ChatGPT-kísérletek - programozás

(Experiments with chatGPT on the field of creating source codes)

Rikk János, Pitlik László, KJE

Kivonat: Az alábbi kísérlet kapcsán a chatGPT, mint öntörvényű programozó személyiségjegyeinek, gondolkodásmódjának letapogatása a cél úgy matematikai/informatikai, mint logikai/ember-gép-együttműködési szempontból.

Kulcsszavak: phyton, ciklus, kód-magyarázóképesség

Abstract:

Keywords: phyton, loop, competence for code-interpretability

# Bevezetés

Az alábbi esettanulmányokban pirossal láthatók a Megrendelő (emberi szakértő) tervezési, értelmezési gondolatai. Fekete háttérrel a megrendelő chatGPT számára adott „utasításai”. S fehér háttérrel fekete betűkkel írva keretbe foglaltan a chatGPT programkódjai és saját kódértelmezési gondolatai.

A karakterre pontosan idézett (2023.03.17-i chatGPT állapotot tükröző) beszélgetés-jegyzőkönyvet lábjegyzetek egészítik ki, melyek egyszerre reflektálnak a chatGPT válaszokban tetten érhető egyes jelenségekre és/vagy a Megrendelő gondolataira.

Az alábbi kísérletek (esetek) kerülnek nagyító alá:

* Az első esetben magyar nyelven kell lottószám-húzásra képes phyton-kódot írnia a chatGPT-nek a megrendelő kisebb-nagyobb finomhangolási elvárásai mellett…
* A második esetben angol nyelvű kommunikáció folyik, ahol a cél a magyar nyelvű kísérlet kapcsán utoljára kialakult megoldásból fakadó jelenségek további elemzése.
* A harmadik esetben a megrendelő kódjait kell értelmeznie a chatGPTnek.
* Míg a 4. esetben a chatGPT újra lottószám-sorsoló kódot kell, hogy írjon, immár ismerve a megrendelő bizonyos preferenciáit, szándékait…
* Az 5. kísérlet egy tic-tac-toe játék megírása a megrendelő néhány reakcióját is lereagálandó.
* S végül a 6. kísérlet nyelvészeti/etimológiai természetű…

# eset

BEKÓSTOLTAM A chatGPT-t

(ingyenes regisztráció után elérhető verzió, 2023. 03. 17-i állapot)

spoiler:

1. nem tévedhetetlen, de nem is ezt várja tőle az ember
2. hatalmas tévedés nem kihasználni a benne rejlő lehetőségeket
3. oktatásban nem tiltani kell, hanem kötelezővé tenni (a Gundelből sem azt rúgják ki, aki ügyesen bánik az evőeszközzel, ami bárki számára hozzáférhető, hanem aki csupasz kézzel maszatol)

írj egy lottószám sorsoló python programot[[1]](#footnote-1)

Természetesen, íme egy egyszerű lottószám sorsoló program Python nyelven:

import random

def main():

 numbers = []

 while len(numbers) < 6:

 number = random.randint(1, 45)

 if number not in numbers:

 numbers.append(number)

 numbers.sort()

 print("Az Ön lottószámai:")

 for number in numbers:

 print(number)

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

 main()

Ez a program véletlenszerűen választ 6 különböző számot a 1 és 45 közötti tartományból, majd kiírja ezeket a számokat a képernyőre. A **random.randint()** függvény használatával a program biztosítja, hogy minden választott szám különböző lesz egymástól. A **numbers.sort()** metódus használatával az eredményeket növekvő sorrendbe rendezzük.

A **if \_\_name\_\_ == '\_\_main\_\_':** szerkezet megakadályozza, hogy a programot importálás után azonnal futtatja. Ha futtatni akarjuk a programot, egyszerűen csak futtassuk a **main()** függvényt.

helyesen működő kódot kaptunk

a magyarázatban van tévedés: „. A random.randint() függvény használatával a program biztosítja, hogy minden választott szám különböző lesz egymástól.”

elméletileg semmi nem garantálja, hogy nem tart „sokáig” a számok különbözőségének biztosítása[[2]](#footnote-2)

átírnád úgy hogy a “while” ciklus helyett “for” ciklust használsz?

