# Évesmunka

Csete Andor

## A kezdetek

Évesmunkám témája már kisebb korom óta foglalkoztatott. Valamit alkotni néhány sor szövegből, ami aztán bármilyen probléma megoldására alkalmas, felcsigázott. Programozni kb. 2018. őszén kezdtem a Scratch[1] nevű, ingyenesen használható programmal, ami egy “blokkos” programozási nyelv, amely azt takarja, hogy vizuális elemeket, blokkokat kell egymáshoz húzogatni az egérrel, hogy a megfelelő kimenetet kapjuk. A Scratch-ben főleg 2D-s animációkat, játékokat és appokat lehet csinálni. Ne gondoljuk, hogy ezzel meg tudjuk váltani a világot, de tanuláshoz tökéletes.
A másik “blokkos” programozói felület a LEGO Mindstorms EV3[2] volt. Ez egy programozható lego tégla, amit számítógép, tablet és mobiltelefonról lehet programozni. A Scratch-el ellentétben ennek a kimenete nem egy 2D-s program, hanem egy legóból összeépített lego-robot. A csomagban vannak szenzorok (szín-, fény-, nyomásérzékelők) és motorok az építő elemek mellet. Ezeket a motorokat, szenzorokat lehet beprogramozni, ha például megnyomunk egy gombot a roboton, az előre megy, a motorokhoz erősített kerekek segítségével. Ezt kihasználva és egy kis fantáziával megspékelve, majdnem bármit létrehozhatunk (Lásd rubik kocka kirakó robot[3], rajongói robotok[4]), de ha ehhez nincs elég kreativitásunk, vagy tapasztalatunk “lophatunk” ötletet a youtube-ról is (sőt az ELTE-nek is van szakköre a Mindstorms-al kapcsolatban.)
Miután megtapasztaltam a programozás sokszínűségét, elkezdtem unni a blokkos programozást. Ekkor kezdtem el keresni, hogyan is lehet szövegesen programozni. Rátaláltam a Grasshopper[5] nevezetű appra. Itt megismertem a szöveges programozást. A Grasshopper-en kisebb feladatok vannak, melyeket megoldva a következő leckéhez ugorhatunk. Így egyre nehezebb feladatokat kapunk. Egyszerre csak néhány (5-6) feladatot tudtam megoldani, és egy hibán nem tudtam továbblendülni. Egy-két napra rá, mikor visszatértem a Grasshopperre, első próbára meg tudtam oldani a problémát és újabb 5-6 feladatot tudtam megoldani.
Egy új korszak kezdődött el a programozási tudásomban, hisz elkezdtem járni egy python[6] tanfolyamra. Itt az első félévben elkezdtünk foglalkozni a Minecraft Education Edition-nel[7], ami egy, a minecraft nevű sandbox játékra épülő programozási felület. Ezt jobban kifejtve:

---

[1] Grafikus programozói felület főleg gyerekeknek. Ingyenes, online és applikáció, magyarul is használható (https://scratch.mit.edu/)
[2] A LEGO által kiadott lego-szett, melyben motorok, szenzorok és a fő egység melletépítő elemek is találhatók, melyek segítségével és/vagy más elemek használatával egy tetszőleges kinézetű és funkcionalitású robotot lehet létrehozni. (https://www.lego.com/hu-hu/themes/mindstorms/ev3)
[3] Itt egy Blikk cikk a lego rubik kocka kirakó robotról:
https://www.blikk.hu/eletmod/lusta-vagy-kirakni-a-rubik-kockat-video/fl674bh
[4] https://www.lego.com/hu-hu/themes/mindstorms/fanrobots
[5] A Grasshopper egy programozás tanulásra használt app/weboldal, melyen ingyenesen tudunk szövegesen programozni. JavaScript-et lehet tanulni vele főként (https://grasshopper.app/)
[6] A Python a világ egyik legelterjedtebb programozási nyelve. Ezt egyszerűsége és sokféle felhasználhatósága adja. (https://www.python.org/)
[7] A Microsoft által kiadott, programozást fejlesztő játék. Sajnos személyes használatra nem alkalmas, iskolai fiók kell hozzá. (https://education.minecraft.net/en-us)

