megnyomására.

<body>

    <button onclick="document.getElementById('demo2').textContent='Hello world'">Script indítás</button>

    <p id="demo2">Kattints egy üdvözlésért</p>

</body>

Ugyanez, csak a dátumot kiíratva a „new date()” utasítással (a „new” a „date” előtt egy új objektum létrehozását jelenti, de ez itt nem számít):

<body>

    <button onclick="document.getElementById('demo3').textContent=new Date();">Jelen dátum</button>

    <p id="demo3">Kattints a dátumért</p>

</body>

Ez is ugyanaz, csak „innerHTML” paranccsal.

<body>

    <button type="button" onclick="document.getElementById('demo').innerHTML = Date()">

    Click me to display Date and Time.</button>

<p id="demo">Ez is a dátum</p>

</body>

Itt is ugyanaz történik, csak két gombbal két különböző képet (egy gif és egy jpg) indít el. Az utasításban a getElem..-t után az src-vel mondjuk meg, hogy képfáljt jelenítünk meg.

<body>

    <img id="bulpimage" src="feladat8/bulb-on.gif" alt="">

    <br>

    <button onclick="document.getElementById('bulpimage').src='feladat8/bulb-off.jpg'">Turn

        off the bulb</button>

    <button onclick="document.getElementById('bulpimage').src='feladat8/bulb-on.gif'">Turn

        on the bulb</button>

</body>

Tulajdonság módosítása (ezesetben betűnagyság).

<body>

        <button type="button"

onclick="document.getElementById('demo').style.fontSize='35px'">Click

            Me!</button>

        <p id="demo">JavaScript can change the style of an HTML element.</p>

</body>

Amennyiben összetettebb utasítást szeretnénk, akkor a head tagben és azon belül a script tagben a „function” utasítást használjuk. Ez esetben, amennyiben a gomb megnyomásra kerül, az megkeresi a daySelected függvényt és végrehajtja a függvényben leírt utasításokat (a getElem… utasítás megkeresi a „resulttext” id-t és felülírja annak tartalmát).

<head>

    <script>

        function daySelected() {

            document.getElementById("resulttext").textContent = "Clicked"

        }

    </script>

</head>

<body>

    <p id="resulttext">Click on one of the buttons bellow.</p>

    <button onclick=daySelected()>Click me!</button>

</body>

Az előző példa kiegészítve mégegy utasítással:

<head>

    <script>

        function daySelected() {

            document.getElementById("resulttext").textContent = "Clicked"

document.querySelector("button").style.color ='green'

        }

    </script>

</head>

<body>

    <p id="resulttext">Click on one of the buttons bellow.</p>

    <button onclick=daySelected()>Click me!</button>

</body>

Ebben a példában az adott gombra nyomva az adott id-t módosítjuk (színezzük át). Tehát a gomb meghívja a daySelected függvényt, amely a day elnevezés által utasítja a 2. sorban az adott gomb átszínezésére. Tehát a ’Monday’ id a gombot azonosítja be, a piros ’Monday’ elnevezés pedig azt mondja meg, hogy melyiket színezze át. (ha megcseréljük a két piros szöveget, akkor egyértelművé válik a működés)

Máshogy: a piros ’Monday’ nevű text átvándorol a daySelected(day)-be és meghívja a 2-ik sort, amely rákeres az adott gomb piros szövegére és csak azt a gombot módosítja (színezi át)(9.feladat).

<head>

    <script>

        function daySelected(day) {

            document.getElementById("resulttext").textContent = " Kiválasztott nap";

document.getElementById(day).style.color ='green';

        }

    </script>

</head>

<body>

    <p id="resulttext">Click on one of the buttons bellow.</p>

    <button id="Monday" onclick=daySelected('Monday')>Click me!</button>

    <button id="Tuesday" onclick=daySelected('Tuesday')>Click me!</button>

</body>

A példa kibővítve egy másik színű gombbal és egy kicsit más megoldással, mert már az adott id piros szövegét jeleníti meg (azt az értéket, amit függvényhíváskor megadtunk + a stringet(ha írunk hozzá)).(9. feladat)

<head>

    <script>

        function daySelected(day) {

            document.getElementById("resulttext").textContent = day;

            document.getElementById(day).style.color = 'green';

        }

        function yearSelected(year) {

            document.getElementById("resulttext").textContent = year ' pressed';

            document.getElementById("actualyear").style.color = 'purple';

