1.1 Uvod

1.1.1 Zašto Python?

Python je interpreterski, interaktivni, objektno orjentirani programski jezik, kojeg je 1990. godine zamislio Guido van Rossum. Već do konca 1998., Python je imao bazu od 300.000 korisnika, a od 2000. već su ga prihvatile ustanove kao *MIT*, *NASA*, *IBM*, *Google*, *Yahoo* i druge. Python ne donosi neke nove revolucionarne značajke u programiranju, već na optimalan način ujedinjuje sve najbolje ideje i načela rada drugih programskih jezika. On je jednostavan i snažan istodobno. Više nego drugi jezici on omogućuje programeru više razmišljanja o problemu nego o jeziku. U neku ruku možemo ga smatrati hibridom: nalazi se između tradicionalnih skriptnih jezika (kao što su *Tcl*, *Schema* i *Perl*) i sistemskih jezika (kao što su *C*, *C++* i *Java*). To znači da nudi jednostavnost i lako korištenje skriptnih jezika (poput *Matlab*-a), uz napredne programske alate koji se tipično nalaze u sistemskim razvojnim jezicima. Python je besplatan (za akademske ustanove i neprofitnu upotrebu), open-source software, s izuzetno dobrom potporom, literaturom i dokumentacijom.

1.1.2 Značajke programskog jezika Python

Interpretacija međukôda

Python kôd sprema se u tekst datoteke koje završavaju na `.py`. Program kompilira kôd u niz bytecode-ova koji se spremaju u `.pyc` datoteke koje su prenosive na bilo koje platforme gdje se mogu izvoditi interpretacijom tog međukôda. Python je napisan u *ANSI C* programskom jeziku i raspoloživ je za cijeli niz strojeva i operacijskih sustava uključujući *Windows*, *Unix/Linux* i *MacIntosh*. To znači da će se napisani programski kôdovi moći izvoditi na svim najpoznatijim tipovima računala, te čak i na mnogim mobilnim telefonima, jer se Python ugrađuje i u njihove operacijske sustave.

Jezik visoke razine

Python ima ugrađene tipove podataka visoke razine kao što su liste, n-terci i rječnici, što omogućuje lako i brzo programiranje (fast-prototype programming).

Interaktivnost

Python se može izvoditi na najnižoj razini operacijskog sustava (tzv. terminalski ili konzolni rad, DOS okoliš), ili ugodnije - u različitim okruženjima: IDE (integrated development environment) koji dolazi zajedno s instalacijom Python programskog jezika (slika 1.1), IPython, JPython i sl. Za razvitak programskog koda ili kroz skriptne editore (PyScripter, Eclipse). Za razvitak programa najlakši je interaktivni način rada u kojem se programski kôd piše naredbu za naredbom. Ne postoji razlika u razvojnom i izvedbenom okolišu: u prvom se izvodi naredba za naredbom, a u drugom odjednom čitava skripta.

Čista sintaksa

Sintaksa jezika je jednostavna i očevidna. Uvlake zamjenjuju posebne znakove za definiranje blokova kôda, pa je napisani program vrlo pregledan i jednostavan za čitanje.

Napredne značajke jezika

Python nudi sve značajke očekivane u modernom programskom jeziku: objektu orijentirano programiranje s višestrukim nasljeđivanjem, dohvaćanje izuzetaka ili iznimki (engl. *exception*), redefiniranje standardnih operatora, pretpostavljene argumente, prostore imena (engl. namespaces), module i pakete.

Proširivost

Python je pisan u modularnoj C arhitekturi. Zato se može lako proširivati novi značajkama ili API-ima. (engl. *application programming interface*).

Bogate knjižnice programa

Pythonova knjižnica (engl. *library*), koja uključuje standardnu instalaciju, uključuje preko 200 modula, što pokriva sve od funkcija operacijskog sustava do struktura podataka potrebnih za gradnju web-servera. Glavni Python web site (*www.python.org*) nudi sažeti index mnogih Python projekata i različitih drugih knjižnica.

Potpora

Python ima veliku entuzijastičku zajednicu korisnika koja se svake godine udvostručuje.

1.2 Naš prvi Python program

Cilj programa (izvedenog programskog kôda) je riješiti neku zadaću, od one zabavne do vrlo teške i zahtjevne. Neka se za početak uzme ona lakša i zamisli problem u kojem stroj iz nekog teksta (u našem slučaju to će biti stih iz čuvene Gundulićeve pjesme o slobodi) uzima neku riječ (ali je ne prikazuje), a korisnik onda pogađa, upisuje, koja je to riječ. Da bi igra bila zanimljivija, računalo pomaže igraču, (ako nije pogodio), je li zamišljena riječ leksikografski *iza* ili *ispred* one koju je igrač upisao ('A' je najniže, prvo, a 'Ž' najviše, zadnje). Na primjer, riječ 'sloboda' je leksikografski iza 'draga' jer je 's' iza 'd' (ima veći ASCII kod, broj). Ako je zamišljena riječ ispred one koju je igrač zamislio, program poručuje da se puca naniže. Na koncu, kad se riječ pogodi, ispisuje se čestitka igraču i broj pokušaja u kojem je uspjeh postignut. Cilj igre je, dakako, u što manje pokušaja pogoditi riječ koju je stroj odabrao:

Program 1.1 — Izgled Python programa. –
Primjetite: ključne riječi Python koda (naredbe) prikazane su crvenom bojom, komentari ljubičastom, a stringovi plavom bojom

```
1  import nltk
2  from random import randint
3
4  stih=u"""O lijepa, o draga, o slatka slobodo!
5  Dare u kom' sva blaga višnji nam Bog je do! """
6  print stih,'\n',22*'-',u'(Ivan Gundulić)',4*'-'
7
8  ## rjecnik sadrži popis različitih riječi (rijeci) iz stiha
9  rijeci=nltk.regexp_tokenize(stih.lower(), u"[a-z']+")
10 rjecnik=list(set(rijeci))
11
12 ## inicijalizacija varijabli
13 gotovo=False; korak=0
14 broj=randint(0, len(rjecnik)-1)
```

```
15 zamislio=rjecnik[broj]
16
17 ## glavna petlja
18 while not gotovo:
19     x=raw_input(u"Pogodi zamišljenu riječ:")
20     x=x.lower()
21     if x not in rjecnik:
22         print u"Oprez kod tipkanja! '%s' nije u stihu!" % x
23         continue;
24     if x==zamislio:
25         print u'Čestitamo!'
26         gotovo=True;
27     elif x<zamislio:
28         print (u'Zamišljeno je leksikografski iza "%s". Pucaj
               naviše!'%x)
29     else:
30         print (u'Zamišljeno je leksikografski ispred "%s".
               Pucaj naniže!'%x)
31     korak+=1
32 print 'Pogodak iz %d. puta.' % korak
```

1.2.1 Kako čitati Python program

Iako će se o svim dijelovima Python programa puno govoriti, ovdje će se po linijama prvog programa dotaknuti osnovni pojmovi:

1. linija: učitava se (engl. *import*, uvlači) modul NLTK (**N**atural **L**anguage **T**ool**K**it) da bi se u programu mogla iskoristiti funkcija/metoda za izvlačenje/ekstrakciju riječi iz teksta. NLTK ima više desetaka metoda od kojih ćemo mnoge često koristiti u ovoj knjizi.

