5. Struktura programa u Pythonu

relacijski operatori
and, or, not
slijedni programi
moduli
logički operatori
import
petlje

5.1. Jednostavni programi

U prethodna dva poglavlja upoznali smo dovoljno elemenata programskog jezika *Python* i načina uporabe njegova sučelja tako da možemo sustavno pristupiti pisanju jednostavnih programa. U jednostavnim programima naredbe se izvode jedna iza druge onim redoslijedom kako su napisane. Takve programe možemo nazvati *slijednim* ili *sekvencijskim programima*. U njima naredbe slijede jedna drugu.

Kako se u ovom poglavlju želimo usredotočiti na izučavanje svojstva samog jezika, a ne na način kako iznalaziti postupke za rješavanje različitih problema, postupno ćemo razvijati samo jedan program i to program za uvježbavanje aritmetičkih operacija – moglo bi se reći za igranje s brojevima. Željeli bismo provjeravati kako netko napamet izračunava zbrojeve i umnoške brojeva. Težina zadatka ovisit će o veličini zadanih brojeva.

5.1.1. Zamišljena igra s brojevima

Zamislimo da bi igra trebala rezultirati ovakvim dijalogom u prozoru interaktivnog sučelja:

```
U ovoj igri trebate odgovoriti na 5 pitanja!

Koliki je zbroj brojeva 17 i 25?
Utipkati odgovor: 42
Odgovor je točan!
Koliki je umnožak brojeva 11 i 12?
Utipkati odgovor: 132
Odgovor je točan!
Koliki je zbroj brojeva 27 i 56?
Utipkati odgovor: 73
Odgovor je netočan!
Koliki je umnožak brojeva 15 i 17?
Utipkati odgovor: 255
Odgovor je točan!
```

```
Koliki je zbroj brojeva 28 i 68?
Utipkati odgovor: 86
Odgovor je netočan!

U ovoj igri ste točno odgovorili na 3 od ukupno 5 pitanja.

Želite li nastaviti igru (odgovoriti s da ili ne): ne

Igra završava - lijep pozdrav!
```

Razmislimo jesmo li dosadašnjim saznanjima o *Pythonu* u stanju napisati program s kojim bi se mogao voditi ovakav dijalog?

Odmah vidimo da to neće biti moguće. Za sada ne znamo kako ustanoviti je li odgovor točan jer ne znamo način za uspoređivanje točnog rezultata s otipkanim odgovorom. Prema tome ne možemo ni znati koliko je točnih, a koliko netočnih odgovora.

Jedino što možemo učiniti jest ispis točnog odgovora (bez obzira na to je li utipkan točan ili netočan odgovor). Igrač sam mora izbrojiti točne i pogrešne odgovore.

Isto tako, odgovor na pitanje želi li igrač nastaviti igru ili ne, ne znamo vrednovati.

Dakle, program koji smo u stanju načiniti mogao bi izgledati ovako:

```
# Prva inačica igre s brojevima; primjer_05_A
print('U ovoj igri trebate odgovoriti na 5 pitanja!')
print()                                  # prazni redak
x = 17
y = 25
print('Koliki je zbroj brojeva {0} i {1}?'.format(x, y))
input('Utipkati odgovor: ')
z = x + y
print('Odgovor je:', z)
x = 11
y = 12
print('Koliki je umnožak brojeva {0} i {1}?'.format(x, y))
input('Utipkati odgovor: ')
z = x * y
print('Odgovor je:', z)
x = 27
y = 56
print('Koliki je zbroj brojeva {0} i {1}?'.format(x, y))
input('Utipkati odgovor: ')
z = x + y
print('Odgovor je:', z)
```

```
x = 15
y = 17
print('Koliki je umnožak brojeva {0} i {1}?'.format(x, y))
input('Utipkati odgovor: ')
z = x * y
print('Odgovor je:', z)
x = 28
y = 68
print('Koliki je zbroj brojeva {0} i {1}?'.format(x, y))
input('Utipkati odgovor: ')
z = x + y
print('Odgovor je:', z)
print()                                    # prazni redak
print('Odgovorili ste na 5 pitanja.')
```