        }

    </script>

</head>

<body>

    <p id="resulttext">Click on one of the buttons bellow.</p>

    <button id="Monday" onclick=daySelected('Monday')>Click

        me!</button>

    <button id="Tuesday" onclick=daySelected('Tuesday')>Click

        me!</button>

    <button id="actualyear" onclick=yearSelected('2023')>Click

        me!</button>

</body>

Itt pedig már belép az if és else is (ebben a példában már nem szerepel az év, csak a hét napjai). Még egy funkcióval bővül. Van egy gomb, ami harmadik színű. Minden más esetben zöld marad.(9b feladat)

<head>

    <script>

        function daySelected(day) {

            document.getElementById("resulttext").textContent = day;

            if (day == 'Sunday')

                document.getElementById(day).style.color = 'red';

            else

                document.getElementById(day).style.color = 'green';

        }

    </script>

</head>

<body>

    <p id="resulttext">Click on one of the buttons bellow.</p>

    <button id="Monday" onclick=daySelected('Monday')>Click

        me!</button>

    <button id="Tuesday" onclick=daySelected('Tuesday')>Click

        me!</button>

    <button id="Wednesday" onclick=daySelected('Wednesday')>Click

        me!</button>

    <button id="Thursday" onclick=daySelected('Thursday')>Click

        me!</button>

    <button id="Friday" onclick=daySelected('Friday')>Click

        me!</button>

    <button id="Saturday" onclick=daySelected('Saturday')>Click

        me!</button>

    <button id="Sunday" onclick=daySelected('Sunday')>Click

        me!</button>

</body>

A Feladat 10-nek a jól érthető magyarázata: GMT20231102-155030\_Recording\_2560x1440 \_15p-től

<head>

    <script>

        function btnSelected(btn) {

            if (btn == 'looseBtn')

                document.getElementById("resulttext").textContent = 'Sorry, no luck this time.';

            else

                document.getElementById("resulttext").textContent = 'Congratulations, you won the price!';

        }

    </script>

</head>

<body>

    <p id="resulttext">Click on one of the buttons below.</p>

    <button onclick=btnSelected('looseBtn')>Click me!</button> (a sárga a függvény neve, a piros a változó neve, ami behelyettesítésre kerül a fenti function után következő függvény zárójelében (ezesetben „btn”).

Ez a Feladat 10b. Az else 2. utasításában látszik, hogy ebben a feladatban miért fontos, hogy azonos legyen egy adott gomb id-a és változóneve. Mert a parancs úgy szól, hogy

document.getElementById(btn).style.color = 'red';

vagyis keresd meg azt az id-t, (ami a változó neve (is)) és színezd át pirosra. Míg a zöldnél elég volt megadni a változó nevét, mert csak egy van belőle.

<head>

    <script>

        function btnSelected(btn) {

            if (btn == 'winBtn') {

                document.getElementById("resulttext").textContent = 'Congratulations, you won the price!';

                document.getElementById("winnerBtn").style.color = 'green';

            }

            else {

                document.getElementById("resulttext").textContent = 'Sorry, no luck this time.' + btn + ' clicked.';

                document.getElementById(btn).style.color = 'red';

            }

        }

    </script>

</head>

<body>

    <p id="resulttext">Click on one of the buttons below.</p>

    <button id="looseBtn1" onclick=btnSelected('looseBtn1')>Click

        me!</button>

Egy példa a body-ban az osztálynévre (classname) való keresésre:

 function darkModeChanger() {

            if (document.body.className == 'dark-theme') {

                document.body.className = "light-theme"

                document.getElementById("btn").innerText = "Dark"

            } else {

                document.body.className = "dark-theme"

                document.getElementById("btn").innerText = "Light"

            }

## ISTQB

### Bevezetés

Az ISTQB anyag nagyja benne van az ISTQB dokumentumban. Itt csak vázlatosan van róla szó.

Az International Software Testing Qualifications Board egy nemzetközi szinten működő szoftvertesztelési tanúsító testület.

Az ISTQB a vezető globális tanúsítványi rendszer a szoftvertesztelés területén.

### Szoftvertesztelés

Egy rendszer vagy program kontrollált körülmények melletti futtatása, és az eredmények kiértékelése. Felelős a szoftver termékekkel kapcsolatos hibák megtalálásáért.

