Python alapok!

Változók szabályai Pythonban

☑ Érvényes változónév szabályok
Betűvel vagy aláhúzásjellel kell kezdődni
Példák: nev, _szam, x

A név a-z, A-Z, 0-9 és _ (aláhúzás) karaktereket tartalmazhat
Példák: nev1, pelda_ertek, adat_2

Nem kezdődhet számmal
❌ Hibás: 1nev, 2_adat

Nincs szükség változó típusának előzetes megadására
Python dinamikusan típusos nyelv: x = 5 után x egy egész szám (int), de később akár string is lehet (x = "alma").

A változónév kis- és nagybetűérzékeny
szam, Szam és SZAM három különböző változó.
De általában érdemes kisbetűket alkalmazni

Használj beszédes neveket – ez jó gyakorlat, nem szabály, de javasolt:
Rossz: a, b
Jó: ossz_ertek, felhasznalo_nev

Foglal szavak (kulcsszavak), amelyeket nem használhatsz változónévként
Pythonban ezek előre lefoglalt nevek, amelyek a nyelv működéséhez szükségesek. Ha változónévként próbálod használni, szintaktikai hiba lesz.

Python 3.x foglalt kulcsszavak (teljes lista):

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Ezek nem használhatók változónévként! Például:

python
if = 5 # ❌ SyntaxError
class = "teszt" # ❌ SyntaxError

Jó tanácsok változónév választáshoz
Ne használj túl rövid (pl. x) vagy túl hosszú változóneveket.
Legyen egyértelmű a változó jelentése (nev, kor, ar, ossz_ertek stb.).
Ha több szóból áll, használd az aláhúzást szóelválasztásra (osszeg_ertek).
Ne nevezd el változónak az olyan beépített függvényeket, mint pl. list, str, input, sum, mert ezeket felülírja:

python
list = [1, 2, 3] # rossz ötlet, mert a 'list()' függvényt nem tudod többé elérni

Tizedes jel

Nem vesszőt, hanem pontot használunk

Szöveg ’str’ -zöld
Szám ’int’ - rózsaszín
Tizedes tört - ’float’
True False - ’booll’ halvány kék

Típusok

Alapvető adattípusok Pythonban

1. Számok (Numbers)

Típus	Leírás	Példa
int	Egész szám	x = 10
float	Tört szám (lebegőpontos szám)	pi = 3.14
complex	Komplex szám	z = 2 + 3j

2. Szöveg (String)

Típus	Leírás	Példa
str	Karakterlánc (szöveg)	nev = "Anna"

Lehet ' ', " ", vagy """ """ idézőjelekkel
A karakterek listaként is kezelhetők: nev[0] → 'A'

3. Logikai érték (Boolean)

Típus	Leírás	Példa
bool	Igazságérték	True, False

Például: 5 > 2 → True

4. Sorozatok (Sequence Types)

Típus	**Leírás**	**Példa**
list	Lista (módosítható)	szamok = [1, 2, 3]
tuple	Tuple (nem módosítható)	koordinata = (10, 20)
range	Tartomány (számsorozat)	range(0, 5)

5. Szótár (Dictionary)

Típus	**Leírás**	**Példa**
dict	Kulcs-érték párokból álló struktúra	szemely = {"nev": "Anna", "kor": 25}

6. Halmazok (Sets)

Típus	**Leírás**	**Példa**
set	Egyedi elemek halmaza	halmaz = {1, 2, 3}
frozenset	Nem módosítható halmaz	fs = frozenset([1, 2, 3])

Példák:

```
python
x = 10              # int
y = 3.14            # float
z = 2 + 5j          # complex
nev = "Béla"        # str
igaz = True         # bool
lista = [1, 2, 3]   # list
tuple_ = (4, 5)     # tuple
szotar = {"a": 1}   # dict
halmaz = {1, 2, 3}  # set
```

Típus lekérdezése

A type() függvény segítségével megnézhetjük egy érték adattípusát:

```
python
print(type(10))        # <class 'int'>
print(type("hello"))   # <class 'str'>
```

🔢 Alapvető matematikai operátorok Pythonban

Operátor	Jelentés	Példa	Eredmény
+	Összeadás	5 + 3	8
-	Kivonás	5 - 3	2
*	Szorzás	5 * 3	15
/	Osztás (float eredmény)	5 / 2	2.5
//	Egész osztás	5 // 2	2
%	Maradék (modulus)	5 % 2	1
**	Hatványozás	2 ** 3	8

