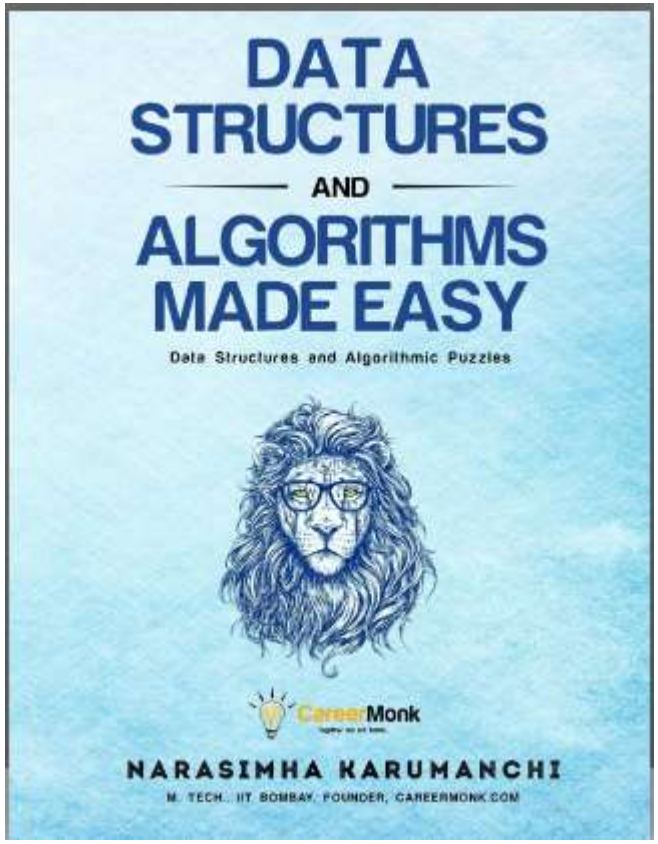
# Algoritmusok 1

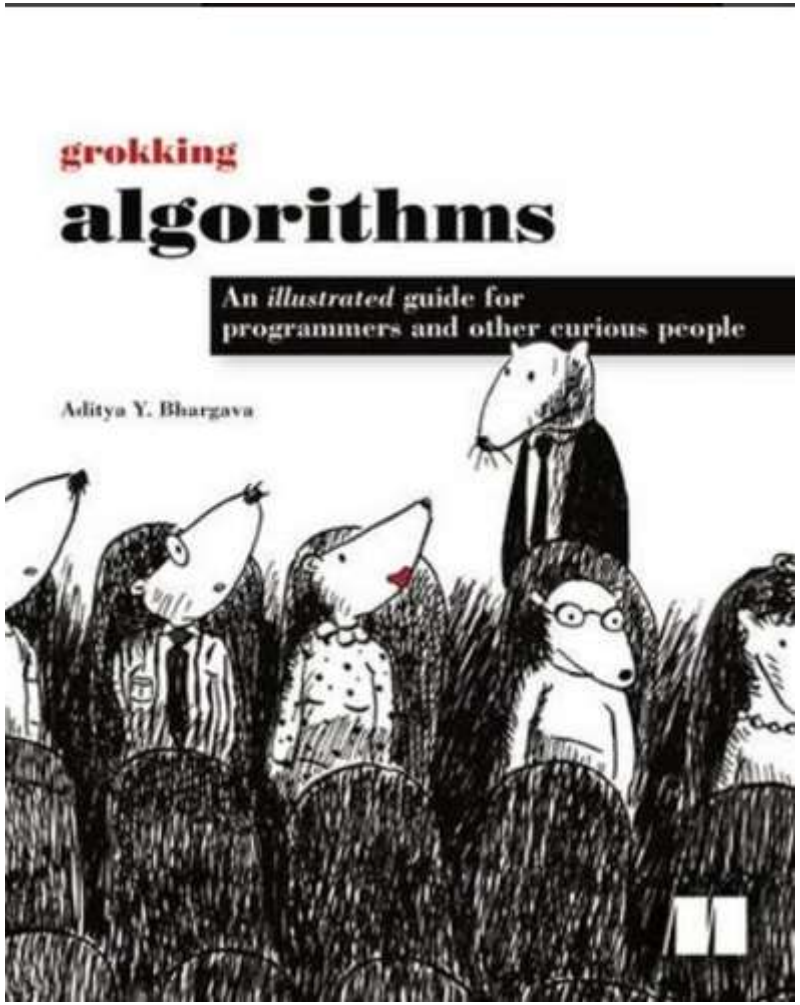
Közös kutakodás

- A szoftverfejlesztés alapjait az adatstruktúrák és az algoritmusok jelentik.
- Az adatstruktúrák és algoritmusok tanulmányozása sok kritikus gondolkodást és agyi edzést tesz szükségessé.
- Számos hatalmas IT-cég, mint például a Google, az Amazon és a Microsoft, valamint a fiatal, feltörekvő startupok, mint például a Linktree és a StackBlitz készít anyagokat az adatstruktúrákra és az algoritmusokra összpontosítva.
- Fontos, hogy tisztában legyünk a tanulmányozásukhoz szükséges legjobb forrásokkal.
- Bizonyos könyvek segítenek mélyebben megérteni az algoritmusok világát, és kiváló alapot nyújtanak a fejlesztői problémamegoldó készségek fejlesztéséhez. → lásd következő lap



