Komentiranje

`#`	početak linijskog komentara

Osnovni tipovi varijabli / objekata

`str`	(string), (uređeni) niz znakova
`int`	(integer), cijeli broj
`float`	broj s pomičnim zarezom
`bool`	(boolean), logička varijabla **True** ili **False**

Složeni tipovi varijabli / objekata

`range`	raspon - uređen (indeksiran) niz integera
`list`	lista - uređena (indeksirana) podatkovna kolekcija (niz objekata)

Konverzije varijabli

`str(object)`	konverzija u string
`int(object)`	konverzija u integer
`float(object)`	konverzija u float
`bool(object)`	konverzija u boolean
`list(iterable)`	konverzija (podatke kolekcije) u listu

Funkcije s objektima

`len(`*`object`*`)`	vraća broj znakova u objektu
`print(`*`object`*`)`	ispisuje objekt (ne vraća ništa)
`type(`*`object`*`)`	ispisuje tip objekta
`id(`*`object`*`)`	ispisuje identitet objekta (adresu u memoriji) *

* Svi objekti u Pythonu imaju svoj jedinstveni ID, koji se dodjeljuje objektu kada se objekt **kreira**.
ID objekta je adresa u memoriji i bit će drugačija svaki put kad se pokrene program.
Objekti kojima se vrijednosti mogu mijenjati bez promjene identiteta zovu se promjenljivi (engl. mutable) objekti, a oni kojima se vrijednost ne može mijenjati bez stvaranja novog objekta istog tipa zovu se nepromjenljivi (engl. immutable) objekti.
Promjena vrijednosti objekta obično se događa operatorom pridružbe ili djelovanjem metode na vrijednost objekta.

Ostale funkcije

`range([start], end, [step])`	vraća sekvencu (raspon) brojeva od "start" do "end", s razlikom od "step"

Deklaracija stringova

`"text"`	klasičan način zapisa stringa
`"""text"""`	zapis stringa koji omogućuje prijelaz u novi red
`f"text {value} text"`	formatirani string koji evaluira vrijednost u vitičastim zagradama

Prilikom deklaracije, mogu se koristiti i jednostruki navodnici! Stringovi nisu identiteno-promijenjivi...

```
» string1 = "Ivan"
» string2 = string1
» id(string1) == id(string2)
"True"
» string1 = "Marko"
» string2
"Ivan"
» id(string1) == id(string2)
"False"
```

... što znači da promjena jedne ne veže promjenu druge.

Osnovne operacije sa stringovima

`"text"[0]`	indeksiranje stringova (`"t"`)
`"text"[0:2:1]`	komad stringa (počevši od 0 do 2 (ne uključujući 2), s korakom 1) (`"te"`)
`"abc" + "def"`	povezivanje stringova (`"abcdef"`)
`"abc" * 2`	umnožavanje stringova (`"abcabc"`)

Posljednji član stringa ima indeks -1, predzadnji -2, itd.

Funkcije sa stringovima

`input(`*`prompt`*`)`	traži vrijednost inputa u prompt (tipa string)

Metode nad stringovima

string`.isnumeric()`	✓	ispisuje **True** ako je string broj, a **False** ako nije
string`.find(`*value, [start], [end]*`)`	✓	vraća indeks prvog pojavljivanja "value"
string`.index(`*value, [start], [end]*`)`	✓	slično kao find, ali u slučaju nepronalaska izbacuje grešku
string`.rfind(`*value, [start], [end]*`)`	✓	vraća indeks zadnjeg pojavljivanja "value"
string`.rindex(`*value, [start], [end]*`)`	✓	slično kao rfind, ali u slučaju nepronalaska izbacuje grešku
string`.lstrip(`*[character]*`)`	✓	uklanja niz "character" (ili razmak) s lijeve strane **kopije** stringa
string`.rstrip(`*[character]*`)`	✓	uklanja niz "character" (ili razmak) s desne strane **kopije** stringa
string`.strip(`*[character]*`)`	✓	uklanja niz "character" (ili razmak) s lijeve i desne strane **kopije** stringa
string`.replace(`*oldvalue, newvalue, [count]*`)`	✓	mijenja "oldvalue" za "newvalue" (u prvih "count" pojavljivanja) **kopije** stringa
string`.count(`*value, [start], [end]*`)`	✓	ispisuje broj ponavljanja "value"
string`.lower()`	✓	ispisuje **kopiju** stringa kojemu su sva slova mala
string`.upper()`	✓	ispisuje **kopiju** stringa kojemu su sva slova velika
string`.capitalize()`	✓	ispisuje **kopiju** stringa kojemu je prvo slovo veliko
string`.split(`*delimiter*`)`	✓	ispisuje **novu listu** nastalu razdvajanjem **kopije stringa** po graničniku
delimiter`.join(`*iterable*`)`	✓	ispisuje **novi string** nastao spajanjem elemenata **kopije iterabla** po graničniku

