

# Alte colecții

# Hash code – Funcții de dispersie

# Funcții de dispersie

- ▶ Obiect *hash-uibil* = care are cod **hash** asociat, returnat de metoda `__hash__()` (= cod de dispersie, valoare hash)

= un număr întreg cu proprietățile

# Funcții de dispersie

- ▶ Obiect *hash-uibil* = care are cod **hash** asociat, returnat de metoda `__hash__()` (= cod de dispersie, valoare hash)

= un număr întreg cu proprietățile

- nu se poate schimba dacă obiectul nu se modifică (de obicei imutabil)
- două obiecte **egale au același hash code**
- este recomandabil ca două obiecte diferite să aibă valori hash diferite (compatibil cu `__eq__()`)

# Funcții de dispersie

- ▶ Obiect *hash-uibil* = care are cod **hash** asociat, returnat de metoda `__hash__()` (= cod de dispersie, valoare hash)

= un număr întreg cu proprietățile

- nu se poate schimba dacă obiectul nu se modifică (de obicei imutabil)
- două obiecte **egale au același hash code**
- este recomandabil ca două obiecte diferite să aibă valori hash diferite (compatibil cu `__eq__()`)

# Funcții de dispersie

**Obiect**



**Număr întreg**

dimensiune variabilă

plajă limitată de valori



apar coliziuni

# Funcții de dispersie

```
t = (1,2)
```

```
print(hash(t)) #print(t.__hash__())
```

```
t = (1,[2,3])
```

```
print(hash(t)) # print(t.__hash__())
```

```
#TypeError: unhashable type: 'list'
```



# Funcții de dispersie

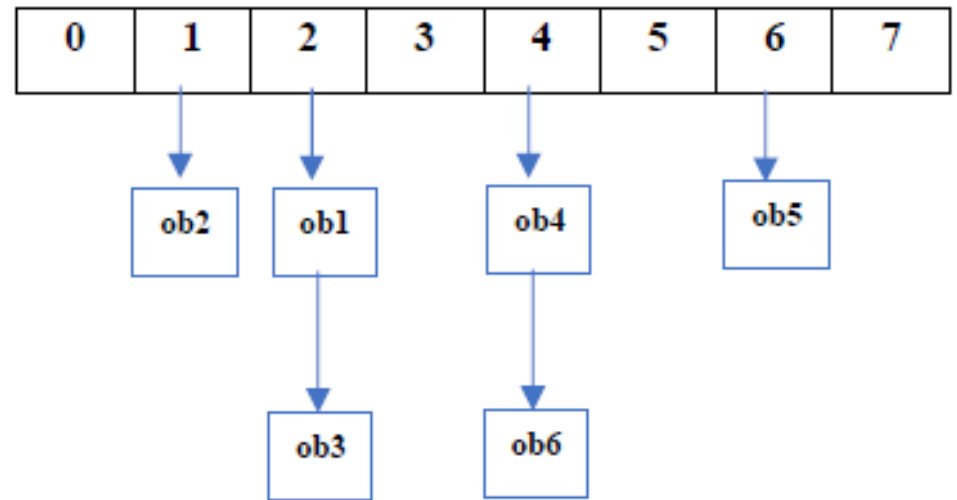
- ▶ Funcția hash (de dispersie) trebuie aleasă astfel încât să se **minimizeze** numărul coliziunilor (obiecte diferite care produc aceleași hash-uri).
- ▶ Obiectele hash-uibile pot fi folosite ca chei pentru structuri de date indexate după chei (se va folosi pentru indexare codul lor hash)



# Tabele de dispersie

3 biți pentru memorarea codului hash

Obiect	Hash code (modulo numărul de bucketuri)
ob1 = "a"	2
ob2 = "bc"	1
ob3 = "ad"	2
ob4 = "m"	4
ob5 = "casa"	6
ob6 = "aab"	4



# Tabele de dispersie

- ▶ structură de date pentru căutare eficientă după chei hash-uibile
- ▶ căutare – în **medie** timpul  $O(1)$  (defavorabil  $O(n)$ )

Căutarea unui obiect în tabel se face, în mare, astfel:

- se determină  $c = \text{indexul asociat codul hash al obiectului (modulo numărul de bucketuri)}$ , se accesează bucketul  $c$
- se caută în acest bucket obiectul (folosind pentru testare metoda `__eq__()` ).

