HALMAZOK
set

készítette: Vastag Atila

2020

SET

A halmaz egyedi értékek rendezetlen „kupaca”. Egy halmaz tetszőleges megváltoztathatatlan adattípusú értékeket tartalmazhat. Ha van két halmazod, akkor végrehajthatsz rajtuk általános halmazműveleteket, mint az unió, metszet és különbség.

Halmaz létrehozása

Még nincsenek értékeid? Nem gond. Létrehozhatsz üres halmazt is.

```
halmaz = set()
print(halmaz)
# set()

halmazElemeinekSzama = len(halmaz)
print(halmazElemeinekSzama)
# 0
```

1. Üres halmaz létrehozásához hívd meg a **set()** függvényt argumentumok nélkül.
2. Az üres halmaz kiírt ábrázolása egy kicsit furcsán néz ki. Talán ezt vártad: {}? Az egy üres szótárat jelölne, nem pedig egy üres halmazt.
3. A furcsa kiírt ábrázolás ellenére ez egy halmaz…
4. …és ennek a halmaznak nincsenek elemei.
5. A Python 2-ből áthozott örökség miatt nem hozhatsz létre üres halmazt két kapcsos zárójellel. Az egy üres szótárat hozna létre, nem pedig egy üres halmazt.

Halmaz létrehozása

Kezdjük az elején. Egy halmazt létrehozni könnyű.

```
halmaz = {1, 2}
print(halmaz)
# {1, 2}
```

1. Egy egy értékkel rendelkező halmaz létrehozásához tedd az értéket kapcsos zárójelek ({}) közé.
2. A halmazok valójában osztályokként vannak megvalósítva, de emiatt most ne aggódj.
3. Egy több értékkel rendelkező halmaz létrehozásához vesszőkel válaszd el az értékeket, és tedd az egészet kapcsos zárójelek közé.

Halmazt egy listából is létrehozhatsz.

```
lista = ['a', 'b', 'mpilgrim', True, False, 42]
halmaz = set(lista)

print(halmaz)
# {'a', False, 'b', True, 'mpilgrim', 42}

print(lista)
# ['a', 'b', 'mpilgrim', True, False, 42]
```

1. Egy listából halmaz létrehozásához használd a **set()** függvényt. (Azok a pedánsok, akik ismerik a halmazok megvalósítását, rá fognak mutatni, hogy ez valójában nem függvényhívás, hanem egy osztály példányosítása.) **Figyeljük meg, hogy ha a listában több ugyaolyan érték szerepel, akokr a halmazban csak EGYSZER jelenik meg. Ezt ki tudjuk használni, akkor amikor elemet csak egszer van szükség!**
2. Ahogyan azt korábban említettem, egy halmaz tetszőleges adattípusú értékeket tartalmazhat. Ahogyan még korábban említettem, a halmazok rendezetlenek. Ez a halmaz nem emlékszik a létrehozásához használt lista eredeti sorrendjére. Ha elemeket adsz a halmazhoz, az nem fog emlékezni a hozzáadás sorrendjére.
3. Az eredeti lista változatlan.

Halmaz módosítása

Meglévő halmazhoz két különböző módon adhatsz értékeket: az **add()** és az **update()** metódusokkal.

```
halmaz = {1, 2}
print(halmaz)
# {1, 2}

halmaz.add(4)
print(halmaz)
# {1, 2, 4}

elemekSzama = len(halmaz)
print(elemekSzama)
# 3
halmaz.add(1)
print(halmaz)
# {1, 2, 4}

elemekSzama = len(halmaz)
print(elemekSzama)
# 3
```

1. Az **add()** metódus egyetlen argumentumot vár, amely tetszőleges adattípusú lehet, és az adott értéket hozzáadja a halmazhoz.
2. Ez a halmaz most 3 elemmel rendelkezik.
3. A halmazok egyedi értékek kupacai. Ha egy már létező értéket próbálsz hozzáadni, akkor nem történik semmi. Nem fog kivételt dobni, csak nem csinál semmit.
4. A halmaznak továbbra is 3 eleme van.

```
halmaz= {1, 2, 3 }
print(halmaz)
# {1, 2, 3}

halmaz.update({2, 4, 6})
print(halmaz)
# {1, 2, 3, 4, 6}

halmaz.update({3, 6, 9}, {1, 2, 3, 5, 8, 13})
print(halmaz)
# {1, 2, 3, 4, 5, 6, 8, 9, 13}

halmaz.update([10, 20, 30])
print(halmaz)
# {1, 2, 3, 4, 5, 6, 8, 9, 10, 13, 20, 30}
```

1. Az **update()** metódus egy argumentumot vár, egy halmazt, és minden elemét hozzáadja az eredeti halmazhoz. Olyan, mintha a halmaz minden egyes elemére meghívtad volna az **add()** metódust.
2. A többször szereplő értékek figyelmen kívül maradnak, mivel a halmazok egy értéket csak egyszer tartalmazhatnak.
3. Az **update()** metódust ténylegesen tetszőleges számú argumentummal hívhatod meg. Két halmazzal hívva az update() metódus az egyes halmazok összes elemét hozzáadja az eredeti halmazhoz (a többször szereplő értékek eldobásával).
4. Az **update()** metódus számos különböző adattípus objektumait képes kezelni, beleértve a listákat is. Egy listával hívva az **update()** metódus a lista összes elemét hozzáadja az eredeti halmazhoz

Elemek eltávolítása a halmazból

Három módon távolíthatsz el egyedi értékeket a halmazból. Az első kettő, a **discard()** és **remove()** között egy apró különbség van.
A listákhoz hasonlóan a halmazoknak is van **pop()** metódusuk.

