Curs 4

miercuri, 24 ianuarie 2024 10:50

Tipurile de date primitive sunt *imutabile*, adică, odată ce acestea au fost create ele nu se mai pot modifica. Dacă o variabilă x = 3 își modifică valoarea în 4, de fapt un nou întreg este creat și atribuit variabilei x.

Liste

Listele sunt unul din tipurile care reprezintă *colecții de elemente*.

O listă e o secvență *finită, ordonată.*

O listă e o secvență *finită, ordonată.*

- listele sunt *finite*, dar pot avea lungime oricât de mare
- *ordinea* elementelor contează: [1; 3; 2] ≠ [3; 1; 2]
- *accesul* la elementele listei e *secvențial* (acces direct doar la primul)
  - diferit de *vector/tablou*: *acces direct* (cu indice) la orice element

Capul listei si coada listei

Elementele unei liste

O listă poate conține ca elemente *orice tip de date*.

[4, 6, 8, 10, 12]

[“sir”, “de”, “caractere”]

O listă poate avea elemente inclusiv *alte liste*.

[[4, 9], [1, 2, 3, 4], [19, 20]]

O listă poate conține elemete de *diferite tipuri*.

[8, ‘a’, [3, 6, 8], “cuvant”, 9]

12

Modul de creare a unei liste

Modul de creare a unei liste

Putem *crea o listă* în mai multe moduri:

lista2 = [2022, 2023, 2024]

Se poate crea o listă și folosind constructorul *list()* astfel:

lista2 = list((2022, 2023)) #similar [2022, 2023]

lista3 = list("sir") # similar cu lista3 = ['s', 'i', 'r']

Accesarea elementelor unei liste

Elementele se pot accesa prin *index*. Indexul unui element poate fi pozitiv [0], [1], [2] etc. sau negativ [-1], [-2],[-3] etc.

```
lista = ["Timisoara", "Arad", "Bucuresti"]

print(lista1[0])          # o sa afiseze Timisoara
print(lista1[1])          # o sa afiseze Arad
print(lista1[-1]) # o sa afiseze Bucuresti, ultimul element
print(lista1[-2])         # o sa afiseze Arad
print(lista1[-3])         # o sa afiseze Timisoara
```

Accesarea elementelor unei liste

Putem să accesăm *mai multe elemente* dintr-o listă specificând indexul elementului de la începutul secvenței și al elementului de la sfârșitul secvenței. Rezultatul este *o nouă listă* conținând elementele specificate.

```
lista = ["Timisoara", "Arad", "Bucuresti", „Iasi"]

print(lista[1:3])
# va afisa ['Arad', 'Bucuresti']
```

```
# noua lista va porni de la elementul 1 pana la elementul
3. Va include primul element, dar nu si pe ultimul
```


Accesarea elementelor unei liste

Putem folosi și *indecși negativi*:

```
lista = ["Timisoara", "Arad", "Bucuresti", "Iasi"]

print(lista[-3:-1])
# va afisa ['Arad', 'Bucuresti']

print(lista[:-1])
# va afisa ['Timisoara', 'Arad', 'Bucuresti']
```

```
print(lista[:-1])
# va afisa ['Timisoara', 'Arad', 'Bucuresti']

print(lista[-2:])
# va afisa ['Bucuresti', 'Iasi']
```

20

Verificarea existenței unui element

Putem verifica dacă *un element este într-o listă* folosind instrucțiunea *in*:

```
lista = ["Timisoara", "Arad", "Bucuresti", "Iasi"]
elem = "Arad"
```

```
elem = "Arad"
if elem in lista:
        print("elementul cautat se afla in lista")
else:
        print("elementul cautat nu se afla in lista")
```

21

Lista[1] = "Craiova" va inlocui "Arad"

Adaugarea unui element în listă

Putem adăuga un element nou în listă, fără a elimina un alt element, vom folosi metoda *append()*. Elementul va fi adăugat la *finalul listei*.

