

Електротехничка школа „Стари град“, Београд
профил Електротехничар информационих технологија
предмет Програмирање, кроз четири године учења

2.разред, фонд часова 2 часа теорије и 2 часа вежби седмично

материјал према часовима теорије и лабораторијским вежбама са програмским језиком Пајтон

материјал за ученике

верзија 1.0

јун 2021

обрађене теме:

- листе
- функције
- матрице
- стрингови
- речници и сетови
- датотеке
- динамичке структуре

наставник Ранковић Небојша

Увод у листе.....	15
Дефиниција листа.....	15
Конверзија у листе	15
Оператор понављања	15
001 Употреба оператора понављања	16
Функција len().....	16
002 Добијање дужине листе	16
Промењивост листа	16
003 Модификација садржаја листе	16
004 Појава изузетка употребом недозвољеног индекса листе	16
Надовезивање листа	17
005 Надовезивање две нумеричке листе	17
006 Надовезивање две стринг листе	17
Итерација по листи са for петљом	17
007 Итерирање по листи са for петљом.....	17
Индексирање	17
008 Штампане елемената листе помоћу петље	18
009 Штампане елемената листе негативним индексима.....	18
010 Унос и приказ n елемената листе	18
011 Унос и приказ n елемената торке.....	18
012 Приказ елемената са непарним индексима	19
013 Приказ елемената у обрнутом редоследу	19
Питања и задаци за самосталан рад	19
Одсецање листа.....	20
014 Одсецање листе према синтакси	20
015 Одсецање изостављањем индекса start	20
016 Одсецање изостављањем индекса kraj	20
017 Одсецање изостављањем оба индекса	21
018 Одсецање коришћењем три индекса	21
019 Одсецање коришћењем негативних индекса	21
Грешке са индексима	21
Задаци са одсецањем листа.....	21
020 Штампане елемената листе преко while петље.....	21
021 Замена са 0 вредности на парним индексима.....	21
022 Надовезати листе са појачаним оператором.....	22
023 Поделити елементе листе	22
Питања и задаци за самосталан рад	22
Претрага листе	22

Оператор in	23
024 Употреба оператора in.....	23
Оператор not in.....	23
025 Употреба оператора not in.....	23
Задаци са претрагом листа.....	24
026 Проналажење жељеног елемента	24
027 Проналажење броја жељеног елемента	24
028 Проналажење истих елемената у две листе	24
029 Количина бројева дељивих са 3 у низу бројева	25
Питања и задаци за самосталан рад	25
Методe листе	25
Метода append	25
030 Унос новог елемента у листу	25
Метод index.....	26
031 Добијање позиције елемента у листи	26
Метод insert	26
032 Уметање вредности на одређену позицију	26
Метода remove	26
033 Одстрањивање елемента према вредности	26
Исказ del	27
034 Одстрањивање елемента према индексу	27
Метода clear.....	27
035 Одстрањивање свих елемената из листе	27
Метода sort	27
036 Сортирати бројеве према вредности	27
037 Сортирати стрингове у листи	27
Метода reverse	27
038 Обрнути редослед елемената у листи	27
Метода count	28
039 Пребројити појављивање елемента	28
Метода pop	28
040 Уклањање елемента према индексу	28
Метода extend	28
041 Придруживање елемената две листе	28
Функције min и max.....	28
042 Открити највећи и најмањи број листе	28
Копирање листа.....	29

043 Копирање листе додељивањем.....	29
044 Копирање листе помоћу петље.....	29
045 Копирање листе надовезивањем	29
Метода сору.....	29
046 Копирање листе методом сору	29
Задачи методе листе.....	29
047 Употреба методе append.....	30
048 Сабирање вредности у листи	30
049 Удруживање елемената две листе	30
Питања и задаци за самосталан рад	31
List Comprehension	31
050 Употреба просте list comprehension	31
051 Употреба сложеније list comprehension	31
052 Селектовање унетих вредности	31
053 Селектовање и измена вредности	32
054 Селектовање вредности према типу.....	32
055 Селектовање стрингова из листе.....	32
056 Селектовање вредности преко методе index	33
Питања и задаци за самосталан рад	33
Дефинисање функције.....	33
Увод у функције	34
Дефинисање функције	34
Позивање функције.....	34
057 Дефинисање и позивање функције	34
058 Дефинисање и позивање функције са main()	35
Локалне промењиве	35
059 Ограничења локалних промењивих.....	35
Досег.....	36
060 Приказ досега промењивих истог имена.....	36
Допремање аргумената функцији	37
061 Употреба аргумента у позиву.....	37
062 Употреба main() функције	37
Допремање већег броја аргумената	38
063 Допремање више аргумената.....	38
Промена у параметрима	38
064 Измена вредности параметара	38
Аргументи службене речи	39

065	Употреба аргумента службене речи	39
Задаци са void функцијама		39
066	Цртање 10 знакова у једном реду.....	40
067	Цртање n знакова у једном реду	40
068	Цртање знакова у више редова	40
069	Цртање правоугаоника 10 x 5.....	40
Питања и задаци за самосталан рад		41
070	Цртање квадрата странице n.....	41
071	Употреба претраге као void функције	41
072	Рачунање квадрата броја преко void функције	42
073	Цртање дијагонале знаковима	42
Питања и задаци за самосталан рад		43
Глобалне промењиве.....		43
074	Поставка глобалне промењиве у коду.....	43
075	Утицај функција на промену вредности глобалне промењиве	44
Лоше особине глобалних промењивих		44
Глобалне константе.....		45
076	Употреба глобалних константи	45
Функције које враћају вредност.....		45
Функције стандардне библиотеке и исказ import.....		45
Генерисање случајних бројева.....		46
077	Генерисање целог случајног броја у опсегу вредности	46
078	Генерисање више случајних бројева у опсегу.....	46
Примери рада са random модулом		47
079	Симулација бацања две стандардне коцкице	47
080	Симулација бацања новчића	47
Функције randrange, random, uniform		47
Просејавање при генерисању случајних бројева		48
081	Доказ добијања исте секвенце псеудослучајних бројева	48
Писање сопствених функција које враћају вредност		49
082	Дефиниција сопствене функције која враћа вредност	49
083	Код са сопственом функцијом	49
Употреба функција које враћају вредност		49
084	Рачунање попушта на цену робе	50
Сопствене функције које враћају стринг и булове вредности		50
085	Функција која враћа стринг	50
086	Испитивање парности.....	50
087	Унос одређеног броја	51

Функције које враћају више вредности.....	51
088 Дефинисање функције која враћа више вредности	51
Модул math.....	52
089 Рачунање хипотенузе	52
Смештање функција у модуле	52
090 Рачунање површине и обима.....	53
Достављање листе као аргумента функције.....	54
091 Функција која рачуна суму вредности из листе бројева	54
Враћање листе из функције.....	55
092 Петља while са листом непознате дужине у функцији	55
093 Рачунање средње вредности са тестова	55
Задаци са return функцијама.....	56
094 Испитивање парности са Буловим вредностима	56
095 Рачунање степена унетог броја	56
096 Ревизија рачунања степена унетог броја	57
Питања и задаци за самосталан рад	57
097 Рачунање апсолутне вредности	57
098 Упоредивање случајних бројева	58
099 Разлика реципрочних бројева	59
Питања и задаци за самосталан рад	59
0100 Функције за рад са листама.....	59
0101 Претварање радијана у степене	60
0102 Заокруживање бројева.....	61
Питања и задаци за самосталан рад	61
Матрице.....	61
Листе.....	61
Вишедимензионалне листе.....	62
Дводимензионалне листе - матрице.....	62
Иницијализација матрица	62
0103 Иницијализација матрице	62
Иницијализација матрица са истим елементима	62
0104 Иницијализација матрице list comprehension.....	62
Изостављање појединих елемената матрице	63
0105 Матрица као листа.....	63
Форматирање матрице помоћу константи	63
0106 Попуњавање матрице преко константи.....	63
Приказ елемената матрице	64

0107 Приказ матрице као табеле	64
Унос новог реда у постојећу матрицу.....	64
0108 Модификација матрице уносом реда	64
Приступање елементима матрице	65
Квадратна матрица	65
0109 Креирање квадратне матрице	66
0110 Квадратна матрица са случајним бројевима	66
Спирална матрица.....	67
Читање матрице као спиралне	67
0111 Читање матрице спирално	67
Задачи са матрицама	68
0112 Креирање матрице функцијама	68
0113 Сумирање елемената матрице	69
0114 Корисничко креирање матрице	69
Питања и задаци за самосталан рад	70
0115 Рачунање суме матрице по редовима	70
0116 Пребројавање елемената матрице	71
Питања и задаци за самосталан рад	73
0117 Креирање матрице непознатих димензија	73
0118 Уређење матрице по дијагоналама	74
0119 Сумирање матрице по дијагоналама	74
Питања и задаци за самосталан рад	76
Стрингови	76
Приступање појединачним карактерима у стрингу	76
Итерација по стрингу са for петљом	77
0120 Итерација преко карактера у стрингу	77
0121 Број пута појаве карактера у стрингу	77
Индексирање	77
0122 Приступ стрингу индексирањем	77
0123 Вишеструко индексирањем	78
Грешка IndexError	78
0124 Појава IndexError изузетка	78
Функција len	78
0125 Излаз функције len()	78
0126 Спречавање итерације преко опсега индекса	78
Надовезивање стрингова	79
0127 Просто надовезивање стрингова	79

0128 Употреба += оператора у надовезивању стрингова	79
Непромењивост стрингова	79
0129 Приказ непромењивости стринга	79
0130 Недозвољено коришћење оператора доделе	80
0131 Доказ непромењивости стринга	80
Одсецање стрингова	80
0132 Одсецање стринга са два индекса	81
0133 Одсецање стринга са десним индексом	81
0134 Одсецање стринга са левим индексом	81
0135 Одсецање стринга без индекса	81
0136 Одсецање стринга са три индекса	81
0137 Одсецање стринга са негативним индексом	82
0138 Креирање шифре одсецањем стринга	82
Тестирање стрингова	82
0139 Употреба оператора in	82
Стринг методе	83
Стринг методе за тестирање	83
0140 Метода isdigit()	83
0141 Методе за тестирање стринга	83
Методе за модификацију стрингова	84
0142 Метода lower()	84
Методе за претрагу и замену	85
0143 Методе за претрагу стринга	85
0144 Метода find()	85
0145 Метода за замену стринга	85
Оператор понављања	86
0146 Употреба оператора понављања	86
Раздвајање стрингова	86
0147 Употреба методе раздвајања стрингова	86
0148 Употреба знаковног оператора	86
0149 Читање делова стрингова помоћу сепаратора	87
Задаци са стринговима	87
0150 Приказ унетих стрингова	87
0151 Употреба функција ASCII кодова	87
0152 Постављање карактера на индекс	88
Питања и задаци за самосталан рад	88
0153 Замена карактера на индексу	88
0154 Замена малих са великим словима	89
Питања и задаци за самосталан рад	89

0155 Избор дела стринга	90
0156 Претрага дела стринга	90
0157 Проналажење цифре у стрингу	91
Питања и задаци за самосталан рад	92
0158 Проналажење великих слова стринг методом	92
0159 Конвертовање датума у другачији формат.....	92
Питања и задаци за самосталан рад	93
Речници и сетови	93
Речници - структура.....	93
Креирање речника	94
0160 Креирање речника	94
Добијање вредности из речника	94
Добијање броја елемената у речнику	94
0161 Приказ дужине речника	94
Употреба различитих типова података у речнику.....	95
0162 Употреба листа у речницима.....	95
Употреба празног речника	95
0163 Употреба празног речника	95
Итерације у речнику.....	95
0164 Итерација преко кључева у речнику.....	96
Оператори in и not in.....	96
0165 Претрага речника без подизања изузетака	96
Додавање елемената у постојећи речник	96
0166 Додавање елемента у речник	96
Брисање елемента из речника.....	97
Метода clear.....	97
0167 Употреба методе clear().....	97
Метода get	97
0168 Употреба методе get()	97
Метода items.....	98
0169 Употреба dictionary view.....	98
0170 Употреба итерације за приступ торкама	98
Метода keys	98
0171 Употреба методе keys()	98
0172 Употреба итерације са методом keys()	98
Метода pop	99

0173 Употреба методе <code>pop()</code>	99
Метода <code>popitem()</code>	99
0174 Употреба методе <code>popitem()</code>	100
Метода <code>values</code>	100
0175 Употреба методе <code>values()</code>	100
Креирање сета	100
Добијање броја елемената у сету	101
Додавање елемената у сет	101
0176 Употреба методе <code>add()</code>	101
0177 Употреба методе <code>update</code> са целим сетовима	101
0178 Употреба методе <code>update</code> са различитим типовима података	101
Одстрањивање елемената из сета	102
0179 Методе одстрањивања елемената из сета	102
0180 Метода <code>clear()</code>	102
Коришћење петље за итерацију преко сета	102
0181 Итерација преко сета	102
Коришћење оператора <code>in</code> и <code>not in</code> за тестирање вредности у сету	103
0182 Употреба оператора	103
Добијање уније сетова	103
0183 Унија сетова	103
Добијање пресека сетова	103
0184 Пресек сетова	103
Добијање разлике сетова	104
0185 Разлика сетова	104
Добијање симетричне разлике сетова	104
0186 Симетрична разлика сетова	104
Добијање субсета и суперсета	104
0187 Субсет и суперсет	105
Задаци са речницима и сетовима	105
0188 Креирање квива са речницима	105
0189 Телефонски именик са речницима са уносом	105
0190 Телефонски именик са речницима са брисањем	106
Питања и задаци за самосталан рад	106
0191 Телефонски именик са речницима са изменама	106
0192 Употреба речника за конверзију у шифру	107
0193 Употреба речника за конверзију у стринг	107
Питања и задаци за самосталан рад	108

0194 Конверзија стринга у сет	108
0195 Употреба сетова	108
Питања и задаци за самосталан рад	109
Увод у рад са датотекама	109
Увод у улазне и излазне фајлове	109
Кораци у коришћењу фајлова	110
Типови фајлова	110
Методе за приступ фајловима	110
Фајлови као објекти	110
Отварање фајла	111
0196 Отварање фајла за читање	111
0197 Отварање фајла за упис	111
Одређивање локације фајла	111
0198 Рад са фајлом на другој локацији	112
Упис података у фајлове	112
0199 Затварање фајла	113
0200 Отварање, упис и затварање истог фајла	113
Читање података из фајла	114
0201 Отварање фајла за читање	114
0202 Читање фајла по линијама	114
Надовезивање нове линије у стринг	115
0203 Надовезивање линија	116
Читање стринга и одстрањивање \n из стринга	116
0204 Употреба методе rstrip()	116
0205 Употреба методе у уређивању стрингова	117
Придодавање података у постојећи фајл	117
0206 Придодавање података у фајл	117
Упис и читање нумеричких података	118
0207 Конверзија бројног податка	118
0208 Конверзија током читања	119
Коришћење петљи у раду са фајловима	119
0209 Употреба фајлова у трговини	119
Читање фајла помоћу петље и детекција краја фајла	120
0210 Читање са детекцијом краја линије	120
Коришћење for петље за читање линија	121
0211 Итерација for петље по линијама фајла	121

0212 Манипулација фајлова са подацима о филмовима	122
Увод у записе	122
0213 Упис записа о запосленима у фајл	123
0214 Читање записа о запосленима	124
Тип података bytes	124
0215 Тип података bytes	125
Упис бинарних података у фајл.....	125
0216 Бинарни упис	125
Читање бинарних података из фајла	125
0217 Бинарно читање.....	125
Рад са бројевима у бинарним фајловима.....	125
0218 Објекти у бинарни фајл	125
Декодирање и кодовање текста у бинарни формат	126
0219 Декодирање	126
0220 Кодовање.....	126
0221 Декодирање и кодовање текста у бинарни формат.....	126
Задаци са датотекама	126
0222 Основно манипулисање фајлом	126
0223 Корисничко именовање фајла	127
0224 Приказ кориснички именованог фајла	127
0225 Креирање изгледа података у фајлу.....	127
Питања и задаци за самосталан рад	127
0226 Сабирање бројчаних података из фајла.....	128
0227 Манипулација подацима у два фајла	128
0228 Приказ садржаја фајла по линијама	129
Питања и задаци за самосталан рад	129
0229 Придодавање новог садржаја у фајл	129
0230 Брисање одређеног садржаја из фајла	130
0231 Модификација одређеног садржаја у фајлу.....	130
Питања и задаци за самосталан рад	131
0232 Смештање података о филмовима у фајлу.....	131
0233 Читање записа о запосленима у фајлу	132
0234 Уметање белине у текст у фајлу	133
Питања и задаци за самосталан рад	134
Стек	134
Стек као тип података	134
Рад са стеком у Пајтону.....	135
0235 Симулација стека употребом листа.....	135

Структура реда у програмирању.....	136
Ред	137
Ред као имплементација листе	137
0236 Симулација реда употребом листа	137
Ред као имплементација класе collections.deque	137
0237 Симулација реда употребом класе deque.....	138
Задаци са стеком и редом	138
0238 Манипулација листом бројева као стеком	138
0239 Манипулација листом стрингова као стеком.....	139
0240 Манипулација методом deque као стеком	139
Питања и задаци за самосталан рад	140
Изузеци.....	140
Грешке у Пајтону	140
0241 Генерисање синтаксне грешке и изузетка	140
Подизање изузетка	141
0242 Употреба команде raise.....	141
Питања и задаци за самосталан рад	141
Појава AssertionError изузетка	141
0243 Употреба тврдње assertion	141
Руковање изузецима.....	142
0244 Блокови try и except	142
0245 Реакција на AssertionError	142
Питања и задаци за самосталан рад	143
Руковање вишеструким изузецима	143
0246 Исписивање поруке о неизвршеној функцији	143
Питања и задаци за самосталан рад	144
Команда else	144
0247 Употреба else блока.....	144
0248 Блок else са хватањем изузетка.....	145
Употреба finally команде	145
0249 Употреба finally.....	146
Врсте изузетака.....	147
Задаци са изузецима.....	148
0250 Примена изузетка на контролу температуре.....	148
0251 Примена изузетка на контролу уноса	148
Питања и задаци за самосталан рад	149

Рекурзије	149
Увод у рекурзије	149
0252 Рекурзија као бесконачна петља.....	149
0253 Рекурзија са контролом прекида	150
Употреба рекурзија	150
Рачунање факторијала коришћењем рекурзије	151
0254 Рачунање факторијала рекурзијом.....	151
0255 Сумирање рекурзијом.....	152
Задаци са рекурзијама	152
0256 Множење рекурзијом	152
0257 Претрага највећег елемента рекурзијом	153
0258 Фибоначијев низ рекурзијом	153
Питања и задаци за самосталан рад	153
Претраге и сортирање.....	154
Секвенцијално претраживање листа	154
0259 Секвенцијална претрага	154
Број појављивања елемената у листи	155
0260 Секвенцијално бројање елемената.....	155
0261 Претрага елемента у сортираној листи	155
0262 Претрага најмањег елемента у листи.....	156
Бинарна претрага сортиране листе	156
0263 Бинарна претрага	156
Сортирање секвенце.....	157
Сортирање методом избора	157
0264 Метода избора у сортирању.....	157
Сортирање методом замене суседа.....	158
0265 Метода замене суседа у сортирању.....	158
Bubble-sort.....	159
0266 Метода сортирања bubble-sort.....	159
Задаци претраге и сортирања	160
0267 Секвенцијално претраживање броја у листи	160
0268 Бинарно претраживање броја у листи	161
0269 Детекција појаве броја у листи.....	161
0270 Сортирање листе методом избора.....	162
0271 Сортирање листе методом bubble-sort	163
Питања и задаци за самосталан рад	163
Коришћена литература	164

Увод у листе

Дефиниција листа

Листа је објекат који садржи више ствари који су заправо подаци.

Свака од ствари смештена у листи је елемент листе.

Исказ који креира листу целих бројева:

```
parni_brojevi = [2, 4, 6, 8, 10]
```

Ствари у угластим заградама и раздвојене зарезима су елементи листе.

Када се изврши претходни исказ, промењива `parni_brojevi` ће указивати на листу.

Елементи у листи не морају да буду бројеви:

```
imena = ['Miki', 'Kiki', 'Jovan', 'Ana', 'Dragana']
```

Такође, листа може садржати елементе различитих типова података:

```
info = ['Dejan', 30, 34.589]
```

Функција `print` приказује изглед целе листе:

```
print(info)
```

даје:

```
['Dejan', 30, 34.589]
```

Конверзија у листе

Пајтон има уграђену функцију `list()` која конвертује неке типове објеката у листе.

Нпр, пошто је `range()` функција која враћа итерабилни податак, а то је објекат који садржи серију вредности преко које се може итерирати.

Може се користити исказ попут:

```
brojevi = list(range(5))
```

После извршења исказа дешава се следеће:

- Функција `range` се позива са 5 достављеним као аргументом, функција враћа итерабилне вредности 0, 1, 2, 3, 4
- Итерабилна вредност је пренешена као аргумент у функцију `list()`, функција `list()` враћа листу [0, 1, 2, 3, 4]
- `lista[0, 1, 2, 3, 4]` је додељена `brojevi` промењивој

```
broj = list(range(1, 10, 2))
```

Када се придодају три аргумента у функцију `range`, први аргумент је почетна вредност, други лимитира низ а трећи је вредност корака.

Зато ће листа `broj` имати следеће елементе:

```
[1, 3, 5, 7, 9]
```

Оператор понављања

Симбол `*` се користи оператор математичке операције множења.

Али, ако је операнд са леве стране оператора `*` секвенца (попут листе) а са десне стране оператора `*` је целобројна вредност, оператор `*` постаје оператор понављања (`repetition operator`).

Оператор понављања прави више копија листе и све их међусобно удружује.

Синтакса за употребу оператора понављања: `lista * n`

Овде је `lista` секвенца листе а `n` је број потребних копија листе.

001 Употреба оператора понављања

```
broj = [0] * 5
```

```
print(broj)    #даје [0, 0, 0, 0, 0]
```

Види се да оператор понављања прави 5 копија листе `[0]` и све копије удружује у једну листу.

Функција `len()`

Пајтон има уграђену функцију `len` која враћа дужину низа, попут листа.

002 Добијање дужине листе

```
>>> moja_lista = [10, 20, 30, 40]
```

```
>>> velicina = len(moja_lista)
```

```
>>> print(velicina)
```

```
4
```

Прва линија кода додељује листу промењивој `moja_lista`.

Друга линија кода позива функцију `len`, додајући промењиву `moja_lista` као аргумент.

Функција враћа вредност 4, што је број елемената у листи.

Функција `len` се може користити за спречавање `IndexError` изузетка при итерацији по листи са петљом.

Промењивост листа

Листе у Пајтону су промењиве (`mutable`), што значи да се њихови елементи могу мењати.

То значи да се форма `list[indeks]` може појавити на левој страни оператора доделе.

003 Модификација садржаја листе

```
>>> brojevi = [1, 2, 3]
```

```
>>> print(brojevi)
```

```
[1, 2, 3]
```

```
>>> brojevi[0] = 99
```

```
>>> print(brojevi)
```

```
[99, 2, 3]
```

Када се користи индексирани израз за доделу вредности елементу листе, мора се користити дозвољен индекс за постојећи елемент или ће се појавити `IndexError` изузетак.

004 Појава изузетка употребом недозвољеног индекса листе

```
>>> brojevi = [1, 2, 3]
```

```
>>> brojevi[5] = 100
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```


IndexError: list assignment index out of range

Пошто у првој линији дефинисано да листа има 3 елемента, са индексима од 0 до 2, покушај мењања непостојећег елемента у низу изазива прекид програма и подизање изузетка.

Да би се користио индексирани израз за попуњавање листе вредности, прво се мора направити листа.

Надовезивање листа

Надовезивање (concatenate) значи удруживање два ствари и за то се користи + оператор.

005 Надовезивање две нумеричке листе

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = [5, 6, 7, 8]
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4, 5, 6, 7, 8]
```

006 Надовезивање две стринг листе

```
>>> imena_devojci = ['Ana', 'Dragana', 'Gordana']
>>> imena_decaci = ['Milan', 'Dejan', 'Zoran']
>>> imena = imena_devojci + imena_decaci
>>> print(imena)
['Ana', 'Dragana', 'Gordana', 'Milan', 'Dejan', 'Zoran']
```

Итерација по листи са for петљом

Познате су технике приступа појединачном елементу унутар секвенце.

Многе од тих техника се могу применити и у листама.

Први начин је употреба петље а други начин је индексирање.

007 Итерирање по листи са for петљом

```
broj = [99, 100, 101, 102]
for n in broj:
    print(n)
```

што даје:

```
99
100
101
102
```

Индексирање

Други начин да се приђе појединачном елементу у листи је са индексом.

Сваки елемент у листи има индекс који одређује позицију елемента у листи.

Индекси започињу са 0, први елемент има индекс 0, други 1, итд.

Последњи елемент у листи од n елемената има индекс n - 1.

moja_lista = [10, 20, 30, 40] има 4 елемента у листи и њихови индекси су 0, 1, 2 и 3.

Елементи ове листе се могу одштампати на овај начин:

```
print(moja_lista[0], moja_lista[1], moja_lista[2], moja_lista[3])
```

008 Штампање елемената листе помоћу петље

```
moja_lista = [10, 20, 30, 40]
indeks = 0
while indeks < 4:
    print (moja_lista[indeks])
    indeks += 1
```

Такође се могу користити негативни индекси за идентификовање позиције елемента релативно у односу на крај листе.

Пајтон интерпретер додаје негативне индексе на дужину листе за одређивање позиције елемента.

Индекс -1 идентификује последњи елемент у листи, -2 идентификује следећи до њега, итд.

009 Штампање елемената листе негативним индексима

```
>>> moja_lista = [10, 20, 30, 40]
>>> print(moja_lista[-1], moja_lista[-2], moja_lista[-3], moja_lista[-4])
40 30 20 10
```

Ако се напише погрешан индекс за листу, нпр у претходном случају : `print(moja_lista[4])`, појавиће се објава о грешци а то се за овај случај назива изузетак о грешци индекса (`IndexError exception`).

010 Унос и приказ n елемената листе

Унети n елемената у листу целих бројева и приказати садржај листе

```
n = int(input("Unesi broj elemenata u listi: "))
A = [0 for x in range(n)]
print(A)
for x in range(n):
    A[x] = int(input("A(" + str(x) + ") = "))

print("Lista sada izgleda ovako: ", A)
```

011 Унос и приказ n елемената торке

Унети n елемената у торку целих бројева и приказати садржај торке

```
n = int(input("Unesi broj elemenata u listi: "))
A = [0 for x in range(n)]
print(A)
for x in range(n):
    A[x] = int(input("A(" + str(x) + ") = "))
```

```
M = tuple(A)
print("Toraka izgleda ovako: ", M)
```

012 Приказ елемената са непарним индексима

Из листе од n елемената, приказати само елементе са непарним индексима

```
n = int(input("Unesi broj elemenata u listi: "))
A = [0 for x in range(n)]
for x in range(n):
    A[x] = int(input("A(" + str(x) + ") = "))

print("Elementi liste sa neparnim indeksima su: ", end = '')
for x in range(n):
    if x % 2 != 0:
        print(A[x], " ", end = '')

print()
```

013 Приказ елемената у обрнутом редоследу

Исписати све елементе листе од n елемената у обрнутом редоследу

```
n = int(input("Unesi broj elemenata u listi: "))
A = [0 for x in range(n)]
for x in range(n):
    A[x] = int(input("A(" + str(x) + ") = "))

print("Elementi liste u pocetnom redosledu: ", end = '')
for x in range(n):
    print(A[x], " ", end = '')

print("\nElementi liste u obrnutom redosledu: ", end = '')
for x in range(n, 0, -1):
    print(A[x - 1], " ", end = '')
print()
```

Питања и задаци за самосталан рад

Питања

1. Шта је листа у Пајтону?
2. Како се креира листа?
3. Да ли су сви елементи листе истог типа података?
4. Како се други објекти конвертују у листу?
5. Како функционише оператор понављања?
6. Чему служи функција len()?
7. Шта је излаз функције len()?
8. Која су ограничења у креирању листа?
9. Који се оператор користи у надовезивању листа?

Задаци

001 Креирати листу са 5 вредности 0.0 а затим на прву и последњу позицију листе унети вредност

1.

002 Попунити листу са прва три цела броја почевши од датог целог броја. Конвертовати листу у торку.

003 Из листе [1, 0, 10, -2.0, -5, 4] на екрану исписати само вредности мање од 0.

004 Креирати листу са бројевима од 0 до 8, а затим креирати другу листу која ће на истој позицији имати исте бројеве као у првој листи ако су дељиви са 3.

Одсецање листа

Понекад је потребно изабрати више од једног елемента из секвенце.

У Пајтону, могу се писати изрази који бирају делове секвенце, познате као одсечци.

Одсечак (slice) је неки број ствари које су заједно одстрањене из секвенце.

Када се узме одсечак из листе, добија се неки број елемената из листе:

```
naziv_liste[start : kraj]
```

где је start индекс првог елемента у одсечку, а kraj је индекс који ограничава крај одсечка.

Синтаксом се добија листа која садржи копију елемената од индекса start до (не укључујући) елемент са индексом kraj.

014 Одсецање листе према синтакси

```
>>> dani = ['pon', 'uto', 'sre', 'cet', 'pet', 'sub', 'ned']
```

```
>>> sredina = dani[2:5]
```

```
>>> print(sredina)
```

```
['sre', 'cet', 'pet']
```

Види се да исказ dani[2:5] изводи одсецање из оригиналне листе од индекса 2 до индекса 4 чиме се креира нова листа која се зове sredina.

015 Одсецање изостављањем индекса start

```
>>> brojevi = [1, 2, 3, 4, 5]
```

```
>>> print(brojevi[:3])
```

```
[1, 2, 3]
```

```
>>> print(brojevi)
```

```
[1, 2, 3, 4, 5]
```

Ако се изостави индекс start у изразу за одсецање, Пајтон користи 0 као почетни индекс по дифолту.

Види се да одсецањем дела оригиналне листе се не мења оригинална листа јер одсецањем се прави копија одсечка оригиналне листе.

016 Одсецање изостављањем индекса kraj

```
>>> brojevi = [1, 2, 3, 4, 5]
```

```
>>> print(brojevi[3:])
```

```
[4, 5]
```

Види се да ако се изостави индекс краја одсечка, по дифолту се иде до краја листе.

017 Одсецање изостављањем оба индекса

```
>>> brojevi = [1, 2, 3, 4, 5]
>>> print(brojevi[:])
[1, 2, 3, 4, 5]
```

Ако се изоставе оба индекса у одсечку, добија се копија целе листе.

018 Одсецање коришћењем три индекса

```
>>> brojevi = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(brojevi)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(brojevi[1:8:2])
[2, 4, 6, 8]
```

Трећи број у примеру је корак одсецања и он упућује да се у одсечак укључује сваки други елемент почевши од задатог индекса.

019 Одсецање коришћењем негативних индекса

```
>>> brojevi = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(brojevi[-5:])
[6, 7, 8, 9, 10]
```

Могуће је користити и негативне индексе у одсецању при чему Пајтон додаје негативан индекс на дужину листе да би се добила позиција на коју упућује индекс.

Грешке са индексима

Погрешни индекси не доводе до појаве изузетака:

- Ако крај индекс указује на позицију изван краја листе, Пајтон ће употребити дужину листе уместо тога
- Ако start индекс специфицира позицију пре почетка листе, Пајтон ће аутоматски употребити 0
- Ако је start индекс већи од крај индекса, одсецање ће вратити празну листу

Задаци са одсецањем листа

020 Штампане елемената листе преко while петље

Коришћењем while петље и функције len() над листом [10, 20, 30, 40] одштампати на екрану само појединачне елементе листе

```
moja_lista = [10, 20, 30, 40]
indeks = 0
while indeks < len(moja_lista):
    print (moja_lista[indeks])
    indeks += 1
```

021 Замена са 0 вредности на парним индексима

Листи [10, 20, 30, 40, 50] заменити на парним индексима (индекс 0 није паран) постојеће вредности у 0

```
lista = [10, 20, 30, 40, 50]
```

```
for i in range(5):
    if ((i % 2 == 0) and (i != 0)):
        lista[i] = 0
```

```
print(lista)
```

022 Надовезати листе са појачаним оператором

Коришћењем појачаног оператора доделе надовезати две листе

```
lista1 = [10, 20, 30]
lista2 = ["ja", "ti", "mi"]
lista1 += lista2
print(lista1, lista2)
```

023 Поделити елементе листе

Поделити елементе листе [1, 2, 3, 4] у две листе са истим бројем елемената

```
start_lista = [1, 2, 3, 4]
duzina_novih_lista = len(start_lista) // 2
lista1 = start_lista[: duzina_novih_lista]
lista2 = start_lista[duzina_novih_lista :]
print(lista1, lista2)
```

Питања и задаци за самосталан рад

Задаци

005 Од листе [1, 2, 3, 4, 5] направити lista_leva = [1, 2] , lista_centar = [3] , lista_desna = [4, 5].

006 Ако листа има непаран број елемената, направити нову листу која садржи централни елемент прве листе.

007 После уноса елемената листе, ако листа има непаран број елемената, креирати листу са једним централним елементом, листу са елементима лево од централног елемента и листу са елементима десно од централног елемента. Ако листа има паран број елемената изоставити само листу са централним елементом.

008 Заменити места првом и последњем елементу у листи.

009 Заменити места било која два елемента у листи.

Претрага листе

Често претпостављамо да постоји у листи елемент са одређеном вредности али нисмо у то сигурни.

Пролазак кроз листу у потрази за тачно одређеним елементом, према његовој вредности, се назива претрага листе по елементима листе.

