String

2025

1 String adattípus

A string szöveges adatok tárolására szolgáló adattípus, amely karakterek sorozata. A karakterlánc létrehozás után már nem megváltoztatható (immutable).
A Python string típusa unicode karaktereket használ, így minden magyar karaktert helyesen tárol és meg tud jeleníteni: `"Árvíztűrő tükörfúrógép"`

1.1 Stringek létrehozása

Pythonban a szöveg többféleképen is megadható. Az egy soros szövegeket vagy szimpla `''`, vagy dupla `""` idézőjelek közé írhatjuk.
`'ez egy szöveg'` és `"ez is egy szöveg"`, mindkettő forma használható.
Többsoros szöveg esetén háromszoros (`'''` vagy `"""`) idézőjeleket írunk. `'''Ez egy többsoros szöveg.'''`

A különbség akkor jelentkezik a két alak között, amikor a szövegen belül egyszeres vagy dupla idézőjelet szeretnénk használni: - `'ez 'egy' szöveg'` helytelen lesz, helyette `'ez \'egy\' szöveg'` vagy `"ez 'egy' szöveg"` - `"ez is "egy" szöveg"` helytelen lesz, helyette `"ez is \"egy\" szöveg"` vagy `'ez is "egy" szöveg'`

[ ]:
```
# Írja ki, hogy: ez 'egy' szöveg
print('ez \'egy\' szöveg')
print("ez 'egy' szöveg")
# írja ki, hogy: ez is "egy" szöveg
print("ez is \"egy\" szöveg")
print('ez is "egy" szöveg')
```

1.2 Escape karakterek

Az escape karakterek (feloldójel) olyan karakterek, amelyek speciális jelentéssel bírnak, és amelyeket a szövegfeldolgozó rendszerek különféle módokon értelmeznek. Ezek a karakterek backslash (\) karakterrel kezdődnek, és utána egy másik karakter következik, ami meghatározza az escape karakter jelentését. - soremelés (new line): `\n` - kocsi vissza (carriage return): `\r` - tabulálás: `\t` - idézőjelek (szövegtípustól függően szükséges): `\'` vagy `\"` - backslash karakter: `\\` - backspace karakter: `\b` - karakter hexadecimális értéke: `\xhh` pl.: ± értéke: `\xb1` - unicode karakter értéke: `\uhhhh` pl.: értéke: `\u2764`

1.3 r-string

A Pythonban a karakterláncok előtt álló `r` vagy `R` prefix (raw-string) segítségével, ki lehet kapcsolni az escape karakterek speciális értelmezését, így megjeleníti őket, mint karaktereket.
Az r-string gyakran hasznos például reguláris kifejezések vagy fájl elérési útvonalak esetén, ahol a backslash karakterek speciális jelentéssel bírnak. `dokumentumok = r"C:\Users\Username\Documents"`

1.4 f-string

Az f-string (formázott string) a Python egyik formázási módszere, amely lehetővé teszi a változók beágyazását a stringekbe a kifejezések használatával. Az f-stringek a Python 3.6.0 verziójától kezdve érhetők el.

Az f-stringek rendkívül kényelmesek és olvashatóak, és lehetőséget biztosítanak kapcsos zárójelek (`{}`) közé helyezve a változók, kifejezések, metódusok beágyazására a stringekbe.

Formázási lehetőségek: https://cheatography.com/brianallan/cheat-sheets/python-f-strings-number-formatting/

:	kitöltés	irány	előjel	#	0	szélesség	elválasztó	.pontosság	típus
	karakter	<	+			szám	_	.szám	szöveg: s
		>	-				,		szám: n
		^	' '						int: d, b, o, x, X, c
		=							float: e, E, f, F, g, G, %

[ ]:
```python
nev = "Python"
kor = 31
szoveg = f"Név: {nev}, kor: {kor}"

#formázási lehetőségek
print(f"Név: {nev}, kor: {kor}")
print(f"Név: {nev:30}, kor: {kor}")
print(f"Név: {nev:030}, kor: {kor}")
print(f"Név: {nev:>30}, kor: {kor}")
print(f"Név: {nev:*^30}, kor: {kor:#x}")
print(f"Név: {nev:-^30}, kor: {kor:.2f}")
```

1.5 Stringekkel végezhető műveletek

1.5.1 String hossza

A `len()` függvénnyel lekérdezhető a string hossza.