2. linija: programa uvlači iz modula `random` funkciju `randint()` s kojom će se generirati slučajan cijeli broj. Dok je prva linija programa učitala u memoriju računala sve funkcije koje modul/biblioteka NLTK ima, prema sintaksi druge linije učitana je samo jedna, ona koja se željela koristiti. Programeru se preporuča baš takav način pisanja programa, kako se memorija ne bi opteretila nepotrebnim funkcijama, niti usporio rad kod njihovog učitavanja.

4-5: U varijablu *stih* sprema se predivan Gundulićev stih, a zahvaljujući *Unicode* zapisu (rječica '*u*' ispred stringa) može imati i sve dijakritičke znakove hrvatskoga jezika (ćčđšž).

6. linija: naredbom *print()* ispisuje se stih i podvlači crtom s imenom autora.

8. linija: svaki znak desno od znaka '#' je komentar u programu koje Python interpreter ne uzima u obzir. Komentari služe za lakše čitanje programskog koda i preporuka je stavljati ih što više u programski kod.

9. linija: u varijablu '*rijeci*' sprema se rezultat koji daje funkcija *regex_tokenize()* koja se kao metoda nalazi u modulu NLTK. Svaka funkcija ima svoje ime iza kojeg slijedi otvorena i zatvorena zagrada. Unutar tih zagrada mogu se nalaziti i argumenti (u ovom slučaju to su dva argumenta: *stih.lower()* metoda i *unicode* string).

10. linija: u varijablu '*rjecnik*' sprema se rezultat koji daje/vraća funkcija *list()*, a ona ima funkciju *set()* kao svoj ulazni argument.

13-15: kao što u komentaru linije 12 piše, slijedi inicijalizaciji varijabli ('*gotovo*', '*broj*' i '*zamislio*'). Varijabla '`gotovo`' postavlja se u stanje neistine (engl. *False*), varijabla '`korak`' inicijalizira se sa `0`, a u varijablu '`zamislio`' sprema se riječ koja je slučajnim brojem (između 0 i broja riječi) dohvaćena iz rječnika svih riječi dobivenih iz stiha. Dvije

ili više naredbi na istoj programskoj liniji odjeljuju se znakom ';'.

18. linija: Kako u komentaru 17. linije piše, od ovog mjesta počinje programska petlja `while` sve dok varijala `gotovo` ne postane istinita (`True`), a to će se dogoditi tek kad varijabla '`zamislio`' bude jednaka vrijednosti varijable '`x` koju igrač puni tipkanjem znakova preko funkcije *raw_input()*.

19-30: funkcija `row_input()` služi za učitavanje korisnikovih znakova preko tipkovnice (nakonš to ispiše string upisan u njenom prvom argumentu), a `print` ispisuje znakove na zaslon računala. Programsko upravljanje s naredbama '*ako/akoinače/inače*' ('*if/elif/else*') na linijama 21., 24., 27. i 29. izvodi se s obzirom na korisnikov odgovor. U slučaju točnog odgovora ispisuje se čestitka (linija 25.), postavlja varijabla *gotovo* (linija 26.) i izlazi iz programske petlje (linija 32.).

32. linija: Izlaskom iz petlje, ispisuje se varijabla '*korak*' - koliko je koraka trebalo igraču dok postigne pogodak. Varijabla '*korak*' povećava se svakim obilaskom petlje (linija 31.). Treba primjetiti kako u Pythonu ne postoje oznake početka i konca bloka naredbi koje se izvršavaju unutar programske petlje, nego se to ostvaruje uvlakama (engl. *identation*). Na taj način programer je prisiljen pisati strukturirani kôd, lagan za čitanje. Primjetite također kako sve upravljače naredbe (`while`, `if`, `elif`, `else`) završavaju znakom dvotočke (':'). Njeno izostavljanje javlja sintaktičku pogrešku, tako čestu programeru početniku.

Neka se sada promotri jedan drugi Python program, kako bi se uočila svojstva i način pisanja programskog kôda.

Program 1.2 Izgled Python programa

```
1  from random import randint
2  gotovo=False; korak=0
3  broj=randint(1,100)
4  while not gotovo:
5      x=int(raw_input("Pogodi zamisljeni broj izmedju 1 i 100:"))
6      if x==broj:
7          print 'Cestitamo!'
8          gotovo=True;
9      elif x<broj:
10         print 'Pucaj navise!'
11     else:
12         print 'Pucaj nanize!'
13     korak+=1
14 print 'Pogodak iz %d puta' %korak
```

Prva linija programa uvlači (engl. *import*) iz modula `random` funkciju `randint()` s kojom će se generirati slučajan cijeli broj. Varijabla '`gotovo`' postavlja se u Bool-ovo stanje neistine (engl. *False*), a varijabla '`korak`' inicijalizira se sa `0`. Znak ';' služi za odvajanje naredbi pisanih na istoj liniji. U varijablu '`broj`' sprema se slučajni broj (između 1 i 100), kojeg korisnik pogađa. Kako se vidi, postoji petlja `while` koju program u izvođenju vrti sve dok varijala `gotovo` ne postane istinita, tj. `True`, a to se dogadja kad zamišljeni '`broj`' bude jednak, od korisnika izabranoj, vrijednosti varijable '`x`'. U slučaju da to nije ispunjeno, program 'pomaže' korisniku savjetom da pogađa naviše ili naniže. Pritom se varijabla '`korak`' svaki put povećava za `1`, kako bi na koncu, nakon čestitke, bilo ispisano i koliko koraka je trebalo da se do nje dođe.

Treba primjetiti kako u Pythonu ne postoje oznake početka i konca bloka naredbi (kao što su to vitičaste zagrade u *C*-jeziku ili `begin-end` u *Pascal*-u), nego se to ostvaruje uvlakama. Na taj način korisnik je prisiljen pisati strukturirani kod, lagan za čitanje. Sve upravljače naredbe

(`if`, `while`, `else` i sl.) završavaju sa znakom dvotočke (':'). Treba također uočiti kako je rad sa ulazom i izlazom u Pythonu jednostavan (poput *Basic*-a) - postoje dvije funkcije: `row_input()` za ulaz i `print` za izlaz. Prva ispisuje poruku korisniku i učitava niz znakova (*string*) koji korisnik upiše, a druga samo ispisuje string i/ili sadržaj varijabli. Moguće je također i formatiranje izlaza (zadnja *print* naredba).