Реализација претраге листе по елементима листе се врши помоћу оператора **in**.

Оператор in

У Пајтону се оператор in користи за одређивање да ли се одређени елемент налази у листи.

Општи формат израза са in оператором за претрагу одређеног елемента у листи:

element in lista

Овде је element елемент за којим се трага а lista је назив листе у којој се трага за тим елементом.

Овакав формат враћа вредност True ако је елемент пронађен у листи а False ако није.

024 Употреба оператора in

```
sifre_proizvoda = ['VJ45', 'N32K', 'RP01', 'TTA1']
sifra_trazeni_proizvod = input("Uneti sifru trazenog proizvoda: ")
if sifra_trazeni_proizvod in sifre_proizvoda:
    print(sifra_trazeni_proizvod, "se nalazi u listi proizvoda.")
else:
    print(sifra_trazeni_proizvod, "se ne nalazi u listi proizvoda.")
```

Uneti sifru trazenog proizvoda: N32K

N32K se nalazi u listi proizvoda.

Uneti sifru trazenog proizvoda: ABVG

ABVG se ne nalazi u listi proizvoda.

Види се да у коду се добија од корисника шифра производа која се додељује променљивој sifra_trazeni_proizvod.

Помоћу условне структуре се врши претрага по листи у потрази за одређеним производом према његовој шифри.

Оператор not in

У Пајтону се оператор not in користи за одређивање да ли се одређени елемент не налази у листи.

Општи формат израза са not in оператором за претрагу одређеног елемента у листи:

element not in lista

Овде је element елемент за којим се трага а lista је назив листе у којој се трага за тим елементом.

Овакав формат враћа вредност True ако елемент није пронађен у листи а False ако је елемент пронађен у листи.

025 Употреба оператора not in

```
sifre_proizvoda = ['VJ45', 'N32K', 'RP01', 'TTA1']
sifra_trazeni_proizvod = input("Uneti sifru trazenog proizvoda: ")
if sifra_trazeni_proizvod not in sifre_proizvoda:
    print(sifra_trazeni_proizvod, "se ne nalazi u listi proizvoda.")
else:
    print(sifra_trazeni_proizvod, "se nalazi u listi proizvoda.")
```

Uneti sifru trazenog proizvoda: ABCD

ABCD se ne nalazi u listi proizvoda.

Uneti sifru trazenog proizvoda: TTA1
TTA1 se nalazi u listi proizvoda.

Задаци са претрагом листа

026 Проналажење жељеног елемента

Проверити да ли се у листи A = [20, 30, 44, 10, 0, -1] налази елемент који корисник захтева
A = [20, 30, 44, 10, 0, -1]

```
nasao = False
trazeni_element = int(input("Uneti trazeni element: "))
for i in A:
    if i == trazeni_element:
        nasao = True

if nasao:
    print("Trazeni element se nalazi u listi.")
else:
    print("Trazeni element se ne nalazi u listi.")
```

027 Проналажење броја жељеног елемента

Ако корисник наведе колико елемената треба да има листа, проверити да ли се у листи налази тражени елемент и колико таквих елемената има у листи

```
n = int(input("Uneti broj elemenata u listi: "))
A = [0 for i in range(n)]
print("Uneti elemente u listu: ")
for i in range(n):
    A[i] = int(input("A(" + str(i) + ") = "))

nasao = False
trazeni_element = int(input("Uneti trazeni element: "))
puta_pronadjjen = 0
for i in A:
    if i == trazeni_element:
        nasao = True
        puta_pronadjjen += 1

if nasao:
    print("Trazeni element se nalazi u listi.")
    print("Pronadjjen je", puta_pronadjjen, " puta." )
else:
    print("Trazeni element se ne nalazi u listi.")
```

028 Проналажење истих елемената у две листе

Дате су две листе: A = [0, 1, 2] и B = [-1, 0, 2]. Приказати индексе истих елемената који се налазе на истим позицијама у обе листе. Ако нема таквих елемената, исписати одговарајућу поруку


```

A = [0, 1, 2]
B = [-1, 0, 2]
nasao = False
for i in range(3):
    if A[i] == B[i]:
        nasao = True
        print("Pronadjena je ista vrednost na indeksu", i)

if not(nasao):
    print("Nije nista pronadjeno.")

```

029 Количина бројева дељивих са 3 у низу бројева

Проверити колико је бројева дељивих са 3 у опсегу целих бројева од 331 до 567 коришћењем `not in` оператора

```

A = list(range(331, 567))
B = [1, 2]
deljiv = 0
for i in A:
    if (i % 3) not in B:
        deljiv += 1

print(deljiv)

```

Питања и задаци за самосталан рад

Задаци

010 Корисник тражи да ли се одређени цео број налази у листи [5, 0, 0, 5, 1, 1, 2] и да ли се налази одређени број пута.

011 Корисник уноси две листе са познатим и истим бројем елемената. Колико има различитих вредности на истим позицијама у обе листе?

012 Корисник уноси опсег целих бројева као листу. Колико је у тој листи има бројева који нису дељиви ни са 6 ни са 10?

Методе листе

Листе имају на располагању велики број метода које омогућавају додавање елемената, одстрањивање елемената, промену редоследа елемената итд.

Метода `append`

Ова метода се користи за додавање вредности у листу.

Ствар која се придодаје као аргумент је придружена као последњи елемент у листи.

Синтакса употребе методе: `lista.append(vrednost)`

030 Унос новог елемента у листу

```
lista_brojeva = [6, 7, 8]
lista_brojeva.append(100)
print(lista_brojeva)           #[6, 7, 8, 100]
```

Метод index

Ако је потребно знати на којој се позицији налази вредност у листи, користи се овај метод.

Придодаје се аргумент у методу index а враћа се индекс првог елемента у листи који садржу вредност која се тражи.

Ако вредност није пронађена у листи, метод подиже изузетак ValueError.

Синтакса употребе методе: lista.index(vrednost)

031 Добијање позиције елемента у листи

```
lista_hrane = ["pica", "keks", "sladoled"]
hocu_da_jedem = 'keks'
print("Hrana",   hocu_da_jedem,   "se nalazi u listi na poziciji",
lista_hrane.index(hocu_da_jedem))
```

Метод insert

Овај метод омогућава уметање вредности у листу на жељену позицију.

Придодају се два аргумента методи insert: индекс где вредност треба да се смести у листи и вредност која се жели уметнути у листу.

Уметнута вредност се смешта на жељену позицију а елементи који су на позицијама после позиције нове вредности се померају за по једно место, без уклањања било којег елемента у листи.

Синтакса употребе методе: lista.insert(pozicija, vrednost)

Листа је назив листе у коју се умеће вредност, позиција је индекс на који се смешта вредност у листу.

032 Уметање вредности на одређену позицију

```
lista_brojeva = [3, 5, 6]
lista_brojeva.insert(1, 4)
print(lista_brojeva)           #[3, 4, 5, 6]
```

Метода remove

Ова метода одстрањује вредност из листе.

У методи се вредност појављује као аргумент, и први елемент који има ту вредност се отклања из листе.

Овиме се смањује величина листе за један.

Све остале вредности се померају за једну позицију према почетку листе.

Ако вредност не постоји у листи, појављује се изузетак ValueError.

Синтакса употребе методе: lista.remove(vrednost)

Вредност је елемент која већ постоји у листи и то као макар један елемент листе.

033 Одстрањивање елемента према вредности

```
lista_hrane = ["pica", "keks", "sladoled"]
lista_hrane.remove("keks")
print(lista_hrane)          #["pica", "sladoled"]
```

Исказ del

У неким ситуацијама је потребно отклонити елемент на одређеном индексу, без обзира која вредност се налази на том индексу.

Синтакса употребе методе: del lista[pozicija]

034 Одстрањивање елемента према индексу

```
lista_hrane = ["pica", "keks", "sladoled"]
del lista_hrane[1]
print(lista_hrane)          #["pica", "sladoled"]
```

Метода clear

Ова метода одстрањује све елементе из листе али не уклања саму листу.

Синтакса употребе методе: lista.clear()

035 Одстрањивање свих елемената из листе

```
lista_hrane = ["pica", "keks", "sladoled"]
lista_hrane.clear()
print(lista_hrane)          #[]
```

Метода sort

Ова метода преуређује елементе листе тако да се они појаве у растућем редоследу (од најниже ка највишој вредности).

036 Сортирати бројеве према вредности

```
moja_lista = [9, 1, 0, 2, 8, 6, 7, 4, 5, 3]
print('Originalni redosled:', moja_lista)
moja_lista.sort()
print('Uredjen redosled:', moja_lista)      #[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

037 Сортирати стрингове у листи

```
moja_lista = ['alfa', 'gama', 'delta', 'beta']
print('Originalni redosled:', moja_lista)
moja_lista.sort()
print('Uredjen redosled:', moja_lista)      #['alfa', 'beta', 'delta', 'gama']
```

Метода reverse

Ова метода обрће редослед елемената у постојећој листи.

038 Обрнути редослед елемената у листи

```
moja_lista = [1, 2, 3, 4]
print('Originalni redosled:', moja_lista)
```

```
moja_lista.reverse()
print('Obrnut redosled:', moja_lista)
```

Метода count

Ова метода приказује број појављивања тражене вредности у листи.

Синтакса употребе методе: lista.count(vrednost)

039 Пребројити појављивање елемента

```
moja_lista = [5, 2, 5, 5, 0, -1, 5, 5, 5]
broj_pojavljivanja_broja_5 = moja_lista.count(5)
print('Broj 5 se pojavljuje', broj_pojavljivanja_broja_5, 'puta u listi',
moja_lista, '.')
```

Метода pop

Ова метода уклања елемент из листе који се налази на одређеној позицији у листи.

Синтакса употребе методе: lista.pop(indeks_pozicije_u_listi)

040 Уклањање елемента према индексу

```
moja_lista = ["pica", "keks", "sladoled"]
moja_lista.pop(2)
print(moja_lista)           #["pica", "keks"]
```

Метода extend

Ова метода додаје на крај прве листе, све елементе који се налазе у некаквој другој секвенци (листа, сет, торка...).

Синтакса употребе методе: lista1.extend(sekvenca)

041 Придруживање елемената две листе

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista1.extend(lista2)
print(lista1)           #[1, 2, 3, 4, 5, 6]
```

Функције min и max

У Пајтону се две функције min и max користе за рад са секвенцама.

Функција min прихвата секвенце (попут листа) као аргументе и враћа елемент која има најмању вредност у секвенци.

Функција max прихвата секвенце (попут листа) као аргументе и враћа елемент која има највећу вредност у секвенци.

042 Открити највећи и најмањи број листе

```
moja_lista = [5, 4, 3, 2, 50, 40, 30]
print('Najnizu vrednost ima', min(moja_lista))
print('Najvecu vrednost ima', max(moja_lista))
```

Копирање листа

У Пајтону, додела једне промењиве другој промењивој чини да обе упућују на исти објекат у меморији.

043 Копирање листе додељивањем

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = list1
>>> print(list1)
[1, 2, 3, 4]
>>> print(list2)
[1, 2, 3, 4]
>>> list1[0] = 99
>>> print(list1)
[99, 2, 3, 4]
>>> print(list2)
[99, 2, 3, 4]
```

Сада је потребно направити копију листе тако да list1 и list2 упућују на две различите али по садржају исте листе.

Један начин да се то оствари је са петљом која копира сваки елемент листе.

044 Копирање листе помоћу петље

```
list1 = [1, 2, 3, 4]
list2 = []
for item in list1:
    list2.append(item)
```

Једноставнији и ефикаснији начин да се исто оствари је употреба оператора надовезивања.

045 Копирање листе надовезивањем

```
list1 = [1, 2, 3, 4]
list2 = [] + list1
```

Сада list1 и list2 указују на две одвојене али идентичне листе.

Метода copy

Ова метода враћа копију наведене листе.

Синтакса употребе методе: lista1.copy()

046 Копирање листе методом copy

```
lista1 = [1, 2, 3]
lista2 = lista1.copy()
print(lista2)           #[1, 2, 3]
```

Задаци методе листе

047 Употреба методе append

У празну листу корисник уноси имена све док не унесе неку посебну реч. Користити методу append

```
lista = []
ponovo = 'da'
while ponovo == 'da':
    ime = input('Unesi ime: ')
    lista.append(ime)
    print('Da li treba dodati jos jedno ime?')
    ponovo = input('da ili bilo sta drugo za ne: ')
    print()
print('Ovo su unesena imena:')
for ime in lista:
    print(ime)
```

048 Сабирање вредности у листи

Корисник не зна колико ће целобројних вредности да унесе у листу. Написати програм који сабира све унешене вредности у листу и приказује добијени збир

```
A = []
jos = "d"
i = 0
zbir = 0
while jos == "d":
    x = int(input("A(" + str(i) + ") = "))
    A.append(x)
    i += 1
    zbir += x
    jos = input("Da li da treba uneti jos brojeva (d/n)? ")

print("Lista", A, "ima zbir unetih brojeva u listu", zbir)
```

049 Удруживање елемената две листе

Дате су две листе: A = [0, 1, 2] и B = [-1, 0, 2]. У трећу листу поставити вредност која се налазе у обе листе

```
A = [0, 1, 2]
B = [-1, 0, 2]
C = []
for i in range(3):
    for j in range(3):
        if A[i] == B[j]:
            C.append(A[i])
            break

print(C)
```

Питања и задаци за самосталан рад

Задаци

013 Коришћењем метода листе написати програм који омогућава кориснику да унесе целе бројеве као елементе листе, избацује из листе све негативне бројеве а затим приказује листу.

014 Елементе торке (4, 0, 3) унети у листу [-5, 100, 200, -40, 77] на позиције које су исте као њихове вредности (елемент 0 на позицију 0, елемент 3 на позицију 3...). Елементи листе на тим позицијама се одстрањују из листе.

015 Из листе [5, 5, 6, 0, 1, 2, 5, 2, 1] избацити све поновљене вредности.

List Comprehension

Омогућава краћи код када је потребно креирати нову листу на основу вредности из већ постојеће листе.

Синтакса: `nova_lista = [izraz for broj in sekvenca if uslov == True]`

Као излаз се добија нова листа док секвенца остаје непромењена.

У синтакси, `uslov` је филтер којим се задржавају само вредности којима је услов испуњен.

Исказ `list comprehension` каже, за сваку вредност у секвенци која испуњава дати услов сместити у нову листу резултат израза као елемент те листе.

Израз може бити резултат математичке операције, логичке операције или податак било којег типа.

050 Употреба просте list comprehension

У нову листу сместити вредности из дате листе које су дељиве са 5

```
brojevi = [5, 6, 10, 15, 16]
nova_lista = [x for x in brojevi if x % 5 == 0]
print(nova_lista)
```

051 Употреба сложеније list comprehension

У нову листу сместити квадрате вредности из дате листе које су дељиве са 5

```
brojevi = [5, 6, 10, 15, 16]
nova_lista = [x ** 2 for x in brojevi if x % 5 == 0]
print(nova_lista)
```

052 Селектовање унетих вредности

Корисник уноси целе бројеве. У две различите листе сместити посебно позитивне и негативне бројеве. После завршетка уноса бројева приказати садржај листа и број унешених нула

```
pozitivni = []
negativni = []
broj_nula = 0
jos = "d"
while jos == "d":
    x = int(input("Uneti nov ceo broj: "))
    if x > 0:
```

```

        pozitivni.append(x)
    elif x < 0:
        negativni.append(x)
    else:
        broj_nula += 1
    jos = input("Da li da treba uneti jos brojeva (d/n)? ")

print("Uneti pozitivni brojevi su", pozitivni)
print("Uneti negativni brojevi su", negativni)
print("Uneto je", broj_nula, "nula")

```

053 Селектовање и измена вредности

Из листе [5, -3, 2, -10, 0, 2, -2] прво избацити вредности дељиве са 5 а преостале негативне вредности променити у 0

```

lista = [5, -3, 2, -10, 0, 2, -2]
nova_lista = []
for i in range(len(lista)):
    print(lista[i])
    if lista[i] < 0 and lista[i] % 5 != 0:
        nova_lista.append(0)
    elif lista[i] % 5 != 0 or lista[i] == 0:
        nova_lista.append(lista[i])
    else:
        pass

print(nova_lista)

```

054 Селектовање вредности према типу

У нову листу сместити само целе бројеве из листе ["ja", 0, -4, "-4", 3.05]

```

lista = ["ja", 0, -4, "-4", 3.05]
nova_lista = []
for i in range(len(lista)):
    if type(lista[i]) == type(int()):
        nova_lista.append(lista[i])

print(nova_lista)

```

055 Селектовање стрингова из листе

Корисник уноси стрингове у листу. Прихватити само оне стрингове који су дугачки између 5 и 10 знакова

```

lista = []
jos = "d"
while jos == "d":
    nov_string = input("Uneti nov string: ")
    if len(nov_string) >= 5 and len(nov_string) <= 10:

```



```

        lista.append(nov_string)
    jos = input("Da li da treba uneti jos stringova (d/n)? ")

print(lista)

```

056 Селектовање вредности преко методе index

Коришћењем методе index открити све индексе појављивања вредности 2 у листи [0, 2, 1, 2, 0, 2, 2].

```

lista = [0, 2, 1, 2, 1, 2, 2]
indeksi = []
i = 0
while True:
    if lista[i] == 2:
        indeksi.append(lista.index(2))
        lista[i] = 0
    i += 1
    if i == len(lista):
        break

print(indeksi)

```

Питања и задаци за самосталан рад

Задаци

- 016 Корисник пуни листу са 4 броја. Коришћењем list comprehension сместити позитивне целе бројеве (као и нуле) у једну а негативне целе бројеве у другу листу.
- 017 Ученици у одељењу се обележавају бројевима од 1 до 10. Ученици који иду на веронауку су 3, 5, 8. Ученици који су девојчице су 1, 3, 7, 8, 9, 10. Који ученици су дечаци и иду на грађанско?
- 018 Корисник уноси целе бројеве у листу. Креирати нову листу са вредностима „да“ или „не“ у зависности од тога да ли је унети цео број или квадрат или куб неког од једноцифрених целих бројева.
- 019 Највећу вредност у корисничкој листи целих бројева сместити на почетак листе, затим парне бројеве по растућем редоследу (као и 0), затим најмању вредност у листи а затим непарне вредности по опадајућем редоследу.
- 020 Дата је листа са презименом фудбалера, средњом оценом игре са две децимале и његов број голова у сезони ['fudbaler1', 4.34, 2, 'fudbaler2', 5.11, 3, 'fudbaler3', 3.76, 2]. Раздвојити податке по листама са презименима, средњим оценама и головима. Приказати податке о појединачном фудбалеру по редовима.

Дефинисање функције

Увод у функције

Сваки задатак, без обзира колико је комплексан, се може поделити на већи број мањих задатака, којима се приступа појединачно.

Сваки појединачни задатак се може решити креирањем посебне функције у коду.

Функције су групе исказа које постоје унутар програма са циљем решавања одређеног задатка.

Функције се затим могу извршавати у жељеном редоследу да би се решио комплексан задатак.

Овакав приступ се назива и „подели и освоји“.

Разлози за модулисање програма са функцијама:

- Једноставнији код за разумевање
- Функцијама се избегава дуплирање кода
- Олакшано тестирање кода
- Коришћење истих функција у различитим програмима
- Омогућава кодирање по тимовима програмера

Дефинисање функције

Да би се креирала функција, прво се мора дефинисати функција.

Општи формат дефиниције функције:

```
def ime_funkcije():    #header функције i njime se označava početak definicije funkcije
    iskaz              #početak bloka funkcija
    iskaz
    ...
```

Блок су искази који ће се реализовати сваки пут када се функција стартује.

Цео блок је увучен за један таб у односу на почетак хедера функције.

Позивање функције

Дефиниција функције специфицира шта функција ради, али не изазива стартовање те функције.

Да би се функција стартовала, функција мора да се позове (call): `ime_funkcije()`

Када се функција позове, интерпретер скаче на дефиницију функције која је позвана и реализује исказе у блоку.

Када се заврши реализација блока, интерпретер се враћа на линију кода где је реализован позив функције и наставља одатле даљи рад на коду.

Када се ово деси, каже се да се функција вратила (returns) у програм.

057 Дефинисање и позивање функције

```
def poruka():
    print("Ja sam veoma vazan covek.")
    print("Imam diplomu vaznosti.")
```

```
poruka()
```

Уобичајено је да се у програмима користи велики број функција, од којих је једна увек присутна: `main()`.

Ова се функција стартује одмах по почетку реализације програма и она позива друге функције из програма по потреби.

У Пајтону није неопходно да постоји `main()` функција али је сврсисходно да се користи приликом модуларног програмирања.

058 Дефинисање и позивање функције са `main()`

Пример 31: Дефинисати и позвати функцију која приказује поруку на екрану коришћењем `main()` функције

```
def poruka():
    print("Ja sam veoma vazan covek.")
    print("Imam diplomu vaznosti.")

def main():
    print("Imam poruku za tebe!")
    poruka()
    print('Dovidjenja.')
```

`main()`

У овим примерима је приказана дефиниција и позивање функције која не враћа никакву вредност у `main()` део кода.

Овакве функције се у програмирању називају `void` функције.

Локалне промењиве

Када се додели вредност промењивој унутар функције, тада се и креира локална промењива.

Локална промењива припада функцији у којој је креирана и само искази који су написани унутар те исте функције могу приступити тој промењивој.

Када неки исказ изван функције покуша приступити локалној промењивој унутар те функције, јавиће се грешка.

059 Ограничења локалних промењивих

```
def uzmi_ime():
    ime = input("Daj mi neko ime: ")
    print("Imam diplomu vaznosti.")

def main():
    print("Hitno mi treba neko ime!")
    uzmi_ime()
    print('Hvala', ime)
```

`main()`

```

Hitno mi treba neko ime!
Daj mi neko ime: mica
Imam diplomu vaznosti.
Traceback (most recent call last):
  File "C:\Users\nera\source\repos\PythonApplication1\PythonApplication1\PythonApplication1.py", line 10, in <module>
    main()
  File "C:\Users\nera\source\repos\PythonApplication1\PythonApplication1\PythonApplication1.py", line 8, in main
    print('Hvala', ime)
NameError: name 'ime' is not defined
Press any key to continue . . .

```

У примеру су две функције: `main()`, `uzmi_ime()`; промењива `ime` је локална промењива унутар функције `uzmi_ime()` јер је дефинисана у тој функцији.

При покушају приступа промењивој `ime` из функције `main()` јавља се `NameError` грешка јер за линију кода у којој се промењива `ime` тражи, ова промењива заправо не постоји пошто није дефинисана унутар исте функције у којој се и тражи.

Досег

Досег (scope) промењиве је део програма унутар којег се може користити промењива.

Промењива је видљива само оним исказима који су заједно са том промењивом у истом досегу.

Досег промењиве је функција унутар које је промењива и креирана.

Такође, локалној промењивој се не може прићи исказима који јесу у истом досегу као и промењива ако се линија кода са тим исказима налази пре него што је дефинисана локална промењива.

Пошто су локалне промењиве у једној функцији сакривене од осталих функција у истом програму, зато је могуће да свака од просталих функција има дефинисану промењиву са истим именом.

060 Приказ досега промењивих истог имена

```

def kosmaj():
    ptice = 500
    print('Na Kosmaju ima', ptice, 'vrsta ptica.')

def kopaonik():
    ptice = 800
    print('Na Kopaoniku ima', ptice, 'vrsta ptica.')

def main():
    kosmaj()           #poziv funkcije kosmaj()
    kopaonik()         #poziv funkcije kopaonik()

main()

```

Na Kosmaju ima 500 vrsta ptica.

Na Kopaoniku ima 800 vrsta ptica.

Иако постоје две различите промењиве са имеом `ptice` у овом програму, само је једна од њих у сваком тренутку видљива пошто су дефинисане у различитим досезима (функцијама).

Допремање аргумената функцији

Понекад је корисно при позиву функције допремити (passed) већу количину података тој функцији.

Подаци који се допремају функцији се називају аргументи.

Да би функција добила намењене аргументе, функција мора припремити одређени број промењивих за добијање података кроз аргументе.

Промењиве које се називају параметри, су посебне промењиве којима се достављају вредности преко аргумената при позиву функције.

061 Употреба аргумента у позиву

```
def duplo(a):  
    rezultat = a * 2  
    print('Dostavljena je vrednost', a, "i dobijena dupla vrednost",  
          rezultat)  
  
def main():  
    vrednost = 5  
    duplo(vrednost)
```

main()

У функцији main(), промењива vrednost је добила вредност 5, а затим се користи као аргумент у позиву функције duplo(vrednost).

Име функције је duplo, и њен циљ је да прихвати вредност из позива функције као аргумент и прикаже ту вредност као дуплирану.

У хедеру функције, у загради је наведен параметар а.

Овај параметар добија вредност која се налази у аргументу vrednost при позиву функције.

У овом примеру се могло извести позивање функције и овако: duplo(5)

Почетна идеја употребе функција јесте да се комплексан задатак подели на већи број мањих задатака, а сваки од мањих да се подели на још мање задатке и тако даље.

На крају се сваки од најситнијих задатака претвори у посебне функције.

Позивање функција се може извршити и из других функција.

062 Употреба main() функције

```
def uvod():  
    print('Ovaj program pretvara sadrzaj case u litre.')  
    print('Formula je: 1 casa = 0.2 litre tecnosti.')  
    print()  
  
def case_u_unce(case):  
    litre = case * 0.2  
    print('Konverzija u', litre, 'litara.')  
  
def main():  
    uvod()
```

```
potrebno_casa = int(input('Unesi broj casa: '))
case_u_unce(potrebno_casa)
```

```
main()
```

У коду се види да се из `main()` функције позивају све преостале функције, што је и смисао постојања `main()` функције.

Допремање већег броја аргумената

Често је корисно писати функције које могу примити више аргумената истовремено.

063 Допремање више аргумената

Пример 36: употребити више аргумената приликом позива функције

```
def prikaz_sume(broj1, broj2):
    rezultat = broj1 + broj2
    print(rezultat)
```

```
def main():
    print('Suma 12 i 45 je')
    prikaz_sume(12, 45)
```

```
main()
```

Види се да се користе две параметра, `broj1` и `broj2`, унутар хедера функције `prikaz_sume` раздвојени зарезом.

Често се сви наведени параметри у загради хедера функције називају листа параметара.

У функцији `main()` се позива функција `prikaz_sume()` са два аргумента 12 и 45.

Ови аргументи се достављају према позицији у одговарајуће параметре у функцији (12 у `broj1` а 45 у `broj2`).

Уместо конкретних вредности, аргументи могу бити промењиве које морају упућивати на конкретне вредности приликом позивања функције.

Промена у параметрима

Када се аргумент достави функцији, параметар промењива функције ће указивати (reference) на вредност аргумента.

Ипак, било каква промена која се изведе над параметар промењивој неће утицати на аргумент.

064 Измена вредности параметара

```
def main():
    x = 99
    print('Vrednost je', x)
    promeni_me(x)
    print('U main() funkciji vrednost je i dalje', x)
```

```
def promeni_me(y):
```

```
print('Ovde se menja vrednost.')
y = 0
print('Sada je vrednost', y)
```

```
main()
```

У коду, промењива x је достављена као аргумент функцији promeni_me, а то значи да ће параметар y такође указивати на вредност 99.

Промењива y мења вредност у 0 и тиме се не мења вредност промењиве x у функцији main().

Када се нека вредност достави функцији путем аргумента а функција не може променити ту вредност аргумента се назива достављање преко вредности (pass by value).

Овај начин комуникације између функција је једносмеран, тј само од прве функције ка позваној функцији.

Аргументи службене речи

Осим преноса вредности од аргумента према параметру по позицији у позиву функцији, Пајтон омогућава и писање аргумента са службеном речи да би се специфицирало коју вредност аргумент треба да преда параметру:

```
ime_parametra = vrednost
```

Аргумент који је написан у складу са овом синтаксом се назива аргумент службене речи (keyword argument).

065 Употреба аргумента службене речи

```
def main():
    show_interest(korak=0.01, vremenski_period=10, osnovica=10000.0)

def show_interest(osnovica, korak, vremenski_period):
    ulog = osnovica * korak * vremenski_period
    print('Ulog ce biti $', format(ulog, ',.2f'), sep='')

main()
Ulog ce biti $1,000.00
```

Види се да редослед аргумената службених речи не одговара редоследу параметара у хедеру функције.

То значи да се аргументи службене речи достављају не према позицији већ су битне само пренете вредности.

Могуће је промешати позиционе аргументе и аргументе службене речи у позиву функције, али позициони аргументи се морају навести први а аргументи службене речи после њих.

```
show_interest(10000.0, korak=0.01, vremenski_period=10)
```

Овде је први аргумент (10000.0) достављен према својој позицији а друга два су аргументи службене речи.

Задаци са void функцијама

066 Цртање 10 знакова у једном реду

```
def crtanje_linije(x, y):  
    for i in range(x):  
        print(y, end = '')  
  
def main():  
    crtanje_linije(10, 'o')  
    print()  
  
main()
```

067 Цртање n знакова у једном реду

```
def crtanje_linije(x, y):  
    for i in range(x):  
        print(y, end = '')  
  
def main():  
    broj = int(input("Unesti duzinu linije: "))  
    crtanje_linije(broj, 'o')  
    print()  
  
main()
```

068 Цртање знакова у више редова

Написати функцију која црта знакове 'o' у 5 редова, по један у сваком реду.

```
def crtanje_linije(x, y):  
    for i in range(x):  
        print(y)  
  
def main():  
    crtanje_linije(5, 'o')  
    print()  
  
main()
```

069 Цртање правоугаоника 10 x 5

Написати функцију која црта попуњени правоугоник димензија 10x5 (дужина x висина).

```
def crtanje_linije(x, y, znak):  
    for i in range(x):  
        for j in range(y):  
            if j != y - 1:  
                print(znak, end = '')  
            else:  
                print(znak)
```



```
def main():
    crtanje_linije(5, 10, 'o')
    print()
```

main()

Питања и задаци за самосталан рад

Задаци

021 Написати функцију која црта ивицу правоугаоника димензија 10x5 са знацима 'o'.

022 Написати програм који помоћу функције црта ивицу квадрата димензије n.

023 Написати програм који помоћу функције сабира све бројеве у опсегу бројева од 250 до 420.

024 Написати функцију која сабира све целе бројеве у датом опсегу

070 Цртање квадрата странице n

Написати функцију која уноси жељени знак за попуњавање квадрата и која црта квадрат дате дужине.

```
def kvadrat():
    znak = input("Znak za crtanje kvadrata je: ")
    a = int(input("Uneti broj znakova u stranici kvadrata: "))
    for i in range(a):
        for j in range(a):
            if j == a - 1:
                print(znak)
            else:
                print(znak, end = "")
```

```
def main():
    kvadrat()
    print()
```

main()

Znak za crtanje kvadrata je: x

Uneti broj znakova u stranici kvadrata: 4

XXXX

XXXX

XXXX

XXXX

071 Употреба претраге као void функције

Написати функцију која испитује корисника да ли се његово име налази у датој листи имена и даје одговарајућу поруку у оба случаја (налази се у листи или не налази се у листи).

```
def trazi_po_listi():
    nasao = False
    a = ["Jovan", "Ana", "Dragan", "Pera", "Zorana", "Bojana", "Nikola"]
    ime = input("Uneti ime: ")
    for x in a:
        if ime == x:
            nasao = True
    if nasao:
        print("Ime", ime, "se nalazi u listi.")
    else:
        print("Ime", ime, "se ne nalazi u listi.")

def main():
    trazi_po_listi()
    print()
```

```
main()
Uneti ime: Dragan
Ime Dragan se nalazi u listi.
Uneti ime: Anica
Ime Anica se ne nalazi u listi.
```

072 Рачунање квадрата броја преко void функције

Написати програм који добија број од корисника и функцију која рачуна квадрат тог броја.

```
def kvadrat(x):
    stepen = x * x
    print("Kvadrat broja " + str(x) + " je " + str(stepen) + ".")

def main():
    broj = float(input("Uneti broj: "))
    kvadrat(broj)
    print()
```

```
main()
Uneti broj: 3.6
Kvadrat broja 3.6 je 12.96.
```

073 Цртање дијагонале знаковима

Написати функцију која исцртава дијагоналу са 5 знакова 'o' (дијагонала почиње горе лево а завршава се доле десно).