✗ - metoda **nema** povrat tj. `return`
✓ - metoda **ima** povrat tj. `return`

Aritmetički operatori

+	zbrajanje
-	oduzimanje
*	množenje
**	potenciranje
/	dijeljenje
//	cjelobrojno dijeljenje
%	ostatak cjelobrojnog djeljenja

Operatori dodjele

=	a = 5
+=	a = a + 5
-=	a = a - 5
*=	a = a * 5
**=	a = a ** 5
/=	a = a / 5
//=	a = a // 5
%=	a = a % 5

Operatori usporedbe vrijednosti

`==`	jednako
`!=`	nije jednako
`>`	veće
`>=`	veće ili jednako
`<`	manje
`<=`	manje ili jednako

Izlaz može biti `True` ili `False`

Operatori usporedbe adresa u memoriji

`is`	jednakost adresa u memoriji
`is not`	nejednakost adresa u memoriji

Operator provjerava `id(object1) == id(object2)`
Izlaz može biti `True` ili `False`

Logički operatori

`and`	istinito ako su sve tvrdnje točne
`or`	istinito ako je barem jedna tvrdnja točna
`not`	inverzija (negacija) istinitosti tvrdnje

Izlaz može biti `True` ili `False`

Operatori članstva

`in`	točnost postojanja člana u sekvenci
`not in`	netočnost postojanja člana u sekvenci

Izlaz može biti `True` ili `False`

Neistinite (lažne) vrijednosti

`False`	definicijska "laž"
`None`	"vrijednost" varijable bez vrijednosti (a = None)
`0`	0 tipa cijelog broja
`0.0`	0 tipa broja s pomičnim zarezom
`""`	prazni string (u bilo kojem formatu)
`[]`	prazan niz
`()`	prazna n-torka
`{}`	prazan rječnik
`set()`	prazan skup
`range(0)`	prazan raspon

Grananje

`if` *`<condition 1>`*`:`	postavljanje prvog uvjeta
`<code block 1>`	kod koji se izvršava ako je prvi uvjet zadovoljen
`elif` *`<condition 2>`*`:`	postavljanje drugog uvjeta
`<code block 2>`	kod koji se izvršava ako je drugi uvjet zadovoljen
`else:`	pokrivanje svih ostalih uvjeta
`<code block 3>`	kod koji se izvršava ako prvi i drugi uvjet nisu zadovoljeni

`if`, `elif` i `else` moraju koristiti iste indentacije!
`<code block 1>`, `<code block 2>` i `<code block 3>` moraju koristiti iste indentacije!

Zadovoljavanje bilo kojeg od uvjeta tj. "grane" rezultira izlaskom iz "stabla" i nastavljanjem izvršavanja daljnjeg koda.

Ni `elif` ni `else` sekcije nisu obavezne.
... `elif` sekcija nije nužna ako se kod dijeli u samo dvije grane.
... `else` sekcija nije nužna u slučajevima tipa *else-do-nothing*.

`while` petlja

`while` *<condition>*:	`<condition>` je uvjet iteracije
<code block>	kod koji se izvršava u svakoj iteraciji

`while` petlja se izvršava kad nije unaprijed poznat broj potrebnih iteracija.

Petlja se izvršava dok je uvjet petlje zadovoljen. Kako bi završila, unutar same petlje mora doći do izmijene uvijeta.

