Милан Чабаркапа

PYTHON – ОСНОВНИ ТИПОВИ ПОДАТАКА

Рачунар може обављати разне операције: аритметичке, сортирања, распознавања речи или облика, планирати лет ракете и још доста тога што тешко можемо и претпоставити. Међутим, у свим тим процесима он мора располагати подацима: бројевима и симболима, односно елементима који садрже информацију неопходну за правилно извршавање програма који решава одређени проблем. Подаци се чувају у меморији рачунара и могу бити разних типова (целобројни, реални, стринговни итд).

Типом се дефинише:

- колико меморијског простора заузима податак одређеног типа;
- начин регистровања податка у рачунару;
- скуп могућих вредности елемената програма који су тог типа;
- скуп операција које се могу извршавати над подацима датог типа.

Нумеричке вредности 12 и 34 припадају целобројном типу (**int**) тако да се над њима може извршавати аритметичка операција сабирања (+). Међутим, ако се исти подаци ставе између апострофа: `'12'` и `'34'`, они представљају податке типа **string**. Над њима се такође може реализовати операција +, али за стрингове она има друго значење: *конкатенацију* или *дописивање другог податка иза првог*. Резултат операције над целим бројевима $12 + 34$ је 46, а операције над стринговима: `'12' + '34'` је `'1234'`.

Од типа податка зависи какво ће бити његово унутрашње (машинско) представљање и број бајтова који ће се користити за његово регистровање. Из овога можемо закључити да сви типови података имају ограничен дијапазон вредности, што одступа од наше математичке представе о бесконачности скупова целих и реалних бројева.

Основни типови података су:

- *нумерички*: *целобројни* (означавају се са **int**) и *реални* (означавају се са **float**);
- *стринг* (низ знакова – означава се са **str**);
- *логички* (означава се са **bool**).

У програмском језику Python користи се динамичка типизација, што значи да се тип променљиве одређује према типу вредности која јој се додељује у

току извршавања програма. Према томе, у разним деловима програма иста променљива може чувати вредности разних типова (што је недопустиво у програмским језицима C/C++, C#, Java, Pascal). На пример у следећем сегменту

```
>>> x=2
>>> type(x)         # provera tipa promenljive x
<class 'int'>
>>> x='abrakadabra'
>>> type(x)         # provera tipa promenljive x
<class 'str'>
```

променљивој `x` се прво додељује целобројна вредност, а затим стринг.

Нумерички типови података (int – целобројни, float – реални)

Скуп вредности *целобројног типа* је подскуп скупа целих бројева чији дијапазон вредности је веома велики, јер за целобројну вредност интерпретер аутоматски резервише довољно меморијског простора (ако је у том моменту расположиво). У другим програмским језицима интервал целобројних вредности најчешће је од -2147483648 до 2147483648, док у Python-у можете израчунати чак 2147483648**100. Дакле, највећу целобројну вредност – у другим програмским језицима – можете степеновати са 100. Ако у интерактивном режиму пробате да ово израчунате добићете резултат у више линија.

Цели бројеви се представљају низом цифара којима може претходити знак + или −, на пример: +4, 4, −100, 1987, −1001. Знак + се може изоставити ако је број позитиван.

Реални тип у Python-у је подскуп скупа реалних бројева који се у меморији рачунара региструју другачије од целих бројева чак и када заузимају исти број бајтова. Реални подаци у Python-у се региструју у 8 бајтова и означавају са **float**.

Запис реалних бројева се разликује од записа целих бројева по томе што садржи децималну тачку која раздваја целобројни и разломљени део. На пример, реални бројеви су 2.34, −11.657, 0.5, −0.6. Ако је целобројни део реалног броја 0, запис може почињати децималном тачком, на пример: .98, .65. Такође, ако реалан број има нулу иза децималне тачке, на пример 12.0, тада се може писати као 12. – дакле без нуле.

Реални бројеви се могу приказивати и у научној нотацији. Како већина програмских едитора немају могућност приказивања експонента броја и специјалног симбола, Python незнатно модификује научну нотацију. На пример, реалан број се у Python-у приказује као 7.022*e*21. Реалан број лево од *e* (може се писати и велико *E*) назива се *мантиса*, а десно *експонент* броја 10. Симбол *e* (или *E*) у запису броја изражава децимални поредак и чита се „помножити са 10 на“. На пример:

Запис у програмирању	Математички запис
$2.36E6$	$2.36 \cdot 10^6$ или 2 360 000
$-0.35E-4$	$-0.35 \cdot 10^{-4}$ или -0.000035
$123.45E-2$	$123.45 \cdot 10^{-2}$ или 1.2345

Број који је написан у програму назива се нумерички литерал, на пример: 12.34, 23, 21., .34, итд.

