საქართველოს ტექნიკური უნივერსიტეტი

გულნარა ჯანელიძე

Python
დაპროგრამების ენა

(მეთოდური მითითებები ლაბორატორიული სამუშაოების შესასრულებლად)

თბილისი
2019

უაკ 681.3.06

წიგნში წარმოდგენილია ლაბორატორიული სამუშაოების შესრულების აღწერილობა Python დაპროგრამების ენაზე. ლაბორატორიული სამუშაო მოიცავს, როგორც დასამუშავებელ მასალას, ასევე დამოუკიდებელ დავალებებს. განკუთვნილია როგორც ინფორმატიკის, ასევე სხვა სფეროს სტუდენტებისთვის.

ISBN 978-9941-8-1695-6

http//www.gtu.ge

შინაარსი

ზოგადი მითითებები

ლაბორატორიული სამუშაოების შესრულების თანამიმდევრობა

ლაბორატორიული ციკლი შედგება 15 თემისგან. ლაბორატორიული სამუშაოს მიზანია python გარემოში დაპროგრამების შესწავლა.

ლაბორატორიული სამუშაოების ციკლის დაწყებამდე, საგნის სილაბუსის საშუალებით, სტუდენტი ეცნობა სამუშაოთა ჩამონათვალს და ძირითად მიზანს. ყოველ კომპიუტერზე იქმნება მოცემული ჯგუფის საქაღალდე და მასში სტუდენტის ინდივიდუალური საქაღალდე, სადაც განთავსდება მიღებული შედეგების ფაილები (Doc გაფართოებით) ანუ ელექტრონული ანგარიშები, რომლებიც სემესტრის ბოლოს გადაიტანება CD-ზე.

წინასწარი მომზადება

თითოეული ლაბორატორიული სამუშაოს აღწერა შედგება შემდეგი ნაწილებისგან:

- სამუშაოს მიზანი;
- დავალება;
- დავალების შესაბამისი პროგრამული კოდი python ენაზე;
- მიღებული შედეგის Screenshot.

მოცემული თემები უნდა იქნას შესწავლილი ლაბორატორიული სამუშაოს დაწყებამდე.

შესასრულებელი დავალება

ყოველ ლაბორატორიულ სამუშაოში შედის შესასრულებელი დავალება, რომელიც ფორმულირებულია ნაწილში „დავალება“. სტუდენტი წინასწარ, დამოუკიდებლად უნდა გაეცნოს თავის დავალებას და თუ მას გაუჩნდა შეკითხვები დავალებასთან მიმართებით, კითხვები უნდა დასვას სამუშაოს დაწყებამდე.

დავალების შესრულება

სტუდენტი დავალებებს შეასრულებს ლაბორატორიულ სამუშაოში წარმოდგენილი პრაქტიკული მაგალითების საფუძველზე.

მიღებული შედეგების ეკრანზე წარმოდგენის შემდეგ ხდება ტექსტური ფაილების სახით მათი ასლების შექმნა და შენახვა სტუდენტის ინდივიდუალურ საქაღალდეში მოცემული თარიღის მითითებით.

ლაბორატორიული სამუშაოს ანგარიში

ლაბორატორიული სამუშაოს ელექტრონული ანგარიში უნდა შეიცავდეს შემდეგ პუნქტებს:

- ლაბორატორიული სამუშაოს თემა;
- შესასრულებელი დავალებები;
- ტექსტურ ფაილში გადატანილი კოდები და მიღებული შედეგების ასლები Screenshot -ების სახით.

ლაბორატორიული სამუშაო 1
თემა: python გარემო, ენის სინტაქსი, რიცხვებთან მუშაობა

დასამუშავებელი საკითხები:

- IDLE- დამუშავების გარემო;
- ენის სინტაქსი;
- შეტანა, გამოტანა, კომენტარი;
- რიცხვებთან მუშაობა;
- რიცხვების გარდაქმნის ფუნქციები.

IDLE- დამუშავების გარემო

Python სისტემის ინსტალაციის შემდეგ გახსენით IDLE (Integrated Development and Learning Environment). გაიხნება ინტერაქტული რეჟიმი, დავწეროთ პირველი პროგრამა: "hello world".

```
print("Hello world!")
```

შეიტანეთ კოდი IDLE ფანჯარაში და დააჭირეთ Enter ღილაკს. შედეგი გამოვა ფანჯარაში:

შეიტანეთ შემდეგი სტრიქონები და მიიღეთ შედეგები:

```
print(3 + 4)  #7
print(3 * 5)  #15
print(3 ** 2)  #9
```

პროგრამის ფაილის სახით შესანახად ინტერაქტულ რეჟიმში აირჩიეთ File → New File ბრძანება და დააჭირეთ Ctrl + N კომბინაციას.

გახსნილ ფანჯარაში შეიტანეთ შემდეგი კოდი:

```
name = input("What is your name? ")
print("Hello,", name)
```

დააჭირეთ F5 ღილაკს ან აირჩიეთ Run → Run Module ბრძანება. პროგრამის გაშვებამდე სისტემა შემოგთავაზებთ ფაილის შენახვას. შეინახეთ ფაილი სასურველ ადგილას და ამის შემდეგ პროგრამა გაეშვება.

შედეგი:

```
What is your name? anna
Hello, anna
```

ენის სინტაქსი

დაწერეთ რამდენიმე ინსტრუქცია ერთ სტრიქონში, გამოყოფილი წერტილ-მძიმეებით:

```
a = 1; b = 2; print(a, b)
```

თუმცა ამის გაკეთება ხშირად არასასურველია.

შესაძლებელია ერთი ინსტრუქციის რამდენიმე სტრიქონად დაწერა:

```
if (a == 1 and b == 2 and
c == 3 and d == 4):  # აქ ორიწერტილი უნდა
    print('spam' * 3)
```

რთული ინსტრუქციის ტანი განათავსეთ ძირითადი ინსტრუქციის სტრიქონშივე:

```
if x > y: print(x)
```

შეტანა, გამოტანა, კომენტარი

>>>a=5

a

გამოიტანს: 5

განვიხილოთ მაგალითები:

#ეს პირველი კომენტარია

abc=3   #ეს მეორე კომენტარია

#... ეს მესამე კომენტარია

STRING="# ეს არ არის კომენტარი"

რიცხვები

მაგალითები:

>>> 2+2

4

>>> # ეს კომენტარია

... 2+2

4

>>> 2+2 # კომენტარი კოდის სტრიქონშივეა

4

>>> (50-5*6)/4

5

>>> # მთელრიცხვა გაყოფისას შედეგი მრგვალდება

... # ნაკლებობით

... 7/3

2

```
>>> 7/-3
-3
```
ორი რიცხვის შეკრება:
```
print(6 + 2)  # 8
```
ორი რიცხვის გამოკლება:
```
print(6 - 2)  # 4
```
ორი რიცხვის გამრავლება:
```
print(6 * 2)  # 12
```
ორი რიცხვის გაყოფა:
```
print(6 / 2)  # 3.0
```
ორი რიცხვის მთელრიცხვა გაყოფა:
```
print(7 // 2)  # 3
```
ახარისხება:
```
print(6 ** 2)  # 36
```
ნაშთის მიღება გაყოფის შედეგად:
```
print(7 % 2)  # 1
```
გამოსახულების ამოხსნა:
```
number = 3 + 4 * 5 ** 2 + 7
print(number)  # 110
```
წინა მაგალითში მოქმედებების მიმდევრობის გადასაწყობად გამოყენებულია ფრჩხილები:
```
number = (3 + 4) * (5 ** 2 + 7)
print(number)  # 224
```
მინიჭების ოპერატორის (=) გამოყენება:
```
>>> width = 20
>>> height = 5*9
>>> width * height
```

900

მნიშვნელობის მინიჭება ერთდროულად რამდენიმე ცვლადზე:

```
>>> x = y = z = 0 # x, y და z ცვლადებს  მიენიჭებათ 0
>>> x
0
>>> y
0
>>> z
0
```

მინიჭების სპეციალური ოპერაციები:

+= შეკრების შედეგის მინიჭება;

-= გამოკლების შედეგის მინიჭება;

*= გამრავლების შედეგის მინიჭება;

/= გაყოფის შედეგის მინიჭება;

//= მთელრიცხვა გაყოფის შედეგის მინიჭება;

**=რიცხვის ხარისხის მინიჭება;

%= გაყოფისგან მიღებული ნაშთის მინიჭება.

მაგალითები:

```
number = 10
number += 5
print(number)  # 15
 number -= 3
print(number)  # 12
 number *= 4
print(number)  # 48
```

რიცხვების გარდაქმნის ფუნქციები

ფუნქციები: int() და float()

მოცემული კოდის შესრულებისას გენერირდება შეცდომა, რადგან პირველი რიცხვი წარმოადგენს სტრიქონს.:

```
first_number = "2"
second_number = 3
third_number = first_number + second_number
```

შეცდომის აღმოფხვრისთვის სტრიქონი გარდავქმნათ რიცხვად int() ფუნქციის დახმარებით:

```
first_number = "2"
second_number = 3
third_number = int(first_number) + second_number
print(third_number) # 5
```

გამოვიყენოთ float() ფუნქცია, მცურავწერტილიან რიცხვად გარდაქმნისთვის:

```
first_number = 2.0001
second_number = 5
third_number = first_number / second_number
print(third_number) # 0.40002000000000004
```

შედეგის დასამრგვალებლად გამოვიყენოთ round() ფუნქცია:

```
first_number = 2.0001
second_number = 0.1
third_number = first_number + second_number
print(round(third_number, 4))  # 2.1001
```

მაგალითში ოპერატორები შეიცავენ შერეული ტიპების ოპერანდებს, შედეგად მიიღება მცურავწერტილიანი ტიპი:

```
>>> 4 * 2.5 / 3.3
3.0303030303030303
>>> 7.0 / 2
3.5
```

კომპლექსური რიცხვი: a + bi ტიპის გამოსახულებაა, a — ნამდვილი რიცხვია, ხოლო b — წარმოსახვითი ნაწილი.

კომპლექსური რიცხვის მიღება:

```
>>> x = complex(1, 2)
>>> print(x)
(1+2j)
>>> y = complex(3, 4)
>>> print(y)
(3+4j)
```

z კომპლექსური რიცხვიდან ნაწილების ამოღება: z.real და z.imag გამოყენებით:

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

მაგალითი უჩვენებს, რომ მთელი და მცურავწერტილიანი რიცხვების გარდაქმნის ფუნქციები (int(), long() და float()) არ მუშაობენ კომპლექსური რიცხვებისთვის:

```
>>> a=1.5+0.5j
```

```
>>> float(a)
Traceback (innermost last):
File "<stdin>", line 1, in ?
TypeError: can't convert complex to float; use e.g.
>>> a.real
1.5
```

Python გამოყენება სამაგიდო კალკულატორის სახით:

```
>>> tax = 17.5 / 100
>>> price = 3.50
>>> price * tax
0.61249999999999993
```

როგორც მაგალითიდან ჩანს, ბოლო გამოტანილი მნიშვნელობა შეინახულ იქნა ცვლადში.

დავალება 1

კვალიატურიდან შეიტანეთ დარიცხული თანხა, გამოითვალეთ საშემოსავლო დაბეგვრა - დარიცხულის 20%, საპენსიო ფონდისთვის დაბეგრილი - დარიცხულის 2%, სულ დაბეგრილი თანხა და ხელზე გასაცემი თანხა. ყველა მიღებული მნიშვნელობა გამოიტანეთ კონსოლზე.

დავალება 2

მოცემული რიცხვებიდან (7, 9) მიიღეთ კომპლექსური რიცხვი და გამოიტანეთ კონსოლზე.

დავალება 3

მოცემულია კომპლექსური რიცხვი: a=3.7+5.9j ამოიღეთ ამ რიცხვიდან ნამდვილი და წარმოსახვითი ნაწილები.

ლაბორატორიული სამუშაო 2

თემა: მმართველი ოპერატორები

დასამუშავებელი საკითხები:

- if-elif-else ინსტრუქცია;
- While ციკლი;
- for ციკლი;
- continue ოპერატორი;
- ჩადგმული ციკლები.

if-elif-else ინსტრუქცია

if-elif-else ინსტრუქციის ზოგადი სახე:

```
if test1:
   state1
elif test2:
   state2
else:
   state3
```

მაგალითი ბეჭდავს 'true' -ს რადგან 1 არის ჭეშმარიტი:

```
>>> if 1:
...     print('true')
... else:
...     print('false')
...
true
```

მაგალითში შედეგი დამოკიდებულია იმაზე, თუ რა შეიტანა მომხმარებელმა:

```
a = int(input())
```

```
if a < -5:
   print('Low')
elif -5 <= a <= 5:
   print('Mid')
else:
   print('High')
```

ლოგიკური ოპერატორები:

მაგალითში X მნიშვნელობიდან გამომდინარე A -ს მიენიჭება Y ან Z, მოცემული რიცხვებისთვის A--ს მიენიჭება -3:

```
X=1
Y=3
Z=7
if  X:
   A = Y
else:
   A = Z
print(A)
```

იგივე ინსტრუქცია, შეიძლება დაიწეროს უფრო მოკლედ:

```
A = Y if X else Z
```

განვიხილოთ კიდევ ერთი მაგალითი:

```
>>> x = int(input("შეიტანეთ რიცხვი: "))
>>> if x < 0:
... x = 0
... print (’უარყოფითს ვცვლით 0-ით’)
... elif x == 0:
... print (’ნული’)
```

... elif x == 1:

... print ('ერთი')

... else:

... print ('მეტი')

შედარების ოპერაციის მაგალითები:

```
a = 5
b = 6
result = 5 == 6  # ვინახავთ ოპერაციის შედეგს ცვლადში
print(result)  # False - 5 არ უდრის 6
print(a != b)  # True
print(a > b)  # False - 5 ნაკლებია 6
print(a < b)  # True

bool1 = True
bool2 = False
print(bool1 == bool2)  # False - bool1 არ უდრის bool2
```

ლოგიკური ოპერაციის მაგალითები:

and (ლოგიკური გამრავლება)

ა) age = 22

weight = 58

result = age > 21 and weight == 58

print(result) # True

ბ) age = 22

weight = 58

isMarried = False

result = age > 21 and weight == 58 and isMarried

print(result) # False, რადგან isMarried = False

or (ლოგიკური შეკრება)

```
age = 22
isMarried = False
result = age > 21 or isMarried
print(result)  # True, რადგან  age > 21 ტოლია True
```

not (ლოგიკური უარყოფა):

```
age = 22
isMarried = False
print(not age > 21)  # False
print(not isMarried)  # True
```

მოქმედებების მიმდევრობის შეცვლისთვის მრგვალი ფრჩხილების გამოყენება:

```
age = 22
isMarried = False
weight = 58
result = (weight == 58 or isMarried) and not age > 21  # False
print(result)
```

While ციკლი

ციკლის სინტაქსი:

```
while პირობა_გამოსახულება:
  ინსტრუქციები
```

კოდი ბეჭდავს 5-დან 15-მდე კენტ რიცხვებს:

```
i = 5
>>> while i < 15:
...     print(i)
...     i = i + 2
...
5
```

7
9
11
13

ფიბონაჩის რიცხვების გამოთვლის პროგრამული კოდი:

```
>>> # ფიბონაჩის მწკრივი:
... # ორი წინა რიცხვის ჯამი განსაზღვრავს
... # მომდევნოს
... a, b = 0, 1
>>> while b < 10:
... print (b)
... a, b = b, a+b
...
1
1
2
3
5
8
```