A szoftverminőség-biztosítás a szoftverfejlesztés szerves részét képezi. A szoftverfejlesztési életciklus (SDLC) minden részéhez kapcsolódik. Megállapítja, hogy a szoftver, mint termék teljesíti-e a meghatározott követelményeket, megfelel-e a célnak. Statikus és dinamikus folyamat is lehet, a szabályoknak, előírásoknak megfelelőségét biztosítja. *(A tesztelők azonosítják a pontokat, ahol a rendszer nem az elvárt módon működik).*

**Tesztelésre szükség van, mert:**

* a szoftverben mindig van hiba, hisz emberek fejlesztik
* a hibáknak mindig ára van (hírnév, költség és idő)
* a termékben lévő hibákat még az üzembe helyezés előtt (mielőtt a végfelhasználók használni kezdenék) meg kell találni, hogy a termék minőségét, megbízhatóságát növeljük

**A tesztelés segít:**

* meghatározni, hogy a termék teljesíti-e a meghatározott követelményeket és célokat
* a szoftver fejlesztése, karbantartása és működése során fellépő programhibák és ezekkel járó meghibásodások megtalálásában
* azért használjuk, hogy segítsen csökkenteni a meghibásodások kockázatát egy működő környezetben és hozzájáruljon a szoftver jobb minőségéhez (a tesztelés a fejlesztési kockázatok csökkentésének egyik eszköze)
  + **kockázat**: ami még nem történt meg és lehet, hogy nem is fog soha megtörténni, de a jövőben negatív következményeket okozhat ezért lehetséges problémaforrás, amit kezelni kell! A kockázat kiszámítható: „hatás” és „valószínűség” formájában jelenik meg, ezért vizsgálni kell a lehetséges hatásokat és a bekövetkezés valószínűségét meg kell minden esetben becsülni és menedzselni az adott kockázatot.

#### Általános tesztelési tevékenységek:

* Tesztelés tervezése
* Tesztelés irányítása
* Teszt feltételek megválasztása
* Feltételek alapján tesztesetek megtervezése
* Tesztesetek végrehajtása
* Eredmény ellenőrzése
* Teljesítési és kilépési feltételek vizsgálata
* Tesztelési folyamatról és rendszerről jelentés
* Teljesítési (összefoglaló) jelentés készítése

### Tesztelési alapelvek

#### Korai tesztelés

A tesztelési ciklust a lehető legkorábban el kell kezdeni.

A szoftver- vagy rendszerfejlesztési életciklus legkorábbi stádiumában kezdjük el, és az előre meghatározott célokra összpontosítsunk – pl. dokumentációk tesztje, komponensek, integráció és rendszer tesztelése) Fő célja: a lehető legtöbb meghibásodást idézze elő

Gazdaságosság: könnyen és olcsón javítható

#### A tesztelés függ a körülményektől

A különböző alkalmazásokat különbözőképpen kell tesztelni.

Ha a biztonság kritikus szempont, másképp kell tesztelni, mint egy e-kereskedelmi oldalt, vagy egy telefonos alkalmazást – pl.: háttérbe le lehet rakni telefonon, több típusú telefonon is ki kell próbálni; másképp tesztelünk ha csak 1 napunk lenne letesztelni valamint, mintha 5 napunk)

#### Hibák csoportosulása

Ahol már találtunk hibát/hibákat nagy valószínűséggel többet is találhatunk.

A véges idő miatt a tesztelést a legfontosabb modulokra *(amely nélkül használhatatlan a program)* kell végrehajtani. A hibák eloszlása nem egyenletes a programban, a megtalált hibák többsége a legösszetettebb modulban van illetve a legújabb modulokban időszakosan az élesítést követően mindig több van mint később, amikor már használatban van egy ideje.

#### Féregirtó paradoxon

Ha mindig ugyanazokat a teszteket futtatjuk, egy idő után az azonos tesztkészlettel nem találunk új hibát.

Az újratesztelés során (pl. regressziós tesztnél) a teszteket, tesztadatokat idővel felülvizsgálni, változtatni, bővíteni kell.

#### Hibák látszólagos hiánya

A tesztelés csak a programhibák jelenlétét jelzi. A tesztelés képes felfedni a hibákat, de azt nem, hogy nincs hiba.

A teszteléssel csökken az esélye, hogy a szoftverben felfedezetlen programhibák maradnak, de ha nem találunk hibát, az nem a rendszer hibátlanságának bizonyítéka. Bármennyi fehér hattyút látunk, nem állíthatjuk, hogy minden hattyú fehér.

#### Nem lehetséges kimerítő tesztelés

Mindennek ​​minden kombinációját nem lehet kipróbálni.

