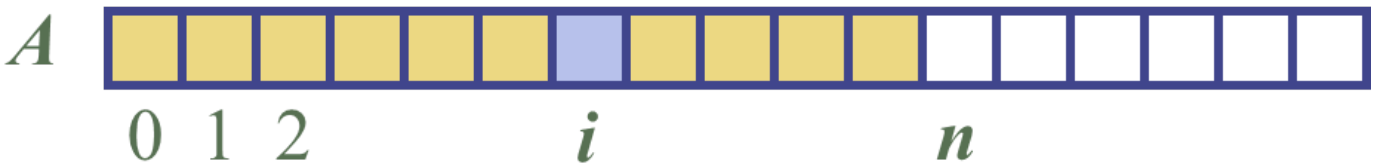
Nizovi

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

2019.

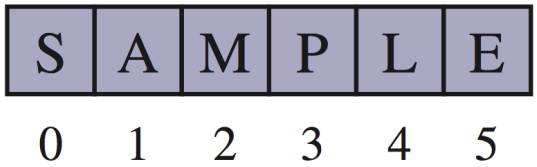
Python i nizovi

- Python ima ugrađene tipove `list`, `tuple` i `str`
- svaki od ovih tipova omogućava pristup elementima po indeksu, npr. `A[i]`
- svaki od ovih tipova interno koristi **niz** za skladištenje podataka
- niz je skup susednih memorijskih lokacija koje mogu biti adresirane pomoću sukscesivnih indeksa koji počinju od 0

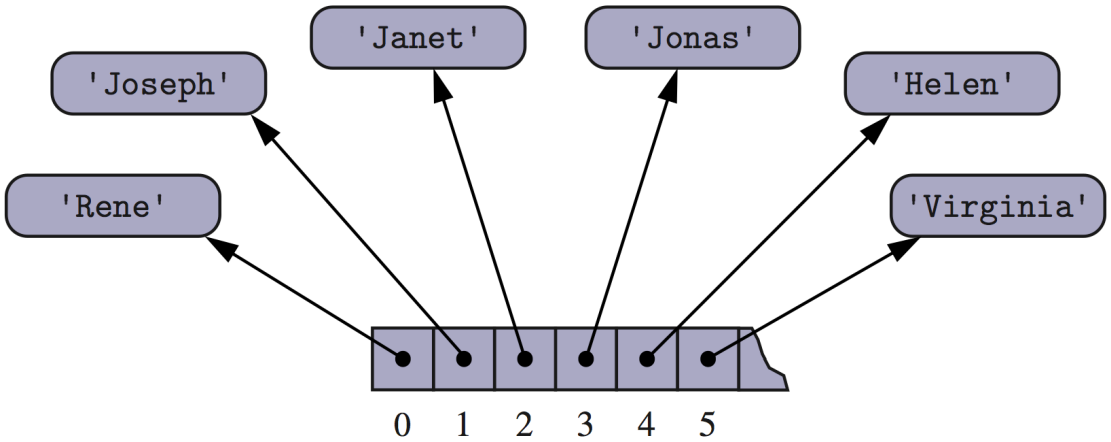


Nizovi karaktera / nizovi referenci na objekte

- niz može da čuva primitivne elemente, na primer karaktere, predstavljajući kompaktni niz



- niz može čuvati i reference na objekte



Kompaktni nizovi

- podrška za rad sa kompaktnim nizovima nalazi se u modulu `array`
- ovaj modul definiše klasu `array` koja predstavlja kompaktni niz za primitivne tipove podataka
- konstruktor za `array` kao prvi parametar očekuje slovo koje označava tip elemenata

```
primes = array('i', [2, 3, 5, 7, 11, 13, 17, 19])
```

