2 Programski paket `numpy`

element.hr

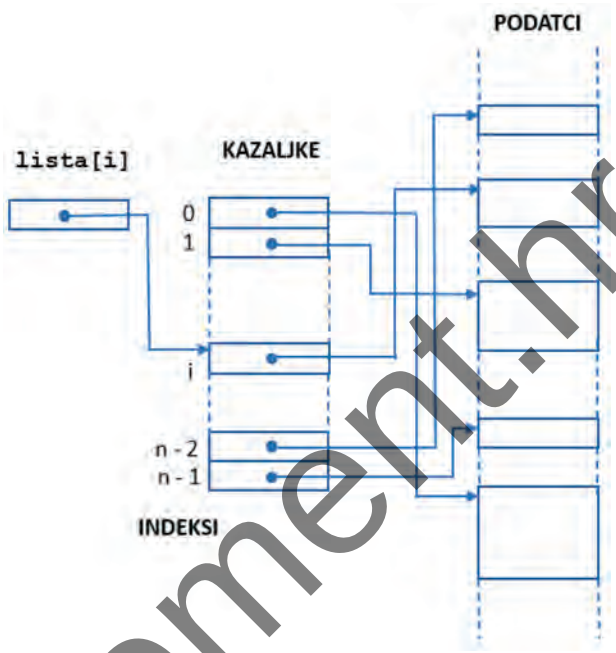
2.1. Podatkovna struktura `array`

Pokazalo se da programski jezik *Python*, koji je inače široko prihvaćen zbog jednostavne sintakse i dobre čitljivosti programskog teksta, u svom izvornom obliku nije prikladan za programiranje velikih numeričkih problema. Za to postoje dva osnovna razloga:

1. u *Pythonu* se nizovi brojeva, koje se uobičajeno pojavljuju pri računanju, pohranjuju u strukturi `list`
2. *Python* je tzv. interpretativni jezik čije se naredbe tek pri izvođenju programa interpretiraju, tj. prevode u niz strojnih naredbi, što znatno usporava odvijanje programa.

U *Pythonu* osim toga postoji i struktura `tuple` koja podsjeća na listu, ali su njezini elementi određeni prilikom oblikovanja i ne mogu se nakon toga mijenjati. Umjesto uglatih zagrada kod liste struktura `tuple` ograđuje se okruglim zagradama.

Elementi liste mogu biti razmješteni razbacano u spremniku računala, a indeksi zapravo odabiru lokacije u kojima su smještene kazaljke na te elemente. Takva organizacija omogućuje jednostavno dinamičko stvaranje i preoblikovanje lista što je vrlo korisno za ostvarenje fleksibilnih programskih rješenja. Slika 2.1. prikazuje primjer razmještaja liste u spremniku računala.