1.2.2 Kako izvesti Python program

Python kôd može se izvoditi na više načina:

Preko konzole ili terminala

To je rijetki, najjednostavniji način rada na najnižoj sistemskoj razini, a započinje pozivom `python` u terminalskom (Linux) ili DOS 'Command prompt' okolišu za Windows:

Zaslon 1.1

```
C:\>python
Python 2.7.4 (default, Apr  6 2013, 19:54:46)
Type "help", "copyright", "credits" or "license" for more info
>>>
```

Linux i DOS terminalska okoliš tek se neznatno razlikuje.

Python interepreter ispisuje svoj znak, (engl. prompt) >>>, iza kojeg korisnik može unijeti (utipkati) naredbu, koju završa s *ENTER tipkom* (ili *Return* tipkom). Zatim Python izvodi tu naredbu, ispisuje rezultat i ponovo ispisuje prompt očekujući novu naredbu:

Zaslon 1.2

```
>>> 1+1
2
>>> 'hrvatski' + ' jezik'
'hrvatski jezik'
>>> 3 * 5
15
>>> 3 * 'Sloboda! '
'Sloboda! Sloboda! Sloboda! '
>>>
```

Na ovaj način Python se može koristiti kao jednostavan, ali i vrlo složen kalkulator ili pomagalo za jezične obradbe.

U slučaju naredbi koje pretpostavljaju druge naredbe u nastavku, npr. naredbe programskih petlji (`for`, `while`, ...), Python interpreter ispisuje '...' kako bi korisnik lakše načinio uvlaku za nove naredbe:

Zaslon 1.3

```
>>> for x in 'slatka':
...     print x
...
s
l
a
t
k
```

```
a
>>>
```

Iako moguć, konzolni rad se rijetko koristi, jer Python dolazi s boljim, grafičkim sučeljem, tzv. *IDLE* (**I**ntegrated **D**eve**L**opment **E**nvironment) kojeg je razvio sam Pythonov autor.

Češća upotreba terminalskog rada od gore opisanog je poziv već gotovog programa, za koji nije razvijeno grafičko sučelje (GUI - graphical user interface) nego se komunikacija s korisnikom ostvaruje jednostavnim upisom preko tipkovnice. Takav je naš prvi Python program (1.1) pa ćemo ga sad izvesti na taj jednostavan način (u primjeru su preko tipkovnice utipkane riječi (`slobodo, blaga, dar, dare, lijepa`) nakon što je program ispisao '`Pogodi zamišljenu riječ`':

Zaslon 1.4 — Izvođenje Python programa s naredbene linije (DOS).

```
C:\>python pogodi.py
O lijepa, o draga, o slatka slobodo!
Dare u kom' sva blaga višnji nam Bog je do!
---------------------- (Ivan Gundulić) ----
Pogodi zamišljenu riječ: slobodo
Zamišljeno je leksikografski ispred "slobodo". Pucaj naniže!
Pogodi zamišljenu riječ: blaga
Zamišljeno je leksikografski iza "blaga". Pucaj naviše!
Pogodi zamišljenu riječ: dar
Oprez kod tipkanja! 'dar' nije u stihu!
Pogodi zamišljenu riječ: dare
Zamišljeno je leksikografski iza "dare". Pucaj naviše!
Pogodi zamišljenu riječ: lijepa
Čestitamo!
Pogodak iz 4. puta.

C:\>
```

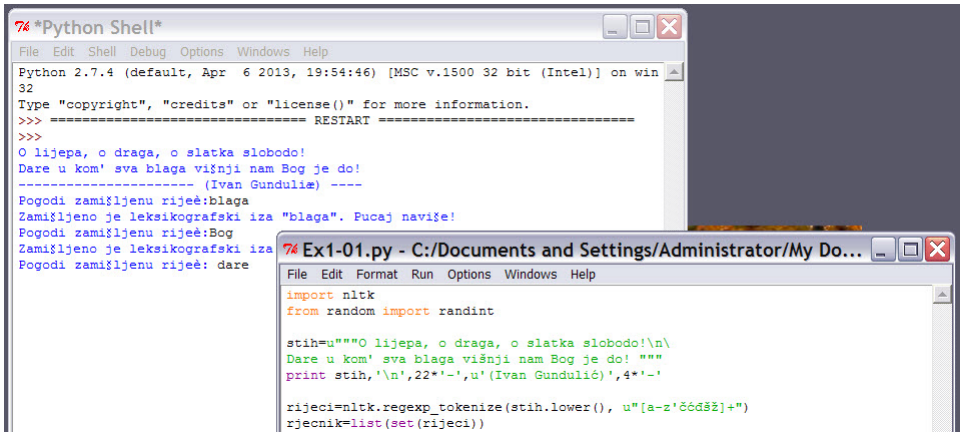
Skriptni rad

U instalacijskom paketu s Pythonom dolazi **IDLE** koji koristi grafički *Tkinter GUI framework* i prenosiv je na sve Python platforme koje imaju *Tkinter* potporu (Windows, Linux, XoS). IDLE omogućuje istodobno interaktivni (terminalski, konzolni) rad i skriptni rad - pokretanje niza Python naredbi uređenih i spremljenih s pomoću prikladnog editora. Tako se u jednom grafičkom okviru (prozoru) može držati i uređivati Python program, a u drugom izvršavati i interaktivno sudjelovati u njegovoj izvedbi, kao što pokazuje slika 1.1:

Osim standardnog, korisniku su na raspolaganju i mnoga druga rješenja za razvitak Python programa i njegovo izvodjenje. Neka od njih se ugradjuju u već postojeće razvojne okoline, npr. *Eclipse*, dok su druge samostalne i nadogradjuju se sa svakom novom inačicom programa. Jedna od takvih vrlo popularnih je *PyScripter* okolina, pokazana na slici 1.2:

Umetnuti (eng. *embeded*) kôd - veza s drugim programskim jezicima

Python nije samo programski jezik koji se izvodi instalacijom u vaša stolna ili prijenosna računala, njegove inačice mogu se naći i za robote, mobilne telefonske uređaje, industrijska upravljanja i slično. Python je zamišljen i kao proširiv programski kôd ne samo vlastitim funkcijama (Python objektima, modulima i paketima) nego i s drugim programskim jezicima. Iako se češće unutar Pythona mogu pozivati funkcije iz drugih programa (npr. *C*-a za slučaj programskog ubrzanja), moguće je također Python kôd u obliku izvornih tekst naredbi izvoditi i unutar programa pisanog u drugom programskom jeziku, koristeći *Python runtime API*, na primjer unutar *C*-programa:



