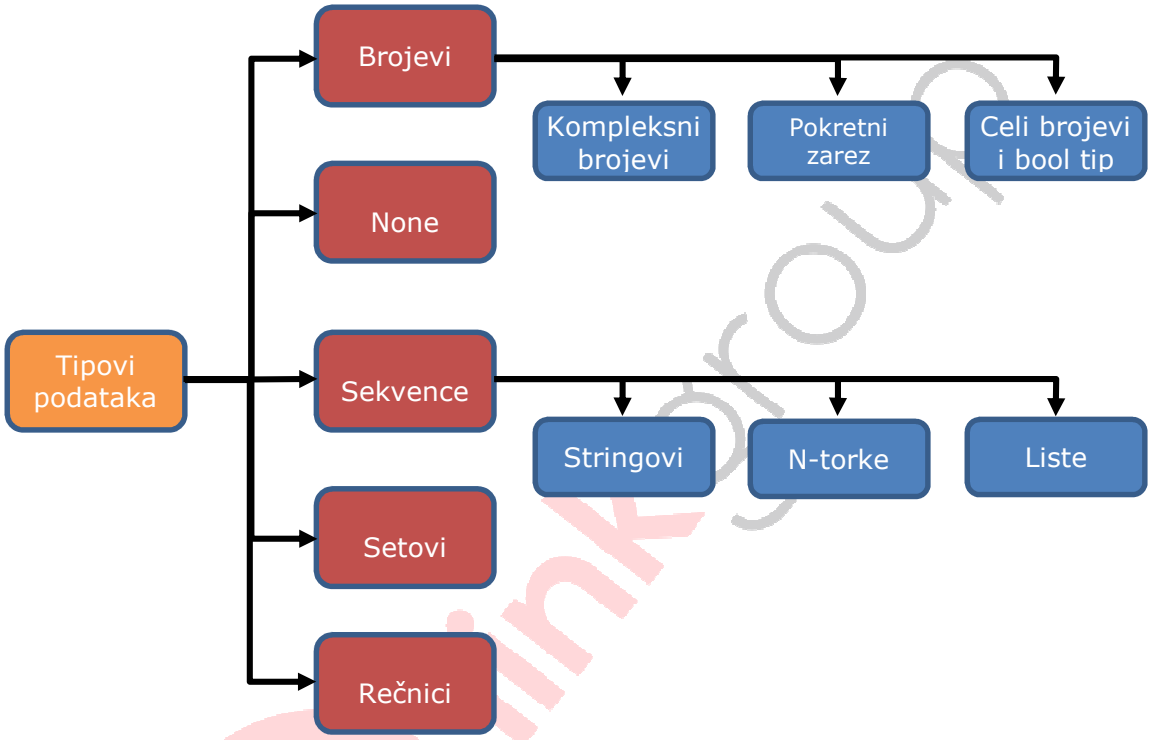
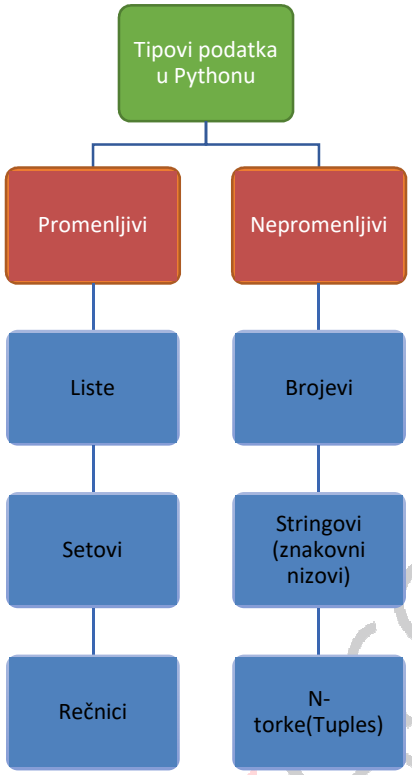
Tipovi podataka