```
def dijagonala(x, y, z):
```

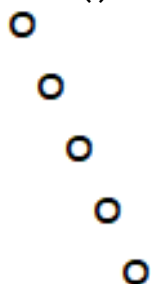
```

for i in range(x):
    for j in range(x):
        if i == j:
            print(y)
            break
        elif i != j:
            print(z, end = "")

def main():
    dijagonala(5, "o", " ")
    print()

```

main()



Питања и задаци за самосталан рад

Задаци

025 Написати функцију која уноси жељени знак за попуњавање правоугаоника, жељене димензије правоугаоника и која црта задати правоугаоник.

026 Написати програм који добија два броја од корисника и функцију која даје разлику квадрата тих бројева.

027 Написати програм који добија број од корисника и функцију која приказује реципрочну вредност унетог броја.

028 Написати програм који добија број од корисника и функцију која исцртава дијагоналу са задатим бројем знакова 'o' (дијагонала почиње горе десно а завршава се доле лево)

Глобалне промењиве

Када се креира промењива преко оператора доделе унутар неке функције, промењива постаје локална промењива за ту функцију па њој се може прићи само преко исказа који су такође унутар исте функције.

Када се промењива креира изван свих функција у програму, промењива је глобална.

Глобална промењива је доступна свим исказима у програму, без обзира да ли су написани унутар или изван било каквих функција у програму.

074 Поставка глобалне промењиве у коду

```
broj = 10
```

```
def prikaz_vrednosti():  
    print(broj)
```

```
prikaz_vrednosti()
```

Променљива broj је глобална променљива и њена вредност је 10 и дефинисана је изван било какве функције.

Функција prikaz_vrednosti() исписује вредност променљиве broj.

Да би се вредност глобалне променљиве могла мењати унутар функције, онда се променљива мора декларисати као глобална променљива и унутар функције.

075 Утицај функција на промену вредности глобалне променљиве

```
broj = 0
```

```
def prikazi_broj():  
    print('Uneti broj je', broj)
```

```
def main():  
    global broj  
    broj = int(input('Unesi broj: '))  
    prikazi_broj()
```

```
main()  
Unesi broj: 10  
Uneti broj je 10
```

Унутар функције main() се користи global службена реч за декларисање променљиве broj и помоћу исказа се додељује вредност од стране корисника.

Лоше особине глобалних променљивих

Потребно је ограничити употребу глобалних променљивих или уопште не користити глобалне променљиве у кодовима:

- Глобалне променљиве чине дебаговање (тражење грешака у коду) тешким; било који исказ у целом програму може променити вредност глобалне променљиве па је тешко открити који исказ је доделио погрешну вредност
- Функције које користе глобалне променљиве су обично зависне од њихових вредности што је потенцијално опасно по њихово функционисање
- Глобалне променљиве чине програм тешким за разумевање јер се утицај на њихово модификовање може наћи било где у програму

Из ових разлога, најбоље је креирати локалне променљиве и доставити их као аргументе функцијама.

Глобалне константе

Назив глобалне константе се односи на вредности које се не могу променити у програму.

Из тог разлога глобалне константе јесу глобалне вредности али пошто се не мењају у програму нема проблема као са модификацијама глобалних промењивих.

Пајтон не допушта креирање правих глобалних константи, већ се оне симулирају преко глобалних промењивих.

076 Употреба глобалних константи

```
GLAVNI_BOG = "Zeus"
```

```
def mali_bogovi():  
    print(GLAVNI_BOG + " i sitni bogovi.")
```

```
def srednji_bogovi():  
    print(GLAVNI_BOG + " i mediokritetni bogovi.")
```

```
def main():  
    print("Potrebno je preispitati znanje o starogrckim bogovima.")  
    mali_bogovi()  
    srednji_bogovi()
```

```
main()
```

У програму се користи глобална константе GLAVNI_BOG.

Обично се пишу имена константи са великим словима чиме се визуелно упозорава на непромењивост вредности.

Разлози за коришћење глобалних константи:

- Програм се лакше чита
- Ако је потребно променити вредност глобалне константе, потребно је то урадити само у исказу где је константа и декларисана

Функције које враћају вредност

До сада су дефинисане void функције које се позивају а затим и извршавају.

По завршетку извршавања void функције, контрола над програмом се враћа на линију кода који је одмах после линије позива функције.

Функција која враћа вредност је посебан тип функције која се позива, извршава а затим и враћа израчунату вредност у исту линију кода из које је и позвана.

Враћена вредност се користи као било која друга вредност: може се доделити некој промењивој, приказати на екрану, користити у математичким изразима...

Функције стандардне библиотеке и исказ import

Пајтон има стандардну библиотеку функција у којој се налазе предефинисане функције.

Ове функције се називају функције библиотеке и помажу кориснику у решавању различитих проблема.

До сада су коришћене неке од тих функција библиотеке: print, input, range.

Неке од Пајтонових функција библиотеке су уграђене у Пајтон интерпретер.

Ако је потребно користити ове уграђене функције у програму, потребно је само позвати ту функцију.

Али, велики број функција библиотеке су смештене у фајловима који се називају модули.

Модули се копирају на рачунар приликом инсталације Пајтона; нпр функције за извођење математичких операција се налазе у једном модулу, функције за рад са фајловима су смештене у другом модулу...

Да би се позвала функција смештена у модулу, потребно је написати import исказ на почетку програма.

Исказ import каже интерпретеру назив модула који садржи функције које се желе користити у програму.

Нпр, за коришћење низа математичких функција потребно је импортовати математички модул: import math.

На овај начин се учита садржај math модула у меморију и све функције из тог модула су доступне програму.

Генерисање случајних бројева

Случајни бројеви се користе у играма, симулацијама, статистичким програмима и апликацијама заштите података.

Пајтон омогућава неколико функција библиотеке за рад са случајним бројевима и оне су смештене у модулу random у стандардној библиотеци.

Да би се користиле ове функције прво се мора импортовати модул: import random.

Функција randint се налази у модулу random и да би се она могла користити мора се употребити дот нотација у програму: random.randint.

Са леве стране тачке (дот) је назив модула а са десне стране тачке је назив функције.

```
broj = random.randint(1, 100)
```

У загради се налазе два аргумента и они говоре да се тражи случајан цео број у опсегу између 1 и 100, укључујући и 1 и 100.

Када се изврши написана линија кода, позив функције ће генерисати случајан цео број, вратиће га као резултат функције на место позива што значи да ће промењива broj добити ту вредност.

077 Генерисање целог случајног броја у опсегу вредности

```
import random
def main():
    slucajan_broj = random.randint(1, 10)
    print('Broj je', slucajan_broj)
```

```
main()
```

У овом задатку генерише се један случајан цео број у опсегу од 1 до 9.

078 Генерисање више случајних бројева у опсегу

```
import random
def main():
```

```

for count in range(5):
    number = random.randint(1, 100)
    print(number)

```

main()

У овом задатку се генерише пет случајних целих бројева у опсегу између 1 и 99. Могуће је заменити код у телу for петље са: print(random.randint(1, 100))

Примери рада са random модулом

079 Симулација бацања две стандардне коцкице

```

import random
MIN = 1 #konstanta za minimalan moguci broj na kocki
MAX = 6 #konstanta za maksimalan moguci broj na kocki

def main():
    ponovo = 'd'
    #simulacija bacanja kockice
    while ponovo == 'd' or ponovo == 'D':
        print('Bacam kockice ...')
        print('Dobijeni su brojevi:')
        print(random.randint(MIN, MAX))
        print(random.randint(MIN, MAX))
        #ponavljanje bacanja?
        ponovo = input('Da ponovo bacam kockice? (d = da): ')

```

main()

080 Симулација бацања новчића

```

import random
GLAVA = 1
PISMO = 2
BACANJE = 10

def main():
    for novcic in range(BACANJE):
        if random.randint(GLAVA, PISMO) == GLAVA:
            print('glava')
        else:
            print('pismo')

```

main()

Функције randrange, random, uniform

Модул random стандардне библиотеке садржи различите функције за рад са случајним бројевима.

Функција randrange добија аргументе на исти начин као и range функција, само је разлика у томе што она не враћа листу вредности.

Функција randrange враћа случајно изабране вредности из секвенце вредности.

```
broj = random.randrange(10) #promenjiva broj dobija slučajnu vrednost između 0 i 9
```

```
broj = random.randrange(5, 10) #promenjiva broj dobija slučajnu vrednost između 5 i 9
```

```
broj = random.randrange(0, 101, 10) #promenjiva broj dobija slučajnu vrednost između 0 i 100 po koraku 10
```

Функција random враћа случајно генерисан реалан број у опсегу од 0.0 до 1.0 али не укључујући и 1.0.

```
broj = random.random()
```

Функција uniform враћа случајно генерисани реалан број али у назначеном опсегу вредности.

```
broj = random.uniform(1.0, 10.0)
```

Просејавање при генерисању случајних бројева

Бројеви генерисани са функцијама у random модулу нису заиста случајни бројеви већ псеудослучајни бројеви.

Они се рачунају путем формуле која се иницијализује са вредности која се назива вредност просејавања (seed value).

Вредност сидовања се користи у рачунању којим се враћа следећи случајан број у серији.

Када се random модул импортује, он добија системско време (system time) из компјутерског унутрашњег часовника и користи ово време као основу за вредност сидовања.

Системско време је целобројна вредност којом се представља тренутно време и датум, до стотог дела секунде.

Пошто се системско време мења сваког стотог дела секунде, може се рећи да се увек генерише другачија секвенца случајних бројева импортовањем random модула.

Када је потребно генерисати увек исту секвенцу случајних бројева може се позвати random.seed функција за добијање вредности сидовања: random.seed(10)

У овом примеру, вредност 10 је специфицирана као вредност сидовања.

Када програм позове random.seed функцију допремајући увек исти аргумент, онда ће се увек произвести иста секвенца псеудослучајних бројева.

081 Доказ добијања исте секвенце псеудослучајних бројева

```
import random
random.seed(10)
print(random.randint(1, 100))
print(random.randint(1, 100))
print(random.randint(1, 100))
print(random.randint(1, 100))
```

Сваки пут када се стартује програм из примера 46, увек се добија иста секвенца бројева између 0 и 99.

Писање сопствених функција које враћају вредност

Сопствене функције које враћају вредност се пишу на исти начин као што се пишу void функције, са само једним изузетком: на крају функције мора бити **return** исказ.

Општи формат дефиниције функције која враћа вредност:

```
def ime_funkcije():
    iskaz
    iskaz
    ...
    return iskaz
```

Исказ после команде **return** ће бити послат на део кода који је позвао функцију и то може бити било која вредност, променљива или израз који даје неку вредност.

082 Дефиниција сопствене функције која враћа вредност

```
def suma(broj1, broj2):           #ime funkcije je suma, broj1 i broj2 su parametri funkcije
    rezultat = broj1 + broj2
    return rezultat              #funkcija vraća vrednost na koju upućuje promenjiva rezultat
```

083 Код са сопственом функцијом

```
def suma(godine_tvoje, godine_prijatelj):
    rezultat = godine_tvoje + godine_prijatelj
    return rezultat

def main():
    godine_tvoje = int(input('Unesi broj godina: '))
    godine_prijatelj = int(input("Unesi godine tvog najboljeg prijatelja:
"))
    zajedno = suma(godine_tvoje, godine_prijatelj)
    print('Zajedno imate', zajedno, 'godina starosti.')

main()
```

Пошто команда return може да врати израз, може се написати и: **return** godine_tvoje + godine_prijatelj

Употреба функција које враћају вредност

Функције које враћају вредност поједностављују код, избегавају дуплицирање кода, повећавају могућности тестирања кода, повећавају брзину развоја кода и побољшавају тимско програмирање.

Један од начина да се користе је враћање вредности коју је корисник унео:

```
def upisi_cenu():
    cena = float(input("Unesi cenu stvari: "))
    return cena
```

У главном делу кода се позива функција која враћа унету цену од стране корисника: uneta_cena = upisi_cenu()

Често се функције које враћају вредност користе за поједностављење комплексних математичких израчунавања пошто омогућавају разбијање комплексних формула на мање и лакше разумљиве делове.

084 Рачунање попушта на цену робе

POPUST_U_PROCENTIMA = 0.20

```
def main():
    stvarna_cena = unesi_stvarnu_cenu()
    prodajna_cena = stvarna_cena - izracunaj_popust(stvarna_cena)
    print('Prodajna cena je', prodajna_cena, 'dinara.')

def unesi_stvarnu_cenu():
    cena = float(input("Unesi stvarnu cenu proizvoda: "))
    return cena

def izracunaj_popust(cena):
    return cena * POPUST_U_PROCENTIMA

main()
```

У програму се користе три функције од које су две функције које враћају вредност.

Функција `unesi_stvarnu_cenu()` се позива без аргумената, омогућава кориснику да унесе стварну цену производа која се одмах конвертује у реалан број и тај број се и враћа на место позива.

Функција `izracunaj_popust(stvarna_cena)` се позива са једним аргументом, рачуна укупан попуст на основу глобалне константе `POPUST_U_PROCENTIMA` и ту вредност враћа на место позива функције.

Функција `main()` не враћа вредност већ служи да на једном месту обухвати све позиве функција и да прикаже коначан резултат на екрану.

Позив функције `main()` је једина линија кода изван свих функција у коду.

Сопствене функције које враћају стринг и булове вредности

085 Функција која враћа стринг

```
def uzmi_ime():
    ime = input('Unesi ime: ')
    return ime
```

Булове функције враћају `True` или `False`.

Обично се користе за тестирање услова и враћањем `True` или `False` указују да ли је услов испуњен.

086 Испитивање парности

```
def jeste_paran(broj):
    if (broj % 2) == 0:
        status = True
```

```

else:
    status = False
return status

broj = int(input('Unesi broj: '))
if jeste_paran(broj):
    print('Broj je paran.')
else:
    print('Broj je neparan.')

```

Овако креирана функција враћа булову вредност којом се осликава резултат испитивања парности унетог броја.

Булове функције се могу користити и за валидацију улазног кода.

087 Унос одређеног броја

```

def broj_nije_dobar(x):
    if x != 100 and x != 200 and x != 300:
        status = True
    else:
        status = False
    return status

broj = int(input('Unesi validan broj: '))
while broj_nije_dobar(broj):
    print('Validan broj je 100, 200 ili 300.')
    broj = int(input('Unesi validan broj: '))

print('Broj', broj, "je validan.")

```

Функције које враћају више вредности

Пајтонове функције нису ограничене да враћају само једну вредност: return izraz1, izraz2, ...

088 Дефинисање функције која враћа више вредности

```

def uzmi_ime_i_prezime():
    prvo = input('Unesi ime: ')
    drugo = input('Unesi prezime: ')
    return prvo, drugo

ime, prezime = uzmi_ime_i_prezime()
print("Uneto ime i prezime:", ime, prezime)

```

Вредности које су излистане у return исказу се достављају, по редоследу у којем су и исписане, промењивима на левој страни оператора доделе.

После извршења овог исказа, вредност промењиве прво ће се доделити промењивој име, а вредност друго ће се доделити промењивој презиме.

Ако број промењивих са леве стране оператора доделе није идентичан са бројем промењивих којима се враћа вредност из функције, појавиће се грешка.

Модул math

Модул math у Пајтоновој стандардној библиотеци садржи неколико функција које су корисне за извођење математичких операција.

функција	опис	функција	опис
acos(x)	враћа arccos од x, у радијанима	asin(x)	враћа arcsin од x, у радијанима
atan(x)	враћа arctg од x, у радијанима	cos(x)	враћа cos од x, у радијанима
sin(x)	враћа sin од x, у радијанима	tan(x)	враћа tg од x, у радијанима
degrees(x)	ако је x угао у радијанима, враћа угао конвертован у степене	radians(x)	ако је x угао у степенима, враћа угао конвертован у радијане
ceil(x)	враћа најмањи цео број који је већи или једнак са x	floor(x)	враћа највећи цео број који је мањи или једнак са x
log(x)	враћа природни логаритам од x	log10(x)	враћа логаритам основе 10 од x
exp(x)	враћа експонент од x	sqrt(x)	враћа квадратни корен од x
hypot(x, y)	враћа дужину хипотенузе између тачака (0, 0) и (x, y)		

Обично ове функције узимају једну или више вредности као аргументе, изводе математичку операцију коришћењем аргумената и враћају резултат (све функције враћају float, осим ceil и floor које враћају целобројне вредности).

089 Рачунање хипотенузе

```
import math
def main():
    #a i b su katete pravouglog trougla a c je hipotenuza
    a = float(input('Unesi duzinu stranice a: '))
    b = float(input('Unesi duzinu stranice b: '))
    c = math.hypot(a, b)
    print('Duzina hipotenuze je', c)
main()
```

Модул math такође дефинише две промењиве, pi и e, којима су додељене математичке вредности бројева пи и експонента e: `povrsina_kruga = math.pi * poluprecnik**2`

Смештање функција у модуле

Како програми постају већи и комплекснији, потребно је организовати код пошто и он постаје тежи за праћење.

Познато је да је боље поделити код у функцији због лакшег одржавања и правилније логике кода. Како се све више функција креира у програму, треба се одлучити на смештање функција у модуле. Модул је једноставно фајл који садржи Пајтон код.

Када се програм подели на модуле, сваки модул треба да садржи функције које решавају на неки начин повезане задатке.

Подела функција на модуле се назива модулизација што чини програм лакшим за одржавање и тестирање.

Ако и у другим програмима је потребно користити исте функције, довољно је импортовати модул у програмима.

090 Рачунање површине и обима

Модул `krug` садржи две функције: `povrsina` и `obim`.

```
#modul krug.py
import math
def povrsina(poluprecnik):
    return math.pi * (poluprecnik ** 2)

def obim(poluprecnik):
    return 2 * math.pi * poluprecnik
```

Модул `pravougaonik` садржи две функције: `povrsina` и `obim`.

```
#modul pravougaonik.py
import math
def povrsina(duzina, sirina):
    return duzina * sirina

def obim(duzina, sirina):
    return 2 * (duzina + sirina)
```

Оба фајла садрже дефиниције функција, али не садрже код који позива ове функције (позиви су у програмима који импортују ове модуле).

Име модула мора се завршавати са `.py` (иначе се неће моћи импортовати у друге програме).

Име модула не сме бити исто као нека Пајтонова службена реч.

Када се импортује модул (`import pravougaonik`), Пајтон интерпретер чита овај исказ и тражи фајл `pravougaonik.py` у истом фолдеру као што програм тражи модуло да би га импортовао.

Ако пронађе фајл, учита га у меморију а ако га не пронађе, јавља се грешка.

Када је модуло импортован могу се позивати његове функције.

```
import krug
import pravougaonik
IZBOR_POVRSINE_KRUGA = 1
IZBOR_OBIMA_KRUGA = 2
IZBOR_POVRSINE_PRAVOUGAONIKA = 3
IZBOR_OBIMA_PRAVOUGAONIKA = 4
IZBOR_IZLASKA_IZ_PROGRAMA = 5
def main():
    izbor = 0
    while izbor != IZBOR_IZLASKA_IZ_PROGRAMA:
        prikaz_menija()
        izbor = int(input('Unesi izbor: '))
        if izbor == IZBOR_POVRSINE_KRUGA:
```

```

        poluprecnik = float(input("Unesi poluprecnik kruga: "))
        print('Povrsina kruga je', krug.povrsina(poluprecnik))
    elif izbor == IZBOR_OBIMA_KRUGA:
        poluprecnik = float(input("Unesi poluprecnik kruga: "))
        print('Obim kruga je', krug.obim(poluprecnik))
    elif izbor == IZBOR_POVRSINE_PRAVOUGAONIKA:
        duzina = float(input("Unesi duzinu pravougaonika: "))
        sirina = float(input("Unesi sirinu pravougaonika: "))
        print('Povrsina pravougaonika je',
pravougaonik.povrsina(duzina, sirina))
    elif izbor == IZBOR_OBIMA_PRAVOUGAONIKA:
        duzina = float(input("Unesi duzinu pravougaonika: "))
        sirina = float(input("Unesi sirinu pravougaonika: "))
        print('Obim pravougaonika je', pravougaonik.obim(duzina,
sirina))
    elif izbor == IZBOR_IZLASKA_IZ_PROGRAMA:
        print('Izlazak iz programa...')
    else:
        print('Greska: nedozvoljen izbor.')

def prikaz_menija():
    print(' OPCIJE')
    print('1) Povrsina kruga')
    print('2) Obim kruga')
    print('3) Povrsina pravougaonika')
    print('4) Obim pravougaonika')
    print('5) Izlaz iz programa')

main()

```

Достављање листе као аргумента функције

Коришћење листа као аргументе функција омогућава да се многе операције које се раде над листама користе као засебне функције.

При позивању ових функција, листе се достављају као аргументи.

091 Функција која рачуна суму вредности из листе бројева

```

def izracunaj_sumu(lista_vrednosti):
    suma = 0
    for x in lista_vrednosti:
        suma += x
    return suma

def main():
    brojevi = [2, 4, 6, 8, 10]
    print('Suma je', izracunaj_sumu(brojevi))

```

```
main()
```

Враћање листе из функције

Функција може да врати упућивање (reference) на листу.

Овиме се омогућава писање функција које креирају листе и додају елементе у листе а затим враћају упућивање на листе тако да остали делови програма могу радити са њима.

092 Петља while са листом непознате дужине у функцији

```
def uzmi_vrednosti():
    vrednosti = []
    ponovo = 'd'
    while ponovo == 'd':
        broj = int(input('Unesi broj: '))
        vrednosti.append(broj)
        print('Hoces li da uneses jos jedan broj?')
        ponovo = input('d = da, bilo sta drugo = ne: ')
        print()
    return vrednosti

def main():
    brojevi = uzmi_vrednosti()
    print('Brojevi u listi su:')
    print(brojevi)
```

```
main()
```

093 Рачунање средње вредности са тестова

```
def uzmi_rezultate():
    rezultati_sa_testova = []
    ponovo = 'd'
    while ponovo == 'd':
        vrednost = float(input('Unesi rezultat sa testa: '))
        rezultati_sa_testova.append(vrednost)
        print('Hoces li da uneses jos jedan rezultat?')
        ponovo = input('d = da, bilo sta drugo = ne: ')
        print()
    return rezultati_sa_testova

def dobijanje_sume(lista_vrednosti):
    suma = 0.0
    for x in lista_vrednosti:
        suma += x
    return suma
```

```
def main():
    rezultati = uzmi_rezultate()
    zbir = dobijanje_sume(rezultati)
    najlosiji_rezultat = min(rezultati)
    zbir -= najlosiji_rezultat
    srednja_vrednost = zbir / (len(rezultati) - 1)
    print('Srednja vrednost, sa izbacenim najslabijim rezultatom je:',
srednja_vrednost)
```

main()

У програму се користе две корисничке дефинисане функције: `dobijanje_sume` (добива резултате тестова од корисника и враћа упућивање на листу која садржи те резултате) и `uzmi_rezultate` (доставља листу `rezultati` као аргумент и враћа суму вредности у листи).

Уграђена функција `min` узима листу `rezultati` а враћа најнижу вредност у листи.

Задаци са return функцијама

094 Испитивање парности са Буловим вредностима

Написати функцију која испитује да ли је унети број паран а која враћа Булову вредност.

```
def jeste_paran(broj):
    if (broj % 2) == 0:
        status = True
    else:
        status = False
    return status

broj = int(input('Unesi broj: '))
if jeste_paran(broj):
    print('Broj je paran.')
else:
    print('Broj je neparan.')
```

095 Рачунање степена унетог броја

Написати програм са функцијама за израчунавање степена броја.

```
def unos_osnove():
    x = float(input("Uneti osnovu: "))
    return x

def unos_stepena():
    x = float(input("Uneati stepen: "))
    return x

def stepenovanje(x, y):
    z = x ** y
    return z
```



```
def prikaz_rezultata(x, y, z):
    print("Za osnovu", str(x), "na stepen", str(y), "dobija se", str(z))

def main():
    osnova = unos_osnove()
    stepen = unos_stepena()
    rezultat = stepenovanje(osnova, stepen)
    prikaz_rezultata(osnova, stepen, rezultat)

main()
```

096 Ревизија рачунања степена унетог броја

Написати ревизију програма за рачунање степена броја са што мање линија кода.

```
def unos_osnove():
    return float(input("Uneti osnovu: "))

def unos_stepena():
    return float(input("Uneati stepen: "))

def stepenovanje(x, y):
    return x ** y

def prikaz_rezultata(x, y, z):
    print("Za osnovu", str(x), "na stepen", str(y), "dobija se", str(z))

def main():
    osnova = unos_osnove()
    stepen = unos_stepena()
    prikaz_rezultata(osnova, stepen, stepenovanje(osnova, stepen))

main()
```

Питања и задаци за самосталан рад

Задаци

- 029 Написати програм са функцијама које израчунавају збир првих n целих бројева.
- 030 Написати програм са функцијама за рачунање збира целих бројева у датом опсегу.
- 031 Написати програм који приказује поруку који је од два унета броја већи или да су исти коришћењем функција.

097 Рачунање апсолутне вредности

Написати програм са функцијама за израчунавање апсолутне вредности унетог броја.

```
def unos_broja():
    x = float(input("Uneti broj: "))
```

```

        return x

def apsolutna(broj):
    if broj >= 0:
        return broj
    else:
        return -(broj)

def prikaz_rezultata(x, y):
    print("Apsolutna vrednost broja", x, "je", y)

def main():
    broj = unos_broja()
    rezultat = apsolutna(broj)
    prikaz_rezultata(broj, rezultat)

main()

```

098 Упоређивање случајних бројева

Програм генерише два случајна цела броја у опсегу од 1 до 100, упоређује их и приказује одговарајућу поруку на екрану о њиховом односу.

```

import random

def generisanje_brojeva():
    x = random.randint(1, 100)
    y = random.randint(1, 100)
    return x, y

def poredjenje_brojeva(x, y):
    if x > y:
        return x, y, False
    elif x < y:
        return y, x, False
    else:
        return 0, 0, True

def prikaz_rezultata(x, y, i):
    if i:
        print("Generisani brojevi su isti.")
    else:
        print("Veci broj je", x, "manji broj je", y)

def main():
    broj1, broj2 = generisanje_brojeva()
    veci, manji, isti = poredjenje_brojeva(broj1, broj2)

```

```
prikaz_rezultata(veci, manji, isti)
```

```
main()
```

099 Разлика реципрочних бројева

Написати програм са функцијама које рачунају разлику две реципрочне вредности два броја.

```
def unos_brojeva():  
    x = float(input("Uneti prvi broj: "))  
    y = float(input("Uneti drugi broj: "))  
    return x, y  
  
def recipročno(x):  
    return (1 / x)  
  
def izracunati_razliku(rec1, rec2):  
    return (rec1 - rec2)  
  
def prikaz_rezultata(x, y, z):  
    print("Razlika recipročnih vrednosti brojeva", x, "i", y, "je",  
format(z, ".2f"))  
  
def main():  
    broj1, broj2 = unos_brojeva()  
    rec1 = recipročno(broj1)  
    rec2 = recipročno(broj2)  
    razlika = izracunati_razliku(rec1, rec2)  
    prikaz_rezultata(broj1, broj2, razlika)  
  
main()
```

Питања и задаци за самосталан рад

Задаци

032 Написати програм са функцијама које рачунају површину и запремину коцке.

033 Написати програм са функцијама које рачунају запремине две коцке и одређују која коцка има већу запремину.

034 Написати програм са функцијама које приказују најмањи број од три унета.

0100 Функције за рад са листама

Написати програм са функцијама којима се уноси n елемената у листу целих бројева и приказује садржај листе.

```
def unos_broja_elementa():  
    x = int(input("Unesi broj elemenata u listi: "))  
    return x
```

```

def inicijalna_lista(x):
    L = [0 for i in range(x)]
    return L

def unos_elemenata_u_listu(x, L):
    for i in range(x):
        L[i] = int(input("A(" + str(i) + ") = "))
    return L

def prikaz_rezultata(L):
    print(L)

def main():
    n = unos_broja_elemenata()
    A = inicijalna_lista(n)
    A = unos_elemenata_u_listu(n, A)
    prikaz_rezultata(A)

main()

```

0101 Претварање радијана у степене

Написати програм са функцијама које претварају унети угао у радијанима у степене.

```

import math

def pitanje():
    return input("Da li je ugao u radijanima (r) ili u stepenima (s)? ")

def unos_ugla():
    return float(input("Uneti ugao: "))

def konverzija(x, y):
    alfa = x
    if y == "r":
        alfa = math.degrees(x)
    return alfa

def prikaz_rezultata(x):
    print("Ugao je", x, "stepeni.")

def main():
    jedinica_ugla = pitanje()
    ugao = unos_ugla()
    konv_ugao = konverzija(ugao, jedinica_ugla)
    prikaz_rezultata(konv_ugao)

```

```
main()
```

0102 Заокруживање бројева

Написати програм који омогућава кориснику да током рада уноси реалан број и заокружује га на већи цео ако је децималан део већи од 0.5 а заокружује га на мањи цео ако је децималан део мањи или једнак са 0.5.

```
import math

def uneti_broj():
    x = float(input("Uneti realan broj: "))
    return x

def zaokruzivanje_broja(x):
    y = math.floor(x)
    if x - y > 0.5:
        y = math.ceil(x)
    return y

def prikaz_rezultata(x, y):
    print("Uneti broj", x, "je zaokruzen na", y)

def main():
    broj = uneti_broj()
    zaokruzen_broj = zaokruzivanje_broja(broj)
    prikaz_rezultata(broj, zaokruzen_broj)

main()
```

Питања и задаци за самосталан рад

Задаци

035 Написати програм са функцијама које сваком елементу листе додају његову двоструку вредност.

036 Написати програм са функцијама које одређују који је унети угао већи ако је један у степенима а други у радијанима.

037 Написати програм са функцијама за израчунавање мале дијагонале, велике дијагонале, површине и запремине коцке за дату страницу коцке.

Матрице

Листе

Листе су сложени типови података и базирају се на осталим типовима података.

Сваки податак у листи јесте елемент листе и њему се може приступити преко његове позиције у листи.

Позиција елемента у листи је одређена његовим индексом.

Индекс је целобројна вредност и први елемент у листи има индекс 0.

Елементи листе се увек смештају у повезаним меморијским локацијама.

Листа може бити једнодимензионалана, дводимензионалана (матрица) или вишедимензионалана (коцка).

Код једнодимензионалне листе димензија је и дужина листе (листа је димензије n).

Код вишедимензионалних листа се каже да је листа димензија $m \times n \times z$.

Вишедимензионалне листе

Када листа за своје елементе има друге листе, таква листа се назива вишедимензионална листа:

```
podlista_1 = [[lista_A_1], [lista_A_2], ...[lista_A_n]]
```

```
podlista_2 = [[lista_B_1], [lista_B_2], ...[lista_B_n]]
```

```
...
```

```
ime_visedimenzionalne_liste = [[podlista_1], [podlista_2], ...[podlista_n]]
```

Од вишедимензионалних листи најчешће се користе дводимензионалне листе или матрице.

Дводимензионалне листе - матрице

Број листа у матрици представља број редова(врста).

Број појединачних елемената у листи матрице је број колона.

Иницијализација матрица

```
A = [[1, 2, 3], [4, 5, 6]] #matrica A je inicijalizovala dve vrste po tri kolone elemenata
```

Пре било какве модификације матрице, матрица се мора иницијализовати, тј. матрица мора бити попуњена са неким елементима.

Могуће је прво доделити матрици само један елемент а касније додати и остале.

0103 Иницијализација матрице

```
A = [[0]]
```

```
print("Matrica A se sastoji od", A)
```

```
A = [[0], [0]]
```

```
print("Matrica A je sada", A)
```

У примеру је матрица A иницијализована као једнодимензионална листа са једним елементом 0.

Касније је постала дводимензионална листа (матрица), са два реда и једном колоном.

Сада је то матрица A реда 2×1 .

Иницијализација матрица са истим елементима

Ако је познато унапред какве су димензије потребне матрице, најлакше је иницијализовати матрицу постављањем свих елемената на исте почетне вредности, најчешће на 0.

0104 Иницијализација матрице list comprehension

Иницијализовати матрицу A 3×3

```
A = [[0 for x in range(3)] for y in range(3)]
```

што је исто као да је иницијализована директним уписивањем потребног броја елемената по подлистама:

```
A = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Изостављање појединих елемената матрице

Пајтон допушта да се не попуне сва места у матрици са елементма, и то само као последњи елементи у редовима.

0105 Матрица као листа

```
T = [[11, 12, 5], [15, 6, 10], [10, 8, 12, 5], [12, 15, 8, 6]]
print(T)
```

што даје: `[[11, 12, 5], [15, 6, 10], [10, 8, 12, 5], [12, 15, 8, 6]]`

У примеру се види да у прва два реда матрице има по три елемента (три колоне су попуњене) а у друга два су по четири елемента.

Пајтон допушта ову недоследност у реализацији матрице пошто су у они у сагласности са промењивости структуре типа података листа (оставља могућност да се накнадно унесу елементи).

Исто тако:

```
T = [[11, 12], [15, 6, 10], [10, 8, 12, 5], [12, 15, 8, 6, 1000]]
```

И ова матрица је синтаксно правилно иницијализована иако сваки ред матрице има неједнак број попуњених колона.

Форматирање матрице помоћу константи

0106 Попуњавање матрице преко константи

```
REDOVI = 2
```

```
KOLONE = 2
```

```
def main():
    A = [[0 for x in range(KOLONE)] for y in range(REDOVI)]
    print(A)
    for red in range(REDOVI):
        for kolona in range(KOLONE):
            A[red][kolona] = int(input("A[" + str(red) + "][" + str(kolona)
+ "] = "))
    print(A)

main()
```

```
[[0, 0], [0, 0]]
A[0][0] = 1
A[0][1] = 2
A[1][0] = 3
A[1][1] = 4
[[1, 2], [3, 4]]
```

У примеру је приказано како се помоћу константи REDOVI и KOLONE може иницијализовати матрица са 0 као почетним елементима.

Такође, исте константе се користе за појединачни унос елемената за свако место у матрици понаособ.

Прво се уносе елементи у ред 0, па у ред 1.

Приказ елемената матрице

0107 Приказ матрице као табеле

Написати код којим се елементи матрице приказују у виду матричних елемената

```
def main():
    A = [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10, 11], [12, 13, 14, 15]]
    for red in A:
        for kolona in red:
            print(str(kolona), "\t", end = " ")
        print()
```

main()

1	2	3	4
5	6	7	
8	9	10	11
12	13	14	15

У примеру се користе две for петље при чему прва петља итерира по елементима матрице A, а то су појединачни редови матрице A.

Друга петља итерира по појединачним редовима матрице A, тј по елементима у колонама одређеног реда матрице.

Приказ матрице је форматиран као низ конвертованог елемента матрице у стринг, ескејп карактер табулар (за константно развајање елемената) и argument кључна реч end која недозвољава прелазак у следећи ред приказа.