განვიხილოთ პროგრამა, რომელიც ითვლის რიცხვის ფაქტორიალს:

```
number = int(input("შეიტანეთ რიცხვი: "))
i = 1
factorial = 1
while i <= number:
   factorial *= i
   i += 1
print("რიცხვის ", number, "ფაქტორიალი ტოლია", factorial)
```

შედეგი:

შეიტანეთ რიცხვი: 5

რიცხვის 5 ფაქტორიალი ტოლია 120

for ციკლი

ციკლის ფორმალური განსაზღვრა:

```
for int_var in ფუნქცია_range:
   ინსტრუქციები
```

ფაქტორიალის გამოთვლის მაგალითი:

```
number = int(input("შეიტანეთ რიცხვი: "))
factorial = 1
for i in range(1, number+1):
   factorial *= i
print("ფაქტორიალი", number, "-ის ტოლია", factorial)
```

range ფუნქციის ფორმები:

- range(stop) - აბრუნებს ყველა მთელ რიცხვს 0-დან stop-მდე;
- range(start, stop) - აბრუნებს ყველა მთელ რიცხვს შუალედში: start-დან (ჩათვლით) stop-მდე (არ ჩათვლით). ფაქტორიალის გამოთვლის მაგალითში სწორედ ეს ფორმა იყო გამოყენებული;
- range(start, stop, step) - აბრუნებს მთელ რიცხვებს შუალედში: start-დან (ჩათვლით) stop-მდე (არ ჩათვლით), რომლებიც იზრდებიან step მნიშვნელობით.

range ფუნქციის გამოყენების მაგალითები:

```
range(5)          # 0, 1, 2, 3, 4
```

```
range(1, 5)        # 1, 2, 3, 4
range(2, 10, 2)    # 2, 4, 6, 8
  range(5, 0, -1)    # 5, 4, 3, 2, 1
```

გამოვიტანოთ მიმდევრობით ყველა რიცხვი 0-დან 4-მდე:

```
for i in range(5):
   print(i, end=" ")
```

მაგალითში ციკლი ამოარჩევს სიიდან ელემენტებს მიმდევრობით:

```
#რამდენიმე სტრიქონის სიგრძის განსაზღვრა:
a = ['კატა', 'სპილო', 'კურდღელი']
for x in a:
   print (x, len(x))
```

შედეგი:

კატა 4
სპილო 5
კურდღელი 8

continue ოპერატორი

მაგალითში ‘o’ ასოს შემთხვევაში ციკლის ტანი არ შესრულდება:

```
for i in "hello world":
   if i=="o":
      continue
   print(i, end='')
```

შედეგი:

```
hell wrld
```

break ოპერატორი

მაგალითში ციკლი შეწყვეტს, მუშაობას 'O' სიმბოლოსთან:

```
for i in 'hello world':
...     if i == 'o':
...         break
...     print(i * 2, end='')
```

შედეგი:
hheellll

მოცემულ მაგალითში ციკლში გამოყენებულია else. ინსტრუქციის ბლოკი else-ის შიგნით შესრულდება მხოლოდ იმ შემთხვევაში, თუ ციკლიდან გასვლა მოხდა break-ის გარეშე:

```
>>> for i in 'hello world':
...     if i == 'a':
...         break
... else:
...     print('a ასო არ არის სტრიქონში')
```

a ასო არ არის სტრიქონში

განვიხილოთ მაგალითი, ვალუტის გამცვლელი პუნქტი:

```
print("დასრულებისთვის დააჭირეთ Y")
 while True:
    data = input("შეიტანეთ თანხა გაცვლისთვის: ")
    if data.lower() == "y":
        break  # ციკლიდან გასვლა
    money = int(data)
```

```
    cache = round(money / 2.56)
    print("დაიცემა", cache, "დოლარი")
print("გაცვლითი პუნქტის მუშაობა დასრულებულია")
```

შედეგი:

```
== RESTART: C:/Users/Guliko/AppData/Local/Programs/
დასრულებისთვის დააჭირეთ Y
შეიტანეთ თანხა გაცვლისთვის: 100
დაიცემა 39.06 დოლარი
შეიტანეთ თანხა გაცვლისთვის: 200
დაიცემა 78.12 დოლარი
შეიტანეთ თანხა გაცვლისთვის: 450
დაიცემა 175.78 დოლარი
შეიტანეთ თანხა გაცვლისთვის: y
გაცვლითი პუნქტის მუშაობა დასრულებულია
>>>
```

ჩადგმული ციკლები

განვიხილოთ გამრავლების ცხრილის მაგალითი:

```
for i in range(1, 10):
    for j in range(1, 10):
        print(i * j, end="\t")
    print("\n")
```

შიდა ციკლის ცალკეულ იტერაციაში კონსოლზე გამოიტანება i და j რიცხვების ნამრავლი. მივიღებთ შედეგს:

```
== RESTART: C:/Users/Guliko/AppData/Local/Programs/Python/Python36-32/bv.py
1       2       3       4       5       6       7       8       9

2       4       6       8       10      12      14      16      18

3       6       9       12      15      18      21      24      27

4       8       12      16      20      24      28      32      36

5       10      15      20      25      30      35      40      45

6       12      18      24      30      36      42      48      54

7       14      21      28      35      42      49      56      63

8       16      24      32      40      48      56      64      72

9       18      27      36      45      54      63      72      81
```

დავალება 1

კლაიატურიდან შეიტანეთ სამი რიცხვი, შეამოწმეთ ამ რიცხვებით აიგება თუ არა სამკუთხედი. თუ აიგება, გამოითვალეთ მისი ფართობი და თუ არ აიგება, კონსოლზე გამოიტანეთ შესაბამისი შეტყობინება.

დავალება 2

ციკლის გამოყენებით ამობეჭდეთ 2-დან 20-მდე ლუწი რიცხვები ზრდის მიხედვით

დავალება 3

ციკლის გამოყენებით ამობეჭდეთ 20-დან 2-მდე ლუწი რიცხვები კლების მიხედვით.

დავალება 4

კლავიატურიდან შეიტანეთ სტრიქონი: ,,information", პროგრამამ სტრიქონიდან ამობეჭდოს ყველა სიმბოლო ,,i" სიმბოლოს გარდა.

დავალება 5

კლავიატურიდან შეიტანეთ სტრიქონი: ,,information", პროგრამამ სტრიქონიდან ამობეჭდოს ყველა სიმბოლო ,,a" სიმბოლომდე.

დავალება 6

ჩადგმული ციკლების გამოყენებით მიიღეთ შემდეგი ცხრილი:

1	3	5	7	9
3	9	15	21	27
5	15	25	35	45
7	21	35	49	63
9	27	45	63	81

ლაბორტორიული სამუშაო 3
თემა: სტრიქონები

დასამუშავებელი საკითხები:

- სტრიქონის განსაზღვრა;
- ოპერაციები სტრიქონებზე;
- ფუნქციები და მეთოდები სტრიქონებთან სამუშაოდ.

სტრიქონის ტიპის განსაზღვრა

>>> 'spam eggs'

spam eggs

>>> 'doesn\'t'

doesn't

>>> "doesn't"

doesn't

>>> '"Yes," he said.'

"Yes," he said.

>>> "\"Yes,\" he said."

"Yes," he said.

>>> '"Isn\'t," she said.'

"Isn\'t," she said.

გრძელი სტრიქონული გამოსახულების დაყოფა რამდენიმე სტრიქონად:

hello = "ეს გრძელი სტრიქონული გამოსახულებაა, რომელიც შეიცავს \n\
რამდენიმე სტრიქონს\n\"

print (hello)

შედეგი იქნება:

ეს გრძელი სტრიქონული გამოსახულებაა, რომელიც შეიცავს

რამდენიმე სტრიქონს

სხვანაირად, ტექსტი შეიძლება ჩაისვას სამმაგ ბრჭყალებში: " " " ან ’ ’ ’.

```
print ("""
Usage: thingy [OPTIONS]
-h                    Display this usage message
-H hostname        Hostname to connect to
""")
```

გამოიტანს შედეგს:

```
Usage: thingy [OPTIONS]
-h                    Display this usage message
-H hostname       Hostname to connect to
```

სტრიქონების გაერთიანება და გამრავლება:

```
>>> word = ’Help’ + ’A’
>>> word
HelpA
>>> ’<’ + word*5 + ’>’
<HelpAHelpAHelpAHelpAHelpA>
```

ერთმანეთის გვერდით ჩაწერილი ორი სტრიქონის გაერთიანება:

```
word='help''a'
```

შედეგი:

```
'helpa'
```

ეს მეთოდი მუშაობს მხოლოდ ერთმანეთის გვერდით დაწერილი სტრიქონებისთვის:

```
>>> ’str’ ’ing’ # სწორია
’string’
```

```
>>> 'str'.strip() + 'ing' # სწორია
'string'
>>> 'str'.strip() 'ing' # შეცდომაა
File "<stdin>", line 1
'str'.strip() 'ing'
^
SyntaxError: invalid syntax
```

სტრიქონის ნებისმიერი სიმბოლოს ამოღება და ჭრით ქვესტრიქონის მიღება:

```
>>> word[4]
'A'
>>> word[0:2]
'He'
>>> word[2:4]
'lp'
```

სტრიქონში სიმბოლოს შეცვლის მცდელობა განსაზღვრულ პოზიციაში ან ქვესტრიქონში გამოიწვევს შეცდომას:

```
>>> word[0] = 'x'
Traceback (innermost last):
File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
>>> word[:-1] = 'Splat'
Traceback (innermost last):
File "<stdin>", line 1, in ?
TypeError: object doesn't support slice assignment
```

ჭრის ინდექსების გამოტოვება:

```
>>> word[ :2] # პირველი-ორი სიმბოლო
```

’He’

>>> word[2:] # მთელი სტრიქონი გარდა პირველი ორი სიმბოლოსა

’lpA’

ჭრის ოპერაციის სასარგებლო ვარიანტია: s[:i] + s[i:], რომელიც ტოლია s-ის.

>>> word[:2] + word[2:]

’HelpA’

>>> word[:3] + word[3:]

’HelpA’

თუ ინდექსი აჭარბებს სტრიქონის სიგრძეს, იგი დამუშავდება თითქოს სტრიქონის სიგრძის ტოლი იყოს. თუ ზედა საზღვარი ნაკლებია ქვედაზე, დაბრუნდება ცარიელი სტრიქონი.

>>> word[1:100]

’elpA’

>>> word[10:]

’’

>>> word[2:1]

’’

ინდექსებს შეიძლება ჰქონდეთ უარყოფითი მნიშვნელობები ბოლოდან გადათვლისთვის:

>>> word[-1] # ბოლო სიმბოლო

’A’

>>> word[-2] # ბოლოსწინა სიმბოლო

’p’

>>> word[-2:] # ბოლო ორი სიმბოლო

```
'pA'
>>> word[:-2] # ბოლო ორი სიმბოლოს გარდა
'Hel'
```

-0 იგივეა, რაც 0, ანუ არ გადაითვლება ბოლოდან.

```
>>> word[-0]    # (რამდენადაც -0 იგივეა, რაც 0)
'H'
```

უარყოფითი ინდექსები ჭრებში, რომლებიც გადიან საზღვრებიდან, მუშავდება თითქოს ისინი ნულის ტოლი იყოს:

```
>>> word[-100:]
'HelpA'
>>> word[-10] # შეცდომაა
Traceback (innermost last):
File "<stdin>", line 1
IndexError: string index out of range
```

ჩაშენებული ფუნქცია len() აბრუნებს სტრიქონის სიგრძეს:

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

სტრიქონები აპოსტროფებში და ბრჭყალებში

```
S = 'spam"s'
S = "spam's"
```

ეკრანირებული მიმდევრობები - მომსახურე სიმბოლოები

ბრჭყალის წინ დგას სიმბოლო 'r' (ნებისმიერ რეგისტრში):.

```
S = r'C:\newt.txt'
```

სტრიქონი არ შეიძლება დამთავრდეს შებრუნებული სლეშის სიმბოლოთი. გადაწყვეტის გზები:

```
S = r'\n\n\\'[:-1]
S = r'\n\n' + '\\'
S = '\\n\\n'
```

სტრიქონები სამმაგ აპოსტროფებში ან ბრჭყალებში

სამმაგი ბრჭყალის გამოყენება:

```
>>> c = '''ეს ძალიან დიდი
...სტრიქონია, ტექსის
... მრავალსტრიქონიანი ბლოკი'''
>>> c
'ეს ძალიან დიდი\nსტრიქონია,
ტექსტის\nმრავალსტრიქონიანი ბლოკი'
>>> print(c)
ეს ძალიან დიდი
სტრიქონია, ტექსის
მრავალსტრიქონიანი ბლოკი
```

საბაზო ოპერაციები სტრიქონებზე

კონკატენაცია (შეკრება):

```
>>> S1 = 'spam'
>>> S2 = 'eggs'
>>> print(S1 + S2)
'spameggs'
```

სტრიქონის დუბლირება:

```
>>> print('spam' * 3)
spamspamspam
```

სტრიქონის სიგრძე - len ფუნქცია

```
>>> len('spam')
4
```

ინდექსზე წვდომა:

```
>>> S = 'spam'
>>> S[0]
's'
>>> S[2]
'a'
>>> S[-1]
‘m’
>>> S[-2]
'a'
```

სტრიქონიდან ფრაგმენტის ამოჭრა.

```
>>> s = 'spameggs'
>>> s[3:5]
'me'
>>> s[2:-2]
'ameg'
>>> s[:6]
'spameg'
>>> s[1:]
'pameggs'
>>> s[:]
'spameggs'
```

ბიჯის მითითება ჭრის დროს:

```
>>> s[::-1]
```

```
'sggemaps'
>>> s[3:5:-1]
''
>>> s[2::2]
'aeg'
```

ფუნქციები და მეთოდები სტრიქონებთან სამუშაოდ

ყველა ფუნქციას და მეთოდს შეუძლია მხოლოდ შექმნას ახალი სტრიქონი:

```
>>> s = 'spam'
>>> s[1] = 'b'
Traceback (most recent call last):
  File "", line 1, in
    s[1] = 'b'
TypeError: 'str' object does not support item assignment
>>> s = s[0] + 'b' + s[2:]
>>> s
'sbam'
```

სტრიქონებთან მუშაობის მაგალითები

1) პროგრამა ბეჭდავს ორ სტრიქონს:

```
name = "Tom"
surname = 'Smith'
print(name, surname)  # Tom Smith
```

2) ორი სტრიქონი გაერთიანდა და შემდეგ იბეჭდება გაერთიანებული სტრიქონი:

```
name = "Tom"
surname = 'Smith'
```

```
fullname = name + " " + surname
print(fullname)  # Tom Smith
```

3) მოცემულ პროგრამაში საჭიროა შეიკრიბოს სტრიქონი და რიცხვი. ამ შემთხვევაში საჭიროა რიცხვის გადაყვანა სტრიქონად str() ფუნქციის გამოყენებით:

```
name = "Tom"
age = 32
info = "Name: " + name + " Age: " + str(age)
print(info)  # Name: Tom Age: 32
```

4) გამოიტანს ტექსტს ორ სტრიქონად:

```
print ("ვსწავლობთ python ენაზე დაპროგრამებას\n-სტრიქონებთან მუშაობა საინტერესოა")
```

კონსოლზე გამოვა ორი სტრიქონი:

ვსწავლობთ python ენაზე დაპროგრამებას
სტრიქონებთან მუშაობა საინტერესოა

5) ტექსტში ბრჭყალების ასახვისთვის, მის წინ იწერება შებრუნებული სლეში:

```
print("Cafe \"Central Perk\"")
```

შედეგი: Cafe „Central Perk“

6) სტრიქონების შედარება:

```
str1 = "1a"
str2 = "aa"
str3 = "Aa"
print(str1 > str2)  # False, რადგან str1 - ში პირველი სიმბოლო ციფრია
```

print(str2 > str3) # True, რადგან str2 -ში პირველი სიმბოლო ქვედა რეგისტრშია.