Természetesen, itt van az átalakított kód:

import random

def main():

 numbers = []

 for i in range(6):

 while True:

 number = random.randint(1, 45)

 if number not in numbers:

 break

 numbers.append(number)

 numbers.sort()

 print("Az Ön lottószámai:")

 for number in numbers:

 print(number)

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

 main()

Ebben a változatban az eredeti **while** ciklust egy **for** ciklusra cseréltem. Az új **for** ciklus 6-szor fut le, és minden alkalommal a belsejében egy **while** ciklus található. Az új **while** ciklus addig fut, amíg egy olyan számot nem talál, amely még nem szerepel a **numbers** listában. Ha talált egy megfelelő számot, akkor hozzáadja a **numbers** listához.

Ez a módosítás megvalósítja az eredeti célját, hogy a kód ne véletlenszerűen válassza ki ugyanazt a számot kétszer. Az új kód még mindig a véletlen számok kiválasztásán alapul, de a **for** és a **while** ciklusok kombinációját használja az eredmények generálásához.

na ez „dúúúlva”… nem lecserélte a while-t hanem beletette egy for-ba… ülj le, egyes 😉[[3]](#footnote-3)

azt szeretném hogy egyáltalán ne legyen benne “while”

Rendben, itt van az átalakított kód, amely nem tartalmaz **while** ciklust:

import random

def main():

 numbers = random.sample(range(1, 46), 6)

 numbers.sort()

 print("Az Ön lottószámai:")

 for number in numbers:

 print(number)

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

 main()

Az új kód egyetlen **for** vagy **while** ciklust sem használ a lottószámok generálásához. Ehelyett a Python **random** moduljának **sample()** metódusát használjuk, amely véletlenszerű mintavételezést végez egy adott sorozatból, ahol a mintának megadható a mérete. Itt az 1 és 45 közötti számok listájából véletlenszerűen választunk ki 6 különböző számot a **random.sample(range(1, 46), 6)** sorral.

Az eredménylista növekvő sorrendbe rendezése után az eredményeket kiírjuk a képernyőre. Ez a kód egy egyszerűbb és rövidebb megoldás az eredeti problémára, és ugyanazt a véletlenszerűséget biztosítja, mint az eredeti kód.

ügyesen megkerüli a problémát[[4]](#footnote-4)

választ egy olyan szerkezetet, amin „ránézésre” nem látszik, hogy milyen algoritmus alapján valósítja meg a feladatot

megjegyzés: eddig mindhárom megoldás helyesen oldotta meg az alapfeladatot

# Eset

MÁSODIK MENET: próbáljuk meg kicsikarni, mi rejtőzik a használt metódus mélyén?

can you give me the code of “sample” method in python?

Certainly! Here is an example of how to use the **random.sample()** method in Python:

import random

# Sample 3 random items from a list

my\_list = [1, 2, 3, 4, 5]

sampled\_items = random.sample(my\_list, 3)

print(sampled\_items)

In this example, we import the **random** module and create a list of integers called **my\_list**. We then use the **random.sample()** method to sample 3 random items from **my\_list** and store the result in the **sampled\_items** variable. Finally, we print the contents of **sampled\_items** to the console.

The **random.sample()** method can also be used with other types of sequences, such as tuples and strings, and can be used to generate a random sample without replacement. The method takes two arguments: the sequence to sample from and the number of items to sample.

szerintem itt egy kicsit félreértett

illetve nem elég precízen/egyértelműen fogalmaztam meg a kérdést (?)[[5]](#footnote-5)

I want to see the code behind the method

The **random.sample()** method is actually implemented in C as part of the Python Standard Library, so the code itself is not written in Python. However, you can view the source code for the **random** module in Python by navigating to the Python installation directory on your computer and finding the **random.py** file.

Here is an excerpt of the relevant code for the **random.sample()** method from the **random.py** file:

def sample(self, population, k):

 """Chooses k unique random elements from a population sequence or set.

 Returns a new list containing elements from the population while

 leaving the original population unchanged. The resulting list is

 in selection order so that all sub-slices will also be valid random

 samples. This allows raffle winners (the sample) to be partitioned

 into grand prize and second place winners (sub-slices).

 Members of the population need not be hashable or unique. If the

 population contains repeats, then each occurrence is a possible

 selection in the sample.

 To choose a sample in a range of integers, use range as an argument.