Muṭimi

# Mulțimi

## Clasa set

- ▶ Elemente unice (mulțime)
- ▶ **Nu sunt indexate** (nu `s[0]`)
- ▶ Neordonată, **nu păstrează ordinea de inserare a elementelor**
- ▶ **mutabilă**
- ▶ alcătuită din elemente “*imutabile*”, de fapt *hash-uibile*

# Mulțimi

## Creare

```
s = {7, 5, 13}
```

```
s = {1, 2, 3, 2, 1}
```

```
s = {} #NU, este dictionar
```

```
s = {(1,2), (2,1), (3,1)} #OK
```

```
s = {[1,2], 3} #NU
```

```
#TypeError: unhashable type: 'list'
```

# Muṭimi

## Creare

- ▶ `set([iterabil])`
  - `s = set() #multimea vida`
  - `s = set("multime")`
  - `s = set(["multime"])`
  - `s = set(("multime"))`

# Mulțimi

## Creare

```
ls = [2, 3, 1, 3, 2, 6]
```

```
s = set(ls) #elementele distincte
```

```
cuv = "aceeasi"
```

```
s = set(cuv)
```



# Mulțimi

## Creare

- `set()` - mulțimea vida
- **!!! `s = {}` nu este multimea vida**



# Multi

## Creare

- Comprehensiune

```
s = {x for x in range(1,5)}
```

# Mulțimi

## Creare

- Comprehensiune

```
s = {x for x in range(1,5)}
```

```
ls = [2, 3, 1, 3, -2, 6, 2]
```

```
s = {x for x in ls if x>0}
```

```
#elementele distincte pozitive din ls
```



# Mulțimi

## Accesare

- ▶ **nu sunt indexate** => **NU** `s[0]`, `feliere` etc
- ▶ **parcurgere element cu element**

```
for x in s:  
    print(x)
```

# Muṭimi

## Operatori

- ▶ `in`, `not in`

- ▶ `==`, `!=`

`{1,2,3} == {3,2,1}`

# Mulțimi

## Operatori

- ▶  $<$ ,  $\leq$ ,  $>$ ,  $\geq$  testează **incluziunea**

$$\{1, 2\} < \{1, 2, 4\}$$

$$\{2, 4\} > \{2, 3\}$$

# Mulțimi

## Operatori

### ► | reuniune

```
s1 = {5,7,10}; s2 = {5,10,15,20}; s3 = {8}
```

```
s = s1 | s2 | s3
```

# Mulțimi

## Operatori

- ▶ | reuniune

$s1 = \{5, 7, 10\}; s2 = \{5, 10, 15, 20\}; s3 = \{8\}$

$s = s1 \mid s2 \mid s3$

- ▶ & intersecție
- ▶ – diferența
- ▶  $\wedge$  diferența simetrică

# Mulțimi

**Pentru incluziune și operații cu mulțimi – există și metode**

- diferența: metodele pot primi un iterabil, operatorii doar set**



# Mulțimi

Pentru incluziune și operații cu mulțimi – există și metode care **returnează o nouă mulțime**

- diferența: metodele pot primi un iterabil, operatorii doar set
- `issubset`, `issuperset`
- `union`, `difference`, `intersection`, `symmetric_difference` ...

# Mulțimi

Pentru incluziune și operații cu mulțimi – există și metode care **modifică mulțimea**

- `update`, `difference_update`,  
`intersection_update`,  
`symmetric_difference_update` ...

# Muṭimi

```
s1 = {5,7,10}
```

```
s2 = {5,10,15,20}
```

```
s3 = {8}
```

```
s4 = [5,6] #s4="ab"
```

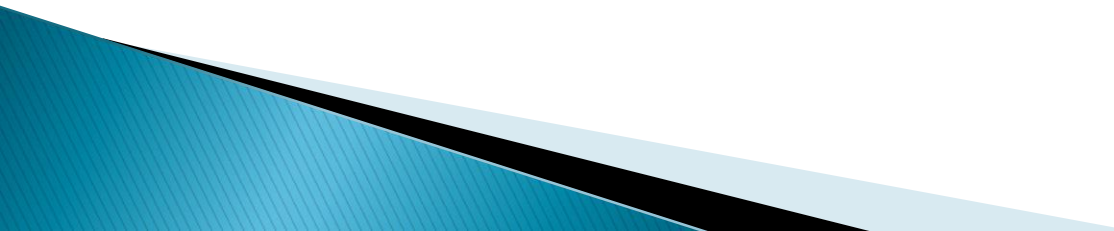
```
s = s1.union(s2,s3,s4) #s1.update(s2,s3,s4)
```

```
print(s1)
```

```
print(s)
```

```
#s=s1|s2|s3|s4
```

```
s1.intersection_update(s2)
```