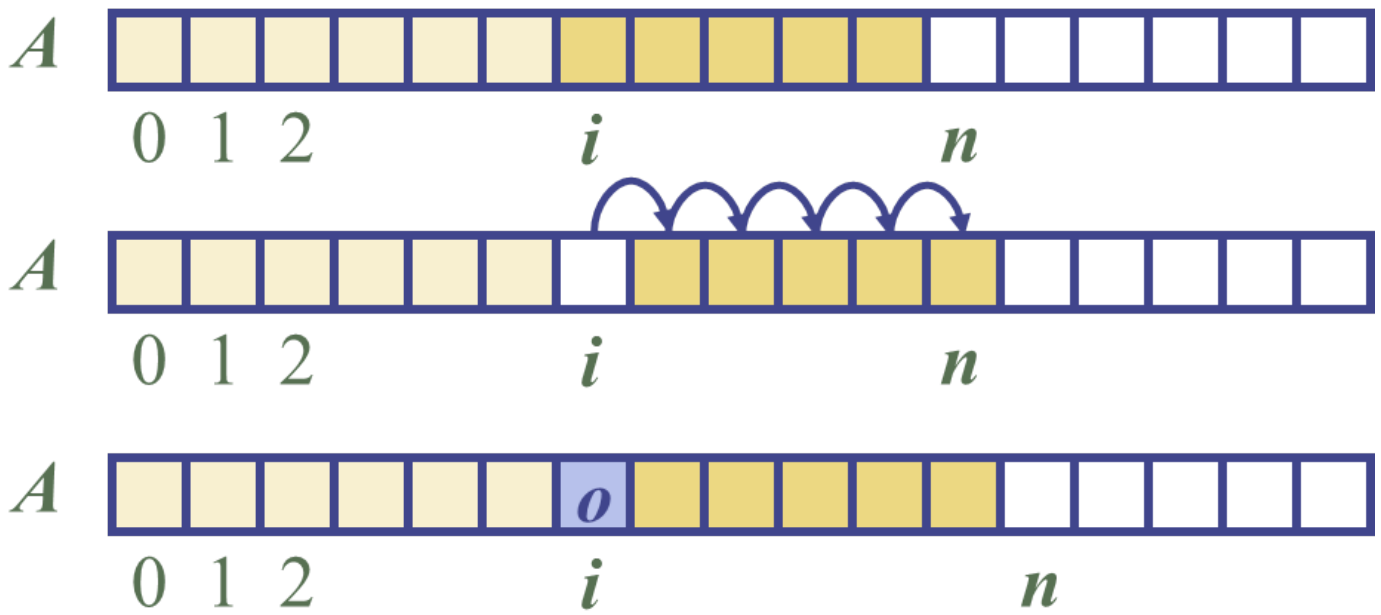
Tipovi elemenata za `array`

- klasa `array` prepoznaje sledeće oznake tipa elemenata

kod	tip podatka	veličina
`'c'`	char	1
`'b'`	signed char	1
`'B'`	unsigned char	1
`'u'`	Unicode char	2
`'h'`	signed short int	2
`'H'`	unsigned short int	2
`'i'`	signed int	2
`'I'`	unsigned int	2
`'l'`	signed long	4
`'L'`	unsigned long	4
`'f'`	float	4
`'d'`	double	8