Slika 1.1: IDLE okoliš

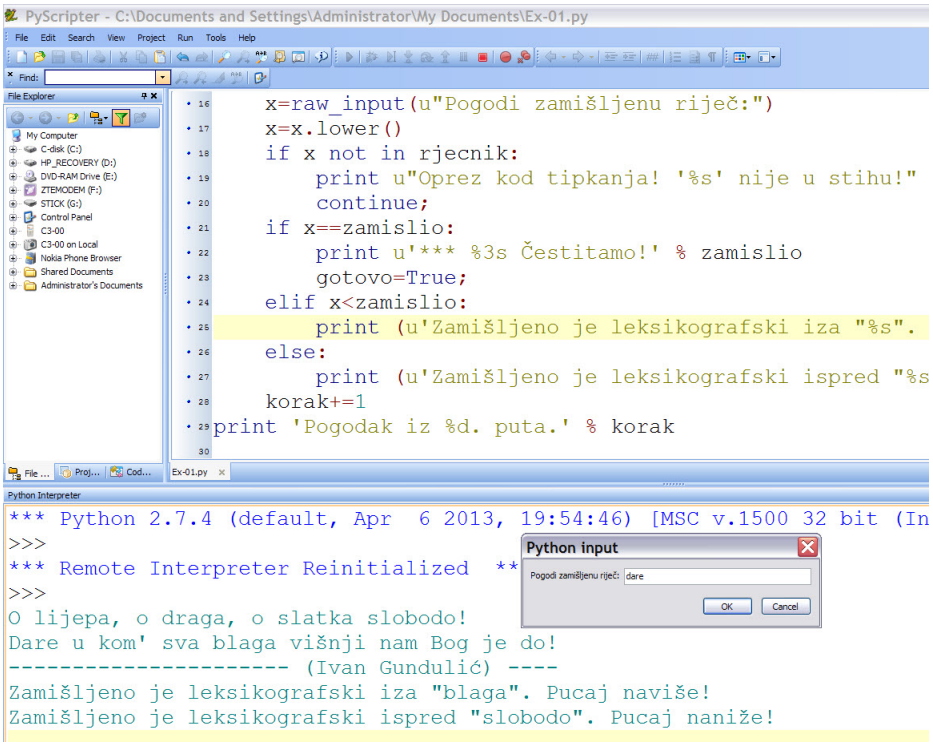
Zaslon 1.5

```
#include <Python.h>
. . .
Py_Initialize();
PyRun_SimpleString("x = a + 2*pi");
```

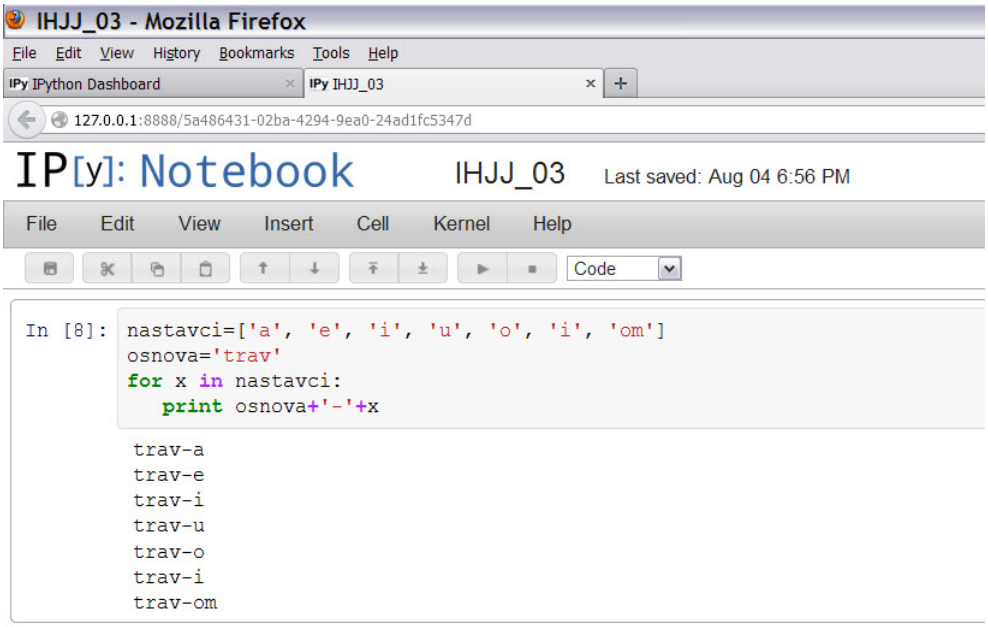
Python varijante

Python je bogat varijantama, programskim rješenjima koja se naslanjaju na ustroj i sintaksu drugih programskih jezika, pa postoji *Jython* - naslonjen na Java programski jezik, *Cython* - na C++ programski jezik i sl. Među njima, svakako treba istaknuti *IPython* koji je načinio najveći iskorak, jer omogućuje sve oblike dosadašnje Python uporabe, a k tomu dodaje i mogućnost izvođenja programa kroz WEB-sučelje s pomoću WEB preglednika (*Mozzila Firefox* ili *Google Chrome*), kako pokazuje slika 1.3.

IPython ima instalacijsku inačicu i onu portabilnu, pa se može izvoditi i s memorijske kartice (engl. *memory stick*) što je često vrlo dobro rješenje. Pritom je korisniku pružena mogućnost pokretanja *IPython*a u terminalskom, grafičkom i skriptnom okolišu (*QT-konzola* i *PyScripter*), te *WEB bilježnicama* (eng. notebook).



Slika 1.2: PyScripter



Slika 1.3: IPython

2 — Od podatka do izraza

Računalni program je programskim naredbama realiziran algoritam, koje se izvršavaju nad nekom *vrstom* ili *tipom* (binarno spremljenih) *podataka*. Binarni podatak je niz jedinica i ništica ('1' i '0', koje zovemo 'bitovi') spremljenih u memoriji računala. Memorija računala može se pojednostavljno predočiti kao niz ladica, od kojih svaka ima svoju oznaku, adresu, a u svakoj ladici je spremljena neka (binarna) vrijednost: svaka adresa ima i svoj binarni sadržaj. Centralni procesor (*CPU* - **c**entral **p**rocessing **u**nit) računala nad tim binarnim podacima izvršava osnovne aritmetičko-logičke operacije i kao rezultat daje isto takav niz jedinica i ništica (ipak ne s istim redoslijedom :)). Srećom nećemo proučavati programiranje na razini bita. Tek treba spomenuti da sve što vidimo na zaslonu računala: broj, riječ, slika, film, i sl. u pozadini ima binarni zapis. Fizikalno, taj zapis je niz visokih (5V) i niskih (oko 0V) električkih, naponskih razina, jer radimo s električnim računalom.