append(). Elementul va fi adăugat la *finalul listei*.

```
lista = ["Timisoara", "Arad", "Bucuresti"]
lista.append("Resita")
```

Se inserează pe ultima poziție. Noua listă va fi:
['Timisoara', 'Craiova', 'Cluj-Napoca', 'Resita']

23

Adaugarea unui element în listă

Putem adăuga un element nou în listă pe o

Putem adăuga un element nou în listă pe o anumită poziție, fără a elimina un alt element, folosind metoda *insert()*:

```
lista = ["Timisoara", "Arad", "Bucuresti"]
lista.insert(1, "Resita")
```

Se inserează pe poziția 1. Noua listă va fi:

['Timisoara', 'Resita', 'Craiova', 'Cluj-Napoca']

Adaugarea elementelor unei alte liste

Pentru a adauga elementele unei alte liste în lista curentă vom folosi metoda *extend()*. Elementele vor fi adaugate *la finalul listei curente*.

```
lista = [3, 4, 5]
lista2 = [101, 102, 110]
lista.extend(lista2)
print(lista)

# va afisa [3, 4, 5, 101, 102, 110]
```

Eliminarea elementelor din listă

Metoda *remove()* va elimina doar *prima apariție* a elementului din listă.

```
lista = [1, 2, 3, 4, 5, 2]
lista.remove(2)
print(lista)

# va afisa [1, 3, 4, 5, 2]
```

Putem *elimina* un element specificând *indexul* său cu metoda *pop()*

```
lista = [1, 2, 3, 4, 5, 2]
lista.pop(2)
print(lista)

# va afisa [1, 2, 4, 5, 2]
```

29

Eliminarea elementelor din listă

Dacă *nu specificăm indexul* în cadrul metodei pop() se va elimina *ultimul element* al listei.

```
lista = [1, 2, 3, 4, 5]
lista.pop()
print(lista)

# va afisa [1, 2, 3, 4]
```

Eliminarea elementelor din listă

Pentru a șterge un element din listă putem folosi *del*

```
lista = [1, 2, 3, 4, 5]
del lista[0]
print(lista)

# va afisa [2, 3, 4, 5]
```

del poate șterge și întreaga listă:

del lista

31

Eliminarea elementelor din listă

Dacă vrem să eliminăm toate elementele dintr-o listă folosim metoda *clear()*

```
lista = [1, 2, 3, 4, 5]
lista.clear()
```

```
lista.clear()
print(lista)

# va afisa lista vidă: []
```

32

Parcurgerea element cu element

Putem parcurge fiecare element al unei liste cu ajutorul instrucțiunii *for*

```
lista = [1, 2, 3]
for x in lista:
        print(x)
```

Va afișa:

1

2

3

33

Sortarea unei liste

Pentru a sorta elementele unei liste vom folosi metoda

Pentru *a sorta* elementele unei liste vom folosi metoda *sort()*. Această metodă sortează implicit elementele din listă în *ordine crescătoare, respectiv alfabetică*.

```
numere = [1, 3, 2, 6, 5, 4]
cuvinte = ['unu', 'doi', 'trei']
numere.sort()
cuvinte.sort()
print(numere, cuvinte)
```

Va afișa:

```
[1, 2, 3, 4, 5, 6]
['doi', 'trei', 'unu']
```

36

Sortarea unei liste

Sortarea unei liste

Pentru a sorta elementele unei liste *descrescător* vom folosi metoda sort() cu argumentul *reverse = True*.

```
numere = [1, 3, 2, 6, 5, 4]
cuvinte = ['unu', 'doi', 'trei']
numere.sort(reverse = True)
cuvinte.sort(reverse = True)
print(numere, cuvinte)
```

Va afișa:

```
[6, 5, 4, 3, 2, 1]
['unu', 'trei' , 'doi']
```

37

Sortarea unei liste

Putem folosi *criterii specifice de sortare* a unei liste utilizând argumentul *key = functie*. Va aplica mai întâi funcția pe fiecare element al listei și apoi va ordona în funcție de rezultatul funției.

```
def functie(x):
  return abs(x - 10)

lista = [1, 2, 10, 11, 29]
lista.sort(key = functie)
print(lista)
```