Tipovi podataka predstavljaju skupove vrednosti i mogućih operacija nad tim vrednostima. Srećom, Python podržava dosta različitih tipova podataka koji olakšavaju rad u tom programskom jeziku. Generalna podela tipova podataka se može videti na slici 6.1:



Slika 6.1. Podela standardnih tipova podataka u Pythonu

Sa slike 6.1. se može videti da se brojevi dodatno dele na kompleksne brojeve, brojeve sa pokretnim zarezom i na cele i logičke brojeve, a da u sekvence spadaju stringovi, n-torke i liste. Pored ove podele, sve tipove podataka moguće je podeliti i na promenljive i nepromenljive.



Slika 6.2. Standardni tipovi podataka u Pythonu

Promenljivi tipovi podataka su oni koji dozvoljavaju da se njihove trenutne vrednosti menjaju, dok su nepromenljivi tipovi podataka oni kod kojih se jednom dodeljena vrednost ne može menjati.

Brojevi

O brojevima je bilo reči u prethodnim lekcijama. Koristimo ih u matematičkim operacijama, zajedno sa promenljivama kada god želimo da dodelimo i koristimo brojeve. Python razaznaje tri podvrste klase int(), a to su celi brojevi i logički bojevi, brojevi sa pokretnim zarezom i kompleksni brojevi.

Celi brojevi i logički brojevi (int i bool)

Maksimalne vrednosti celih brojeva (tip int) u Pythonu su nešto veće nego u drugim programskim jezicima zato što Python 3.x ne razlikuje podtipove kao što su *short int / long int* i zavise od arhitekture na kojoj se pokreće, ali i od samog interpretera. Zbog ovoga se za Pythonov tip *int* kaže da je proizvoljne dužine.

Logičke vrednosti mogu biti samo **True** ili **False**.

Brojevi sa pokretnim zarezom

Ovaj set brojeva se odnosi na sve one vrednosti sa decimalnim zarezom: 1.1, 3.13, 4.794. Kreira se pomoću funkcije float():

Primer brojeva sa pokretnim zarezom (float):

```
a=float(-10.14)
print(a)
```

Objašnjenje:

Realan broj možemo napraviti i ako izuzmemo float funkciju i napišem broj za tačkom: a=-10.14

Kompleksni brojevi

Kompleksni brojevi su pojam iz matematike. Ovi brojevi se sastoje iz realnog i imaginarnog dela u formatu i pišu se u formatu x+yj, gde je broj x realan deo, a y imaginaran. U Pythonu postoji podrška i za ove brojeve, tako da prevodilac razume komandu 1+2j i neće vratiti grešku. Kreiraju se pomoću ugrađene funkcije `complex(x,y)`.

Primer kompleksnih brojeva

```
b=1+2j
print(b)
d=complex(1,2)
print(d)
```

Objašnjenje:

Ispis će biti isti koji god način da odaberemo.

Poseban tip podatka: None

Ovaj tip podatka koristimo kada trenutno ne znamo kog tipa će vrednost promenljive biti ili trenutno želimo da ona ostane prazna. Definiše se isto kao i druge promenljive, pomoću operatora dodeljivanja, na sledeći način: var_a = None. Treba imati na umu da None nije isto što i False, prazan string niti 0.

Sekvence

Sekvence u Pythonu predstavljaju kolekcije indeksiranih elemenata. U sekvence spadaju i promenljivi i nepromenljivi tipovi podataka. U Pythonu, tri najkorišćenija tipa podataka koji se smeštaju u sekvence su: stringovi, liste i n-torke (tuples).