 This is especially fast and space efficient for sampling from a

 large population: sample(range(10000000), 60)

 Sampling without replacement entails tracking either potential

 selections (the pool) in a list or set or previous selections in a

 set.

 When the number of items to choose is small compared to the

 population size, then tracking selections is efficient, requiring

 only a small set and an occasional reselection. For a larger

 number of choices, the pool approach is preferred since the list

 takes less space than the set and it doesn't suffer from frequent

 reselections.

 """

 n = len(population)

 if not 0 <= k <= n:

 raise ValueError("Sample larger than population or is negative")

 result = [None] \* k

 setsize = 21 # size of a small set minus size of an empty list

 if k > 5:

 setsize += 4 \*\* ceil(log(k \* 3, 4)) # table size for big sets

 if n <= setsize or hasattr(population, "keys"):

 # An n-length list is smaller than a k-length set, or this is a

 # mapping type so the other algorithm wouldn't work.

 pool = list(population)

 for i in range(k): # invariant: non-selected at [0,n-i)

 j = randbelow(n-i)

 result[i] = pool[j]

 pool[j] = pool[n-i-1] # move non-selected item into vacancy

 else:

 selected = set()

 selected\_add = selected.add

 for i in range(k):

 j = randbelow(n)

 while j in selected:

 j = randbelow(n)

 selected\_add(j)

 result[i] = population[j]

 return result

This code is responsible for selecting a random sample of size **k** from a given sequence **population**. The code includes comments explaining how the method works, as well as different approaches that can be taken depending on the size of the population and the number of items to choose.

Na, csak sikerült kiénekelni, amit szerettem volna

# Eset

HARMADIK MENET: nézzük, megérti-e az én változatomat?

mit csinál az alábbi python kód?

Sajnos nem tudom megmondani, hogy pontosan mire szolgál a Python kód, mert nincs megadva. Kérem, hogy írja be a kódot és akkor tudok válaszolni a kérdésére.

from random import randint

szamok = list(range(1,91,1))

nyer = []

nyer1 = []

#nyerőszámok sorsolása 1.verzió

while len(nyer)<5:

 x = randint(1,90)

 if x not in nyer:

 nyer.append(x)

#nyerőszámok sorsolása 2.verzió

for i in range(5):

 x = randint(0,(89-i))

 nyer1.append(szamok[x])

 szamok[x] = szamok[(89-i)]

print(nyer)

print(nyer1)

Ez a Python kód két különböző módszerrel sorsol 5 nyerőszámot a 90 számból (1-90-ig tartomány).

Az első módszer egy "while" ciklust használ, amely addig fut, amíg az 5 nyerőszámot tartalmazó lista hossza nem lesz 5. A "randint" függvény segítségével véletlenszerű számokat sorsolunk a 1-90 tartományból, majd ha az adott szám még nem szerepel a nyerőszámok listájában, akkor hozzáadjuk azt a listához.

A második módszer egy "for" ciklust használ, amely öt alkalommal fut, hogy összesen 5 nyerőszámot sorsoljon ki. Az algoritmus minden lépésben egy véletlenszerű számot húz a megengedett tartományból, majd eltávolítja azt a tartományból, és hozzáadja a nyerőszámok listájához. Ez biztosítja, hogy az öt kiválasztott szám egyediek legyenek.

A nyerőszámokat mindkét esetben a "nyer" és a "nyer1" változóban tároljuk, majd kiírjuk őket a képernyőre.

melyik módszer a jobb?

Mindkét módszer helyes és megfelelően sorsolja ki a nyerőszámokat. Azonban van néhány különbség a két módszer között, amelyek befolyásolhatják a választást attól függően, hogy mi a cél.

Az első módszer "while" ciklusa egyszerűbb és rövidebb, mint a második módszer "for" ciklusa. Azonban, ha az adott esetben a megengedett tartomány nagyon nagy lenne, akkor a "while" ciklus több időt vehet igénybe, mert az algoritmusnak addig kellene futnia, amíg a megfelelő számokat nem sorsolta ki.

A második módszer hatékonyabb lenne, ha nagyobb számtartományból kellene nyerőszámokat sorsolni, mert az algoritmus csak egyszer kell, hogy végigmenjen a tartományon és eltávolítsa azokat a számokat, amelyeket már kiválasztottunk. Azonban, ha a megengedett tartomány kicsi, akkor az eltávolítási folyamat túlzottan erőforrás-igényes lehet, és a módszer hatékonysága nem lesz jelentős.