Python interpreter poput CPU jedinice izvršava aritmetičko-logičke operacije (uvjetna grananja i petlje) nad svojim 'podacima'. Pythonovi 'podaci' zovu se *objekti* (engl. *object*). Python i njegovi objekti samo su (na višoj razini) apstraktni model CPU i binarnih podataka, jer je njihova realizacija na koncu dakako i jedino, binarni zapis.

Objekt je osnovna jezgra cijelog Pythona, pa se veli: "*Sve je objekt!*". *Objektno programiranje* u Pythonu (nakon uvodjenja pojma klase i objekta u C++ i posebno Java programskog jezika) značajno se razlikuje od *strukturog programiranja* (npr. u Pascalu ili C-u).

Svaki objekt može imati najviše četiri svojstva, od koji su prva dva (adresa i tip) implicitno sadržana u svakom objektu, a ostala dva ovise o namjeni objekta:

- *adresa objekta* - adresa u memoriji gdje je objekt spremljen;
- *tip objekta* - veličina i ustroj memorijskog prostora kojeg objekt zauzima i na kojem se stanovite metode (obrade) mogu tj. smiju obavljati;
- *vrijednost objekta* - sadržaj memorijskog prostora spremljenog objekta
- *metoda ili funkcija objekta* - programski kôd koji se primjenjuje na vrijenosti(ma) objekta. Funkcija objekta poziva se sintaksom *ime_objekta.ime_metode()*, a unutar zagrada mogući su argumenti.

Adresa i *tip objekta* su nepromjenljive veličine i definiraju se stvaranjem, generiranjem, objekta. Svaki objekt može imati jednu ili više vrijednosti, te jednu ili više metoda.

Tip objekta može biti *skalar* ili *ne-skalar* (složeni tip). Skalarni tipovi su nedjeljivi, kao neki atomi programskog jezika. Python ima samo četiri skalarna tipa:

- cijeli broj (*int*), na primjer: `100`, `-67`
- realni broj (*float*), na primjer: `61.45`, `2.7E-4`
- logička vrijednost (*bool*) - `True` ili `False`
- ništa (*None*) - bez vrijednosti

Neki objekti dolaze sa Python interpreterom, zovemo ih *ugrađeni objekti* (engl. *built-in objects*), a druge stvara sâm korisnik preko Python klasa.

Objekti kojima se vrijednost(i) 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. Očevidno je da su skalarni tipovi nepromjenljivi objekti.

Imenovanjem adrese dobije se *varijabla*, a podatak bez imena, zove se *literal*.

2.1 Literali

Definicija 2.1 Literal

`Literal označuje vrijednost podatka koja se neposredno, direktno, pojavljuje u programskoj naredbi.`

Ovisno o memorijskom prostoru koji zauzima (tj. tipu objekta literala) razlikujemo (među jednostavnim):

```
144                     # Cjelobrojni literal
13.514                  # Realni (Floating-point) literal s decimalnom točkom
'IHJJ'                  # String literal, niz znakova
```

a među složenim (sastavljenim od više jednostavnih), najvažniji su:

```
[ 'i', 'pa', 'te' ]     # Lista, listina ili popis (engl. list)
( 450, 320, '600' )     # n-terac (engl. tuple)
{ 'a':72, 'b':1.4 }     # rječnik (engl. dictionary)
```

Iako se može činiti da su svi jednostavni literali skalarnog tipa, ipak *string* literal nije skalar, već se slično složenom podatku *liste* ili *n-terca* može rastavljati na manje dijelove (npr. slovo ili dijelove niza, tzv. kriške).

Definicija 2.2 Tip podatka

`Tip podatka je njegov nutarnji ustroj i potrebno ga je poznavati kako bismo mogli upravljati objektima i njihovim međusobnim odnosima.`

Funkcija id() vraća adresu objekta, a type() govori o njegovom tipu:

Zaslon 2.1

```
>>> id(144)
11924036
>>> type(144)
<type 'int'>
>>> id('IHJJ')
21460800
>>> type('IHJJ')
<type 'str'>
>>> id([ 'i', 'pa', 'te' ] )
```

```
21711008
>>> type([ 'i', 'pa', 'te' ] )
<type 'list'>
>>>
```

Adresa objekta je jedna posve nevažna značajka, na koju se ne može utjecati, nju stvara Python interpreter, a njen iznos (broj) ne može se predvidjeti. To znači da će funkcija *id()* na Vašem stroju vratiti (ispisati) vjerojatno drugačiju vrijednost nego što pokazuje zaslon 2.2. Puno važnija značajka je ona o tipu objekta i neovisna je o stroju (cijeli broj je uvijek tipa 'int', niz znakova tipa 'string' i td.).

Postoje ugrađeni tipovi podataka, koji dolaze s instalacijom Pythona, od kojih osnovne pokazuje tablica 2.1. Tvorbom objekata, korisnik stvara i vlastite tipove podataka.

Kategorija tipa podataka	**Ime tipa podatka**	**Opis**
Prazno (None)	`NoneType`	'null' objekt
Brojevi	`IntType`	Cijeli broj
	`LongType`	Dugi cijeli broj
	`FloatType`	Realni broj s pom. zarezom
	`ComplexType`	Kompleksni broj
Nizovi	`StringType`	Niz znakova (string)
	`UnicodeType`	Unicode (string)
	`ListType`	Listina, popis ili lista
	`TupleType`	n-terac
Preslikavanje	`DictType`	Rječnik
Klase, razredi	`ClassType`	Definicija klase
Instanca klase, objekt	`InstanceType`	Stvaranje instance
Datoteka	`FileType`	Datoteka - podaci na mediju
Moduli	`ModuleType`	Modul (skup objekata)

Tablica 2.1: Neki ugrađeni tipovi podataka

Literali cijelih brojeva mogu se zapisati na različite načine, no najčešći je decimalni, a ponekad (npr. za zapis boje) heksadecimalni. *Decimalni literal* je predstavljen nizom znamenki gdje je prva znamenka različita od nule, dok *heksadecimalni literal* koristi početni niz `0x` nakon čega slijedi niz heksadecimalnih znamenki (0 do 9 i A do F bilo velikim ili malim slovom). Na, primjer:

```
1,  23,  3493          # Decimalni cijeli brojevi
0x1,  0x17,  0xda5     # Heksadecimalni cijeli brojevi
```

Za velike cijele brojeve Pyton poznaje i tzv. *long integer* - literal cijelog broja s puno znamenki. U sintaksi se razlikuje od običnog cijelog broja po slovu 'L' na kraju znamenki. Na primjer:

```
1L,  23L, 99999333493L          # Dugački cijeli brojevi (decimalni zapis)
0x1L,  0x17L,  0x17486CBC75L    # Dugački cijeli brojevi (heksadecimalni zapis)
```

Razlika između dugačkog i običnog cijelog broja je u tome što dugački cijeli broj nema određenu numeričku granicu - može biti toliko dug koliko računalo ima memorije. Običan cijeli broj uzima samo nekoliko okteta memorije i ima minimalnu i maksimalnu vrijednost koju određuje arhitektura stroja. Ako želite saznati koliki je najveći cijeli broj koji Python prepoznaje, možete iskoristiti vrijednost '*maxint*' iz objekta'*sys*'.