[ ]:
```python
szoveg = "Ez egy példa szöveg."

# a szöveg hossza
hossz = len(szoveg)
```

```
print(hossz)
```

1.5.2 Indexelés és szeletelés

A string karaktereit indexek `[]` segítségével érhetjük el. A stringek indexei nullától kezdődnek. A szeletelés segítségével részleteket vághatunk ki a stringből (lásd lista szeletelése rész). A karakterlánc elemei nem változtathatók meg, ezért az értékadás művelet nem használható.

[ ]:
```
szoveg = "Hello, Python!"

# a 8. betű
karakter = szoveg[7]  # 'P'
# az első 5 betű
szelet = szoveg[0:5]  # 'Hello'
# szöveg betűinek megfordítása
visszafele = szoveg[::-1] # '!nohtyP ,olleH'

# nem módosítható, ezért nem értelmezett:
# szoveg[0] = 'B'
```

1.5.3 String összefűzés

Stringeket össze lehet fűzni a `+` vagy a `+=` operátorral.

[ ]:
```
vezetek_nev = "Kiss"
kereszt_nev = "János"

# nevek összefűzése
teljes_nev = vezetek_nev + " " + kereszt_nev
print(teljes_nev)

print("1"+"2") # 12
```

1.5.4 Átalakítás

Kisbetűre `lower()`, nagybetűre `upper()`, elsőt nagybetűre `capitalize()`, szavankénti nagybetűre `title()` vagy kis és nagybetűk felcserélésével `swapcase()` átalakíthatjuk a szöveget egy új szövegre.

[ ]:
```
nev = "nemecsek Erno"
print(nev.lower())      # kisbetűs
print(nev.upper())      # nagybetűs
print(nev.capitalize()) # első karakter nagy a többi kicsi
print(nev.title())      # minden szó első karaktere nagy
print(nev.swapcase())   # felcseréli a kis és nagy betűket
```

1.5.5 Csere

Szövegrész vagy karakter cseréje `replace()`. Szövegrészek törlésére is használható, ha a kicserélendő szövegrészt egy üres szöveggel helyettesítjük.

[ ]:
```
szoveg = "Hello, World!"
print(szoveg.replace("H", "B"))  # Bello, World!
print(szoveg.replace("World", "Python")) # Hello, Python!
print(szoveg.replace(",", "")) # Hello World!
```

1.5.6 Szöveg darabolása

A `split()` függvény egy megadott szövegrész szerint feldarabolja a szöveget, a szövegdarabokat egy listában adja meg.

[ ]:
```
szoveg = "Hello, World!"
# darabolás "," szerint
print(szoveg.split(",")) # ['Hello', ' World!']
```

1.5.7 Keresés

Szövegrészlet keresésére két függvény áll rendelkezésre. A `find()`és az `index()` között az a különbség, hogyha nem található meg a kereset rész, akkor a `find` -1-et add vissza, az `index` viszont hibát dob. További paraméterként megadható a keresés kezdetének és végének pozíciója. Léteznek jobbról induló verzióik `rfind()`és `rindex()`.

[ ]:
```
szoveg = "Hello, World!"

# keressük a "World" szót
index = szoveg.find("World")  # 7
# keressük az "o" karaktert
index = szoveg.find("o")      # 4
# keressük az "o" karaktert, a 6. betűtől
index = szoveg.find("o", 5)   # 8
# keressük az "o" karaktert jobbról
index = szoveg.rfind("o")     # 8
# keressük a "?" karaktert
index = szoveg.find("?")      # -1

# keressük a "World" szó kezdő indexét
index = szoveg.index("World") # 7
print(index)
```

1.5.8 Tartalmazás vizsgálata

Az `in` operátorral lehet tesztelni, hogy egy szövegrész megtalálható-e egy másikban. Kis- és nagybetűk különbözőek.

[ ]:
```
szoveg = "almafa"

# a 'alma' szó része-e a szövegnek?
print("alma" in szoveg)
```

```
# az 'Almafajta' szövegben nem szerepel-e a szoveg
print(szoveg not in "Almafajta")                    # kis- és nagybetűk számítanak
print(szoveg.lower() not in "Almafajta".lower())  # kis- és nagybetűk így nem␣
 ↪számítanak

# az üres string mindennek a része
print("" in szoveg)
```

1.5.9 Megszámlálás

Egy keresett szövegrész előfordulásának a számát adja a `count()` függvény.