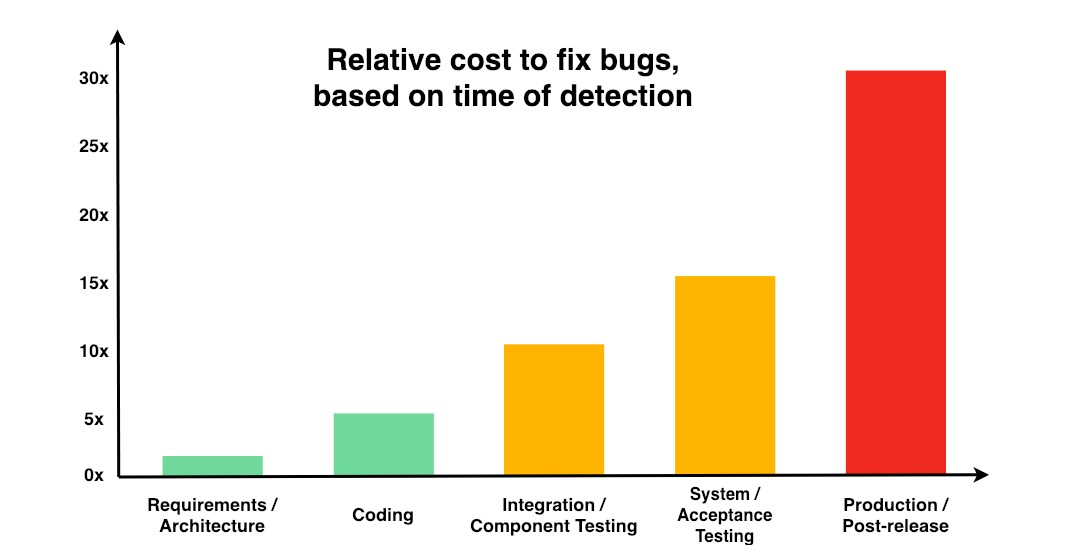
Azaz a bementek és előfeltételek minden kombinációjára – kiterjedő tesztelés nem lehetséges *(helyette elemezzük a kockázatokat és állítsunk fel fontossági sorrendet az összpontosítás hatékonysága érdekében)*

#### Hibátlan rendszer téveszméje

A hibák megtalálása és kijavítása hasztalan, ha a fejlesztett rendszer használhatatlan, vagy nem felel meg a felhasználói igényeknek, elvárásoknak.

### Megtérül a tesztelés?

**ROI: Return on Investment** - annak a megállapítása, hogy a befektetés mikor térül meg a többihez képest



Tehát azt lehet mondani, hogy igen megtérül, a hiba kijavításának költsége annál nagyobb, minél később találjuk meg. Akárcsak ha arra gondolunk, hogy ha újra kell végezni több folyamatot + összes addigi tesztet meg kell ismételni.

Célszerű már a tervezési fázisban bevonni a tesztelőket, egyrészt az elkészült dokumentumok ellenőrzésére, másrészt a tesztelhető kritériumok megfogalmazására. Ez a **shift-left testing**.

Ha látszólag a rendszerterv követelményeinek megfelelően alakul a fejlesztés, de hibás követelmények vagy hibás rendszerterv alapján kezdődik meg, akkor a fejlesztés során születendő termék bukásra van ítélve.

### Mi az a hiba?

Két fő szempont szerint tudjuk csoportosítani a hibákat.

# folyt

## A 3 fő kérdéscsoport:

### Tesztelési technikák

1.Black-box testing

Ekvivalenciaparticionálás

Határérték elemzés

Döntési tábla tesztelés

Állapotátmenet tesztelés

Használati eset tesztelés

2.White-box testing

Utasítás és tesztlefedettség

Döntési tesztelés és lefedettség

3.Gray-box testing

Statikus tesztelés

Dinamikus tesztelés

4.Experience-based testing

Hibasejtés

Felderítő tesztelés

Ellenőrző lista alapú tesztelés

### Tesztelési szintek

1.Egységteszt - Unit test

2.Komponens teszt - Component test

API tesztelés

3.Komponens kapcsolati teszt vagy más néven Integrációs teszt - Integration test

Integrációs tesztelés

Pact Contract tesztelés

4. Rendszerteszt - System test

Regressziós tesztelés

Release tesztelés

Smoke tesztelés

Sanity tesztelés

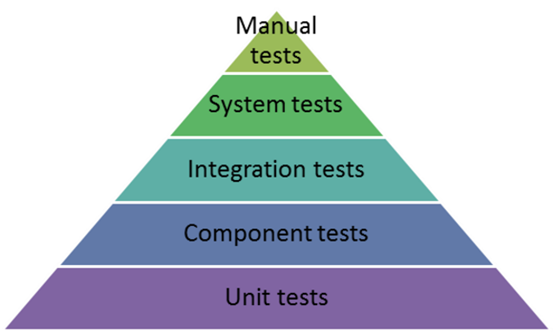
5.Elfogadási/átvételi teszt - Acceptance test

