# Rapid projekt: Monty-Hall probléma megoldása pythonban

## Ember vs ChagGPT

Tartalom

[Feladat: 2](#_Toc148340011)

[Kódok: 2](#_Toc148340012)

[Összehasonlítás: 4](#_Toc148340013)

[Futásidő: 4](#_Toc148340014)

[Konklúzió: 4](#_Toc148340015)

### Feladat:

2023.10.14-i adatbázisszerkezetek és algoritmusok órán gondolatkísérletként elhangzott a Monty-Hall probléma (<https://en.wikipedia.org/wiki/Monty_Hall_problem>) excelben/bármilyen programozási nyelven történő megoldása.

### Kódok:

#### emberi kód:

from random import randrange

import time

start = time.time()

print(" \_   \_\_              \_         \_\_\_        \_        ")

print("| | / /             | |       / \_ \      | |       ")

print("| |/ /  \_\_\_  \_\_\_ \_\_\_| | \_\_\_\_\_/ /\_\ \\_   \_| |\_ \_\_\_  ")

print("|    \ / \_ \/ \_\_/ \_\_| |/ / \_ \  \_  | | | | \_\_/ \_ \ ")

print("| |\  \  \_\_/ (\_\_\\_\_ \   <  \_\_/ | | | |\_| | || (\_) |")

print("\\_| \\_/\\_\_\_|\\_\_\_|\_\_\_/\_|\\_\\_\_\_\\_| |\_/\\_\_,\_|\\_\_\\_\_\_/ ")

#keszitunk egy ures listat a vegeredmenyhez

v=[]

# vagyazvan, hogy elore definialom a lehetseges listakat es azt generalom bele a nagylistaba

'''

v1=[0,0,1]

v2=[0,1,0]

v3=[1,0,0]

for i in range(1000):

    choose=randrange(3)

    if choose==0:

        v.append(v1)

    if choose==1:

        v.append(v2)

    if choose==2:

        v.append(v3)

'''

# vagyaz, hogy elore feltoltom nullakkal es a generalt indexet updatelem 1-esre (v2.0)

lista=[0,0,0]

for i in range(1000):

    lista[randrange(3)]=1

    v.append(lista)

    lista=[0,0,0]

#lista a tippet tartalmazza

harmas=[1,2,3]

nynv=0 #nyert, nem valtoztatott

nyv=0 #nyert, valtoztatott

vnv=0 #veszitett es nem valtoztatott

vv=0 #veszitett es valtoztatott

for i in range(len(v)):

    #kigeneralunk egy aktualis tippet

    akt=randrange(len(harmas))

    tipp=harmas[akt]

    #kitoroljuk az adott elemet, igy legkozelebb egy n-1 listabol kell generalni csak, valamint nemlehet ujra ugyanazt megadni

    del harmas[akt]

    akt=randrange(len(harmas))

    #a szukitett listabol tippel a showman

    showman=harmas[akt]

    harmas=[1,2,3]

    #megadjuk hogy random valtoztatott-e az ember a tippjen

    valtoztate=randrange(2)

    #kiirjuk, hogy jobban lassunk

    #print(f'vitrinek: {v[i]} | tipp: {tipp} | showman: {showman} | valtoztat-e: {valtoztate}')

    if valtoztate==1:

        tipp=showman

    if ((v[i].index(0)+1)==tipp) and valtoztate==0:

        nynv=nynv+1

    if ((v[i].index(0)+1)==tipp) and valtoztate==1:

        nyv=nyv+1

    if ((v[i].index(0)+1)!=tipp) and valtoztate==0:

        vnv=vnv+1

    if ((v[i].index(0)+1)!=tipp) and valtoztate==1:

        vv=vv+1

#vegeredmeny kiirasa

print(f'nyert,valtoztatott: {nyv}')

print(f'nyert,NEM valtoztatott: {nynv}')

print(f'veszitett,valtoztatott: {vv}')

print(f'veszitett,NEM valtoztatott: {vnv}')

end = time.time()

#futasido kiirasa

print(f"Time taken: {(end-start)\*10\*\*3:.03f}ms")

#### ChatGPT kódja:

import random

import time

start = time.time()

#a chatgpt nagyon helyesen def-be irja meg a programot

def monty\_hall(simulations=1000, switch=True):

    win\_count = 0

    i=0

    for \_ in range(simulations):

        i=i+1

        # Az autó helyét és a játékos választását véletlenszerűen kiválasztjuk. -> chatgpt komment

        car\_position = random.randint(1, 3)

        player\_choice = random.randint(1, 3)