Va afișa:

[10, 11, 2, 1, 29]

38

Inversarea ordinii unei liste

Pentru a *inversa ordinea* elementelor unei liste folosim metoda *reverse()*

```
numere = [1, 2, 3, 4, 5, 6]

numere.reverse()
print(numere)
```

Va afișa:

[6, 5, 4, 3, 2, 1]

[6, 5, 4, 3, 2, 1]

39

Copierea unei liste

Pentru a inversa copia o listă într-o altă listă vom folosi metoda *copy()* sau constructorul *list()*

```
lista = [1, 2, 3, 4, 5, 6]
lista1 = lista.copy()
lista2 = list(lista)
```

Dacă folosim operatorul de atribuire =, nu se va copia conținutul unei liste în altă listă. Va fi doar o *referință spre prima listă*, iar orice schimbare în

copia conținutul unei liste în altă listă. Va fi doar o *referință spre prima listă*, iar orice schimbare în prima listă se va regăsi dacă folosim noul obiect.

40

Concatenarea a două liste

Folosind *operatorul +* pentru 2 liste vom concatena conținutul acestora *într-o nouă listă*.

```
lista1 = ["a", "b", "c"]
lista2 = [1, 2, 3]

lista3 = lista1 + lista2
print(lista3)
```

```
print(lista)

# va afisa ['a', 'b', 'c', 1, 2, 3]
```

41

Functii pentru liste: map()

Putem aplica o funcție pe fiecare element al unei liste cu *funcția map()*.

Funcția *map()* are 2 argumente, primul este o *funcție*, iar al doilea este *lista* pe care se aplică funcția.

```
lista_noua = map(functie, lista)
```

```
lista_noua = map(functie, lista)
```

42

Functii pentru liste: map()

Exemplu:

```
def patrat(x):
    return x*x

numere = [1, 2, 3, 4]
lista = map(patrat, numere)
```

```
lista = map(patrat, numere)
print(list(lista))
```

Va afișa:

[1, 4, 9, 16]

43

Functii pentru liste: reduce()

Funcția *reduce()* aplică secvențial o funcție dată pe elementele unei liste.

Are 2 argumente și returnează rezultatul aplicării secvențiale a funcției

secvențiale a funcției

rezultat = reduce(functie, lista)

Funcția e definită în modulul *functools*.

45

Functii pentru liste: reduce()

Mod de lucru al funcției *reduce()*:

- La primul pas, se aplică funcția cu *primele 2 elemente din listă* și se reține rezultatul

elemente din listă și se reține rezultatul

- După care se aplică funcția *cu rezultatul* obținut la pasul precedent și *următorul element* din listă și se reține rezultatul
- Pasul anterior *se repetă* până se epuizează toate elementele din listă și se returnează *rezultatul final*

46

Functii pentru liste: filter()

Funcția *filter(functie, lista)* testează fiecare element al listei cu funcția dată și returnează

Funcția *filter(functie, lista)* testează fiecare element al listei cu functia dată și returnează doar elementele care sadisfac condiția din funcție.

Functia primita ca parametru trebuie sa returneze *True* sau *False*

Rezultatul va conține doar elementele pentru care funcția *returnează True*

48

Adresele elemetelor unei liste

Pentru a afla adresa din memorie a unui element dintr-o listă putem aplica funcția *id()*.