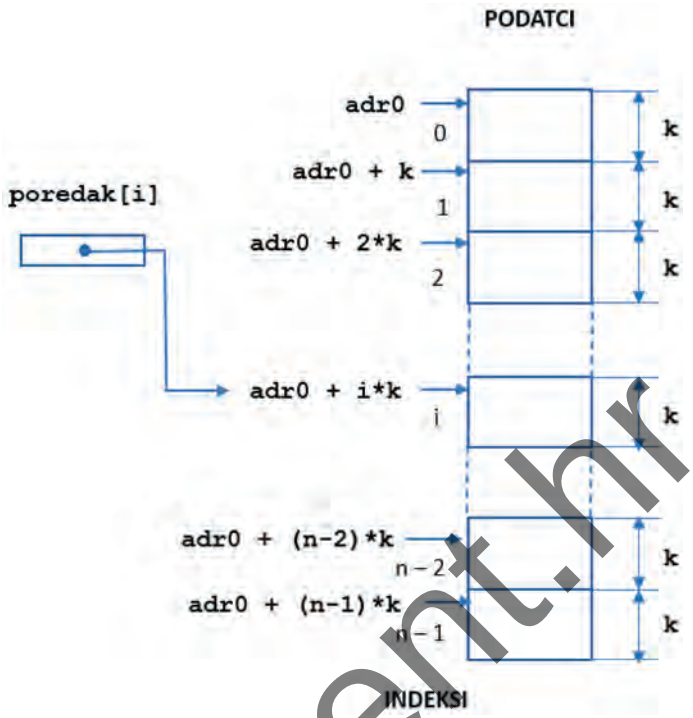


Slika 2.1. Razmještaj liste u spremniku računala

element.hr

Međutim, takva je organizacija neprikladna za smještanje velike količine brojčanih podataka s kojima treba obavljati mnogo aritmetičkih operacija. Posredno adresiranje pojedinih elemenata liste putem kazaljki znatno usporava odvijanje izračunavanja. Podatcima će se moći mnogo brže pristupati ako su oni pohranjeni kao kompaktna cjelina. U programskim jezicima kao što su C/C++ i FORTRAN predviđen je razmještaj niza srodnih podataka u strukturu koju nazivamo `poretkom` (engl. `array`). Takav je razmještaj prikazan slikom 2.2.

Ovakav smještaj moguć je samo kada su svi podatci istog tipa jer svi elementi poretka moraju biti jednake veličine, tj. imati jednaki broj bajtova. Osim toga, veličina poretka mora biti unaprijed određena. To malo ograničenje u odnosu na liste, čija se veličina može nesmetano mijenjati, ne smeta mnogo u organizaciji



Slika 2.2. Razmještaj poretka u spremniku računala

programskih rješenja. Štoviše, ovakav pristup olakšava izračunavanje adrese pojedinog elementa. Iz slike 2.2. može se zaključiti da je adresa elementa s indeksom `i` jednaka `adr0+i*k`, gdje je `adr0` početna adresa cijelog poretka i `k` broj bajtova potrebnih za smještanje jednog elementa.

Drugo ograničenje, zbog toga što je *Python* interpretacijski jezik, može se zaobići tako da se dijelovi funkcija unaprijed pripreme u već prevedenom strojnom obliku. Štoviše, ti dijelovi funkcija uobičajeno se i pišu u drugim programskim jezicima (pretežito su to jezici C i FORTAN). Takva programska rješenja pripremaju i temeljito ispituju radne skupine sastavljene od vrhunskih programera, vrsnih matematičara i stručnjaka za pojedina područja primjene.

Ovako pripremljene funkcije javno su dostupne i postale su sastavni dijelovi tzv. otvoreno dostupnih programa (engl. *open-source software*). Otvoreno dostupne funkcije mogu se primijeniti kao dijelovi modularno građenih *Python* programa. Veliki broj funkcija pripremljenih za različita programska rješenja čini svojevrsni ekosustav *Pythona* u koji se može jednostavno pristupiti. Čitav taj otvoreni sustav

element.hr

pohranjen je u javno dostupan repozitorij (skladište) pod nazivom *Python Package Index* – PyPI.

U Prilogu A opisan je način dobavljanja takvih funkcija i pokazano je kako se dobavljaju tri paketa funkcija koja se rabe u ovom udžbeniku. To su paketi:

- `numpy` koji uvodi u okružje *Pythona* poredak `array` kao novi tip podataka, kao i niz funkcija koje su pripremljene uz prethodno prevođenje i izvode se mnogo brže nego funkcije napisane u samom jeziku *Python*; paket se poziva kao da je jedinstveni modul iako on sadrži i više potpaketa
- `scipy` koji sadrži niz potpaketa za rješavanje pojedinih klasa problema numeričke matematike kao što su rješavanje sustava linearnih jednadžbi, rješavanje diferencijalnih jednadžbi i optimiranje (pronalaženje ekstrema funkcija više varijabli)
- `matplotlib` koji sadrži mnogo potpaketa za raznovrsna crtanja, no za vizualizaciju primjera u ovom će se udžbeniku rabiti samo jedan od njih s nazivom `pyplot`.