Znakovni nizovi (stringovi)

Stringovi predstavljaju uređeni niz znakova/karaktera koji je nepromenljiv pa, kao i ostale sekvence u Pythonu, podržavaju sve operacije sa indeksiranjem elemenata. Obuhvaćeni su jednostrukim navodnicima (' '), dvostrukim navodnicima (" ") ili, ako je reč o znakovnom nizu koji je ispisan u više nivoa, onda se koriste trostruki navodnici (''' ''' ili """ """). Dakle, stringovi mogu sadržati bilo koji karakter. Ako u znakovnom nizu koji je obuhvaćen jednostrukim navodnicima treba takođe da se nađe i jednostruki navodnik, onda se u tom

slučaju koristi izlazna sekvenca "\' ". Isto pravilo važi i za dvostruke i trostruke navodnike, ali se oni mogu kombinovati, pa je sintaksički sasvim ispravan sledeći izraz: s_three = '"""test"""'.

Naredna tabela prikazuje taj specifičan slučaj:

Primer korišćenja izlazne sekvence u stringu

Radno okruženje

```
s = 'test'
print(s)
s_two = 'te\'st'
print(s_two)
s_three = "te\"st"
print(s_three)
s_four = """te\"""st"""
print(s_four)
```

Kao što vidite, rezultat prikazuje kako da implemetiramo na više načina da navodnik bude unutar stringa.

Izlazne sekvence su posebni karakteri koji imaju specifičnu svrhu kada se koriste unutar stringa. Neki od najkorišćenijih su:

Izlazne sekvence	
Karakter	**Značenje**
"\b"	briše karakter ulevo
"\n"	prelazak u novi red
"\t"	horizontalni tabulator
"\\"	izlazna sekvenca za obrnutu kosu crtu

Tabela 6.1. Izlazne sekvence

Liste

Liste predstavljaju promenljive sekvence. Elementi listi mogu biti različiti tipovi podataka. Pristupanje jednom ili više elemenata liste se vrši uz pomoć indeksa tog elementa. Nakon dodavanja novih elemenata, redosled u listama se ne menja. Indeksi elemenata u listama, a i u svim drugim sekvencama Pythona počinju od nule. Dakle, nulti element svake sekvence je prvi element iste.

Prazne liste se kreiraju koristeći uglaste zagrade.

Ako pokušamo da pristupimo indeksu koji ne postoji u listi, dobićemo IndexError tip greške. Na sledećoj ilustraciji je primer kreiranja jedne liste i rukovanja njome.

Radno okruženje

```
lst = ['element1', 'element2', 3]

print(lst[0]) # accessing the first element of the list
print(lst[1:]) # access to all elements except the first

 print(lst[-1]) # accessing the last element of the list

print(lst[3]) # accessing an index that does not exist gives an error
```

Objašnjenje:

Ako napišemo lst[0] izdvajamo prvi element liste, sa lst[1:] izdvajamo sve element osim prvog odn. elemente od pozicije 1 do kraja, sa lst[-1] izdvajamo element na poslednjoj poziciji i ako pokušamo da pristupimo elementu na poziciji 3 odn. na poziciji na kojoj se ne nalazi nijedan element dolazi do greške.

Navešćemo i neke od korisnih metoda Python listi koje će se bazirati na listi iz prethodnog primera (lst = ['element1', 'element2', 3]). Ove metode menjaju samu listu trenutno, pa nije potrebno dodeljivati rezultat tih metoda novoj promenljivoj.