Тек када се прикажу сви елементи једног реда матрице активира се наредба print(), којом се прелази у следећи ред за приказ следећег реда матрице.

Унос новог реда у постојећу матрицу

0108 Модификација матрице уносом реда

Унос новог реда у матрицу преко методе insert

```
def main():
    A = [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10, 11], [12, 13, 14, 15]]
    print(A)
```



```

A.insert(2, [0, 0, 0, 0, 0])
for red in A:
    for kolona in red:
        print(str(kolona), "\t", end = " ")
    print()

```

```

main()
[[1, 2, 3, 4], [5, 6, 7], [8, 9, 10, 11], [12, 13, 14, 15]]
1      2      3      4
5      6      7
0      0      0      0      0
8      9      10     11
12     13     14     15

```

Коришћењем методе insert извршило се уметање новог реда матрице као трећег реда матрице A.

Приступање елементима матрице

Приступање елементима матрице се врши преко индекса: A[0] [0] је позиција првог елемента матрице A.

A[1] [0] је позиција елемента који се налази у другој врсти и првој колони матрице A.

A[0] [1] је позиција елемента који се налази у првој врсти и другој колони матрице A.

Квадратна матрица

Посебна врста матрице је квадратна матрица и она има исти број врста и колоне.

Код ње постоје и главна и помоћна дијагонала.

Дијагонала матрице дели матрицу на горњи и доњи троугао.

Индекси елемената на главној дијагонали су исти.

За горњи троугао увек важи да је $i < j$

За доњи троугао увек важи да је $i > j$

\searrow		
A ₀₀		
	\searrow	
	A ₁₁	
		\searrow
		A ₂₂

Индекси елемената на помоћној дијагонали имају збир једнак $n-1$, где је n број врста и колоне.

Изнад помоћне дијагонале однос је $i+j < n-1$

Испод помоћне дијагонале однос је $i+j > n-1$

		\swarrow
		A ₀₂
	\swarrow	
	A ₁₁	
\swarrow		
A ₂₀		

```

Unesi broj redova matrice: 3
Unesi broj kolona matrice: 4
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
67      26      46      22
46      7       100     72
44      42      29      97

```

0109 Креирање квадратне матрице

Написати програм који креира квадратну матрицу реда n, смешта 1 у главну дијагоналу матрице, смешта 2 у горњи троугао матрице и смешта 3 у доњи троугао матрице

```

n = int(input("Unesi dimenziju kvadratne matrice: "))
A = [[0 for i in range(n)] for j in range(n)]
for i in range(n):
    for j in range(n):
        if i == j:
            A[i][j] = 1
        else:
            if i < j:
                A[i][j] = 2
            else:
                A[i][j] = 3

```

0110 Квадратна матрица са случајним бројевима

Написати програм који креира матрицу жељених димензија па је попуњава случајно генерисаним целим бројевима.

```

import random

def main():
    REDOVI = int(input("Unesi broj redova matrice: "))
    KOLONE = int(input("Unesi broj kolona matrice: "))
    A = [[0 for i in range(KOLONE)] for j in range(REDOVI)]
    print(A)
    for red in range(REDOVI):
        for kolona in range(KOLONE):
            A[red][kolona] = random.randint(1, 100)

    for red in A:
        for kolona in red:
            print(str(kolona), "\t", end = " ")
        print()

main()

```

Спирална матрица

Свака матрица са n редова и m колона се може прочитати као спирална матрица.

Читање матрице као спирале започиње у левом горњем углу матрице, креће се до краја нултог реда матрице, затим преко последње колоне матрице до последњег реда матрице у супротном смеру па нагоре по нултој колони матрице до почетка првог реда матрице па спирално у круг.

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Читање матрице као спиралне

Користи се једна промењива (smr) која указује на смер читања елемената матрице.

Ако је смер читања матрице са лева на десно $smr = 0$, за смер читања одозго на доле $smr = 1$, за смер читања са десна на лево $smr = 2$, за смер читања одоздо на горе $smr = 3$.

Користе се четири промењиве ($gore$, $dole$, $levo$, $desno$) које указују на индексе елемената матрице на теменима матрице.

Почетне вредности указују на индексе граничних редова и колона матрице који још нису прочитани: $gore = 0$ (последњи горњи ред матрице који још није прочитан је 0), $dole = 3$ (последњи доњи ред матрице који још није прочитан је 3), $levo = 0$ (последња лева колона матрице која још није прочитана је 0), $desno = 3$ (последња десна колона матрице која још није прочитана је 3).

Сваки пут када се прочита горњи гранични ред матрице, $gore$ се повећава за 1 пошто тај ред матрице више није гранични.

Сваки пут када се прочита десни гранични ред матрице, $desno$ се смањује за 1 пошто тај ред матрице више није гранични.

Читање матрице као спирале се завршава када $levo$ постане веће од $desno$ и када $gore$ постане веће од $dole$.

0111 Читање матрице спирално

```
def spiralna_matrica(red, kolona, A):  
    gore = 0  
    dole = red - 1  
    levo = 0  
    desno = kolona - 1  
    smr = 0  
  
    while (gore <= dole and levo <= desno):
```

```

if smer == 0:
    #citanje sa leva na desno
    for i in range(levo, desno + 1):
        print(A[gore][i], end = " ")
    #procitan je granicni red pa se menjaju vrednosti
    gore += 1
    smer = 1

elif smer == 1:
    #citanje odozgo nadole
    for i in range(gore, dole + 1):
        print(A[i][desno], end = " ")
    #procitan je granicni red pa se menjaju vrednosti
    desno -= 1
    smer = 2

elif smer == 2:
    #citanje sa desna na levo
    for i in range(desno, levo - 1, -1):
        print(A[dole][i], end = " ")
    dole -= 1
    smer = 3

elif smer == 3:
    #citanje odozdo na gore
    for i in range(dole, gore - 1, -1):
        print(A[i][levo], end = " ")
    levo += 1
    smer = 0

print()

def main():
    matrica = [[1, 2, 3, 4], [12, 13, 14, 5], [11, 16, 15, 6], [10, 9, 8, 7]]
    spiralna_matrica(4, 4, matrica)

```

main()

Дaje:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16

Задаци са матрицама

0112 Креирање матрице функцијама

Написати програм који коришћењем функција креира матрицу реда 4 x 3, попуњава је са 2 а затим и приказује матрицу.

```
def kreiranje_matrice(x, y):
```

```

M = [[2 for j in range(y)] for i in range(x)]
return M

def prikaz_rezultata(M):
    for red in M:
        for kolona in red:
            print(str(kolona), "\t", end = "")
        print()

def main():
    A = kreiranje_matrice(4, 3)
    prikaz_rezultata(A)

main()

```

0113 Сумирање елемената матрице

Написати програм који коришћењем функција креира матрицу реда 3 x 2, попуњава први ред са 0, други са 1 а трећи са 2, затим и приказује матрицу и укупан збир свих елемената матрице.

```

def kreiranje_matrice():
    M = [[0, 0], [1, 1], [2, 2]]
    return M

def racunanje_sume_elemenata_matrice(M):
    suma = 0
    for red in M:
        for kolona in red:
            suma += kolona
    return suma

def prikaz_rezultata(M, suma):
    for red in M:
        for kolona in red:
            print(str(kolona), "\t", end = "")
        print()
    print("Ukupan zbir elemenata je", suma)

def main():
    A = kreiranje_matrice()
    zbir = racunanje_sume_elemenata_matrice(A)
    prikaz_rezultata(A, zbir)

main()

```

0114 Корисничко креирање матрице

Написати програм који коришћењем функција креира матрицу реда $m \times n$, попуњава је елементима а затим и приказује матрицу.

```
def unos_dimenzije_matrice():
    x = int(input("Uneti broj redova matrice: "))
    y = int(input("Uneti broj kolona matrice: "))
    return x, y

def inicijalizacija_matrice(x, y):
    M = [[0 for j in range(y)] for i in range(x)]
    return M

def popunjavanje_matrice(x, y, M):
    for i in range(x):
        for j in range(y):
            M[i][j] = int(input("A[" + str(i) + "][" + str(j) + "] = "))

    return M

def prikaz_rezultata(M):
    for red in M:
        for kolona in red:
            print(str(kolona), "\t", end = " ")
        print()

def main():
    m, n = unos_dimenzije_matrice()
    A = inicijalizacija_matrice(m, n)
    A = popunjavanje_matrice(m, n, A)
    prikaz_rezultata(A)

main()
```

Питања и задаци за самосталан рад

Задаци

038 Написати програм који коришћењем функција креира квадратну матрицу реда 3, попуњава је са потребном вредности а затим и приказује матрицу.

039 Написати програм који коришћењем функција креира матрицу реда 2×3 , попуњава прву колону са 0, другу са 1 а трећу са 2, затим и приказује матрицу и укупан збир свих елемената матрице.

040 Написати програм који коришћењем функција креира матрицу $\begin{bmatrix} 3 & 4 \\ 7 & 0 \\ 2 & 2 \end{bmatrix}$, приказује матрицу и суме појединачних редова матрице.

0115 Рачунање суме матрице по редовима

Написати програм који коришћењем функција креира матрицу реда $m \times n$, попуњава је елементима, израчунава суму сваког реда и приказује матрицу и израчунате суме.

```
def unos_dimenzije_matrice():
    x = int(input("Uneti broj redova matrice: "))
    y = int(input("Uneti broj kolona matrice: "))
    return x, y

def inicijalizacija(x, y):
    M = [[0 for j in range(y)] for i in range(x)]
    N = [0 for i in range(x)]
    return M, N

def popunjavanje_matrice(x, y, M):
    for i in range(x):
        for j in range(y):
            M[i][j] = int(input("A[" + str(i) + "][" + str(j) + "] = "))

    return M

def racunanje_sume_redova(x, y, M, N):
    for i in range(x):
        for j in range(y):
            N[i] += M[i][j]

    return N

def prikaz_rezultata(M, N):
    for red in M:
        for kolona in red:
            print(str(kolona), "\t", end = " ")
        print()

    print(N)

def main():
    m, n = unos_dimenzije_matrice()
    A, S = inicijalizacija(m, n)
    A = popunjavanje_matrice(m, n, A)
    S = racunanje_sume_redova(m, n, A, S)
    prikaz_rezultata(A, S)

main()
```

0116 Пребројавање елемената матрице

Написати програм који открива колико је елемената матрице парно, непарно, једнако 0, веће од 0 и мање од 0.

```
def unos_dimenzije_matrice():
    x = int(input("Uneti broj redova matrice: "))
    y = int(input("Uneti broj kolona matrice: "))
    return x, y

def inicijalizacija_matrice(x, y):
    M = [[0 for j in range(y)] for i in range(x)]
    return M

def popunjavanje_matrice(x, y, M):
    for i in range(x):
        for j in range(y):
            M[i][j] = int(input("A[" + str(i) + "][" + str(j) + "] = "))

    return M

def prebrojavanje_po_parnosti(x, y, M):
    par, npar, nul = 0, 0, 0
    for i in range(x):
        for j in range(y):
            if M[i][j] == 0:
                nul += 1
            elif M[i][j] % 2 == 0:
                par += 1
            elif M[i][j] % 2 == 1:
                npar += 1

    return par, npar, nul

def prebrojavanje_po_vrednosti(x, y, M):
    veci, manji = 0, 0
    for i in range(x):
        for j in range(y):
            if M[i][j] > 0:
                veci += 1
            elif M[i][j] < 0:
                manji += 1

    return veci, manji

def prikaz_rezultata(M, par, npar, nul, veci, manji):
    for red in M:
        for kolona in red:
            print(str(kolona), "\t", end = " ")
```



```

print()

print("Broj parnih elemenata u matrici je", par)
print("Broj neparnih elemenata u matrici je", npar)
print("Broj nula elemenata u matrici je", nul)
print("Broj elemenata vecih od nula u matrici je", veci)
print("Broj elemenata manjih od nula u matrici je", manji)

def main():
    m, n = unos_dimenzije_matrice()
    A = inicijalizacija_matrice(m, n)
    A = popunjavanje_matrice(m, n, A)
    parni, neparni, nule = prebrojavanje_po_parnosti(m, n, A)
    veci_od_0, manji_od_0 = prebrojavanje_po_vrednosti(m, n, A)
    prikaz_rezultata(A, parni, neparni, nule, veci_od_0, manji_od_0)

main()

```

Питања и задаци за самосталан рад

Задаци

041 Написати програм који после уноса матрице реда $m \times n$ креира нову матрицу чији су елементи квадрати елемената прве матрице.

042 Написати програм који помоћу функција попуњава матрицу димензије 3×3 а затим креира нову матрицу истих димензија која прву колону прве матрице има као последњу колону.

0117 Креирање матрице непознатих димензија

Написати програм којим се уносе целобројне вредности у матрицу непознатих димензија

```

A = []
b = []
unos = input("Uneti n za nov red, vrednost za element matrice, q za kraj: ")
while unos != "q":
    while unos != "n":
        b.append(int(unos))
        unos = input("Uneti: ")
        if unos == "q" or unos == "n":
            break

    A.append(b)
    b = []
    unos = input("Uneti: ")

print(A)

```

0118 Уређење матрице по дијагоналама

Написати програм који коришћењем функција креира квадратну матрицу реда n у којој су сви елементи на главној дијагонали једнаки 1, изнад ње једнаки 2 а испод ње једнаки 3.

```
def unos_dimenzije_matrice():
    return int(input("Uneti dimenziju kvadratne matrice: "))

def inicijalizacija_matrice(x):
    M = [[0 for i in range(x)] for j in range(x)]
    return M

def popunjavanje_matrice(x, M):
    for i in range(x):
        for j in range(x):
            if i == j:
                M[i][j] = 1
            else:
                if i < j:
                    M[i][j] = 2
                else:
                    M[i][j] = 3
    return M

def prikaz_rezultata(M):
    for red in M:
        for kolona in red:
            print(str(kolona), "\t", end = " ")
        print()

def main():
    n = unos_dimenzije_matrice()
    A = inicijalizacija_matrice(n)
    A = popunjavanje_matrice(n, A)
    prikaz_rezultata(A)
```

```
main()
Uneti dimenziju kvadratne matrice: 4
1       2       2       2
3       1       2       2
3       3       1       2
3       3       3       1
```

0119 Сумирање матрице по дијагоналама

Написати програм за израчунавање сума елемената матрице који се налазе изнад и посебне суме елемената који се налазе испод помоћне дијагонале квадратне матрице реда n .

```
def unos_dimenzije_matrice():
    return int(input("Uneti dimenziju kvadratne matrice: "))
```

```

def inicijalizacija_matrice(x):
    M = [[0 for i in range(x)] for j in range(x)]
    return M

def popunjavanje_matrice(x, M):
    for i in range(x):
        for j in range(x):
            M[i][j] = int(input("A[" + str(i) + "][" + str(j) + "] = "))

    return M

def racunanje_sume_iznad(x, M):
    suma = 0
    for i in range(x):
        for j in range(x):
            if i + j < x - 1:
                suma += M[i][j]
    return suma

def racunanje_sume_ispod(x, M):
    suma = 0
    for i in range(x):
        for j in range(x):
            if i + j > x - 1:
                suma += M[i][j]
    return suma

def prikaz_rezultata(M, suma1, suma2):
    for red in M:
        for kolona in red:
            print(str(kolona), "\t", end = " ")
        print()

    print("Zbir elemenata iznad pomocne dijagonale je", suma1)
    print("Zbir elemenata ispod pomocne dijagonale je", suma2)

def main():
    n = unos_dimenzije_matrice()
    A = inicijalizacija_matrice(n)
    A = popunjavanje_matrice(n, A)
    suma_iznad = racunanje_sume_iznad(n, A)
    suma_ispod = racunanje_sume_ispod(n, A)
    prikaz_rezultata(A, suma_iznad, suma_ispod)

main()

```

Uneti dimenziju kvadratne matrice: 3

A[0][0] = 1

A[0][1] = 2

A[0][2] = 3

A[1][0] = 1

A[1][1] = 2

A[1][2] = 3

A[2][0] = 1

A[2][1] = 2

A[2][2] = 3

1 2 3

1 2 3

1 2 3

Zbir elemenata iznad pomocne dijagonale je 4

Zbir elemenata ispod pomocne dijagonale je 8

Питања и задаци за самосталан рад

Задаци

043 Написати програм који коришћењем функција креира квадратну матрицу реда n, попуњава је елементима, израчунава суму главне и помоћне дијагонале и приказује матрицу и израчунате суме.

044 Написати програм који коришћењем функција креира матрицу реда m x n, попуњава је елементима тако што ако је збир индекса позиције елемента између 5 и 8 убацује 1 а иначе се убацује 0, и приказује матрицу.

045 Матрицу [[“p”, “a”, “j”], [“c”, “o”, “t”], [“i”, “o”, “o”], [“a”, “l”, “n”], [“n”, “e”, “j”]] прочитати као спиралну матрицу.

Стрингови

Приступање појединачним карактерима у стрингу

У досадашњем раду, стрингови су се користили као улаз са тастатуре или као излаз на екрану рачунара.

Пајтон има велики број алата и програмерских техника којима се претражује и манипулише стринговима.

Рад са појединачним карактерима у стрингу је употребљив у случајевима када се примењују технике провере стринга: провера унете лозинке према задатим параметрима (једно велико слово, макар једна цифра...).

Најчешће се користе две технике за приступ појединачним знацима у стрингу: употреба for петље и индексирање.

Итерација по стрингу са for петљом

Општи формат приступа знаку у стрингу са for петљом:

```
for promenjiva in string:
```

```
    iskaz
```

```
...
```

Где је променјива име променјиве а string је или стринг литерал или променјива која указује на стринг.

Сваки пут када се изврши итерација у петљи, променјива ће указати на копију знака унутар string-а, почевши од првог карактера. То се назива итерација преко карактера у стрингу.

0120 Итерација преко карактера у стрингу

```
ime = 'Julija'
for znak in ime:
    print(znak)
```

У задатку, променјива ime указује на стринг са 6 карактера, а то значи да ће петља да итерира шест пута, у првој итерацији ime указује на копију првог знака у стрингу а то је слово J, у другој на слово u...

0121 Број пута појаве карактера у стрингу

```
def main():
    suma = 0
    moj_string = input("Uneti recenicu: ")
    for znak in moj_string:
        if znak == 'T' or znak == 't':
            suma += 1

    print('Znak T se pojavljuje', suma, 'puta u recenici.')

main()
Uneti recenicu: Tamo je tviter samotan.
Znak T se pojavljuje 4 puta u recenici.
```

Индексирање

Индексирање је други начин за приступ појединачним знацима у стрингу.

Сваки карактер у стрингу има индекс којим се описује позиција карактера у низу карактера стринга.

Индекси се обележавају од 0, па је индекс последњег знака у стрингу једнак броју знакова умањен за један.

И празна места и знаци интерпункције заузимају једну позицију па и њима одговара један индекс у стрингу.

0122 Приступ стрингу индексирањем

```
recenica = "Ceo svet u kutiji."
```

```
znak = recenica[4]
```

Израз `recenica[4]` враћа копију карактера са индексом 4 у стрингу `recenica`.
Када се други исказ у задатку изврши, промењива `znak` ће указивати на знак 's'.

0123 Вишеструко индексирањем

```
recenica = "Ceo svet u kutiji."  
print(recenica[4], recenica[2], recenica[11])  
Даје: s o k
```

Такође се могу користити и негативни бројеви као индекси, чиме се указује на почетак бројања индекса са краја стринга; последњи знак у стрингу има индекс -1, следећи -2...

Грешка IndexError

Грешка `IndexError` (exception) ће се појавити када се покуша користити индекс који је изван опсега за одређени стринг.

Ако стринг има укупно 6 карактера, то значи да су њихови индекси од 0 до 5. Употреба индекса изван тог опсега индекса ће изазвати појаву `IndexError` грешке.

0124 Појава `IndexError` изузетка

```
recenica = "Beograd"  
print(recenica[7])  
IndexError: string index out of range
```

Функција len

Осим што функција `len` даје дужину секвенце, она може дати и дужину стринга, тј број карактера у стрингу.

0125 Излаз функције `len()`

```
grad = "Beograd"  
print(len(grad))  
Даје: 7
```

Други исказ у задатку позива функцију `len` са промењивом `grad` као аргументом.
Функција враћа вредност 7, пошто је то број знакова у стрингу `Beograd`.

0126 Спречавање итерације преко опсега индекса

Употреба функције `len` за спречавање итерације преко дозвољеног опсега индекса у стрингу

```
grad = 'Beograd'  
indeks = 0  
while indeks < len(grad):  
    print(grad[indeks])  
    indeks += 1
```

Сада вредност индекса не може отићи преко 6 и на тај начин се спречава појава `IndexError` ехцеption.

Надовезивање стрингова

Једна од најчешће коришћених операција над стринговима је надовезивање (concatenation), којом се описује додавање другог стринга на крај првог стринга.

Оператор `+` даје за резултат стринг који је комбинација два стринга као операнда.

0127 Просто надовезивање стрингова

```
poruka = 'Zdravo ' + 'svete'
print(poruka)
Даје: Zdravo svete
```

Може се користити и оператор `+=` да би се извршило надовезивање стрингова.

0128 Употреба `+=` оператора у надовезивању стрингова

```
slova = 'kop'
slova += 'ati'
print(slova)
Даје: kopati
```

Битно је приметити да операнд са леве стране `+=` оператора мора бити постојећа промењива. Ако би се ту користила непостојећа промењива, јавила би се грешка.

Непромењивост стрингова

У Пајтону, стрингови су непромењиви.

То значи, да када се једном креира стринг, он се не може променити.

Неке операције, попут надовезивања, дају погрешан утисак да је дошло до модификације постојећег стринга.

0129 Приказ непромењивости стринга

```
def main():
    ime = 'Dragana'
    print('Ime je', ime)
    ime = ime + ' Misic'
    print('Sada je ime', ime)
```

```
main()
```

Даје:

Ime je Dragana

Sada je ime Dragana Misic

Прво се додели стринг **'Dragana'** промењивој `ime`, а затим се изврши надовезивање два стринга и додели добијени резултат промењивој `ime`.

Међутим, пошто се стрингови не могу мењати, додела новог стринга истој вредности се изводи креирањем новог стринга и његовим додељивањем истој промењивој.

Тада се стара вредност, у овом случају стринг **'Dragana'** више не користи јер не постоји промењива која указује на њу.

Зато се не може користити израз у облику, `string[indeks]` на левој страни оператора доделе.

0130 Недозвољено коришћење оператора доделе

```
ime = 'Bill'
```

```
ime[0] = 'J'
```

```
TypeError: 'str' object does not support item assignment
```

Овај пример изазива грешку пошто покушава променити вредност првог карактера у датом стрингу.

0131 Доказ непромењивости стринга

```
rec1 = input("Uneti u promenjivu rec1: ")
```

```
print("Memorijska lokacija na koju ukazuje promenjiva rec1 je: ", id(rec1))
```

```
rec = rec1
```

```
print("Promenjiva rec: ", rec)
```

```
print("Memorijska lokacija na koju ukazuje promenjiva rec je: ", id(rec))
```

```
rec2 = input("Uneti u promenjivu rec2: ")
```

```
print("Memorijska lokacija na koju ukazuje promenjiva rec2 je: ", id(rec2))
```

```
rec1 += rec2
```

```
print("Sadržaj promenjive rec1 je sada: ", rec1)
```

```
print("Memorijska lokacija na koju ukazuje promenjiva rec1 je: ", id(rec1))
```

```
print("Sadržaj promenjive rec je: ", rec)
```

```
print("Memorijska lokacija na koju ukazuje promenjiva rec je: ", id(rec))
```

Uneti u promenjivu rec1: Ja

Memorijska lokacija na koju ukazuje promenjiva rec1 je: 1219009863728

Promenjiva rec: Ja

Memorijska lokacija na koju ukazuje promenjiva rec je: 1219009863728

Uneti u promenjivu rec2: Ti

Memorijska lokacija na koju ukazuje promenjiva rec2 je: 1219009860592

Sadržaj promenjive rec1 je sada: JaTi

Memorijska lokacija na koju ukazuje promenjiva rec1 je: 1219009863824

Sadržaj promenjive rec je: Ja

Memorijska lokacija na koju ukazuje promenjiva rec je: 1219009863728

Одсецање стрингова

Сваки пут када се изврши одсецање (slicing) стринга, добија се низ карактера из унутрашњости стринга.

Овако добијен низ карактера се назива подстинг (субстринг).

Општи формат за одсецање стринга:

```
string[start : kraj]
```

У општем формату, start је индекс првог карактера у одсечку (slice) док је kraj индекс којим се обележава завршетак одсечка.

У оваквом формату, израз ће вратити стринг са копијом карактера од индекса start до (не укључујући) и индекс kraj.

0132 Одсецање стринга са два индекса

```
puno_ime = 'Dusko Zeka Dugousko'
srednje_ime = puno_ime[6:10]      #srednje_ime = Zeka
```

У случају да се изостави start индекс као ознака одсечка, претпоставља се вредност 0 за индекс почетка одсечка, којом се означава први индекс у стрингу.

0133 Одсецање стринга са десним индексом

```
puno_ime = 'Dusko Zeka Dugousko'
ime = puno_ime[:5]                #ime = Dusko
```

У случају да се изостави индекс kraj, претпоставља се вредност дужине стринга за индекс краја одсечка.

0134 Одсецање стринга са левим индексом

```
puno_ime = 'Dusko Zeka Dugousko'
prezime = puno_ime[11:]           #prezime = Dugousko
```

Ако се изоставе и индекс start и индекс kraj, претпоставља се да је одсечак цео оригинални стринг.

0135 Одсецање стринга без индекса

```
puno_ime = 'Dusko Zeka Dugousko'
moj_string = puno_ime[:]          #moj_string = Dusko Zeka Dugousko
```

Други начин добијања целог оригиналног стринга као одсечка:

```
moj_string = puno_ime[0:len(puno_ime)]
```

У општем формату одсецања стринга, може се унети и трећа вредност која представља корак (step) којом се одређује колико ће се карактера прескакати у избору карактера за одсецање.

0136 Одсецање стринга са три индекса

```
puno_ime = 'Dusko Zeka Dugousko'
print(puno_ime[0:20:2])           #DsoZk uoso
```

Могу се користити и негативни бројеви као индекси за одсецање стрингова.

Негативни бројеви се односе према крају оригиналног стринга.

0137 Одсецање стринга са негативним индексом

```
puno_ime = 'Dusko Zeka Dugousko'
prezime = puno_ime[-8:]          #prezime = Dugousko
```

Интересантно је знати да погрешни индекси код одсецања стрингова не изазивају појаву грешке. Ако је индекс крај изван опсега индекса стринга, Пајтон ће аутоматски користити дужину стринга. Ако је индекс start изван опсега индекса стринга, Пајтон ће аутоматски користити 0 уместо њега. Ако је start индекс већи од крај индекса, одсецање ће аутоматски вратити празан стринг.

0138 Креирање шифре одсецањем стринга

Написати програм који додељује сваком ученику име за логовање на рачунарски систем школе тако што узима прва три знака из имена ученика, прва три знака из презимена ученика и последња три знака из ID броја ученика и надовезује их у стринг.

Модуло login:

```
def dobijanje_login_ime(ime, prezime, idbroj):
    deo1 = ime[0 : 3]
    deo2 = prezime[0 : 3]
    deo3 = idbroj[-3 :]
    login_ime = deo1 + deo2 + deo3
    return login_ime
```

Главни део програма:

```
import login

def main():
    ime = input('Uneti ime ucenika: ')
    prezime = input('Uneti prezime ucenika: ')
    idbroj = input('Uneti ucenikov ID broj: ')
    print('Ucenikovo login ime je:')
    print(login.dobijanje_login_ime(ime, prezime, idbroj))
```

```
main()
```

Тестирање стрингова

Ако је потребно проверити да ли се неки стринг налази у другом стрингу користи се оператор in.

Општи формат:

```
string1 in string2
```

где су стринг1 и стринг2 или стринг литерали или променљиве које упућују на стрингове.

0139 Употреба оператора in

```
tekst = 'Proslo je skoro sedam godina.'
if 'sedam' in tekst:
    print('String "sedam" je pronadjen u tekstu.')
else:
```

```
print('String "sedam" nije pronadjen u tekstu.')
```

Ако је потребно одредити да ли се стринг не садржи у другом стрингу користи се оператор `not in`.

Стринг методе

Методе су функције које припадају неком објекту и изводе операције на том објекту.

Неке стринг методе у Пајтону изводе следеће операције: тестирање вредности стрингова, извођење различитих модификација, претрага за субстринговима и замена секвенце карактера.

Општи формат позива стринг методе:

```
string_promenjiva.metoda(argumenti)
```

У општем формату, `string_promenjiva` је промењива која указује на стринг, `metoda` је назив методе која се позива, а `argumenti` су један или више аргумената који се прикључују методи.

Стринг методе за тестирање

Табела неких Пајтон стринг метода за тестирање:

метода	опис
<code>isalnum()</code>	Враћа <code>True</code> ако стринг садржи само слова алфабета или цифре и да има дужину од најмање једног карактера. Иначе враћа <code>False</code>
<code>isalpha()</code>	Враћа <code>True</code> ако стринг садржи само слова алфабета и да има дужину од најмање једног карактера. Иначе враћа <code>False</code>
<code>isdigit()</code>	Враћа <code>True</code> ако стринг садржи само цифре и да има дужину од најмање једног карактера. Иначе враћа <code>False</code>
<code>islower()</code>	Враћа <code>True</code> ако стринг садржи само мала слова алфабета и да има дужину од најмање једног карактера. Иначе враћа <code>False</code>
<code>isspace()</code>	Враћа <code>True</code> ако стринг садржи само карактере белог места и да има дужину од најмање једног карактера. Иначе враћа <code>False</code> . Карактери белог места су празнине, карактер нове линије и таб.
<code>isupper()</code>	Враћа <code>True</code> ако стринг садржи само велика слова алфабета и да има дужину од најмање једног карактера. Иначе враћа <code>False</code>

0140 Метода `isdigit()`

```
string1 = '1200'
if string1.isdigit():
    print(string1, 'sadrzi samo cifre.')
else:
    print(string1, 'sadrzi i karaktere koji nisu cifre.')

string2 = '123abc'
if string2.isdigit():
    print(string2, 'sadrzi samo cifre.')
else:
    print(string2, 'sadrzi i karaktere koji nisu cifre.')
```

0141 Методе за тестирање стринга

```
def main():
```

```

korisnicki_string = input('Uneti string: ')
print('Ovo je otkriveno o unetom stringu:')
if korisnicki_string.isalnum():
    print('String je alfanumericki.')
if korisnicki_string.isdigit():
    print('String ima samo cifre.')
if korisnicki_string.isalpha():
    print('String ima samo slova.')
if korisnicki_string.isspace():
    print('String ima samo karaktere belog mesta.')
if korisnicki_string.islower():
    print('Sva slova u stringu su mala.')
if korisnicki_string.isupper():
    print('Sva slova u stringu su velika.')

```

main()

Методе за модификацију стрингова

Иако су стрингови непромењиви, иако то значи да се заправо они не могу модификовати, постоји доста метода које враћају модификоване верзије стрингова.

метода	опис
lower()	Враћа копију стринга са свим словима конвертованим у мала слова. Било који карактер који је већ мали или није слово, остаје непромењен
lstrip()	Враћа копију стринга са уклоњеним свим водећим белим карактерима (бели карактери који су на почетку стринга)
lstrip(znak)	Аргумент znak је стринг који садржи карактер. Враћа копију стринга са свим инстанцама знак стринга који се појављују на почетку одстрањеног стринга
rstrip()	Враћа копију стринга са уклоњеним свим последњим белим карактерима (бели карактери који су на крају стринга)
rstrip(znak)	Аргумент znak је стринг који садржи карактер. Враћа копију стринга са свим инстанцама знак стринга који се појављују на крају одстрањеног стринга
strip()	Враћа копију стринга којем су сви бели карактери на почетку и на крају стринга одстрањени
strip(znak)	Враћа копију стринга са свим инстанцама знака који се појавио на почетку и на крају стринга одстрањеним
upper()	Враћа копију стринга са свим словима конвертованим у велика слова. Било који карактер који је већ велики или није слово, остаје непромењен

0142 Метода lower()

```

ponovo = 'd'
while ponovo.lower() == 'd':
    print('Zdravo')
    print('Da li treba ponovo pokazati?')
    ponovo = input('d = da, bilo sta drugo = ne: ')

```

На овај начин је обезбеђено да се петља извршава без обзира на величину унетог слова d.

Методе за претрагу и замену

Често се у програмима захтева претрага за субстринговима или стринговима који се појављују унутар других стрингова.

метода	опис
endswith(substring)	Аргумент substring је стринг. Метода враћа True ако се стринг завршава са субстрингом
find(substring)	Аргумент substring је стринг. Метода враћа најнижи индекс у стрингу где је субстринг пронађен. Ако субстринг није пронађен, метода враћа -1
replace(stari, novi)	И old и new аргументи су стрингови. Метод враћа копију стринга са свим инстанцама стринга old замењеним са стрингом new
startswith(substring)	Аргумент substring је стринг. Метода враћа True ако стринг започиње са субстрингом

0143 Методе за претрагу стринга

```
fajl = input('Uneti ime fajla: ')
if fajl.endswith('.txt'):
    print('To je ime tekstualnog fajla.')
elif fajl.endswith('.py'):
    print('To je ime Pajton sors fajla.')
elif fajl.endswith('.doc'):
    print('To je ime dokumenta tekst procesora.')
else:
    print('Nepoznat tip fajla.')
```

У задатку се одређује да ли стринг се завршава са одређеним субстрингом.