Zaslon 2.2

```
>>> import sys
>>> sys.maxint
2147483647
>>> type(sys.maxint)
<type 'int'>
>>> sys.maxint+1  # povećanjem barem za 1 najvećeg dobivamo long
2147483648L
>>> type(sys.maxint+1)
<type 'long'>
>>> 2**64  # uvijek dvostruko - broj zrna pšenice na šahovskoj ploči
18446744073709551616L
>>> type(2**64)
<type 'long'>
>>>
```

Realni literal (broj s pomičnim zarezom) predstavljen je nizom decimalnih znamenki koje uključuju decimalni zarez, tj. točku (.), exponent (*e* ili *E*, te + ili - iza, s jednom ili više znamenki na kraju), ili oboje. Vodeći znak decimalnog literala ne smije biti *e* ili *E*,a može biti bilo koja znamenka ili točka (.). Na primjer:

```
0.,  1.0,  .2,  3.,  4e0,  5.e0, 6.0e0
```

Pythonova decimalna vrijednost odgovara uobičajena 53 bita preciznosti na modernim računalima.

Kompleksni broj sastavljen je od dviju decimalnih vrijednosti, jedne za realni, a druge za imaginarni dio. Moguće je pristupiti dijelovima kompleksnog objekta z kao samo-čitajućim "read-only" atributima `z.real` i `z.imag`. *Imaginarni literal* dobije se dodavanjem znaka '`j`' ili '`J`' realnom literalu:

```
1j,  1.j,  1.0j,  1e0j,  1.e0j,  1.0e0j
```

Znak *J* (ili *j*) na kraju literala označuje kvadratni korijen od -1, što je uobičajena oznaka imaginarnog dijela u elektrotehničkoj praksi (neke druge discipline koriste znak '`i`' u tu svrhu, ali Python je izabrao baš znak `j`).

Osim dijakritičkih znakova i naglasaka, tu su npr. i problemi zapisa upravnog govora. Kako načiniti string literal koji će uključiti navodnike, ako ih Python korsiti kao početak i kraj stringa. Na primjer, kako ispisati rečenicu - I reče Bog: 'Neka bude svjetlost!' Kad bi Python imao samo znak ' za početak i konac stringa, rezultat bi bio porazan:

Zaslon 2.3

```
>>> 'I reče Bog: 'Neka bude svjetlost!' '
  File "<interactive input>", line 1
    'I reče Bog: 'Neka bude svjetlost!' '
                      ^
SyntaxError: invalid syntax
>>>
```

Srećom, string se može omeđiti s jednim, dva ili tri jednostruka ili dvostruka navodnika, pa ćemo u interaktivnom okolišu koristeći par dvostrukih navodnika, omogućiti uporabu jednostrukih, kako dio stringa:

Zaslon 2.4

```
>>> "I reče Bog: 'Neka bude svjetlost!' "
"I re\xc4\x8de Bog: 'Neka bude svjetlost!' "
>>> print "I reče Bog: 'Neka bude svjetlost!' "
I reče Bog: 'Neka bude svjetlost!'
>>>
```

Dakako, mogao se string označiti izvana dvostrukim, a unutra (upravni govor) s jednostrukim, rezultat bi bio ispravan. Jedino se navodnici ne bi smjeli presijecati, to jest, smiju se gnijezditi jedno u drugo, ali kad je nešto započeto s jednom vrstom, ne smije doći druga, nego se prva završi.

Primjetite da se u gornjem interaktivnom rješenju string ispisuje s heksadecimalnim znamenkama (s dva okteta 0xc4 i 0x8d za slovo 'č' u riječi 'reče'), dok *print* naredba taj kod pretvara i ispisuje odgovarajući unicode znak.

Svaka podatčana vrijednost bilo kojeg tipa u Pythonu ima i svoju logičku, tzv. *Bool*-ove (engl. *boolean*) vrijednost. Ona se izražava *True* i *False* literalom. Praznom stringu (""), numeričkoj ništici (0 ili 0.), praznom nizu bilo kojeg tipa (listi '[]', n-tercu '()' ili rječniku '{}') pridružena je logička ništica (*False*), a svemu ostalom logička jedinica (*True*). S ovim logičkim vrijednostima (*True* i *False*) ostvarit će se sve usporedbe i odnosi u izrazima (preko *odnosnih operatora*) i sva grananja (*if/else*) i upravljanje tijekom programa. Logički literali ravnaju se po *Boolovoj logici*, u kojoj su najznačajnije dvije tablice istine: *AND* (I) i *OR* (ILI). Prva govori da je je istina onda i samo onda ako su obje postavke istinite, a druga ako je bilo koja od njih istinita.

2.2 Operatori

Povezivanje i usporedba objekata u Pythonu se, kao i u drugim programskim jezicima, ostvaruje preko operatora.

Definicija 2.3 Operator

Operator je funkcija predstavljena posebnim znakom kojim se djeluje na jedan ili više objekata. Razlikuju se aritmetički i logički operatori.

Među aritmetičkim razlikujemo unarne (koji djeluju na jedan objekt) i binarne koji djeluju između dva objekta. Unarni operatori su aritmetički '+' i '-'. Minus ('-') govori o negativnoj vrijednosti literala. Isti znak, kako je uobičajeno, predstavlja binarni operator oduzimanja.

Python nudi uobičajene numeričke operacije, kako se vidi u tablici 2.2 gdje su sa 'x' i 'y' označeni numerički literali.

Operacija	Opis
$x+y$	Zbrajanje
$x-y$	Oduzimanje
$x*y$	Množenje
x/y	Dijeljenje
$x**y$	Potenciranje ($x \wedge y$)
$x\%y$	Modulo funkcija (x mod y)
$-x$	Unarni minus
$+x$	Unarni plus

Tablica 2.2: Numeričke operacije

Većina objekata može se uspoređivati s obzirom na jednakost (==) (ili nejednakost (!=) i poredak (<, <=, >, >=). Rezultat usporedbe su Bool-ove vrijednosti (True ili False). Operatori usporedbe (tzv. *odnosni operatori*) prikazani su u tablici 2.2.

Operator	Opis
`x < y`	Manje nego
`x > y`	Veće nego
`x == y`	Jednako
`x != y`	Nije jednako (isto kao <>)
`x >= y`	Veće nego ili jednako
`x <= y`	Manje nego ili jednako

Tablica 2.3: Operatori usporedbe

Operatori usporedbe mogu se nizati, čime se implicira logična I (and) funkcija. Na primjer:

```
a < b <=  c < d
```

ima isto značenje kao:

```
a < b  and  b <= c  and  c < d
```

Oblik prvog niza je mnogo čitljiviji, a provjerava svaki podizraz samo jednom.