U ovom poglavlju opisuju se osnovne mogućnosti paketa `numpy`, dok će uporaba drugih dvaju paketa biti razmatrana u sljedećim poglavljima.

element.hr

2.2. Osnovna obilježja modula `numpy`

Načela modularnog programiranja opisana su u *Prilogu B* na jednostavnim primjerima modula. Želi li se modul primijeniti, treba ga dobaviti instrukcijom `import`. Uobičajeno je prilikom importiranja modula iz praktičnih razloga skratiti njegov naziv. Iako je `numpy` cijeli paket, on se interpretira i kao modul pa se na takav način i importira. Za taj modul uobičajila se u širokom krugu korisnika pokrata `np`. Nakon importiranja modula može se u interaktivnom sučelju IDLE naredbom `dir(np)` pregledati sva imena koja se u modulu nalaze. Tako se može pisati:

```
>>> import numpy as np
>>> dir(np)
```

U dobivenom popisu imena su nazivi pojedinih objekata, varijabli, funkcija ili metoda koji se nalaze u tom modulu. Znamo da se naredbom `help()` mogu saznati

sve nužne informacije o svakom imenu iz tog popisa. Tako se, primjerice, za ime `np.array` dobiva opis funkcije `array()` i svih njezinih parametara. Dio tog opisa izgleda ovako:

```
>>> help(np.array)
    Help on built-in function array in module numpy:

    array(...)
        array(object, dtype=None, *, copy=True, order='K',
              subok=False, ndmin=0, like=None)

    Create an array.

    Parameters
    ----------
    object : array_like
        An array, any object exposing the array interface, an
        object whose ``__array__`` method returns an array,
        or any (nested) sequence.
        If object is a scalar, a 0-dimensional array containing
        object is returned.
    dtype : data-type, optional
        The desired data-type for the array. If not given, numpy
        will try to use a default ``dtype`` that can represent
        the values (by applying promotion rules when necessary.)
```

element.hr

2.3. Jednodimenzijski poredak

2.3.1. Prevođenje liste u poredak

Jednodimenzijski poredak (1D poredak) osnovna je struktura podatka koji se rabi u numeričkoj matematici. Svaka se lista može prevesti u poredak funkcijom `np.array()`. Lista se, primjerice, može oblikovati na sljedeći način:

```
>>> a = [1,2,3,4]
>>> a
    [1, 2, 3, 4]
>>> print(f'lista a = {a}')
    lista a = [1, 2, 3, 4]
```

Interni prikaz liste dobivamo pisanjem imena varijable `a`, a izgled ispisa funkcijom `print()`. Prevođenjem liste u poredak dobivamo:

```
>>> import numpy as np
>>> p_a = np.array(a)
>>> p_a
    array([1, 2, 3, 4])
>>> print(f'poredak p_a = {p_a}')
    poredak p_a = [1 2 3 4]
```

Poredak se od liste razlikuje tako što se ispred sadržaja u internom prikazu pojavljuje naziv `array`, dok se funkcijom `print()` pri ispisu unutar uglatih zagrada pojavljuju vrijednosti elemenata, ali bez zareza između njih.

Prilikom definiranja poretka može se listu navesti i neposredno kao argument funkcije `np.array()`:

```
>>> p_b = np.array([3,4,5,6])
>>> p_b
    array([3, 4, 5, 6])
>>> print(f'poredak p_b = {p_b}')
    poredak p_b = [3 4 5 6]
```

Jednako tako funkcija `np.array()` u poredak će prevesti i n-torku (`tuple`) elemenata:

```
>>> p_c = np.array((7,8,9,10))
>>> p_c
    array([7, 8, 9, 10])
>>> print(f'poredak p_c = {p_c}')
    poredak p_c = [7 8 9 10]
```

U prethodnim su primjerima svi elementi tipa `int`. To je vidljivo iz ispisa njihovih vrijednosti. No posebnim atributom `dtype` može se to i ustanoviti:

```
>>> p_c.dtype
    dtype('int64')
```

Vidimo da su elementi poretka smješteni u 64 bita, odnosno 8 bajtova (ovisno o inačici instaliranog okruženja elementi mogu biti smješteni i u 32 bita). Prema tome, u taj poredak mogu biti smješteni cijeli brojevi iz intervala $[-2^{63}, 2^{63}-1]$. U to se možemo uvjeriti programom `program_2_1.py`.