1a<2a. # True

aa<ba # True

ba<ca # True

უმჯობესია ორივე სტრიქონი გადაყვანილ იქნას ერთნაირ რეგისტრში.

7) პროგრამაში სტრიქონები გადადის ერთნაირ რეგისტრებში და ხდება მათი შედარება:

```
tr1 = "Tom"
str2 = "tom"
print(str1 == str2)  # False - არ არის ტოლი
 print(str1.lower() == str2.lower())  # True
```

8) სტრიქონის განსაზღვრულ პოზიციაში სიმბოლოს შეცვლის მცდელობა იწვევს შეცდომას:

```
>>> word = 'strength'
>>> word[2] = 'y'
```

TypeError: 'str' object does not support item assignment

9) სტრიქონში სიმბოლოს შეცვლა შეიძლება შემდეგი სახით:

```
>>> word = word[:3] + '!' + word[4:]
            'str!ngth
```

10) სტრიქონში სიმბოლოს შეცვლა შეიძლება შემდეგი სახითაც:

```
>>> word = word.replace('!','e')
        'strength'
```

11)ათვლა ხდება ბოლოდან:

```
        >>> word[-1]
```

h

12) ერთი ტიპის ბრჭყალები ნებისმიერად ჩაშენდება სხვა ტიპის ბრჭყალებში:

```
>>> '123'
'123'
>>> "7'8''9"
"7'8''9"
```

13) გრძელი სტრიქონები გაყოფილია რამდენიმე ნაწილად შებრუნებული სლეშით:

```
>>> s = 'this is first word\
and this is second word'
```

14) სტრიქონების დიდი ნაკრები და მთელი ტექსტი შეიძლება ჩაისვას სამმაგ ბრჭყალებში:

```
>>> print """
One
Two
Three
"""
```

15)მაგალითში გამოყენებულია მმართველი მიმდევრობები:

```
>>> s ='a\nb\tc'
>>> print s
a
b   c
```

16) მაგალითში სტრიქონი შედგება სამი რიცხვის ბინარული მიმდევრობისგან - ორი რვაობითი და ერთი თექვსმეტობითი:

```
>>> s = '\001\002\x03'
```

```
>>> s
'\x01\x02\x03'
>>> len(s)
3
```

17) სტრიქონის ჭრები:

```
>>> word = 'strength'
>>> word[4]
n
>>> word[0:2]
st
>>> word[2:4]
re
```

18) ჭრაში გამოტოვებულია ჯერ პირველი სიმბოლო, შემდეგ - მეორე სიმბოლო:

```
>>> word[:3]
str
>>> word[5:]
gth
```

19) სტრიქონიდან სიმბოლოების მიმდევრობის არჩევა ციკლურობით:

```
>>> s = '1234567890'
>>> s[::2]
'13579'
>>> s[1:10:2]
'24680'
>>> s[::-1]
'0987654321'
```

20) სტრიქონების გამრავლება:

```
>>> '123' * 3
'123123123'
```

20) სტრიქონების ფორმატირება. პროცენტის მარცხნივ მიეთითება სტრიქონი, მარჯვნივ - მნიშვნელობა ან მნიშვნელობათა ჩამონათვალი:

```
>>> s = 'Hello %s' % 'world'
>>> s
'Hello world'
 >>> s = 'one  %s %s' % ('two','three')
>>> s
'one  two three'
```

21) რიცხვის სტრიქონში გადასაყვანად გამოყენებულია რიცხვითი სპეციფიკატორი - %d ან %f:

```
>>> s = 'one  %d %f' % (2 , 3.5)
>>> s
'one  2 3.500000'
```

22) მაგალითში შედეგის სტრიქონს ექნება სიგრძე 10 სიმბოლო, წილად ნაწილში გამოყოფილია 5 სიმბოლო:

```
>>> x = 4/3
>>> '%10.5f' % x
'  1.33333'
```

23) ხარვეზები მარცხნიდან შეიძლება დაფორმატდეს ნულებით:

```
>>> from math import pi
>>> '%015.10f' % pi
'0003.1415926536'
```

24) მაგალითში ფორმატირებისთვის გამოყენებულია Template - სტრიქონების შაბლონები:

```
>>> from string import Template
>>> s = Template('1 $two 3 4 $five')
>>> d={}
>>> d['two']=2
>>> d['five']=5
>>> s.substitute(d)
'1 2 3 4 5'
```

25) ქვესტრიქონის ძებნა სტრიქონში:

```
>>> s = 'The find method finds a substring'
>>> s.find('find')
4
>>> s.find('finds')
16
>>> s.find('findsa')
-1
```

26) join - აერთიანებს გამომყოფი ნიშნით სტრიქონების ნაკრებს:

```
>>> seq = ['one','two','three']
>>> sep = ','
>>> sep.join(seq)
'one,two,three'
```

27) split - დაყოფს სტრიქონს მიმდევრობად:

```
>>> s = '/usr/local/bin'
>>> s.split('/')
['', 'usr', 'local', 'bin']
```

28) replace - სტრიქონში ერთ ქვესტრიქონს ჩაანაცვლებს მეორეთი:

```
>>> s = 'replace method returns a string'
>>> s.replace('returns','return')
'replace method return a string'
```

29) strip - შლის ხარვეზს მარცხნიდან და მარჯვნიდან:

```
>>> ' this is whitespace string '.strip()
'this is whitespace string'
```

30) translate - აკეთებს მრავლობით შეცვლას. მაგალითში ყველა სიმბოლო ‘1’ შეიცვლება სიმბოლო ‘3’-ით, ხოლო სიმბოლო ‘2’, შეიცვლება სიმბოლოთი - ‘4’:

```
>>> from string import maketrans
>>> table = maketrans('12', '34')
>>> '1212 5656'.translate(table)
'3434 5656'
```

დავალება 1

კლავიატურიდან შეიტანეთ ორი სტრიქონი. გააერთიანეთ ისინი. გამოითვალეთ გაერთიანებული სტრიქონის სიგრძე. კონსოლზე გამოიტანეთ გაერთიანებული სტრიქონი და მისი სიგრძე.

დავალება 2

მოცემულ სტრიქონში: "i'm Richard", სახელი Richard ჩაანაცვლეთ თქვენი სახელით და გამოიტანეთ მიღებული სტრიქონი.

დავალება 3

მოცემულ სტრიქონში: „Technology is everywhere, yet few of us are in a position to take advantage of it“ იპოვეთ და გამოიტანეთ ინდექსი, საიდანაც იწყება სიტყვა: „position“. შემდეგ წაშალეთ მოცემულ სტრიქონში ყველა ხარვეზი და გამოიტანეთ მიღებული სტრიქონი.

ლაბორატორიული სამუშაო 4
თემა: სიები, კორტეჟი

დასამუშავებელი საკითხები:

- სიის განსაზღვრა;
- ოპერაციები სიებზე;
- მეთოდები სიებთან სამუშაოდ;
- კორტეჟის განსაზღვრა;
- კორტეჟის სიად გარდაქმნა;
- ოპერაციები კორტეჟის ელემენტებზე.

სიის მაგალითი:

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

სიის ელემენტებზე მიმართვა, სიის ჭრა, გაერთიანება, გამრავლება:

```
>>> a[0]
'spam'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boe!']
```

['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam',
'eggs', 100, 'Boe!']

სიის ცალკეული ელემენტის შეცვლა:

```
>>> a
['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', 'eggs', 123, 1234]
```

```
>>> # რამდენიმე ელემენტის შეცვლა:
... a[0:2] = [1, 12]
>>> a
[1, 12, 123, 1234]
```

```
>>> # წაშლა“
... a[0:2] = []
>>> a
[123, 1234]
```

```
>>> # ჩასმა:
... a[1:1] = ['bletch', 'xyzzy']
>>> a
[123, 'bletch', 'xyzzy', 1234]
```

```
>>> # თავისივე ასლის ჩასმა დასაწყისში:
>>> a[:0] = a
>>> a
[123, 'bletch', 'xyzzy', 1234, 123, 'bletch', 'xyzzy',
1234]
```

სიებისთვის ჩაშენებული ფუნქცია len() გამოყენება:

```
>>> len(a)
8
```

ჩაშენებული სიები, როდესაც ერთი სია შეიცავს სხვა სიას:

```
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> len(p)
3
>>> p[1]
[2, 3]
>>> p[1][0]
2
>>> p[1].append('xtra')
>>> p
[1, [2, 3, 'xtra'], 4]
>>> q
[2, 3, 'xtra']
```

სიის შექმნა:

```
a = []
for i in range(1,15):
...     a.append(i)
```

სიის შექმნა:

```
a = [i for i in range(1,15)]
```

სიის შექმნა შემთხვევითი რიცხვების გენერატორით:

```
from random import randint
l = [randint(10,80) for x in range(10)]
```

სიის შექმნა სიების გენერატორის მეშვეობით:

```
>>> c = [c * 3 for c in 'list']
print(c)
შედეგი:
['lll', 'iii', 'sss', 'ttt']
```

სიების გენერატორის უფრო რთული კონსტრუქცია:

```
>>> c = [c * 3 for c in 'list' if c != 'i']
>>> print (c)
['lll', 'sss', 'ttt']
```

sort მეთოდი:

```
>>> l = [8, 2, 6, 5, 7]
>>> l.sort()
>>> l
[2, 5, 6, 7, 8]
```

მაგალითები:

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print(a.count(333), a.count(66.25), a.count('x'))
2 1 0

>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]

>>> a.index(333)
1
>>> a.remove(333)
>>> a
```

```
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

მაგალითები: სიების შექმნა, შეცვლა, წაშლა და მუშაობა სიის ელემენტებთან

სია შეიძლება შეიქმნას ჩამოთვლილთაგან ერთ-ერთი ხერხით:

```
>>> a = []
>>> type(a)
<class 'list'>
>>> b = list()
>>> type(b)
<class 'list'>
```

სიის შექმნა წინასწარ მოცემული მონაცემთა ნაკრებით:

```
>>> a = [1, 2, 3]
>>> type(a)
<class 'list'>
```

უკვე არსებული სიის ასლის შექმნის ხერხები:

```
>>> a = [1, 3, 5, 7]
>>> b = a[:]
>>> print(a)
[1, 3, 5, 7]
```

```
>>> print(b)
[1, 3, 5, 7]
```

ან შეიძლება ასეც:

```
>>> a = [1, 3, 5, 7]
>>> b = list(a)
>>> print(a)
[1, 3, 5, 7]
>>> print(b)
[1, 3, 5, 7]
```

a სიი შეცვლით b სიაც შეიცვლება:

```
>>> a = [1, 3, 5, 7]
>>> b = a
>>> print(a)
[1, 3, 5, 7]
>>> print(b)
[1, 3, 5, 7]
>>> a[1] = 10
>>> print(a)
[1, 10, 5, 7]
>>> print(b)
[1, 10, 5, 7]
```

მოცემულ მაგალითში სიას ემატება ელემენტი:

```
>>> a = []
>>> a.append(3)
>>> a.append("hello")
>>> print(a)
[3, 'hello']
```

სიიდან ელემენტის ამოგდება remove(x) მეთოდის გამოყენებით:

```
>>> b = [2, 3, 5]
>>> print(b)
[2, 3, 5]
>>> b.remove(3)
>>> print(b)
[2, 5]
```

სიის ელემენტის ამოგდება მისი ინდექსის მიხედვით:
del სიის_სახელი[ინდექსი]

```
>>> c = [3, 5, 1, 9, 6]
>>> print(c)
[3, 5, 1, 9, 6]
>>> del c[2]
>>> print(c)
[3, 5, 9, 6]
```

ელემენტების წაშლა del ბრძანების გამოყენებით:

```
>>> a
[-1, 1, 66.6, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.6, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.6, 1234.5]
```

სიის ელემენტის მნიშვნელობის შეცვლა მისი ინდექსის მიხედვით:

```
>>> d = [2, 4, 9]
>>> print(d)
[2, 4, 9]
>>> d[1] = 17
>>> print(d)
[2, 17, 9]
```

სიის გასუფთავება მისი ხელახალი ინიციალიზებით:

```
>>> a = [3, 5, 7, 10, 3, 2, 6, 0]
>>> a=[]
```

სიის ბოლო ელემენტზე წვდომისთვის გამოყენებულია უარყოფითი ინდექსი:

```
>>> a = [3, 5, 7, 10, 3, 2, 6, 0]
>>>a[-1]
 0
```

სიიდან ქვესიის მიღება:

```
>>> a[1:4]
[5, 7, 10]
```

მაგალითში ხდება სიის ბოლოში ელემენტის დამატება. იგივე ოპერაცია შეიძლება შესრულდეს ასეც: a[len(a):] = [x].

```
>>> a = [1, 2]
>>> a.append(3)
>>> print(a)
[1, 2, 3]
```

აფართოვებს არსებულ სიას b სიიდან ყველა ელემენტის დამატების გზით. ექვივალენტურია ბრძანება: a[len(a):] = b.

```
>>> a = [1, 2]
```

```
>>> b = [3, 4]
>>> a.extend(b)
>>> print(a)
[1, 2, 3, 4]
```

x ელემენტს ჩასვამს i პოზიციაში. პირველი არგუმენტი არის ელემენტის ინდექსი, რომლის შემდეგაც ჩაისმება x ელემენტი:

```
>>> a = [1, 2]
>>> a.insert(0, 5)
>>> print(a)
[5, 1, 2]

>>> a.insert(len(a), 9)
>>> print(a)
[5, 1, 2, 9]
```

წაშლის x ელემენტის პირველ შესვლას სიიდან:

```
>>> a = [1, 2, 3]
>>> a.remove(1)
>>> print(a)
[2, 3]
```

pop() მეთოდი წაშლის ელემენტს მითითებული პოზიციიდან და გამოიტანს მას. თუ არგუმენტი არ არის მითითებული წაიშლება სიის ბოლო ელემენტი:

```
>>> a = [1, 2, 3, 4, 5]
>>> print(a.pop(2))
3
>>> print(a.pop())
```

```
5
>>> print(a)
[1, 2, 4]
```

clear() წაშლის სიიდან ყველა ელემენტს. ექვივალენტურია შემდეგის del a[:]:

```
>>> a = [1, 2, 3, 4, 5]
>>> print(a)
[1, 2, 3, 4, 5]
>>> a.clear()
>>> print(a)
[]
```

აბრუნებს ელემენტის ინდექსს:

```
>>> a = [1, 2, 3, 4, 5]
>>> a.index(4)
3
```

აბრუნებს მითითებული ელემენტის შესვლათა რაოდენობას სიაში:

```
>>> a=[1, 2, 2, 3, 3]
>>> print(a.count(2))
2
```

ალაგებს სიის ელემენტებს ზრდადობით:

```
>>> a = [1, 4, 2, 8, 1]
>>> a.sort()
>>> print(a)
[1, 1, 2, 4, 8]
```

სიას გადაალაგებს უკუმიმდევრობით:

```
>>> a = [1, 3, 5, 7]
>>> a.reverse()
>>> print(a)
[7, 5, 3, 1]
```

აბრუნებს სიის ასლს. ექვივალენტურია შემდეგის: a[:].

```
>>> a = [1, 7, 9]
>>> b = a.copy()
>>> print(a)
[1, 7, 9]
>>> print(b)
[1, 7, 9]
>>> b[0] = 8
>>> print(a)
[1, 7, 9]
>>> print(b)
[8, 7, 9]
```

ქმნის სიას მთელი რიცხვებისგან 0-დან n-მდე, სადაც n უნდა შეიტანოთ წინასწარ:

```
n = int(input())
a = []
for i in range(n):
    a.append(i)
print(a)
```

[0, 1, 2, 3, 4, 5, 6, 7]

წინა მოქმედება შეიძლება ჩაიწეროს უფრო მარტივადაც:

```
n = int(input())
a = [i for i in range(n)]
print(a)
```

ან კიდევ უფრო მარტივად, იმ შემთხვევაში თუ თქვენ მეტი აღარ გინდათ n-ის გამოყენება:

```
a = [i for i in range(int(input()))]   შეგვაქვს რიცხვი;
print(a)
```

ინდექსები და ჭრები

სიის ელემენტზე მიმართვა ინდექსის მიხედვით.

```
>>> a = [1, 3, 8, 7]
>>> a[0]
1
>>> a[3]
7
>>> a[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

მაგალითში სიის ელემენტებზე მიმართვისთვის გამოყენებულია უარყოფითი ინდექსები.