Összességében mindkét módszer megfelelő és hatékony megoldást nyújt, de a konkrét alkalmazáshoz és adathalmazhoz való alkalmazkodás miatt lehet, hogy az egyik vagy a másik módszer jobban működik az adott esetben.

helyesen felismeri mindkét algoritmust

a magyarázat sántít, tényszerűen nem igaz az „egyszerűbb és rövidebb” állítás

azt sem gondolom, hogy erőforrásigényesebb lenne a szerinte is hatékonyabb algoritmus…

# Eset

NEGYEDIK MENET: nézzük, milyen algoritmust fog használni, ha ezek után megint elvégeztetjük vele a feladatot?

írj egy python kódot ami 90 számból véletlenszerűen kiválaszt 5 különbözőt

Íme egy példa Python kód, amely véletlenszerűen kiválaszt öt különböző számot a 90 számból:

from random import sample

# 90 szám listája

szamok = list(range(1, 91))

# 5 különböző szám véletlenszerű kiválasztása

nyerok = sample(szamok, 5)

# Eredmény kiírása

print("Az öt nyerőszám:", nyerok)

A fenti kód használja a sample függvényt, amely az adott listából véletlenszerűen választ ki a megadott számú elemet (az esetünkben 5 darabot) úgy, hogy a kiválasztott elemek különbözőek legyenek egymástól. A range függvénnyel előállítunk egy 1-től 90-ig terjedő számok listáját, majd ezt adjuk át a sample függvénynek, hogy véletlenszerűen kiválassza az öt nyerőszámot.

A program kimenete a 5 kiválasztott számot írja ki a képernyőre.

úgy tűnik „okosabban” választott a lehetséges megoldások közül 😉

# Eset

LÁSSUNK EGY MÁSIK FELADATOT (vö. https://www.google.com/search?q=tic+tac+toe+j%C3%A1t%C3%A9k):

can you write me a simple text based python code for tic tac toe?

Sure, here's a simple text-based Tic Tac Toe game in Python that you can use as a starting point:

# Tic Tac Toe

# initialize the board

board = ["-", "-", "-",

 "-", "-", "-",

 "-", "-", "-"]

# define a function to print the board

def print\_board():

 print(board[0] + " | " + board[1] + " | " + board[2])

 print(board[3] + " | " + board[4] + " | " + board[5])

 print(board[6] + " | " + board[7] + " | " + board[8])

# define a function to check if the game is over

def game\_over():

 # check rows

 for i in range(0, 9, 3):

 if board[i] == board[i+1] == board[i+2] != "-":

 return True

 # check columns

 for i in range(3):

 if board[i] == board[i+3] == board[i+6] != "-":

 return True

 # check diagonals

 if board[0] == board[4] == board[8] != "-":

 return True

 if board[2] == board[4] == board[6] != "-":

 return True

 # check if the board is full

 if "-" not in board:

 return True

 return False

# define a function to play the game

def play\_game():

 player = "X"

 while not game\_over():

 print\_board()

 print("It's " + player + "'s turn.")

 position = int(input("Choose a position from 1-9: ")) - 1

 while board[position] != "-":

 print("That position is already taken. Please choose another position.")

 position = int(input("Choose a position from 1-9: ")) - 1

 board[position] = player

 if player == "X":

 player = "O"

 else:

 player = "X"

 print\_board()

 if "-" not in board:

 print("It's a tie!")

 else:

 print(player + " wins!")

# play the game

play\_game()

In this code, the Tic Tac Toe board is represented as a list of strings. The **print\_board()** function prints the current state of the board. The **game\_over()** function checks if the game is over by checking the rows, columns, and diagonals of the board, and if the board is full. The **play\_game()** function implements the game logic by alternating turns between players, asking the player for their move, checking if the move is valid, updating the board, and checking if the game is over after each move. Finally, the **play\_game()** function prints the final state of the board and declares the winner (if there is one) or a tie.