Неки реални бројеви се у Python-у представљају приближно. На пример, број π има бесконачно много цифара и, очекивано, може се само приближно регистровати. Међутим, због ограниченог броја бајтова који се користе за регистровање реалних бројева, у Python-у се и неки бројеви са коначним бројем цифара региструју приближно. На пример, број 99.123909090909089898 Python ће регистровати као 99.1239090909091:

```
>>> x=99.123909090909089898
>>> x
99.1239090909091
```

Када неку вредност сачувате у меморији, важно је да знате који је њен тип јер од тога зависи у којим операцијама може учествовати. Тип податка можете проверити у интерактивном режиму помоћу уграђене функције `type()`. На пример, следећом инструкцијом:

```
>>> type(1)
<class 'int'>
```

установљено је да је **int** – тип нумеричког литерала 1. Док за нумерички литерал 1.0:

```
>>> type(1.0)
<class 'float'>
```

добијамо да му је тип **float**.

У следећој табели дате су операције које се могу реализовати над **int** и **float** подацима.

Операција	Оператор	Коментари
Сабирање	`x + y`	`x` и `y` могу бити **float** или **int** типа
Одузимање	`x - y`	`x` и `y` могу бити **float** или **int** типа
Множење	`x * y`	`x` и `y` могу бити **float** или **int** типа
Дељење	`x / y`	`x` и `y` могу бити **float** или **int** типа. Резултат је увек **float** типа. На пример: $5/1.6 = 3.125$; $6/3 = 2.0$.

Операција	Оператор	Коментари
Целобројно дељење	`x // y`	`x` и `y` могу бити **float** или **int** типа. Резултат је највећи цео број који је мањи или једнак количнику `x` и `y`. Вредност резултата је целобројна ако су оба аргумента цели бројеви. На пример: $11.8//4 = 2.0$; $11.0//4 = 2.0$; $11//4 = 2$.
Остатак при дељењу	`x % y`	`x` и `y` могу бити **float** или **int** типа. Резултат је остатак при дељењу `x` са `y`. На пример: $11 \% 4 = 3$; $11.8 \% 4 = 3.800000000000000007$ – појављује се грешка због нетачног представљања података у рачунару.
Цели део и остатак при дељењу	`divmod(x,y)`	Враћа два броја: `p` – цели део и `q` – остатак при дељењу `x` са `y`. На пример, за `(p,q)=divmod(11,4)` `p` добија вредност 2, а `q` вредност 3.
Степеновање	`x ** y`	`x` и `y` могу бити **float** или **int** типа. Резултат је x^y. На пример, $2{*}{*}3 = 8$.
Конверзија у **float**	`float(x)`	Функција конвертује нумеричку или стринг вредност у **float**. На пример: `float`$(22) = 22.0$; `float('22')` $= 22.0$.
Конверзија у **int**	`int(x)`	Функција конвертује нумеричку вредност `x` у **int** тако што одбацује децимални део. Стринг `x` у коме је цео број конвертује у **int**. На пример: `int`$(22.34) = 22$.
Апсолутна вредност	`abs(x)`	Функција враћа апсолутну вредност од `x`. На пример: `abs`$(-2) = 2$.
Заокругљивање	`round(x)`	Заокругљује реалан број `x` на најближи цели број. Резултат је увек **int** типа. На пример: `round`$(12.34) = 12$; `round`$(6.66) = 7$; `round`$(6.5) = 6$.
Заокругљивање на `n` децимала	`round(x,n)`	Заокругљује реалан број `x` на `n` цифара иза децималне тачке. На пример: `round`$(12.345467, 3) = 12,346$.

Резултат аритметичке операције (`+, -, *, /, //, %`) између целобројне (**int**) и реалне (**float**) вредности увек је реална вредност.

Аритметички изрази

Израз је синтаксна јединица језика, која дефинише поредак и начин израчунавања неке вредности коришћењем:

- операнада (константе, промењљиве, функције);
- операцијских симбола;
- овалних заграда.