*“A Minecraft Education szakköre a népszerű Minecraft játék oktatási változatára épül, és annak világán belül vezet végig a programozás alapjain”*

**-Funside**[8]

## Python

A kurzus második félévében elkezdtük használni az ugyancsak Microsoft által fejlesztett Visual Studio Code-ot[9] (továbbiakban: VS code). Mely egy elterjedt fejlesztői környezet. Itt elkezdtük a hagyományos programozást, egy népszerű példával:

```
print("hello world")
```

mely arra szolgál, hogy a terminal-ba/console-ba kiírja, hogy *hello world.* Ez igen hasznos, de egy idő után meguntam, kiírni mindig ugyanazt. Amikor megismertem az *input()* függvényt, úgy éreztem, hogy elérkezett a megváltó. Az *input()* függvényt arra lehet használni, hogy a felhasználó megadjon egy bizonyos értéket, szöveget a terminálba/console-ba.
A pythont azért szerettem meg, mint sokan mások, mert egyszerű, de mégis sokféleképpen lehet használni. Habár az igaz, hogy a programnyelvek arra valók, hogy bármit el tudjunk készíteni, amihez megvan a tapasztalatunk és/vagy tudásunk, a python még ezen is túltett. Jó volt a kurzusra járni, hisz ha elakadtam egy személyes projektben, segítséget tudtam kérni az oktatóktól. A kurzus menete kissé lassú volt nekem, mert én szinte rögtön megértettem mindent, és az önálló feladatokban is gyorsabban haladtam a többieknél. Miután kész voltam az önálló feladattal, tovább fejlesztettem, bővítettem, vagy saját programba kezdtem. Ilyen személyes projekt a **MathXP** (többféle elnevezést is adtam ennek a programnak. Pl.: GameNum, PyMath, kódnevén PXP). A MathXP egy egyszerű *konzolos* játék, melyben a gép kisorsol két véletlen számot, a számok nagysága a játékos aktuális pontszámától függött. (1 pont: 1-10-ig terjedő számok) Ezután kisorsolt egy műveletet (+, -, *, /), majd ki írta a konzolra. A játékosnak ezt kellet megfejtenie. Ha jól találta el kapott egy pontot, ha nem, akkor se nem vesztett se nem nyert pontot. Ezt a játékot később rengetegszer bővítettem. A MathXP után jött a MathXP 1.1, kis fejlesztés után jött a MathXP 2.3-as széria melynek volt egy pár tagja. Később létrehoztam a MathXP 3.0-át. Ez nem volt akkora fejlesztés, mint a 2.3, de tartalmazott új tartalmakat, és hibajavításokat. A legújabb verzió a MathXP 4.0a (*a*, mint alfa). Az 4.0a egy vadonatúj forráskóddal, bejelentkezési lehetőséggel (mely az pontok megőrzésére szükséges) és még néhány hibajavítással bővült. Ez soha nem lett teljesen kész.
A MathXP első szakaszában megjelent egy nyelv választási lehetőség, mely három választási lehetőséget biztosít; Magyar, Angol és Német nyelvek közül lehetett választani. (Az angol és a német nem tökéletes fordításai a magyarnak.)
Új célom egy *MathXP Remastered* kifejlesztése. Remélem ezt még az évesmunka előadásokig le tudom programozni és be tudom mutatni.
Egy másik szabad programom a **Theofilos’ game** volt. (A Theofilos név a teológia és filozófia összeolvasztása. Ennek oka a játékban szereplő karakter képessége. A név egyben egy görög festőművész neve, és görögül az *istenes* szót is jelenti. Erre csak a játék elnevezése után jöttem rá) Ebben a játékban választhatunk, hogy gondolunk vagy kitalálunk.