# Mulțimi

## Metode – modificare

- `add(element)` – adăugare element
- `update(iterabil_1[, ..., iterabil_n])`  
–adauga elementele iterabililor primiți ca parametru
- `remove(element)` – eliminare element,  
eroare dacă elementul nu este în mulțime
- `discard (element)` – eliminare element

# Muṭimi

```
s = {5,7,10}
```

```
s.add(11)
```

```
s.update({1,2})
```

```
s.update([3,1,10], "ab")
```

```
print(s)
```

```
s.remove(7)
```

```
s.discard(15) #s.remove(15)
```

```
print(s)
```



# Mulțimi imutabile

# Mulțimi imutabile

## Clasa frozenset

- ▶ Aceleași metode ca la set, mai puțin cele care modifică mulțimea
- ▶ Creare folosind constructorul:

```
s1 = frozenset(iterabil)
```

```
svid = frozenset()
```



# Mulțime imutabilă

## ► Clasa frozenset

```
s1 = frozenset([3,5,4,5])
```

```
s2 = {4,6}
```

```
s = s1 | s2
```

```
print(s,type(s)) #frozenset
```

```
s = s1.union(s2)
```

```
print(s,type(s)) #frozenset
```

```
s = s2 | s1
```

```
print(s,type(s)) #set
```



# Dicționare

<https://docs.python.org/3.9/library/stdtypes.html#dict>

# Dicționare

- Un **dicționar** – colecție de **perechi cheie și valoare** (fiecărei chei îi este asociată o valoare)
- Cheia – este unică și **imutabilă** + hash-uibilă => toate componentele cheii trebuie să fie imutabile
- ▶ Dicționarele sunt **mutabile**

# Dictionare

```
t = (1,2) #poate fi cheie in dictionar
```

```
t = (1,[2,3]) #nu poate fi cheie in dictionar
```



# Dicționare

- Indexate după cheie, nu după index:

**`d[cheie] = valoarea asociata`**

- Implementare internă – căutare eficientă (medie  $O(1)$ ) după cheie (după codul hash asociat cheii)  
=> v. tabele de dispersie

# Dicționare

- Valoarea asociată cheii – orice tip
- Cheile pot avea tipuri diferite

# Dictionare

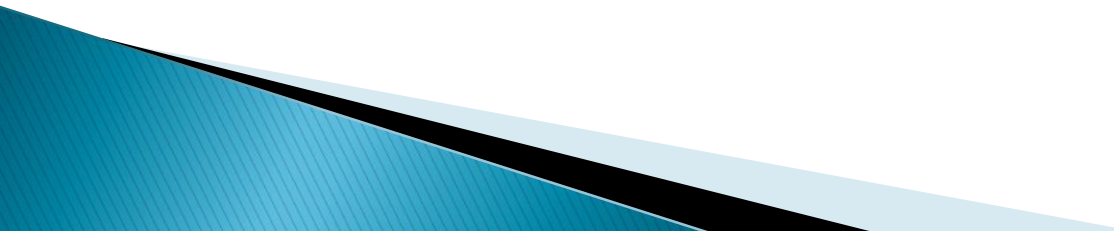
## Creare

```
d = {} #vid
```

```
d = {"a":2, "b":3}
```

```
#valorile - pot avea tip diferit
```

```
d = {"a":2, "b":3, "-":"semn"}
```



# Dicționare

## Creare

```
#valorile pot fi mutabile
```

```
d = {1:[2,3], 2:[1], 3:[1]}
```

# Dictionare

## Creare

```
#cheile trebuie sa fie hash-uibile
```

```
d = { (1,2): "tuplu", 3: "numar",  
      frozenset({2,4}): "frozen set" }
```

```
# d = { (1,2): "tuplu", 3: "numar",  
      {2,4}: "set" } #NU
```



# Dictionare

## Creare

► `dict(secventa_de_perechi_chei_valoare)`

```
d = dict([("a",1) , ("b",2)])
```

```
print(d)
```

```
d = dict(("a",1) , ("b",2))
```

```
print(d)
```