Példák használatra

python

```
a = 10
b = 3

print(a + b)   # 13
print(a - b)   # 7
print(a * b)   # 30
print(a / b)   # 3.333...
print(a // b)  # 3
print(a % b)   # 1
print(a ** b)  # 1000
```

Fontos tudnivalók

- Az / mindig **lebegőpontos számot (float)** ad vissza akkor is, ha az osztás pontos (10 / 2 → 5.0).
- A // egészrészt ad vissza, **levágva a tizedeseket**.
- A % maradékos osztást végez (pl. páros szám vizsgálatához: x % 2 == 0).
- A ** hatványozás, tehát 2 ** 4 azt jelenti: $2^4 = 16$

Haladó: Összetett értékadó operátorok

Operátor	Jelentés	Példa	Ekvivalens vele
+=	Növelés	x += 1	x = x + 1
-=	Csökkentés	x -= 2	x = x - 2
*=	Szorzás és értékadás	x *= 3	x = x * 3
/=	Osztás és értékadás	x /= 4	x = x / 4
//=	Egész osztás és értékadás	x //= 2	x = x // 2
%=	Maradék és értékadás	x %= 3	x = x % 3
**=	Hatványozás és értékadás	x **= 2	x = x ** 2

Adatbevitel – input() függvény

- Alap szintaxis:

python
valtozo = input("Írj be valamit: ")

Ez mindig **szöveget (str)** ad vissza, még akkor is, ha számot ír be a felhasználó!

Példa:
python
nev = input("Mi a neved? ")
print("Szia,", nev + "!")

Kimenet:
Mi a neved? Laci
Szia, Laci!

🔢 Számbevitel – típuskonverzióval

Mivel az input() mindig **stringet** ad vissza, ha számot szeretnél beolvasni, akkor azt **át kell alakítani**:

- Egész szám (int):

python
kor = int(input("Hány éves vagy? "))
print("Jövőre", kor + 1, "éves leszel.")

- Tört szám (float):

python
magassag = float(input("Add meg a magasságod (m): "))

print("A megadott magasság:", magassag, "méter")

⚠ Hibakezelés – ha nem számot ír be a felhasználó

Az alábbi módon lehet elkerülni hibát, ha pl. valaki nem számot ír be:

python

```
try:
    szam = int(input("Adj meg egy számot: "))
    print("A szám kétszerese:", szam * 2)

except ValueError:
    print("Ez nem szám!")
```

🧠 Tipp: Egyszerre több adat beolvasása

python

```
a, b = input("Adj meg két számot szóközzel elválasztva: ").split()
a = int(a)
b = int(b)
print("Összegük:", a + b)
```

🖨 print() függvény – alap szintaxis

python

```
print(kifejezés1, kifejezés2, ..., sep=' ', end='\n')
```

☑ Alapvető használat

python

```
print("Helló, világ!")
```

Kimenet:

Helló, világ!

🔁 Több érték kiíratása egyszerre

python

```
nev = "Anna"
kor = 25
print("Név:", nev, "Kor:", kor)
```

Kimenet:

Név: Anna Kor: 25

(A print() alapból szóközt tesz az értékek közé.)

🔧 Gyakori paraméterek

1. sep – elválasztó

A sep paraméter szabályozza, hogy **mi kerüljön a kiírt értékek közé**.

python
print("2025", "05", "20", sep="-")

Kimenet:
2025-05-20

---

2. end – sorvége

Az end paraméter határozza meg, **mi kerüljön a sor végére** (alapból: \n, azaz sortörés).

python
print("Első sor", end=" | ")
print("Második sor")

Kimenet:
Első sor | Második sor

🎯 Példák változókkal

python
a = 5
b = 10
print("Az összeg:", a + b)

Kimenet:
Az összeg: 15

🔤 F-String – formázott kiírás (Python 3.6+)

python
nev = "Laci"
kor = 30
print(f"{nev} {kor} éves.") # f betű a string előtt!

Kimenet:

Laci 30 éves.

📜 Összefoglalás: print() használata

Feladat	Példa
Szöveg kiírása	print("Helló!")
Változó kiíratása	print(szam)
Több érték kiírása	print(a, b, c)
Elválasztó módosítása	print("a", "b", "c", sep="-")
Sorvégződés módosítása	print("Üdv", end="!")
Formázott kiírás (f-string)	print(f"{nev} {kor} éves.")