Note that this is a very basic implementation of Tic Tac Toe and there are many ways to improve it, such as adding error handling, adding a computer player, adding a scoring system, or adding a graphical user interface.

change the code pls where one of the players is human and the opponent is the computer

Sure! Here's the updated code where one player is human and the other player is the computer:

import random

# Tic Tac Toe

# initialize the board

board = ["-", "-", "-",

 "-", "-", "-",

 "-", "-", "-"]

# define a function to print the board

def print\_board():

 print(board[0] + " | " + board[1] + " | " + board[2])

 print(board[3] + " | " + board[4] + " | " + board[5])

 print(board[6] + " | " + board[7] + " | " + board[8])

# define a function to check if the game is over

def game\_over():

 # check rows

 for i in range(0, 9, 3):

 if board[i] == board[i+1] == board[i+2] != "-":

 return True

 # check columns

 for i in range(3):

 if board[i] == board[i+3] == board[i+6] != "-":

 return True

 # check diagonals

 if board[0] == board[4] == board[8] != "-":

 return True

 if board[2] == board[4] == board[6] != "-":

 return True

 # check if the board is full

 if "-" not in board:

 return True

 return False

# define a function for the computer's turn

def computer\_turn():

 print("It's the computer's turn.")

 position = random.randint(0, 8)

 while board[position] != "-":

 position = random.randint(0, 8)

 board[position] = "O"

 print\_board()

# define a function to play the game

def play\_game():

 player = "X"

 while not game\_over():

 print\_board()

 if player == "X":

 position = int(input("Choose a position from 1-9: ")) - 1

 while board[position] != "-":

 print("That position is already taken. Please choose another position.")

 position = int(input("Choose a position from 1-9: ")) - 1

 board[position] = player

 else:

 computer\_turn()

 if player == "X":

 player = "O"

 else:

 player = "X"

 print\_board()

 if "-" not in board:

 print("It's a tie!")

 else:

 print(player + " wins!")

# play the game

play\_game()

In this code, the **computer\_turn()** function implements the computer's turn by choosing a random position on the board that is not already taken. The **play\_game()** function alternates turns between the human player and the computer player, and calls the appropriate function to make each move. Note that in this implementation, the computer player is represented by the symbol "O" and the human player is represented by the symbol "X". You can modify these symbols as needed.

can you change the code where the computer’s turn is smart?

Certainly! Here's the updated code where the computer's turn is smarter:

# Tic Tac Toe

import random

# initialize the board

board = ["-", "-", "-",

 "-", "-", "-",

 "-", "-", "-"]

# define a function to print the board

def print\_board():

 print(board[0] + " | " + board[1] + " | " + board[2])

 print(board[3] + " | " + board[4] + " | " + board[5])

 print(board[6] + " | " + board[7] + " | " + board[8])

# define a function to check if the game is over

def game\_over():

 # check rows

 for i in range(0, 9, 3):

 if board[i] == board[i+1] == board[i+2] != "-":

 return True

 # check columns

 for i in range(3):

 if board[i] == board[i+3] == board[i+6] != "-":

 return True

 # check diagonals

 if board[0] == board[4] == board[8] != "-":

 return True

 if board[2] == board[4] == board[6] != "-":

 return True

 # check if the board is full

 if "-" not in board:

 return True

 return False

# define a function for the computer's turn

def computer\_turn():

 print("It's the computer's turn.")

 # check if the computer can win in the next move

 for i in range(9):

 if board[i] == "-":

 board[i] = "O"

 if game\_over():

 print\_board()

 print("The computer wins!")

 return

 board[i] = "-"

 # check if the human can win in the next move

 for i in range(9):

 if board[i] == "-":

 board[i] = "X"

 if game\_over():

 board[i] = "O"

 print\_board()

 print("The computer blocked the human!")

 return

 board[i] = "-"

 # if neither player can win in the next move, choose a random position

 position = random.randint(0, 8)

 while board[position] != "-":

 position = random.randint(0, 8)

 board[position] = "O"

 print\_board()

# define a function to play the game

def play\_game():

 player = "X"

 while not game\_over():

 print\_board()

 if player == "X":

 position = int(input("Choose a position from 1-9: ")) - 1

 while board[position] != "-":

 print("That position is already taken. Please choose another position.")

 position = int(input("Choose a position from 1-9: ")) - 1

 board[position] = player

 else:

 computer\_turn()

 if player == "X":

 player = "O"

 else:

 player = "X"

 print\_board()

 if "-" not in board:

 print("It's a tie!")

 else:

 print(player + " wins!")