# Dictionare

## Creare

▶ `dict.fromkeys(iterabil_chei[,valoare_default])`

=> Toate valorile asociate cheilor sunt None sau  
valoare\_default (dacă aceasta se specifică)

```
d = dict.fromkeys("aeiou",0)
```

```
print(d)
```



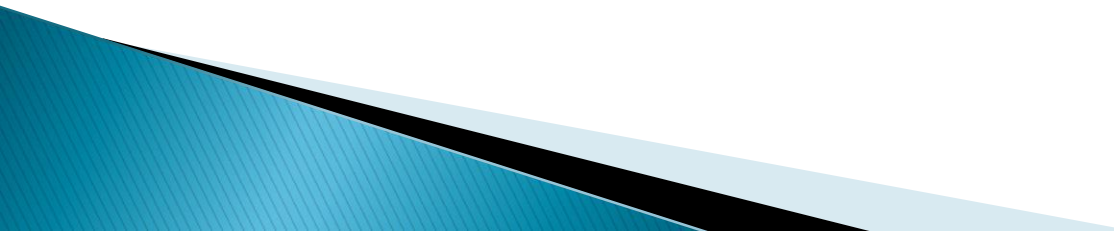
# Dictionary

## Create

- Comprehensiune

```
d = {x:0 for x in "aeiou"} # dict.fromkeys  
print(d["e"])
```

```
d = {x:chr(ord(x)+1) for x in "aeiou"}  
print(d["e"])
```



# Dicționare

## Accesare elemente

- `d[cheie]` => **eroare** dacă nu există cheia
- `d.get(cheie[,valoare_default])`  
=> returnează valoarea asociată cheii; dacă cheia nu există returnează **None** sau, dacă este specificată, `valoare_default` dată ca al doilea parametru
- `d.setdefault` – discutată la actualizare

# Dictionare

```
d = {"a":2, "b":3}

print(d["a"])

# print(d["c"]) #KeyError: 'c'

print(d.get("c")) #None

print(d.get("c",0)) #0
```

# Dicționare

## Actualizare

- `d[cheie] = valoare` – inserează dacă nu există cheia
- `d.setdefault(cheie[, valoare]) =>` inserează în dicționar cheia dată dacă nu există, cu valoarea `None` sau cea specificată la al doilea parametru și **returnează valoarea cheii (existentă sau tocmai inserată)**

# Dictionare

```
d = {"a":2, "b":3 }
```

```
x = d.setdefault("c",0)
```

```
print(x,d)
```



# Dictionare

```
d = {"a":2, "b":3 }
```

```
x = d.setdefault("c",0)
```

```
print(x,d)
```

```
x=d.setdefault("c",10) #exista, nu se actualizeaza
```

```
print(x,d)
```





# Dictionare

```
d = {"a":2, "b":3 }
```

```
x = d.setdefault("c",0)
```

```
print(x,d)
```

```
x=d.setdefault("c",10) #exista, nu se actualizeaza
```

```
print(x,d)
```

```
d["c"] = 7 #se actualizeaza
```

```
print(d)
```

```
d["e"] = 8 #se insereaza
```



# Dicționare

## Actualizare

- `d.update(dictionar)`
- `d.update(iterabil cheie-valoare)` =>  
reuniune de dicționare, cu actualizarea cheilor care există deja

```
d = {"a":2, "b":3 }
```

```
d.update({"b":4, "altceva":0})
```

```
d.update([("f",5), ("g",4)])
```

```
print(d)
```

# Dictionare

## Actualizare

- `d.pop(cheie[,valoare]) =>` elimină cheia (cu valoarea asociată) și returnează valoarea asociată; dacă cheia nu există dă **eroare** sau returnează **valoarea furnizată la al doilea parametru**
- `del d[cheie]`
- `d.clear()`

# Dictionare

```
d = {"a":2, "b":3 , "c":4}
```

```
#d.pop("A") #eroare KeyError: 'A'
```

```
x=d.pop("a")
```

```
print(x)
```

```
print(d)
```

```
x=d.pop("A", 0)
```

```
print(x)
```

```
print(d)
```

```
del d["b"] #ca si pop
```



# Dicționare

## Accesare + Parcurgere

- `d.keys()` => chei (un view care se updateaza automat odata cu schimbarea dictionarului *d*)
- `d.values()` => valori
- `d.items()` => tupluri (cheie, valoare)