U interaktivnom će se prozoru odvijati dijalog sličan ovome:

```
>>> ============================ RESTART ============================
>>>
U ovoj igri trebate odgovoriti na 5 pitanja!

Koliki je zbroj brojeva 17 i 25?
Utipkati odgovor: 22
Odgovor je: 42
Koliki je umnožak brojeva 11 i 12?
Utipkati odgovor: 132
Odgovor je: 132
Koliki je zbroj brojeva 27 i 56?
Utipkati odgovor: 83
Odgovor je: 83
Koliki je umnožak brojeva 15 i 17?
Utipkati odgovor: 255
Odgovor je: 255
Koliki je zbroj brojeva 28 i 68?
Utipkati odgovor: 93
Odgovor je: 96

Odgovorili ste na 5 pitanja.

Igra završava – lijep pozdrav!
>>>
```

Ovakav ćemo tekst vidjeti u prozoru sučelja ako je igrač na upite programa utipkavao redom brojeve: *22*, *132*, *83*, *255* i *93*.

S ovakvim programom ne možemo biti nikako zadovoljni. On ima niz nedostataka.

- Već smo ustanovili da nemamo mogućnost uspoređivanja točnog rezultata (koji program izračunava) s utipkanim rezultatom.
- Vidimo da iste naredbe moramo prepisivati pet puta (za veći broj pitanja to bi bilo mukotrpno).
- Nemamo mogućnost upita igrača želi li i dalje igrati.
- Igru možemo ponoviti samo ponovnim pokretanjem programa, ali će nam se pritom ponavljati uvijek isti brojevi čime igra gubi svoj smisao.

Očigledno je da nam trebaju neke jezične konstrukcije ako želimo unaprijediti ovaj program kako bi se on ponašao slično zamišljenom.

5.2. Donošenje odluka i grananja u programima

5.2.1. Relacijski operatori

Ustanovili smo da u prethodno opisanoj igri ne možemo otkriti je li utipkani odgovor točan ili netočan. Nedostajala nam je mogućnost uspoređivanja utipkanih brojeva s točnim rezultatima koji se unutar programa točno izračunavaju.

Upravo u tu svrhu u *Pythonu* (a tako je i u drugim programskim jezicima) postoje operatori uspoređivanja, tzv. relacijski operatori. Relacijski operatori uspoređuju dva operanda. Rezultat usporedbe ima vrijednosti *True* ili *False*. Prema tome, izraz usporedbe koji se sastoji od dva operanda i relacijskog operatora je *logički sud* koji može biti istinit ili lažan. Podsjetimo se što smo naučili o logičkom ili Booleovu tipu u odjeljku 3.2.3. Umjesto naziva *logički sud* mi ćemo se češće koristiti nazivom *uvjet*. Ako je logički sud istinit, reći ćemo da je uvjet ispunjen, a ako je logički sud lažan, reći ćemo da uvjet nije ispunjen.

Svi relacijski operatori koji postoje u *Pythonu* prikazani su tablicom 5.1. Operatori >=, <=, ==, != pišu se kao dva uzastopna znaka. Posebno treba obratiti pažnju na operator *jednako* ==. Ispuštenjem jednog znaka = on se odmah pretvara u znak pridruživanja što može potpuno promijeniti smisao programa.

Operator	Značenje simbola
>	veće od
<	manje od
>=	veće od ili jednako
<=	manje od ili jednako
==	jednako
!=	nije jednako

Tablica 5.1. *Relacijski operatori*

Pogledajmo u interaktivnom sučelju kako operatori djeluju:

```
>>> 17 > 35
False
>>> 17 < 20
True
>>> 17 == 18
False
>>> 17 != 18
True
>>>
```

Jasno je da nema smisla uspoređivati parove brojeva koje unaprijed znamo. Zbog toga se u relacijskom izrazu obično pojavlju varijable čije se vrijednosti tijekom izvođenja programa mijenjaju.

