NOTIȚE CURS 5

Tabela de dispersie

s = "test" → s.__hash__() = un cod
→ hash(s) = același cod
] în cadrul programului este ac., la alte rulări poate fi diferit

obiect1 == obiect2 ⇒ hash-urile sunt obligatoriu egale
obiect1 != obiect2 ⇒ se poate întâmpla să aibă ac. hash-uri din cauza coliziunii (principiul lui Dirichlet)

hash = x % 8 , 16, 11, 27, 22, 19, 6

0 → [16]
1
2
3 → [11] → [27] → [19]
4
5
6 → [22] → [6]
7

MULȚIMI → mutabilă, fără duplicate, pot fi val. neomogene, nu au index
sunt implementate folosind tabele de dispersie

1. **creare** → cu s = set() / scriem noi /
s = { scriem noi }
s = {x for x in range(val.)}
↳ sau altă formă de scriere

2. **accesare** → doar prin parcurgere
s = { 1, 2, 3, 2, 2, 4, 1, 1, 7 }
for x in s:
print(x) → 1 2 3 4 7

3. **operatori** → in, not in = O(1)! / O(n)
↳ dacă un elem. aparține mulțimii
<, <=, >, >=, ==, != ↳ din p.d.v. MATEMATIC, adică s < t ⇒ s ⊂ t
↳ se vor compara valorile
| (reuniunea), & (intersecția), - (diferență), ^ (diferența simetrică)

6. funcții

len(m) → nr. elem.

set(secv) → creează o mulțime

min / max

sum(m) → suma, toate elem trebuie să fie numerice

sorted(m) → sortare implicit crescător

...

7. metode

S.add(val) → dacă val. nu ∃ deja, va fi adăugată

S.update(secv) → adaugă toate elem din secv.

S.remove(val) → dacă ∃ o va șterge, dacă nu lansează o eroare
se rez. cu try: except

S.discard(val) → -"- fără eroare

S.clear → șterge complet

S.union(secv)
S.intersection
S.difference
S.symmetric_difference
} → furnizează mulțime ost. fără a modifica S

-"- cu _update (ex. S.union_update(secv) va modifica direct în S

DICȚIONARE → tablou asociativ de tipul cheie : valoare, cheile sunt „indecșii" și sunt imutabile

1. creare: → d = { 6 : -100, "Popa Ion" : (13.1, 9.50), (1,2,3) : [1,2,3] }

d["A"] = 65

d = {x : chr(x) for x in range(ord('A'), ord('Z')+1)}

d = dict([(65,'A'), (66,'B')])
↓ listă cu tupluri

2. accesare: → d['A'] = 65

del d['A']

dacă încercăm să accesăm o cheie inexistentă, va fi lansată o eroare (o vom rez. cu try: except)

3. operatori

in, not in → pt. testarea apartenenței = O(1) ! (preferabil) maxim O(n)
se caută CHEIA

==, != → se verifică dacă cele 2 dicționare au ac. perechi de tip cheie: valoare

4. funcții

- len(d) → numărul total de chei
- dict (secv)
- min / max
- sum → suma cheilor care trebuie să fie de tip numeric
- sorted → furnizează o listă formată din cheile sortate desc.

5. metode

- d.get (cheie) → afișază / returnează val. cheii sau None în caz contrar
- d.keys()
- d.values()
- d.items()

} → val = una din ele → va furniza o listă cu chei / val. / tupluri cu items

- d.update(secv) → adaugă cheile: valori în d
- d.pop (cheie) → șterge perechea și furnizează valoarea
- d.clear()

Spre exemplu, dicționarele sunt foarte folositoare pt. a afla frecvența cuvintelor într-o propoziție

```
f= open ("text.txt")
prop = f.read()
f.close
```
] deschiderea și citirea din fișier

```
#înlocuim semnele de punctuație din propoz. pt. a nu fi preluate
for sep in ".,:;?!":
    prop = prop.replace(sep, " ")
cuvinte = prop.split()  # creez o listă cu toate cuv din propoz.
# creez un dicționar unde cheile sunt cuv din propoz și valorile vor fi
  inițializate cu 0 - frecvența inițială
~~for cuv in cuvinte~~: frcuv = { cuv: 0 for cuv in set(cuvinte) }
    ~~frcuv [~~
```
set(cuvinte) → o fac mulțime pt. a nu se repeta valorile

```
# calculăm frecvența fiecărui cuv
    for cuv in cuvinte:  #parcurgem lista unde avem toate cuv și de mai multe
                         ori
# afișăm frecvența cuvintelor distincte
    for cuv in fcuv:
        print(f"cuvântul {cuv} apare de {fcuv[cuv]} ori")
```