0144 Метода find()

```
string = 'Jednom davno u dalekoj galaksiji.'
pozicija = string.find('dalekoj')
if pozicija != -1:
    print('Rec "dalekoj" je pronadjena na indeksu', pozicija)
else:
    print('Rec "dalekoj" nje pronadjena.')
```

У примеру, метод find претражује одређени субстринг унутар стринга.

Метода враћа најнижи индекс појаве субстринга, ако га има.

Ако не постоји, враћа се -1.

0145 Метода за замену стринга

```
stari_string = 'Jednom davno, u dalekoj galaksiji.'
novi_string = stari_string.replace('galaksiji', 'biblioteci')
```



```
datum_lista = datum_string.split('/')
print(datum_lista)
['11', '26', '2014']
```

0149 Читање делова стрингова помоћу сепаратора

```
def main():
    datum_string = '26/11/2019'
    datum_lista = datum_string.split('/')
    print('Dan:', datum_lista[0])
    print('Mesec:', datum_lista[1])
    print('Godina:', datum_lista[2])
```

```
main()
```

```
Dan: 26
```

```
Mesec: 11
```

```
Godina: 2019
```

Задаци са стринговима

0150 Приказ унетих стрингова

Написати програм који за унето име и презиме исписује сваки од података у посебном реду а затим и оба податка у истом реду.

```
def main():
    ime = input("Unesi ime: ")
    prezime = input("Unesi prezime: ")
    print("Ispis imena i prezimena u dva reda:")
    print(ime, prezime, sep = '\n')
    print("Ispis imena i prezimena u istom redu:")
    print(ime, prezime, sep = ' ')
```

```
main()
```

0151 Употреба функција ASCII кодова

Написати програм који ће у стринг уписати све велика слова енглеског језика и онда их приказати.

```
def main():
    slova = [] #inicijalizacija liste
    #smestanje svih velikih slova kao elemenata u listu
    for i in range(ord('A'), ord('Z') + 1):
        slova += chr(i)

    linija_stringa = "" #inicijalizacija stringa
    #preuzimanje elemenata iz liste i kreiranje stringa
    for i in slova:
        linija_stringa += str(i)
```

```
print (linija_stringa)  #prikaz stringa
```

```
main()
```

0152 Постављање карактера на индекс

Написати програм који ће у стрингу додати карактер на одређену позицију.

```
def main():
    nova = ""
    recenica = input("Unesi string: ")
    poz = int(input("Na kojem indeksu treba dodati novi znak? "))
    znak = input("Unesi nov znak: ")
    for i in range(len(recenica)):
        if i == poz:
            nova += znak
            nova += recenica[i]

    print(nova)
```

```
main()
```

Питања и задаци за самосталан рад

Задаци

046 Написати програм који употребљава функције ASCII кодова уз употребу сопствених функција.

047 Написати програм који у стринг од 10 карактера додаје знак "*" на позиције 2, 5 и 9.

048 Написати програм који знак "a" замењује са "A" где год да се налази у стрингу.

0153 Замена карактера на индексу

Написати програм који ће у стрингу заменити карактер на одређеној позицији.

```
def main():
    nova = ""
    recenica = input("Unesi string: ")
    poz = int(input("Na kojem indeksu je znak koji treba zameniti? "))
    znak = input("Unesi nov znak: ")
    for i in range(len(recenica)):
        if (i < poz):
            nova += recenica[i]
        elif (i > poz):
            nova += recenica[i]
        else:
            nova += znak

    print(nova)
```



```
main()
```

0154 Замена малих са великим словима

Написати програм који ће сва мала слова заменити са истим а великим словима.

```
def unos_stringa():
    poc_str = input("Unesi string: ")
    return poc_str

def konverzija_stringa_u_listu(poc_str):
    poc_lista = list(poc_str)
    return poc_lista

def mala_u_velika(poc_lista):
    for x in range(len(poc_lista)):
        broj = ord(poc_lista[x])
        if broj >= 97 and broj <= 122:
            broj -= 32
            poc_lista[x] = chr(broj)

    return poc_lista

def konverzija_liste_u_string(nov_lista):
    nov_string = ""
    for i in nov_lista:
        nov_string += str(i)
    return nov_string

def prikaz_stringa(nov_string):
    print("Modifikovani string je:", nov_string + ".")

def main():
    pocetni_string = unos_stringa()
    pocetna_lista = konverzija_stringa_u_listu(pocetni_string)
    nova_lista = mala_u_velika(pocetna_lista)
    novi_string = konverzija_liste_u_string(nova_lista)
    prikaz_stringa(novi_string)
```

```
main()
```

Питања и задаци за самосталан рад

Задаци

049 Написати програм који ће променити све знакове од позиције 2 до позиције 5 у стрингу у знак "*" .

050 Написати програм који конвертује унети стринг у листу коришћењем функција а затим свако

појављивање елемента “а” претвара у елемент “А”.

0155 Избор дела стринга

Написати програм који омогућава кориснику да одабере део унетог стринга. Приказати део стринга који је корисник одабрао.

```
def unos_stringa():
    poc_str = input("Unesi string: ")
    return poc_str

def definisanje_indeksa_odsecka(poc_string):
    indeks1 = int(input("Unesti indeks starta odsecka: "))
    indeks2 = int(input("Unesti indeks kraja odsecka: ")) + 1
    return indeks1, indeks2

def kreiranje_odsecka(indeks1, indeks2, poc_string):
    nov_string = poc_string[indeks1 : indeks2]
    return nov_string

def prikaz_stringa(nov_string):
    print("Odabrani deo stringa je:", nov_string + ".")

def main():
    pocetni_string = unos_stringa()
    start, kraj = definisanje_indeksa_odsecka(pocetni_string)
    novi_string = kreiranje_odsecka(start, kraj, pocetni_string)
    prikaz_stringa(novi_string)

main()
```

0156 Претрага дела стринга

Написати програм који проналази у стрингу “Velika lepa obala” стринг “lepa”.

```
def unos_stringova():
    vel_str = input("Unesi veliki string: ")
    mal_str = input("Unesi mali string: ")
    return vel_str, mal_str

def otkrivanje_stringa_u_stringu(vel_str, mal_str):
    duz_mal = len(mal_str)
    duz_vel = len(vel_str)
    nasao = False
    for x in range(duz_vel):
        if x == duz_vel - duz_mal + 1:
            break
        else:
            if vel_str[x : (x + duz_mal)] == mal_str:
```

```

        nasao = True
    return nasao

def prikaz_rezultata(odgovor):
    if odgovor == False:
        poruka = "se ne nalazi"
    else:
        poruka = "se nalazi"

    print("Trazeni string", poruka, "u datom stringu.")
def main():
    veliki_string, mali_string = unos_stringova()
    nalazi_se = otkrivanje_stringa_u_stringu(veliki_string, mali_string)
    prikaz_rezultata(nalazi_se)
main()

```

0157 Проналажење цифре у стрингу

Написати програм коришћењем Пајтон метода који даје поруку "Има cifru" ако у унетом стрингу има макар једна цифра а ако нема ниједна цифра даје поруку "Nema cifru".

```

def unos_stringa():
    poc_str = input("Uneti string: ")
    return poc_str

def provera_cifre_u_stringu(poc_str):
    odg = False
    for i in poc_str:
        if i.isdigit() == True:
            odg = True
            break

    return odg

def prikaz_rezultata(odg):
    if odg == False:
        print("Nema cifru.")
    else:
        print("Ima cifru.")

def main():
    pocetni_string = unos_stringa()
    odgovor = provera_cifre_u_stringu(pocetni_string)
    prikaz_rezultata(odgovor)

main()

```

051 Написати програм који конвертује унети стринг у листу коришћењем функција а затим свако појављивање два узастопна иста слова третира као грешку у куцању па избацује једно од тих слова из листе.

052 Написати програм који омогућава кориснику да одабере део унетог стринга и да га постави на почетак и на крај стринга.

053 Написати програм који открива колико пута се мањи стринг појављује у већем стрингу.

0158 Проналажење великих слова стринг методом

Написати програм коришћењем Пајтон метода који преброји колико је карактера у датом стрингу са великим словима, испише њихов број и испише саме карактере као посебан стринг.

```
def unos_stringa():
    poc_str = input("Uneti string: ")
    return poc_str

def kreiranje_novog_stringa(poc_str):
    #trenutni broj velikih slova u stringu
    br = 0
    #string u koji ce se smestati velika slova iz pocetnog stringa
    vel_sl_str = ""
    for i in poc_str:
        if i.isupper() == True:
            br += 1
            vel_sl_str += i

    return br, vel_sl_str

def prikaz_rezultata(br, vel_sl_str):
    print("U pocetnom stringu ima", br, "velikih slova.")
    print("i ona su: ", vel_sl_str, ".")

def main():
    pocetni_string = unos_stringa()
    broj, velika_slova_string = kreiranje_novog_stringa(pocetni_string)
    prikaz_rezultata(broj, velika_slova_string)

main()
```

0159 Конвертовање датума у другачији формат

Написати програм коришћењем Пајтон метода који унети датум у облику стринга у формату "dd/mm/yyyy" конвертује у стринг формата: "Mesec Dan, Godina" (10/05/2011 даје Мај 10, 2011.godine).

```

def unos_datuma():
    dat_str = input("Unesi datum u formatu dd/mm/yyyy (bez prve 0): ")
    return dat_str

def ekstrakcija_delova_datuma(dat_str):
    lista = dat_str.split("/")
    d = lista[0]
    m = lista[1]
    g = lista[2]
    return d, m, g

def prikaz_rezultata(M, d, m, g):
    print(str(M[m - 1]), d + ",", g + ".godine")

def main():
    MESECI = ["Januar", "Februar", "Mart", "April", "Maj", "Jun", "Jul",
"Avgust",\
    "Septembar", "Oktobar", "Novembar", "Decembar"]
    datum_string = unos_datuma()
    dan, mesec, godina = ekstrakcija_delova_datuma(datum_string)
    mesec = int(mesec)
    prikaz_rezultata(MESECI, dan, mesec, godina)

main()

```

Питања и задаци за самосталан рад

Задаци

- 054 Написати програм који допушта кориснику унос стринга по стринг све док корисник не унесе негативан одговор на питање о даљем уносу. Програм на излазу даје средњу вредност броја празних места у свим унетим стринговима.
- 055 Стринг је са великим словима на почетку сваке нове речи. Написати програм који убацује празна места пре великих слова и велика слова конвертује у мала ("StanilPomirisiOveRuze." даје "Stani i pomirisi ove ruze.")
- 056 Написати програм који открива да ли се у стрингу налази екстензија за формате фајлова (.exe, .com, .bat).

Речници и сетови

Речници - структура

У Пајтону, речник (dictionary) је објекат који садржи колекцију података.

Сваки елемент који је смештен у речник се састоји из два дела: од кључа (key) и од вредности (value).

Зато се елементи речника често називају и парови кључ-вредност (key-value pairs).

Постојање оваквих парова се често назива и мапирање јер је сваки кључ мапиран на одређену вредност.

Да би се добила одређена вредност из речника, користи се кључ који је придружен траженој вредности.

Ово је слично са процесом претраге тражене речи у класичном речнику, где су тражене речи кључеви а дефиниције тражених речи су вредности.

Креирање речника

Речник се креира убацивањем елемената унутар витичастих заграда.

Сваки елемент се састоји од кључа, двотачке и вредности, при чему се сваки следећи елемент одваја зарезом.

0160 Креирање речника

```
telefonski_imenik = {"Marko":'345-9876', 'Ana':"555-6666"}
```

Овај исказ креира речник и додељује га промењивој `telefonski_imenik`.

Кључеви у овом речнику су "Marko" и 'Ana'; док су вредности '345-9876' и "555-6666".

Вредности у речницима могу бити било којег типа података.

Кључеви у речницима могу бити само непромењивог типа података: стрингови, целобројне вредности, децималне вредности или торке.

Добијање вредности из речника

Елементи у речнику нису поређани по некаквом задатом редоследу.

Редослед којим су елементи приказани може да се разликује од редоследа којим су елементи убацивани у речник.

То значи да речници нису секвенце попут листа, торки или стрингова.

Зато се не може користити бројчани индекс као позиција елемента у речнику за добијање вредности.

Да би се добила вредност из речника, пише се:

```
naziv_recnika[kljuc]
```

Где је `naziv_recnika` промењива која указује на речник, док је `kljuc` баш кључ из речника.

Ако написани кључ постоји у речнику, дати израз враћа вредност која је додељена кључу.

Ако написани кључ не постоји у речнику, јавља се `KeyError` грешка.

Креирање празног речника : `recnik = {}`

Добијање броја елемената у речнику

Употребом предефинисане функције `len` може се добити број елемената у речнику:

0161 Приказ дужине речника

```
telefonski_imenik = {"Marko":'345-9876', 'Ana':"555-6666", "Zoran":"111-0000"}
```

```
x = len(telefonski_imenik)
```

```
print("Recnik", telefonski_imenik, "je duzine", str(x))
```

Даје: `Recnik {"Marko": '345-9876', 'Ana': "555-6666", "Zoran": "111-0000"}` је дужине 3

Употреба различитих типова података у речнику

Кључ мора бити непромењивог типа података док вредност може бити било који тип података. То значи да као вредност могу се користити и листе:

0162 Употреба листа у речницима

```
test_rezultati = { 'Miki' : [88, 92, 100], 'Ana' : [95, 74, 81], \
    'Dragan' : [72, 88, 91], 'Zorana' : [70, 75, 78] }
print(test_rezultati)
print(test_rezultati['Dragan'])
x = test_rezultati['Dragan']
print(x)
```

Вредности смештене у речник могу бити различитих типова података:

```
x = { 'abc':1, 999:'sretan rođendan', (3, 6, 9):[3, 6, 9]}
```

У датом примеру се види да се као кључ користе стринг, целобројна вредност и торка; док као вредност се користе целобројна вредност, стринг и листа.

Оваква особина речника се може искористити за много практичније потребе као што је креирање речника који садржи различите податке о запосленом у компанији:

```
zaposleni = { 'ime':"Dragana", 'id':12345, 'plata':66450.89 }
```

У датом примеру је креиран речник који је додељен промењивој `zaposleni`.

Речник се састоји од три елемента са кључевима типа стринг и са вредностима који су стринг, целобројни и децимални тип података.

Употреба празног речника

Добра пракса је креирање празног речника који се касније попуњава са елементима:

0163 Употреба празног речника

```
recnik = {}
recnik["Dragan"] = 'student'
recnik["Ana"] = "doktor"
print(recnik)
Даје: {"Dragan": 'student', "Ana": "doktor" }
```

Такође се може користити уграђена функција `dict()` која креира празан речник: `recnik = dict()`

Итерације у речнику

У речнику се може користити петља `for` за итерацију преко свих кључева у речнику у следећем формату:

```
for promenjiva in naziv_recnika:
    iskaz
```

iskaz

...

Овако петља итерира једном преко сваког од елемената у речнику, тако што сваки пут када петља итерира, променљива добија следећи кључ из речника.

0164 Итерација преко кључева у речнику

```
test_rezultati = { 'Miki' : [88, 92, 100], 'Ana' : [95, 74, 81], \
                  'Dragan' : [72, 88, 91], 'Zorana' : [70, 75, 78] }
```

```
for x in test_rezultati:
    print(x)
```

Променљива x добија кључеве из речника и приказују се само кључеви.

Оператори in и not in

За програм није добро да се прекине његово извршавање ако се покушало добити вредност у речнику помоћу кључа који не постоји.

Да би се то спречило, може се користити оператор in за испитивање да ли кључ уопште постоји у речнику пре него се покуша добити вредност повезана са тим кључем.

0165 Претрага речника без подизања изузетака

```
telefonski_imenik = {"Marko":'345-9876', 'Ana':"555-6666", "Zoran":"111-0000", \
                    "Dragan":"987-6543"}
if "Dragan" in telefonski_imenik:
    print(telefonski_imenik["Dragan"])
if "Dragan" not in telefonski_imenik:
    print("Кључ 'Dragan' не постоји у речнику.")
```

Додавање елемената у постојећи речник

Речници су променљиви објекти.

Може се увек додати нови пар кључ-вредност у речник употребом следеће синтаксе:

```
naziv_recnika[kljuc] = vrednost
```

где је naziv_recnika променљива која указује на речник, а kljuc је кључ из речника.

Ако kljuc већ постоји у речнику, вредност тог кључа постаје vrednost.

Ако kljuc не постоји у речнику, он ће се додати у речник заједно са vrednost која постаје његова придружена вредност у речнику.

0166 Додавање елемента у речник

```
telefonski_imenik = {"Marko":'345-9876', 'Ana':"555-6666", "Zoran":"111-0000", \
                    "Dragan":"987-6543"}
telefonski_imenik["Jovana"] = "012-3210"
print(telefonski_imenik)
```



```
telefonski_imenik ["Marko"] = '345-9876'  
print(telefonski_imenik)  
Дaje telefonski_imenik = {"Marko": '345-9876', 'Ana': "555-6666",  
"Zoran": "111-0000", "Dragan": "987-6543", "Jovana" : "012-3210"}
```

Овакав резултат значи да је немогуће дуплицирати кључ у речнику и да ако се додели вредност постојећем кључу, нова вредност замењује постојећу.

Брисање елемента из речника

Коришћењем исказа `del` може се обрисати пар кључ-вредност из речника.

```
del naziv_recnika[kljuc]
```

Ако се покуша обрисати непостојећи кључ из речника, појављује се `KeyError` грешка.

Метода `clear`

Ова метода одстрањује све елементе из речника остављајући речник празан:

```
naziv_recnika.clear()
```

0167 Употреба методе `clear()`

```
telefonski_imenik = {'Ana': '555-1111', 'Dragan': '555-2222'}  
print(telefonski_imenik)  
telefonski_imenik.clear()  
print(telefonski_imenik)  
{ 'Ana': '555-1111', 'Dragan': '555-2222' }  
{ }
```

Метода `get`

Ова метода је алтернатива за оператор `[]` при добијању вредности из речника.

Овај метод не узрокује појаву грешке ако не постоји тражени кључ у речнику.

Формат методе је:

```
naziv_recnika.get(kljuc, difolt)
```

У овом формату, `naziv_recnika` је назив речника, `kljuc` је кључ кји се тражи у речнику, `difolt` је дифолтна вредност која се враћа ако `kljuc` није пронађен у речнику.

Ако постоји `kljuc` у речнику, враћа се додељена вредност том кључу.

0168 Употреба методе `get()`

```
telefonski_imenik = {'Ana': '555-1111', 'Dragan': '555-2222'}  
vrednost = telefonski_imenik.get('Ana', 'Uneti kljuc nije pronadjen')  
print(vrednost)  
vrednost = telefonski_imenik.get('Mica', 'Uneti kljuc nije pronadjen')  
print(vrednost)  
555-1111
```

```
Uneti kljuc nije pronadjen
```

Метода items

Ова метода враћа све кључеве из речника заједно са придруженим вредностима. Они се враћају као посебан тип секвенце која се назива поглед речника (dictionary view).

0169 Употреба dictionary view

```
telefonski_imenik = {'Ana': '555-1111', 'Dragan': '555-2222'}  
print(telefonski_imenik.items())
```

Даје:

```
dict_items([('Ana', '555-1111'), ('Dragan', '555-2222')])
```

Види се да се добија секвенца која се састоји од две торке: ("Ana", "555-1111") и ("Dragan", "555-2222").

Може се користити петља за итерацију преко ових торки у секвенци:

0170 Употреба итерације за приступ торкама

```
telefonski_imenik = {'Ana': '555-1111', 'Dragan': '555-2222'}  
for kljuc, vrednost in telefonski_imenik.items():  
    print(kljuc, vrednost)
```

Ana 555-1111

Dragan 555-2222

Сада петља позива items() методу, која враћа секвенцу торки које садрже парове кључ-вредност из речника.

Петља итерира једном за сваку торку из секвенце, и вредности из торки се додељују у kljuc и vrednost променљиве.

Метода keys

Ова метода враћа све кључеве из речника као поглед речника, што је врста секвенце. Сваки елемент у погледу речника је кључ у речнику.

0171 Употреба методе keys()

```
telefonski_imenik = {'Ana': '555-1111', 'Dragan': '555-2222'}  
print(telefonski_imenik.keys())
```

Даје: dict_keys(['Ana', 'Dragan'])

Следећи пример приказује употребу итерације преко секвенце која се добија употребом keys методе.

0172 Употреба итерације са методом keys()

```
telefonski_imenik = {'Ana': '555-1111', 'Dragan': '555-2222'}  
for kljuc in telefonski_imenik.keys():  
    print(kljuc)
```

Даје:
Ana
Dragan

Метода pop

Ова метода враћа вредност која је придружена одређеном кључу и одстрањује кључ-вредност пар из речника.

Ако кључ није пронађен, метода враћа дефолт вредност.

Општи формат методе:

```
naziv_recnika.pop(kljuc, difolt)
```

Овде је kljuc кључ који се тражи у речнику, difolt је вредност која ће се вратити ако се кључ не пронађе.

0173 Употреба методе pop()

```
telefonski_imenik = {'Ana': '555-1111', 'Dragan': '555-2222'}  
broj_telefona = telefonski_imenik.pop('Ana', 'Uneti kljuc nije pronadjen')  
print(broj_telefona)  
print(telefonski_imenik)  
broj_telefona = telefonski_imenik.pop('Milan', 'Uneti kljuc nije  
pronadjen')  
print(broj_telefona)  
print(telefonski_imenik)  
555-1111  
{'Dragan': '555-2222'}  
Uneti kljuc nije pronadjen  
{'Dragan': '555-2222'}
```

Метода pop прво прослеђује 'Ana' као кључ који се тражи; вредност која је повезана са кључем 'Ana' се враћа и додељује променљивој broj_telefona; пар кључ-вредност који садржи кључ 'Ana' се одстрањује из речника.

Када се други пут позива метода pop, прослеђује се 'Milan' као кључ који се тражи; али пошто се кључ није пронашао у речнику, стринг 'Uneti kljuc nije pronadjen' је додељен променљивој broj_telefona.

Метода popitem

Ова метода враћа случајно изабран пар кључ-вредност и одстрањује тај пар кључ-вредност из речника.

Враћени пар кључ-вредност је у виду торке.

Формат методе:

```
naziv_recnika.popitem()
```

Могу се користити искази доделе у следећем формату да би се доделио променљивима враћени кључ и вредност:

```
k, v = naziv_recnika.popitem()
```

Овакав тип доделе вредности се назива вишеструка додела (multiple assignment) пошто се истовремено вредности додељују више од једне променљиве.

0174 Употреба методе popitem()

```
telefonski_imenik = {'Ana': '555-1111', 'Dragan': '555-2222'}
print(telefonski_imenik)
kljuc, vrednost = telefonski_imenik.popitem()
print(kljuc, vrednost)
print(telefonski_imenik)
{'Ana': '555-1111', 'Dragan': '555-2222'}
Dragan 555-2222
{'Ana': '555-1111'}
```

Метода popitem() доводи до KeyError грешке ако се позове на празном речнику.

Метода values

Ова метода враћа све вредности из речника (без кључева из речника) у облику погледа речника. Сваки елемент у погледу речника је вредност из речника.

0175 Употреба методе values()

```
telefonski_imenik = {'Ana': '555-1111', 'Dragan': '555-2222'}
for x in telefonski_imenik.values():
    print(x)
Даје:
555-1111
555-2222
```

Креирање сета

Сет је објекат који садржи елементе који се не могу понављати; елементи у сетовима су неуређеног редоследа и могу бити различитих типова података.

Да би се креирао један сет, користи се уграђена функција set(): `moj_set = set()`

После реализације овог кода, променљива `moj_set` указује на празан сет.

У загради као аргумент функције set() се може појавити један аргумент.

Тај аргумент мора бити објекат који садржи итерабилне елементе, попут листе, торке или стринга.

Појединачни елементи објекта који се придодаје као аргумент постају елементи сета.

```
moj_set = set(['a', 'b', 'c'])
```

Овако се придодаје листа у виду аргумента функције set.

После ове линије кода, променљива `moj_set` указује на сет који садржи елементе 'a', 'b' и 'c'.

Ако се придода стринг као аргумент функције set, сваки појединачни знак у стрингу постаје елемент сета:

```
moj_set = set('abc')
```

После реализације линије кода, променљива `moj_set` ће указати на сет који се састоји од три елемента: 'a', 'b' и 'c'.

Ако се покуша креирати сет са више истих елемената, само први од истих елемената ће постати елемент сета:

```
moj_set = set('aaaabc') #daje set koji se sastoji od 'a', 'b' i 'c'
```

Потребно је креирати сет који се састоји од елемената који су стрингови са једним или више знакова:

```
moj_set = set(['jedan', 'dva', 'tri']) #daje set koji se sastoji od elemenata 'jedan', 'dva', 'tri'
```

Добијање броја елемената у сету

```
moj_set = set([1, 2, 3, 4, 5])  
print(len(moj_set)) #daje 5.
```

Додавање елемената у сет

Сетови су промењиви објекти па је зато допуштено додавати и одстрањивати елементе из сетова. За додавање се користи метода add.

0176 Употреба методе add()

```
moj_set = set()  
moj_set.add(1)  
moj_set.add(2)  
moj_set.add(3)  
print(moj_set)  
moj_set.add(2)  
print(moj_set)
```

Дaje:

```
{1, 2, 3}
```

```
{1, 2, 3}
```

Види се да после покушаја додавања истог елемента у сет (број 2) тај елемент није додат и није дошло до појаве грешке, већ се игнорише метода add.

Коришћењем методе update, може се истовремено додати група елемената у сет:

0177 Употреба методе update са целим сетовима

```
set1 = set([1, 2, 3])  
set2 = set([8, 9, 10])  
set1.update(set2)  
print(set1)  
print(set2)
```

Дaje:

```
{1, 2, 3, 8, 9, 10}
```

```
{8, 9, 10}
```

0178 Употреба методе update са различитим типовима података

```
moj_set = set([1, 2, 3])  
moj_set.update('abc')
```

```
print(moj_set)
Даје:
{1, 2, 3, 'a', 'b', 'c'}
```

Одстрањивање елемената из сета

Да би се одстранио елемент из сета може се користити или метода `remove` или метода `discard`. Разлика је у томе како се методе понашају ако се затражи одстрањивање елемента који не постоји у сету: метода `remove` подиже `KeyError` грешку док метода `discard` не подиже грешку.

0179 Методе одстрањивања елемената из сета

```
moj_set = set([1, 2, 3, 4, 5])
print(moj_set)
moj_set.remove(1)
print(moj_set)
moj_set.discard(5)
print(moj_set)
moj_set.discard(99)
moj_set.remove(99)
{1, 2, 3, 4, 5}
{2, 3, 4, 5}
{2, 3, 4}
Traceback (most recent call last):
  File "C:\Users\nera\Desktop\proba\PythonApplication2\PythonApplication2.py", line 12, in <module>
    moj_set.remove(99)
KeyError: 99
```

Сви елементи у сету се могу одстранити позивањем методе `clear`.

0180 Метода `clear()`

```
moj_set = set([1, 2, 3, 4, 5])
print(moj_set)
moj_set.clear()
print(moj_set)
```

```
Даје:
{1, 2, 3, 4, 5}
set() #prazan set
```

Коришћење петље за итерацију преко сета

0181 Итерација преко сета

```
moj_set = set(['a', 'b', 'c'])
for x in moj_set:
    print(x)
```

Петља итерира једном за сваки елемент из сета.

```
Даје:
a
b
c
```

Коришћење оператора in и not in за тестирање вредности у сету

0182 Употреба оператора

```
moj_set = set([1, 2, 3])
if 1 in moj_set:
    print("Element 1 se nalazi u setu.")
if 99 not in moj_set:
    print("Element 99 se ne nalazi u setu.")
```

Добијање уније сетова

Унија два сета је нов сет који садржи све елементе оба сета.

У Пајтону се позива метода union или се користи оператор | за добијање уније два сета.

```
set1.union(set2)          set1 | set2
```

0183 Унија сетова

```
set1 = set([1, 2, 3])
set2 = set([10, 20, 30])
set3 = set([4, 5, 6])
set4 = set1.union(set3)
set5 = set2 | set1
print(set4)
print(set5)
```

Даје:

```
{1, 2, 3, 4, 5, 6}
{1, 2, 3, 10, 20, 30}
```

У првом делу примера позива се метод union објекта set1 који има за аргумент објекат set3.

У другом делу примера користи се бинарни оператор | који над два објекта set2 и set1 реализује операцију уније сетова.

Добијање пресека сетова

Пресек (intersection) два сета је сет који садржи само оне елементе који се налазе у оба сета.

У Пајтону се за добијање пресека може користити метод intersection или оператор &:

```
set1.intersection(set2)    set1 & set2
```

0184 Пресек сетова

```
set1 = set([1, 2, 10])
set2 = set([10, 20, 5])
set3 = set([4, 5, 6])
set4 = set1.intersection(set2)
set5 = set1 & set3
print(set4)
print(set5)
```

Даје:
{10}
set()

Добијање разлике сетова

Разлика (difference) између објеката set1 и set2 су елементи који се појављују у set1 али се не појављују у set2 објекту.

У Пајтону се за добијање разлике може користити метод difference или оператор -:

set1.difference(set2) set1 - set2

0185 Разлика сетова

```
set1 = set([1, 2, 3, 4, 5])
set2 = set([2, 4, 6, 8])
set3 = set([6, 7, 8, 9])
set4 = set1.difference(set2)
set5 = set2 - set3
print(set4)
print(set5)
```

Даје:
{1, 3, 5}
{2, 4}

Добијање симетричне разлике сетова

Симетрична разлика (symmetric difference) између два сета јесте сет који садржи елементе који нису заједнички за та два сета.

У Пајтону се за добијање симетричне разлике може користити метод symmetric_difference или оператор ^:

0186 Симетрична разлика сетова

```
set1 = set([1, 2, 3, 4, 5])
set2 = set([2, 4, 6, 8])
set3 = set([6, 7, 8, 9])
set4 = set1.symmetric_difference(set2)
set5 = set2 ^ set3
print(set4)
print(set5)
```

Даје:
{1, 3, 5, 6, 8}
{2, 4, 7, 9}

Добијање субсета и суперсета

Ако постоје set1 и set2 а set1 садржи све елементе који припадају set2, онда се каже да је set2 субсет (подсет) од set1.

Такође, може се рећи да је у том случају set1 суперсет (надсет) од set2.

0187 Субсет и суперсет

```
set1 = set([1, 2, 3, 4, 5])
set2 = set([2, 4])
set3 = set([6, 7, 8, 9])
print("set1 =", set1)
print("set2 =", set2)
print("set3 =", set3)
print("Da li je set2 podset od set1?", set2.issubset(set1))
print("Da li je set1 superset od set2?", set1.issuperset(set2))
print("Da li je set2 podset od set3?", set2.issubset(set3))
print("Da li je set3 superset od set2?", set3.issuperset(set2))
```

```
set1 = {1, 2, 3, 4, 5}
set2 = {2, 4}
set3 = {8, 9, 6, 7}
Da li je set2 podset od set1? True
Da li je set1 superset od set2? True
Da li je set2 podset od set3? False
Da li je set3 superset od set2? False
```

Задаци са речницима и сетовима

0188 Креирање квиза са речницима

Написати програм који користи речник за креирање квиза о главним градовима европских држава.

```
glavni_gradovi = {"Srbije" : "Beograd", "Grcke" : "Atina", "Bugarske" :
"Sofija", "Rumunije" : "Bukurest"}
gotovo = False
print("Kviz glavnih gradova Evrope")
while not gotovo and len(glavni_gradovi) > 0:
    zemlja, glavni_grad = glavni_gradovi.popitem()
    print("Koji je glavni grad", zemlja + "? Pritisni d za kraj.")
    odgovor = input("Uneti odgovor: ")
    if odgovor.lower() == "d":
        gotovo = True
    elif odgovor == glavni_grad:
        print("\nTacno!\n")
    else:
        print("\nIzvini. Glavni grad", zemlja, "je", glavni_grad + "\n")
if len(glavni_gradovi) == 0:
    print("Gotovo.")
```

0189 Телефонски именик са речницима са уносом

Написати програм који користи речнике за креирање телефонског именика са именима и телефонима познаника. Ако корисник унесе особу која није у именику, омогућити унос телефона те особе.

```
telefonski_imenik = {"Marko":'345-9876', 'Ana':"555-6666", "Zoran":"111-0000",\
"Dragan":"987-6543"}
ime = input("Uneti ime osobe: ")
if ime in telefonski_imenik:
    print("Ova osoba vec postoji u telefonskom imeniku")
else:
    print("Unesi telefon ", ime, ":", end = "")
    x = input()
    telefonski_imenik[ime] = x
print(telefonski_imenik)
```

0190 Телефонски именик са речницима са брисањем

Написати програм који користи речнике за креирање телефонског именика са именима и телефонима познаника. Омогућити брисање жељене особе из телефонског именика.

```
telefonski_imenik = {"Marko":'345-9876', 'Ana':"555-6666", "Zoran":"111-0000",\
"Dragan":"987-6543"}
ime = input("Uneti ime osobe koja se brise: ")
if ime not in telefonski_imenik:
    print("Ova osoba ne postoji u telefonskom imeniku")
else:
    broj = telefonski_imenik[ime]
    print("Iz imenika je odstranjena osoba", ime, "sa brojem", broj)
    del telefonski_imenik[ime]
print(telefonski_imenik)
```

Питања и задаци за самосталан рад

Задаци

057 Написати програм који користи речнике за креирање телефонског именика са именима и телефонима познаника. Омогућити да свака особа може имати два различита броја телефона.

058 Написати програм који користи речнике за креирање списка људи и њихових датума рођења. Апликација треба да има опције за унос нове особе са телефонским бројем у именик, приказ броја телефона према имену особе, поруко о непостојећој особи у именику, брисање постојеће особе из именика, унос другог телефонског броја за постојећу особу у именику.