Logički sudovi (ili uvjeti) mogu biti složeni. Primjerice, ako želimo znati je li neki broj *x* veći od ili jednak nekoj donjoj granici *x_d* i istovremeno manji od ili jednak nekoj gornjoj granici *x_g*, primjerice za zadane vrijednosti: *x_d* = 0, *x_g* = 1000, *x* = 700 ili za *x* = 1100. U tom slučaju moramo ispitati dva uvjeta:

```
>>> x_d = 0
>>> x_g = 1000
>>> x = 700
>>> x >= x_d
True
>>> x <= x_g
True
>>> x = 1100
>>> x >= x_d
True
>>> x <= x_g
False
>>>
```

Vidljivo je da je broj 700 u zadanim granicama, a broj 1100 nije. Dva uvjeta koja smo ispitivali mogli bismo ispitati jednim ispitivanjem ako od dva logička suda stvorimo složeni logički sud. To je moguće učiniti poslužimo li se pravilima *algebre logičkih sudova* (*logičkom algebrom* ili *Booleovom algebrom*).

5.2.2. Logički operatori i logički izrazi

Za stvaranje bilo kojeg složenog suda dovoljna su nam tri operatora. Oni su prikazani u tablici 5.2. Operator **`and`** obavlja *operaciju* I, odnosno *konjunkciju*, operator **`or`** obavlja *operaciju* ILI, odnosno *disjunkciju*, dok operator **`not`** obavlja *operaciju* NE ili *negaciju*, odnosno *komplementiranje*. U trećem su stupcu navedeni uobičajeni simboli logičke algebre.

Operator	Naziv operacije	Algebarski simbol
`and`	*I operacija, konjunkcija*	$\wedge$
`or`	*ILI operacija, disjunkcija*	$\vee$
`not`	*NE operacija, komplementiranje*	$\neg$

Tablica 5.2. *Logički operatori*

U interaktivnom sučelju ispitajmo kako ti operatori djeluju:

```
>>> False and False
False
>>> False and True
False
>>> True and False
False
>>> True and True
True
>>>
```

Prema tome, *I* operacija daje rezultat `True` samo onda kada oba operanda imaju vrijednost `True`.

U odjeljku 3.2.3 spomenuli smo da se umjesto vrijednosti `False` i `True` mogu upotrebljavati i vrijednosti 0 i 1. Utvrdimo je li to uistinu tako:

```
>>> 0 and 0
0
>>> 0 and 1
0
>>> 1 and 0
0
>>> 1 and 1
1
>>>
```

Jednako tako, možemo ispitati i djelovanje operatora **`or`**:

```
>>> False or False
False
>>> False or True
True
>>> True or False
True
>>> True or True
True
```

```
>>> 0 or 0
0
>>> 0 or 1
1
>>> 1 or 0
1
>>> 1 or 1
1
>>>
```

Dakle, rezultat *ILI* operacije jednak je `True` (odnosno 1) ako samo jedan od operanada ima vrijednost `True` (odnosno 1). Drugim riječima, rezultat *ILI* operacije bit će jednak `False` (odnosno 0) samo onda ako oba operanda imaju vrijednost `False` (odnosno 0).

Opisivanje operacija riječima pomoći će nam pri razmišljanju kada budemo pokušavali rješavati neki problem.

Operator negacije `not` djeluje na jedan operand i invertira (`True` postaje `False`, `False` postaje `True`) njegovu vrijednost. Zbog toga operaciju negacije (komplementiranja) nazivamo još i **inverzijom**:

```
>>> not True
False
>>> not False
True
>>> not 0
True
>>> not 1
False
>>>
```

Zanimljivo je da se `not 0` ispisuje kao vrijednost `True`, a ne vrijednost `1` te da se `not 1` ispisuje kao vrijednost `False`, a ne vrijednost `0`, kako bismo to očekivali prema ispisima rezultata operacija `and` i `or`. To je jedna od rijetkih nedosljednosti u jeziku *Python*.

Uporabom logičkih operacija mogu se jednostavni sudovi kombinirati u složene sudove. U tom slučaju treba voditi računa o redoslijedu izvođenja logičkih operacija. Slično kao i kod aritmetičkih, i logičke operacije imaju prioritet izvođenja. Prioritet izvođenja dan je sljedećom tablicom:

Redoslijed	Operacija
1.	`not`
2.	`and`
3.	`or`

Tablica 5.3. *Redoslijed izvođenja logičkih operacija*

Ilustrirajmo to na nekoliko primjera:

```
>>> True or False and True
True
>>> False or True and False
False
>>> False or not False
True
```

Kod složenih izraza istovremeno možemo imati više vrsta operacija: i aritmetičke i relacijske i logičke. Redoslijed izvođenja operacija dan je sljedećom tablicom:

Redoslijed	Operacija
1.	aritmetički
2.	relacijski
3.	logički

Tablica 5.4. *Redoslijed izvođenja operacija*

```
>>> 2 + 3 < 4
False
>>> 2 * 3 - 1 > 3 * 3 - 5
True
>>> 3 + 4 > 5 and 3 + 5 > 4 and 4 + 5 > 3
True
>>>
```

Sada se ispitivanje je li broj `x` veći od ili jednak donjoj granici `x_d` i istovremeno manji od ili jednak gornjoj granici `x_g` može provesti ovim složenim uvjetom:

```
(x_d <= x) and (x <= x_g).
```

Taj će uvjet biti istinit ako se vrijednost varijable `x` nalazi u zadanim granicama. Primijetimo da izraz možemo napisati i bez zagrada:

```
x_d <= x and x <= x_g
```

Međutim, isti smo uvjet mogli napisati i na sljedeći način:

```
x_d <= x <= x_g
```

Ispitat ćemo to u interaktivnom sučelju:

```
>>> x_d = 0
>>> x_g = 1000
>>> x = 700
>>> (x >= x_d) and (x <= x_g)
True
>>> x_d <= x <= x_g
True
```

```
>>> x = 1100
>>> x_d <= x <= x_g
False
>>> x = -1
>>> (x >= x_d) and (x <= x_g)
False
>>>
```

Vidjet ćemo ubrzo da će nam pri rješavanju problema od znatne koristi biti vješta i pouzdana uporaba logičkih sudova i vrednovanje logičkih izraza. Donošenje različitih odluka u programima isključivo ovisi o logičkim izrazima. Poželjno je da oni budu što je moguće jednostavniji i razumljiviji. U tome nam može pomoći poznavanje logičke algebre. Ovdje se nećemo njome podrobnije baviti i spomenut ćemo samo neke činjenice.

Zanimljivo je razmotriti kako se zamjenom relacijskih operatora pri oblikovanju nekog uvjeta mijenja oblik logičkog izraza koji opisuje taj uvjet. Nekada nam to može pomoći pri pojednostavnjivanju programa i, što je još važnije, olakšati razumijevanje problema koji rješavamo. Pogledajmo tablicu 5.5.

U njoj su u krajnje lijevom stupcu napisani sudovi koje smo imenovali sa *G.* U krajnje desnom stupcu napisani su sudovi s istim vrijednostima operanada, ali s tzv. suprotnim relacijskim operatorima. Naime, kako je 5 veće od 2, onda je jasno da 5 nije manje od ili jednako 2.

Ako, primjerice, izrazu koji smo dobili nekim relacijskim operatorom pridružimo logičku varijablu *G*, možemo se zapitati koji bi relacijski operator trebalo primijeniti tako da uz iste vrijednosti operanada bude istinit komplement od *G*. Nazovimo *K* = `not`*G* istinit.

sud G	*vrijednost* G	*vrijednost* K = not G	sud K
x > 2	*1*	*0*	x <= 2
x > 6	*0*	*1*	*x* <= 6
x = 5	*1*	*0*	*x* != 5
x <= 3	*0*	*1*	*x* > 3

Tablica 5.5. *Primjeri uporabe suprotnih operatora (za x = 5)*

Iz tablice se može zaključiti da se zamjenom relacijskog operatora nekim drugim u nekom sudu može dobiti negacija tog suda. Nazovimo takav zamjenski operator suprotnim operatorom. Parovi međusobno suprotnih relacijskih operatora prikazani su u tablici 5.6.

Operator	Suprotni operator
>	<=
<	>=
>=	<
<=	>
==	!=
!=	==

Tablica 5.6. *Međusobno suprotni relacijski operatori*

5.2.3. Donošenje odluka u programima – izbor alternativnih blokova naredbi

Slijedni programi imaju vrlo skromne mogućnosti, kao što smo to vidjeli i na primjeru naše igre s brojevima. Bilo bi poželjno na temelju ispitivanja nekog uvjeta odabrati jednu od dvije mogućnosti (jer njegova vrijednost može biti samo `True` ili `False`). Prije negoli to zapišemo u obliku kako to određuje jezik *Python* opisat ćemo mogućnosti u hrvatskoj varijanti tzv. *pseudojezika* (prefiks pseudo- označuje da nije riječ o stvarnom jeziku, već o ne suviše čvrsto definiranom jeziku kojim možemo relativno slobodno opisivati programe). U tom će se jeziku upotrebljavati ključne riječi *ako je*, *onda* i *inače* (engl. *if*, *then*, *else*). Uporabom tih ključnih riječi možemo napisati program u sljedećem obliku:

```
...
ako je uvjet onda
{
     naredba1_1;
     ...
     naredba1_n
}
inače
{
     naredba2_1;
     ...
     naredba2_m;
}
...
```