Felhasználói elfogadási tesztelés (UAT)

Üzemeltetési elfogadási tesztelés (OAT)

Szerződéses és szabály alapú elfogadási tesztelés

Alfa és béta tesztelés



### Teszt típusok

1.Funkcionális tesztelés

2.Nem funkcionális tesztelés

Teljesítményteszt (Performance test)

Stressztest (Stress test)

Terheléses teszt (Load test)

Kitartási teszt (Endurance testing)

Csúcsteszt (Spike test)

Skálázhatósági test (Scalability test)

Konkurencia teszt (Concurrency Testing)

Kapacitás tesztelés (Capacity Testing)

Használhatósági teszt (Usability test)

Nemzetköziesség teszt (i18n)

Biztonsági teszt (Security test)

Kompatibilitás teszt (Compatibility testing)

Accessibility test (a11y)

a kockázat biztosan kérdés lesz a modulzárón:

**kockázat**: ami még nem történt meg és lehet, hogy nem is fog soha megtörténni, de a jövőben negatív következményeket okozhat ezért lehetséges problémaforrás, amit kezelni kell! A kockázat kiszámítható: „hatás” és „valószínűség” formájában jelenik meg, ezért vizsgálni kell a lehetséges hatásokat és a bekövetkezés valószínűségét meg kell minden esetben becsülni és menedzselni az adott kockázatot.

mit kell tartalmaznia egy hibariportnak?

**Tartalmaznia kell:**

* Hiba neve és azonosítója
* Terület ahol jelentkezik a hiba
* **Alkalmazás verziószáma**
* **Súlyosság/fontosság**
* **Környezet**
* Személy aki találta
* Előfordulásának ideje
* **Hiba reprodukálásának lépései**
* **Aktuális eredmény**
* **Elvárt eredmény**
* **Logok/screenshotok**
* Teszt adat (test user)
* Platform/Böngésző és böngésző verzió
* Egyéb (linkelt teszt eset, linkelt fejlesztés, megjegyzés…stb)

------------

# 2. modul

## Programming Basics

## OOP Basics

oop basics, programming basics, java basics, java core, junit, sql basics

statikus modellek: a különbség az, h egy megvalósítás tervét szeretnénk leírni , vagy azt h hogyan működjön (dinamikus modellek)

Az állapot diagrammban (amit hívnak állapot-átmenet diagrammnak is) az ovális alakzat kezdő vagy végállapotot, a téglalap az eseményt, a rombusz pedig egy feltételt jelent.

instrukció:

mind a 6 diagrammnak egy kicsit utánaolvasni

----------------------

interjún általában kérdezik(tesztelésről):

* milyen tesztelési tecnikák vannak?
* milyen szintek vannak?
* milyen teszttípusok vannak?
* milyen kötelező elemei vannak a teszttervenek?
* milyen kötelező elemei vannak a tesztesetnek?

A dokumentációk közül ezt a kettőt kérdezik meg a legtöbbször (azon belül is inkább a hibariportot):

* milyen fontos jellemzői vannak a hibariportnak? (mit kell beletenni?)
* milyen fontos jellemzői vannak a tesztjegyzőkönyvnek? (mit kell beletenni?)

interjún általában kérdezik(OOP-ról):

* milyen OOP alapelvek vannak? és ezek rövid jellemzői

# Markdown Cheatsheet

Adam Pritchard edited this page on Mar 27, 2022 · [97 revisions](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet/_history)