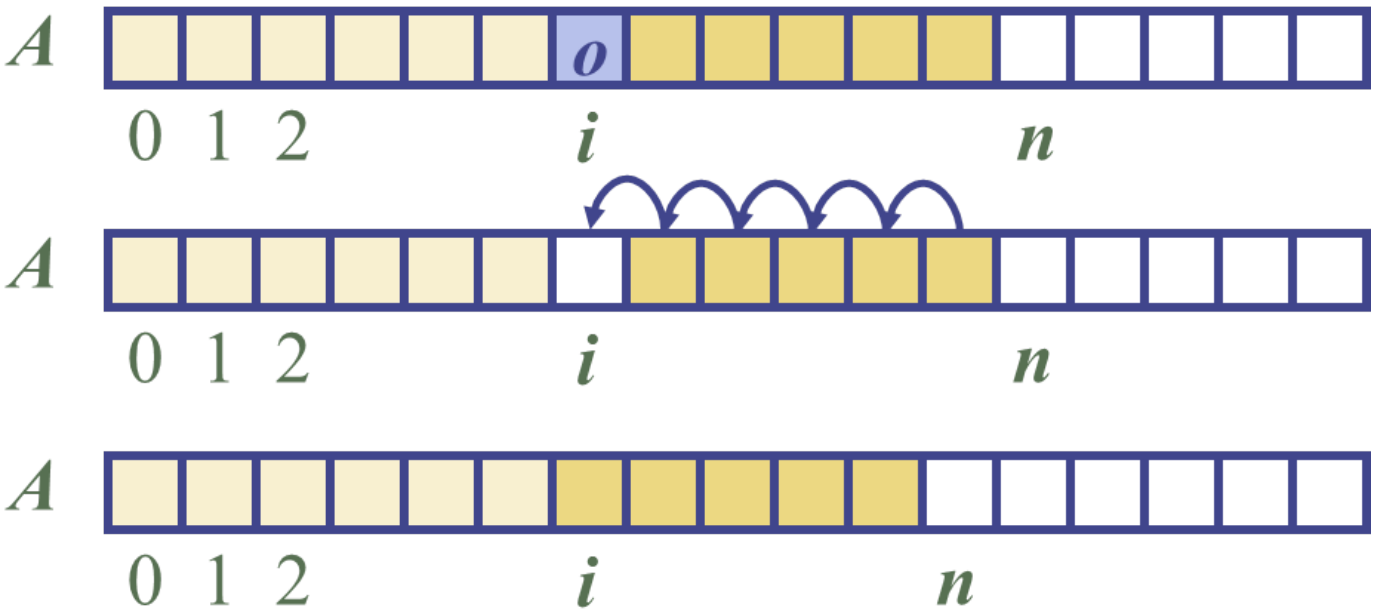
Ubacivanje elementa

- u operaciji add(i, o) treba napraviti mesta za novi element pomeranjem $n - i$ elemenata $A[i], \ldots, A[n-1]$ u desno za jedno mesto
- u najgorem slučaju $(i = 0)$ za ovo je potrebno $O(n)$ vreme



Uklanjanje elementa

- u operaciji remove(i) treba popuniti rupu na mestu elementa koji se uklanja pomeranjem $n - i - 1$ elemenata $A[i+1], \ldots, A[n-1]$ u levo za jedno mesto
- u najgorem slučaju ($i = 0$) za ovo je potrebno $O(n)$ vreme



Performanse niza

- za implementaciju liste pomoću niza
  - prostor koji zauzima struktura u memoriji je $O(n)$
  - pristup i-tom elementu je u $O(1)$ vremenu
  - ubacivanje i uklanjanje su u $O(n)$ vremenu u najgorem slučaju

Performanse niza

- šta je zaista najgori slučaj kod dodavanja?
  - niz popunjen do kraja
  - zauzmemo novi (veći) niz u memoriji
  - prepišemo sve podatke iz starog niza
  - odbacimo stari niz
- moramo unapred znati veličinu niza!

Strategije za proširenje niza

- koliko velik treba da bude novi niz prilikom proširenja?
  - inkrementalna strategija: novi niz će biti duži za neko konstantno c
  - strategija dupliranja: novi niz će biti duplo duži od prethodnog

Poređenje strategija

- poredimo strategije analizirajući ukupno vreme $T(n)$ potrebno za obavljanje n operacija ubacivanja
- krećemo od niza dužine 1
- amortizovano vreme add operacije: prosečno vreme potreno za operaciju za niz od n operacija, $T(n)/n$

Poređenje strategija: inkrementalna

- pravimo novi niz $k = n/c$ puta
- ukupno vreme $T(n)$ za seriju od n operacija ubacivanja je proporcionalno sa:

$$n + c + 2c + 3c + 4c + \ldots + kc =$$

$$n + c(1 + 2 + 3 + \ldots + k) =$$

$$n + ck(k + 1)/2$$

- c je konstanta, sledi da $T(n)$ je $O(n + k^2)$ odnosno $O(n^2)$
- $\Rightarrow$ amortizovano vreme operacije ubacivanja je $O(n)$

Poređenje strategija: dupliranje

- pravimo novi niz $k = \log_2 n$ puta
- ukupno vreme $T(n)$ za seriju od n operacija ubacivanja je proporcionalno sa:

$$n + 1 + 2 + 4 + 8 + \ldots + 2^k =$$

$$n + 2^{k+1} =$$

$$3n - 1$$

- $T(n)$ je $O(n)$
- $\Rightarrow$ amortizovano vreme operacije ubacivanja je $O(1)$

Implementacija u Pythonu 1

```
class DynamicArray:
  def __init__(self):
    self._n = 0                    # stvarni broj elemenata
    self._capacity = 1             # kapacitet niza
    self._A = self._make_array(self._capacity) # zauzimanje niza
                                               # u memoriji
  def __len__(self):
    return self._n                 # vrati broj elemenata

  def __getitem__(self, k):
    if not 0 <= k < self._n:
      raise IndexError('invalid index')
    return self._A[k]              # dobavi element po indeksu
```

Implementacija u Pythonu 2

```
def append(self, obj):
  if self._n == self._capacity:          # da li je niz popunjen?
    self._resize(2 * self._capacity)     # udvostruči mu kapacitet
  self._A[self._n] = obj
  self._n += 1

def _resize(self, c):
  B = self._make_array(c)                # novi (veći) niz
  for k in range(self._n):               # prepiši vrednosti u njega
    B[k] = self._A[k]
  self._A = B
  self._capacity = c

def _make_array(self, c):
  import ctypes
  return (c * ctypes.py_object)()        # pogledaj dok za ctypes
```