Операција дефинише акције које треба извршити над операндима. За разлику од традиционалног математичког записа, неопходно је писати све операцијске знаке. На пример, математички израз $5(2x + 3y)$ у Python-у се мора писати навођењем симбола за операцију множења ($*$):

```
5*(2*x+3*y).
```

Све операције се деле на *унарне* и *бинарне*. Унарне операције дефинишу акције на једном операнду, а бинарне – на два. На пример, `-a` је унарна операција, а `a+b` бинарна. Не препоручује се писање два операцијска знака непосредно један за другим: `a*-b`. У овом случају, боље је други операнд ставити између заграда: `a*(-b)`. Резултат је у оба случаја исти јер је унарни минус вишег приоритета од операције множења.

Приоритет аритметичких операција

Приоритет аритметичких операција дефинише редослед извршавања операција у изразу. Приоритет операција је дат следећом табелом:

Операција	Коментар
1. `**`	Степеновање
2. `-`	Промена знака
3. `*,/,//,%`	Множење, дељење, целобројно дељење, остатак при дељењу
4. `+,-`	Сабирање и одузимање

При одређивању приоритета, поштују се следећа четири основна правила:

1. Операнд који се налази између две операције различитог приоритета, повезује се са операцијом вишег приоритета.
2. Операнд који се налази између две операције истог приоритета, повезује се са операцијом која му је са леве стране.
3. Израз између заграда се израчунава као посебан операнд.
4. Операције истог приоритета се извршавају слева надесно, изузев операције степеновања, која се извршава здесна улево.

Приоритет се може регулисати коришћењем заграда. Када су заграде уметнуте једне у друге по дубини, израчунавање је почев од најдубљих. При писању израза увек се мора контролисати упареност заграда: број отворених заграда мора бити једнак броју затворених заграда.

У следећој табели дати су неки аритметички изрази и њихове вредности.

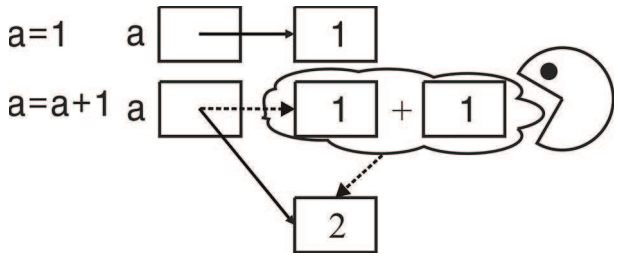
Аритметички израз	Израчунавање израза	Вредност израза
`1 + 2 * 3`	`1 + 6`	7
`(1 + 2) * 3`	`3 * 3`	9
`4 % 2`	`0`	0
`2 * 3 ** 2`	`2 * 9`	18
`2 ** 3 ** 2`	`2 ** 9`	512
`(2 ** 3) ** 2`	`8 ** 2`	64
`32 / 0`	`ZeroDivisionError: division by zero`	
`32 % 0`	`ZeroDivisionError: division by zero`	

Треба имати у виду да оно што је у математици нетачно, на пример, конструкција `a=a+1` (ако знак = третирамо као једнакост), у програмирању има смисла. Ово је за програмера апсолутно исправна конструкција: израчунава суму вредности која се налази у објекту на који реферише променљива `a` и вредности 1 која се налази у другом објекту, и добијени резултат уписује у нови објекат чију адресу додељује променљивој `a`. Сада је вредност на коју реферише променљива `a` за 1 већа од претходне вредности.

Према томе, извршавањем следећег програмског сегмента:

```
a=1
a=a+1
print(a)
```

исписује се: 2.



Доделе као у претходном примеру:

```
<promenljiva>=<promenljiva><znak><izraz>
```

у којима се вредност променљиве увећава за вредност неког израза могу се писати у скраћеном облику

```
<promenljiva><znak>=<izraz>
```

где `<znak>` може да буде: `*, /, //, %, +, -, **`.

На пример, додела:

`x+=2` исто је што и `x=x+2`,

`x-=2` исто је што и `x=x-2`,

`x/=2` исто је што и `x=x/2`,

`x*=2` је исто што и `x=x*2`,

`x%=2` је исто што и `x=x%2`.

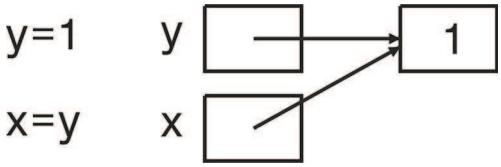
С обзиром да се операција доделе реализује здесна налево у Python програмима може се једном инструкцијом иницијализовати више променљивих.