Feltételes utasítás

A Pythonban a **feltételes utasításokat** akkor használjuk, ha a programnak **döntést** kell hoznia – vagyis ha **valami igaz vagy hamis**, és ennek megfelelően különböző utasításokat kell végrehajtani.
if - ha
else - különben

◇ Alap: if utasítás

python
```
if feltétel:
    utasítás
```

Példa:
python
```
szam = 10
if szam > 0:
    print("A szám pozitív.")
```

◇ if – else: két lehetőség

python
```
if feltétel:
    utasítás1
else:
    utasítás2
```

Példa:

```
python
szam = -5
if szam >= 0:
    print("Nem negatív")
else:
    print("Negatív szám")
```

- **if – elif – else: több lehetőség**

```
python
if feltétel1:
    utasítás1
elif feltétel2:
    utasítás2
else:
    utasítás3
```

Példa:

```
python
jegy = 4
if jegy == 5:
    print("Jeles")
elif jegy == 4:
    print("Jó")
elif jegy == 3:
    print("Közepes")
elif jegy == 2:
    print("Elégséges")
else:
    print("Elégtelen")
```

☑ **Logikai operátorok feltételvizsgálatban**

Operátor	Jelentés	Példa
==	egyenlőség	a == b
!=	nem egyenlő	a != b
>	nagyobb mint	a > b
<	kisebb mint	a < b
>=	nagyobb vagy egyenlő	a >= b
<=	kisebb vagy egyenlő	a <= b
and	és	a > 0 and b > 0
or	vagy	a > 0 or b > 0
not	tagadás	not a > 0

🔍 **Példa összetett feltételre:**

python

```
kor = 18
allampolgar = True

if kor >= 18 and allampolgar:
    print("Szavazhat")
else:
    print("Nem szavazhat")
```

🎯 **Fontos:**

- A **feltételek után kettőspont (:)** szükséges!
- A **beljebb húzott sorok (indentálás)** mutatják, hogy mely utasítások tartoznak az adott blokkhoz.
- **Pythonban nincs switch** – helyette több elif használható.

A **if – if – if** és a **if – elif – else** szerkezetek **különböző logikát** valósítanak meg, és nem felcserélhetők minden helyzetben.

🔍 **1. if – if – if:**

👉 **Minden feltételt külön vizsgál meg**, egymástól **függetlenül**.

python

```
jegy = 5
```

```
if jegy == 5:
   print("Jeles")
if jegy == 4:
   print("Jó")
if jegy == 3:
   print("Közepes")
```

➕ Előny:

- Több igaz feltétel is lefuthat egymás után.
- Jó, ha **több feltétel egyszerre is igaz lehet**.

➖ Hátrány:

- Akkor is ellenőrzi a többi feltételt, ha már talált egy igazat.
- Nincs kizáró logika.

🔍 **2. if – elif – else:**

👉 Csak **az első igaz feltételt hajtja végre**, a **többit kihagyja**.

python

```
jegy = 5

if jegy == 5:
   print("Jeles")
elif jegy == 4:
   print("Jó")
elif jegy == 3:
   print("Közepes")
else:
   print("Ismeretlen jegy")
```

➕ Előny:

- Hatékonyabb, mert amint talál egy igaz feltételt, **kilép**.
- Jó, ha a feltételek **kölcsönösen kizárják** egymást.

➖ Hátrány:

- Csak az első igaz feltételt vizsgálja meg, a többit nem.

🧠 Mikor melyiket használd?

Használat	Szerkezet	Mikor?
Több igaz feltétel is lehet egyszerre	if – if – if	Pl.: Ha valaki nagykorú **és** dolgozik is
Csak egy feltétel lehet igaz (pl. osztályzatok)	if – elif – else	Pl.: Vizsgaeredmény kiírása, menüpont választás
Több opció közül kizárólag egy legyen kiválasztva	if – elif – elif – else	Szinte mindig ez a menüből választásnál, kategorizálásnál

---

□ Példa: különbség működés közben

if – if:

python
```
x = 10
if x > 5:
    print("Nagyobb mint 5")
if x > 0:
    print("Pozitív szám")
```

Kimenet:

Nagyobb mint 5
Pozitív szám

if – elif:

python
```
x = 10
if x > 5:
    print("Nagyobb mint 5")
elif x > 0:
    print("Pozitív szám")
```

Kimenet:

Nagyobb mint 5
(A második nem fut le.)