```
>>> a = [1, 3, 8, 7]
>>> a[-1]
7
```

```
>>> a[-4]
1
>>> a[-5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

სიებისთვის ჭრების გამოყენება:

```
>>> a = [1, 3, 8, 7]
>>> a[:]
[1, 3, 8, 7]
>>> a[1:]
[3, 8, 7]
>>> a[:3]
[1, 3, 8]
>>> a[::2]
[1, 8]
```

ყველა ეს პარამეტრი შეიძლება იყოს უარყოფითი:

```
>>> a = [1, 3, 8, 7]
>>> a[::-1]
[7, 8, 3, 1]
>>> a[:-2]
[1, 3]
>>> a[-2::-1]
[8, 3, 1]
>>> a[1:4:-1]
[]
```

ბოლო მაგალითში მიღებულია ცარიელი სია, რამდენადაც **START < STOP,** ხოლო STEP უარყოფითია. იგივე მოხდება, თუ მნიშვნელობის დიაპაზონი აღმოჩნდება ობიექტის საზღვრებს გარეთ:

```
>>> a = [1, 3, 8, 7]
>>> a[10:20]
[]
```

ჭრების საშუალებით ელემენტის დამატება და წაშლა:

```
>>> a = [1, 3, 8, 7]
>>> a[1:3] = [0, 0, 0]
>>> a
[1, 0, 0, 0, 7]
>>> del a[:-3]
>>> a
[0, 0, 7]
```

დავალება 1

მოცემულია სია: cars = ['BMW', 'Mercedes', 'Ford']
გამოიტანეთ სიის პირველი ელემენტი; გამოიტანეთ სიის ბოლო ელემენტი.

დავალება 2

დავალება 1-ში მოცემულ სიას დაამატეთ: 'Audi'

დავალება 3

დავალება 2-ში მიღებულ სიას დაამატეთ 'Mitsubishi' მე-3 ადგილას.

დავალება 4

დავალება 3-ში მიღებულ სიიდან გააკეთეთ ჭრები და ჩანაცვლება შემდეგი პირობებით:
ამოიღეთ სიის პირველი-ორი ელემენტი;
ამოიღეთ მესამედან დაწყებული სიის ყველა ელემენტი;
მეორე ელემენტი შეცვალეთ სხვა ელემენტით.

დავალება 5

მოცემული ორი სიიდან მიიღეთ გაერთიანებული სია:

```
first = ['a', 'b', 'c']
second = ['d', 'e', 'f']
```

დავალება

დავალება 6

მოცემულია ჩაშენებული სია:

```
Company = [
   'Apple',
   ['Samsung', 'Huawei'],
   'Windows'
]
```

გამოიტანეთ სრული სია;
გამოიტანეთ სიის პირველი ელემენტი;
გამოიტანეთ სიის ბოლო ელემენტი;
გამოიტანეთ სიაში ჩაშენებული სიის პირველი ელემენტი;
გამოიტანეთ სიაში ჩაშენებული სიის მეორე ელემენტი.

დავალება 7

მოცემულია სია:

```
Presidents = ['Kennedy', 'Nixon', 'Clinton', 'Bush', 'Obama']
```

აქედან მიიღეთ შედეგი:

1 - Kennedy
2 - Nixon
3 - Clinton
4 - Bush
5 - Obama

დავალება 8

მოცემულია სია:
d = [2, 4, 12, 5]
შეამოწმეთ შედის თუ არა ცვლადი 4 ამ სიაში და თუ შედის გამოიტანეთ შესაბამისი შეტყობინება.

დავალება 9

შექმენით სია, რომელიც შევსებულია 1-დან 10-მდე ნატურალური რიცხვებით

დავალება 10

მოცემული ნიმუშის მიხედვით დაწერეთ კოდი, რომელიც დააგენერირებს და გამოიტანს [-20 20] შუალედიდან 15 შემთხვევით რიცხვს.

დავალება 11

დაწერეთ პროგრამა, რომელიც შექმნის და გამოიტანს სიას:
[2,4,6,8,10,12,14,16,18,20]

დავალება 12

მოცემულია სია:
a = [2,-2,4,-4,7,5]
აქედან მიიღეთ b სია, რომელიც შედგება a სიის ელემენტების კვადრატებისგან.

დავალება 13

მოცემულია სია: [3,9,2,7,5,8,12,10] დაალაგეთ კლების მიხედვით.

კორტეჟი

კორტეჟი (tuple) მცირე ზომისაა სიასთან შედარებით:

```
>>> a = (1, 2, 3, 4, 5, 6)
>>> b = [1, 2, 3, 4, 5, 6]
>>> a.__sizeof__()
36
>>> b.__sizeof__()
44
```

ცარიელი კორტეჟის შექმნა:

```
>>> a = tuple() # tuple() ჩაშენებული ფუნქციის დახმარებით
>>> a
()
>>> a = () # კორტეჟის ლიტერალის დახმარებით
>>> a
()
>>>
```

კორტეჟის შექმნა ერთი ელემენტისგან:

```
>>> a = ('s')
>>> a
's'
```

მაგრამ აქ მივიღეთ სტრიქონი. ჩვენ გვინდოდა კორტეჟის მიღება, რისთვისაც კოდი შემდეგნაირად უნდა დაიწეროს:

```
>>> a = ('s', )
>>> a
('s',)
```

მიღებულ იქნა კორტეჟი.

კორტეჟი შეიძლება შეიქმნას ასეც:

```
>>> a = 's',
>>> a
('s',)
```

კორტეჟის შექმნა იტერირებული ობიექტიდან tuple() ფუნქციის გამოყენებით:

```
>>> a = tuple('hello, world!')
>>> a
('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```

მაგალითში ჩანს, რომ კორტეჟის ელემენტების შეცვლა არ შეიძლება:

```
>>> b = (1, 2, 3)
>>> print(b)
(1, 2, 3)
>>> b[1] = 15
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    b[1] = 15
TypeError: 'tuple' object does not support item assignment
```

ცარიელი კორტეჟის შექმნის მაგალითები:

```
>>> a = ()
>>> print(type(a))
<class 'tuple'>
```

```
>>> b = tuple()
>>> print(type(b))
<class 'tuple'>
```

კორტეჟის შექმნა tuple() ფუნქციით:

```
>>> a = tuple((1, 2, 3, 4))
>>> print(a)
(1, 2, 3, 4)
```

კორტეჟის ელემენტის ცვლილების მცდელობისას მიიღება შეცდომა:

```
>>> a = (1, 2, 3, 4, 5)
>>> print(a[0])
1
>>> print(a[1:3])
(2, 3)
>>> a[1] = 3
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    a[1] = 3
TypeError: 'tuple' object does not support item assignment
```

ცალკეული ელემენტის წაშლა კორტეჟიდან იძლევა შეცდომას:

```
>>> a = (1, 2, 3, 4, 5)
>>> del a[0]
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    del a[0]
TypeError: 'tuple' object doesn't support item deletion
```

კორტეჟის მთლიანად წაშლა: del a

სიის კორტეჟში გადაყვანა:

```
>>> lst = [1, 2, 3, 4, 5]
>>> print(type(lst))
<class 'list'>
>>> print(lst)
[1, 2, 3, 4, 5]
>>> tpl = tuple(lst)
>>> print(type(tpl))
<class 'tuple'>
>>> print(tpl)
(1, 2, 3, 4, 5)
```

კორტეჟის სიად გარდაქმნა:

```
>>> tpl = (2, 4, 6, 8, 10)
>>> print(type(tpl))
<class 'tuple'>
>>> print(tpl)
(2, 4, 6, 8, 10)
>>> lst = list(tpl)
>>> print(type(lst))
<class 'list'>
>>> print(lst) #[2, 4, 6, 8, 10]
```

ამორჩევა ბიჯით:

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
print(numbers[1:11:2])
(1, 3, 5, 7, 9)
```

დავალება 1

მოცემულია სია და კორტეჟი:

a = (10, 2.13, "square", 89, 'C')

>>> b = [1, 2, 3]

სია გარდაქმენით კორტეჟად და კორტეჟი სიად, გამოიტანეთ შედეგები და დარწმუნდით, რომ გარდაქმნა სწორად შესრულდა.

დავალება 2

მოცემულია კორტეჟი, რომელიც შეიცავს სიას. შეცვალეთ სიის მეორე ელემენტი 15-ით და გამოიტანეთ მიღებული კორტეჟი.

nested = (1, "do", ["param", 10, 20])

დავალება 3

მოცემულია ორი კორტეჟი. გამოიტანეთ გაერთიანებული კორტეჟი:

a=(1,)

b = ([1,2,3], 'word')

დავალება 4

მოცემულია ორი კორტეჟი. tup1-დან ამოიღეთ პირველი ელემენტი, ხოლო tup2-დან მეორე-მეხუთე ელემენტები და გამოიტანეთ:

tup1 = ('physics', 'chemistry', 1997, 2000)

tup2 = (1, 2, 3, 4, 5, 6, 7)

დავალება 5

მოცემული კორტეჟიდან მიიღეთ კორტეჟი მესამე ელემენტის გარეშე:

tup1 = (1, 2, 3)

დავალება 6

წაშალეთ მოცემული კორტეჟი სრულად:

tup = ('physics', 'chemistry', 1997, 2000)

დავალება 7

მოცემულ კორტეჟში დაითვალეთ 1-ების რაოდენობა:

t = (1, 2, 7, 1, 6)

დავალება 8

მოცემულ კორტეჟში იპოვეთ რიცხვი 7-ის ინდექსი:

t = (1, 2, 7, 1, 6)

დავალება 9

მოცემული კორტეჟიდან ამოიღეთ ყოველი მეორე ელემენტი:

p=(2,5,6,8,9,2,1,2)

ლაბორატორიული სამუშაო 5
თემა: ლექსიკონები (dictionary)

დასამუშავებელი საკითხები:

- ლექსიკონის განსაზღვრა;
- სიის გარდაქმნა ლექსიკონად;
- ოპერაციები ლექსიკონის ელემენტებზე;
- კომპლექსური ლექსიკონები;
- მეთოდები ლექსიკონებთან სამუშაოდ.

ლექსიკონის განსაზღვრის მაგალითები:

```
users = {1: "Tom", 2: "Bob", 3: "Bill"}
elements = {"Au": "ოქრო", "Fe": "რკინა", "H": "წყალბადი", "O": "ჟანგბადი"}
```

მაგალითში ლექსიკონის, როგორც გასაღებები, ასევე მნიშვნელობები სხვადასხვა ტიპის არის:

```
objects = {1: "Tom", "2": True, 3: 100.6}
```

ცარიელი ლექსიკონი ელემენტების გარეშე:

```
objects = {}
```

ან ასე:

```
objects = dict()
```

სიის გარდაქმნა ლექსიკონად

მაგალითში მოცემულია სიის გარდაქმნა ლექსიკონად **dict()** ჩაშენებული ფუნქციის დახმარებით. ცალკეული ჩაშენებული სია შედგება ორი ელემენტისგან - პირველი ელემენტი გახდება გასაღები, ხოლო მეორე - მნიშვნელობა:

```
users_list = [
```

```
    ["+111123455", "Tom"],
    ["+384767557", "Bob"],
    ["+958758767", "Alice"]
]
users_dict = dict(users_list)
print(users_dict)  # {"+111123455": "Tom", "+384767557": "Bob", "+958758767": "Alice"}
```

შედეგი:

```
{'+111123455': 'Tom', '+384767557': 'Bob', '+958758767': 'Alice'}
```

ორგანზომილებიანი კორტეჟის გარდაქმნა ლექსიკონად:

```
users_tuple = (
    ("+111123455", "Tom"),
    ("+384767557", "Bob"),
    ("+958758767", "Alice")
)
users_dict = dict(users_tuple)
print(users_dict)
```

ელემენტის ამოღება, შეცვლა

ლექსიკონის ელემენტებზე წვდომის ზოგადი ფორმა:

```
dictionary[key]
```

ლექსიკონიდან ელემენტის მნიშვნელობის ამოღება გასაღების მიხედვით:

```
users = {
    "+11111111": "Tom",
    "+33333333": "Bob",
```

```
    "+55555555": "Alice"}
# ვიღებთ ელემენტს გასაღებით "+11111111"
print(users["+11111111"])      # Tom
```

ელემენტის მნიშვნელობის დაყენება გასაღებით "+33333333":

```
users["+33333333"] = "Bob Smith"
print(users["+33333333"])      # Bob Smith
```

მითითებული გასაღებით ელემენტის მნიშვნელობა არ აღმოჩნდა, ამიტომ მოხდება მისი დამატება:

```
users["+4444444"] = "Sam"
```

მნიშვნელობის მიღების მცდელობა გასაღებით, რომელიც არ არის ლექსიკონში, გამოიტანს შეცდომას:

```
user = users["+7777777"]    # KeyError
```

ლექსიკონში გასაღების არსებობის წინასწარი შემოწმება **key in dictionary** მოსახულებით. თუ გასაღები არის ლექსიკონში, მაშინ მოცემული გამოსახულება აბრუნებს True მნიშვნელობას:

```
key = "+4444444"
if key in users:
    user = users[key]
    print(user)
else:
    print("ელემენტი არ არის ნაპოვნი")
```

ელემენტის მიღება **get(key) და get(key, default)** მეთოდებით:

```
key = "+8888888" # გასაღები არ არის ლექსიკონში
user = users.get(key) # აბრუნებს None
print(user)
```

შედეგი: None

```
user = users.get(key,"Unknown user")#აბრუნებს Unknown user
print(user)
```

შედეგი: Unknown user

ელემენტის წაშლა

გასაღების მიხედვით ელემენტის წაშლა **del** ოპერატორით:

```
users = {
    "+11111111": "Tom",
    "+33333333": "Bob",
    "+44444444": "Alice"
}
 del users["+44444444"]
print(users)
```

ელემენტის წაშლამდე მოცემული გასაღებით ელემენტის არსებობის შემოწმება:

```
key = "+44444444"
if key in users:
    user = users[key]
    del users[key]
    print(user, "წაშლილია")
else:
    print("ელემენტი არ არის ნაპოვნი")
```