0191 Телефонски именик са речницима са изменама

Написати програм који користи речнике за креирање телефонског именика са именима и телефонима познаника. Омогућити измену броја телефона жељене особе из телефонског именика.

```
telefonski_imenik = {"Marko": '345-9876', 'Ana': "555-6666", "Zoran": "111-0000", \
"Dragan": "987-6543"}
ime = input("Uneti ime osobe ciji se broj telefon menja: ")
if ime not in telefonski_imenik:
    print("Ova osoba ne postoji u telefonskom imeniku")
else:
    broj = input("Uneti novi broj telefona:")
    print("Osoba", ime, "ima nov broj:", broj)
    telefonski_imenik[ime] = broj
print(telefonski_imenik)
```

0192 Употреба речника за конверзију у шифру

Написати програм који користи речник за конверзију стринга у бројчану шифру.

```
def main():
    recnik = {"a" : "0", "b" : "1", "c" : "2", "d" : "3", \
             "e" : "4", "i" : "5", "o" : "6", "r" : "7", "u" : "8"}
    rec = unos_stringa()
    sifra = konverzija_reci(rec, recnik)
    print(sifra)

def unos_stringa():
    r = input("Uneti rec: ")
    return r

def konverzija_reci(r, r_nik):
    s = ""
    for x in r:
        if x in r_nik.keys():
            s += r_nik[x]
        else:
            s += x
    return s

main()
```

0193 Употреба речника за конверзију у стринг

Написати програм који користи речник за конверзију бројчане шифре у стринг.

```
def main():
    recnik = {"a" : "0", "b" : "1", "c" : "2", "d" : "3", \
             "e" : "4", "i" : "5", "o" : "6", "r" : "7", "u" : "8"}
    sifra = unos_sifre()
```

```

    rec = konverzija_sifre_u_rec(sifra, recnik)
    print(rec)

def unos_sifre():
    s = input("Uneti sifru: ")
    return s

def konverzija_sifre_u_rec(s, r_nik):
    r = ""
    for x in s:
        for kljuc, vrednost in r_nik.items():
            if x == vrednost:
                r += kljuc

    return r

main()

```

Питања и задаци за самосталан рад

Задаци

059 Написати програм који користи речник за проверу исправности добијене бројчане шифре.

060 Написати програм који користи речнике за дељење 5 насумичних карата из шпила од 52 карте.

0194 Конверзија стринга у сет

Написати програм који прима стринг, речи из стринга претвара у елементе сета и открива да ли се у сету појављује назив воћа (јабука, крушка, манго).

```

while 1:
    recenica = input("Unesi string: ")
    rec = ""
    skup1 = set()
    for i in recenica:
        if i != " ":
            rec += i
        else:
            if rec == "jabuka" or rec == "kruska" or rec == "mango":
                skup1.add(rec)
            rec = ""
            if rec == "jabuka" or rec == "kruska" or rec == "mango":
                skup1.add(rec)
    print("U stringu se nalazi voce:", skup1)

```

0195 Употреба сетова

Проверити да ли се у стрингу појавило воће (јабука, крушка, манго) а затим приказати које воће недостаје.

```
while 1:
    recenica = input("Unesi string: ")
    rec = ""
    skup1 = set()
    skup2 = set(["kruska", "jabuka", "mango"])
    for i in recenica:
        if i != " ":
            rec += i
        else:
            if rec == "jabuka" or rec == "kruska" or rec == "mango":
                skup1.add(rec)
            rec = ""
            if rec == "jabuka" or rec == "kruska" or rec == "mango":
                skup1.add(rec)

    print("U stringu se nalazi voce:", skup1)
    print("Setu sa vocem nedostaje: ")
    for i in skup2.difference(skup1):
        print(i)
```

Питања и задаци за самосталан рад

Задаци

- 061 Написати програм који користи сетове да би открио имена ученика који тренирају фудбал а не и кошарку, кошарку а не и фудбал, и фудбал и кошарку.
- 062 Написати програм који користи сетове да би открио који су заједнички бројеви за три скупа бројева а који су бројеви који припадају само појединачним скуповима.
- 063 Открити колико парних и непарних целих бројева, у датом опсегу, припада скупу бројева дељивих са 5, употребом сетова.

Увод у рад са датотекама

Увод у улазне и излазне фајлове

Рад са едитором кода у програмским језицима подразумева употребу RAM меморије која је непостојана, обрисива меморија, која не може неограничено чувати једном унети податак.

Да би се сачували једном унети подаци, они се морају чувати у фајловима, који се обично смештају на компјутерском хард диску.

На тај начин се унети подаци неће неповратно обрисати већ се могу неограничени број пута позивати и користити.

Програмери називају процес чувања података у фајловима „уписивање података“ (writing data) у фајл.

Током уписивања податак у фајл, подаци се копирају из променљиве у RAM меморији у фајл.

Израз излазни фајл (output file) се користи за опис фајла у који се уписују подаци (програм смешта своје излазне резултате у излазни фајл).

Процес добијања података из фајла се назива „читање података“ (reading data) из фајла.

Када се прочита податак из фајла, он се копира у RAM меморију и на њега упућује промењива.

Израз улазни фајл (input file) се користи за опис фајла из којег се читају подаци (програм добија улазне податке из улазног фајла).

Кораци у коришћењу фајлова

Увек постоје три корака који се морају спровести да би се фајл могао користити у програму:

1. Отварање фајла – отварање фајла креира везу између фајла и програма; отварање излазног фајла обично креира фајл на диску и омогућава програму да уписује податке у фајлу; отварање улазног фајла омогућава програму да прочита податке из фајла
2. Рад са фајлом – податак се или уписује у фајл (ако је у питању излазни фајл) или чита из фајла (ако је у питању улазни фајл)
3. Затварање фајла – када програм заврши са употребом фајла, фајл мора да се затвори; затварање фајла раскида везу фајла и програма

Типови фајлова

Постоје два типа фајлова: текстуални и бинарни.

Текст (text) фајлови садрже податке који су кодовани у виду текста, коришћењем кодова као што су ASCII или Unicode.

Чак иако фајл користи бројеве, ови бројеви се смештају у фајл као низ карактера.

То резултује да фајл се може отворити и прегледати у текст едитору као што је Notepad.

Бинарни (binary) фајл садржи податке који нису конвертовани у текст.

Подаци који су смештени у бинарном фајлу су форматирани да их користи само неки специфични програм.

Зато, њихов садржај се не може прегледати помоћу текст едитора.

Методe за приступ фајловима

Програмски језици углавном користе два различита начина за приступ подацима у фајловима: секвенцијални приступ и директан приступ.

Када се ради са секвенцијалним приступом фајловима (sequential access file) приступа се подацима од почетка фајла до краја фајла.

Ако је потребно читати податак који је смештен на самом крају фајла, прво се морају прочитати сви подаци који су пре њега у фајлу, тј не може се директно скочити на жељени податак (приступ песми на аудио касети).

Када се ради са директним приступом фајловима (direct access file, random access file) може се директно прићи било којем податку у фајлу без обзира где се налази (приступ подацима на аудио CD-диску).

Фајлови као објекти

Фајлови се идентификују преко њиховог имена (filename).

Савремени оперативни системи идентификују фајлове и преко слике фајла (икона).

Оперативни системи имају специфичне начине и правила за именовање фајлова.

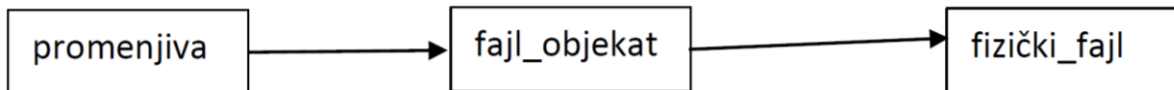
Многи оперативни системи користе екстензије имена фајлова (filename extensions) које су кратке секвенце знакова који се појављују на крају имена фајла а после тачке (dot) (.jpeg, .txt, .docx).

Ове екстензије обично указују на тип података који је смештен у фајлу (.jpeg указује да је у питању графичка слика која је компресована према JPEG стандарду).

Да би програм радио са фајловима на хард диску, програм мора креирати фајл објекат у меморији.

Фајл објекат је објекат који је придружен посебном фајлу и омогућава да програм ради са тим фајлом.

У програму, променљива указује на фајл објекат и користи се за извођење свих операција над тим фајлом.



Отварање фајла

Да би се отворио фајл, користи се функција open.

Функција open креира **фајл објекат** и придружује га са физичким фајлом на диску.

Општи формат коришћења функције open:

```
promenjliva = open(ime_fajla, mod)
```

Овде је promenjliva име променљиве која указује на фајл објекат.

Са модом се одређује начин рада и употребе фајла приликом његовог отварања.

Најчешће коришћени модови су:

‘r’ отвара се фајл само за читање; фајл се не може променити нити се у фајл може било шта уписати

‘w’ отвара се фајл само за уписивање; ако фајл постоји, брише се њен садржај; ако фајл не постоји, креира се

‘a’ отвара се фајл за уписивање; сви подаци уписани у фајл ће бити придодати на крај фајла; ако фајл не постоји, креира га

0196 Отварање фајла за читање

фајл musterije.txt садржи податке о муштеријама и потребно је да се отвори за читање

```
musterije_fajl = open('musterije.txt', 'r')
```

Када се овај исказ реализује, фајл под именом musterije.txt ће бити отворен а променљива musterije_fajl ће указивати на фајл објекат који се може користити за читање података из фајла.

0197 Отварање фајла за упис

нека је потребно креирати фајл са именом prodaja.txt и нека је потребно уписати податке у тај фајл

```
prodaja_fajl = open('prodaja.txt', 'w')
```

Када се овај исказ изврши, фајл под именом prodaja.txt ће бити креиран а променљива prodaja_fajl ће указивати на фајл објекат који ће се користити за упис података у фајл.

Одређивање локације фајла

Ако се у раду са фајлом користи име фајла у којем нема путање (path) као аргумента функције open, Пајтон интерпретер претпоставља да је локација фајла иста као за сам програм.

0198 Рад са фајлом на другој локацији

Ако се програм стартује и реализује следећи исказ, фајл test.txt је креиран у истом фолдеру:

```
test_fajl = open('test.txt', 'w')
```

Ако треба отворити фајл на другој локацији, може се приказати и путања и име фајла као аргументи који се придружују функцији open.

Ако се упише путања у стринг литералу, треба осигурати да префикс стринга буде са словом r:

```
test_fajl = open(r'C:\Users\temp\test.txt', 'w')
```

Овај исказ креира фајл test.file у фолдеру 'C:\Users\temp\test.txt'.

Префикс r одређује да је стринг заправо сирови стринг (raw string).

То чини да Пајтон интерпретер чита знак косе црте (backslash) као литерал.

Без r интерпретер ће претпоставити да знак косе црте је део ескејп секвенце и појавиће се грешка.

Упис података у фајлове

Методе су функције које припадају објекту и изводе неке операције користећи тај објекат.

Када се једном отвори фајл, користе се методе тог фајла објекта за извођење операција над фајловима.

Нпр, фајл објекти имају методу write која се користи за упис података у фајл.

Општи формат за позив методе write:

```
fajl_promenjiva.write(string)
```

У датом формату, fajl_promenjiva је промењива која указује на фајл објекат, док је string само стринг који ће бити уписан у фајл.

Фајл мора претходно бити отворен за упис (коришћењем 'w' или 'a' мода) или ће се појавити грешка.

Нека musterija_fajl указује на фајл објекат, а фајл је отворен за упис података са 'w' модом.

Ово је пример како би се стринг "Dragan Jovanovic" уписао у фајл:

```
musterija_fajl.write('Dragan Jovanovic')
```

Други начин уноса стринга у фајл:

```
ime = 'Dragan Jovanovic'
```

```
musterija_fajl.write(ime)
```

На овај начин исказ уписује вредност, која указује на ime промењиву, у фајл који је придружен са промењивом musterija_fajl.

У овом случају резултат ће бити упис стринга 'Dragan Jovanovic' у фајл (слично је и са нумеричким подацима).

Када програм заврши рад са фајлом, треба затворити фајл.

Затварањем фајла се раскида веза програма са фајлом.

У неким оперативним системима, ако нема затварања фајла после рада са фајловима, може доћи до озбиљног губљења података.

Ово се дешава пошто подаци који су написани у фајл се прво уписују у бафер (један меморијски део).

Када је бафер попуњен, оперативни систем уписује садржај бафера у фајл.

На овај начин се побољшава перформанса оперативног система, пошто упис података у оперативну меморију је брже него упис података на хард диск.

Процес затварања излазног фајла приморава све несачуване податке који преостају у баферу да буду уписани у фајл.

У Пајтону се користи метод `close` за затварање фајла.

0199 Затварање фајла

```
musterija_fajl.close()
```

На овај начин се затвара фајл који је повезан са промењивом `musterija_fajl`.

0200 Отварање, упис и затварање истог фајла

Написати програм који отвара излазни фајл, уписује податак у фајл а затим га и затвара

```
def main():
    izlazni_fajl = open('kosarkasi.txt', 'w')

    izlazni_fajl.write('Vlade Divac\n')
    izlazni_fajl.write('Aleksandar Djordjevic\n')
    izlazni_fajl.write('Dragan Kicanovic\n')

    izlazni_fajl.close()
```

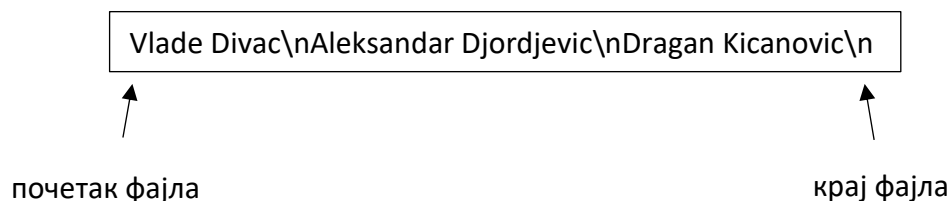
```
main()
```

У примеру, отвара се `kosarkasi.txt` фајл коришћењем 'w' мода(ово чини да се фајл креира и отвори са упис).

Такође се креира фајл објекат у меморији и додељује га промењивој `izlazni_fajl`.

Затим се уписују три стринга у фајл.

Када се програм заврши, три ствари ће бити уписане у `kosarkasi.txt` фајл.



Примећује се да се сваки стринг уписан у фајл завршава са `\n`, што је ескејп карактер за нови ред.

Овде `\n` само раздваја ствари које су у фајлу, али такође чини да се оне појаве у различитим линијама када се посматрају у текст едитору:

Vlade Divac
Aleksandar Djordjevic
Dragan Kicanovic

Читање података из фајла

Ако је фајл отворен за читање (употребом 'r' мода) може се користити метод read за читање садржаја фајла (или пренос садржаја фајла негде у меморију).

Када се позове метод read, он враћа садржај фајла у виду стринга.

0201 Отварање фајла за читање

```
def main():  
    #otvara fajl pod imenom kosarkasi.txt  
    ulazni_fajl = open('kosarkasi.txt', 'r')  
  
    sadrzaj_fajla = ulazni_fajl.read() #cita sadrzaj fajla  
    ulazni_fajl.close() #zatvara fajl  
  
    print(sadrzaj_fajla) #stampa podatke koji su ucitani u memoriju
```

main()

Код даје резултат:

Vlade Divac
Aleksandar Djordjevic
Dragan Kicanovic

Отварањем текстуалног фајла се истовремено креира фајл објекат и додељује промењивој ulazni_fajl.

Са методом read се чита садржај фајла као стринг и смешта у меморију рачунара.

Иако метод read омогућава лако читање целог садржаја фајла са само једним исказом, ефикасније је читати и обрадити део по део садржаја фајла.

Нпр, ако се фајл састоји од велике количине података у виду бројева, ефикасније је читати број по број као стринг, конвертовати га из стринга у број и извршити све потребне аритметичке операције за добијање резултата.

У Пајтону се користи метод readline за читање једне линије из фајла.

Линија је стринг карактера који се завршава са ескејп карактером \n.

0202 Читање фајла по линијама

```
def main():  
    ulazni_fajl = open('kosarkasi.txt', 'r')  
  
    linija1 = ulazni_fajl.readline()  
    linija2 = ulazni_fajl.readline()
```

```
linija3 = ulazni_fajl.readline()
ulazni_fajl.close()
```

```
print(linija1)
print(linija2)
print(linija3)
```

```
main()
```

Даје резултат:

Vlade Divac

Aleksandar Djordjevic

Dragan Kicanovic

Види се да после сваке одштампане линије се појављује празан ред; то је зато што се сваки учитани ред из фајла на крају реда састоји од ескејп карактера `\n`.

Када се фајл отвори за читање, посебна вредност позната као позиција читања (read position) се интерно додељује за тај фајл.

Позиција читања за фајл означава локацију следеће ствари која ће бити прочитана из фајла.

На почетку читања, позиција читања указује на ствар која је прва у фајлу.

Vlade Divac\nAleksandar Djordjevic\nDragan Kicanovic\n



позиција читања

Када се позове метод `readline`, чита се прва линија из фајла која се добија као стринг који се додељује променљивој `linija1` а то је `"Vlade Divac\n"`.

После тога, позиција читања се помера на почетак следеће линије у фајлу:

Vlade Divac\nAleksandar Djordjevic\nDragan Kicanovic\n



позиција читања

После читања свих линија у фајлу, позиција читања указује на крај фајла (више не указује ни на какав податак у фајлу).

Ако последња линија у фајлу се не завршава са `\n`, метода `readline` може вратити линију и без `\n`.

Надовезивање нове линије у стринг

Најчешће подаци у фајловима нису у оригиналном облику стринг литерали већ вредности у меморији на које су упућивале променљиве.

Тако се ради када се траже подаци од корисника који се касније смештају у фајлове.

Када програм уписује податке који су раније унети од стране корисника, најчешће је неопходно надовезати `\n` ескејп карактер на те податке пре него се они упишу у фајл. Тако се осигурава да сваки део података ће бити уписан у различиту линију у фајлу.

0203 Надовезивање линија

```
def main():
    print('Unesi imena tri prijatelja: ')
    ime1 = input('Prijatelj 1: ')
    ime2 = input('Prijatelj 2: ')
    ime3 = input('Prijatelj 3: ')

    moj_fajl = open('prijatelji.txt', 'w')

    moj_fajl.write(ime1 + '\n')
    moj_fajl.write(ime2 + '\n')
    moj_fajl.write(ime3 + '\n')

    moj_fajl.close()
    print('Imena prijatelja su upisana u prijatelji.txt fajl.')

main()
```

Дaje:

```
Unesi imena tri prijatelja:
Prijatelj 1: Miki
Prijatelj 2: Ana
Prijatelj 3: Darko
Imena prijatelja su upisana u prijatelji.txt fajl.
Сада ће у фајлу prijatelji.txt бити следећи садржај:
```

Miki\nAna\nDarko\n

Читање стринга и одстрањивање `\n` из стринга

Ескејп карактер `\n` има корисну сврху унутар фајла: раздваја ствари једне од других у фајлу. Понекад настају проблеми приликом читања стрингова из фајла коришћењем `readline` методе. Један од могућих проблема је појава празног реда после исписа сваког стринга из линије фајла. У Пајтону постоји метода `rstrip` која одстрањује одређене карактере са краја стринга.

0204 Употреба методе `rstrip()`

```
ime = 'Miki Maus\n'
ime = ime.rstrip('\n')
```

У примеру се прво додељује стринг `'Miki Maus\n'` променљивој `ime`. У следећој линији кода се позива метода `rstrip` која враћа копију садржаја променљиве `ime` (а то је стринг) али без одстрањеног знака `\n`. И такав промењен стринг је поново додељен променљивој `ime`.

0205 Употреба методе у уређивању стрингова

```
def main():
    ulazni_fajl = open('kosarkasi.txt', 'r')
    linija1 = ulazni_fajl.readline()
    linija2 = ulazni_fajl.readline()
    linija3 = ulazni_fajl.readline()
    linija1 = linija1.rstrip('\n')
    linija2 = linija2.rstrip('\n')
    linija3 = linija3.rstrip('\n')
    ulazni_fajl.close()
    print(linija1)
    print(linija2)
    print(linija3)

main()
```

Vlade Divac
Aleksandar Djordjevic
Dragan Kicanovic

Придодавање података у постојећи фајл

Када се користи 'w' мод за отварање излазног фајла а фајл са тим именом већ постоји на диску, постојећи фајл ће бити обрисан и нов празан фајл истог имена ће бити креиран.

Понекад је потребно сачувати стари фајл и придодати нове податке на већ постојећи садржај фајла.

Придодавање података у фајл значи уписивање нових података на крај постојећих података.

У Пајтону се може користити мод 'a' за отварање излазног фајла у моду придодавања (append mode), што значи:

- ако фајл већ постоји, неће бити избрисан; ако фајл не постоји, беће креиран
- када се подаци уписују у фајл, биће уписани на крају постојећег садржаја фајла

0206 Придодавање података у фајл

Нека у фајлу prijatelji.txt већ постоје уписана имена пријатеља:

Miki
Ana
Darko

Следећи код отвара фајл и придодаје податке у постојећи садржај:

```
moj_fajl = open('prijatelji.txt', 'a')
moj_fajl.write('Zoran\n')
moj_fajl.write('Milena\n')
moj_fajl.write('Jovan\n')
moj_fajl.close()
```

Добија се:

Miki

Ana

Darko

Zoran

Milena

Jovan

Упис и читање нумеричких података

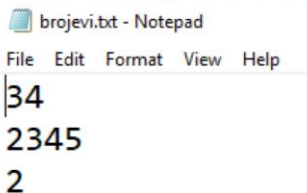
Стрингови могу бити уписани директно у фајл са методом write, али се прво бројеви морају конвертовати у стрингове пре уписа у фајл.

Коришћењем уграђене функције str, врши се конвертовање из бројног приказа у стринг.

0207 Конверзија бројног податка

```
def main():
    izlazni_fajl = open('brojevi.txt', 'w')
    broj1 = int(input('Unesi prvi broj: '))
    broj2 = int(input('Unesi drugi broj: '))
    broj3 = int(input('Unesi treci broj: '))
    izlazni_fajl.write(str(broj1) + '\n')
    izlazni_fajl.write(str(broj2) + '\n')
    izlazni_fajl.write(str(broj3) + '\n')
    izlazni_fajl.close()
    print('Podaci su upisani u brojevi.txt fajl.')
```

```
main()
Unesi prvi broj: 34
Unesi drugi broj: 2345
Unesi treci broj: 2
Podaci su upisani u brojevi.txt fajl.
```



Ако је потребно прочитати претходно унешене бројеве из фајла:

```
ulazni_fajl = open('brojevi.txt', 'r')
vrednost = ulazni_fajl.readline()
ulazni_fajl.close()
```

У коду се користи метода readline за читање линије из фајла.

Промењива vrednost ће указивати на прву прочитану линију у фајлу а то је '34\n'.

Ово може представљати проблем ако је била намера да радимо са бројчаним вредностима, па се зато мора извршити конверзија у неки од нумеричких типова податка:

```
ulazni_fajl = open('brojevi.txt', 'r')
string_ulaz = ulazni_fajl.readline()
vrednost = int(string_ulaz)
ulazni_fajl.close()
```

Сада промењива string_ulaz указује на прву линију у фајлу: '34\n'.

Затим у следећој линији се врши конверзија промењиве string_ulaz из стринга у целобројну вредност и резултат се додељује промењивој vrednost.

После овог исказа, промењива vrednost ће указивати на целобројну вредност 34 пошто функције int и float игноришу приликом конверзије ескејп карактере на крају стринга који им се додели као аргумент.

Бољи начин за читање стринга из фајла и његову конверзију:

```
ulazni_fajl = open('brojevi.txt', 'r')
vrednost = int(ulazni_fajl.readline())
ulazni_fajl.close()
```

У другој линији кода се позива метод readline као аргумент функције int.

При његовом позиву, метод readline враћа стринг, који се предаје функцији int која га конвертује у целобројну вредност и резултат конверзије се додељује промењивој vrednost.

0208 Конверзија током читања

```
def main():
    ulazni_fajl = open('brojevi.txt', 'r')
    broj1 = int(ulazni_fajl.readline())
    broj2 = int(ulazni_fajl.readline())
    broj3 = int(ulazni_fajl.readline())
    ulazni_fajl.close()
    suma = broj1 + broj2 + broj3
    print('Brojevi su:', broj1, broj2, broj3)
    print('Njihov zbir je:', suma)
```

```
main()
```

Коришћење петљи у раду са фајловима

Ретки су програми који користе фајлове за смештање малих количина података, обично су то огромне колекције података.

За рад са великим количинама података у фајловима, најчешће се користе петље.

0209 Употреба фајлова у трговини

Добијање количине продаје у три дана од стране корисника и уписивање те вредности у фајл prodaja.txt

```
def main():
    broj_dana = int(input('Koliko dana je trajala prodaja? '))
    prodaja_fajl = open('prodaja.txt', 'w')

    for i in range(1, broj_dana + 1):
```

```
prodaja = float(input('Uneti prodaju za dan #' + str(i) + ': '))
prodaja_fajl.write(str(prodaja) + '\n')
```

```
prodaja_fajl.close()
print('Podaci su upisani u prodaja.txt.')
```

```
main()
```

Koliko dana je trajala prodaja? 3

Uneti prodaju za dan #1: 30

Uneti prodaju za dan #2: 25

Uneti prodaju za dan #3: 45

Podaci su upisani u prodaja.txt.

prodaja.txt - Notepad

File Edit Format View Help

30.0

25.0

45.0

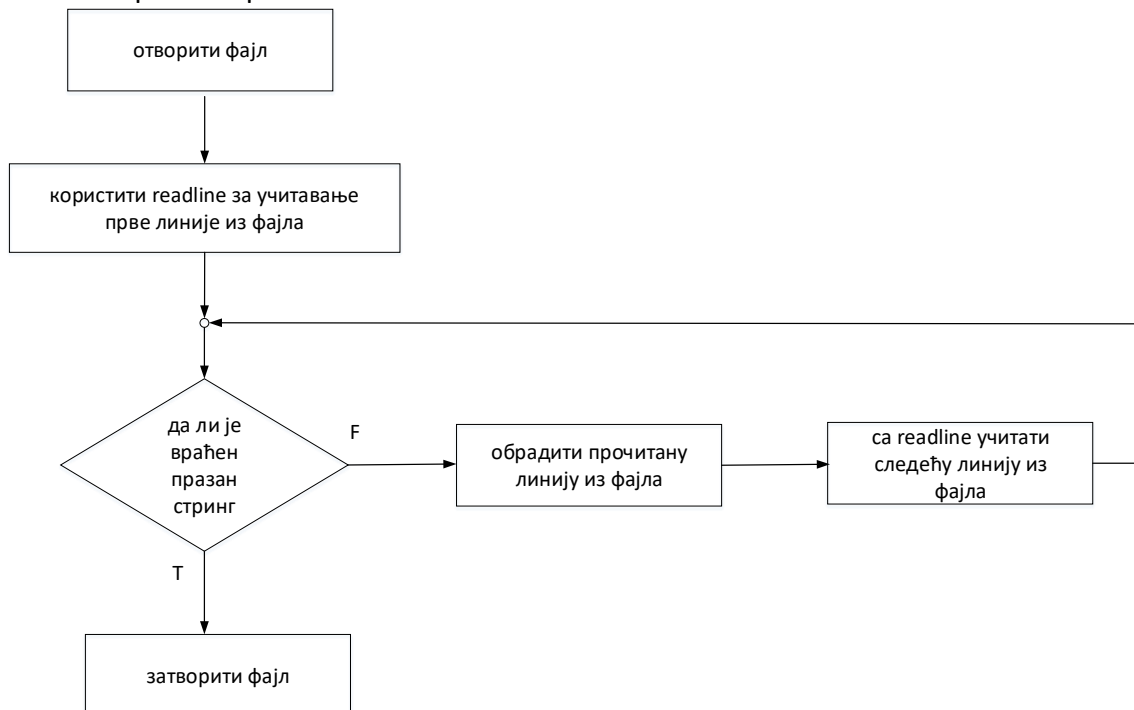
Читање фајла помоћу петље и детекција краја фајла

Често је потребно читати садржај фајла а да није познато колики је број података смештен у фајл.

Овде се поставља проблем детекције када се са читањем података дошло до краја фајла.

У Пајтону, метода `readline` враћа празан стринг (' ') када се покуша прочитати податак после последњег податка у фајлу.

Зато је могуће овде користити `while` петљу са испитивањем услова да ли је вредност враћена са `readline` празан стринг.



0210 Читање са детекцијом краја линије


```
def main():
    prodaja_fajl = open('prodaja.txt', 'r')
    linija = prodaja_fajl.readline()
    while linija != '':
        kolicina = float(linija)
        print(format(kolicina, '.2f'))
        linija = prodaja_fajl.readline()

    prodaja_fajl.close()
```

```
main()
Дaje:
30.00
25.00
45.00
```

Коришћење for петље за читање линија

У многим програмским језицима се користи техника детекције краја фајла слична претходном враћању празног стринга употребом readline методе.

У Пајтону се може користити for петља која аутоматски чита линију у фајлу без тестирања услова који би указао на крај фајла.

На такав начин ће петља аутоматски да се заустави на крају фајла.

Ова техника је једноставнија јер нема потребе за посебним тестирањем услова за крај фајла.

Општи формат петље:

```
for promenjiva in fajl_objekat:
    iskaz
    iskaz
```

...

Овде је promenjiva име променљиве а fajl_objekat је променљива која упућује на фајл објекат.

Петља ће итерирати по једном за сваку линију у фајлу.

0211 Итерација for петље по линијама фајла

```
def main():
    prodaja_fajl = open('prodaja.txt', 'r')
    for linija in prodaja_fajl:
        kolicina = float(linija)
        print(format(kolicina, '.2f'))
    prodaja_fajl.close()
```

```
main()
30.00
20.00
45.00
```

0212 Манипулација фајлова са подацима о филмовима

Написати програм који омогућава унос времена трајања кратких филмова у секундама и чување тих података у фајлу и читање садржаја фајла, приказ времена трајања и добијање укупног времена трајања свих филмова

```
def main():
    broj_filmova = int(input('Koliko filmova je u projektu? '))
    video_fajl = open('vremena_filmova.txt', 'w')
    print('Uneti vreme trajanja svakog filma.')
    for i in range(1, broj_filmova + 1):
        vreme_trajanja = float(input('Film #' + str(i) + ': '))
        video_fajl.write(str(vreme_trajanja) + '\n')

    video_fajl.close()
    print('Vremena trajanja su sacuvana u fajlu vremena_filmova.txt.')

    video_fajl = open('vremena_filmova.txt', 'r')
    ukupno = 0.0
    i = 0
    print('Ovo su vremena trajanja za svaki film:')
    for linija in video_fajl:
        vreme_trajanja = float(linija)
        i += 1
        print('Film #', i, ': ', vreme_trajanja, sep='')
        ukupno += vreme_trajanja

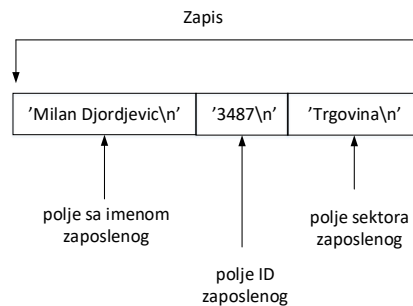
    video_fajl.close()
    print('Ukupno vreme trajanja svih filmova je', ukupno, 'sekundi.')
```

```
main()
Koliko filmova je u projektu? 3
Uneti vreme trajanja svakog filma.
Film #1: 1.98
Film #2: 0.34
Film #3: 0.98
Vremena trajanja su sacuvana u fajlu vremena_filmova.txt.
Ovo su vremena trajanja za svaki film:
Film #1: 1.98
Film #2: 0.34
Film #3: 0.98
Ukupno vreme trajanja svih filmova je 3.3 sekundi.
```

Увод у записе

Када се податак уписује у фајл, често се организује у записе (records) и поља (fields). Запис је потпуни сет података који описују неку ствар док је поље један податак унутар записа.

Нпр, ако је потребно сместити податке о запосленима компаније у фајл; фајл ће садржати запис о сваком запосленом а сваки запис ће бити колекција поља попут имена, ИД броја и сектора унутар компаније.



Сваки пут када се упише запис у фајл са секвенцијалним приступом, уносе се подаци у поља која чине тај запис, једно за другим.

У примеру запис за једног запосленог се састоји од три поља: поље са именом запосленог, поље са ИД запосленог и поље са сектором запосленог.

0213 Упис записа о запосленима у фајл

```
def main():
    broj_zaposlenih = int(input('Koliko zapisa treba kreirati? '))
    zaposleni_fajl = open('zaposleni.txt', 'w')
    for i in range(1, broj_zaposlenih + 1):
        print('Uneti podatke o zaposlenom #', i, sep='')
        ime_zaposlenog = input('Ime: ')
        id_zaposlenog = input('ID broj: ')
        sektor_zaposlenog = input('Sektor: ')

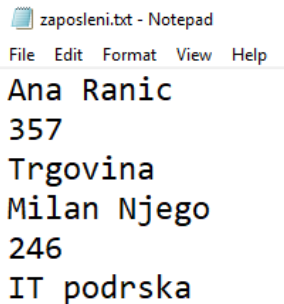
        zaposleni_fajl.write(ime_zaposlenog + '\n')
        zaposleni_fajl.write(id_zaposlenog + '\n')
        zaposleni_fajl.write(sektor_zaposlenog + '\n')
        print()

    zaposleni_fajl.close()
    print('Podaci o zaposlenima su upisani u fajl zaposleni.txt.')
```

```
main()
Koliko zapisa treba kreirati? 2
Uneti podatke o zaposlenom #1
Ime: Ana Ranic
ID broj: 357
Sektor: Trgovina
```

```
Uneti podatke o zaposlenom #2
Ime: Milan Njego
ID broj: 246
Sektor: IT podrška
```

Podaci o zaposlenima su upisani u fajl zaposleni.txt.



```
zaposleni.txt - Notepad
File Edit Format View Help
Ana Ranic
357
Trgovina
Milan Njego
246
IT podrška
```

Када се чита запис из фајла са секвенцијалним приступом, чита се податак из сваког поља један за другим, све док се не прочита цео запис о једном запосленом.