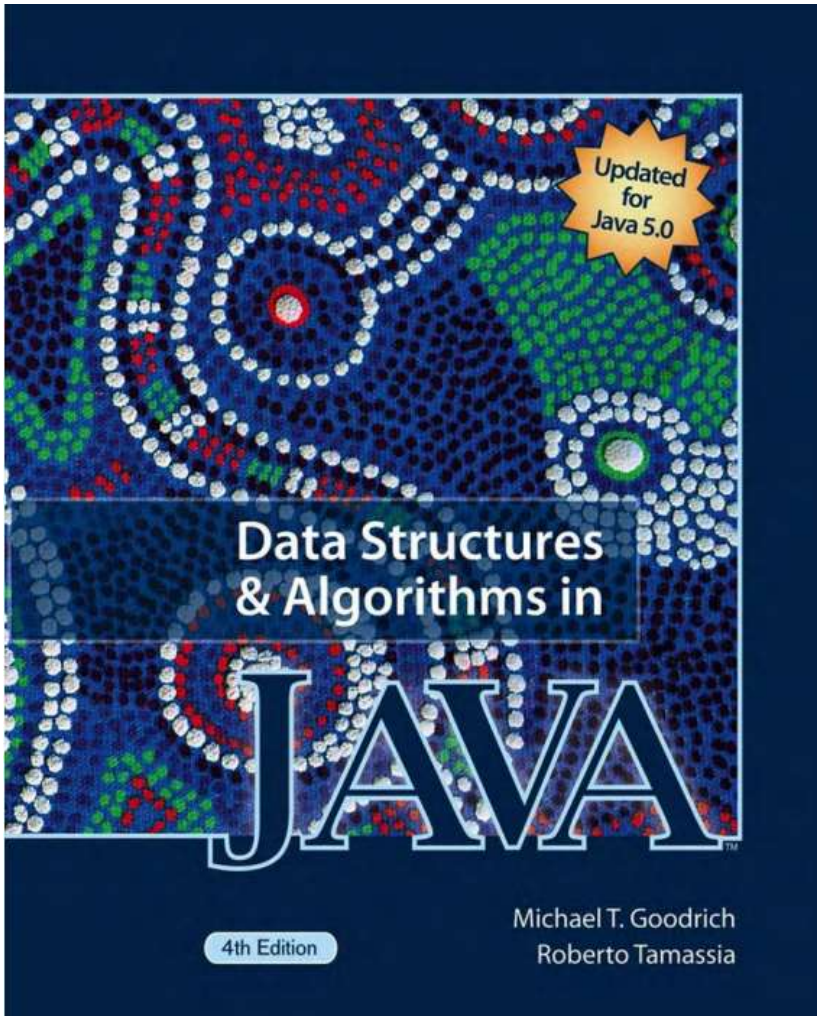

**Adatstruktúrák és algoritmusok egyszerűen**:

- Ez a könyv bemutatja az adatstruktúrák és algoritmusok alapjait.
- Megismerheted a tömböket, karakterláncokat és adatfákat, valamint az algoritmusok működését.
- A gyakorlatok C/C++ kódot használnak, így ha ismered ezt a nyelvet, fantasztikus hely lehet a kezdőknek is.

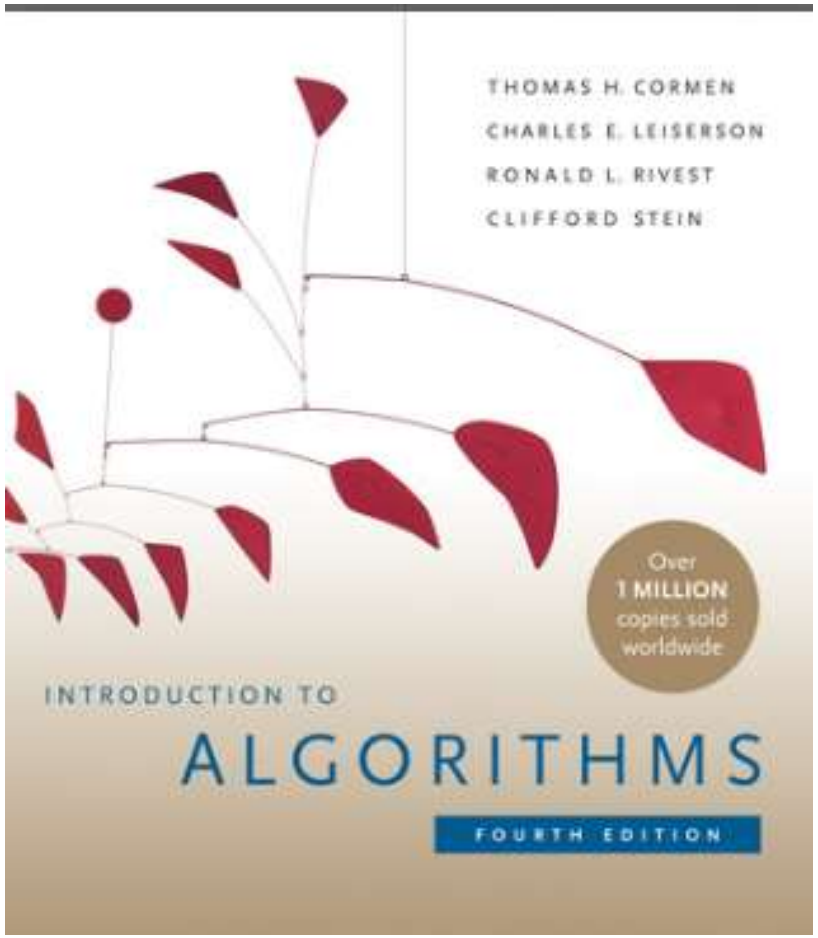



**Grokking algoritmusok**:

- Ez a könyv a valós algoritmusokkal kapcsolatos problémákat járja körül.
- A Python nyelvet használja, és több mint 400 kép segíti az adatszerkezetek és algoritmusok megértését.
- Nagyon szájbarágósan és képekkel illusztrálva mutatja be a könyv az alap adatszerkezeteket és algoritmusokat.




- **Adatstruktúrák és algoritmusok JAVA-ban**:
- Ez a könyv részletesen ismerteti az adatstruktúrákat és algoritmusokat, különös tekintettel a Java nyelvre.
- A matematikai elemzések igényesek, de érthetőek az olvasók széles köre számára.
- A könyv számos illusztrációt tartalmaz az adatstruktúrákról és azok működéséről.