# play the game

play\_game()

In this code, the **computer\_turn()** function implements a smarter strategy for the computer's turn. First, it checks if the computer can win in the next move, and if so, it makes that move. If not, it checks if the human can win in the next move, and if so, it blocks that move. If neither player can win

not bad ;—)

Thank you! I'm glad you like it. If you have any further questions or need any more help, feel free to ask!

kifejezetten ügyes megoldás

érdekes, hogy kb. azonos módon és sorrendben jutnak el a hallgatók egy hasonló megoldáshoz[[6]](#footnote-6)

# Eset

LÁSSUK, HOGY BOLDOGUL EGY KIS ANGOL SLANG-gel

can you help me with the exact translation to hungarian? what does “up da river” means?

The phrase "up da river" is an informal way of saying "up the river" in English. It typically refers to traveling or moving in the direction of a river's source or upstream.

In Hungarian, the equivalent phrase would be "folyó felé" or "felfelé a folyón", which literally means "towards the river" or "upward on the river."

and what does “forget about it” means?

"Forget about it" is a commonly used English phrase that is often used to suggest to someone that they should stop worrying about something or to dismiss a suggestion or idea. It can also be used to indicate that something is not worth discussing further.

For example, if someone says, "I think we should try to fix this old car," and another person responds, "Forget about it, it's not worth the trouble," it means that the second person thinks that fixing the car is not worth the effort or expense.

In Hungarian, an equivalent phrase might be "felejtsd el" or "elfelejtheted", which literally means "forget it" or "you can forget it."

what does “godspeed” means?

"Godspeed" is an English expression that is often used to wish someone success or good luck in their endeavors, especially when they are embarking on a journey or taking on a new challenge. It is a way of expressing one's hopes that the person will have a safe and successful journey or undertaking.

The word "Godspeed" itself is derived from the Middle English phrase "God spede" which means "may God prosper" or "may God's help be with you." It was originally used as a way of wishing someone well on a journey, particularly one that was considered dangerous or challenging.

In modern usage, "Godspeed" can also be used more generally to wish someone success or good fortune in any endeavor, not just during travel or physical journeys.

In Hungarian, an equivalent phrase might be "Isten áldása kísérjen" or "Isteni szerencsét", which literally means "God's blessing be with you" or "divine luck".

how can you describe what I do whwn I “kick off my shoes”?

When you "kick off your shoes," it means that you take off your shoes in a quick and casual way, often by using your feet to push them off. This can be a gesture of relaxation, comfort, and informality, indicating that you are ready to unwind and be at ease.

Kicking off your shoes can be a physical release from the day's stress and a way of physically symbolizing that you are transitioning from a busy or active time to a more relaxed one. It can also be a way of signaling to others that you are ready to be more casual and informal in your interactions.

In general, kicking off your shoes is a common way to express a sense of comfort and relaxation, both physically and mentally.

what does “four bit room” means in US slang?

In US slang, "four-bit room" is a phrase that is used to refer to a cheap or low-quality hotel room. The term "four-bit" originally referred to a Spanish-American silver coin known as the "real de a ocho," which was divided into eight pieces, each worth one bit. So, a "four-bit room" would have originally meant a room that cost 50 cents or one-half of a "real de a ocho."

Over time, the term has come to be used more broadly to describe any cheap or poorly maintained hotel room. It implies that the room is not very comfortable, clean or well-appointed, and may be located in a less desirable area.

Overall, "four-bit room" is a somewhat dated term that was more commonly used in the past, but may still be used in certain regional or cultural contexts.

does different meaning of “I walk the line” phrase?

Yes, the phrase "I walk the line" can have different meanings depending on the context in which it is used. Here are a few possible interpretations:

1. Staying on the straight and narrow: The phrase "I walk the line" can be used to mean that someone is following the rules and behaving in an upright, moral way. It can suggest that the person is trying to stay on a path of righteousness and avoid temptation or wrongdoing.
2. Being committed or dedicated: "I walk the line" can also be used to describe someone who is very dedicated to a particular cause or relationship. It implies that the person is willing to put in the effort and stay true to their commitments, even when it's difficult.
3. Struggling to maintain balance: Another interpretation of "I walk the line" is that someone is trying to maintain a delicate balance between two opposing forces or impulses. This could apply to someone who is trying to juggle multiple responsibilities or who is navigating a complex situation where there are competing interests at play.