- list.append(element) – dodaje novi element na kraj liste.
  lst.append('element3') -> lst = ['element1', 'element2', 3, 'element3']
- list.insert(index, element) – dodaje novi element na zadati indeks i tako pomera sve elemente (od tog elementa) za jedno mesto udesno.
  lst.insert(0, 'element0') -> lst = ['element0','element1', 'element2', 3, 'element3']
- list.extend(list_two) – dodaje elemente list_two listi nad kojom je metoda extend() pozvana.
  lst.extend(['element4', 'element5']) -> lst = ['element0','element1', 'element2', 3, 'element3', 'element4', 'element5']
  Takođe, liste se mogu spajati i operatorom (+) na sledeći način: lst + ['element4', 'element5'], što nam daje isti rezultat.
- list.index(element) – vratiće nam index na kojem se traženi element nalazi; ako zadamo nepostojeći element, dobićemo ValueError.
- list.remove(elem) – obrisaće prvi primerak traženog elementa.
- list.pop(index) – ako se ne prosledi parametar indeks, obrisaće se poslednji element, u suprotnom obrisaće se element koji se nalazi na prosleđenom indeksu.
- list.sort() – sortira datu listu po rastućem redosledu.
- list.reverse() – okreće trenutni redosled u listi; ista funkcionalnost se postiže i izrazom: lst[::-1].

N-torke (Tuples)

N-torke su jako slične listama, sa samo par razlika. Prva i najbitnija razlika je da spadaju u nepromenljive tipove podataka, što znači da kada se jednom doda element, ne može se menjati. Kada se jednom definiše n-torka, nije moguće ponovo dodavati nove elemente. Za definisanje n-torke se koriste obične zagrade, pa bi tako naš primer sa listama prebačen u n-torke izgledao ovako: new_tuple = ('element1', 'element2', 3, 'element3')

Sama implementacija n-torki je mnogo brža nego implementacija listi.

Primer n-torki

```
new_tuple = ('element1', 'element2', 'element3')
print(new_tuple)
```

Ispisaće se: `('element1', 'element2', 'element3')`

Setovi

Setovi su neuređena vrsta kolekcija bilo kojeg tipa vrednosti koje ne sadrže duplikate i promenljivi su. Tipovi podataka kao što su stringovi, liste i n-torke se mogu prebaciti u setove, ali će duple vrednosti u njima biti odstranjene. Prebacivanje tipova podataka u setove se vrši naredbom set(). Ili, ako želimo da definišemo set, koristimo vitičaste zagrade {}, ali u tom slučaju ne smemo ih definisati kao prazne, jer onda dobijamo tip podatka rečnik.

Primer setova

```
my_set = {1, 2, 3}
print(my_set)

my_set2 = set([1,2,3,3])
print(my_set2)
```

Objašnjenje:

Na koji god način da napravimo set, ili uz pomoć vitičastih zagrada ili funkcijom set rezultat je isti.

Rečnici

Tip podataka *rečnik* u programskom jeziku Python predstavlja kolekciju koju čini grupa ključ–vrednost parova. Elementi u rečniku su indeksirani pomoću samih ključeva, ne indeksa. Pravilo je da ti ključevi moraju biti jedinstveni, što znači da dva ista ključa ne mogu postojati. Inicijalizuju se na način prikazan na sledećoj ilustraciji:

Primer inicijalizacije rečnika i pristupanje rečniku po ključu
Kod

`dictionary = {"key1":'element1', "key2":'element2', "key3":3}`
`print(dictionary)`
`print(dictionary['key1'])`
`print(dictionary['key3'])`

Tabela 6.2. Inicijalizacija rečnika i pristupanje rečniku po ključu

Probajte iskucati kod i pogledajte rezultat:

Radno okruženje

Neke od korisnih metoda za rad sa rečnicima su:

- dictionary.keys() – vraća listu u kojoj se nalaze samo ključevi rečnika;
- dictionary.values() – vraća listu vrednosti bez ključeva;
- dictionary.items() – vraća listu ključ–vrednost parova u vidu n-torki (svaki od tih elemenata liste predstavlja par n-torki);
- dictionary.pop('key1') – izbacuje i ključ i vrednost iza zadatog ključa;
- dictionary.clear() – briše sve ključeve i vrednosti iz rečnika.

Pitanje

Kog je tipa promenljiva ako smo je inicijalizovali kao my_variable = (2,3, 'key1')?