Logikai kifejezések

A **logikai kifejezések** (más néven *logikai feltételek* vagy *boolean expressions*) a Pythonban olyan kifejezések, amelyek **logikai értéket adnak vissza**: True vagy False. Ezek a **feltételes utasítások** (pl. if, while) alapját képezik.

◇ Egyszerű logikai kifejezések – összehasonlító operátorokkal

Operátor	Jelentés	Példa	Eredmény
==	egyenlő	5 == 5	True
!=	nem egyenlő	3 != 4	True
>	nagyobb	6 > 2	True
<	kisebb	2 < 1	False
>=	nagyobb vagy egyenlő	3 >= 3	True
<=	kisebb vagy egyenlő	2 <= 5	True

◇ Logikai operátorok – feltételek kombinálásához

Operátor	Jelentés	Példa	Eredmény
and	és	x > 0 and x < 10	csak akkor True, ha mindkettő igaz
or	vagy	x < 0 or x > 100	ha **legalább egy** igaz, akkor True
not	tagadás	not(x > 10)	ellentétét adja

🧠 Példák logikai kifejezésekre

python

```
x = 7

print(x == 7)       # True
print(x != 5)       # True
print(x > 3 and x < 10)   # True
print(x < 3 or x > 10)    # False
print(not(x == 7))   # False
```

□ Logikai kifejezések if-ben

python

```
nev = "Anna"
kor = 18

if kor >= 18 and nev != "":
```

```
print("Felnőtt, van neve")
```

⚠ Pythonos érdekesség: „Igaznak értékelt” dolgok

Pythonban **nemcsak a True és False lehet logikai érték**, hanem sok **más típus is konvertálódik logikai értékké**, például:

◇ "Hamisnak" számít:

- 0, 0.0
- '' (üres string)
- [], {}, (), set() (üres tárolók)
- None

◇ "Igaznak" számít:

- Minden nem üres, nem nulla érték.

Példa:

python

```
if []:
    print("Ez nem fut le")  # üres lista -> False
if "Hello":
    print("Ez lefut")       # nem üres string -> True
```

Véletlen számok

A **véletlenszámok generálása** a Pythonban a random nevű beépített modul segítségével történik.

🎲 1. random modul importálása

python

```
import random
```

Ezután használhatók a véletlenszám-generáló függvények.

🔢 2. Véletlen egész szám generálása – randint()

python

```
random.randint(alsó, felső)
```

- **Mindkét határt beleérti** (zárt intervallum)

Példa:

python

```
import random

szam = random.randint(1, 10)
```

print("A véletlen szám:", szam)

□ 3. Véletlen tört szám – random() és uniform()

random.random()

- 0 és 1 közötti tört számot ad vissza ($0 \leq x < 1$)

python
r = random.random()
print(r)

random.uniform(a, b)

- két tetszőleges szám közötti **tört szám**

python
r = random.uniform(1, 5)
print(r) # pl. 3.748372

🎰 4. Listaelemek közül véletlen választás – choice()

python
valasztas = random.choice(["alma", "körte", "barack"])
print(valasztas)

🔄 5. Listaelemek véletlen sorrendbe állítása – shuffle()

python
lista = [1, 2, 3, 4, 5]
random.shuffle(lista)
print(lista)

🧾 Összefoglaló: random legfontosabb függvényei

Függvény	Leírás
randint(a, b)	Egész számot ad [a, b] intervallumból
random()	Véletlen tört szám [0, 1) intervallumból
uniform(a, b)	Véletlen tört szám [a, b] intervallumból
choice(lista)	Véletlenszerűen kiválaszt egy elemet
shuffle(lista)	Véletlenszerűen összekeveri a listát
sample(lista, k)	k darab **egyedi** elemet választ

💡 Gyakorlati példa – Dobókocka

python
```
import random

dobas = random.randint(1, 6)
print("A dobókocka eredménye:", dobas)
```

while ciklus

A ChatGPT ezt mondta:

A while ciklus a Pythonban **addig hajtja végre az utasításokat, amíg a megadott feltétel igaz (True)**. Akkor használjuk, ha **nem tudjuk előre, hányszor kell ismételni** valamit – csak azt tudjuk, hogy **meddig**.

🔁 while ciklus – alapszerkezet

python
```
while feltétel:
    utasítás(ok)
```

- A ciklus minden iteráció előtt ellenőrzi a feltételt.
- Ha a feltétel **igaz**, akkor lefut a blokk.
- Ha a feltétel **hamis**, a ciklus **leáll**.