1. A **discard()** metódus egyetlen értéket vár argumentumként, és eltávolítja azt az értéket a halmazból.
2. Ha a **discard()** metódust a halmazban nem létező értékkel hívod, akkor nem csinál semmi. Nem fog kivételt dobni, csak nem csinál semmit.
3. A **discard()** metódus is egyetlen értéket vár argumentumként, és szintén eltávolítja azt az értéket a halmazból.
4. Itt a különbség: ha az érték nem létezik a halmazban, akkor a **remove()** metódus egy *KeyError* kivételt dob.

1. A **pop()** metódus egyetlen értéket távolít el a halmazból, és visszaadja az értéket. Mivel azonban a halmazok rendezetlenek, nincs „utolsó” értékük, így nincs lehetőség az eltávolított érték befolyásolására. Teljesen tetszőleges.
2. A **clear()** metódus minden értéket eltávolít a halmazból, és egy üres halmazt hagy. Ez egyenértékű az ***halmaz = set()*** hívással, amely egy új üres halmazt hoz létre, és az ***halmaz*** változó korábbi értékét felülírja.
3. Üres halmazból nem lehet értéket kivenni a **pop()** metódussal, ez egy *KeyError* kivételt dob

```
halmaz = {1, 3, 6, 10, 15, 21, 28, 36, 45}
print(halmaz)
#{1, 3, 36, 6, 10, 45, 15, 21, 28}

halmaz.discard(10)
print(halmaz)
#{1, 3, 36, 6, 45, 15, 21, 28}

halmaz.discard(10)
print(halmaz)
#{1, 3, 36, 6, 45, 15, 21, 28}

halmaz.remove(21)
print(halmaz)
#{1, 3, 36, 6, 45, 15, 28}

halmaz.remove(21)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 21
```

```
halmaz = {1, 3, 6, 10, 15, 21, 28, 36, 45}
print(halmaz)
# {1, 3, 36, 6, 10, 45, 15, 21, 28}

halmaz.pop()
print(halmaz)
# 1

halmaz.pop()
print(halmaz)
# 3

print(halmaz)
# {36, 6, 45, 15, 21, 28}

halmaz.clear(21)
print(halmaz)
# set()

halmaz.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'pop from an empty set'
```

Általános halmazműveletek

- Annak teszteléséhez, hogy egy érték eleme-e egy halmaznak, használd az **in** operátort. Ez ugyanúgy működik, mint a listák esetén.

```
halmaz = { 2, 4, 5, 9, 12, 21, 30, 51, 76, 127, 195 }

exists: bool = 30 in halmaz
print(exists)
# True

exists: bool = -30 in halmaz
print(exists)
# False
```

Általános halmazműveletek

- Az **union()** metódus egy új halmazt ad vissza, amely tartalmazza az összes, valamelyik halmazban jelen lévő elemet.

halmazA = { 2, 4, 5, 9, 12, 21, 30, 51, 76, 127, 195 }
halmazB = {1, 2, 3, 5, 6, 8, 9, 12, 15, 17, 18, 21}

halmazA.**union**(halmazB)
print(halmazA**)**
{1, 2, 195, 4, 5, 6, 8, 12, 76, 15, 17, 18, 3, 21, 30, 51, 9, 127}

Általános halmazműveletek

- Az **intersection()** metódus egy új halmazt ad vissza, amely tartalmazza az összes, mindkét halmazban jelen lévő elemet.

halmazA = { 2, 4, 5, 9, 12, 21, 30, 51, 76, 127, 195 }
halmazB = {1, 2, 3, 5, 6, 8, 9, 12, 15, 17, 18, 21}

halmazA.**intersection**(halmazB)
print(halmazA**)**
{9, 2, 12, 5, 21}

Általános halmazműveletek

- A **difference()** metódus egy új halmazt ad vissza, amely tartalmazza az a_set halmazban jelen lévő, de a b_set halmazban jelen nem lévő elemet.

halmazA = { 2, 4, 5, 9, 12, 21, 30, 51, 76, 127, 195 }
halmazB = {1, 2, 3, 5, 6, 8, 9, 12, 15, 17, 18, 21}

halmazA.**difference**(halmazB)
print(halmazA**)**
{195, 4, 76, 51, 30, 127}

- A **symmetric_difference()** metódus egy új halmazt ad vissza, amely tartalmazza az összes, pontosan egy halmazban jelen lévő elemet.

halmazA = { 2, 4, 5, 9, 12, 21, 30, 51, 76, 127, 195 }
halmazB = {1, 2, 3, 5, 6, 8, 9, 12, 15, 17, 18, 21}

halmazA.**symmetric_difference**(halmazB)
print(halmazA**)**
{1, 3, 4, 6, 8, 76, 15, 17, 18, 195, 127, 30, 51}

Általános halmazműveletek

- A halmazA a halmazB részhalmaza – az halmazA minden eleme a halmazB-nek is eleme.

halmazA = { 1, 2, 3}
halmazB = { 1, 2, 3, 4}

isSubSet = halmazA.**issubset**(halmazB)
print(isSubSet**)**
True

- A kérdést megfordítva a halmazB az halmazA szülőhalmaza, mert az halmazA elemei a halmazB-nek is elemei.

```
halmazA = { 1, 2, 3}
halmazB = { 1, 2, 3, 4}

isSuperSet = halmazA.issuperset(halmazB)
print(isSuperSet)
# True
```