[ ]:
```
szoveg = "Hello, World!"

# mennyi "l"-t tartalmaz
db = szoveg.count("l")
print(db)
```

1.5.10 Csonkítás

A `strip()` függvény eltávolít minden kezdő és záró szóközt, vagy a megadott karaktereket. Léteznek csak a szöveg elejét `lstrip()` vagy végét `rstrip()` vizsgáló változatok.

[ ]:
```
szoveg = "     Hello, World!     "
print(szoveg.strip()) # Hello, World!
print(szoveg.rstrip(" !")) #       Hello, World
```

1.5.11 Listaelemek összefűzése

A `join()` függvényben megadott lista elemei közé bekerül a megadott szövegrész, és összefűzi egyetlen szöveggé.

[ ]:
```
nevek = ["János", "Tamás", "Ákos"]
print(nevek)

# Fűzzük össze a neveket egy szövegbe, vesszővel elválasztva
nevsor = ", ".join(nevek)
print(nevsor)
```

1.5.12 Ismétlés

A szövegek ismétlésére használható a `*` műveleti jel, amely a szöveg előtt és utáni is használható.

[ ]:
```
print(10*"3")  # 3333333333
print("10"*3)  # 101010
```

1.5.13 Megjelenítést módosítók

A szöveg mezőszélességét és azon belüli elhelyezkedését lehet beállítani a következő függvényekkel:

```
[ ]: szoveg = "Hello, World!"
     print(szoveg.center(50)) # "                    Hello, World!                    "
      ↪
     print(szoveg.rjust(50))  # "                                          Hello, World!"
     print(szoveg.ljust(50))  # "Hello, World!                                          "
```

1.5.14 Bejárás

A karakterlánc elemeit (karaktereit) a `for` ciklus segítségével be tudjuk járni.

```
[ ]: szoveg = "Ez egy szoveg"
     for betu in szoveg:
         print(betu)
```

1.5.15 Beolvasás

A felhasználó az `input()` függvény segítségével tud beolvasni egy szöveget a konzolról. A szöveg az Enter zárókarakter leütéséig lesz beolvasva.

```
[ ]: szoveg = input("Kérem adjon meg egy mondatot:")
     print("A megadott szöveg:", szoveg)
```

1.5.16 Összehasonlítás

Két szó összehasonlítására a használhatjuk a következő műveleti jeleket: ==, !=, >, <
A teljes egyezés akkor igaz, ha mindkét szó azonos hosszúságú és minden karakterük megegyezik. A kis és nagybetűk különbözőek, és a nagybetűk előbb helyezkednek el a sorrendben.

```
[ ]: szo1 = "alma"
     szo2 = "almafa"
     print(szo1 == "alma")
     print(szo1 < szo2)
     print(szo1 > szo2)
```

1.5.17 Karakterek vizsgálata

Számos karakterrel kapcsolatos ellenőrzési művelet áll rendelkezésre: - `isalpha()`: Ellenőrzi, hogy az összes karakter az angol ábécé betű-e. - `isdigit()`: Ellenőrzi, hogy az összes karakter számjegy-e. - `isalnum()`: Ellenőrzi, hogy az összes karakter betű vagy számjegy-e. - `islower()`, isupper(): Ellenőrzi, hogy az összes karakter kisbetű vagy nagybetű-e. - `isspace()`: Ellenőrzi, hogy az összes karakter szóköz, tabulátor vagy sorvége karakter-e.

```
[ ]: print("Hello".isalpha())
     print("123".isdigit())
     print("abc123".isalnum())
     print("hello".islower())
     print("HELLO".isupper())
     print(" \t\n".isspace())
```

2 Gyakorló feladatok

1. Feldat

   Készítsen egy programot, amely bekér egy mondatot, majd kiírja, hány karakterből áll az utolsó szó a mondatban.

[ ]:
```
mondat = "Ez egy hosszú mondat."
darabok = mondat.split(' ')
utolso = darabok[-1].strip(" -,.!?")
print(f"Az utolso szó: {utolso}, {len(utolso)} karakter hosszú")
```

2. Feladat

   Készítsen egy programot, amely bekér egy mondatot, majd kiszámolja, hány betűből áll a mondat (kizárva a szóközöket és az írásjeleket).