# Dictionary

```
d = {"a":2, "b":3, "-":7}

print(d.keys(), type(d.keys())) #tip dict_keys
print(d.values(), type(d.values()))
print(d.items(), type(d.items()))

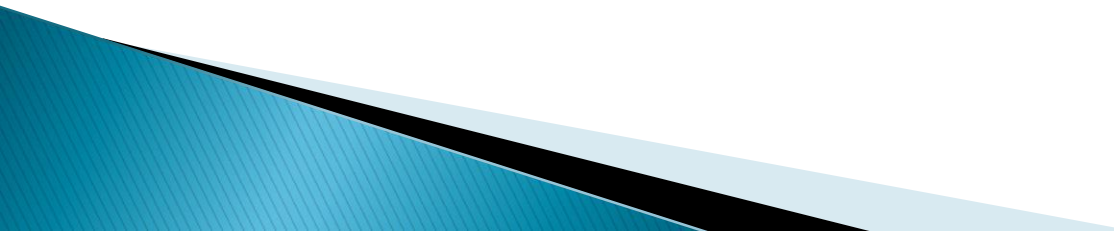
ls_key=d.keys()

print(ls_key)

d.update({"c":5, "e":2, "a":4})

print(d)

print(ls_key)
```



# Dictionare

```
d = {"a":2, "b":3, "-":7}
```

```
for x in d: #dupa cheie
```

```
    print(x, d[x])
```

```
for p in d.items():
```

```
    print(p, type(p))
```

```
for p in d.values():
```

```
    print(p, type(p))
```



# Dictionare

```
#ce face secventa de cod?
```

```
d = {"a":2, "b":3, "c":7}
```

```
d1 = {x:d[x] for x in d.keys()-{"b"}}
```

```
print(d1)
```



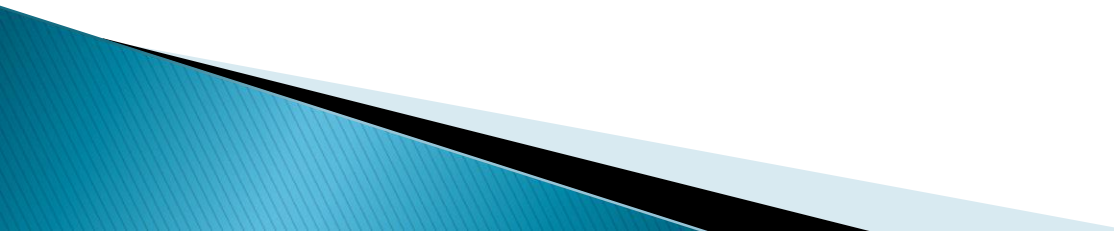


# Dicționare

## Operatori

- `in`, `not in` (după cheie)
- `==` , `!=`

## Metode comune

- `max`, `min` – pentru cheie
  - `len`
- 

# Dictionare

```
d1 = {"a":2, "b":3 , "c":4}
```

```
d2 = {"b":3 , "a":2, "c":4}
```

```
print(d1 == d2)
```

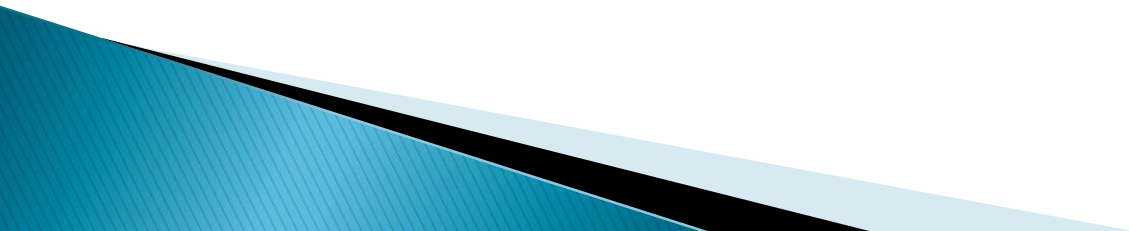
```
print(len(d1))
```

```
print(max(d1))
```



# Dicționare

**Exemplul 1** – Frecvența caracterelor dintr-un text dat pe o linie



# Dicționare

**Exemplul 2** – se dă o listă de  $n$  puncte în plan prin coordonate și etichetă. Dacă un punct apare de mai multe ori în listă se păstrează ultima etichetă asociată lui. Se citește un nou punct  $(x,y)$ . Să se afișeze eticheta acestuia, dacă a fost dată.

5

1 2 punctul 1

1 3 punctul 2

2 5 punctul 3

1 2 punctul 1 nou

4 1 punctul 4

3 2