Exemplu:

lista = [0, 1, 2, 3, 5, 8, 13]

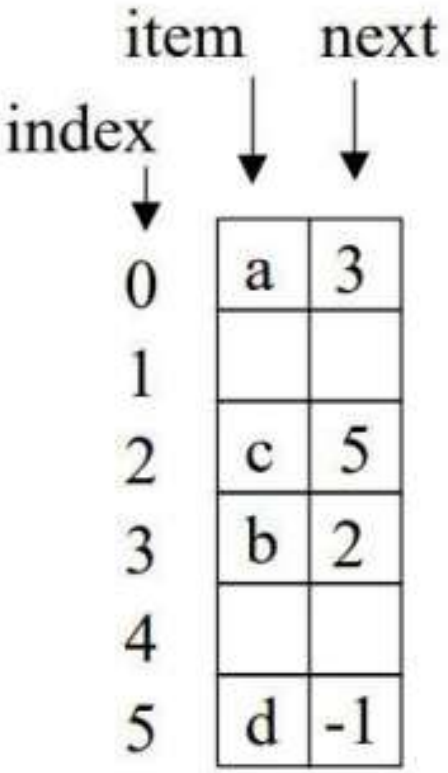
print(id(lista[0]))

print(id(lista[1]))

Va afișa:

2163888750800

2163888750832



51

Adresele elemetelor unei liste

Adresele elemetelor unei liste

De regulă,adresa din memorie a unui byte se exprimă în baza 16. Pentru a afișa adresa în baza 16 putem folosi funcția *hex()*

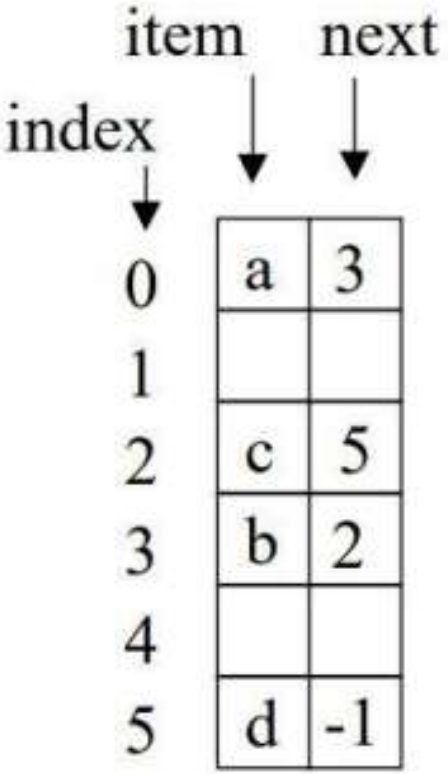
Exemplu:

```
lista = [0, 1, 2, 3, 5, 8, 13]
print(hex(id(lista[0])))
print(hex(id(lista[1])))
```

Va afișa:

0x227882c00d0

0x227882c00f0



Capul și coada listei (Head, Tail)

Dacă avem o listă și vrem să extragem *capul listei și coada* ei putem scrie:

```
lista = [0, 1, 2, 3, 4, 5]
head, tail = lista[0], lista[1:]

# head = 0
# tail= [1, 2, 3, 4, 5]
```

53

Funcții recursive – Creare listă

```
def create_list_recurse(start, end):
    if start > end:
        return []
    return [start] + create_list_recurse(start + 1, end)

create_list_recurse(0, 9)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

create_list_recurse(3, 1) #edge case returns empty
# []
```

[]

https://stackoverflow.com/questions/62854078/how-to-create-a-recursive-function-to-create-a-list-of-values 54

Funcții recursive – Număr de elemente

Recursivitate:

```
def lungime(lista):
    if lista == []:
        return 0
    return 1 + lungime(lista[1:])

lungime([0, 1, 2, 3, 4, 5])
```

Tail recursion:

```
def lungime2(lista, nr=0):
    if lista == []:
        return nr
    return lungime2(lista[1:], 1+nr)

lungime2([0, 1, 2, 3, 4, 5])
```

lungime([0, 1, 2, 3, 4, 5])
#6

lungime2([0, 1, 2, 3, 4, 5])
#6

55

Funcții recursive – Contine elementul

Funcția recursivă care ne indică dacă un element este într-o listă sau nu:

```
def contine (x, lista):
    if (lista == []):
        return False
    return x == lista[0] or contine(x, lista[1:])
```

```
    return x == lista[0] or contine(x, lista[1:])

print(contine(4,[1,2,3]))
#False
```

56