`for` petlja

`for` *<iterator>* `in` *<iterable>*:	`<iterator>` je jedinični element strukture tj. podatkovne kolekcije `<iterable>`
<code block>	kod koji se izvršava u svakoj iteraciji

`for` petlja se izvršava kad je unaprijed poznat broj potrebnih iteracija.

U svakoj iteraciji iterator dobiva novu vrijednost. Petlja završava tek kad završe sve iteracije.

Struktura	Iterable	Iterator
Range	`range(3, 20, 2)`	`3, 5, 7, 9, 11, 13, 15, 17, 19`
String	`"apple"`	`a, p, p, l, e`
List	`["apple", "banana", "cherry"]`	`apple, banana, cherry`
Tuple	`("apple", "banana", "cherry")`	`apple, banana, cherry`
Set	`{"apple", "banana", "cherry"}`	`apple, banana, cherry`
Dictionary	`{"brand": "Ford", "model": "Mustang", "year": 1964}`	`brand, model, year`

Naredba `break`

`for` *<iterator>* `in` *<iterable>*:	
`if` *<condition>*:	postavljanje uvjeta koji prekida petlju
`break`	prekid petlje
<code block>	kod koji se (u suprotnom) izvršava u svakoj iteraciji

`break` služi kako bi se **prekinulo** izvršavanje **petlje**.

Naredba `continue`

`for` *<iterator>* `in` *<iterable>*`:`	
`if` *<condition>*`:`	postavljanje uvjeta kojim se preskače trenutna iteracija
`continue`	preskok iteracije
<code block>	kod koji se (u suprotnom) izvršava u svakoj iteraciji

`continue` služi kako bi se **preskočilo** izvršavanje **iteracije**.

Definiranje funkcije

`def` *<name>*`(`*<parameters>*`):`	definiranje imena i postavljanje parametara (odvojenih zarezom) koji se koriste u bloku koda
<code block>	blok koda kojeg će funkcija izvršavati svakim pozivanjem
`return` *<value>*	povrat (rezultat) funkcije

`return` vraća rezultat funkcije i **izlazi iz funkcije**, slično kao i `break`
funkcija ne mora imati `return` ako ne vraća rezultat, npr. ako radi samo `print`
funkcija ne mora imati `parameters` ako nema ulazne podatke.

`z = f(x,y)`
`z -` *<value>*
`f -` *<name>*
`x,y -` *<parameters>*

Pozivanje funkcije

<name>`(`*<arguments>*`)`	pozivanje funkcije koja nije imala `return`
<variable>`=`*<name>*`(`*<arguments>*`)`	pozivanje funkcije koja je imala `return` i pohranjivanje njenog rezultata u `variable`

`a = f(2,5)`
`a -` *<variable>*
`f -` *<name>*
`2,5 -` *<arguments>*

Parametri funkcije

`def student(ime, prezime="Horvat", godina=1):` `  print(ime, prezime, "je", godina, '.  godina')`	definiranje funkcije **student** s 1 obaveznim i 2 opcionalna parametara ispis funkcije

Prvo se definiraju svi obavezni parametri, a zatim svi opcionalni parametri.

Pozicijski argumenti

`student("Ivan")`	Ivan Horvat je 1. godina
`student("Ivan", "Kovač", 2)`	Ivan Kovač je 2. godina
`student("Ivan", "Kovač")`	Ivan Kovač je 1. godina
`student("Ivan", 2)`	Ivan 2 je 1. godina

Pozicijski argumenti zahtjevaju definirani redoslijed i dodjelujuje se s lijevo na desno.
"Nespareni" argumenti dobivaju podrazumjevanu vrijednost.

Argumenati ključnih riječi

`student(ime="Ivan")`	Ivan Horvat je 1. godina
`student(ime="Ivan", godina=2)`	Ivan Horvat je 2. godina
`student(prezime="Kovač", ime="Ivan")`	Ivan Kovač je 1. godina

Argumenti ključnih riječi ne zahtjevaju definirani redoslijed.
Nedefinirani argumenti dobivaju podrazumjevanu vrijednost.

Miješani argumenti

`student("Ivan", godina=2)`	Ivan Horvat je 2. godina
`student("Ivan", "Kovač", godina=2)`	Ivan Kovač je 2. godina

Pozicijski argumenti se definiraju prije argumenata ključnih riječi.