**Kezdőkönyv az algoritmusokról**:

- Ez a könyv az algoritmusok alapos ismeretét célozza meg.
- Segít a hatékony programozásban és a memóriatakarékos megoldásokban.

Az algoritmusok tanulásához számos online forrás áll rendelkezésre, amelyek segíthetnek mélyebben megérteni ezeket a kulcsfontosságú témákat.

Itt van néhány ajánlás:

- **[Szegedi Tudományegyetem Online Algoritmusok](https://www.inf.u-szeged.hu/~cimreh/Online_algoritmusok.pdf)**:

Ez a dokumentum részletesen bemutatja az online algoritmusokat, a síbérléstől a gépköltséges ütemezésig. Tartalmaz példákat és matematikai elemzéseket is.

- **[Coursera](https://www.coursera.org/)**:

A Coursera online kurzusokat kínál, amelyek az algoritmusokról és a gépi tanulásról szólnak. Olyan egyetemek oktatói tartanak itt kurzusokat, mint a Stanford, a Princeton és a Michigan.

- **[GeeksforGeeks](https://www.geeksforgeeks.org/)**:

A GeeksforGeeks egy hatalmas online forrás, amely részletesen bemutatja az algoritmusokat, adatszerkezeteket és programozási technikákat. Rengeteg cikket, példát és gyakorlati feladatot találsz itt.

# **Algoritmusok elsajátítása**:

Lépések:

1. **A probléma pontos meghatározása**: Világosan fogalmazd meg a problémát.
2. **Felbontás kisebb részekre**: Bontsd le a problémát egyszerűbb részfeladatokra.
3. **Minden részfeladat megoldásának definiálása**: Határozd meg a megoldást minden részfeladatra.
4. **Megvalósítás**: Implementáld a megoldást.
5. **Hatékonyság optimalizálása**: Végül törekedj a hatékony megoldásra.

**Ne feledd, az algoritmusok gyakorlása időt és kitartást igényel, de a folyamatos gyakorlás hatékonyabbá tesz a problémamegoldásban!**

**Gyakorlás és ismétlés**:

- Oldj meg minél több algoritmusos feladatot. Használj online platformokat, mint a LeetCode vagy a HackerRank.
- Ismételd át a már megoldott feladatokat, hogy rögzítsd a tanultakat.

**Könyvek és online források**:

- Olvass könyveket az algoritmusokról.
- Használj online kurzusokat, például a Coursera-t vagy a Khan Academy-t.

**Gyakorlás papíron**:

- Írd le az algoritmusokat és a megoldásokat papíron. Ez segít jobban megérteni a lépéseket.

**Összehasonlítások és példák**:

- Helyezd át az algoritmusokat mindennapi élethelyzetekbe. Például, hogyan rendeznéd a zoknikat egy kupacból vagy keresnél egy szót a szótárban.

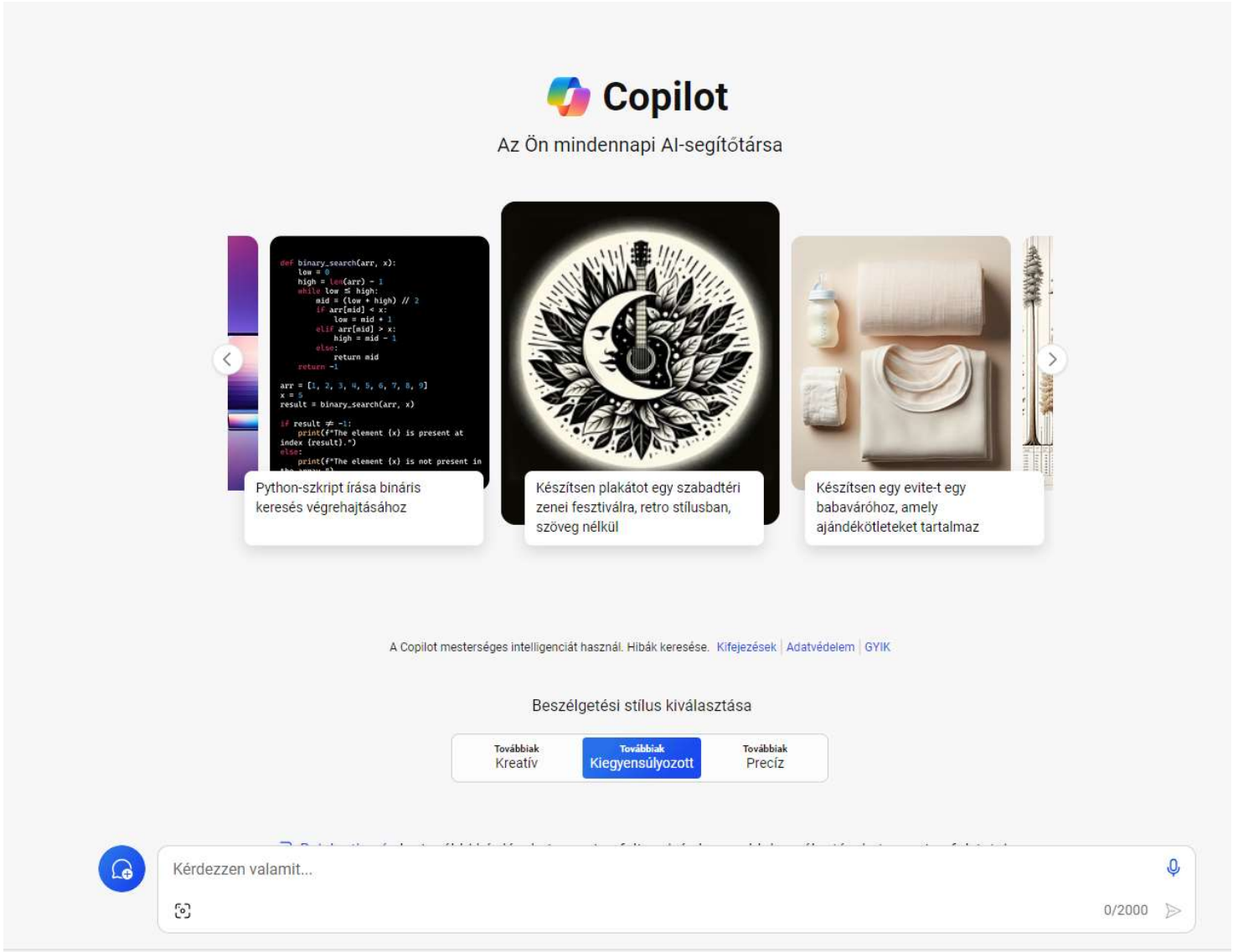
**Mesterséges intelligencia és gépi tanulás**:

- Tanulj meg algoritmusokat a gépi tanulás és a mesterséges intelligencia területén. Ez segít mélyebben megérteni az algoritmusok működését.

# Hogyan írjunk algoritmust?

A cmabrigde-i etegyemen kéüszlt eigyk tnuamálny sznreit a szvkaaon bleül nincs jlneestőgée aannk, mkénit rdeeőzndenk el a btűek: eyegüdl az a fntoos, hgoy az eslő és az uolstó betű a hlyéen lgyeen, ha a tböbrie a lgnoegyabb özeássivssazsg a jleezmlő, a sövzeg aokkr is tleejs mrtébéekn ovasalthó mraad. A jneleésg mgáayzrataa az, hgoy az erbemi agy nem eyedgi btűeket, hneam tleejs sazakvat ovals. Íme a bzoinytéík!

# COPILOT használata



Itt van egy egyszerű pszeudo kód az algoritmus számára, amely megőrzi a szavak első és utolsó karaktereit, miközben a többi karaktert összekeveri:

```
Függvény szavakOsszekeverese(szoveg):
    szavak = szoveg.szavakraBontasa()  # A szöveg szavakra bontása
    ujSzoveg = üresLista  # Üres lista létrehozása az új szövegnek
    minden szó a szavak között:
        ha a szó hossza > 3:
            elsoKarakter = szo[0]
            utolsoKarakter = szo[-1]
            kozepsoKarakterek = szo[1:-1]
            osszekevertKozepso = kozepsoKarakterek.osszekeverese()
            ujSzo = elsoKarakter + osszekevertKozepso + utolsoKarakter
        különben:
            ujSzo = szo  # Ha a szó hossza <= 3, hagyjuk változatlanul
        ujSzoveg.hozzaad(ujSzo)  # Az új szót hozzáadjuk az új szöveghez
    visszatérés ujSzoveg  # Visszaadjuk az új szöveget

# Példa használat:
bemenetiSzoveg = "Ez egy példa mondat."
ujSzoveg = szavakOsszekeverese(bemenetiSzoveg)
kiir(ujSzoveg)
```

Ez a pszeudo kód egy egyszerű megközelítést mutat be arra, hogyan lehet megőrizni a szavak olvashatóságát, miközben az első és utolsó karaktereket változatlanul hagyjuk. Természetesen a pontos implementáció a választott programozási nyelvtől függ, de ez a kód segíthet elindulni!

**Copilot**

Persze, itt van egy egyszerű Python program, amely megvalósítja ezt a feladatot:

**Python**

```python
def osszekever_szavak(text):
    # Szavak szétválasztása a szóközök mentén
    szavak = text.split()

    # Minden szó első és utolsó karakterének megtartása, többi karakter összekeverése
    uj_szavak = []
    for szo in szavak:
        if len(szo) > 2:
            elso_karakter = szo[0]
            utolso_karakter = szo[-1]
            koztes_karakterek = szo[1:-1]
            osszekevert_koztes_karakterek = ''.join(random.sample(koztes_karakterek, len(koztes_karakterek)))
            uj_szo = elso_karakter + osszekevert_koztes_karakterek + utolso_karakter
            uj_szavak.append(uj_szo)
        else:
            uj_szavak.append(szo)

    # Összekevert szavak visszaillesztése a szövegbe
    uj_text = ' '.join(uj_szavak)
    return uj_text

# Példa használat:
bemeneti_szoveg = "Ez egy példa mondat, amit összekeverünk."
osszekevert_szoveg = osszekever_szavak(bemeneti_szoveg)
print(osszekevert_szoveg)
```



Ez a program a bemeneti szöveget szavakra bontja, majd minden szó első és utolsó karakterét megtartja, a többi karaktert pedig összekeveri. A példa bemenet “Ez egy példa mondat, amit összekeverünk.” esetén az alábbi kimenetet adhatja:

# prog.hu

- A belső betűk keverését elég kezdetleges - egyetlen sorban leírható - algoritmus végzi, mely nem tudná előállítani az összes lehetséges sorrendet.
- Ideális esetben az összes permutációnak egyenlő eséllyel kellene kijönnie.
- Csak egyszer kell végigmenni a szövegen. Nem kell az egész fájlt beolvasni, hanem egyszerre csak egy szó tárolandó futás közben.
- Rekurzív függvényhívás teljesen felesleges. (Amikor szükséges volna, általában akkor is ajánlatos iterációval csinálni.)
- A program olvashatóságát makrók és kommentek alkalmazásával könnyíthetjük meg.

```
#include <stdio.h>
#define file_name       "blabla.txt"
#define max_word_length 30
#define start           srand(time(NULL));                  \
                        FILE *tf;                           \
                        if(!(tf=fopen(file_name,"rt")))     \
                          {fprintf(stderr,"Sorry\n");exit(-1);}
#define stop            printf("\n\n"); system("pause"); return 0;
#define csere(x,y)      {int d=x ; x=y; y=d;}
#define shufffle        for(j=2; j<w; j++) if(rand()%2) csere(word[1],word[j])
#define print_word      for(j=0; j<=w; j++) printf("%c",word[j]);
char plus_letters[] ="áéíóöőúüűÁÉÍÓÖŐÚÜŰ", // tetszés szerint kibővíthetjük az ABC-t
    word[max_word_length],                  // az aktuális szó tárolója
    c;
int j,w;
////////////////////////////////////////////////////////////////////////////////
 int is_letter(char x) {                    // egy karakter vizsgálata
  if (('a'<=x)&&(x<='z')) return 1;        // angol kisbetű
  if (('A'<=x)&&(x<='Z')) return 1;        // angol nagybetű

  int j=0;
  while(plus_letters[j] != '\0') {          // további elfogadható karakter
    if(x == plus_letters[j]) return 1;
    else ++j;
  }
  return 0;                                 // nem betű
}
```

```
int main(){

  start

  while(!feof(tf)) {                    // fájl kiolvasása
    w=-1;                               // az aktuális szó utolsó karakterének indexe
    do {
      fscanf(tf,"%c",&c);               // karakter beolvasása
      if(is_letter(c))  word[++w]=c;    // ha az ABC-nk eleme, akkor a szóhoz adjuk
      else break;                       // nincs az ABC-ben, vége a szónak
    }
    while(1<2);

    if(2<w) shufffle                    // a belső betük egyszerű permutálása
    if(0<=w) print_word                 // szó kiírása
    printf("%c",c);                     // a szó utáni karaktert csak most írja ki
  }

  stop
}
```

# Szendi Bianka:

In fact, there never was a Cambridge researcher (the earliest form of the meme actually circulated without that particular addition), but there is some science behind why we can read that particular jumbled text.

The phenomenon has been given the slightly tongue-in-cheek name "Typoglycaemia," and it works because our brains don't just rely on what they see - they also rely on what we expect to see.

In 2011, researchers from the University of Glasgow, conducting unrelated research, found that when something is obscured from or unclear to the eye, human minds can predict what they think they're going to see and fill in the blanks.

https://www.sciencealert.com/word-jumble-meme-first-last-letters-cambridge-typoglycaemia

```python
from random import sample
def typoglycemia(ws):
    for w in ws.split():
        if len(w) <= 3 or all(x==w[1]
                for x in w[2:-1]):
            print(w,end=' ')
            continue
        w2 = w
        while w[0] + w[1:-1] + w[-1] == w2:
            w = w[0] + ''.join(sample(w[1:-1],len(w[1:-1]))) + w[-1]
        print(w,end=' ')

#typoglycemia(input())

txt = "This is a very long text to test the code and it does not makes sence so today is a day weird and this is a weird text that you cannot understand and so nothing metters"

typoglycemia(txt)

```

https://www.sololearn.com/en/compiler-playground/c0MGzd94tlid/?ref=app

# Algoritmus leírása:

1. **Split the text into words**: Start by splitting the input text into individual words. This could be achieved by splitting the text based on spaces or punctuation marks.
2. **Process each word**:
   - For each word, check its length.
   - If the word has less than 3 characters, leave it unchanged.
   - If the word has 3 or more characters, shuffle the characters between the first two and last two characters. You can achieve this by:
     - Extracting the first and last characters of the word.
     - Randomly shuffling the characters in between (excluding the first and last characters).
     - Combining the first, shuffled middle, and last characters back together.
3. **Join the words back into text**: After processing each word, join them back together to form the final scrambled text.

Pszeudo kódja:

```
function scrambleText(text):
    words = splitTextIntoWords(text)
    scrambledWords = []
    for each word in words:
        if length(word) < 3:
            scrambledWords.append(word)
        else:
            scrambledWord = scrambleWord(word)
            scrambledWords.append(scrambledWord)
    scrambledText = joinWordsIntoText(scrambledWords)
    return scrambledText

function scrambleWord(word):
    firstChar = word[0]
    lastChar = word[length(word) - 1]
    middleChars = shuffleChars(word[1:length(word) - 1])
    scrambledWord = firstChar + middleChars + lastChar
    return scrambledWord

function shuffleChars(chars):
    shuffledChars = randomly shuffle characters in chars
    return shuffledChars
```

Mire lehet használni?

processing each letter individually. While typoglycemia itself isn't typically used for any specific purpose, it's interesting to study in fields like cognitive psychology and linguistics to better understand how the brain processes language. Additionally, it can serve as a fun exercise or game, challenging people to unscramble words and still understand them despite the jumbled letters.

# Princzes Barnabás:

**Typoglycemia** – eredeti diasor:

https://docs.google.com/presentation/d/1M3WYUdgqeuzW6PlAENWZTEkuXIiQ1vPFV6h20kXop9o/edit?usp=sharing

**Definíció:** Az olvasási folyamatnak arról a sajátosságáról, hogy a leírt szöveg megértéséhez nem szükséges, hogy a szavak betűi helyes sorrendben legyenek. Elegendő, ha a szó első és utolsó betűje a helyén marad, a többi lehet bármilyen sorrendben, mégis megértjük a írás jelentését.

**Lingvisztika:**

- Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy

Cambridge-i kutató visszavezeti a mémet

- CaSe MiXiNg

Sokkal zavaróbb

- Olvass úgy, mint egy diszlexiás

**Algoritmusok:**

- Deciphering Typoglycemia articles using Recurrent Neural Networks - Sachin Srivastava

MI megoldás, de a kivonatnál mélyebbre nem jutottam.

- "Typocaptcha" - an alternative to CAPTCHA? - User Experience Stack Exchange

Nem elég bonyolult captcha-nak.