ელემენტის წაშლა **pop(key)** და **pop(key, default)** მეთოდებით:

```
users = {
```

```
    "+11111111": "Tom",
    "+33333333": "Bob",
    "+55555555": "Alice"
}
key = "+55555555"
user = users.pop(key)#წაშლის და დააბრუნებს წაშლილ ელემენტს
print(user) # Alice
user = users.pop("+4444444", "Unknown user")#ელემენტი არ არის
print(user) # Unknown user
```

ყველა ელემენტის წაშლა **clear()** მეთოდის გამოყენებით:

```
users.clear()
```

ლექსიკონის კოპირება და გაერთიანება

copy() მეთოდი აკოპირებს ლექსიკონის შემადგენლობას და აბრუნებს ახალ ლექსიკონს:

```
users={"+1111111":"Tom","+3333333": "Bob","+5555555": "Alice"}
   users2 = users.copy()
```

update() მეთოდით users ლექსიკონში ჩაემატება users2 ლექსიკონი. ამ დროს users2 რჩება უცვლელად:

```
users = {"+1111111": "Tom","+3333333": "Bob","+5555555": "Alice"}
users2 = {"+2222222": "Sam","+6666666": "Kate"}
users.update(users2)
 print(users)  # {"+1111111": "Tom", "+3333333": "Bob", "+5555555": "Alice", "+2222222": "Sam", "+6666666": "Kate"}
print(users2)  # {"+2222222": "Sam", "+6666666": "Kate"}
```

მაგალითში ორივე საწყისი ლექსიკონი რჩება უცვლელად, ხოლო გაერთიანების შედეგი იქნება მესამე ლექსიკონი:

```
users3 = users.copy()
users3.update(users2)
```

ლექსიკონის გადარჩევა

ლექსიკონის გადარჩევისთვის for ციკლის გამოყენება:

```
users = {
    "+11111111": "Tom",
    "+33333333": "Bob",
    "+55555555": "Alice"
}
for key in users:
    print(key, " - ", users[key])
```

შედეგი:

```
+11111111  -  Tom

+33333333  -  Bob

+55555555  -  Alice
```

ელემენტების გადარჩევის სხვა ხერხი, **items()** მეთოდის გამოყენებით:

```
for key, value in users.items():
    print(key, " - ", value)
```

ცალკე გასაღების გადარჩევა keys() მეთოდის გამოყენებით:

```
for key in users.keys():
        print(key)
```

შედეგი:

+11111111
+33333333
+55555555

მხოლოდ მნიშვნელობის გადასარჩევად values() მეთოდის გამოყენება:

```
for value in users.values():
    print(value)
```

კომპლექსური ლექსიკონები

ჩაშენებული ლექსიკონი:

```
users = {
  "Tom": {
    "phone": "+971478745",
    "email": "tom12@gmail.com"
  },
  "Bob": {
    "phone": "+876390444",
    "email": "bob@gmail.com",
    "skype": "bob123"
  }
}
```

ჩაშენებული ლექსიკონის ელემენტზე მიმართვა:

```
old_email = users["Tom"]["email"]
print(old_email)
```

შედეგი:

tom12@gmail.com

მნიშვნელობის მიღების მცდელობა გასაღების მიხედვით, რომელიც არ არის ლექსიკონში:

```
tom_skype = users["Tom"]["skype"]   # KeyError
```

შეცდომის თავიდან ასაცილებლად ლექსიკონში გასაღების არსებობის შემოწმება:

```
key = "skype"
if key in users["Tom"]:
    print(users["Tom"]["skype"])
else:
print("skype is not found")
```

მაგალითები:

ლექსიკონის შექმნა ლიტერალის დახმარებით:

```
>>> d = {}
>>> d
{}
>>> d = {'dict': 1, 'dictionary': 2}
>>> d
{'dict': 1, 'dictionary': 2}
```

ლექსიკონის შექმნა dict ფუნქციის დახმარებით:

```
>>> d = dict(short='dict', long='dictionary')
>>> d
{'short': 'dict', 'long': 'dictionary'}
>>> d = dict([(1, 1), (2, 4)])
>>> d
{1: 1, 2: 4}
```

ლექსიკონის შექმნა fromkeys მეთოდის დახმარებით:

```
>>> d = dict.fromkeys(['a', 'b'])
>>> d
{'a': None, 'b': None}
>>> d = dict.fromkeys(['a', 'b'], 100)
>>> d
{'a': 100, 'b': 100}
```

ლექსიკონის შექმნა ლექსიკონის გენერატორების დახმარებით, რომლებიც ძალიან ჰგვანან სიების გენერატორებს:

```
>>> d = {a: a ** 2 for a in range(7)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

ლექსიკონში ჩანაწერის დამატება და გასაღების მნიშვნელობის ამოღება:

```
>>> d = {1: 2, 2: 4, 3: 9}
>>> d[1]
2
>>> d[4] = 4 ** 2
>>> d
{1: 2, 2: 4, 3: 9, 4: 16}
>>> d['1']
Traceback (most recent call last):
  File "", line 1, in
    d['1']
KeyError: '1'
```

როგორც მაგალითიდან ჩანს, ახალ გასაღებზე მინიჭება აფართოვებს ლექსიკონს, არსებულ გასაღებზე მინიჭება მას

გადააწერს მნიშვნელობას, ხოლო არარსებული გასაღების ამოღება იძლევა შეცდომას.

დავალება 1

მოცემულია ლექსიკონი:

a = {'cat': 'კატა', 'dog': 'ძაღლი', 'bird': 'ჩიტი', 'mouse': 'თაგვი'}

გამოიტანეთ რომელიმე მნიშვნელობა გასაღების მიხედვით

დავალება 2

დავალება 1-ში მოცემულ ლექსიკონს დაამატეთ ახალი ელემენტი და გამოიტანეთ მიღებული ლექსიკონი

დავალება 3

წინა დავალებით მიღებულ ლექსიკონში შეცვალეთ მნიშვნელობა გასაღების მიხედვით.

დავალება 4

წაშალეთ ერთ-ერთი ელემენტი და გამოიტანეთ მიღებული ლექსიკონი

დავალება 5

მოცემულია ლექსიკონი:

nums = {1: 'one', 2: 'two', 3: 'three'}

ა)გამოიტანეთ მხოლოდ გასაღებები;

ბ) გამოიტანეთ მხოლოდ მნიშვნელობები;

გ) გამოიტანეთ გასაღებები და მნიშვნელობები ტირეთი გამოყოფილი ერთმანეთისგან.

დავალება 6

მოცემულია ჩაშენებული ლექსიკონი:

```
dict= {
   "11111": {
      "name": "Salome",
      "group": "108121",
```

```
        },
    "22222": {
        "name": "Akaki",
        "group": "108122",
        "phone": "2323433"
    }
}
```

გამოიტანეთ პირველი ლექსიკონიდან გვარი, ხოლო მეორედან - ჯგუფი.

გამოიტანეთ პირველიდან ტელეფონი.

დავალება 7

მოცემულია ორი ლექსიკონი:

dict1 = {"000001": "ერთი","000002": "ორი","000003": "სამი", "000004": "ოთხი"}

dict2 = {"000005": "ხუთი","000006": "ექვსი", "000007":"შვიდი"}

მიიღეთ გაერთიანებული ლექსიკონი, ისე რომ ორივე მოცემული დარჩეს უცვლელი.

დავალება 8

გაასუფთავეთ წინა დავალებით მიღებული გაერთიანებული ლექსიკონი და შეამოწმეთ.

clear(), copy(), fromkeys(), get(), pop(), popitem(), setdefault(), update() მეთოდების გამოყენება ლექსიკონებში

დავალება 1

მოცემულია ლექსიკონი:

a = {'cat': 'კატა', 'dog': 'ძაღლი', 'bird': 'ჩიტი', 'mouse': 'თაგვი', 'elephant': 'სპილო'}

წაშალეთ ლექსიკონის ყველა ელემენტი და გამოიტანეთ მიღებული ლექსიკონი.

დავალება 2

მოცემულია ლექსიკონი:

nums = {1: 'one', 2: 'two', 3: 'three'}

დააკოპირეთ მოცემული ლექსიკონი. კოპირებულს დაუმატეთ მეოთხე ელემენტი, შინაარსიდან გამომდინარე. გამოიტანეთ თავდაპირველი ლექსიკონი და დამატებული ლექსიკონი.

დავალება 3

მოცემულია სია: `a = [1, 2, 3]`

შექმენით მოცემული სიიდან ლექსიკონი, რომლის ელემენტები იქნება გასაღებები. გამოიტანეთ მიღებული ლექსიკონი.

დავალება 4

დავალება 3-ში მოცემული სიიდან შექმენით ლექსიკონი, რომლის გასაღებები იქნება სიის ელემენტები, ხოლო მნიშვნელობები 10-ის ტოლი.

დავალება 5

მოცემულია ლექსიკონი:

nums = {1: 'one', 2: 'two', 3: 'three'}

გამოიყენეთ შესაბამისი მეთოდი და გამოიტანეთ რომელიმე ელემენტი გასაღების მიხედვით.

დავალება 6

მოცემულია ლექსიკონი:

nums = {1: 'one', 2: 'two', 3: 'three'}

ა) წაშალეთ ლექსიკონიდან რომელიმე ელემენტი მითითებული გასაღების მიხედვით და გამოიტანეთ მიღებული ლექსიკონი.

ბ) წაშალეთ თავისუფალი ელემენტი და გამოიტანეთ მიღებული ელემენტი.

დავალება 7

მოცემულია ლექსიკონი:

nums = {1: 'one', 2: 'two', 3: 'three'}

დაამატეთ ლექსიკონში ელემენტი აღნიშნული მეთოდით და გამოიტანეთ მიღებული ლექსიკონი.

დავალება 8

მოცემულია ლექსიკონი:

nums = {1: 'one', 2: 'two', 3: 'three'}

დაამატეთ ლექსიკონს სხვა ლექსიკონი, მაგ: {6: 'six', 7: 'seven'}

გამოიტანეთ მიღებული ლექსიკონი

ლაბორატორიული სამუშაო 6

თემა: სიმრავლე

დასამუშავებელი საკითხები:

- სიმრავლის განსაზღვრა;
- ოპერაციები სიმრავლეზე;
- სიმრავლეთა შორის დამოკიდებულება;
- frozen set ტიპი.

სიმრავლის განსაზღვრა

სიმრავლე (set) შედეგად გამოიტანს მხოლოდ უნიკალურ მნიშვნელობებს:

```
users = {"Tom","Bob","Alice", "Tom"}
print(users)    # {"Tom","Bob","Alice"}
```

set ფუნქციის გამოყენება ცარიელი სიმრავლის შესაქმნელად:

```
users = set()
```

სიმრავლის სიგრძის განსაზღვრისთვის len() ჩაშენებული ფუნქციის გამოყენება:

```
users = {"Tom","Bob","Alice"}
print(len(users)}   # 3
```

ელემენტების დამატება სიმრავლეში

თითო ელემენტის დასამატებლად **add()** მეთოდის გამოყენება:

```
users = set()
users.add("Sam")
print(users)
```

შედეგი:

{'Sam'}

ელემენტების წაშლა

მაგალითში **remove()** მეთოდი გამოყენებულია ერთი ელემენტის წასაშლელად. ელემენტის არარსებობის შემთხვევაში შეცდომის დაგენერირების თავიდან ასაცილებლად წაშლამდე in ოპერატორით შემოწმებულია ელემენტის არსებობა სიმრავლეში:

```
users = {"Tom", "Bob", "Alice"}
user = "Tom"
if user in users:
    users.remove(user)
print(users)
```

შედეგი:

{'Bob', 'Alice'}

ელემენტის წასაშლელად **discard()** მეთოდის გამოყენება, რომელიც არ აგენერირებს შეცდომას ელემენტის არ არსებობის დროს:

```
user = "Tim"
users.discard(user)
```

ყველა ელემენტის წასაშლელად **clear()** მეთოდის გამოყენება:

```
users.clear()
```

სიმრავლის გადარჩევა

ელემენტების გადასარჩევად for ციკლის გამოყენება:

```
users = {"Tom","Bob","Alice"}
 for user in users:
        print(user)
```

გადარჩევისას ცალკეული ელემენტი განთავსდება user ცვლადში.

ოპერაციები სიმრავლეებზე

copy() მეთოდით ერთი სიმრავლის შემადგენლობის სხვა ცვლადში დაკოპირება:

```
users = {"Tom","Bob","Alice"}
users3 = users.copy()
```

ორივე სიმრავლის გამოტანით დავრწმუნდებით, რომ users და users3 შეიცავს ერთიდაიგივე ელემენტებს:

```
print (users)
print (users3)
```

შედეგი:

```
{'Alice', 'Tom', 'Bob'}
{'Alice', 'Tom', 'Bob'}
```

union() მეთოდი აერთიანებს ორ სიმრავლეს და აბრუნებს ახალ სიმრავლეს:

```
users = {"Tom","Bob","Alice"}
users2 = {"Sam","Kate", "Bob"}
users3 = users.union(users2)
print(users3)
```

შედეგი:

```
{'Kate', 'Sam', 'Tom', 'Bob', 'Alice'}
```

intersection() მეთოდით სიმრავლეთა თანაკვეთის ოპერაციის შესრულება. შედეგად მიიღება ახალი სიმრავლე:

```
users = {"Tom","Bob","Alice"}
users2 = {"Sam","Kate", "Bob"}
 users3 = users.intersection(users2)
print(users3)
```

შედეგი:

```
{'Bob'}
```

intersection მეთოდის ნაცვლად შეიძლება გამოყენებულ იქნას ლოგიკური გამრავლების ოპერაცია:

```
users = {"Tom","Bob","Alice"}
users2 = {"Sam","Kate", "Bob"}
 print(users & users2)
```

მიიღება იგივე შედეგი:

```
{'Bob'}
```

სიმრავლის სხვაობის მიღება difference() მეთოდით ან გამოკლების ოპერაციით:

```
users = {"Tom","Bob","Alice"}
users2 = {"Sam","Kate", "Bob"}
 users3 = users.difference(users2)
print(users3)          # {"Tom", "Alice"}
print(users - users2)   # {"Tom", "Alice"}
```

სიმრავლეთა შორის დამოკიდებულება

issubset() მეთოდი დაადგენს არის თუ არა მიმდინარე სიმრავლე სხვა სიმრავლის ქვესიმრავლე:

```
users = {"Tom", "Bob", "Alice"}
```

```
superusers = {"Sam", "Tom", "Bob", "Alice", "Greg"}
 print(users.issubset(superusers))   # True
print(superusers.issubset(users))   # False
```

issuperset() მეთოდი, პირიქით, აბრუნებს True, თუ მიმდინარე სიმრავლე შეიცავს ქვესიმრავლეს:

```
users = {"Tom", "Bob", "Alice"}
superusers = {"Sam", "Tom", "Bob", "Alice", "Greg"}
print(users.issuperset(superusers))   # False
print(superusers.issuperset(users))   # True
```

frozen set ტიპი

frozen set - სიმრავლის ტიპი, რომელიც არ შეიძლება შეიცვალოს. მის შესაქმნელად გამოიყენება frozenset ფუნქცია:

```
users = frozenset({"Tom", "Bob", "Alice"})
```

მაგალითები სიმრავლეებზე:

სიმრავლის შექმნა:

```
>>> a = set([1,2,3,4,5])
>>> a
{1, 2, 3, 4, 5}
```

შეიძლება სიმრავლის ჩაწერა გამარტივებული ფორმით:

```
>>> b = {1,3,5}
>>> b
{1, 3, 5}
```

სიმრავლის შექმნა:

```
>>> a = set()
>>> a
set()


>>> a = set('hello')
>>> a
{'h', 'o', 'l', 'e'}
>>> a = {'a', 'b', 'c', 'd'}
>>> a
{'b', 'c', 'a', 'd'}
>>> a = {i ** 2 for i in range(10)} #სიმრავლეთა გენერატორი
>>> a
{0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
>>> a = {}   # ეს არ შეიძლება!
>>> type(a)
<class 'dict'>
```

სიმრავლე ვერ შეინახავს ორ ერთნაირ ელემენტს:

```
>>> set([1,1,1,2,2,2,3,3,3,4])
 set([1, 2, 3, 4])  # {1,2,3,4}


>>> set("hello")
set(['h', 'e', 'l', 'o'])   # {'h', 'e', 'o', 'l'}
```

მაგალითში მოცემულია სიმრავლის გადაცემა for ციკლში:

```
a = set([1,2,3,4,5])
```

```
for i in a:
    print (i)
```

შედეგი:

```
1
2
3
4
5
```

სიმრავლე გამოყენებულია გამეორებადი ელემენტების წასაშლელად:

```
>>> words = ['hello', 'daddy', 'hello', 'mum']
>>> set(words)
{'hello', 'daddy', 'mum'}
```

frozenset გამოყენების მაგალითი:

```
>>> a=set([1,2,3,4,5])
>>> b = frozenset([1,2,3,4,5])
>>> a == b
True
>>>a.add(6)
>>>a
{1, 2, 3, 4, 5, 6}
b.add(6)
Traceback (most recent call last):
  File "<pyshell#67>", line 1, in <module>
    b.add(6)
AttributeError: 'frozenset' object has no attribute 'add'
```

მოცემულ მაგალითში სია გარდაიქმნება სიმრავლედ, ხოლო შემდგომ სიმრავლე გარდაიქმნება სიად. რის შედეგადაც სიმრავლე გათავისუფლდება დუბლირებული ელემენტებისგან:

```
>>> lst = ["a","c","b","c","c","d","g","d","e","a","g","f","b"]
>>> lst
['a', 'c', 'b', 'c', 'c', 'd', 'g', 'd', 'e', 'a', 'g', 'f', 'b']
>>> newlst = list(set(lst))
>>> newlst
['a', 'c', 'b', 'e', 'd', 'g', 'f']
```

a და b სიმრავლეების გაერთიანება:

```
>>> a = [1,2,4,5,7]
>>> b = [2,3,4,5,6,7,8,9]
>>> a
[1, 2, 4, 5, 7]
>>> b
[2, 3, 4, 5, 6, 7, 8, 9]
>>> set(a)|set(b)
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

სიმრავლეთა გამოკლება:

```
>>> a
[1, 2, 4, 5, 7]
>>> b
[2, 3, 4, 5, 6, 7, 8, 9]
>>> set(a)-set(b)
set([1])

>>> set(b)-set(a)
set([8, 9, 3, 6])
```

სიმრავლეთა თანაკვეთა:

```
>>> set(a)&set(b)
set([2, 4, 5, 7])
```

სიმრავლეთა სიმეტრიული სხვაობა. აბრუნებს ელემენტებს, რომლებიც არ იკვეთებიან გადაცემულ სიმრავლეებში:

```
>>> set(a)^set(b)
set([1, 3, 6, 8, 9])
```

მაგალითში მოწმდება ერთი სიმრავლის ელემენტი შედის თუ არა მეორეში, ფაქტობრივად ერთი სიმრავლე არის თუ არა მეორეს ქვესიმრვლე:

```
>>>a=set([1,2,3,4,5,6,7,8,9]) #1-დან 9-მდე ნატურალური რიცხვების სიმრავლე
>>> b = set([3,4,5,6,7]) # 3-დან 7-მდე ნატურალური რიცხვების სიმრავლე
>>> c = set([3,5,7]) # 3, 5 და 7 რიცხვების სიმრავლე
>>> a
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b
set([3, 4, 5, 6, 7])
>>> c
set([3, 5, 7])
>>> e = set([3,5,7,9,0])
>>> d=set([9, 3, 5, 7])
>>> e
set([0, 9, 3, 5, 7])
>>> a
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> b
```

```
set([3, 4, 5, 6, 7])
>>> a>b
True
>>> b>a # "a" არ არის  "b" -ს ქვესიმრავლე
False
>>> b
set([3, 4, 5, 6, 7])
>>> c
set([3, 5, 7])
>>> c>b
False
>>> b>c # "c" არის"b"-ს ქვესიმრავლე
True
>>> a>c # "c" არის"a"-ს ქვესიმრავლე
True
>>> a>b>c # c" არის "b" და "a"-ს ქვესიმრავლე, "b" არის "a"-ს ქვესიმრავლე
True
>>> a
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> c
set([3, 5, 7])
>>> d
set([9, 3, 5, 7])
>>> a>d>c  #True
```

მასალა განმეორებისთვის
სიები, კორტეჟი, ლექსიკონი, სიმრავლე

სიების მაგალითები:

append() - ელემენტის დამატება

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
```

pop() - ელემენტის წაშლა და წაშლილის გამოტანა

```
>>> a = [1, 2, 3, 4]
>>> a.pop()
4
>>> a
[1, 2, 3]
>>> a.pop(0)
1
>>> a
[2, 3]
```

sort() - სორტირება

```
>>> a = [5, 8, 4, 3]
>>> sorted(a)
[3, 4, 5, 8]
>>> a
[5, 8, 4, 3]
>>> a.sort()
```

```
>>> a
[3, 4, 5, 8]
```

reverse() - უკუმიმდევრობით დალაგება

```
>>> a = [5, 8, 4, 3]
>>> sorted(a, reverse=True)
[8, 5, 4, 3]
>>> a
[5, 8, 4, 3]
>>> a.sort(reverse=True)
>>> a
[8, 5, 4, 3]
```

გამოიყენეთ ფუნქციები - sum, min, max

კორტეჟის მაგალითები:

მნიშვნელობების პარალელური მინიჭება:

```
>>> x, y = 3, 5
>>> x, y
(3, 5)
```

მნიშვნელობების გაცვლა:

```
>>> x, y = y, x
>>> x, y
(5, 3)
```

კორტეჟი შეიცავს სტრიქონს, სიას და კორტეჟს:

```
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

შედეგი:

```
('mouse', [8, 4, 6], (1, 2, 3))
```

my_tuple კორტეჟიდან სიის პირველი ელემენტის გამოტანა:

```
print(my_tuple[1][0])
```

8

Count() მეთოდი

```
my_tuple = ('a','p','p','l','e')
print(my_tuple.count('p'))
```

შედეგი:

2

Index() მეთოდი

```
my_tuple = ('a','p','p','l','e')
print(my_tuple.index('l'))
```

შედეგი:

3

in ოპერაცია

```
my_tuple = ('a','p','p','l','e')
print('a' in my_tuple)
```

შედეგი:

True

```
print('b' in my_tuple)
```

შედეგი:

False

იტერაციული გამოტანა

```
for name in ('John','Kate'):
    print("Hello",name)
```

შედეგი:

Hello John

Hello Kate

ლექსიკონის მაგალითები:

ლექსიკონიდან ყველა ელემენტის გამოტანა ორი ხერხით:

```
cars = {
   'BMW': 'x5',
   'Mercedes-Benz': 'E220',
   'Audi': 'Q7'
}
for key in cars:
   print ("%s -> %s" % (key, cars[key]))
print( '-' * 15)
print ('2 მაგალითი')
print ('-' * 15)
for key in cars.keys():
   print ("%s -> %s" % (key, cars[key]))
```

შედეგი:

```
BMW -> x5
Mercedes-Benz -> E220
Audi -> Q7
---------------
2 მაგალითი
---------------
BMW -> x5
Mercedes-Benz -> E220
Audi -> Q7
```

del ბრძანება

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']
```

dict
{'Age': 7, 'Class': 'First'}

dict.clear()
{}

del dict - წაშლის მთლიანად ლექსიკონს.

copy() მეთოდი

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
studentX=Boys.copy()
studentY=Girls.copy()
print(studentX)
print(studentY)
```

update() მეთოდი

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Dict.update({"Sarah":9})
print(Dict)
```

შედეგი:

{'Tim': 18, 'Charlie': 12, 'Tiffany': 22, 'Robert': 25, 'Sarah': 9}

items() მეთოდი

ლექსიკონიდან მიიღება კორტეჟებისგან შედგენილი სია:

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print("Students Name: %s" % list(Dict.items()))
```

შედეგი:

Students Name: [('Tim', 18), ('Charlie', 12), ('Tiffany', 22), ('Robert', 25)]

შეამოწმებს Dict ლექსიკონის ცალკეული ელემენტი შედის თუ არა Boys ლექსიკონში და გამოიტანს შესაბამისად: True ან False

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
for key in list(Dict.keys()):
   if key in list(Boys.keys()):
      print(True)
   else:
      print(False)
```

შედეგი:

True

True

False

True

sort() - ლექსიკონის სორტირება

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
Students = list(Dict.keys())
Students.sort()
for S in Students:
    print(":".join((S,str(Dict[S]))))
```

შედეგი:

Charlie:12

Robert:25

Tiffany:22
Tim:18

სიმრავლე

სიმრავლის შექმნა:

```
items = {"arrow", "spear", "arrow", "arrow", "rock"}
items
{'arrow', 'spear', 'rock'}
```

ამოწმებს წევრი თუ არის სიმრავლეში:

```
if "rock" in items:
   print("Rock exists")
```

შედეგი:
Rock exists

```
if "clock" not in items:
   print("Cloak not found")
```

შედეგი:
Cloak not found

სიიდან სიმრავლის შექმნა

```
numbers = set([10, 20, 20, 30, 40, 50])
print(numbers)
```

შედეგი:
{40, 10, 50, 20, 30}

adds, discards, removes მეთოდები

ცარიელ სიმრავლეში ელემენტების დამატება:

```
items = set()
items.add("cat")
```

```
items.add("dog")
items.add("gerbil")
print(items)
```
შედეგი:
```
{'dog', 'cat', 'gerbil'}
```

discards და removes მეთოდებით სიმრავლედან წაშალეთ თითო ელემენტი.

issubset(),issuperset(),intersection() მეთოდები

```
numbers1 = {1, 3, 5, 7}
numbers2 = {1, 3}

if numbers2.issubset(numbers1):
   print("Is a subset")
```
შედეგი:
```
Is a subset
```

```
if numbers1.issuperset(numbers2):
   print("Is a superset")
```
შედეგი:
```
Is a superset
```

```
print(numbers1.intersection(numbers2))
```
შედეგი:
```
{1, 3}
```

union() მეთოდი

```
set1 = {1, 2, 3}
set2 = {6, 5, 4, 3}
```

```
set3 = set1.union(set2)
print(set3)
```

შედეგი:

```
{1, 2, 3, 4, 5, 6}
```

frozenset() სიმრავლე

person სიმრავლედან frozenset სიმრავლის შექმნა:

```
person = {"name": "John", "age": 23, "sex": "male"}
fSet = frozenset(person)
```

სიიდან frozenset სიმრავლის შექმნა:

```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
```

frozenset სიმრავლეში ცვლილების შეტანა იძლევა შეცდომას:

```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
x[1] = "strawberry"
```

ლაბორატორიული სამუშაო 7
თემა: ფუნქციები

დასამუშავებელი საკითხები:

- ფუნქციის განსაზღვრის სინტაქსი;
- ფუნქციის პარამეტრები არის გასაღებური სიტყვები;
- პარამეტრები დუმილით;
- ფუნქცია სხვადასხვა რაოდენობის არგუმენტით;
- ფუნქცია სხვადასხვა რაოდენობის გასაღებური არგუმენტით;
- ფუნქცია დასაბრუნებელი მნიშვნელობით;
- ცვლადის ხედვის არე;
- lambda(),filter(),reduce(),map() ფუნქციები.

ფუნქციის განსაზღვრის სინტაქსი

ფუნქცია უპარამეტრო:

```
def a_function():
    print("You just created a function!")
```

ფუნქცია პარამეტრებით:

```
def my_function(argument):
    print (argument)
my_function("abracadabra")
```

ფუნქცია არ შეიცავს კოდს:

```
def empty_function():
    pass
```

ფუნქცია იღებს ორ არგუმენტს

```
def bigger(a,b):
```

```
    if a > b:
        print (a)
    else:
        print (b)
```

ფუნქციის კორექტული გამოძახებაა:

```
bigger(5,6)
```

ფუნქციის არაკორექტული გამოძახებაა:

```
bigger()
bigger(3)
bigger(12,7,3)
```

პარამეტრები არის გასაღებური სიტყვები

```
def person(name, age):
    print (name, "is", age, "years old")
```

ასეთ შემთხვევაში ფუნქციის გამოძახებისას პარამეტრის რიგითობას მნიშვნელობა არა აქვს:

```
person(age=23, name="John")
```

პარამეტრები დუმილით

განსაზღვრულია space ფუნქცია:

```
def space(planet_name, center="Star"):
    print (planet_name, "is orbiting a", center)
```

space ფუნქციის გამოძახება შეიძლება შემდეგი სახით:

```
space("Mars")
```

მიღებული შედეგი:

```
Mars is orbiting a Star
```

space ფუნქციის გამოძახება შეიძლება შემდეგი სახითაც:

```
space("Mars", "Black Hole")
```

მიღებული შედეგი:
Mars is orbiting a Black Hole

ფუნქცია სხვადასხვა რაოდენობის არგუმენტით - *args

```
def unknown(*args):
    for argument in args:
        print (argument)
```

ა) unknown ფუნქცია მიიღებს ორ არგუმენტს:
unknown("hello","world")
შედეგად დაბეჭდავს ორივე სიტყვას.

ბ) unknown ფუნქცია მიიღებს ხუთ არგუმენტს:
unknown(1,2,3,4,5)
შედეგად დაბეჭდავს ყველა რიცხვს, ცალკეულს ახალი სტრიქონიდან.

გ) ფუნქციას არ გადაეცემა არგუმენტი:
unknown()
ფუნქცია არაფერს არ დაბეჭდავს.

ფუნქცია სხვადასხვა რაოდენობის გასაღებური არგუმენტით - **kwargs

```
def many(*args, **kwargs):
    print( args )
    print( kwargs )

  many(1, 2, 3, name="Mike", job="programmer")
```

შედეგი:
```
# (1, 2, 3)
# {'job': 'programmer', 'name': 'Mike'}
```

return გასაღებური სიტყვის გამოყენება ფუნქციაში

```
def bigger(a,b):
  if  a > b:
      return a
  return b
```

მაგალითში ჩანს, რომ მსგავს შემთხვევაში სიტყვა else შეიძლება არც გამოვიყენოთ.