element.hr

```
# program_2_1.py
'''
Modul numpy za smještanje cijelih brojeva u poretke odabire:
   - četiri bajta, tj. 32 bita, što se označava kao tip int32 ili
   - osam bajtova, tj. 64 bita, što se označava kao tip int64.
'''
import numpy as np
# primjeri lista
n = 32
d = [-2**(n-1),2**(n-1)-1]
m = 64
f = [-2**(m-1),2**(m-1)-1]

if __name__ == '__main__':
    p_d = np.array(d, dtype='int32')
    print(f'lista d =     {d}\nporedak p_d = {np.array(d)}
          p_d.dtype = {p_d.dtype}')
    e = [d[0], d[1]+1]
    p_e = np.array(e)
    print(f'lista e =     {e}\nporedak p_e = {np.array(e)}
          p_e.dtype = {p_e.dtype}')
    #
    p_f = np.array(f)
    print(f'\nlista f =     {f}\nporedak p_f = {np.array(f)}  p_f.dtype
          = {p_f.dtype}')
    g = [f[0], f[1]+1]
    p_g = np.array(g)
    print(f'lista g =     {g}\nporedak p_g = {np.array(g)}
          p_g.dtype = {p_g.dtype}')
```

Program 2.1. Primjer stvaranja poredaka

Izgled ispisa tog programa je ovakav:

```
lista d =     [-2147483648, 2147483647]
poredak p_d = [-2147483648  2147483647]     p_d.dtype = int32
lista e =     [-2147483648, 2147483648]
poredak p_e = [-2147483648  2147483648]     p_e.dtype = int64

lista f =     [-9223372036854775808, 9223372036854775807]
poredak p_f = [-9223372036854775808  9223372036854775807]
          p_f.dtype = int64
lista g =     [-9223372036854775808, 9223372036854775808]
poredak p_g = [-9.22337204e+18  9.22337204e+18]     p_g.dtype = float64
```

Poredak p_d se sastoji od dva elementa: najmanjeg i najvećeg cijelog broja koji se mogu smjestiti u 32 bita, odnosno u interval $[-2^{31}, 2^{31}-1]$. Ako se u listi d poveća gornja granica samo za jedan, dobiva se lista e koja pri prevođenju u

element.hr

poredak više ne može biti smještena u 32-bitne elemente, tako da je pripadni poredak `p_e` tipa `int64`, što znači da se u takvu listu mogu pohranjivati brojevi iz intervala $[-2^{63}, 2^{63}-1]$. Tako lista `f` sadrži upravo te granične vrijednosti pa se može prevesti u poredak `p_f` koji je još uvijek tipa `int64`. Međutim, uvećamo li u listi `f` gornju graničnu vrijednost samo za jedan, dobivamo listu `g`, koja će se prevesti u poredak `p_g` čiji tip je `float64`. Dakle, funkcija `np.array()` odabire tip poretka u skladu sa zadanim vrijednostima liste koju prevodi u poredak.

Ako je samo i jedan element liste tipa `float`, generirani poredak bit će tipa `float`:

```
>>> h = [1,2,3.,4]
>>> p_h = np.array(h)
>>> p_h
    array([1., 2., 3., 4.])
>>> p_h.dtype
    dtype('float64')
```

Prilikom prevođenja lista u poretke može se tip poretka eksplicitno odabrati u pozivu funkcije `np.array()`:

```
>>> k = [1,2,3,4]
>>> p_k = np.array(k, dtype=float)
>>> print(p_k)
    [1. 2. 3. 4.]
>>> p_k.dtype
    dtype('float64')
```

Pri definiranju poretka `p_k` nije posebno zatraženo da se želi definirati tip `float64`, međutim, za odabranu inačicu instaliranog razvojnog okruženja `[MSC v.1941 64 bit (AMD64)]` to se implicitno podrazumijeva. Brojevi tipa `float` pohranjuju se kao 64-bitovni brojevi u skladu sa standardom IEEE 745.