**Python:**

https://github.com/h1g0/typoglycemia.py/blob/master/typoglycemia.py

```
print((s:=' ').join(map(lambda w:(w[0]+''.join(__import__('random').sample(w[1:-1],len(w)-2))+w[-1:])if len(w)>1 else w,'ez az a mondat amire a '.split(s)))+'.')
```

Továbbiak:

https://www.iaria.org/conferences2023/filesDataSys23/PetreDini_Tutorial_ACommonSenseTake.pdf

“A Common-sense Take on Awareness about AI-induced Human Society” Prof. Dr. Petre Dini - International Academy, Research, and Industry Association

AI, Generative-AI, LLM, ML, Deep Learning, Graphs, Ontologies, Taxonomies, etc. are only the pick of iceberg of human thinking process, perception & storage of data and building knowledge upon

Mesterséges Intelligencia, nagy nyelvi modellek, gráfok, ontológiák, taxonómiák... csupán a jéghely csúcsa az emberi gondolkodás, észlelés, adattárolás, tudásszerzés szintjén

**Ultravalók:**

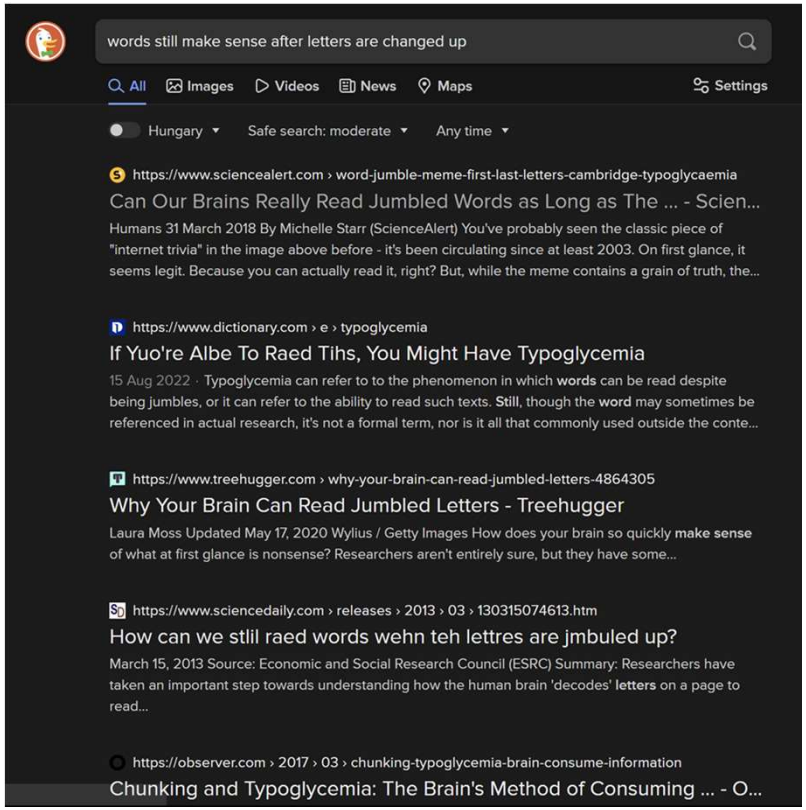
- A tudásszerzés és a tanulás módjai emberek között is különbözőek  generációról generációra.
- Az a fajta tanulás amit MI-vel csinálunk csak egy ismeretlen fokú közelítés ahhoz ahogy az emberi agy valójában működik.

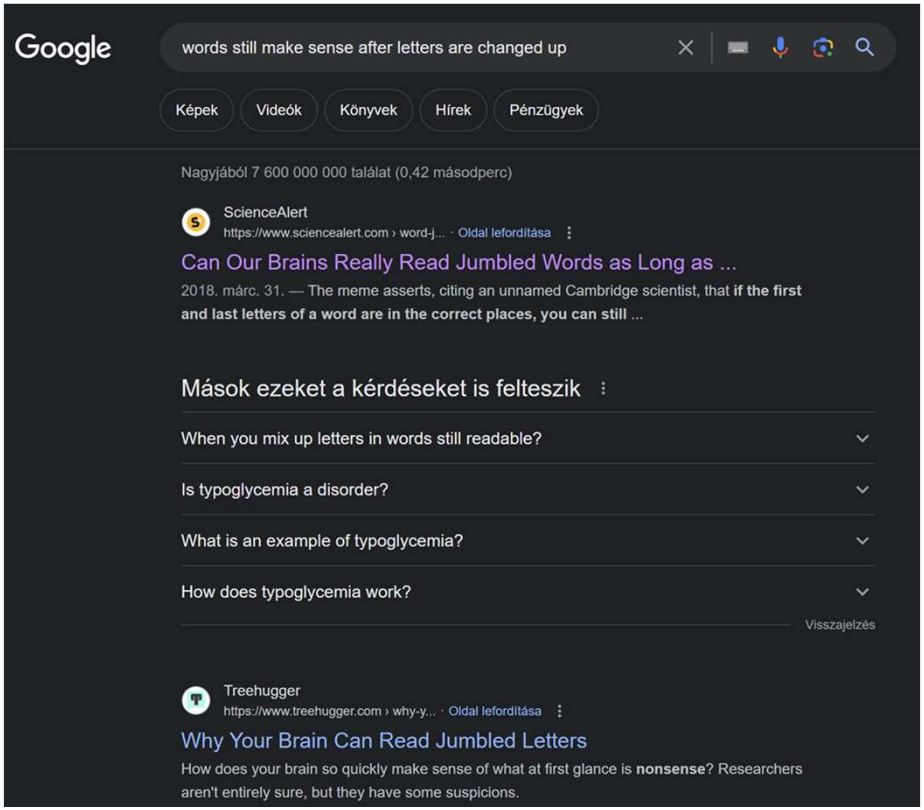
(typoglycemia <-> szerkesztőkben autocorrect)

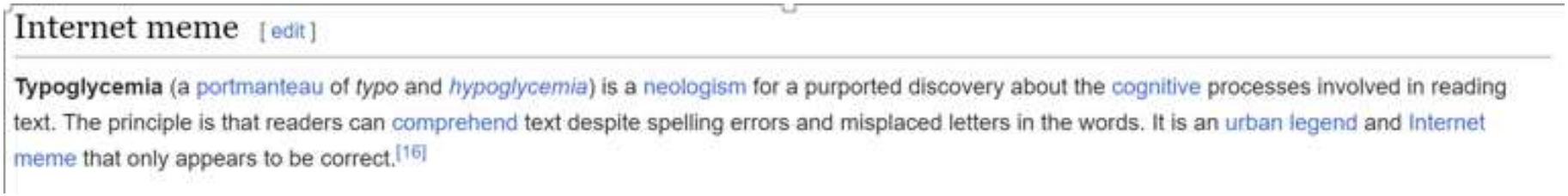
- A digitális készségvesztés a MI alapú megközelítések / eszközök legnagyobb veszélye.
- Veszély: A tanulási görbénk sokkal lassabb mint a minket körülvevő technológia evolúciója. A halogatás nem választható opció, a változásokat folyamatosan figyelnünk kell.