Primjeri krivog pozivanja funkcija

`student()`	pozivanje funkcije bez obaveznih argumenata
`student(ime="Ivan", 2)`	definiranje argumenta bez ključne riječi nakon onog s ključnom riječi
`student("Ivan", 2, prezime="Kovač")`	dvostruko definiranje argumenta (pozicija 2 i ključna riječ "prezime")
`student(kolegij="Matematika")`	definiranje nepostojećeg parametra

Vidljivost varijabli

- **globalne** varijable definirane su u glavnom tijelu programa i vidljive su svim funkcijama
  - varijable definirane unutar neke petlje (npr. iteratori) vidljive su i izvan te petlje
  - varijable definirane u bloku koda unutar grananja vidljive su i izvan grananja
- **lokalne** varijable definirane su unutar neke funkcije i vidljve su toj funkciji i njenim pod-funkcijama
  - varijable definirane unutar funkcije mogu se postaviti globalnima korištenjem ključne riječi `global`

`global` *<variable>*	postavljanje `<variable>` globalnom
<variable>`=`*<value>*	definiranje vrijednosti varijable

Ako je određena varijabla definirana globalno, a zatim i više puta lokalno (rekurzivno) unutar funkcija, njena vrijednost u najunutarnjijoj funkciji imat će "najlokalniju" vidljivu vrijednost.

Liste

list `= [1, 2, 3]`	definiranje liste
list `= [[1, 2, 3], [1, 2, 3], [1, 2, 3]]`	definiranje ugniježđene liste

Liste su **uređene** strukture podataka što znači da je postoji redoslijed članova.
Članovi liste ne moraju biti isti tipovi podataka.
Liste su identiteno-promijenjive...

```
» list1 = [12, 9, 3, 7]
» list2 = list1
» id(list1) == id(list2)
"True"
» list1.append(1)
» list2
[12, 9, 3, 7, 1]
» id(list1) == id(list2)
"True"
```

... što znači da promjena jedne veže promjenu druge.

Osnovne operacije s listama

`list[index]`	indeksiranje lista
`list[[start]:[end][:step]]`	komad liste
`list[index] = value`	postavljanje nove vrijednosti člana niza
`list[[start]:[end][:step]] = list`	postavljanje nove vrijednosti komada liste s drugom listom (brisanje i umetanje)
`[1, 2, 3] + [4, 5, 6]`	povezivanje listi (`[1, 2, 3, 4, 5, 6]`)
`[1, 2, 3] * 2`	umnožavanje listi (`[1, 2, 3, 1, 2, 3]`)
`del list[start[:end][:step]]`	briše član ili komad liste
`color = [255, 43, 19]` `red, green, blue = color`	definiranje liste #... ... i raspakiravanje - pridruživanje po elemantima
`item = [4, "Pizza", "Plain", 16.98]` `quantity, *others, price = item`	definiranje liste ##... ... i raspakiravanje - pridruživanje po elemantima

Posljednji član liste ima indeks -1, predposljednji -2, itd.
Lista se može jednostavo rastaviti ako ima jednak broj elemenata.
Lista se može "složeno" rastaviti tako da jedan element (označen s *) sakuplja sav višak.

Metode nad listama

`list.append(object)`	✗	dodavanje objekta na kraj **originalne** liste
`list.extend(iterable)`	✗	dodavanje rastavljene iterable na kraj **originalne** liste
`list.insert(index, object)`	✗	dodavanje objekta ispred člana pod indeksom na **originalnoj** listi
`list.clear()`	✗	prazni **originalnu** listu
`list.remove(value)`	✗	briše prvi član u **originalnoj** listi koji ima vrijednost *value*
`list.pop([index])`	✓	uklanja zadnji član u **originalnoj** listi (član pod indeksom) i vraća uklonjenu vrijednost
`list.count(value)`	✓	ispisuje broj ponavljanja "value"
`list.reverse()`	✗	invertira **originalnu** listu
`list.sort([reverse=True])`	✗	(naopako) sortira **originalnu** listu
`list.copy()`	✓	kopira listu (korisno jer su liste identiteno-promjenljive)

✗ - metoda **nema** povrat tj. `return`
✓ - metoda **ima** povrat tj. `return`