Curs 4: list, tuple, dict

1. Clasa list:

- Are la bază o construcţie de tip array
- Are proprietăţi asemănătoare cu a listelor dar şi a vectorilor din C++
- Obiectele de tip list sunt MUTABILE (îşi pot schimba valoarea fără a fi nevoie să se genereze un nou obiect cu un alt id) şi ITERABILE
- Obiectele de tip list sunt indexabile (pornind de la 0) folosind operatorul [ ].
- Spre deosebire de clasa string, putem atribui/schimba valorile unui element/slice din lista
- Fiind iterabila, putem folosi operatorul unar *, pentru “despachetarea” unei liste
- Atunci când facem un **slice** din lista, se generează un **nou obiect**, cu un nou id (shallow copy). Dacă sliceului vom atribui un nou obiect iterabil, modificarea se va vedea si in lista de provenienta a sliceului. Totusi, dac in slice facem update la o singura valoare, sliceul fiind obiect separat, schimbarea nu se va “vedea” în lista originala. Vedeţi exemplul 4.1.1

Exemplul 4.1.1 Declararea unei liste, schimbarea unei valori din lista, verificarea daca un element exista in lista, slice-uri.

O lista a catora metode din clasa list. A se observa ca aceste metode pot modifica direct valoarea obiectului apelant, spre deosebire de cazul clasei str, unde obiectul apelant nu se modifica, ci metodele returnau un nou string cu valoarea dorită.

Nume metodă	Descriere	Complexitate	Exemplu
`list.`**`append`**(*x*)	Extinde (la dreapta) lista apelantă cu un element căruia i se atribuie valoarea lui **x**. Lungimea listei creşte cu 1.	O(1)	4.1.2
`list.`**`extend`**(*iterable*)	concatenează la dreapta lui list toate elementele din iterable. Parametrul orimit nu trebuie sa fie neapărat tot o listă, ci orice obiect iterabil (set, string, range, dicţionar, tuple, etc) Echivalent ca rezultat cu: list[len(list):]=iterable sau list+=iterable	O(len(iterable))	4.1.3
`list.`**`insert`**(*i*, *x*)	inserează un element cu valoarea lui **x** pe poziţia **i.** Dimensiunea listei creşte cu 1.	O(k) k - numărul de elemente ce sunt shiftate	4.1.4

`list.`**`remove`**(*x*)	Şterge prima apariţie a lui **x** din list. Aruncă ValueError în cazul in care elementul căutat nu există în list.	O(n)	4.1.5
`list.`**`pop`**([*i=-1*])	returnează şi şterge elementul de pe poziţia i din listă. Dacă nu este specificat indicele, atunci se elimină ultimul element din listă (indexul -1)	O(k) atunci când se specifică indicele O(1) când se elimină ultimul element	4.1.6
`list.`**`clear`**()	Şterge elementele din lista, dar nu dereferenţiază lista (aşa cum ar face-o del(list)). Echivalent cu list=[ ]	O(1)	-
`list.`**`index`**(*x*[, *start*[, *end*]])	Caută pe **x** în list şi returnează indicele primei apariţii. Opţional se pot adăuga 2 parametri de *start* şi *end* pentru a căuta pe **x** în un anumit interval. În ambele cazuri indexarea se face de la primul element al lui list. În cazul în care **x** nu este în list, se aruncă un ValueError	O(n)	4.1.7
`list.`**`sort`**(*key=None*, *reverse=False*)	sorteaza lista crescator, cu conditia ca lista sa conţină elemente comparabile. Elementele care sunt considerate egale îşi vor păstra ordinea originală între ele (stable sort)	O(n log n) timsort	4.1.8
`list.`**`reverse`**()	Inversează ordinea elementelor din listă	O(n)	-
`list.`**`copy`**()	Crează şi întoarce un nou obiect egal în valoare cu obiectul apelant (dar id diferit)	O(n)	4.1.9

2. Clasa tuple:

- **imutabil**
- **iterabil**
- **indexabil**

Declararea unui tuplu, modificarea lui, iterarea printr-un tuplu, accesarea unui element.

Exemplul 4.2.1

Dece tuple si nu list?

- folosim tupluri pentru colecţii de obiecte heterogene. Listele ar trebui sa contina obiecte de acelaşi tip (omogene), deşi nu există o restricţie în acest sens.

- Deoarece tuplurile sunt imutabile, iteraţia printr-o astfel de structură este mai rapidă
- Tuplurile care conţin la rândul lor doar elemente imutabile pot fi folosite pe post de chei în dicţionare. Listele, fiind ele însele mutabile, nu pot fi folosite pe post de cheie (nu se poate face hashing pe ceva care îşi poate schimba valoarea)
- ”Write protection”

Nume metodă	Descriere	Complexitate	Exemplu
`tuple.`**`count(x)`**	returnează numărul aparițiilor lui **x** în tuplet	O(n)	
`tuple.`**`index(x)`**	caută prima apariție a lui **x** în tuplet și returnează indexul ei. În cazul în care **x** nu se găsește în tuplet, se aruncă o ValueError. (Existenţa se poate verifica mai simplu folosind cuvântul cheie **in**)	O(n)	

3. Clasa dict():

- **mutabil**
- **iterabil**
- **indexabil**

Dicţionarul este o enumerare de perechi de forma <key, value>, unde operaţiile de inserţie, interogare și update se fac în timp constant. În dicţionar cheile trebuie să fie unice, iar elementele care sunt chei trebuie sa fie de un tip **imutabil.** (ex: listele nu pot fi chei, dar tupletele pot)

Nume metodă	Descriere	Complexitate	Exemplu
`dict.`**`clear()`**	elimină toate valorile din dicţionar	O(n)	
`dict.`**`get(x,[val=None])`** `sau` **`dict[x]`**	returnează valoarea de la cheia **x**. Dacă nu există cheia **x**, se returnează o valoare *val* setată implicit cu None	O(1)	4.3.1
`dict.`**`update({key,value})`**	dacă **key** nu se găsește printre cheile din dicţionar, atunci se inserează în dicţionar perechea (key,value). Dacă acea cheie se află deja în dicţionar, se actualizează doar valoarea. Echivalent cu	O(1)	4.3.2

	dict[key]=value		
`dict.`**`keys()`**	returnează o listă formată din mulţimea cheilor	O(n)	
`dict.`**`values()`**	returnează o listă formată din mulţimea valorilor	O(n)	
`dict.`**`items()`**	returnează o listă formată din perechile (cheie,valoare)	O(n)	4.3.3