# Németh Ábel:

## Part 1: Az egész csak egy meme



# Part 2: Az email csak félig igaz

- Olvashatósági szabályok:
  - Egymás melletti betűk felcserélése
  - Rövid szavak
  - Funkcionális szavak (a, az, egy)
  - Nem hozhatunk létre más szavakat
  - Eredeti hangzás megtartása
  - Szöveg előreláthatósága
- Ha csak az eredeti szabályt használjuk, vagyis az első és utolsó betű legyen helyes és randomizáljuk a többit, teljesen olvashatatlan szöveget kaphatunk

> *Soaesn of mtiss and mloelw ftisnflurues,*
> *Csloe boosm-feinrd of the mrtuniag sun;*
> *Cnponsiirg wtih him how to laod and besls*
> *Wtih friut the viens taht runod the tahtch-eevs run*

That's the first four lines of the poem "To Autumn" by John Keats.

> *Season of mists and mellow fruitfulness,*
> *Close bosom-friend of the maturing sun;*
> *Conspiring with him how to load and bless*
> *With fruit the vines that round the thatch-eves run*

# Part 3: Kapcsolódó tanulmányok

2011 Glasgow: Az agy nem hagyatkozik arra amit lát, hanem komplex sejtésekkel tölti ki a hiányzó részeket.

Dr Fraser Smith, from the same Institute, said: “[...]Effectively, our brains construct an incredibly complex jigsaw puzzle using any pieces it can get access to. These are provided by the context in which we see them, our memories and our other senses.”

Dr Muckli added: “Sometimes the brain’s guess can be so convincing that we see visual illusions; in our example there was no visual illusion seen – the white space was not filled-in by an actual illusion. Nevertheless we found a way to reveal the brains guess of what lies behind the object.”

Az agy minimalizálja a meglepetést.

Ennek köze lehet ahhoz hogy az idősek mentális képességei romlanak mikor új helyre költöznek.

Graham Rawlinson PhD-je bemutatta a “Typoglycaemia” hatását

Ez a meme lehetséges eredete

"word recognition skills develop which are not only not taught but which develop despite sometimes fairly specific teaching in alternative skills"

A PhD thézis a szófelismerés és tanulási módszerek közötti eltérésre hívja fel a figyelmet

Később oktatási pszichológusként dolgozott

# Sources

Science Alert *Can Our Brains Really Read Jumbled Words as Long as The First And Last Letters Are Correct?*

Wikipedia *Transposed letter effect*

Glasgow University News WHAT OUR EYES CAN'T SEE, THE BRAIN FILLS IN

Graham Rawlinson level a New Scientist-hez (1999)

The Significance of Letter Position in Word Recognition PhD Thesis, 1976, Nottingham University, by Graham Rawlinson

# Feladatok felosztása

1. Csoport: rekurzív függvények

Miért használunk rekurzív függvényeket?

Mikor előnyös rekurzív függvényt konstruálni?

Milyen esetekben használhatjuk az Ackermann függvényt?

## Ackermann függvény

*avagy nem minden olyan egyszerű, mint amilyennek látszik*

Definíció: Legyen $A(x, y)$ a következő rekurzív módon definiált függvény:

$$A(0, y) := y + 1 \qquad y \geq 0$$
$$A(x, 0) := A(x - 1, 1) \qquad x \geq 1$$
$$A(x, y) := A(x - 1, A(x, y - 1)) \qquad x, y \geq 1$$

Határozzuk meg pl. $A(4, 2)$ és $A(5, 1)$ értékét!

Az Ackermann-függvény nagyon gyorsan növekszik, így már kis bemenetek esetén is hatalmas értékeket vesz fel.

Például *A(4, 2)* értéke körülbelül **2×10^19728**, és még *A(4, 3)* is olyan hatalmas, hogy a fizikai univerzum méretében nem lenne elegendő hely a tízes számrendszerbeli kifejtéséhez, tízes számrendszerben **19729** számjegyre van szükség a felírásához.