[ ]:
```
mondat = "Ez egy hosszú mondat."
karakterek = " -,.!?-"
csak_betuk = mondat
for c in karakterek:
    csak_betuk = csak_betuk.replace(c,"")

print(f"Csak betűk {len(csak_betuk)} száma")
```

3. Feladat

   Készítsen egy programot, amely bekér egy szót vagy mondatot, majd megszámolja, hány magánhangzó van benne.

[ ]:
```
mondat = "Ez egy hosszú mondat."
karakterek = "aáeéiíoóöőuúüű"

darab = 0
for c in karakterek:
    darab += mondat.lower().count(c)

print(f"Magánhangzók száma: {darab}")
```

4. Feladat

   Készítsen egy programot, amely bekér egy mondatot, majd megszámolja és kiírja, hány nagybetűs és hány kisbetűs karakter található benne.

[ ]:
```
mondat = "Ez egy hosszú mondat."
kisbetu = 0
nagybetu = 0

for c in mondat:
    if(c.islower()): kisbetu += 1
    if(c.isupper()): nagybetu += 1
```

```
print(f"Kisbetű: {kisbetu}")
print(f"Nagybetű: {nagybetu}")
```

5. Feladat

   Készítsen egy programot, amely bekér egy szót vagy mondatot, majd kicseréli az összes magánhangzót * karakterre.

[ ]:
```
mondat = "Ez egy hosszú mondat."
karakterek = "aáeéiíoóöőuúüű"

for c in karakterek:
    mondat = mondat.replace(c,"*").replace(c.upper(),"*")

print(mondat)
```

6. Feladat

   Készítsen egy programot, amely bekér egy mondatot, majd kiírja a mondatban lévő szavakat párosával felcserélt sorrendben.

[ ]:
```
mondat = "Ez egy hosszú mondat."
darabok = mondat.split(' ')

for i in range(0,len(darabok),2):
    darabok[i], darabok[i+1] = darabok[i+1], darabok[i]

print(" ".join(darabok))
```

7. Feladat

   Készítsen egy programot, amely bekér egy szót vagy mondatot, majd ellenőrzi, hogy a szó vagy mondat palindrom-e (azaz fordítva olvasva is ugyanaz-e). (A szóközök, a kis és nagy betűk nem számítanak)

[ ]:
```
mondat = "Indul a görög aludni"

szo = mondat.replace(" ","").lower()
palindrome = True
for i in range(len(szo)//2):
    if (szo[i] != szo[-(i+1)]):
        palindrome = False
        break

print(palindrome)
```

8. Feladat

   Készítsen egy programot, amely bekér két szót, majd megállapítja, hogy azok ugyanazok-e, vagy ha nem, akkor azt írja ki, amelyik előbbre van az ábc-ben. A kis és

nagy betűk nem számítanak.

[ ]:
```
szo1 = "asztal"
szo2 = "Asztal"

if (szo1.lower() == szo2.lower()):
    print("A két szó azonos")
elif (szo1.lower() < szo2.lower()):
    print(szo1, szo2)
else:
    print(szo2, szo1)
```

9. Feladat

   Készítsen egy programot, amely bekér egy szót vagy mondatot, majd kiírja minden második karakterét.

[ ]:
```
mondat = "Indul a görög aludni"

uj_mondat = ""
for i in range(0,len(mondat),2):
    uj_mondat += mondat[i]

print(uj_mondat)

# vagy
print(mondat[::2])
```

10. Feladat

    Készítsen egy programot, amely bekér egy mondatot, majd kiírja a mondatban szereplő szavak első betűit egyetlen szóban. (Mozaik szó)

    A kihagyható szavakat tegye bele egy listába és azokat ne vegye figyelembe.

[ ]:
```
nev = "Mechatronkia, Optika és Gépészeti Informatika Tanszék"

mozaik_szo = ""
kihagyandok = ["a", "az", "egy", "és", "Tanszék"]
for szo in nev.split(" "):
    if szo not in kihagyandok:
        mozaik_szo += szo[0]

print(mozaik_szo)
```

11. Feladat

    Készítsen egy programot, amely bekér egy szót, majd kiírja úgy, hogy a szó minden második karaktere nagybetűs legyen, a többi kisbetűs.

[ ]:
```python
nev = "Mechatronkia, Optika és Gépészeti Informatika Tanszék"

uj_nev = ""
for i,c in enumerate(nev):
    if i%2 == 0:
        uj_nev += c.lower()
    else:
        uj_nev += c.upper()

print(uj_nev)
```