2.3 Varijable ili promjenljivice

Definicija 2.4 Varijabla

Imenovana adresa ili identifikator objekta zove se varijabla. Njezin sadržaj je adresa memorijske lokacije. Na taj način ne pamte se brojevi adresa, nego ime varijable koju je korisnik sam zadao.
Ime varijable ne smije počinjati brojkom, u sebi ne smije imati praznine i posebne znakove (osim '_') i ne smije biti jedna od 30 ključnih riječi koje Python prepoznaje kao svoje naredbe.

Varijabla nema svoj vlastiti tip, ona je uvijek samo jedan broj, broj memorijske lokacije, za razliku od podatka na kojeg pokazuje (kojeg referencira), tj. čiju adresu pohranjuje. Budući da je adresa jedinstvena u smislu broja okteta, ona može *povezati/referencirati* objekte različitih tipova, što znači da je *dinamična*, da se može mijenjati prilikom izvršenja programa. U jednom trenutku varijabla može pokazivati na cjelobrojni podatak, a već u sljedećem na realni, kompleksni, string ili neki složeni tip.

U Pythonu nema deklaracija. Postojanje varijable ovisi o naredbi koja povezuje (eng. *binding*) varijablu i podatak; drugim riječima, naredbi koja imenuje neki objekt, bilo kojeg tipa. Moguće je odvezati (eng. *unbinding*) varijablu resetiranjem njenog imena, tako da više ne sadrži referencu na taj objekt. Naredba `del` odvezuje reference.

Povezivanje reference koja je već povezana poznato je kao *re-povezivanje* (eng. *rebinding*). Re-povezivanje ili odvezivanje reference nema nikakav učinak na objekt s koji je referenca bila povezana, osim što objekt nestaje ako više ne postoji nikakva referenca koja se na njega odnosi. Odvezani objekti sami će nestati iz memorije. Za to se brine poseban Python modul, ugrađen u interpreter. Takvo automatsko čišćenje objekata bez referenci zove se sakupljanje smeća (engl. *garbage collecting*).

Zaslon 2.5 — varijabla je referenca.

```
>>> a=505
>>> id(a)
14717628
>>> type(a)
<type 'int'>
>>> a='Dobar dan!'
>>> type(a)
<type 'str'>
>>> a
'Dobar dan!'
>>> print a
Dobar dan!
>>> id(a)
21392288
>>> a
'Dobar dan!'
>>> del(a)
>>> a
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
NameError: name 'a' is not defined
>>>
```

2.4 Izraz

Iako već literal ili varijabla u Pythonu predstavljaju izraz, uobičajeno je da se pod pojmom izraza (eng. *expression*) smatra povezanost literala i varijabli s pomoću aritmetičko-logičkih operatora prema sintaktički ispravnim pravilima. Svaki izraz ima svoju logičku funkciju (*True* ili *False*), a može imati i aritmetičku, ako radi s numeričkim vrijednostima literala i varijabli.

2.4.1 Pridružba

Naredbe pridruživanja mogu biti *obične* ili *proširene*. Obično pridruživanje varijabli (npr. `name=value`) je način stvaranja nove varijable ili re-povezivanja postojeće varijable na novu vrijednost (tj. promjena vrijednosti koju varijabla referencira). Proširena pridružba (npr. `name+=value`) za isto ime varijable stvara nove reference, re-povezuje varijablu s drugim brojem (adresom). Naredba obične pridružbe u svom najjednostavnijem obliku ima sintaksu:

Definicija 2.5 — pridružba.

```
odredište = izraz
```

Cilj ili odredište (eng. *target*) je poznat kao *lijeva strana* pridružbe, a *izraz* (eng. *expression*) je *desna strana*. Izraz može biti obična varijabla ili više varijabli povezanih operatorima, poziv funkcije ili mnoštvo drugih kombinacija s međusobno povezanim objektima. Kad se naredba pridružbe izvršava, Python izračunava izraz desne strane, te povezuje vrijednost izraza s ciljem na lijevoj strani. Ovo povezivanje ne ovisi o tipu izračunate vrijednosti desne strane, jer se pridružba ionako događa na razini referenca, adresa objekata, a ne njihova sadržaja. Cilj može biti *varijabla/identifikator*, *atribut*, *indeksirani član niza* ili *kriška* (engl. *slicing*) .

Detalji povezivanja ovise o vrsti ili *tipu* cilja:

- *Identifikator* je ime varijable: pridružba na identifikator povezuje sadržaj varijable s tim imenom, upisom adrese pod ime identifikatora.

Zaslon 2.6 — adresa i sadržaj varijable.

```
>>> a='Marko'
>>> a
'Marko'
>>> id(a)
21391904
>>> b=a
>>> id(b)
21391904
>>> a='Ivan'
>>> id(a)
21462464
>>> a
'Ivan'
>>> b
'Marko'
>>> id(b)
21391904
>>>
```

- *Indeksiranje* ima sintaksu `obj[expr]`. Pritom je `obj` objekt, a `expr` je *izraz* koji indeksira mjesto u nizu. Objekt može biti bilo kojeg tipa.

Zaslon 2.7 — indeksiranje.

```
>>> niz=u"Voćka poslije kiše"
>>> niz[0]
u'V'
>>> niz[1]
u'o'
>>> niz[2]
u'\u0107'
>>> print niz[2]
ć
>>> print niz[-1], niz[-2]
e š
>>>
```

  Pridružba na indeksiranje traži da spremnik `obj` poveže svoj član koji je izabran pomoću vrijednosti `expr`, također poznate i kao *indeksni ključ člana* s izračunatom ili pozvanom vrijednošću desne strane.
- *Kriška* (eng. *slicing*) ima sintaksu `obj[start:stop]` ili `obj[start:stop:korak]`. Pritom je `obj` objekt, a `start`, `stop` i `korak` su izrazi koji indeksiraju dio niza objekata. (Dopušteno je izostavljanje članova, pa je `obj[:stop:]` sintaksno ispravna kriška, ekvivalentna s `obj[None:stop:None]`). Pridružba traži od niza objekata `obj` da se povežu ili odvežu neki od njegovih članova.

Zaslon 2.8 — indeksiranje.

```
>>> niz=u"Voćka poslije kiše"
```

```
>>> print niz[:5], niz[9:13], niz[:-5:-1]
Voćka lije ešik
>>>
```

U jednostavnoj pridružbi može biti više ciljeva i znakova jednakosti (=). Na primjer:

Zaslon 2.9 — istodobna pridružba za više varijabli.

```
>>> a = b = c = 0
```

povezuje varijable a, b, i c s vrijednosti 0. Svaki cilj se povezuje s jednim objektom koji vraća izraz, isto kao kad bi se nekoliko jednostavnih naredbi izvršavale jedna za drugom.