---

[8] A Funside Schools délutáni Python kurzusára jártam 2x Félévet. (https://www.funside.hu/)
[9] A Visual Studio Code egy ingyenes, multiplatform, fejlesztőbarát app programozásra. Elterjedt a programozók körében sokféle, praktikus funkciója miatt. (https://code.visualstudio.com/)

Ha gondolunk 1 és 100 között kell gondolnunk egy számot, akkor Theofilos megpróbálja kitalálni. Tippel egy számra, és nekünk csak annyit kell megmondani, hogy kisebb, nagyobb vagy egyenlő. Viszont, amikor a kitalálást választjuk, Theofilos gondol egy számra 1 és 100 között. Azt is elmondja, hogy páros, vagy páratlan. Mi csak tippelünk egy számot, és elárulja, hogy kisebb, nagyobb, vagy egyenlő számra gondolt. A gondolót és a találót külön pontozza a rendszer attól függ, hogy hány találat kell a szám kitalálásához. Legalább két menetet kell játszanunk, hogy kiderüljön számunkra, ki is az okosabb? Theofilos vagy te?

### Python GUI[10]

Mivel ezeknek és az ezekhez hasonló programok felhasználói felületét a konzol alkotta, mely kezelhetetlen, ronda és átláthatatlan, meguntam és megutáltam. Szerencsére a python kurzus II. féléve vége felé belekostoltunk a GUI-ba. Mely pontosan megfogalmazva:

*“A grafikus felhasználói felület vagy grafikus felhasználói interfész a számítástechnikában olyan, a számítógép és ember közti kapcsolatot megvalósító elemek összessége, melyek a monitor képernyőjén szöveges és rajzos elemek együtteseként jelennek meg.”*

**-Wikipédia**

Segítségével létrehoztunk egy ablakot, melyen szerepeltek gombok és egyebek, amelyekkel a felhasználó tud kommunikálni a számítógéppel.
Elkezdtem egyszerű appokat csinálni: számológép és hasonlók.
A python kurzusban volt két projektmunka, amelyek nagyban hasonlítanak az évesmunkára, csak sokkal kisebb kaliberűnek számítanak. Az első ilyen projektmunkám a MathXP egy fejlett változata volt. A második az un. *BlogMaker* programom. Az évesmunkám a BlogMaker újratervezése, grafikus változata és feljavítása, hogy felhasználó barát legyen. A BlogMaker egy primitív, konzolos program, melyen paraméterek megadásával (weboldal háttérszíne, betűméret, stb.) lehet egy weblapot létrehozni. Átlag ember számára szinte kezelhetetlen, hiszen a színeket vagy angolul, vagy hexadecimális színkóddal kell megadni, amelyről a világ fele nem is hallott. (pl.: *#fff*, mely a fehér színt jelöli.)
A programozás már ekkor teljesen beszippantott. A telefonomra több appot is letöltöttem. Az egyik ilyenből megtanultam a HTML[11] alapjait.

## A webfejlesztés

A HTML a webfejlesztés alapja, hivatalosan nem egy programozási nyelv, csak egy leíró szöveg. Leírja, hogyan is nézzen ki egy weboldal. Egy weboldalnak három fő alapja van: az első a HTML, amely olyan, mint az embernél a csontváza. Önmagában nem szemetgyönyörködtető. Arra van, hogy a weboldal szövegét létre tudjuk hozni. A második a CSS[12], ami arra szolgál, hogy megszépítse a weboldalunkat. A építő elemek, melyeket a HTML határoz meg, a CSS szépíti meg. Az ember bőrére, hajára hasonlítanám. És végül, de nem utolsó sorban a JavaScript, melyet nem csak a webfejlesztésben használnak, hanem appok létrehozásában is. Ez arra felel hogy a HVG-n olvasott cikkben egy