### Pages 10



* [Home](https://github.com/adam-p/markdown-here/wiki)
* [Compatibility](https://github.com/adam-p/markdown-here/wiki/Compatibility)
* [Development Notes](https://github.com/adam-p/markdown-here/wiki/Development-Notes)
* [Markdown Cheatsheet](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)
  + [Table of Contents](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#table-of-contents)
  + [Headers](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#headers)
  + [H1](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h1)
  + [H2](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h2)
  + [H3](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h3)
  + [H4](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h4)
  + [H5](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h5)
  + [H6](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h6)
  + [Alt-H1](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#alt-h1)
  + [Alt-H2](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#alt-h2)
  + [Emphasis](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#emphasis)
  + [Lists](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#lists)
  + [Links](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#links)
  + [Images](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#images)
  + [Code and Syntax Highlighting](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#code-and-syntax-highlighting)
  + [Footnotes](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#footnotes)
  + [Tables](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables)
  + [Blockquotes](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#blockquotes)
  + [Inline HTML](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#inline-html)
  + [Horizontal Rule](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#horizontal-rule)
  + [Line Breaks](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#line-breaks)
  + [YouTube Videos](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#youtube-videos)
* [Markdown Here Cheatsheet](https://github.com/adam-p/markdown-here/wiki/Markdown-Here-Cheatsheet)
* [Other Markdown Tools](https://github.com/adam-p/markdown-here/wiki/Other-Markdown-Tools)
* [Press, Posts, Reviews, Etc.](https://github.com/adam-p/markdown-here/wiki/Press,-Posts,-Reviews,-Etc.)
* [Reviews](https://github.com/adam-p/markdown-here/wiki/Reviews)
* [Tips and Tricks](https://github.com/adam-p/markdown-here/wiki/Tips-and-Tricks)
* [Troubleshooting](https://github.com/adam-p/markdown-here/wiki/Troubleshooting)

##### Clone this wiki locally



This is intended as a quick reference and showcase. For more complete info, see [John Gruber's original spec](http://daringfireball.net/projects/markdown/) and the [Github-flavored Markdown info page](http://github.github.com/github-flavored-markdown/).

Note that there is also a [Cheatsheet specific to Markdown Here](https://github.com/adam-p/markdown-here/wiki/Markdown-Here-Cheatsheet) if that's what you're looking for. You can also check out [more Markdown tools](https://github.com/adam-p/markdown-here/wiki/Other-Markdown-Tools).

##### [Table of Contents](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#table-of-contents)

[Headers](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#headers)  
[Emphasis](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#emphasis)  
[Lists](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#lists)  
[Links](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#links)  
[Images](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#images)  
[Code and Syntax Highlighting](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#code)  
[Footnotes](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#footnotes)  
[Tables](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables)  
[Blockquotes](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#blockquotes)  
[Inline HTML](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#html)  
[Horizontal Rule](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#hr)  
[Line Breaks](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#lines)  
[YouTube Videos](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#videos)

## [Headers](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "headers)

# H1

## H2

### H3

#### H4

##### H5

###### H6

Alternatively, for H1 and H2, an underline-ish style:

Alt-H1

======

Alt-H2

------

# [H1](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h1)

## [H2](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h2)

### [H3](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h3)

#### [H4](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h4)

##### [H5](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h5)

###### [H6](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#h6)

Alternatively, for H1 and H2, an underline-ish style:

# [Alt-H1](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#alt-h1)

## [Alt-H2](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#alt-h2)

## [Emphasis](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "emphasis)

Emphasis, aka italics, with \*asterisks\* or \_underscores\_.

Strong emphasis, aka bold, with \*\*asterisks\*\* or \_\_underscores\_\_.

Combined emphasis with \*\*asterisks and \_underscores\_\*\*.

Strikethrough uses two tildes. ~~Scratch this.~~

Emphasis, aka italics, with asterisks or underscores.

Strong emphasis, aka bold, with **asterisks** or **underscores**.

Combined emphasis with **asterisks and underscores**.

Strikethrough uses two tildes.

## [Lists](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "lists)

(In this example, leading and trailing spaces are shown with with dots: ⋅)

1. First ordered list item

2. Another item

⋅⋅\* Unordered sub-list.

1. Actual numbers don't matter, just that it's a number

⋅⋅1. Ordered sub-list

4. And another item.

⋅⋅⋅You can have properly indented paragraphs within list items. Notice the blank line above, and the leading spaces (at least one, but we'll use three here to also align the raw Markdown).

⋅⋅⋅To have a line break without a paragraph, you will need to use two trailing spaces.⋅⋅

⋅⋅⋅Note that this line is separate, but within the same paragraph.⋅⋅

⋅⋅⋅(This is contrary to the typical GFM line break behaviour, where trailing spaces are not required.)

\* Unordered list can use asterisks

- Or minuses

+ Or pluses

1. First ordered list item
2. Another item

* Unordered sub-list.

1. Actual numbers don't matter, just that it's a number
2. Ordered sub-list
3. And another item.

You can have properly indented paragraphs within list items. Notice the blank line above, and the leading spaces (at least one, but we'll use three here to also align the raw Markdown).