- rečnik
- ceo broj
- **n-torka**

Objašnjenje:

Tačan odgovor je da smo na taj način inicijalizovali promenljivu my_variable kao n-torku jer smo koristili obične zagrade (). Da je u pitanju rečnik, koristili bismo {}.

Konverzija tipova

Često se u praksi srećemo sa slučajem da imamo podatak jednog tipa, a potrebno nam je da taj tip pretvorimo u drugi. Recimo da na ulazu u program imamo broj koji je u string reprezentaciji ('3'). Dakle ako bismo pokušali matematičke operacije nad njim – ne bismo dobili željene vrednosti. Da bismo mogli da ga koristimo kao ceo broj (tip int), moramo ga i pretvoriti u tip int. Ovo se zove eksplicitno pretvaranje tipova. Pretvaranje tipova u Pythonu se vrši pomoću ugrađenih funkcija i to:

Funkcije za pretvaranje tipova		
Funkcija	**Upotreba**	**Rezultat**
int()	int('123'); int(42.23)	123; 42;
float()	float(1); float(False)	1.0; 0.0;
complex()	complex(1); complex(1.5)	1+0j; 1.5+0j;
bool()	bool(1); bool(''); bool(0.0)	True; False; False;
str()	str(10); str(True); str(int())	'10'; 'True', '0';

Tabela 6.4. Funkcije za pretvaranje tipova

Napomena

Za proveru tipa bilo kojeg podatka koristimo funkciju type(). Recimo, ako imamo broj sa pokretnim zarezom (float) f = 3.14, naredbom type(f) dobićemo ispis `<type 'float'>`.

U narednoj tabeli se nalazi rezimirani, uporedni prikaz prethodno obrađenih tipova podataka i njihovih opisa:

Tipovi podatka		
Ime	**Tip (type())**	**Opis**
celi brojevi	int	celi brojevi: 3, 4, 10, -40
logički tip	bool	logičke vrednosti: True i False
pokretni zarez	float	decimalni brojevi: 3.14, 20.0, -3.14
kompleksni	complex	kompleksni brojevi: 3+1j, complex(3,1)
none	NoneType	kada trenutno ne želimo da dodelimo vrednost: a = None
stringovi	str	tekstualni podatak: "Marco", "2020", "3.14"
liste	list	uređena promenljiva sekvenca heterogenih elemenata: lst[-3.14, None, "String"]
n-torke	tuple	uređena nepromenljiva sekvenca heterogenih elemenata: tup=(-3.14, "String", 2020)
setovi	set	neuređena kolekcija jedinstvenih elemenata: set([1,2,3,3])
rečnik	dict	neuređena kolekcija ključ:vrednost elemenata: d={1:'Marco', 2:'Steven'}

Tabela 6.5. Rezimirani prikaz tipova podataka

Rezime

- Tipovi podataka predstavljaju skupove vrednosti i mogućih operacija nad tim vrednostima.
- Tipovi podataka u Pythonu se dele na promenljive i nepromenljive.
- Promenljive tipove podataka čine: liste, setovi, rečnici.
- Nepromenljive tipove podataka čine: brojevi, stringovi i n-torke.
- Brojevi se dele na cele brojeve (tip int), logičke (tip bool), sa pokretnim zarezom (tip float), kompleksne brojeve (tip complex), kao i poseban tip podatka – None.
- Sekvence se dele na stringove (tip string, nepromenljiv), liste, (tip list, promenljiv), n-torke (tip tuple, nepromenljiv).
- Elementima sekvence se može pristupiti uz pomoć indeksa tog element (pozicije u sekvenci).
- Kolekcije se dele na setove (tip set, promenljiv) i rečnike (tip dict, promenljiv).
- Konverzija jednog tipa podatka u drugi se vrši pomoću ugrađenih funkcija str() – za stringove, int() – za cele brojeve, float() – za brojeve sa pokretnim zarezom, complex() – za kompleksne brojeve i bool() – za logički tip.