$A(0,y) := y+1 \quad y \geq 0$

$A(x,0) := A(x-1,1) \quad x \geq 1$

$A(x,y) := A(x-1, A(x,y-1))$

$x, y \geq 1$

$A(3,1) \Rightarrow 13$

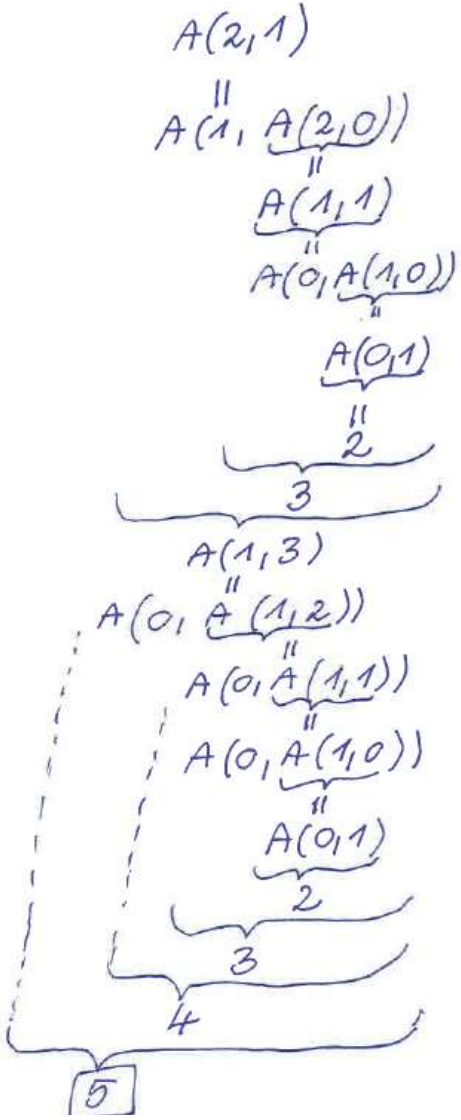
$A(3,2) \Rightarrow 29$

$A(3,3) \Rightarrow 61$

$A(3,8) \Rightarrow 2045$

$A(4,1) \Rightarrow 65533$

$A(4,2) \Rightarrow$ már nem megy!




## 2. csoport: Miért használunk rekurzív függvényeket?

Mikor előnyös rekurzív függvényt konstruálni?

Milyen esetekben használhatjuk a postfix konverziót?

### Postfix konverzió

Minden **kifejezés** vagy egy ***tag***, vagy ***tagok additív műveleti jelekkel elválasztott sorozata***.

Minden kifejezés egyértelműen felbontható $K = t_1 \oplus_1 t_2 \ldots \oplus_m t_{m+1}$ alakban, ahol $\oplus_i \in \{+, -\}$ és $t_i$ tag.

Ekkor $\Phi(K) = \Phi(t_1)\Phi(t_2) \oplus_1 \ldots \Phi(t_{m+1})\oplus_m$

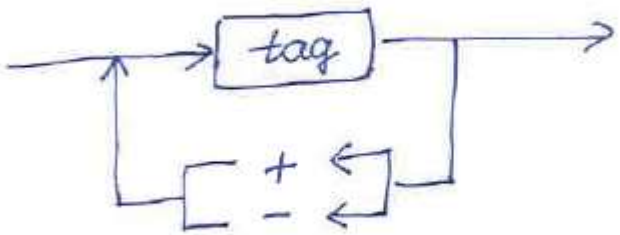
Minden **tag** vagy egy ***tényező***, vagy ***tényezők multiplikatív műveleti jellel elválasztott sorozata***.

Minden tag egyértelműen felbontható $T = t_1 \otimes_1 t_2 \ldots \otimes_m t_{m+1}$ alakban, ahol $\otimes_i \in \{*, /\}$ és $t_i$ tényező.

Ekkor $\Phi(T) = \Phi(t_1)\Phi(t_2) \otimes_1 \ldots \Phi(t_{m+1})\otimes_m$

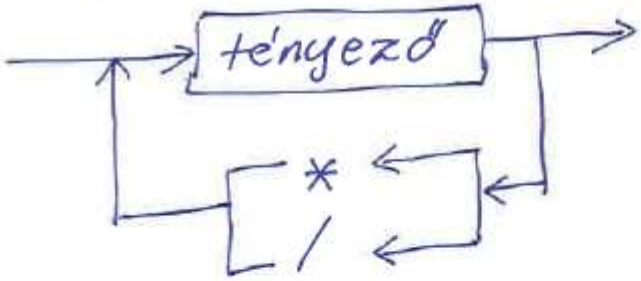
Minden **tényező** vagy ***zárójelbe tett kifejezés***; $(K)$ és $\Phi((K)) = \Phi(K)$, vagy ***azonosító***; $a$ és $\Phi(a) = a$

Kifejezés:

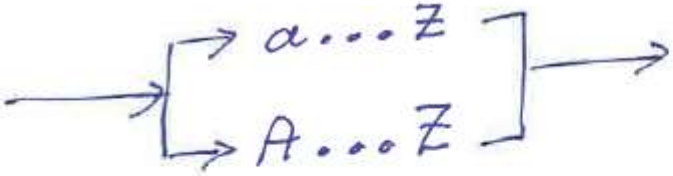



Tényező:


Tag:




Azonosító:




Példák: $(a+b)/(c-d) = ab+cd-/$

$((a+b)*c+d)/(e+f+g) = ab+c*d+ ef+g+/$

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Postfix
{
   public class Postfix
   {
      private static bool Jo = true;
      private static String S;
      private static int i = 0;
      private static char Jel;
      private static String Postform;

      private static void KovJel()
      {
         Jel = S[i];
         i++;
      }
```

```
      public static void Kifejezes()
      {
         char M;
         Tag();
         while (Jo && (Jel == '+' || Jel == '-'))
         {
            M = Jel;
            KovJel();
            Tag();
            Postform = Postform + M;
         }
      }
      private static void Tag()
      {
         char M;
         Tenyezo();
         while (Jo && (Jel == '*' || Jel == '/'))
         {
            M = Jel;
            KovJel();
            Tenyezo();
            Postform = Postform + M;
         }
      }
```

```
private static void Tenyezo()
    {
      if ('a' <= Jel && Jel <= 'z')
      {
        Postform = Postform + Jel;
        KovJel();
      }
      else if (Jel == '(')
      {
        KovJel();
        Kifejezes();
        if (Jo && Jel == ')')
          KovJel();
        else
          Jo = false;
      }
      else
        Jo = false;
    }
```

```
public static String postfix(String K)
    {
      S = K + '.';
      KovJel();
      Postform = "";
      Kifejezes();
      return Postform;
    }

    static void Main(string[] args)
    {
      System.Console.WriteLine("Az átalakítandó
műveletsorozat: a+b*(a-b)/(x+y)");
      System.Console.Write("Átalakítás után a postfix
alak: ");
      System.Console.WriteLine(postfix("(a+b)*(a-b)"));
    }
  }
}
```