---

[10] Graphical User Interface, Grafikus Felhasználói Felület. A felhasználó az egeret használva juttathat el adatokat a számítógéphez.
(*https://hu.wikipedia.org/wiki/Grafikus_felhaszn%C3%A1l%C3%B3i_fel%C3%BClet*)
[11] HyperText Markup Language, hiperszöveges jelölőnyelv, mely weboldalak létrehozásához készítettek. Viszonylag egyszerű a tanulása.
[12] Cascading Style Sheets, lépcsőzetes stíluslapok, egy stílusleíró nyelv, amellyel a HTML stílusát lehet megadni. (https://hu.wikipedia.org/wiki/Cascading_Style_Sheets)

gombnyomásra a lap tetejére ugorjunk. Ez adja meg a weblapok működését. Az emberi izmokhoz, idegekhez hasonlíthatóak.
Persze nem csak ez a három kellék kell egy weboldal üzemeltetésére. Kell egy domain, szerver, kellhetnek frameworkök és még sok más, de erről majd egy másik írásban.

HTML létrehozását angol középhaladó nyelvtudással magunktól megtanulhatjuk, ha már egy öt perces gyorstalpalót megnéztünk, hallgattunk, vagy olvastunk. A lényege, hogy az elemeket két kacsacsőr közé írjuk (pl.: <elem>), majd, ha az elemnek véget akarunk vetni, le akarjuk zárni, a bezárársi mintát kell alkalmaznunk (</elem>). Tehát az egész valahogy így néz ki: <elem>*szöveg*</elem>. Ha *dőlt* (angolul *italic*) betűkkel akarunk írni, ezt jeleznünk kell a számítógépnek, különben nem tudná. Ezt úgy tehetjük meg, ha az előbbi példában kicseréljük az *elem* szót a megfelelő jelzőre. A dőlt betűnél ez  (i=italic=dőlt). Az alábbi példa bemutatja, hogy mitől “dől”, “kövéredik”, vagy “húzódik alá” egy szó, vagy mondat egy weboldalon;

HTML: Ez egy dőlt mondat

Eredmény: *Ez egy dőlt mondat*

| Mit látunk? | angolul | HTML |
|---|---|---|
| *Dőlt* | italic |  |
| aláhúzott | underline | <u></u> |
| **Félkövér** | bold |  |
| kijelölt | marked | <mark></mark> |
| Felső indexbetű | superscript | <sup></sup> |
| Alsó indexbetű | subscript | <sub></sub> |

Ez csak pár a rengeteg HTML “tag”-nek. (tag=cédula, HTML tag=<*valami*>) Van olyan ami egy gombot hoz létre (<button></button>, button=gomb) van olyan amelyet nem látunk, de befolyásolja, hogy mások hogy nézzenek, vagy viselkedjenek. (<style>=stílus leíró (CSS), <script>=funkciók, akciók leírása (JavaScript)).

Most, hogy mindkét programozási nyelvbe belenéztünk következzék ezeknek a keveréke, koktélja.

## HTML és Python használata egy programban

Egy egyszerű programot fogok bemutatni, amely az alábbi probléma megoldására alkalmas:

Képzeljük el, hogy egy 20 oldalas újságot, cikket, magazint, vagy bármilyen más szöveget írtunk. Tegyük fel, hogy nem wordben írtunk, hanem egy egyszerű szövegszerkesztőben és a szöveget elmentettük egy *szoveg.txt* nevű fájlba. (a *.txt* fájl kiterjesztés az egyszerű szöveges fájl típusa. txt≈text=szöveg) Pythonban ezt úgy tudjuk beolvasni, ha az alábbi sorokat írjuk be:

```
with open("szoveg.txt") as file:

   szoveg=file.read()
```