0214 Читање записа о запосленима

Програм за читање записа о запосленима у zaposleni.txt фајлу

```
def main():
    zaposleni_fajl = open('zaposleni.txt', 'r')
    ime = zaposleni_fajl.readline()
    while ime != '':
        id_zaposlenog = zaposleni_fajl.readline()
        sektor_zaposlenog = zaposleni_fajl.readline()
        ime = ime.rstrip('\n')
        id_zaposlenog = id_zaposlenog.rstrip('\n')
        sektor_zaposlenog = sektor_zaposlenog.rstrip('\n')
        print('Ime:', ime)
        print('ID:', id_zaposlenog)
        print('Sektor:', sektor_zaposlenog)
        print()
        ime = zaposleni_fajl.readline()

    zaposleni_fajl.close()

main()
Ime: Ana Ranic
ID: 357
Sektor: Trgovina

Ime: Milan Njego
ID: 246
Sektor: IT podrška
```

Тип података bytes

Тип података bytes у Пајтону је непромењив и смешта секвенцу вредности од 0 до 255 (као 8-битни податак).

0215 Тип података bytes

```
primer_bytes = bytes(4)
print(type(primer_bytes))
print(primer_bytes)
```

Даје:

```
<class 'bytes'>
b'\x00\x00\x00\x00'
```

Упис бинарних података у фајл

За упис бинарних података у фајл, користи се 'wb' мод.

За прикључивање нових бинарних података постојећим у фајлу, користи се 'ab' мод.

0216 Бинарни упис

```
binarni_fajl = open("test.txt", "wb")
binarni_fajl.write(b'\x00')
binarni_fajl.close()
```

Отварањем фајла test.txt не може се приметити никакав садржај пошто је уписан садржај у бинарном облику па се не може читати преко текст едитора.

Читање бинарних података из фајла

За читање бинарних података из фајла, користи се 'rb' мод.

0217 Бинарно читање

```
binarni_fajl = open("test.txt", "rb")
podaci = binarni_fajl.read()
print(podaci)
binarni_fajl.close()
```

Даје: b'\x00'

Рад са бројевима у бинарним фајловима

Да би се нумерички подаци могли користити у бинарном формату, бројеви се еморају конветовати у бајт низ пре уписивања у фајл.

Функција bytearray() враћа бајт репрезентацију објекта.

0218 Објекти у бинарни фајл

Смештање листе бројева у бинарни фајл

```
fajl = open("binfile.bin", "wb")
brojevi = [50, 10, 40, 20, 30]
niz = bytearray(brojevi)
fajl.write(niz)
fajl.close()
```

Декодирање и кодовање текста у бинарни формат

Декодирање бинарног податка у текстуални формат се врши помоћу методе `decode()`.

У заграду се смешта аргумент којим се указује у ком текстуалном формату се врши декодирање из бинарног формата.

0219 Декодирање

```
binarni_podaci = b'Ovo je tekst.'  
tekst = binarni_podaci.decode('utf-8')  
print(tekst)  
binarni_podaci = bytes([65, 66, 67])  
tekst = binarni_podaci.decode('utf-8')  
print(tekst)
```

Дaje:

Ovo je tekst

ABC

Кодовање текста у бинарни формат се врши помоћу методе `encode()`, при чему се као аргумент користи назив формата из којег се врши кодовање.

0220 Кодовање

```
poruka = "Zdravo"  
binarna_poruka = poruka.encode('utf-8')  
print(type(binarna_poruka))
```

Дaje: <class bytes>

0221 Декодирање и кодовање текста у бинарни формат

Текстуални податак се конвертује у бинарни из utf-8 кода и смешта у бинарни фајл. Бинарни податак се чита из бинарног фајла и конвертује у текст формат преко utf-8 кода.

```
fajl = open('binar.txt', 'wb')  
tekst = "Ovo je tekst."  
binarni_podaci = tekst.encode('utf-8')  
fajl.write(binarni_podaci)  
fajl.close()  
fajl = open('test.txt', 'rb')  
binarni_podaci = fajl.read()  
tekst = binarni_podaci.decode('utf-8')  
print(tekst)  
fajl.close()
```

Задаци са датотекама

0222 Основно манипулисање фајлом

Написати програм који отвара фајл `moje_ime.txt`, омогућава упис имена у фајл и затвара фајл.

```
def main():
    moj_fajl = open("moje_ime.txt", 'w')
    moj_fajl.write('Dragan\n')
    moj_fajl.close()
```

```
main()
```

0223 Корисничко именовање фајла

Написати програм који тражи од корисника да именује фајл који жели да се отвори. Затим отвара тражени фајл за читање па га и затвара.

```
def main():
    print("Otvара se fajl za citanje sadržaja.")
    korisnicki_fajl = input("Uneti naziv fajla: ")
    moj_fajl = open(korisnicki_fajl, 'r')
    moj_fajl.close()
```

```
main()
```

0224 Приказ кориснички именованог фајла

Написати програм који отвара фајл и тражи од корисника да унесе име фајла, приказује садржај фајла и затвара фајл.

```
def main():
    print("Otvара se fajl za citanje sadržaja.")
    naziv_fajla = input("Uneti naziv fajla: ")
    korisnicki_fajl = open(naziv_fajla, 'r')
    print(korisnicki_fajl.read())
    korisnicki_fajl.close()
```

```
main()
```

0225 Креирање изгледа података у фајлу

Написати програм који креира фајл brojevi_0_5.txt и у који се уноси низ целих бројева од 0 до 5 у сваком реду по један број.

```
def main():
    print("Otvара se fajl za upis brojeva od 0-5")
    naziv_fajla = open("brojevi_0_5.txt", 'w')
    for i in range(6):
        naziv_fajla.write(str(i) + "\n")
    naziv_fajla.close()
```

```
main()
```

Питања и задаци за самосталан рад

Задаци

064 Користећи фајл brojevi_0_5.txt, написати програм који приказује садржај тог фајла.

065 Написати програм који креира фајл brojevi_0_5.txt и у који се уноси низ целих бројева од 0 до 5 и то у једном реду сви бројеви.

066 Написати програм који креира фајл са називом по жељи корисника и унутра уноси низ целих бројева у опсегу по жељи корисника, сваки број у посебном реду.

067 Написати програм који креира фајл са називом по жељи корисника и унутра уноси низ квадрата целих бројева у опсегу по жељи корисника, сваки број у посебном реду.

0226 Сабирање бројчаних података из фајла

Користећи фајл brojevi_0_5.txt, написати програм који сабира бројеве из фајла и приказује њихов збир.

```
def main():
    print("Otvora se fajl 'brojevi_0_5.txt'")
    fajl = open("brojevi_0_5.txt", 'r')
    suma = 0
    for linija in fajl:
        suma += int(linija)
    print("Zbir brojeva u fajlu: ", suma)
    fajl.close()
```

```
main()
```

0227 Манипулација подацима у два фајла

Написати програм који креира фајл са именом према жељи корисника и уноси у фајл жељене бројеве. Програм затим сабира унете бројеве и резултат смешта у други фајл, zbir.txt и приказује садржај фајла.

```
def main():
    print("Otvora se fajl 'brojevi_0_5.txt'")
    fajl = open("brojevi_0_5.txt", 'r')
    suma = 0
    for linija in fajl:
        suma += int(linija)
    print("Zbir brojeva u fajlu: ", suma)
    fajl.close()
    print("Otvora se novi fajl za upis sadrzaja")
    fajl = open("novi_fajl.txt", 'w')
    fajl.write(str(suma))
    fajl.close()
    fajl = open("novi_fajl.txt", 'r')
    print("Sadrzaj novi_fajl.txt je: ", )
    print(fajl.read())
    fajl.close()
```



```
main()
```

0228 Приказ садржаја фајла по линијама

Написати програм који отвара фајл према жељи корисника, приказује његов садржај по линијама фајла тако што сваки ред приказа започиње са редним бројем линије и две тачке.

```
def main():
    naziv_fajla = input("Uneti naziv fajla: ")
    fajl = open(naziv_fajla, 'r')
    broj_linije = 0
    for linija in fajl:
        broj_linije += 1
        print(str(broj_linije) + ': ' + linija, end = "")
    print()
    fajl.close()
```

```
main()
```

Питања и задаци за самосталан рад

Задаци

068 Написати програм који чита целе бројеве из фајла и рачуна средњу вредност тих бројева као цео број.

069 Написати програм који ће приказати само пет првих линија садржаја фајла. Ако фајл има мање од пет линија приказати садржај целог фајла.

070 Написати програм уписује низ случајних бројева у фајл. Сваки број је у опсегу од 1 до 500. Апликација омогућава кориснику да одреди број случајних бројева у фајлу.

0229 Придодавање новог садржаја у фајл

Написати програм који придодаје ново име у фајл са постојећим именима.

```
def main():
    fajl = open("imena.txt", 'r')
    prikaz_fajla(fajl)
    fajl.close()
    unos_imena(fajl)
    fajl.close()
    fajl = open("imena.txt", 'r')
    prikaz_fajla(fajl)
    fajl.close()

def prikaz_fajla(f):
    print("Prikaz sadrzaja fajla\n")
    print(f.read())
    print()
```

```
def unos_imena(f):
    ime = input("Uneti novo ime za fajl: ")
    fajl = open("imena.txt", 'a')
    fajl.write("\n" + ime)
```

```
main()
```

0230 Брисање одређеног садржаја из фајла

Фајл student.txt садржи неколико записа а сваки запис се састоји од два поља: име студента и резултат на тесту (један испод другог). Написати код који брише запис који садржи име студента "Petar Petrovic".

```
import os
```

```
def main():
    nasao = False
    trazi = "Petar Petrovic"
    fajl = open("student.txt", 'r')
    fajl2 = open("pomoc.txt", 'w')
    ime = fajl.readline()
    while ime != "":
        test = int(fajl.readline()) #cita rezultat sa testa
        ime = ime.rstrip('\n') #skida \n iz imena
        #ako trenutni zapis se ne brise, ide u fajl2
        if ime != trazi:
            fajl2.write(ime + '\n')
            fajl2.write(str(test) + '\n')
        else:
            nasao = True
            ime = fajl.readline()

    fajl.close()
    fajl2.close()

    os.remove('student.txt')
    os.rename('pomoc.txt', 'student.txt')
```

```
main()
```

0231 Модификација одређеног садржаја у фајлу

У фајлу student.txt из претходног задатка променити податак о резултату на тесту студента Марка Марића на 100.

```
import os
```

```
def main():
    nasao = False
```

```

trazi = "Marko Maric"
fajl = open("student.txt", 'r')
fajl2 = open("pomoc.txt", 'w')
ime = fajl.readline()
while ime != "":
    test = int(fajl.readline()) #cita rezultat sa testa
    ime = ime.rstrip('\n') #skida \n iz imena
    #ako trenutni zapis se ne brise, ide u fajl2
    if ime != trazi:
        fajl2.write(ime + '\n')
        fajl2.write(str(test) + '\n')
    else:
        fajl2.write(ime + '\n')
        fajl2.write('100' + '\n')
        nasao = True
    ime = fajl.readline()

fajl.close()
fajl2.close()
os.remove('student.txt')
os.rename('pomoc.txt', 'student.txt')
main()

```

Питања и задаци за самосталан рад

Задаци

071 Написати програм који омогућава кориснику да уноси назив града, назив улице и број зграде као запис са три поља у посебним линијама, у фајл `adrese.txt`. Програм исписује садржај фајла на екрану а затим цео садржај пребацује у нов фајл, `adrese1.txt`.

072 Написати програм који омогућава придодавање нових записа у фајл `adrese1.txt`. Омогућити кориснику да уноси нове записе по жељи.

073 Написати програм који брише жељени запис из фајла `adrese.txt`.

0232 Смештање података о филмовима у фајлу

Написати програм који омогућава унос времена трајања кратких филмова у секундама и чување тих података у фајлу и читање садржаја фајла, приказ времена трајања и добијање укупног времена трајања свих филмова

```

def main():
    broj_filmova = int(input('Koliko filmova je u projektu? '))
    video_fajl = open('vremena_filmova.txt', 'w')
    print('Uneti vreme trajanja svakog filma.')
    for i in range(1, broj_filmova + 1):
        vreme_trajanja = float(input('Film #' + str(i) + ': '))
        video_fajl.write(str(vreme_trajanja) + '\n')

```

```

video_fajl.close()
print('Vremena trajanja su sacuvana u fajlu vremena_filmova.txt.')

video_fajl = open('vremena_filmova.txt', 'r')
ukupno = 0.0
i = 0
print('Ovo su vremena trajanja za svaki film:')
for linija in video_fajl:
    vreme_trajanja = float(linija)
    i += 1
    print('Film #', i, ': ', vreme_trajanja, sep='')
    ukupno += vreme_trajanja

video_fajl.close()
print('Ukupno vreme trajanja svih filmova je', ukupno, 'sekundi.')

```

```

main()
Koliko filmova je u projektu? 3
Uneti vreme trajanja svakog filma.
Film #1: 1.98
Film #2: 0.34
Film #3: 0.98
Vremena trajanja su sacuvana u fajlu vremena_filmova.txt.
Ovo su vremena trajanja za svaki film:
Film #1: 1.98
Film #2: 0.34
Film #3: 0.98
Ukupno vreme trajanja svih filmova je 3.3 sekundi.

```

0233 Читање записа о запосленима у фајлу

Програм за читање записа о запосленима у zaposleni.txt фајлу

```

def main():
    zaposleni_fajl = open('zaposleni.txt', 'r')
    ime = zaposleni_fajl.readline()
    while ime != '':
        id_zaposlenog = zaposleni_fajl.readline()
        sektor_zaposlenog = zaposleni_fajl.readline()
        ime = ime.rstrip('\n')
        id_zaposlenog = id_zaposlenog.rstrip('\n')
        sektor_zaposlenog = sektor_zaposlenog.rstrip('\n')
        print('Ime:', ime)
        print('ID:', id_zaposlenog)
        print('Sektor:', sektor_zaposlenog)
        print()
        ime = zaposleni_fajl.readline()

    zaposleni_fajl.close()

```

```
main()
Ime: Ana Ranic
ID: 357
Sektor: Trgovina
```

```
Ime: Milan Njego
ID: 246
Sektor: IT podrška
```

0234 Уметање белине у текст у фајлу

Фајл `tekst_prljav.txt` садржи стринг у сваком од редова фајла. У стринговима нема белине између речи већ је унет стринг "prazno". Коришћењем помоћног фајла `probni.txt` променити садржај фајла `tekst_prljav.txt` тако да уместо стринга "prazno" се уметне белина.

```
import os

def main():
    pregled_unetih_podataka()
    brisanje_prazno()
    os.remove('tekst_prljav.txt')
    os.rename('probni.txt', 'tekst_prljav.txt')
    pregled_unetih_podataka()

def pregled_unetih_podataka():
    print("\nУ фајлу 'tekst_prljav.txt' су унети следећи подаци:")
    fajl = open("tekst_prljav.txt", "r")
    print(fajl.read())
    fajl.close()

def brisanje_prazno():
    print("\nУ фајлу 'tekst_prljav.txt' се брише реч 'prazno' и уноси белина.")
    fajl1 = open("tekst_prljav.txt", "r")
    fajl2 = open("probni.txt", "w")
    a = ""
    mesto = 0
    for i in fajl1:
        for x in range(len(i)):
            if i[x : (x + 6)] == "prazno":
                mesto = 5
                a += " "
            else:
                if mesto != 0:
                    mesto -= 1
                else:
                    a += i[x]
```

```
fajl2.write(a)
a = ""
```

```
fajl1.close()
fajl2.close()
```

```
main()
```

Питања и задаци за самосталан рад

Задаци

074 Фајл brojevi.txt је напуњен редовима цифара. Креирати текстуални фајл у којем ће сваки ред садржати поруку о редном броју реда и броју цифара у истом реду фајла brojevi.txt (5.ред текстуалног фајла даје поруку о броју цифара у 5.реду brojevi.txt фајла).

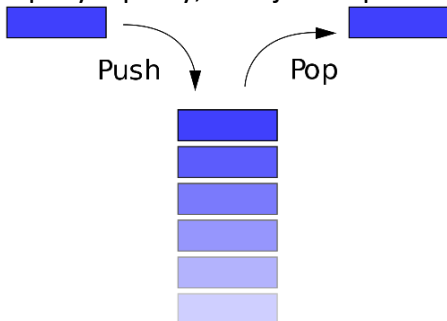
075 Фајл tekst_prljav1.txt садржи стринг у сваком од редова фајла. На почетку сваког реда је стринг "start" а на крају сваког реда је стринг "kraj". Коришћењем помоћног фајла probni.txt променити садржај фајла tekst_prljav.txt тако да се одстрани стрингови "start" и "kraj" из фајла.

076 Фајл brojevi.txt је напуњен редовима цифара. Креирати текстуални фајл у којем ће сваки ред садржати поруку о редном броју реда и збиру цифара у истом реду фајла brojevi.txt (5.ред текстуалног фајла даје поруку о збиру цифара у 5.реду brojevi.txt фајла).

Стек

Стек као тип података

У рачунарству, стек је апстрактни тип података који се користи као колекција елемената.



Користи се и други назив за стек, LIFO (last in first out) са којим се описује редослед коришћења елемената на стеку.

Операције на стеку су: push (додаје елемент на стек), pop (скида елемент са стека), peek (откривање елемента са врха стека без његовог скидања са стека), length (добивања тренутног броја елемената на стеку).

Назив стек се изводи из аналогије са групом физичких објеката који се гомилају једна на друге. При томе, могуће је смештање новог елемента само на врх стека; исто тако, могуће је само скидање тренутног елемента са врха стека.

То значи, да би се дошло до неког елемента раније смештеног у стеку, може се урадити само скидањем свих елемената са стека који су после њега смештени на стек.

У хардверу рачунара, стек се смешта у меморију која има ограничени капацитет, па се може десити да стек нема више простора за примање нових елемената; овакав случај се назива stack overflow.

Рад са стеком у Пајтону

У Пајтону се користе листе као уграђене структура података, заједно са својим методама, да би симулирале операције којима се дефинише рад са стеком.

0235 Симулација стека употребом листа

```
def main():
    test1()
    test2()

#funkcija kreira stek i inicijalizuje ga kao prazan
def nov_stek():
    stek_prazan = []
    return stek_prazan

def duzina_steka(s):
    return len(s)

#stek je prazan ako je njegova duzina 0
def jeste_prazan(s):
    return len(s) == 0

#funkcija dodaje element u stek
def dodaj_element(s, element):
    s.append(element)

#funkcija sklanja poslednje uneti element sa steka
def skini_element(s):
    if (jeste_prazan(s) != True):
        return str(s.pop())
    else:
        return "Stek je prazan."

def test1():
    stek = nov_stek()
    print(skini_element(stek))
    proba1 = duzina_steka(stek)
    print("Duzina steka na pocetku je", proba1)
    dodaj_element(stek, 10)
    dodaj_element(stek, 20)
```

```

proba2 = duzina_steka(stek)
print("Duzina steka posle dva dodata elementa je", proba2)
skini_element(stek)
proba3 = duzina_steka(stek)
print("Duzina steka posle skidanja jednog elementa je", proba3)
print()
if proba1 == 0 and proba2 == 2 and proba3 == 1:
    print("1.testiranje koda steka uspesno!!")
else:
    print("1.testiranje koda steka NEUSPESNO!!")
print()

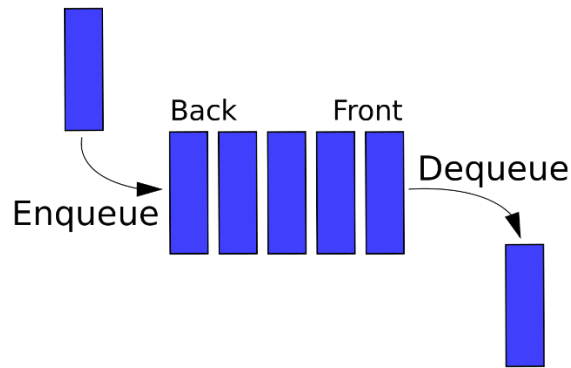
def test2():
    stek = [100, 200, 300]
    proba1 = duzina_steka(stek)
    print("Duzina steka na pocetku je", proba1)
    skini_element(stek)
    proba2 = duzina_steka(stek)
    print("Duzina steka posle skidanja prvog elementa je", proba2)
    skini_element(stek)
    skini_element(stek)
    proba3 = duzina_steka(stek)
    print("Duzina steka posle skidanja treceg elementa je", proba3)
    print()
    if proba1 == 3 and proba2 == 2 and proba3 == 0:
        print("2.testiranje koda steka uspesno!!")
    else:
        print("2.testiranje koda steka NEUSPESNO!!")
    print()

main()

```

Структура реда у програмирању

Ред је колекција објеката која подржава семантику (FIFO – first-in, first-out) за уметање (enqueue) и одстрањивање (dequeue) података а не подржава слободан приступ било којем објекту у реду осим прво уметнутом објекту у реду.



Редови се користе у разноврсним алгоритмима за решавање проблема распореда и паралелног програмирања.

Алгоритми редоследа често користе интерно редове приоритета.

То су специјализовани редови: уместо добијања следећег елемента по редоследу, они узимају елемент који је највећег приоритета.

Приоритет појединачних елемената се одређује према редоследу премињеном на кључеве елемената.

Обичан ред не мења редослед својих елемената.

У Пајтону су уграђени неколико имплементација реда са мало измењеним карактеристикама.

Ред

Ред као имплементација листе

У Пајтону се може лако користити уграђена структура листе као имплементација реда, али због своје мање ефикасности се не користи у алгоритмима.

Ова имплементација се може користити у алгоритмима који захтевају рад са мањим бројем елемената у редовима.

0236 Симулација реда употребом листа

```
q = []
q.insert(0, 'treci')
q.insert(0, 'drugi')
q.insert(0, 'prvi')

print(q)           #daje ['prvi', 'drugi', 'treci']

q.pop(2)           #odstranjuje element 'treci'

print(q)           #daje ['prvi', 'drugi']
```

Ред као имплементација класе collections.deque

Класа deque имплементира ред са два краја који подржава додавање и одстрањивање елемената са оба краја реда.

Пајтонови deque елементи су имплементирани као објекти у двоструко спрегнутој листи чиме имају повећану ефикасност у раду приликом уметања и одстрањивања елемената.

Ипак ова имплементација није ефикасна код приступа елементима у средини реда.
Из свих ових разлога ова имплементација подржава употребу класа deque и за рад са стековима и за рад са редовима.
Често се користи за реализацију структуре података унутар Пајтонове стандардне библиотеке.

0237 Симулација реда употребом класе deque

```
from collections import deque
q = deque()

q.append('prvi')
q.append('drugi')
q.append('treci')

print(q)           #daje deque(['prvi', 'drugi', 'treci'])

q.popleft()        #odstranjuje 'prvi' element
q.popleft()        #odstranjuje 'drugi' element
q.popleft()        #odstranjuje 'treci' element
q.popleft()        #javlja gresku IndexError: "pop from an empty deque"
```

Задаци са стеком и редом

0238 Манипулација листом бројева као стеком

Написати програм који користи дату листу [100, 200, 300] као стек. Уклонити два елемента са стека па додати нов елемент (400) на стек. Приказати садржај стека.

```
def main():
    stek = [100, 200, 300]
    prikaz_steka(stek)
    skini_element(stek)
    prikaz_steka(stek)
    skini_element(stek)
    prikaz_steka(stek)
    dodaj_element(stek, 400)
    prikaz_steka(stek)

#stek je prazan ako je njegova duzina 0
def jeste_prazan(s):
    return len(s) == 0

#funkcija dodaje element u stek
def dodaj_element(s, element):
    print("Dodajem element", str(element), "u stek.")
    s.append(element)

#funkcija sklanja poslednje uneti element sa steka
def skini_element(s):
```

```

    print("Uklanjam poslednji element u steku.")
    if (jeste_prazan(s) != True):
        return str(s.pop())
    else:
        return "Stek je prazan."

def prikaz_steka(s):
    print("Stek trenutno izgleda ovako:", s)
    print()

main()

```

0239 Манипулација листом стрингова као стеком

Написати програм који користи ред као листа са елементима ['prvi', 'drugi', 'treci'], убади елемент 'nulti' као први елемент у реду, а затим се последњи елемент из реда избаци из реда. Приказати садржај реда.

```

def main():
    red = ['prvi', 'drugi', 'treci']
    red = unos_elementa(red, 'nulti')
    prikaz_reda(red)
    indeks = len(red) - 1
    red = odstranjivanje_elementa(red, indeks)
    prikaz_reda(red)

def odstranjivanje_elementa(r, i):
    r.pop(i)
    return r

def unos_elementa(r, element):
    r.insert(0, element)
    return r

def prikaz_reda(r):
    print(r)

main()

```

0240 Манипулација методом deque као стеком

Написати програм који користи ред као имплементацију класе collections.deque. Унети три целобројне вредности у ред, одстранити прву и последњу из реда. Приказати садржај реда.

```

from collections import deque
q = deque()
q.append(3)
q.append(9)
q.append(1)
print(q)

```

```
q.pop()
print(q)
q.popleft()
print(q)
```

Питања и задаци за самосталан рад

Задаци

077 Написати програм који користи празну листу као стек. Унети елементе 100 и 200 у стек. Приказати садржај стека.

078 Написати програм који користи дату листу [100, 200, 300] као стек. Уклонити све елементе из стека. Вратити елементе у стек обрнутим редоследом.

079 Написати програм који користи празан ред као листу. Унети у ред једноцифрени број, његов квадрат и његов куб. Одстранити из реда све осим последње унетог елемента.

080 Написати програм који користи ред као листу [100, 200, 300] да би се на крају добио следећи ред [0, 1, 100].

Изузеци

Грешке у Пајтону

Програм написан у Пајтону прекида са извршавањем ако наиђе на грешку која може бити синтаксна или изузетак (exception).

0241 Генерисање синтаксне грешке и изузетка

Намерним генерисањем синтаксне грешке а касније и изузетка, приказати разлику у реаговању Пајтон интерпретера. Објаснити разлике у реакцији.

```
>>> print(1 + 1))
File "<stdin>", line 1
    print(1 + 1))
      ^
SyntaxError: invalid syntax
```

```
>>> print(0 / 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Види се да је дошло до појаве изузетка.

Изузетак се појави када синтаксно тачан Пајтон код проузрокује грешку.

Последња линија интерпреторске поруке указује каква врста изузетка се десила.

У Пајтону постоји велики број уграђених изузетака што омогућава адекватно реаговање на сваки од њих.

Подизање изузетка

Може се користити команда `raise` да би се подигао изузетак (rising an exception) ако се испуне одређени услови.

0242 Употреба команде `raise`

```
x = 10
if x > 5:
    raise Exception('x ne treba da bude vece od 5. Vrednost x je:
{}'.format(x))
```

Traceback (most recent call last):

raise Exception('x ne treba da bude vece od 5. Vrednost x je: {}'.format(x))

Exception: x ne treba da bude vece od 5. Vrednost x je: 10

Види се да је извршавање кода заустављено, приказан је изузетак који се десио и дато је некакво објашњење шта је изазвало прекид програма.

Питања и задаци за самосталан рад

Задаци

081 Генерисати неколико намерних изузетака у различитим кодовима. Генерисати следеће изузетке: `IndexError`, `NameError`, `IndentationError`, `TypeError`, `ValueError`.

082 Коришћењем команде `raise` генерисати реаговање на изузетак ако је вредност промењиве различита од 0.

Појава `AssertionError` изузетка

Да би се предупредили прекиди, може се креирати тврдња (`Assertion`), која се појављује када је испуњен услов.

Ако је услов испуњен, програм се може наставити са извршавањем.

Ако услов није испуњен, програм подиже `AssertionError` изузетак.

0243 Употреба тврдње `assertion`

```
import sys
assert ('linux' in sys.platform), "Ovaj kod se izvrsava samo na Linux OS."
```

Ако се овај код реализује на компјутеру са Линукс оперативним системом, тврдња неће изазвати изузетак.

Ако се код реализује на компјутеру са Windows оперативним системом, резултат тврдње ће бити `False` и појављује се следеће:

Traceback (most recent call last):

File "<input>", line 2, in <module>

AssertionError: Ovaj kod se izvrsava samo na Linux OS.

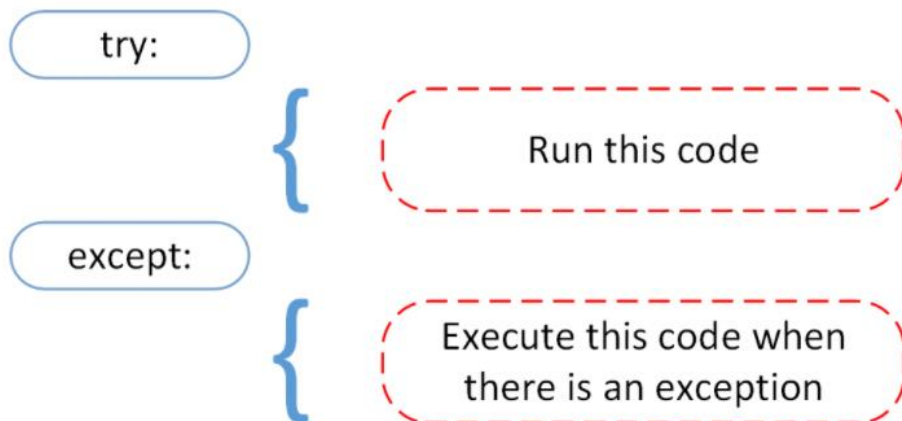
Овиме се зауставља извршавање програма.

Руковање изузецима

Блокови try и except се користе за прикупљање и руковање изузецима.

Део кода који се налази у try блоку се сматра нормалним делом кода.

Део кода који се налази у except блоку је одговор програма на појаву било којег изузетка који би се евентуално појавио у коду унутар try блока.



0244 Блокови try и except

```
def linux_interakcija():  
    assert ('linux' in sys.platform), "Funkcija se izvrsava samo na Linux OS."  
    print('Radi nesto.')
```

```
try:  
    linux_interakcija()  
except:  
    print("Linux funkcija nije izvrшена.")
```

Функција linux_interakcija() може да се стартује само под Линукс оперативним системом. Команда assert ће изазвати појаву AssertionError изузетка ако се функција позове под неким другим оперативним системом.

Када се изузетак догоди унутар програма који садржи ову функцију, програм ће наставити са реализацијом и само ће дати информацију да се функција неће извршити.

0245 Реакција на AssertionError

Креирати део кода који ће реаговати на AssertionError изузетак исписивањем поруке о неизвршеној функцији.

```
import sys  
def linux_interakcija():  
    assert ('linux' in sys.platform), "Funkcija se izvrsava samo na Linux OS."  
    print('Radi nesto.')
```

```
try:
```

```
linux_interakcija()
```

```
except AssertionError as greska:  
    print(greska)  
    print("Linux funkcija nije izvrшена.")
```

Funkcija se izvrшава samo na Linux OS.
Linux funkcija nije izvrшена.

Питања и задаци за самосталан рад

Задаци

083 Коришћењем блокова try и ехсепт направити код који ће реаговати на појаву грешке дељења са 0.

Руковање вишеструким изузецима

Унутар try блока се може користити више од једног позива функције, чиме се предвиђа хватање већег броја могућих изузетака.

Употреба простог ехсепт заглавља сакрива све остале грешке које се могу појавити и зато треба избегавати овакве опште ехсепт блокове.

Боља пракса је коришћење специфичних ехсепт блокова.

0246 Исписивање поруке о неизвршеној функцији

Креирати део кода који ће реаговати на AssertionError изузетак исписивањем поруке о неизвршеној функцији.