Ako je `uvjet` ispunjen (logički izraz je istinit), obavit će se prvi blok naredbi, a ako uvjet nije ispunjen (logički izraz je lažan), obavit će se drugi blok naredbi. Uobičajeno se kaže da se program grana na dvije grane. Pri izvođenju programa obavit će se ili jedna ili druga grana. Iako ćemo često govoriti o grananju, taj naziv nije sasvim opravdan. Naime, grane nekog stabla, nakon što se račvaju iz debla ili drugih većih grana, više se nigdje ne sastaju, dok se nakon obavljanja naredbi iz jednog ili drugog bloka, program u pravilu nastavlja zajedničkim dijelom.

U *Pythonu* se odabir alternativnih blokova naredbi obavlja naredbama u kojima se rabe sljedeće ključne riječi: `if`, `else` i `elif`, iza kojih se stavlja dvotočka. Blokovi naredbi ne ograđuju se posebnim simbolima – oni započinju nakon naredbe koja završava dvotočkom i pišu se uvučeno za odgovarajući broj znakovnih mjesta koja se reguliraju tabulatorom (uobičajeno su četiri mjesta), to znači da je dovoljno i poželjno koristiti se tipkom *(tab)*. Iza zadnje naredbe bloka mora pisati naredba koja nije uvučena. Blok je na taj način jasno grafički određen u tekstu programa.

```
...
if uvjet:
      naredba1_1
      ...
```

```
        naredba1_n
else:
        naredba2_1
        ...
        naredba2_m
...
```

Niz naredbi koji smo označili sa: `naredba1_1`, `naredba1_2`, ..., `naredba1_n` zvat ćemo **blok naredbi** i on će se izvesti ako je `uvjet` istinit. Primijetimo da je isto tako niz naredbi `naredba2_1`, `naredb2_2`, ..., `naredba2_m` jedan blok naredbi koji se izvodi ako je `uvjet` lažan.

Na osnovi navedenog slijedi opći oblik naredbe `if` koji zapisujemo na sljedeći način:

```
...
if uvjet:
    blok_naredbi_1
else:
    blok_naredbi_2
...
```

U interaktivnom sučelju i u editoru nakon dvotočke se automatski uvlači sljedeći redak za četiri slovna mjesta. Naredbe koje slijede smatraju se dijelom bloka i pišu se jedna ispod druge. Nakon što se napiše zadnja naredba prvog bloka i prijeđe u novi red, potrebno je pokazivač pomaknuti za četiri mjesta ulijevo i napisati `else`:. Zatim se na isti način napiše drugi blok (u interaktivnom se sučelju blok završava tako da se pritisne tipka *unos (Enter)* čime se pojavljuje jedan prazan red).

Pogledajmo to na jednom jednostavnom primjeru.

Primjer **5.1.**

Napišimo program koji će zahtijevati utipkavanje jednog prirodnog broja i zatim ispisati je li taj broj djeljiv sa sedam.

Rješenje:

Na temelju svega što smo naučili program možemo osmisliti na sljedeći način:

```
# primjer_05_01_A
print('Program će ispisati je li neki utipkani broj djeljiv sa 7')
broj = int(input('Utipkati prirodan broj: '))
if broj % 7 == 0:
    print('Broj {} je djeljiv sa sedam.'.format(broj))
else:
    print('Broj {} nije djeljiv sa sedam.'.format(broj))
```

Podsjetimo se da funkcija `input()` utipkani broj vraća kao string te taj string treba prevesti u cijeli broj, što obavlja funkcija `int()`. Funkcija `input()` predstavlja nam, dakle, vrijednost koja će biti određena našim utipkavanjem. Zbog toga se ona može napisati neposredno kao parametar funkcije `int()`. Isprobajmo program!