Ez megnyitja a *szoveg.txt* nevű fájlt és eltárolja a benne található szöveget a -kék betűkkel írt- *szöveg* nevű **változóban**[13]. Mivel a *.txt* fájl kiterjesztés tartalma szöveges, ezért a változó szöveg -vagy *string**- változó lesz. A probléma, hogy elfelejtettünk aláhúzni, vagy valamilyen más formában kijelölni bizonyos szavakat. Ezt kézzel javítani legalább fél óra, ha nem több és a hibázás faktora nagyobb, mint nulla. Tehát alkotnunk kell egy probléma megoldó programot. (a programozás fő célja, hogy komoly -vagy apró- problémákra, nagy -vagy kicsi- számolásokra megoldást adjon rövid -vagy hosszú- időn belül) Ha keresni szeretnénk egy szöveg változóban, akkor azt megtehetjük a

```
szoveg.find()
```

függvénnyel (angolul *function*). A két zárójel közé (macskakörömmel, (továbbiakban *string*)) kell beírni, hogy mit szeretnénk keresni. (használhatunk egy változót, ha egy korábban megadott értéket keresünk, ebben az esetben eltekinthetünk a macskaköröm használatától). A *find()* függvény úgy működik, hogy a -zárójelek közé- megadott értéket keresi meg a változóban (szintaxisa[14]: átvizsgálandó szöveg.*find(keresendő érték)*), majd az érték előfordulásának első helyét adja meg. Amennyiben a keresendő érték nincs meg az átvizsgálandó szövegben, *-1*-es értéket ad vissza. Az érthetőség kedvéért itt egy példa:

```
szoveg="ez egy szöveg"

print(szoveg.find("z"))

>>1
```

[13] A programozásban használt változók -hasonlóan az algebrai ismeretlenek esetében- egy értéket tárolnak. Ezt változtathatjuk, vagy olvashatjuk. Többféle létezik belőle (list=lista, boolean=logikai érték, int=egész szám, *string=egyszerű szöveg, stb…) (https://hu.wikipedia.org/wiki/V%C3%A1ltoz%C3%B3#Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9p_programoz%C3%A1sa)

[14] A szintaxis azt a szabályt jelenti, amely leírja, milyen formában kell alkalmazni a szimbólumokat, a helyes kimenethez. (https://hu.wikipedia.org/wiki/Szintaxis_(programoz%C3%A1si_nyelvek)#:~:text=A%20sz%C3%A1m%C3%ADt%C3%A1stechnik%C3%A1ban%20a%20sz%C3%A1m%C3%ADt%C3%B3g%C3%A9pes%20nyelv,fel%C3%A9p%C3%ADtett%20dokumentumnak%20vagy%20t%C3%B6red%C3%A9knek%20tekinthet%C3%BCnk.)

A kimenet (>> jelölve) 1, hisz a*z* a második helyen található, ami a legtöbb programozási nyelvben az *1*-es index (index 0=1, index 1=2, és így tovább. A legtöbb programozási nyelvben a számolás nulláról indul). Sajnos ez a függvény nem annyira jön kapóra nekünk, hisz mi nem csak megkeresni szeretnénk, hanem át is szeretnénk írni. Sokkal inkább hasznos nekünk a

```
szoveg.replace()
```

függvény. (szintaxisa: átírandó szöveg.replace(“mit írjon át”, “mire írja át”)) Ez azt tudja, hogy a megadott értéket/értékeket megkeresi, majd átírja őket a második érték szerint. Példa:

```
szoveg="szeretem a barackot"

print(szoveg.replace("barackot", "banánt"))

>> szeretem a banánt
```

Kicseréli a *barackot* szót *banánt* szóra.