□ Egyszerű példa

python
```
szam = 1
while szam <= 5:
    print(szam)
    szam += 1
```

Kimenet:

1
2
3
4
5

⚠ Vigyázat: végtelen ciklus!

Ha a feltétel **soha nem válik hamissá**, a ciklus **végtelenül fut**:

python

```
while True:
    print("Ez végtelen ciklus!")  # csak Ctrl + C-vel állítható le!
```

Kilépés a ciklusból – break

python

```
while True:
    szam = int(input("Írj be egy számot (0 = kilépés): "))
    if szam == 0:
        break
    print("A szám négyzete:", szam ** 2)
```

Egy iteráció kihagyása – continue

python

```
szam = 0
while szam < 10:
    szam += 1
    if szam % 2 == 0:
        continue  # páros számok kihagyása
    print(szam)
```

Kimenet:

1
3
5
7
9

Gyakorlati példa – jelszó bekérése

python

```
jelszo = ""
while jelszo != "titok123":
    jelszo = input("Add meg a jelszót: ")
print("Sikeres bejelentkezés!")
```

Összefoglalás

Elem	Leírás
while	Ismétel, amíg a feltétel igaz
break	Megszakítja a ciklust
continue	Kihagyja az aktuális ciklusléptést

Ciklusok egymásba ágyazása

A **ciklusok egymásba ágyazása** (más néven **beágyazott ciklusok**) azt jelenti, hogy egy ciklus **egy másik cikluson belül** fut. Ez gyakori szerkezet például **táblázatok**, **mátrixok**, vagy **kombinációk** bejárásánál.

Alapszerkezet – beágyazott ciklus

python
```
for i in range(3):
    for j in range(2):
        print(f"i = {i}, j = {j}")
```

Kimenet:

```
i = 0, j = 0
i = 0, j = 1
i = 1, j = 0
i = 1, j = 1
i = 2, j = 0
i = 2, j = 1
```

Beágyazott while ciklus példa

python
```
i = 1
while i <= 3:
    j = 1
    while j <= 2:
        print(f"{i} - {j}")
        j += 1
    i += 1
```

🔁 for a while-ban (vagy fordítva)

Vegyes ciklusok is lehetnek:

python

```
szamok = [1, 2, 3]
i = 0
while i < len(szamok):
    for j in range(szamok[i]):
        print(f"{szamok[i]} → {j}")
    i += 1
```

□ Gyakorlati példa – Szorzótábla

python

```
for i in range(1, 6):
    for j in range(1, 6):
        print(f"{i*j:2}", end=" ")
    print()  # sortörés minden sor után
```

Kimenet:

```
1  2  3  4  5
 2  4  6  8 10
 3  6  9 12 15
 4  8 12 16 20
 5 10 15 20 25
```

⚠ Tipp a beágyazott ciklusokhoz:

- Mindig **más változónevet** használj (pl. i, j, k), hogy ne keveredjenek össze!
- Ügyelj a **beljebb húzásra** (indentálásra), mert a Python ebben érzékeny!

🗂 Összefoglalás:

Fogalom	Jelentés
Beágyazott ciklus	Egy ciklus egy másik ciklus **testében fut**
for ciklus a for-ban	Gyakori például mátrixok vagy táblázatok kezelésénél
while a while-ban	Kevésbé gyakori, de logikai ismétlődéseknél hasznos
Kevert ciklusok	for a while-ban vagy fordítva – teljesen érvényes

listák

A **lista (list)** az egyik leggyakrabban használt **adatszerkezet Pythonban**, amely egy **több elemet tartalmazó, rendezett és módosítható** gyűjtemény.

📦 Lista létrehozása

python
lista = [1, 2, 3, 4]
szavak = ["alma", "banán", "körte"]
ures = [] # üres lista

📍 Lista tulajdonságai

Tulajdonság	Jelentés
Rendezett	Az elemeknek meghatározott sorrendje van
Indexelhető	lista[0] az első elem
Módosítható	Elemek felülírhatók
Többféle típusú elem	Lehet benne szám, string, más lista is

🔢 Elem elérése indexeléssel

python
gyumolcsok = ["alma", "banán", "narancs"]
print(gyumolcsok[0]) # alma
print(gyumolcsok[-1]) # narancs (utolsó elem)

🔄 Lista bejárása ciklussal

python
for gyumolcs in gyumolcsok:
    print(gyumolcs)