На пример, инструкција

```
x=y=1
```

еквивалентна је пару инструкција:

```
y=1
x=y
```



што значи да се инструкција извршава тако што прво променљива `y` након доделе: `y=1` реферише на објекат у коме је вредност 1, а како следећом доделом `x=y` променљива `x` добија вредност променљиве `y`, то ће и променљива `x` референсати на објекат у коме је вредност 1.

С друге стране, више променљивих се може иницијализовати различитим вредностима у једној инструкцији коришћењем запете за раздвајање променљивих и литерала. На пример, инструкцијом:

```
a, b, c = 1, 2, 3
```

променљива `a` добија вредност 1, променљива `b` добија вредност 2 и променљива `c` добија вредност 3.

Пример 1. Написати програм који одређује суму цифара учитаног троцифреног броја.

Проблем ћемо решити на два начина:

I. издвајањем цифара датог броја слева надесно;

II. издвајањем цифара датог броја здесна налево

а затим сумирањем цифара.

Прво решење. Издвајање цифара слева надесно ћемо илустровати на примеру конкретног броја, рецимо 629.

Прва цифра = 6	Прва цифра слева се може издвојити издвајањем целобројног дела датог броја при дељењу са 100: `cifra1=629 // 100` # даје 6
Преостале цифре = 29	Преостале цифре се могу издвојити издвајањем остатка при дељењу датог броја са 100: `r=629 % 100` # даје 29
Друга цифра = 2	Друга цифра се може добити издвајањем целобројног дела при дељењу остатка (29) са 10: `cifra2=29 // 10` # даје 2
Трећа цифра = 9	Последња, трећа цифра, може се добити као остатак при дељењу 29 са 10: `cifra3=29 % 10` # даје 9

Након претходне анализе, у програму за издвајање цифара само треба број из претходног примера заменити унетим бројем:

```
n=int(input("Unesi trocifren broj:  "))
cifra1=n // 100                    # izdvaja cifru stotina
r=n % 100                          # izdvaja ostatak
cifra2=r // 10                     # izdvaja cifru desetica
cifra3=r % 10                      # izdvaja cifru jedinica
suma=cifra1+cifra2+cifra3
print("Suma cifara= ",suma)
```

Извршавањем програма исписује се:

```
Unesi trocifren broj:  629
Suma cifara= 17
```

Програм се може написати и концизније коришћењем функције `divmod()`:

```
n=int(input("Unesi trocifren broj:  "))
cifra1, r = divmod(n, 100)         # izdvaja prvu cifru i ostatak
cifra2, cifra3 = divmod(r, 10)     # izdvaja drugu i trecu cifru
suma=cifra1+cifra2+cifra3
print("Suma cifara= ",suma)
```

Друго решење. Издвајање цифара здесна налево ћемо илустровати на примеру истог броја.

Трећа цифра=9	Трећа цифра може се добити издвајањем остатка при дељењу датог броја са 10: `cifra3 = 629 % 10` # издваја цифру 9
Преостале цифре=62	Преостале цифре након уклањања треће цифре могу се добити целобројним дељењем задатог броја са 10: `r = 629 // 10` # издваја 62
Друга цифра=2	Друга цифра се може добити издвајањем остатка при дељењу преосталих цифара са 10: `cifra2 = r % 10` # издваја 2
Трећа цифра=6	Трећа цифра се може издвојити целобројним дељењем преосталих цифара са 10: `cifra3 = r // 10` # издваја 6

Након претходне анализе, у програму за издвајање цифара само треба број из претходног примера заменити унетим бројем:

```
n=int(input("Unesi trocifren broj:  "))
```

```
cifra3=n % 10                    # izdvaja cifru jedinica
r=n // 10                        # izdvaja preostale cifre
cifra2=r % 10                    # izdvaja cifru desetica
cifra1=r // 10                   # izdvaja cifru stotina
suma=cifra1+cifra2+cifra3
print("Suma cifara= ",suma)
```

Програм се може написати и концизније коришћењем функције `divmod()`:

```
n=int(input("Unesi trocifren broj:  "))
r, cifra3 = divmod(n, 10)
cifra1, cifra2 = divmod(r, 10)
suma=cifra1+cifra2+cifra3
print("Suma cifara= ",suma)
```

Тип стринг

За рад с текстуалним информацијама у Python-у се користи тип **стринг** у ознаци `str`. Стринг литерал је низ знакова језика између пара апострофа или наводника. На пример, стринг литерали су:

```
"Ovo je string"
'Ovo je string'
''
""
```

Гранични апострофи или наводници не улазе у састав стринг литерала, већ само указују да сви симболи који се налазе између њих представљају једну целину. Последња два стринг литерала (`''` и `""`) репрезентују празан стринг. Треба имати на уму да празан стринг није исто што и стринг који садржи празнину: `" "`.