U oblik poretka mogu se prevesti i pojedinačne (skalarne) vrijednosti. Pritom se dobiva tzv. nul-dimenzijski poredak. Iako se to u prvi mah čini nesuvislim, tako definirane skalarne vrijednosti poprimaju obilježja potrebna za uporabu u modulu `numpy`. Tako se može pisati:

```
>>> import numpy as np
>>> x = 100.
>>> p_x = np.array(x)
>>> print(f'p_x = {p_x}')
    p_x = 100.0
```

element.hr

```
>>> p_x.dtype
    dtype('float64')
```

ili:

```
>>> p_y = np.array(0)
>>> print(f'p_y = {p_y}')
    p_y = 0
>>> p_y.dtype
    dtype('int64')
```

2.4. Pogreške u prikazu brojeva s pomičnom točkom

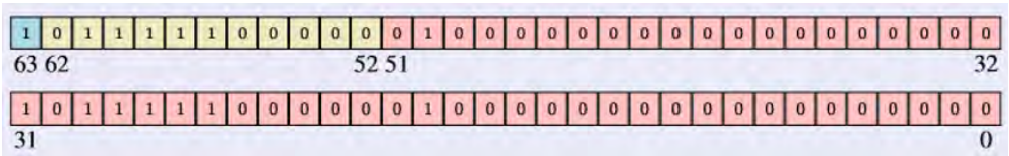
2.4.1. Standard IEEE 745

Standard IEEE 745 propisuje način smještanja brojeva jednostruke preciznosti u 32-bitovne riječi i brojeva dvostruke preciznosti u 64-bitovne riječi. U modulu `numpy` uobičajno se rabi tip `float64` (iako se posebno može definirati i tip `float32`) pa će se detaljnije razmotriti brojeve dvostruke preciznosti u kojemu su 64 bita podijeljena u tri polja:

- polje od jednog bita za predznak σ
- polje od 11 bitova za posmaknuti eksponent E i
- polje od 52 bita za razlomljeni dio signifikanda s.

Implicitno je pretpostavljeno da lijevo od binarne točke uvijek stoji znamenka 1. Iznimno, kada je vrijednost eksponencijalnog polja jednaka nuli ta je znamenka jednaka 0 i takav se signifikand naziva denormiranim signifikandom. Prema tome, signifikand ima ustvari ukupno 53 bita. Raspored bitova u tim poljima prikazan je na slici 2.3.

element.hr



Slika 2.3. Raspored polja u pohrani broja dvostruke preciznosti: b_{63} sadrži bit predznaka, bitovi b_{52} do b_{62} su bitovi eksponenta, a bitovi b_0 do b_{51} pohranjuju razlomljeni dio signifikanda

Vrijednosti brojeva upisanih u ta polja određene su u skladu s tablicom 2.1. Iz tablice je vidljivo da su vrijednosti broja posebno određene:

- uz $E = 0$ (svi bitovi polja eksponenta jednaki 0) i
- uz $E = 2047$ (svi bitovi polja eksponenta jednaki 1).

Ustanovimo da postoji pozitivna i negativna nula te pozitivna i negativna vrijednost za beskonačno.

vrijednost u polju eksponenta E	signifikand s	vrijednosti broja n	objašnjenje
0	$s = 0$	$(-1)^\sigma \times 0$	pozitivna ili negativna nula
0	$s \neq 0$	$(-1)^\sigma \times (0.s) \times 2^{-1022}$	denormirani signifikand
$0 < E < 2047$	$s = 0$ ili $s \neq 0$	$(-1)^\sigma \times (1.s) \times 2^{E-1023}$	normirani signifikand
2047	$s = 0$	$(-1)^\sigma \times \infty$	pozitivno ili negativno beskonačno
2047	$s \neq 0$	*nan*	nije broj – *not a number*

Tablica 2.1. Vrijednost brojeva s pomičnom točkom dvostruke preciznosti prema standardu IEEE 745

element.hr