ცვლადს მივანიჭოთ bigger ფუნქციის შედეგი:

```
num = bigger(23,42)
```

ცვლადის ხედვის არე

გლობალური ცვლადი, ლოკალური ცვლადი

```
age = 44 # გლობალური ცვლადი

def info():
   print (age) # დაბეჭდავს age გლობალურ ცვლადს

def local_info():
   age = 22 # ქმნის ლოკალურ age  ცვლადს
   print (age)

info() # დაბეჭდავს 44
local_info() # დაბეჭდავს 22
```

ფუნქციის შიგნით გლობალური ფუნქციის შესაცვლელად global გასაღებური სიტყვის გამოყენება

```
გლობალური ცვლადი
age = 13
 # ფუნქცია ცვლის გლობალურ ცვლადს
def get_older():
```

```
    global age
    age += 1
 print (age) # ბეჭდავს 13
get_older() # ზრდის age  ცვლადს 1-ით
print (age) # ბეჭდავს 14
```

ფუნქცია ითვლის და აბრუნებს num ცვლადის ფაქტორიალს:

```
def fact(num):
    if num == 0:
        return 1
    else:
        return num * fact(num - 1)
```

lambda() ფუნქცია

lambda() ფუნქცია გამოიყენება მაშინ, როდესაც საჭიროა განისაზღვროს ფუნქცია def func_name(): გარეშე.

```
func = lambda x, y: x + y
func(1, 2)
3
func('a', 'b')
  'ab'
```

შეიძლება ასეც ჩაიწეროს:

```
>>> (lambda x, y: x + y)(1, 2)
  3
>>> (lambda x, y: x + y)('a', 'b')
  'ab'
```

filter() ფუნქცია

გაფილტვრა ხდება filter ფუნქციაში მოცემული პირობის მიხედვით:

```
number_list = range(-5, 5)
less_than_zero = list(filter(lambda x: x < 0, number_list))
print(less_than_zero)
```

შედეგი:

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

reduce() ფუნქცია

ფუნქციას შეუძლია შეასრულოს მცოცავი გამოთვლები ერთმანეთის მომდევნო წყვილებისთვის სიებში.

მთელი რიცხვებისგან შემდგარი სიის ნამრავლის გამოთვლა:

```
from functools import reduce
product = reduce((lambda x, y: x * y), [1, 2, 3, 4])
print (product)
```

reduce აერთიანებს რამდენიმე სიას:

```
reduce(list.__add__, [[1, 2, 3], [4, 5], [6, 7, 8]], [])
```

შედეგი: [1, 2, 3, 4, 5, 6, 7, 8]

map() ფუნქცია

map() ფუნქცია იღებს ორ არგუმენტს, პირველი არის ფუნქცია, რომელიც გამოყენებული იქნება სიის ცალკეული ელემენტისთვის, მეორე - სიაა, რომელიც უნდა დამუშავდეს:

```
>>> m = [1, 2, 3, 4, 5, 6, 7]
>>> list(map(lambda x: x**3, m))
```

მაგალითში ფუნქცია pow იღებს ორ არგუმენტს ცალკეული გამოძახებისას:

```
>>> pow(3,5)
243
>>> pow(2,10)
1024
>>> pow(3,11)
177147
>>> pow(4,12)
16777216
```

კოდის გასამარტივებლად **map()** ფუნქციის გამოყენება:

```
>>> list(map(pow, [3,2, 3, 4], [5,10, 11, 12]))
[243,1024, 177147, 16777216]
```

დავალება 1

შექმენით უპარამეტრო ფუნქცია, რომელიც ორ რიცხვზე შეასრულებს ნებისმიერ ოპერაციას. გამოიძახეთ ფუნქცია და გამოიტანეთ შედეგი.

დავალება 2

შექმენით პარამეტრებიანი ფუნქცია, რომელიც იპოვის ორ რიცხვს შორის უდიდესს. გამოიძახეთ ფუნქცია და გამოიტანეთ შედეგი.

დავალება 3

შექმენით ფუნქცია სამი პარამეტრით, ისე რომ პარამეტრები იყოს გასაღებური სიტყვები. გამოიძახეთ ფუნქცია პარამეტრების ნებისმიერი რიგითობით.

დავალება 4

შექმენით ფუნქცია ორი პარამეტრით, სადაც ერთ-ერთი პარამეტრი ცხადად იქნება წარმოდგენილი. გამოიძახეთ ფუნქცია: ერთი პარამეტრით და ორი პარამეტრით.

დავალება 5

შექმენით ფუნქცია სხვადასხვა რაოდენობის პარამეტრით. გამოიძახეთ იგი: ხუთი სხვადასხვა რიცხვით, სამი სხვადასხვა სიტყვით.

დავალება 6

შექმენით ფუნქცია სხვადასხვა რაოდენობის გასაღებური პარამეტრით. გამოიძახეთ იგი ოთხი პარამეტრით: გვარი, სახელი, დაბადების წელი, ასაკი.

დავალება 7

გამოიძახეთ მოცემული ფუნქციები. იპოვეთ შეცდომის მიზეზი. გაასწორეთ და შეასრულეთ პროგრამა.

```
def function_a():
   a = 1
   b = 2
   return a+b

def function_b():
   c = 3
   return a+c
```

ლაბორატორიული სამუშაო 8

თემა: რიცხვითი მასივები

(NumPy ბიბლიოთეკის გამოყენება)

დასამუშავებელი საკითხები:

- მატრიცის განსაზღვრა;
- მატრიცის ელემენტების დამუშავება;
- მატრიცის შევსება ერთნაირი ელემენტებით;
- მასივის მიღება arange() ფუნქციით;
- მასივებზე ართმეტიკული ოპერაციები;
- ფუნქციები მასივებთან სამუშაოდ;
- მასივის გარდაქმნა ბრტყელი სახით;
- მასივის ფორმის შეცვლა;
- მასივების გაერთიანება;
- ორგანზომილებიანი მასივების ჩადგმული გენერატორები.

მატრიცის განსაზღვრა

მოცემულია 4 x 3 მატრიცის მაგალითი:

```
matrix = [[-1, 0, 1],
          [-1, 0, 1],
          [0, 1, -1],
          [1, 1, -1]]
```

მოცემული ოპერატორი შეიძლება დაიწეროს ერთ სტრიქონად:

```
matrix = [[-1, 0, 1], [-1, 0, 1], [0, 1, -1], [1, 1, -1]]
```

მატრიცის გამოტანა ერთ სტრიქონად შეიძლება ერთი ოპერატორით:

print(matrix)

შედეგი:

```
[[-1, 0, 1], [-1, 0, 1], [0, 1, -1], [1, 1, -1]]
```

მატრიცის გამოტანა ცხრილის სახით, სადაც i – სტრიქონის ნომერია, j – სვეტის ნომერია, ხოლო len(matrix) – სტრიქონების რაოდენობაა მატრიცაში:

```
def printMatrix ( matrix ):
  for i in range ( len(matrix) ):
    for j in range ( len(matrix[i]) ):
      print ( "{:4d}".format(matrix[i][j]), end = "" )
    print ()
```

ფუნქციის გამოძახებით დაიბეჭდება მატრიცა:

```
printMatrix ( matrix )
```

```
  -1   0   1
  -1   0   1
   0   1  -1
   1   1  -1
```

მატრიცის გამოტანის მეორე ხერხი. გარე ციკლი გაივლის მატრიცის სტრიქონებზე (row), ხოლო შიდა ციკლი გაივლის ცალკეული სტრიქონის ელემენტებზე (x):

```
def printMatrix ( matrix ):
  for row in matrix:
    for x in row:
      print ( "{:4d}".format(x), end = "" )
    print ()
```

მატრიცის ინიციალიზება შემთხვევითი რიცხვებით:

```
matrix = [[0, 0, 0],
          [0, 0, 0],
```

```
            [0, 0, 0],
            [0, 0, 0]]
import random
for i in range(4):
   for j in range(3):
      matrix[i][j] = random.randint ( 30, 60 )
      print ( “{:4d}”.format(matrix[i][j]), end = “” )
   print()
```

შედეგის ერთ-ერთი ვარიანტი:

```
 59   40   60
 46   55   38
 57   36   38
 60   45   60
```

მატრიცის ელემენტების დამუშავება

matrix[2][3]-მატრიცის მესამე სტრიქონის მეოთხე ელემენტია;

მატრიცის ელემენტების ნამრავლის გამოთვლა:

```
matrix = [[-1, 2, 1],
          [-1, 3, 1],
          [2, 1, -1],
          [1, 1, -1]]

p = 1
for i in range(4):
   for j in range(3):
      p *= matrix[i][j]
print (p)
```

მატრიცის ელემენტების ჯამის გამოთვლა, სტრიქონების ჯამის გამოსათვლელად გამოყენებულია სტანდარტული sum ფუნქცია:

```
s = 0
for row in matrix:
  s += sum(row)
print (s)
```

მატრიცის შევსება ერთნაირი ელემენტებით

მასივის ნულებით შევსება:

```
import numpy as np
a = np.zeros((3, 5))
print(a)
```

შედეგი:

```
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

მიიღება ერთიანებისგან შედგენილი ორი მატრიცა:

```
import numpy as np
a = np.ones((2, 2, 2))
print(a)
```

შედეგი:

```
[[[ 1.  1.]
  [ 1.  1.]]

 [[ 1.  1.]
  [ 1.  1.]]]
```

მატრიცის შექმნა, რომლის დიაგონალზე არის 1-ები:

```
import numpy as np
a=np.eye(5)
print(a)
```

შედეგი:

```
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
```

მასივის შექმნა შევსების გარეშე. საწყისი შემადგენლობა შემთხვევითია და დამოკიდებულია მეხსიერების მდგომარეობაზე მასივის შექმნის მომენტში:

```
import numpy as np
a=np.empty((3, 3));
print(a)
```

შედეგი:

```
[[  0.00000000e+000   6.92943707e-310   6.92943078e-310]
 [  6.92943079e-310   6.92943706e-310   6.92943706e-310]
 [  6.92943706e-310   6.92943706e-310   4.68846127e-310]]
```

მასივის მიღება arange() ფუნქციით

ა) მთელ რიცხვთა მასივის მიღება:

```
import numpy as np
a=np.arange(20, 50, 5)
print(a)
```

შედეგი:

```
[20 25 30 35 40 45]
```

ბ) ნამდვილ რიცხვთა მასივის მიღება:

```
import numpy as np
a=np.arange(0, 2 ,0.3)
print(a)
```

შედეგი:

```
[ 0.   0.3  0.6  0.9  1.2  1.5  1.8]
```

მასივებზე ართმეტიკული ოპერაციები

```
import numpy as np
a=np.array([20, 30, 40, 50])
b=np.arange(4)
print(a+b)
print(a-b)
print(a*b)
```

შედეგი:

```
[20 31 42 53]
[20 29 38 47]
[  0  30  80 150]
```

sum, max, min ფუნქციები

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6]])
print(np.sum(a))
print(np.max(a))
print(np.min(a))
```

შედეგი:

```
21
6
1
```

მასივის დაბეჭდვა უკუმიმდევრობით

```
import numpy as np
a=np.array([ 0, 1, 8, 8, 8, 8, 8, 343, 512, 729])
print(a[::-1])
```

შედეგი:

```
[729 512 343   8   8   8   8   8   1   0]
```

მასივის გარდაქმნა ბრტყელი სახით:

```
import numpy as np
a=np.array([[ 0, 1, 2],
[ 10, 12, 13],
[100, 101, 102],
[110, 112, 113]])
print(a.ravel())
```

შედეგი:

```
[  0   1   2  10  12  13 100 101 102 110 112 113]
```

მასივის ფორმის შეცვლა

```
import numpy as np
a=np.array([ 0, 1, 2, 10, 12, 13, 100, 101, 102, 110, 112, 113])
a.shape=(6,2)
print(a)
```

შედეგი:

```
[[  0   1]
 [  2  10]
 [ 12  13]
 [100 101]
 [102 110]
 [112 113]]
```

მასივების გაერთიანება vstack და hstack()

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
print(np.vstack((a, b)))
```

შედეგი:

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
print(np.hstack((a, b)))
```

შედეგი:

```
[[1 2 5 6]
 [3 4 7 8]]
```

ორგანზომილებიანი მასივის დამუშავების მაგალითი

მოცემულია ორგანზომილებიანი მასივი nxn. მთავარ დიაგონალზე მდებარე წევრებს (იმ a[i][j] ელემენტებს, რომელთათვისაც i==j) მიენიჭოს მნიშვნელობა 1, ელემენტებს, რომლებიც მდებარეობენ მთავარი დიაგონალის ზევით მიენიჭოს - მნიშვნელობა 0, ხოლო ელემენტებს

მთავარი დიაგონალის ქვევით მიენიჭოს - მნიშვნელობა 2. მაგალითად n==4 სათვის უნდა მივიღოთ მასივი:

1 0 0 0

2 1 0 0

2 2 1 0

2 2 2 1

განვიხილოთ ამოცანის ამოხსნის რამდენიმე ხერხი. ელემენტები, რომლებიც მდებარეობენ მთავარი დიაგონალის მაღლა ეს არის a[i][j] ელემენტები, რომელთათვისაც i<j, ხოლო მთავარი დიაგონალის ქვედა ელემენტებისთვის i>j. ამგვარად, შეიძლება შევადაროთ i და j მნიშვნელობები და მათ მიხედვით განვსაზღვროთ a[i][j] მნიშვნელობა. მივიღოთ შემდეგი ალგორითმი:

```
n = 4
a = [[0] * n for i in range(n)]
for i in range(n):
    for j in range(n):
        if i < j:
            a[i][j] = 0
        elif i > j:
            a[i][j] = 2
        else:
            a[i][j] = 1
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

მოცემულ ალგორითმში ცალკეული ელემენტის დასამუშავებლად უნდა შესრულდეს ერთი ან ორი if ინსტრუქცია. თუ გავართულებთ ალგორითმს შეიძლება თავი ავარიდოთ პირობის ინსტრუქციას.

თავდაპირველად შევავსოთ მთავარი დიაგონალი, რისთვისაც დაგვჭირდება ერთი ციკლი:

```
for i in range(n):
   a[i][i] = 1
```

შემდგომ შევავსოთ 0-ებით ყველა ელემენტი მთავარი დიაგონლის ზევით, რისთვისაც დაგვჭირდება ცალკეული სტრიქონისთვის სვეტის ინდექსი იცვლებოდეს: j=i+1, ..., n-1. ამისთვის დაგვჭირდება ჩადგმული ციკლები:

```
for i in range(n):
   for j in range(i + 1, n):
      a[i][j] = 0
```

ანალოგიურად ვანიჭებთ ელემენტებს მნიშვნელობა 2-ს, სადაც j=0, ..., i-1:

```
for i in range(n):
   for j in range(0, i):
      a[i][j] = 2
```

შეიძლება გავაერთიანოთ გარე ციკლები და მივიღებთ კიდევ ერთ კომპაქტურ ამონახსნს:

```
n = 4
a = [[0] * n for i in range(n)]
for i in range(n):
   for j in range(0, i):
```

```
        a[i][j] = 2
    a[i][i] = 1
    for j in range(i + 1, n):
        a[i][j] = 0
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

შემდეგი ალგორითმი იყენებს სიების განმეორების ოპერაციას სიის მომდევნო სტრიქონის ასაგებად. სიის i -ური სტრიქონი შედგება i რაოდენობის რიცხვი 2 -სგან, შემდეგ მოდის ერთი - რიცხვი 1, შემდეგ კი - n-i-1 რიცხვი 0:

```
n = 4
a = [0] * n
for i in range(n):
    a[i] = [2] * i + [1] + [0] * (n - i - 1)
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

ციკლი შეიძლება შევცვალოთ გენერატორით:

```
n = 4
a = [0] * n
a = [[2] * i + [1] + [0] * (n - i - 1) for i in range(n)]
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

ორგანზომილებიანი მასივების ჩადგმული გენერატორები

გენერატორი ქმნის n ელემენტიან სიას, რომლის ცალკეული ელემენტია m რაოდენობის 0-სგან შედგენილი სია.

```
m=3
```

n=5

[[0] * m for i in range(n)]

შედეგი:

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

ამ დროს შიდა სია აგრეთვე შეიძლება შეიქმნას მაგალითად, ასეთი გენერატორისგან:

[0 for j in range(m)].