Два пара граничника постоје да би се унутар стринга могли користити апострофи или наводници. На пример:

```
'Petar kaze:  "Zdravo Ana"'
"Ana kaze:  'Zdravo Pero'"
```

Ако бисте покушали да унутар стринга ограниченог наводницима ставите наводнике добићете поруку да сте направили синтаксну грешку. На пример:

```
>>> S="Novak Djokovic - "Nole""
SyntaxError:  invalid syntax
```

Ово ће се догодити због тога што Python интерпретер када наиђе на други наводник – који желите да буде део стринга – сматра да је то крај стринга и не зна шта да ради са симболима који се налазе иза њега. У Python-у се овај проблем превазилази коришћењем специјалних симбола који се називају управљачке или escape-секвенце. Иако се пишу са два симбола Python их види као један. Када се испред симбола појави backslash симбол (\), то је најава да симбол који се налази иза њега има специјално значење. Према томе, пар симбола \" унутар стринга

значи да на том месту није крај стринга већ да треба исписати наводник. Такође, пар симбола \' унутар стринга указује да на том месту треба исписати апостроф. Претходна синтаксна грешка се може уклонити ако се испред наводника стави backslash:

```
>>> S="Novak Djokovic - \"Nole\""
>>> print(S)
Novak Djokovic - "Nole"
```

Функција `print()` исписује стринг без граничних симбола.

Управљачка секвенца која се највише користи је за прелазак у нови ред. Она се пише као следећи пар симбола: `\n`. На пример:

```
>>> print("A\nBC\nDEF")
A
BC
DEF
```

Ако у стринг треба да ставите обратну косу црту, просто ставите две: \\.

За поравнавање у тексту користи се управљачка секвенца `'\t'`. На пример:

```
>>> print("AAA\tBBB\tCCC\nDDD\tEEE\tFFF")
AAA     BBB     CCC
DDD     EEE     FFF
```

Стринг литерали се могу писати и између три наводника или три апострофа. Такве стринг литерале можете писати у више линија и унутар њих се могу наћи апострофи или наводници (али не три узастопна) који немају префикс \, као најаву escape-секвенце. На пример:

```
>>> S='''Na sta mislis
kad kazes
"bijelo dugme"'''
>>> S
'Na sta mislis\nkad kazes\n"bijelo dugme"'
>>> print(S)
Na sta mislis
kad kazes
"bijelo dugme"
```

Симбол `'\n'` се уписује у стринг при сваком притиску на тастер `<Enter>`. Свако појављивање овог симбола у стрингу који се исписује коришћењем функције `print()` изазива прелазак у следећи ред.

Задаци за вежбу

1. Од датог троцифреног броја формирај нови број који се добија инвертовањем његових цифара. На пример, од 976 добија се 679.

2. Од датог троцифреног броја формирати нови који се добија уклањањем прве цифре слева и дописивањем здесна. На пример, од 378 добија се 783.
3. Од датог троцифреног броја формирати нови који се добија уклањањем прве цифре здесна и дописивањем слева. На пример, од 378 добија се 837.
4. У датом природном броју већем од 999 одредити цифру стотина. На пример, за 34898 добија се 8.
5. Написати програм којим се избацује цифра десетица у природном броју `x`.
6. Написати програм којим се у природном броју `x` (`x>=100`) замењују места цифре јединица и цифре стотина.
7. Дани недеље су нумерисани на следећи начин: 1 – понедељак, 2 – уторак, . . . , 6 – субота, 7 – недеља. За дати цео број `K` између 1 и 365 одредити који је `K`-ти дан у недељи године у којој је:

- 1. јануар уторак. На пример, за `K=2` – дан у недељи је 3 (среда). Упутство: `K % 7+1`.
- 1. јануар субота. На пример, за `K=3` – дан у недељи је 1 (понедељак). Упутство: `(K+4) % 7+1`.

Математичка гимназија, Краљице Наталије 37, Београд
E-mail: mcabark@gmail.com