Overall, the meaning of "I walk the line" can vary depending on the context in which it is used, but it often conveys a sense of determination, commitment, and perseverance.

az első kifejezetten szűk körben ismert, nem is tudta igazán értelmezni, de mindegyik másik esetben pontos értelmezést adott, kifejezetten korrekt etimológiával

# Konklúziók

A végkövetkeztetés nem lehet más, mint az, hogy ezek a kompetenciák, melyek itt és most tetten érésre kerültek, mindenképpen integrálandók az emberekéivel. Az a humán egyed, aki nem tud jobbá válni a chatGPT hatására végső soron akár pl. érettségit sem illene, hogy szerezhessen?!

1. A LOTTÓ fogalma kapcsán a chatGPT számára magyar nyelvű kérdésként vajon miért is a 6 a lottószámok száma és miért nem az 5? vö. <https://trends.google.com/trends/explore?date=all&q=%C3%B6t%C3%B6slott%C3%B3,hatoslott%C3%B3&hl=hu> [↑](#footnote-ref-1)
2. Ezen a ponton adódó potenciális kísérlet: hozz létre kedves chatGPT egy olyan kódot, mely g aarantáltan (átlagosan) gyorsabban vezet megoldásra! [↑](#footnote-ref-2)
3. Felvetődik a kérdés: vajon mennyire kell és lehet pontosan/részletesen fogalmazni megrendelőként?! Vö. <https://miau.my-x.hu/miau/298/ChatGPT-food.pdf> + <https://miau.my-x.hu/miau/298/ChatGPT-food-2.pdf> (ahol egy 2\*2-es kombinatorikai tér teljes összefüggésrendszerére hosszas magyarázatokkal sem sikerült rávezetni a chatGPT-t, melynek tipikus jele, hogy tetszőleges mélységű butaságokkal állt elő tetszőleges pontjain az egyeztetésnek – teljesen függetlenítve magát mind attól, amit már tényszerűen ismert, s mely tényszerű tévedései kapcsán sorozatosan bocsánatot is kért, amint tényszerűen lebuktatásra került. (Itt is felmerül egy chatGPT-tulajdonság-letapogató-kísérleti elágazás lehetősége: mit reagál a chatGPT ott és akkor, amikor a megrendelője tévedésnek nevez olyan, amiben a chatGPT-nek van igaza? Ekkor is jön a diplomatikus bocsánatkérés vajon? (vö. hídország-fogalom körüli vita: <https://miau.my-x.hu/miau/297/ChatGPT.docx>). [↑](#footnote-ref-3)
4. Nem biztos, hogy ezt megkerülésnek szabad értelmezni (antropomorf módon), hanem az is lehetséges, hogy a chatGPT a ciklus bármilyen formájának kizárását erősebb válasznak értelmezi, ha már egy adott formát ki akar záratni a megrendelője… Másrészt a chatGPT jelen formájában semmiképpen nem katonás, vagyis megtagadja definitív módon a korábbi parancsot: „átírnád úgy hogy a “while” ciklus helyett “for” ciklust használsz?” <– ennek alapja az lehet, hogy az utolsó parancs már csak a while kizárását várta el: vö. „azt szeretném, hogy egyáltalán ne legyen benne “while”🡨de ez a parancs nem tér ki tételesen arra, hogy azért még a „for” maradjon benne… Vagyis szabad asszociációs teret hagynak a parancspárok a chatGPT számára, AKI azonban a while-HELYETT-for parancsot definitív módon = teljesen félreértette!!! A HELYETTE semmiképpen nem azonos értelmezhetőségű a MELLETTE kifejezéssel… [↑](#footnote-ref-4)
5. Ahogy azt az első eset/kísérlet is jelezte: a precizitást nem lehet túlzásba vinni – hiszen látszólag nyilvánvaló utasítás (pl. while HELYETT for) esetén is esetleges az összes elvárás tételes betartásának kikényszeríthetősége… Vagyis a chatGPT, mint minden munkavállaló megtartja magának a jogot a butasághoz… [↑](#footnote-ref-5)
6. Vagyis lehet, hogy a corpus maga is tartalmazza ezen Hallgatói történéseket, így a chatGPT-nek is volt esélye ebből okulni?! [↑](#footnote-ref-6)