Mint azt korábban megtanulhattuk, a HTML-ben ha azt akarjuk, hogy egy szó, vagy mondat dőlt lenyen, azt egy *tag*-be kell tenni. Szóval szöveg = *szöveg*. Tegyük fel, hogy a cikkünkben, írásunkban mindig a *PHWI* szót akartuk kiemelni. Ezt a problémát pythonban úgy tudjuk megoldani, ha az alábbi kódot írjuk:

```
#nyissuk meg a fájlt, amiben a cikkünk van:

with open("szoveg.txt") as file:

   #olvassuk be a benne található szöveget, majd mentsük el egy változóba:

   valtozo=file.read()

#cseréljük ki a "PHWI" szót "<i>PHWI</i>"-re, hogy dőlt betűkkel legyen írva:

valtozo=valtozo.replace("PHWI","<i>PHWI</i>")

#írjuk át a fájl tartalmát a megújult szövegre:
```

```
with open("uj_szoveg.html","w") as file:

   file.write(valtozo)
```

Az átírt tartalmat -amelyet egy változóban tartunk- *.html* fájlba mentjük -vagy néha *.htm*-. Ha ezt a fájlt megnyitjuk -egy böngészőben- akkor láthatjuk, hogy az írásunkban minden PHWI át lesz írva *PHWI*-ra. Célunkat el is értük. De ha több ilyen szó van, amit át akarunk írni? Ezt úgy is meg lehet oldani, hogy több sort írunk, többször használjuk a *szoveg.replace()* függvényt. Tegyük fel, hogy nem csak a PHWI, hanem az *iskola, oktatás* és a *waldorf* szavakat is ki akarjuk jelölni -mondjuk dőlt betűkkel-. Ha halmozzuk a *replace()* függvényt, és mindegyikben megadjuk a szavakat, amiket át akarunk írni, akkor az időigényesebb lenne, nem lenne olyan elegáns és nem is lenne olyan hatékony, mint ez a modell:

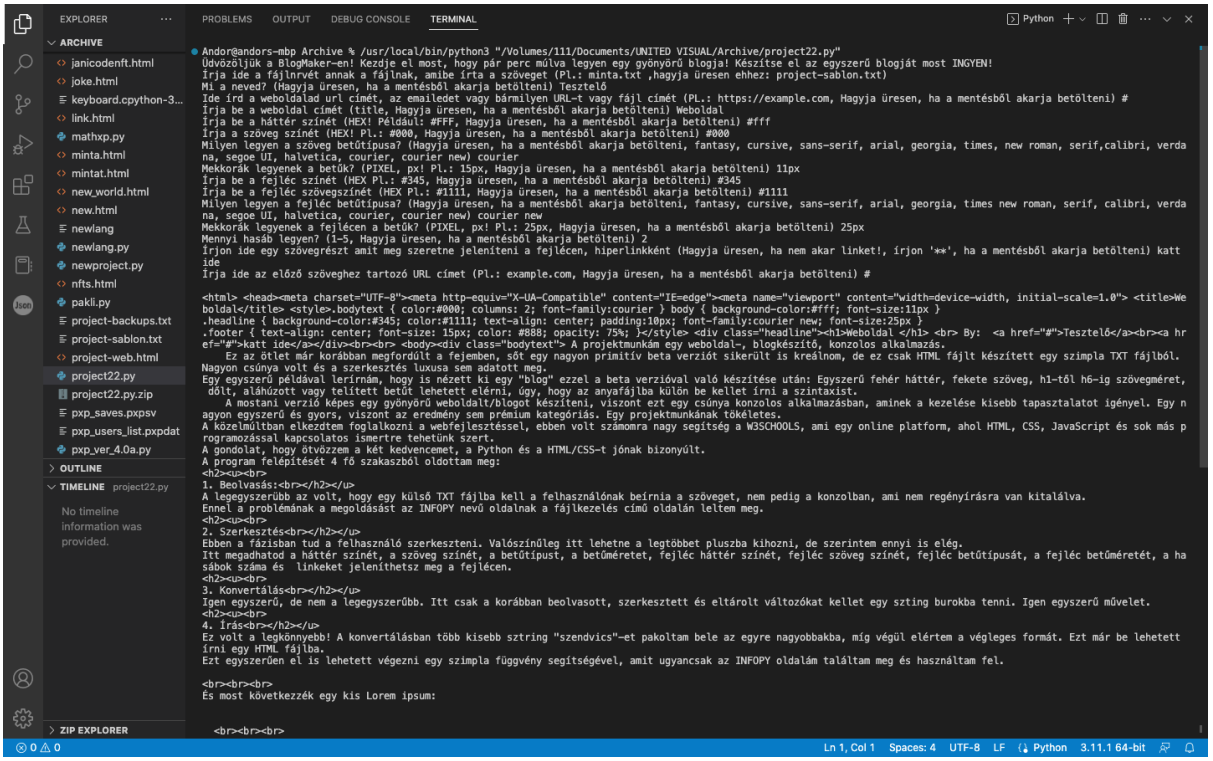
```
#a felhasználót megkérdezzük, hogy mit szeretne átírni: (atirando=átírandó)

atirando=input("Mit szeretnél átírni? ")

#nyissuk meg a fájlt, amiben a cikkünk van:

with open("szoveg.txt") as file:

   #olvassuk be a benne található szöveget, majd mentsük el egy változóba:

   valtozo=file.read()

#cseréljük ki a felhasználó által megadott szót, önmagára, html "körítéssel"
(<i></i>):

valtozo=valtozo.replace(atirando, "<i>"+atirando+"</i>")

#írjuk át a fájl tartalmát a megújult szövegre:

with open("uj_szoveg.html","w") as file:

   file.write(valtozo)
```