🔧 Fontos lista műveletek

Művelet	Példa	Eredmény
Hossz lekérdezése	len(lista)	Lista hossza (elemszám)
Elem hozzáadása	lista.append(új_elem)	Hozzáfűzi a végére
Elem beszúrása	lista.insert(index, érték)	Beszúrja az adott helyre
Elem eltávolítása	lista.remove(érték)	Törli az első ilyen értéket
Index alapján törlés	del lista[index] vagy pop(index)	Törli az adott indexű elemet
Lista kiürítése	lista.clear()	Eltávolít minden elemet

Művelet	Példa	Eredmény
Lista másolása	uj = lista.copy()	Új lista az eredeti másolata
Lista összefűzése	lista1 + lista2	Két lista egyesítése
Rendezés	lista.sort()	Növekvő sorrendbe rendezi
Fordított sorrend	lista.reverse()	Megfordítja a sorrendet

□ Példák

python
```
nevek = ["Anna", "Béla", "Cili"]
nevek.append("Dénes")        # hozzáadás
nevek.remove("Béla")         # törlés
print(len(nevek))            # hossz
print("Cili" in nevek)       # True
```

🔁 Lista generálása – list comprehension

python
```
negyzetek = [x**2 for x in range(1, 6)]
print(negyzetek)  # [1, 4, 9, 16, 25]
```

🧠 Hasznos függvények listákhoz

Függvény	Leírás
sum()	Elemszámok összege (ha számok)
min()	Legkisebb érték
max()	Legnagyobb érték
sorted()	Rendezett lista (új példány)

🧾 Összefoglalás

A lista egy **rugalmas és sokoldalú adattípus**, amely:

- **Többféle típust** is tartalmazhat (akár más listákat is!)
- Könnyen **módosítható**
- Könnyen **bejárható ciklusokkal**

Listák és sztringek

A **listák** és a **sztringek (szövegek)** Pythonban nagyon sok szempontból **hasonlóak**, de fontos különbségek is vannak köztük.

□ Közös jellemzők a listák és sztringek között

Jellemző	Lista példája	Sztring példája
Indexelhető	lista[0]	szoveg[0]
Szeletelhető (slicing)	lista[1:3]	szoveg[1:3]
Bejárható ciklussal	for elem in lista:	for betu in szoveg:
Hossz lekérhető	len(lista)	len(szoveg)

🔍 Példa:

python
```
lista = [1, 2, 3, 4]
szoveg = "alma"

print(lista[1])   # 2
print(szoveg[1])  # 'l'

print(lista[1:3]) # [2, 3]
print(szoveg[1:3])# 'lm'
```

□ Fontos különbségek

Tulajdonság	Lista	Sztring
Módosítható	☑ Igen (mutable)	❌ Nem (immutable)
Elemek típusa	Bármi lehet (pl. int, str…)	Csak karakterek (str)
Módosítás közvetlenül	lista[0] = 99	szoveg[0] = 'b' → ❌ Hiba
Hozzáfűzés	lista.append("új")	szoveg + " új" (új sztringet ad)

□ Példák módosításra

Lista – módosítható:

python
```
lista = [10, 20, 30]
lista[1] = 99
print(lista)  # [10, 99, 30]
```

Sztring – nem módosítható:

python

```
szoveg = "kacsa"
# szoveg[0] = "b"  ❌ HIBA: TypeError
szoveg = "b" + szoveg[1:]  # Trükkös módosítás
print(szoveg)  # "bacsa"
```

🔄 Átalakítás lista ↔ sztring között

Sztring karaktereinek listává alakítása:

python

```
szo = "hello"
karakterek = list(szo)
print(karakterek)  # ['h', 'e', 'l', 'l', 'o']
```

Lista sztringgé alakítása:

python

MásolásSzerkesztés

```
betuk = ['h', 'e', 'l', 'l', 'o']
szo = "".join(betuk)
print(szo)  # "hello"
```

🎯 Gyakorlati különbség – miért fontos?

- Ha **sok módosítást** végzünk karaktereken: **jobb listát használni**, mert hatékonyabb.
- Ha csak **olvasni** akarjuk a szöveget, elég a sztring is.

🧾 Összefoglalás

Típus	Módosítható	Indexelhető	Szeletelhető	Típusai
Lista	☑ Igen	☑ Igen	☑ Igen	Bármi lehet benne
Sztring	❌ Nem	☑ Igen	☑ Igen	Csak karakterek (str)