To have a line break without a paragraph, you will need to use two trailing spaces.  
Note that this line is separate, but within the same paragraph.  
(This is contrary to the typical GFM line break behaviour, where trailing spaces are not required.)

* Unordered list can use asterisks
* Or minuses
* Or pluses

## [Links](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "links)

There are two ways to create links.

[I'm an inline-style link](https://www.google.com)

[I'm an inline-style link with title](https://www.google.com "Google's Homepage")

[I'm a reference-style link][Arbitrary case-insensitive reference text]

[I'm a relative reference to a repository file](../blob/master/LICENSE)

[You can use numbers for reference-style link definitions][1]

Or leave it empty and use the [link text itself].

URLs and URLs in angle brackets will automatically get turned into links.

http://www.example.com or <http://www.example.com> and sometimes

example.com (but not on Github, for example).

Some text to show that the reference links can follow later.

[arbitrary case-insensitive reference text]: https://www.mozilla.org

[1]: http://slashdot.org

[link text itself]: http://www.reddit.com

[I'm an inline-style link](https://www.google.com/)

[I'm an inline-style link with title](https://www.google.com/)

[I'm a reference-style link](https://www.mozilla.org/)

[I'm a relative reference to a repository file](https://github.com/adam-p/markdown-here/blob/master/LICENSE)

[You can use numbers for reference-style link definitions](http://slashdot.org/)

Or leave it empty and use the [link text itself](http://www.reddit.com/).

URLs and URLs in angle brackets will automatically get turned into links. [http://www.example.com](http://www.example.com/) or [http://www.example.com](http://www.example.com/) and sometimes example.com (but not on Github, for example).

Some text to show that the reference links can follow later.

## [Images](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "images)

Here's our logo (hover to see the title text):

Inline-style:

![alt text](https://github.com/adam-p/markdown-here/raw/master/src/common/images/icon48.png "Logo Title Text 1")

Reference-style:

![alt text][logo]

[logo]: https://github.com/adam-p/markdown-here/raw/master/src/common/images/icon48.png "Logo Title Text 2"

Here's our logo (hover to see the title text):

Inline-style: alt text

Reference-style: alt text

## [Code and Syntax Highlighting](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "code-and-syntax-highlighting)

Code blocks are part of the Markdown spec, but syntax highlighting isn't. However, many renderers -- like Github's and Markdown Here -- support syntax highlighting. Which languages are supported and how those language names should be written will vary from renderer to renderer. Markdown Here supports highlighting for dozens of languages (and not-really-languages, like diffs and HTTP headers); to see the complete list, and how to write the language names, see the [highlight.js demo page](http://softwaremaniacs.org/media/soft/highlight/test.html).

Inline `code` has `back-ticks around` it.

Inline code has back-ticks around it.

Blocks of code are either fenced by lines with three back-ticks ```, or are indented with four spaces. I recommend only using the fenced code blocks -- they're easier and only they support syntax highlighting.

```javascript

var s = "JavaScript syntax highlighting";

alert(s);

```

```python

s = "Python syntax highlighting"

print s

```

```

No language indicated, so no syntax highlighting.

But let's throw in a <b>tag</b>.

```

var s = "JavaScript syntax highlighting";

alert(s);

s = "Python syntax highlighting"

print s

No language indicated, so no syntax highlighting in Markdown Here (varies on Github).

But let's throw in a <b>tag</b>.

## [Footnotes](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "footnotes)

Footnotes aren't part of the core Markdown spec, but they [supported by GFM](https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#footnotes).

Here is a simple footnote[^1].

A footnote can also have multiple lines[^2].

You can also use words, to fit your writing style more closely[^note].

[^1]: My reference.

[^2]: Every new line should be prefixed with 2 spaces.

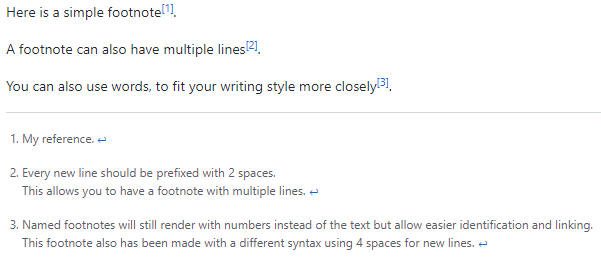
This allows you to have a footnote with multiple lines.

[^note]:

Named footnotes will still render with numbers instead of the text but allow easier identification and linking.

This footnote also has been made with a different syntax using 4 spaces for new lines.