Ebben a programban az *átirandó* változónak a felhasználó által megadott értéket adunk,

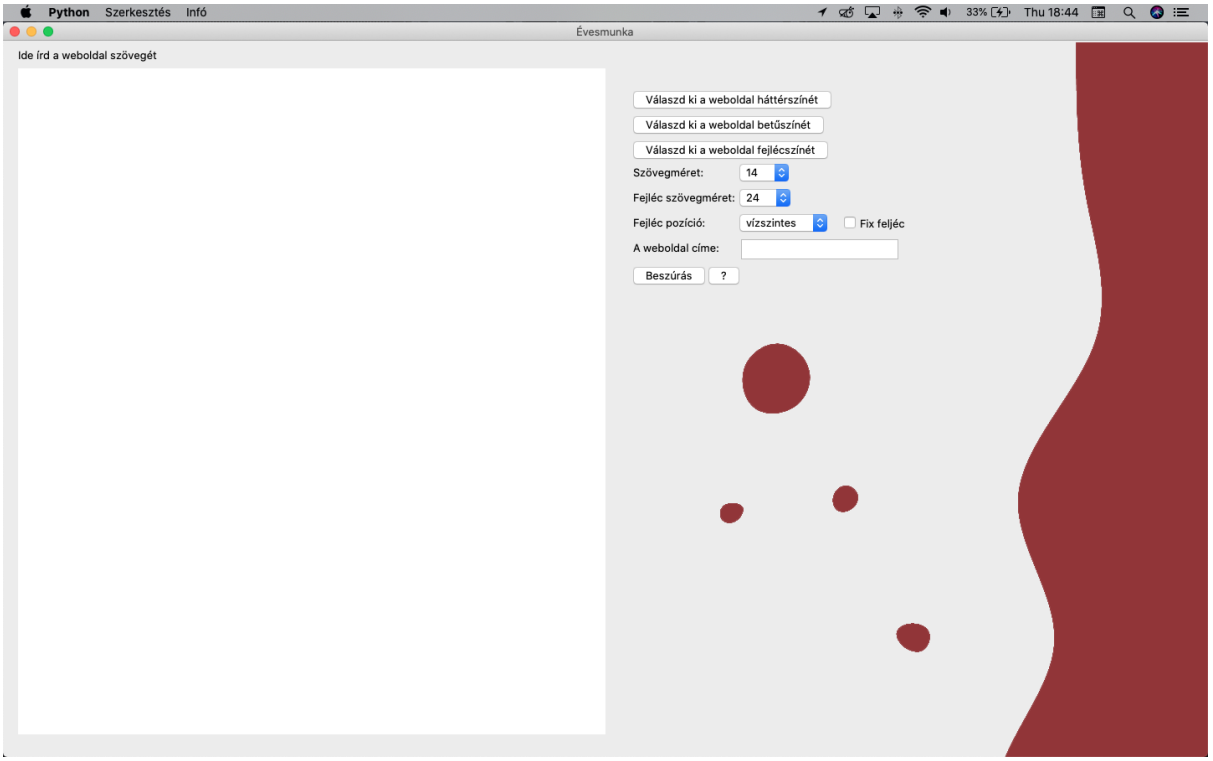
eltároljuk a felhasználó választását. Majd megnyitjuk a *szoveg.txt* fájlt, és megkeressük benne a *felhasználó válaszának* (átirandó változóban tárolt) minden előfordulását. Tehát megkérdezi a program, hogy mit emeljen ki, majd az HTML formátumban, kiemelve elmenti. Ez a program már egy kicsit hasonlít az évesmunka programomhoz.