        # Monty kinyit egy, a játékos által nem választott és autóval nem rendelkező ajtót. -> chatgpt komment

        #itt a boduletes hiba. generalas utan ellenorzi, hogy jo-e a Monty altal tett valasztas (ha pl 10.000 elemrol es 8.000 valasztasi lehetosegrol lenne szo a jatekban, ez a proram elhasalna

        remaining\_choices = [choice for choice in range(1, 4)

                             if choice != player\_choice and choice != car\_position]

        monty\_opens = random.choice(remaining\_choices)

        # Ha a játékos vált, akkor a másik, Monty által nem nyitott ajtót választja. -> chatgpt komment

        if switch:

            player\_choice = next(choice for choice in range(1, 4)

                                 if choice != player\_choice and choice != monty\_opens)

        # Ellenőrizzük, hogy a játékos nyert-e.

        if player\_choice == car\_position:

            win\_count += 1

    return win\_count / simulations

# Futassuk a szimulációt mindkét esetben: váltás és nem váltás. -> chatgpt komment

simulations = 1000

print(f"Winning probability when switching: {monty\_hall(simulations, True):.2%}")

print(f"Winning probability when not switching: {monty\_hall(simulations, False):.2%}")

#futasido kiirasa

end = time.time()

print(f"Time taken: {(end-start)\*10\*\*3:.03f}ms")

### Összehasonlítás:

A ChatGPT a második kérdésre sem oldotta meg helyesen, mivel a showman választásánál folyamatosan ellenőrzi, hogy helyes-e a választás. Jelen példában kiválóan lefut, de ha pl. 10.000 vitrin lenne és 9.999 választási/döntési lehetőség, mire kigenerálná a maradék mondjuk 10-ből 1-et, vélhetően megfeküdne a program.

### Futásidő:

ember: 5.003ms

ChatGPT: 8.001ms

### Konklúzió:

A ChatGPT-t „for educational purposes” miatt tervezték. Kiválóan elmagyarázza és helyesen is ír kódot. Ellenben a tudásbázisa -mivel a netről származik- sok esetben nem képes olyan alapvető körülményeket figyelembe venni ami alapjában véve logikus lenne (pl. erőforrás kihasználása, stb.).

**Genetikai potenciál-elvű automatikus kódjavító eljárás,** mint szakdolgozati / diplomadolgozati / PhD-disszertációként értelmezhető téma

Előzmények: <https://miau.my-x.hu/miau/292/Barta_Gergo_ertekezes.pdf>

Kiindulási állapot: szülessen minél több, egymástól bármilyen módon eltérő (vö. genetikai/evolúció programozás szülői generációja) forráskód a Monty-Hall paradoxon úm. brute-force jellegű (gyakorisági-alapú) „bizonyítására”.

Cél: a legjobb szülőnél is jobb, ill. a legeslegjobb utódmegoldás megtalálása anélkül, hogy bármilyen logikai magyarázatot kelljen a (robot) programozónak adni/kapni (vö. robot GO-stratégia csak log-ok alapján – ill. sakk-stratégia, amőba-stratégia, stb.)

Hasznosság: adott forráskód jobbá tétele kontextus-függetlenül sok-sok korábbi megoldás-kísérlet jóságának feltárásával és összehasonlításával annak érdekében, hogy a legjobb megoldás megtalálási folyamata adott (komplex = nem egy-skálás pl. legrövidebb kód) jóság-definíció esetén automatizálható legyen (= hatékonyságnövekedés = rövidebb idő alatt megtalálni tudni a jobb forráskódot, de legalább azt a forráskód pontot, ahol a beavatkozást az emberi zsenialitásnak kellene megálmodnia tudni)

Feladatok: a szülői populáció (objektum = forráskódvariáns) kialakítása, a legjobb objektum megtalálását lehetővé tevő jóság-rétegek definiálása (log-alapon), ezek adattal való ellátása minden objektum esetére (OAM), a legjobb objektum(ok) levezetése (Y0🡨anti-diszkriminatív modell), a GENETIKAI (Xi=log-alapú attribútumok, Y=jóságindex 🡨termelési függvény = STD-modell), a szimulációs tér értelmezése, a jóság-skálán előrelépést sejtető beavatkozási pontok feltárása, ezek várható hatásának becslése a jóság-indexre, ill. a beavatkozás operatív mikéntjének levezetése