Renders to:



## [Tables](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "tables)

Tables aren't part of the core Markdown spec, but they are part of GFM and Markdown Here supports them. They are an easy way of adding tables to your email -- a task that would otherwise require copy-pasting from another application.

Colons can be used to align columns.

| Tables | Are | Cool |

| ------------- |:-------------:| -----:|

| col 3 is | right-aligned | $1600 |

| col 2 is | centered | $12 |

| zebra stripes | are neat | $1 |

There must be at least 3 dashes separating each header cell.

The outer pipes (|) are optional, and you don't need to make the

raw Markdown line up prettily. You can also use inline Markdown.

Markdown | Less | Pretty

--- | --- | ---

\*Still\* | `renders` | \*\*nicely\*\*

1 | 2 | 3

Colons can be used to align columns.

| **Tables** | **Are** | **Cool** |
| --- | --- | --- |
| col 3 is | right-aligned | $1600 |
| col 2 is | centered | $12 |
| zebra stripes | are neat | $1 |

There must be at least 3 dashes separating each header cell. The outer pipes (|) are optional, and you don't need to make the raw Markdown line up prettily. You can also use inline Markdown.

| **Markdown** | **Less** | **Pretty** |
| --- | --- | --- |
| Still | renders | **nicely** |
| 1 | 2 | 3 |

## [Blockquotes](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "blockquotes)

> Blockquotes are very handy in email to emulate reply text.

> This line is part of the same quote.

Quote break.

> This is a very long line that will still be quoted properly when it wraps. Oh boy let's keep writing to make sure this is long enough to actually wrap for everyone. Oh, you can \*put\* \*\*Markdown\*\* into a blockquote.

Blockquotes are very handy in email to emulate reply text. This line is part of the same quote.

Quote break.

This is a very long line that will still be quoted properly when it wraps. Oh boy let's keep writing to make sure this is long enough to actually wrap for everyone. Oh, you can put **Markdown** into a blockquote.

## [Inline HTML](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "inline-html)

You can also use raw HTML in your Markdown, and it'll mostly work pretty well.

<dl>

<dt>Definition list</dt>

<dd>Is something people use sometimes.</dd>

<dt>Markdown in HTML</dt>

<dd>Does \*not\* work \*\*very\*\* well. Use HTML <em>tags</em>.</dd>

</dl>

*Definition list*

Is something people use sometimes.

*Markdown in HTML*

Does \*not\* work \*\*very\*\* well. Use HTML tags.

## [Horizontal Rule](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "horizontal-rule)

Three or more...

---

Hyphens

\*\*\*

Asterisks

\_\_\_

Underscores

Three or more...

Hyphens

Asterisks

Underscores

## [Line Breaks](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "line-breaks)

My basic recommendation for learning how line breaks work is to experiment and discover -- hit <Enter> once (i.e., insert one newline), then hit it twice (i.e., insert two newlines), see what happens. You'll soon learn to get what you want. "Markdown Toggle" is your friend.

Here are some things to try out:

Here's a line for us to start with.

This line is separated from the one above by two newlines, so it will be a \*separate paragraph\*.

This line is also a separate paragraph, but...

This line is only separated by a single newline, so it's a separate line in the \*same paragraph\*.

Here's a line for us to start with.

This line is separated from the one above by two newlines, so it will be a separate paragraph.

This line is also begins a separate paragraph, but...  
This line is only separated by a single newline, so it's a separate line in the same paragraph.

(Technical note: Markdown Here uses GFM line breaks, so there's no need to use MD's two-space line breaks.)

## [YouTube Videos](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet" \l "youtube-videos)

They can't be added directly but you can add an image with a link to the video like this:

<a href="http://www.youtube.com/watch?feature=player\_embedded&v=YOUTUBE\_VIDEO\_ID\_HERE

" target="\_blank"><img src="http://img.youtube.com/vi/YOUTUBE\_VIDEO\_ID\_HERE/0.jpg"

alt="IMAGE ALT TEXT HERE" width="240" height="180" border="10" /></a>

Or, in pure Markdown, but losing the image sizing and border:

[![IMAGE ALT TEXT HERE](http://img.youtube.com/vi/YOUTUBE\_VIDEO\_ID\_HERE/0.jpg)](http://www.youtube.com/watch?v=YOUTUBE\_VIDEO\_ID\_HERE)

Referencing a bug by #bugID in your git commit links it to the slip. For example #1.

License: [CC-BY](https://creativecommons.org/licenses/by/3.0/)