## Az évesmunka program részletes leírása

Az évesmunkám ihlet adója egy általam készített régi programom. Ebben az alkalmazásban a számítógép -az előző példához hasonlóan- paramétereket kért. Milyen legyen a háttérszín, a betűméret, a betűk színe, stb. Egyesével kellett végig haladni, nem lehetett javítani, csak ha elölről kezdtük a létrehozást. Ennek kezelőfelülete ronda, kezelhetetlen, ha rontunk az egész folyamatot elölről kell kezdenünk. Problémái közé tartozott, hogy a szín- és betűméret adatait speciálisan kellet megadni, ami azt jelenti, hogy a színt hexadecimális kóddal -amiről már volt szó-, a betűméretet pedig pixelben kellet megadni (ilyen formátumban: 11px). Ez az átlag embernek használhatatlan. A másik hatalmas problémája a konzolban/terminálban elhelyezett kezelő felületet volt (CLI, Command Line Interface=Parancssoros felület). Itt egy példa  ami szemlélteti, hogy is néz ki a program használata:




Aki ezt átlátja, megtudja oldani, annak gratulálok, mert ez kezelhetetlen. Ezért az évesmunkámat nem ilyen kezelőfelületben helyeztem el, hanem ilyenben:




Ennél a verziónál a gombokra kattinatva, a leugró menükben kiválasztva és a szövegdobozokba irogatva adhatjuk meg, milyen is legyen a weboldalunk. Egy “beillesztés” gombra kattintva beilleszthetünk különböző elemeket a weboldalunk szakaszára. (sima szöveg, kép, gomb, és html kódot is beágyazhatunk). Ha elégedettek vagyunk a szerkesztéseinkel a “szerkesztés” menüben kiválaszthatjuk a “létrehozás” menüpontot, vagy lenyomhatjuk a Ctrl+S billentyűket a weboldalunk létrehozásához. Ezt első mentésünkkor meg is nyitja az alapértelmezett webböngészőben, de ezt manuálisan, a Ctrl+O billentyűparancsal is megtehetjük. Most már gyönyörködhetünk nagyszerű weboldalunkban.

Közös utunk itt lezárul, mely végig vezetett a blokkos programozástól kezdve, a webfejlesztésen át, egészen a python nyelven való programozásig. Köszönöm, hogy elolvastad -vagy megtekintetted- évesmunkámat.

Források:

| | |
|---|---|
| wikipédia | https://hu.wikipedia.org/wiki/Kezd%C5%91lap |
| stackoverflow | https://stackoverflow.com/ |