```
import sys  
def linux_interakcija():  
    assert ('linux' in sys.platform), "Funkcija se izvrшава samo na Linux OS."  
    print('Radi nesto.')
```

```
try:  
    linux_interakcija()  
    with open("fajl.log") as file:  
        read.data = file.read()  
  
except FileNotFoundError as fnf_error:  
    print(fnf_error)  
  
except AssertionError as error:  
    print(error)  
    print("Linux funkcija nije izvrшена.")
```

Funkcija se izvrшава samo na Linux OS.
Linux funkcija nije izvrшена.

Унутар try блока, одмах се стартује изузетак који рукује са AssertionError и не долази се до дела који рукује покушајем отварања fajl.log.

Када би се исти код стартовао на компјутеру под Линукс оперативним системом, добила би се следећа порука:

[Errno 2] No such file or directory: "file.log"

То значи да је реаговао други ехсепт блок који хвата грешку непроналажења жељеног фајла, пошто се први ехсепт блок није десио (компјутер је под Линукс оперативним системом).

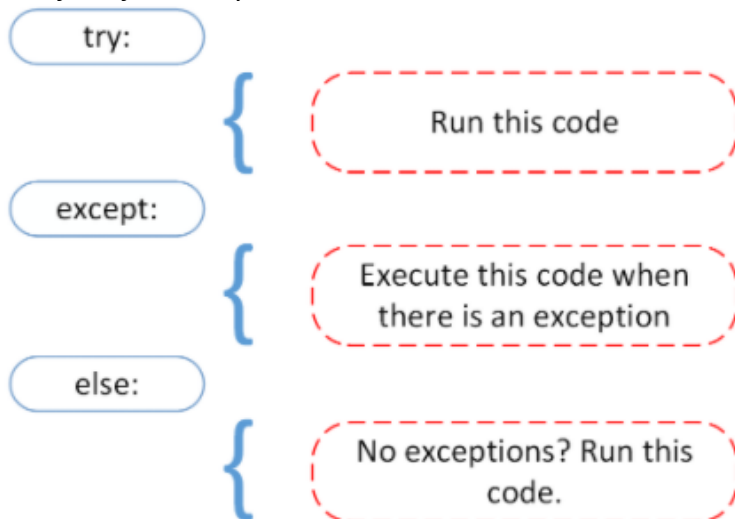
Питања и задаци за самосталан рад

Задаци

084 Руковати са изузецима појаве дељења нулом и погрешног именовања променљиве.

Команда else

Употребом else команде може се наредити програму да изврши део кода само под условом да се није појавио изузетак.



0247 Употреба else блока

```
import sys
def linux_interakcija():
    assert ('linux' in sys.platform), "Funkcija se izvrsava samo na Linux OS."
    print('Radi nesto.')

try:
    linux_interakcija()

except AssertionError as greska:
    print(greska)

else:
    print("Izvrsva se else komanda.")
```


Под Линукс оперативним системом, излаз би био:

Radi nesto.

Izvršava se else komanda.

Види се да пошто се није десио ниједан изузетак, извршила се else наредба.

0248 Блок else са хватањем изузетка

```
import sys
def linux_interakcija():
    assert ('linux' in sys.platform), "Funkcija se izvršava samo na Linux OS."
    print('Radi nesto.')

try:
    linux_interakcija()

except AssertionError as greska:
    print(greska)

else:
    try:
        with open("fajl.log") as file:
            read.data = file.read()

    except FileNotFoundError as fnf_error:
        print(fnf_error)
```

Ако би се код стартовао на Линукс машини:

Radi nesto.

[Errno 2] No such file or directory: "file.log"

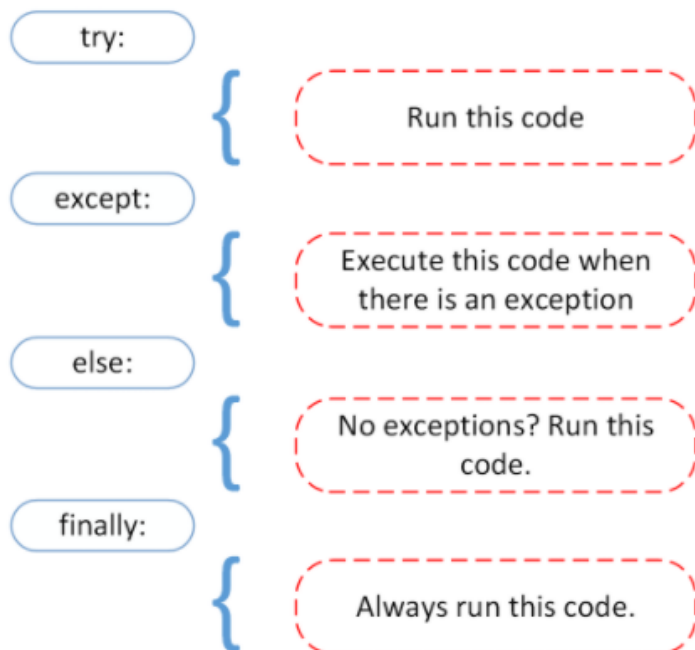
Види се да се функција linux_interakcija() реализовала.

Пошто се није подигао ниједан изузетак, покушано је да се отвори фајл "fajl.log".

Пошто тај фајл не постоји, ухваћен је FileNotFoundError изузетак.

Употреба finally команде

Ако је потребно имплементирати неку акцију за чишћење кода после његовог извршења, то се може извршити употребом finally команде.



0249 Употреба finally

```
import sys
def linux_interakcija():
    assert ('linux' in sys.platform), "Funkcija se izvrsava samo na Linux OS."
    print('Radi nesto.')

try:
    linux_interakcija()

except AssertionError as greska:
    print(greska)

else:
    try:
        with open("fajl.log") as file:
            read.data = file.read()

        except FileNotFoundError as fnf_error:
            print(fnf_error)

finally:
    print("Izvršava se u svakom slučaju.")
```

Funkcija se izvršava samo na Linux OS.
Izvršava se u svakom slučaju.

Сада, све што се налази у finally делу ће се реализовати пошто није битно да ли се десио изузетак унутар try или else блока.

Врсте изузетака

Рб.	Назив изузетка и опис
1	Exception Base class for all exceptions
2	StopIteration Raised when the next() method of an iterator does not point to any object.
3	SystemExit Raised by the sys.exit() function.
4	StandardError Base class for all built-in exceptions except StopIteration and SystemExit.
5	ArithmeticError Base class for all errors that occur for numeric calculation.
6	OverflowError Raised when a calculation exceeds maximum limit for a numeric type.
7	FloatingPointError Raised when a floating point calculation fails.
8	ZeroDivisionError Raised when division or modulo by zero takes place for all numeric types.
9	AssertionError Raised in case of failure of the Assert statement.
10	AttributeError Raised in case of failure of attribute reference or assignment.
11	EOFError Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
12	ImportError Raised when an import statement fails.
13	KeyboardInterrupt Raised when the user interrupts program execution, usually by pressing Ctrl+c.
14	LookupError Base class for all lookup errors.
15	IndexError Raised when an index is not found in a sequence.
16	KeyError Raised when the specified key is not found in the dictionary.
17	NameError Raised when an identifier is not found in the local or global namespace.
18	UnboundLocalError

	Raised when trying to access a local variable in a function or method but no value has been assigned to it.
19	EnvironmentError Base class for all exceptions that occur outside the Python environment.
20	IOError Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
21	IOError Raised for operating system-related errors.
22	SyntaxError Raised when there is an error in Python syntax.
23	IndentationError Raised when indentation is not specified properly.
24	SystemError Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
25	SystemExit Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
26	TypeError Raised when an operation or function is attempted that is invalid for the specified data type.
27	ValueError Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
28	RuntimeError Raised when a generated error does not fall into any category.
29	NotImplementedError Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

Задаци са изузецима

0250 Примена изузетка на контролу температуре

Креирати код који ће реаговати на AssertionError изузетак исписивањем поруке о уносу температуре у Форенхајтовим степенима када пређе одређену границу левог опсега.

```
def farenhajt_u_celizijus(temperatura):
    assert (temperatura >= -20), "Temperatura je previsе niska!"
    return ((temperatura - 32) / 1.8)
```

```
print (farenhajt_u_celizijus(0))
print (int(farenhajt_u_celizijus(10.78)))
print (farenhajt_u_celizijus(-30))
print (farenhajt_u_celizijus(100))
```

0251 Примена изузетка на контролу уноса

Креирати код који ће реаговати унос вредности која није бројчана.

```
def metoda(x):
    print ("Argument", x, "ne sadrzi broj.")
```

```
def brojevi(podatak):
    try:
        return int(podatak)
    except ValueError:
        metoda(podatak)

brojevi("1234")
brojevi("xyz")
```

Питања и задаци за самосталан рад

Задаци

085 Креирати код употребом руковања изузецима који ће реаговати на унос температуре у Форенхајтима испод границе ловог опсега, али ће приказати резултат и последње линије кода.

086 Корисник треба да креира листу целих бројева. Написати код руковањем изузецима који ће реаговати ако се покуша унос вредности која није цео број.

087 Креирати кориснички интерфејс у облику менија са 4 опције. Руковање изузецима се активира ако се унесе број за ставку у менију који није у понуди.

088 Уредити код задатка 087 тако да се понавља унос вредности из менија све док се не унесе нека од дозвољених вредности.

Рекурзије

Увод у рекурзије

Рекурзивне функције су функције које позивају саме себе.

0252 Рекурзија као бесконачна петља

```
def main():
    poruka()

def poruka():
    print('Ovo je rekurzivna funkcija.')
    poruka()

main()
```

Као резултат добија се понављање поруке све док Пајтон не прекине њено понављање.

Функција poruka() позива саму себе и циклус извршавања функције се понавља.

Први проблем са овим начином креирања функција је што се рекурзивна функција не може зауставити и на овај начин личи на бесконачну петљу.

Да би се могла успешно користити, рекурзивна функција мора имати начин контролисања броја пута понављања.

Следећи пример је модификована верзија претходног примера, само што сада функција `poruka()` добија аргумент који одређује колико пута ће се порука исписати на екрану.

0253 Рекурзија са контролом прекида

```
def main():  
    poruka(5)  
  
def poruka(broj_ponavljanja):  
    if broj_ponavljanja > 0:  
        print('Ovo je rekurzivna funkcija.')  
        poruka(broj_ponavljanja - 1)
```

`main()`

Функција `poruka()` у себи садржи `if` исказ који контролише понављање поруке.

Све док је параметар `broj_ponavljanja` већи од 0, порука ће се приказати, функција позива саму себе али се вредност аргумента смањује.

Сваки пут када се функција `poruka()` позове, нова инстанца параметра `broj_ponavljanja` се креира у меморији и добија вредност која је за један мања од претходне.

То значи да ће де одиграти 6 позива функције `poruka()` и да ће у последњем позиву параметар `broj_ponavljanja` бити једнак 0 што значи да више услов за позивање функције није испуњен.

Функција `poruka()` се само једном позива из `main()` функције а остали позиви су када функција позива саму себе.

Број позива функције када позива саму себе се назива **дубина рекурзије**.

У овом примеру, дубина рекурзије је 5.

Када се изведе 6.позив функције `poruka()`, извршава се прва линија кода испод линије позива, а у овом случају нема линије кода.

То значи да се после извршавања 6.позива, контрола функције враћа на 5.позив функције.

Овакво понављање се извршава уназад до 1.позива функције.

Употреба рекурзија

Рекурзија никада није неопходна да би се решио некакав програмерски проблем.

Други начин решавања проблема је коришћењем програмерских петљи.

Иначе, рекурзивни алгоритми се мање користе од итеративних алгоритама пошто процес позивања функције захтева више радњи од стране рачунара (алокација меморије и смештање адреса локација у меморију где се контрола враћа после извршења функције).

Ове акције се називају **прекобројне** (*overhead*), одвијају се увек када се функција позове и оне се не реализују у случају петљи.

Али, неки проблеми се лакше решавају коришћењем рекурзија него петљи; генерално тамо где се петља брзо извршава, могуће је направити брз рекурзиван алгоритам.

Принцип употребе рекурзија: ако је проблем решив одмах без рекурзије, функција га решава; ако проблем не може бити решен одмах, онда га функција смањује на мање али сличне проблеме, па позива саму себе да би решила те мање проблеме.

Прво се идентификује најмање један случај у којем се проблем може решити без рекурзије, и то се назива **основни случај** (base case).

Друго, одреди се начин решавања проблема за све друге случајеве коришћењем рекурзије и то се назива **рекурзивни случај** (recursive case).

У овом кораку, увек се мора смањити проблем на мању верзију оригиналног проблема.

Смањивањем проблема са сваким рекурзивним позивом, основни случај ће бити у одређеном моменту достигнут и рекурзија ће се зауставити.

Рачунање факторијала коришћењем рекурзије

У математици, писање $n!$ означава факторијел броја n .

Факторијел ненегативног броја се може дефинисати:

Ако је $n = 0$ онда је $n! = 1$; ако је $n > 0$ онда је $n! = 1 \times 2 \times 3 \times \dots \times n$.

За употребу рекурзије довољно је уместо $n!$ користити функцију: `faktorijel(n)`

При дизајну рекурзивног алгорита који рачуна факторијел било којег броја, прво се идентификује основни случај, који је део израчунавања који се може решити без употребе рекурзије (ако је $n = 0$ онда је `faktorijel(n) = 1`).

Овиме се решава проблем када је n једнако 0, али како се решава проблем за n веће од 0?

То је рекурзиван случај, или део проблема који се решава коришћењем рекурзије.

Ако је $n > 0$ онда је `faktorijel(n) = n x faktorijel(n - 1)`

Сада позив рекурзији ради са смањеном верзијом проблема, $n - 1$.

Зато се рекурзивно правило може написати и овако:

Ако је $n = 0$ онда је `faktorijel(n) = 1`; ако је $n > 0$ онда је `faktorijel(n) = n x faktorijel(n - 1)`.

0254 Рачунање факторијала рекурзијом

```
def main():
    broj = int(input('Uneti nenegativan broj: '))
    x = faktorijal(broj)
    print('Faktorijal od', broj, 'je', x)
```

```
def faktorijal(n):
    if n == 0:
        return 1
    else:
        return n * faktorijal(n - 1)
```

```
main()
```

Команда `return` се неће одмах извршити, пошто се прво мора реализовати позив функције `faktorijal(n - 1)`.

Функција `faktorijal` саму себе позива пет пута, све док се параметар n не постави на 0.

Рекурзивни алгоритам мора смањити проблем са сваким рекурзивним позивом, а да би се добило решење рекурзија се мора у једном моменту зауставити и то на основном случају (пошто он не захтева рекурзију за своје решавање).

0255 Сумирање рекурзијом

```
def main():
    brojevi = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    moja_suma = opseg(brojevi, 2, 5)
    print('Suma izmedju pozicija 2 i pozicije 5 je', moja_suma)

def opseg(broj_lista, start, kraj):
    if start > kraj:
        return 0
    else:
        return broj_lista[start] + opseg(broj_lista, start + 1, kraj)

main()
```

У примеру је функција `opseg()` која користи рекурзију да би се сабрао опсег бројева унутар листе. Функција користи аргументе: `broj_lista` листу која садржи опсег вредности за сабирање, `start` целобројну вредност која је индекс почетног елемента у опсегу, `kraj` целобројну вредност која указује на индекс крајњег елемента у опсегу.

Основни случај функције је када параметар `start` је већи од `kraj` параметра.

Ако је то тачно, функција враћа вредност 0, иначе функција враћа исказ после наредбе `return`.

Тај исказ враћа збир `broj_lista[start]` и повратну вредност рекурзивног позива.

Приметити да је у позиву рекурзије, почетни елемент у опсегу `start + 1`.

Заправо, овај исказ враћа вредност првог елемента у опсегу сабран са збиром преосталих елемената у опсегу.

Задаци са рекурзијама

Израда лабораторијских вежби: време реализације 80 минута

0256 Множење рекурзијом

Написати програм коришћењем рекурзије који прима два аргумента и враћа њихов производ.

```
def main():
    x, y = unos_brojeva()
    print(mnozenje(x, y))

def unos_brojeva():
    prvi = int(input("Uneti mnozilac: "))
    drugi = int(input("Uneti mnozenik: "))
    return prvi, drugi

def mnozenje(prvi, drugi):
    if prvi == 1:
        return drugi
```



```

else:
    return drugi + mnozenje(prvi - 1, drugi)

```

```
main()
```

0257 Претрага највећег елемента рекурзијом

Написати програм коришћењем рекурзије који приказује само највећи елемент из листе.

```

def main():
    print(max_u_listi([4, 3, 5, 192.6, -2, 14]))

```

```

def max_u_listi(x):
    if len(x) == 1:
        return x[0]
    elif x[0] > x[1]:
        x.pop(1)
        return max_u_listi(x)
    else:
        x.pop(0)
        return max_u_listi(x)

```

```
main()
```

0258 Фибоначијев низ рекурзијом

Написати програм коришћењем рекурзије који приказује бројеве из Фибоначијевог низа.

```

def main():
    print('Prvih 10 brojeva u Fibonacijevom nizu su: ')
    for broj in range(1, 11):
        print(fib(broj))

```

```

def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

```

```
main()
```

Питања и задаци за самосталан рад

Задаци

089 Написати програм коришћењем рекурзије који добија аргумент n и приказује целе бројеве од 1 до n.

090 Написати програм коришћењем рекурзије који приказује одређени симбол онолико пута која

је линија по реду, све до аргумента n.

091 Написати програм коришћењем рекурзије који даје суму свих целих бројева од 1 до унете вредности.

Претраге и сортирање

Секвенцијално претраживање листа

Најједноставнији начин да се утврди да ли постоји тражени елемент у листи је испитивање сваког појединачног елемента из листе да ли је идентичан са траженим елементом.

Претрага се завршава када се пронађе по први пут тражени елемент (пронађе се његова позиција у листи) или када се утврди колико има у листи идентичних елемената са траженим елементом. Овакав поступак се назива секвенцијално претраживање.

Секвенцијално претраживање се обавља над листом елемената који не морају обавезно бити сортирани.

0259 Секвенцијална претрага

```
def main():
    lista = unos_liste()
    prikaz_liste(lista)
    element = unos_elementa_za_pretragu()
    pronadjen = pretraga(lista, element)
    prikaz_rezultata(lista, element, pronadjen)

def unos_liste():
    print("Uneti u listu samo cele brojeve.")
    A = []
    jos = "da"
    while jos == "da":
        x = int(input("Uneti zeljeni broj u listu:"))
        A.append(x)
        print("Da li treba uneti jos jedan element u listu? ")
        jos = input("Uneti da ili bilo sta drugo za ne: ")
    return A

def prikaz_liste(A):
    print(A)

def unos_elementa_za_pretragu():
    a = int(input("Koji element se trazi u listi? "))
    return a

def pretraga(A, a):
    for x in range(len(A)):
        if A[x] == a:
            return True
```

```

        return False

def prikaz_rezultata(A, a, ima):
    print("U listi ", str(A))
    print("trazen je element ", a)
    if ima:
        print("Trazeni element se nalazi u listi.")
    else:
        print("Trazeni element se ne nalazi u listi.")

main()

```

Приметити да се претрага траженог елемента по листи завршава пре достизања краја листе. Употребом секвенцијалне претраге се може прекинути претрага у било којем моменту претраге ако је постигнут услов за излазак из петље претраге. Овакав метод претраге није ефикасан јер се може десити да је на последњем месту у листи налази тражени елемент, што потенцијално повећава број потребних испитивања једнакости.

Број појављивања елемената у листи

Додатак у алгоритму секвенцијалне претраге је проналажење броја појављивања траженог елемента.

У овом алгоритму се увек долази до краја листе елемената пошто се мора открити број појављивања елемента.

0260 Секвенцијално бројање елемената

```

def pretraga(A, a):
    b = 0
    for x in range(len(A)):
        if A[x] == a:
            b += 1
    return b

```

Променљива `b` преноси у главни део кода број појављивања траженог броја у листи.

То значи да број појављивања може бити 0 или било који цео позитиван број.

Остатак функција се не мора мењати у односу на претходни пример, осим функције `prikaz_rezultata()`.

0261 Претрага елемента у сортираној листи

```

def pretraga(A, a) :
    for i in range(len(A)) :
        if A[i] == a :
            return True
        elif A[i] > a :
            return False
    return False

```

У примеру се испитују три могућности: 1) Тражени број постоји у листи 2) Сви бројеви су већи од траженог броја а тражени број није у листи 3) Сви бројеви су мањи од траженог броја а тражени број није у листи

0262 Претрага најмањег елемента у листи

```
def pretraga(A):
    najmanji = A[0]
    for i in range(1, len(A)) :
        if A[i] < najmanji :
            najmanji = A[i]
    return najmanji
```

Бинарна претрага сортиране листе

0263 Бинарна претрага

```
def main():
    lista = [-34, -15, -9, -1, 0, 4, 5, 10, 13, 189, 1020]
    element = unos_elementa_za_pretragu()
    pronadjen = binarna_pretraga(lista, element)
    prikaz_rezultata(lista, element, pronadjen)
```

```
def unos_elementa_za_pretragu():
    a = int(input("Koji element se trazi u listi? "))
    return a
```

```
def binarna_pretraga(A, a):
    nizi = 0
    visi = len(A) - 1
    while nizi <= visi:
        sredina = (visi + nizi) // 2
        if A[sredina] == a:
            return True
        elif a < A[sredina]:
            visi = sredina - 1
        else:
            nizi = sredina + 1

    return False
```

```
def prikaz_rezultata(A, a, ima):
    print("U listi ", str(A))
    print("trazen je element ", a)
    if ima:
        print("Trazeni element se nalazi u listi.")
    else:
```

```
print("Trazeni element se ne nalazi u listi.")
```

```
main()
```

После израчунавања позиције првог, последњег и средњег елемента, прво се испитује да ли је елемент на средњој позицији једнак траженом елементу.

Ако је то случај, одмах се враћа True тј пронађен је тражени елемент.

Ако није, рачуна се да ли је тражени елемент мањи од постојећег елемента на средњој позицији листе.

Ако јесте, модификује се вредност промењиве `visi` као (`visi = sredina - 1`); а ако није модификује се вредност промењиве `visi` као (`nizi = sredina + 1`).

У следећој итерацији петље, само ће се узети у обзир за претрагу део секвенце између овако модификованих позиција за `nizi` и `visi` промењиве.

На тај начин се смањује број потребних претрага да би се добио резултат претраге што повећава ефикасност кода.

Процес се понавља све док се елемент не пронађе или све док промењива `nizi` не постане већа од промењиве `visi`.

То ће се десити када не остане ни један елемент за обраду у листи, чиме се указује да тражени елемент није у датој листи.

Сортирање секвенце

Сортирање значи међусобно упоређивање свих елемената из листе и међусобна замена њихове позиције у листи према вредностима, у зависности од тога да ли се елементи сортирају од најмање према највишој вредности или обрнуто.

Из тог разлога, алгоритми за сортирање имају више пролазака кроз листу него алгоритми за претрагу.

Постоји више алгоритама за сортирање секвенци: метода избора, метода замене суседа, метода уметања, метода поделе.

Сортирање методом избора

0264 Метода избора у сортирању

```
def main():
    lista = unos_liste()
    prikaz_liste(lista)
    lista = sortiranje(lista)
    prikaz_liste(lista)

def unos_liste():
    print("Uneti u listu samo cele brojeve.")
    A = []
    jos = "da"
    while jos == "da":
        x = int(input("Uneti zeljeni broj u listu:"))
        A.append(x)
```

```

        print("Da li treba uneti jos jedan element u listu? ")
        jos = input("Uneti da ili bilo sta drugo za ne: ")
    return A

def prikaz_liste(A):
    print(A)

def sortiranje(A):
    n = len(A)
    for i in range (0, n - 1):
        for j in range (i + 1, n):
            if (A[i] > A[j]):
                b = A[i]
                A[i] = A[j]
                A[j] = b

    return A

main()

```

Метода избора се заснива на упоређивању вредности елемената из две петље.

Спољна петља (од 1. до претпоследњег елемента) упоређује сваки елемент, осим последњег, са свим осталим елементима унутрашње петље (од 2. до последњег елемента).

На тај начин се неће упоређивати елементи на истим позицијама.

Ако је утврђено да је елемент спољне петље већи од елемента унутрашње петље, они замењују места.

Тако се обезбеђује да индекс мањег елемента буде мањи од индекса већег елемента а то је сортирање од мањег елемента ка већим.

Сортирање методом замене суседа

0265 Метода замене суседа у сортирању

```

def sortiranje(A):
    n = len(A)
    for i in range(0, n):
        for j in range(1, n - i):
            if A[j - 1] > A[j]:
                b = A[j]
                A[j] = A[j - 1]
                A[j - 1] = b

    return A

```

У овој методи се спољна петља користи за обезбеђивање n итерација над секвенцом.

Унутрашња петља се користи за избор елемената који се упоређују.

Увек се упоређују два суседна елемента по позицији у секвенци.

Ако је елемент са мањом позицијом већи од елемента са већом позицијом, врши се замена позиција.

Упоредба се врши са смањеним бројем елемената како се приближава крају секвенце.

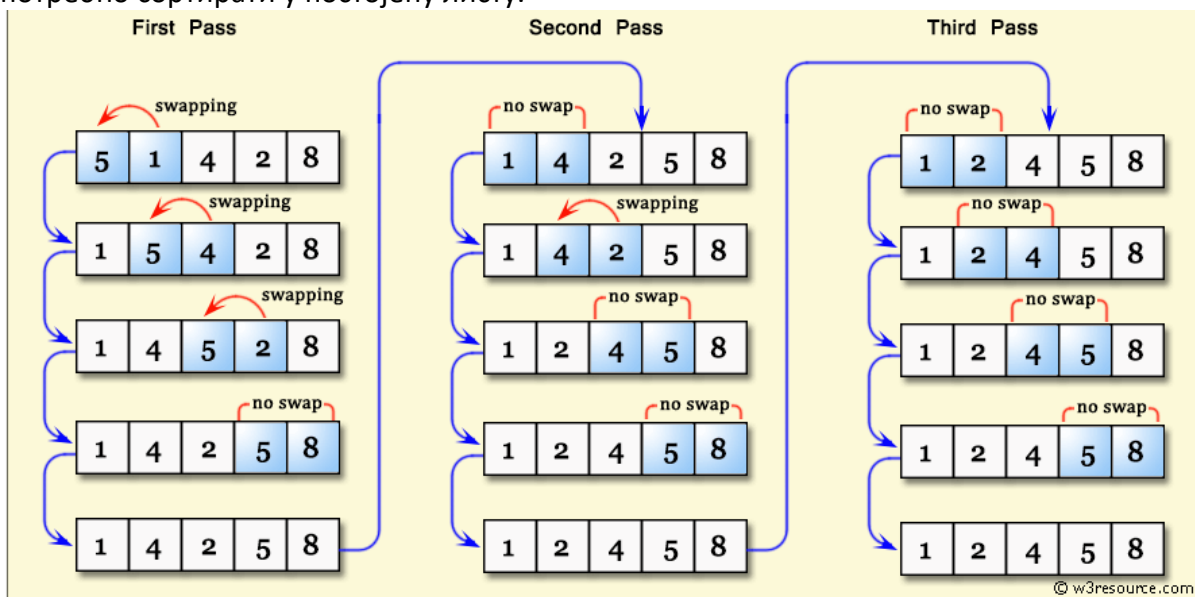
Bubble-sort

Ова метода се заснива на понављању корака сортирања елемената у листи помоћу упоређивања сваког пара суседних вредности и њиховој замени места ако им је редослед погрешан.

Пролазак кроз листу се понавља све док се више замена неће извршити, што значи да је листа сортирана.

Сам алгоритам је доста спор и непрактичан за решавање реалних проблема чак и ако се упоређује са методом избора.

Користи се код сортираних листа којима се могу придодавати појединачни елементи које је потребно сортирати у постојећу листу.



0266 Метода сортирања bubble-sort

```
def bubble_sort(A):  
    for x in range(len(A) - 1, 0, -1):  
        for i in range(x):  
            if A[i] > A[i + 1]:  
                y = A[i]  
                A[i] = A[i + 1]  
                A[i + 1] = y
```

```
A = [14,46,43,27,57,41,45,21,70]  
bubble_sort(A)  
print(A)
```

Види се у коду да постоје две петље: спољна са x бројачем и унутрашња са i бројачем.

Десна граница опсега унутрашње петље зависи од тренутне вредности бројача x и увек је мања од индекса последњег елемента у листи A .

У првом циклусу ($x = 0$, $i = 0$, 7) се уређују позиције за парове елемената који се по први пут упоређују и ефекат је да се елементи са већим вредностима почињу померати удесно у листи.

То значи да ће највећи елемент у листи сигурно бити највећи у било којем пару упоређиваних елемената па ће сигурно бити и постављен на последње место у листи.

Из тог разлога нема потребе више, у следећим циклусима, упоређивати елемент на позицији која је у тренутном циклусу правилно попуњена, па се у сваком следећем циклусу опсег унутрашње петље смањује за један.

Задаци претраге и сортирања

0267 Секвенцијално претраживање броја у листи

Написати програм који користи секвенцијално претраживање у несортираној листи $[1, 4, 5, 0]$ да би се пронашао број 0 у листи. Приказати одговарајућу поруку о проналажењу и позицији елемента 0 у листи.

```
def main():
    lista = [1, 4, 5, 0]
    prikaz_liste(lista)
    element = unos_elementa_za_pretragu()
    pronadjen, pozicija = pretraga(lista, element)
    prikaz_rezultata(lista, element, pronadjen, pozicija)

def prikaz_liste(A):
    print(A)

def unos_elementa_za_pretragu():
    a = int(input("Koji element se trazi u listi? "))
    return a

def pretraga(A, a):
    for x in range(len(A)):
        if A[x] == a:
            return True, x
    return False, None

def prikaz_rezultata(A, a, ima, poz):
    print("U listi ", str(A))
    print("trazen je element ", a)
    if ima:
        print("Trazeni element se nalazi u listi.")
        print("Element je pronadjen na poziciji", str(poz))
    else:
        print("Trazeni element se ne nalazi u listi.")

main()
```


0268 Бинарно претраживање броја у листи

Написати програм који користи бинарно претраживање у сортираној листи [0, 1, 4, 15, 23, 99] да би се детектовало да ли постоји елемент 23 у листи. Приказати одговарајућу поруку о проналажењу и позицији елемента 23 у листи.

```
def main():
    lista = [0, 1, 4, 15, 23, 99]
    prikaz_liste(lista)
    pronadjen, pozicija = binarna_pretraga(lista)
    prikaz_rezultata(lista, pronadjen, pozicija)

def prikaz_liste(A):
    print(A)

def binarna_pretraga(A):
    nizi = 0
    visi = len(A) - 1
    while nizi <= visi:
        sredina = (visi + nizi) // 2
        if A[sredina] == 23:
            return True, sredina
        elif 23 < A[sredina]:
            visi = sredina - 1
        else:
            nizi = sredina + 1
    return False, None

def prikaz_rezultata(A, ima, poz):
    print("U listi ", str(A))
    print("trazen je broj ", 23)
    if ima:
        print("Broj 23 se nalazi u listi.")
        print("Broj 23 se nalazi na poziciji", poz)
    else:
        print("Broj 23 se ne nalazi u listi.")

main()
```

0269 Детекција појаве броја у листи

Написати програм који користи бинарно претраживање у сортираној листи да би се детектовало да ли постоји тражени елемент у листи. Приказати одговарајућу поруку о проналажењу и позицији траженог елемента у листи.

```
def main():
    lista = unos_liste()
    prikaz_liste(lista)
    element = unos_elementa_za_pretragu()
```

```

    pronadjen, pozicija = binarna_pretraga(lista, element)
    prikaz_rezultata(lista, element, pronadjen, pozicija)

def unos_liste():
    print("Uneti u listu samo cele brojeve.")
    A = []
    jos = "da"
    while jos == "da":
        x = int(input("Uneti zeljeni broj u listu:"))
        A.append(x)
        print("Da li treba uneti jos jedan element u listu? ")
        jos = input("Uneti da ili bilo sta drugo za ne: ")
    return A

def prikaz_liste(A):
    print(A)

def unos_elementa_za_pretragu():
    a = int(input("Koji element se trazi u listi? "))
    return a

def binarna_pretraga(A, a):
    nizi = 0
    visi = len(A) - 1
    while nizi <= visi:
        sredina = (visi + nizi) // 2
        if A[sredina] == a:
            return True, sredina
        elif a < A[sredina]:
            visi = sredina - 1
        else:
            nizi = sredina + 1
    return False, None

def prikaz_rezultata(A, a, ima, poz):
    print("U listi ", str(A))
    print("trazen je element ", a)
    if ima:
        print("Trazeni element se nalazi u listi.")
        print("Element se nalazi na poziciji", poz)
    else:
        print("Trazeni element se ne nalazi u listi.")

main()

```

Написати програм који користи методу избора да би сортирао од највећег ка најмањем, целе бројеве унутар листе [55, -34, 5, 0, 99, 23] .

```
def prikaz_liste(A):
    print(A)

def sortiranje(A):
    n = len(A)
    for i in range (0, n - 1):
        for j in range (i + 1, n):
            if (A[i] < A[j]):
                b = A[i]
                A[i] = A[j]
                A[j] = b
    return A

def main():
    lista = [55, -34, 5, 0, 99, 23]
    prikaz_liste(lista)
    lista = sortiranje(lista)
    prikaz_liste(lista)

main()
```

0271 Сортирање листе методом bubble-sort

Написати програм који користи методу bubble-sort да би сортирао од најмањег ка највећем, целе бројеве унутар листе [38, 27, 43, 3, 9, 82, 10].

```
def bubble_sort(A):
    for x in range(len(A) - 1, 0, -1):
        for i in range(x):
            if A[i] > A[i + 1]:
                y = A[i]
                A[i] = A[i + 1]
                A[i + 1] = y

A = [38, 27, 43, 3, 9, 82, 10]
bubble_sort(A)
print(A)
```

Питања и задаци за самосталан рад

Задаци

092 Написати програм који користи секвенцијално претраживање у несортираној листи [0, 4, 0, 0] да би се детектовало колико има елемената 0 у листи. Приказати одговарајућу поруку после проналажења свих елемената 0 у листи.

093 Написати програм који користи методу избора да би сортирао од најмањег ка највећем, целе

бројеве унутар листе.

094 Написати програм који користи методу bubble-sort да би сортирао од највећег ка најмањем, целе бројеве унутар листе користећи функције за решавање појединачних корака алгорита.

Коришћена литература

- Tony Gaddis “Starting Out with Python”
- Allen B.Downey “Think Python”
- Laura Cassel, Alan Gauld “Python Projects”
- Magnus Lie Hetland “Python Algorithms”
- John V. Guttag “Introduction to Computation and Programming using Python”
- David Beazley, Brian K. Jones “Python Cookbook”
- Charles R. Severance “Python for Everybody”
- python.org
- flowgorithm.org, Devin Cook
- realpython.com
- w3schools.com
- learnpython.org
- coursera.org