Cilj u jednostavnoj pridružbi može imati dvije ili više referenci odvojenih zarezima, proizvoljno ograđenih lučnim ili kutnim zagradama. Na primjer:

Definicija 2.6 — višestruka/raspakiravajuća pridružba.

```
a, b, c = x
```

Ovo zahtijeva da x bude niz od tri člana, te povezuje a s prvim članom, b s drugim, te c s trećim. Ova vrsta pridružbe zove se *raspakiravajuća pridružba* i pritom izraz s desne strane mora biti niz s točnim brojem članova koliko ima i referenci u cilju, jer se inače podigne iznimka. Svaka referenca u cilju je jednoznačno povezana s odgovarajućim članom u nizu. Raspakiravajuća pridružba također može izmjenjivati reference. Na primjer:

Zaslon 2.10 — izmjena sadržaja dviju varijabli.

```
>>> a, b = b, a
```

Ovaj izraz re-povezuje a da se pridruži na ono što je u b bilo povezano, i obratno.

2.4.2 Proširena pridružba

Proširena pridružba razlikuje se od obične pridružbe u tomu, što se umjesto znaka jednakosti (=) između cilja i izraza s desne strane koristi *prošireni operator*, tj. binarni operator nakon kojeg slijedi =. Operatori proširene pridružbe su:

Definicija 2.7 prošireni operatori

$+=, -=, *=, /=, //=, \%=, **=, |=, >>=, <<=, \&= \text{i} \wedge=.$

Proširena pridružba (tablica 2.4.2) može imati samo jedan cilj na lijevoj strani, tj. proširena pridružba ne podržava više ciljeva.

Pridružbom objekta stvara se nova referenca (koristi se nova memorijska lokacija). To znači da se promjenom (npr. zbrajanjem s literalom) jedne varijable, stvara ista s istim imenom, ali na drugom mjestu (pa ima drugi adresni broj).

Zaslon 2.11 — varijabla kao referenca.

```
>>> niz=u"I gledam more gdje se k meni penje"
>>> id(niz)
15574536
>>> nova_var=niz  # nova varijabla nova_var
>>> id(nova_var)
15574536
>>> niz=niz+u"\ni slušam more dobrojutro veli"
>>> print niz
I gledam more gdje se k meni penje
i slušam more dobrojutro veli
```

Operacija	Ekvivalentno sa
$x+=y$	$x=x+y$
$x-=y$	$x=x-y$
$x*=y$	$x=x*y$
$x/=y$	$x=x/y$
$x**=y$	$x=x**y$
$x\%=y$	$x=x\%y$
$x\&=y$	$x=x\&y$
$x\|=y$	$x=x\|y$
$x\wedge=y$	$x=x\wedge y$
$x>>=y$	$x=x>>y$
$x<<=y$	$x=x<<y$

Tablica 2.4: Proširena pridružba

```
>>> id(niz)
15575280
>>>
```

Važno je primjetiti kako se pridružbom jedne varijable drugoj samo preslikava/prebacuje adresa memorije pa dvije varijable pokazuju/imaju isti sadržaj. Pokušajte otkriti zašto su naredbom print niz ispisana dva stiha jedan ispod drugog, a ne u nastavku kako bi sugerirao '+' operator nadovezivanja stringova.

2.4.3 Python naredba

Budući da izraz uvijek ima logičku vrijednost (*True* ili *False*) on se redovito koristi u upravljanju programom (programskim petljama i grananjima), tj. nad izrazom se primjenjuju Python naredbe.

Pythonove ključne riječi predočene su u tablici 2.1, a zorno su različitim bojama skupljene prema funkcijskim kategorijama:

and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	import	or	while
continue	exec	if	pass	with
def	finally	in	print	yield

Slika 2.1: Python ključne riječi

Očevidan je razlog zašto ime varijable ne smije biti jednako nekoj Python ključnoj riječi.

Ključna riječ (engl. *keywords*) može stajati i samostalno (npr. naredba *pass*), ali češće dolazi s izrazom, npr.

```
>>> if subjekt=="Hrvatska" and glagol=="pati":
...     print "Priskoči upomoć!"
```

U gornjem programu imamo naredbu 'if' i naredbu 'print'. Prva je povezana s logičkim izrazom koji koristi I (eng. *and*) logičku funkciju. Riječ 'and' nije u funkciji Python naredbe nego logičkog operatora. Osim '*and*' postoji i '*or*' (hrv. *ili*) tablica istine, koja vrijedi za dvije (kao u idućem programu) i više logičkih funkcija. Uz logički negaciju (NOT ili NE) s ovim se tablicama mogu stvoriti bilo koje logičke tablice istine (npr. 'implikacija', 'ekskluzivni ili' i sl.).

Program 2.1 Program provjerava ispravnost Boolovih tvrdnji.

```
1  print "===␣␣␣␣Boolova␣algebra␣␣␣␣===\n"
2  uvjet1=True
3  uvjet2=False
4
5  print "————————␣AND␣(I)␣————————————"
6  print uvjet1,"␣␣and␣",uvjet1, "\t=␣" ,uvjet1 and uvjet1
7  print uvjet1,"␣␣and␣",uvjet2 ,"\t=␣",uvjet1 and uvjet2
8  print uvjet2,"␣and␣",uvjet1, "\t=␣",uvjet2 and uvjet1
9  print uvjet2,"␣and␣",uvjet2, "\t=␣",uvjet2 and uvjet2
10
11 print "\n————————␣OR␣(ILI)␣———————————"
12 print uvjet1,"␣␣or␣",uvjet1, "\t=␣" ,uvjet1 or uvjet1
13 print uvjet1,"␣␣or␣",uvjet2 ,"\t=␣",uvjet1 or uvjet2
14 print uvjet2,"␣or␣",uvjet1, "\t=␣",uvjet2 or uvjet1
15 print uvjet2,"␣or␣",uvjet2, "\t=␣",uvjet2 or uvjet2
16
17 print "————————————————————————————"
```

što na zaslonu kao rezultat daje:

Zaslon 2.12

```
===    Boolova algebra    ===

-------- AND (I) -----------
True   and  True    =  True
True   and  False   =  False
False  and  True    =  False
False  and  False   =  False

-------- OR (ILI) ----------
True   or  True  =  True
True   or  False  =  True
False  or  True  =  True
False  or  False  =  False
---------------------------
>>>
```

Primjetite posebne znakove u stringu ('\n' u 1. liniji i '\t' u linijama 6. do 12.). Prvi je znak za skok u novi red (što znači da će se string sa 3 takva znaka ispisati u tri redka, dakako ovisno