ერთი გენერატორის მეორეში ჩადგმით მიიღება ჩადგმული გენერატორები:

[[0 for j in range(m)] for i in range(n)]

რიცხვი 0-ს შევცვალოთ რაიმე გამოსახულებით, რომელიც დამოკიდებულია სტრიქონისა და სვეტის ნომერზე. მივიღებთ რაიმე ფორმულით შევსებულ სიას.
მაგალითად, გვინდა მივიღოთ შემდეგი მასივი:

0 0 0 0 0 0

0 1 2 3 4 5

0 2 4 6 8 10

0 3 6 9 12 15

0 4 8 12 16 20

ამ მასივში n = 5 სტრიქონს, m = 6 სვეტს, ხოლო ელემენტები გამოითვლება ფორმულით: a[i][j] = i * j.

ასეთი მასივის შესაქმნელად გამოვიყენოთ გენერატორი:

[[i * j for j in range(m)] for i in range(n)]

დავალება 1

ამოცანა:

მოცემულია სამი სტუდენტის შეფასება 4 საგანში ცხრილის სახით:

სტუდენტი	საგანი 1	საგანი 2	საგანი 3	საგანი 4
სტუდენტი 1	22	18	25	17
სტუდენტი 2	11	21	18	23
სტუდენტი 3	24	16	22	15

მოცემული ცხრილი, წარმოდგენილი ორგანზომილებიანი მასივის სახით გამოიყურება შემდეგნაირად:

m[0,0]=22; m[0,1]=18; m[0,2]=25; m[0,3]=17;

m[1,0]=11; m[1,1]=21; m[1,2]=18; m[1,3]=23;

m[2,0]=24; m[2,1]=16; m[2,2]=22; m[2,3]=15;

განსაზღვრეთ და გამოიტანეთ კონსოლზე:

1. მე-2 სტუდენტის შეფასება მე-4 საგანში, ხოლო მე-3 სტუდენტის შეფასება 1-ელ საგანში;
2. ყველა სტუდენტის შეფასება მე-2 საგანში;
3. მე-3 სტუდენტის საშუალო ქულა;
4. მაქსიმალური შეფასება;
5. მინიმალური შეფასება.
6. მე-2 სტუდენტის ქულათა ჯამი.

დავალება 2

ჩადგმული გენერატორების გამოყენებით მიიღეთ შემდეგი ორგანზომილებიანი მასივი:

```
1 2 3 4 5 6
2 3 4 5 6 7
3 4 5 6 7 8
4 5 6 7 8 9
5 6 7 8 9 10
```

დავალება 3

მიიღეთ 4x5 მატრიცა მთელი შემთხვევითი რიცხვებისგან (20, 80) დიაპაზონიდან. გამოითვალეთ და გამოიტანეთ ცალკეული სტრიქონის ელემენტების ჯამი.

დავალება 4

შექმნით 4x4 მატრიცა, რომლის არამთავარი დიაგონალი შევსებულია რიცხვი 1-ით.

დავალება 5

მოცემულია მატრიცა:

```
matrix ([[7, 5, 1],
         [-1, 3, 4],
          [9, 8, -2],
           [0, 11, -5]])
```

გამოიტანეთ მატრიცის ელემენტების ჯამი, უდიდესი ელემენტი და უმცირესი ელემენტი.

დავალება 6

დავალება 5-ში მოცემული გარდაქმენით ბრტყელი სახით და გამოიტანეთ კონსოლზე.

დავალება 7

დავალება 6-ში ბრტყელი სახით გარდაქმნილ მატრიცას ფორმა შეუცვალეთ და წარმოადგინეთ (2,6) სახით.

ლაბორატორიული სამუშაო 9
თემა: ფაილები

დასამუშავებლი საკითხები:

- ფაილის გახსნა და დახურვა;
- ტექსტურ ფაილებთან მუშაობა;
- ფაილის წაკითხვა;
- CSV ფაილები;
- OS მოდული და მუშაობა ფაილურ სისტემასთან;
- ბინარული ფაილები;

ფაილის გახსნა და დახურვა

open(file, mode)

პირველი პარამეტრი წარმოადგენს გზას ფაილამდე. მაგალითად: C:\somedir\somefile.txt. შეიძლება მისამართი არ მიეთითოს. მაგალითად:
somefile.txt.

მეორე პარამეტრი - mode დააყენებს ფაილის გახსნის რეჟიმს.

hello.txt ტექსტური ფაილის გახსნა მისამართის მითითების გარეშე :

```
f = open("hello.txt", "w")
```

ფაილის დახურვა:

```
f.close()
```

ფაილის გახსნა მისამართის მითითებით, კერძოდ, D დისკზე შექმნილ საქაღალდეში:

```
f=open("D:\mydir\hello.txt", "w")
```

ტექსტურ ფაილებთან მუშაობა

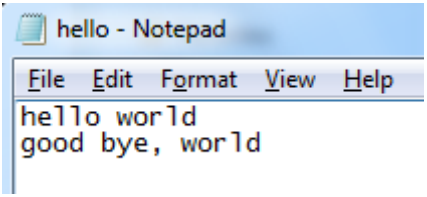
ჩაწერეთ რაიმე ინფორმაცია hello.txt ფაილში.

```
with open("D:\mydir\hello.txt", "w") as file:
    file.write("hello world")
```

დაამატეთ ფაილს კიდევ ერთი სტრიქონი:

```
with open("D:\mydir\hello.txt", "a") as file:
    file.write("\ngood bye, world")
```

hello.txt ფაილს ექნება შემდეგი შინაარსი:



ფაილში ჩაწერის კიდევ ერთი ხერხია print() სტანდარტული მეთოდი, რომელიც გამოიყენება მონაცემების გამოსატანად კონსოლზე:

```
with open("D:\mydir\hello.txt", "a") as hello_file:
    print("\nHello, world", file=hello_file)
```

print მეთოდში მეორე პარამეტრის სახით გადაეცემა ფაილის დასახელება file პარამეტრით, ხოლო პირველი პარამეტრი არის ფაილში ჩასაწერი სტრიქონი.

ფაილის წაკითხვა

ფაილის წაკითხვისას გამოყენებული მეთოდები:

- readline(): ფაილიდან კითხულობს ერთ სტრიქონს;
- read(): კითხულობს ფაილის მთელ შემადგენლობას ერთ სტრიქონად;

– readlines(): კითხულობს ფაილის ყველა სტრიქონს სიაში.

ზემოთ ჩაწერილი ფაილის სტრიქონ-სტრიქონ წაკითხვა:

```
with open("D:\mydir\hello.txt", "r") as file:
  for line in file:
  print(line, end="")
```

ცალკეული სტრიქონის წასაკითხად readline() მეთოდის ცხადი სახით გამოძახება:

```
with open("d:\mydir\hello.txt", "r") as file:
  str1 = file.readline()
  print(str1, end="")
  str2 = file.readline()
  print(str2)
```

გამოიტანს შედეგს:

```
hello world
good bye, world
```

Readline() მეთოდის გამოყენება ფაილის სტრიქონებად წასაკითხად while ციკლში:

```
with open("d:\mydir\hello.txt", "r") as file:
  line = file.readline()
  while line:
    print(line, end="")
    line = file.readline()
```

შედეგი:

```
hello world
good bye, world
Hello, world
```

მცირე ზომის ფაილის ერთიანად წაკითხვა read() მეთოდის გამოყენებით:

```
with open("d:\mydir\hello.txt", "r") as file:
    content = file.read()
    print(content)
```

readlines() მეთოდის გამოყენებით მთლიანი ფაილის წაკითხვა სტრიქონების სიის სახით:

```
with open("d:\mydir\hello.txt", "r") as file:
    contents = file.readlines()
    str1 = contents[0]
    str2 = contents[1]
    str3 = contents[2]
    print(str1, end="")
    print(str2, end="")
    print(str3)
```

ფაილის წაკითხვისას შეიძლება წავაწყდეთ იმ პრობლემას, რომ მისი კოდირება არ ემთხვეოდეს ASCII-ს. ამ შემთხვევაში ჩვენ ცხადი სახით შეგვიძლია მივუთითოთ კოდირება encoding პარამეტრის დახმარებით:

```
filename = " d:\hello.txt"
with open(filename, encoding="utf8") as file:
    text = file.read()
```

პროგრამა მომხმარებლის მიერ შეტანილ სტრიქონებს ჩაწერს ფაილში, შემდეგ მოხდება ფაილის შინაარსის წაკითხვა და ეკრანზე გამოტანა:

```
# ფაილის სახელი
FILENAME = "d:\messages.txt"
```

```
# განვსაზღვროთ ცარიელი სია
messages = list()
for i in range(4):
    message = input("შეიტანეთ სტრიქონი"+str(i+1) + ": ")
    messages.append(message + "\n")
 # სიის ჩაწერა ფაილში
with open(FILENAME, "a") as file:
    for message in messages:
        file.write(message)
 # წაკითხვა ფაილიდან
print("წაკითხული შეტყობინება")
with open(FILENAME, "r") as file:
    for message in file:
        print(message, end="")
```

მიიღება შედეგი:

```
== RESTART: C:/Users/Guliko/AppData/Local/P
შეიტანეთ სტრიქონი 1: Sandro
შეიტანეთ სტრიქონი 2: Lomjaria
შეიტანეთ სტრიქონი 3: Anna
შეიტანეთ სტრიქონი 4: Karden
წაკითხული შეტყობინება
Sandro
Lomjaria
Anna
Karden
>>>
```

CSV ფაილები

csv (Comma Separated Values) ფაილში ცალკეული სტრიქონი წარმოადგენს ცალკეულ ჩანაწერს ან სტრიქონს, რომელიც შედგება მძიმეებით გამოყოფილი ცალკეული

სვეტისგან. განვიხილოთ csv სპეციალური ჩაშენებული მოდულის მუშაობა მაგალითზე:

```
import csv
FILENAME = "users.csv"
users = [
    ["Tom", 28],
    ["Alice", 23],
    ["Bob", 34]
]
 with open(FILENAME, "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerows(users)
```

ფაილში ჩაიწერება ორგანზომილებიანი სია - ფაქტობრივად ცხრილი, სადაც ცალკეული სტრიქონი წარმოადგენს ერთ მომხმარებელს. ხოლო ცალკეული მომხმარებელი შეიცავს ორ ველს - სახელი და ასაკი. ფაქტობრივად ცხრილი შედგება სამი სტრიქონისა და ორი სვეტისგან.

ჩანაწერის დამატება ["Sam", 31]:

```
import csv
FILENAME = "users.csv"
with open(FILENAME, "a", newline="") as file:
    user = ["Sam ", 31]
    writer = csv.writer(file)
    writer.writerow(user)
```

users.csv ფაილს საბოლოოდ ექნება შემდეგი შემადგენლობა:

```
Tom,28
Alice,23
Bob,34
Sam,31
```

ფაილიდან წაკითხვისთვის reader ობიექტის შექმნა:

```
import csv
FILENAME = "users.csv"
with open(FILENAME, "r", newline="") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row[0], " - ", row[1])
```

reader ობიექტის მიღებისას შეიძლება ციკლში მისი ყველა სტრიქონის გადარჩევა:

```
Tom  -  28
Alice  -  23
Bob  -  34
Sam – 31
```

OS მოდული და მუშაობა ფაილურ სისტემასთან

განვიხილოთ **os** ჩაშენებული მოდულის ძირითადი მეთოდები:

- **mkdir()** - ქმნის ახალ საქაღალდეს;
- **rmdir()** - წაშლის საქაღალდეს;
- **rename()** - გადაარქმევს ფაილს;
- **remove()** - წაშლის ფაილს.

საქაღალდეს შექმნა:

```
import os
```

```
os.mkdir("d:\myfolder")
os.mkdir("d://myfolder")
```

საქაღალდეს წაშლა rmdir() ფუნქცია:

```
import os
os.rmdir("d:\myfolder")
```

ფაილის სახელის გადარქმევა:

```
os.rename("d:\mydir\hello.txt","d:\mydir\gamarjoba.txt")
```

ფაილის წაშლა:

```
os.remove("d:\mydir\gamarjoba.txt")
```

ბინარული ფაილები

მოდული pickle

pickle მოდული წარადგენს ორ მეთოდს:

- dump(obj, file) -ჩაწერს obj ობიექტს ბინარულ file ფაილში;
- load(file) - კითხულობს მონაცემებს ბინარული ფაილიდან.

ორი ობიექტის მიმდევრობით ჩაწერა ფაილში. ასევე, მიმდევრობით წაკითხვა ფაილიდან და გამოტანა კონსოლზე:

```
import pickle
FILENAME = "user.dat"
name = "თომა"
age = 19
with open(FILENAME, "wb") as file:
   pickle.dump(name, file)
   pickle.dump(age, file)
```

```
 with open(FILENAME, "rb") as file:
    name = pickle.load(file)
    age = pickle.load(file)
    print("სახელი:", name, "\tასაკი:", age)
```

შედეგი:

```
სახელი: თომა      ასაკი: 19
```

ობიექტების ნაკრების ჩაწერა ფაილში და ფაილიდან წაკითხვა:

```
import pickle
FILENAME = "users.dat"
 users = [
    ["თომა", 28, True],
    ["ელენე", 23, False],
    ["რემი", 34, False]
]
with open(FILENAME, "wb") as file:
    pickle.dump(users, file)
 with open(FILENAME, "rb") as file:
    users_from_file = pickle.load(file)
    for user in users_from_file:
        print("სახელი:", user[0], "\tასაკი:", user[1],
"\tდაქორწინებული:", user[2])
```

შედეგი:

```
სახელი: თომა     ასაკი: 28        დაქორწინებული: True
სახელი: ელენე    ასაკი: 23        დაქორწინებული: False
სახელი: რემი     ასაკი: 34        დაქორწინებული: False
```

დავალება 1

შექმენით D დისკზე myfolder საქაღალდე. გახსენით myfile1 ტექსტური ფაილი ჩასაწერად, რომელიც შეინახეთ თქვენ მიერ შექმნილ საქაღალდეში და მასში შეიტანეთ ნებისმიერი ერთი სტრიქონი.

დავალება 2

დავალება 1-ში შექმნილ ფაილში დაამატეთ რადენიმე სტრიქონი.

დავალება 3

თქვენ მიერ შექმნილი ფაილიდან წაიკითხეთ ცალკეული სტრიქონები და გამოიტანეთ.

დავალება 4

წაიკითხეთ თქვენ მიერ შექმნილი ფაილი ერთიანად და გამოიტანეთ.

დავალება 5

წაიკითხეთ ფაილი სტრიქონების სიის სახით და გამოიტანეთ.

დავალება 6

შექმენით იმავე საქაღალდეში მეორე - myfile2 ტექსტური ფაილი. myfile1-დან ტექსტი გადაწერეთ myfile2-ში. გამოიტანეთ ტექსტი ორივე ფაილიდან.

დავალება 7

ფაილს გადაარქვით სახელი, წაშალეთ ფაილი, წაშალეთ საქაღალდე.

დავალება 8

მოცემული ჩანაწერისგან შექმენით csv ფაილი:

["Kavsadze Megi", 550, 18],
["Askurava Gabi", 551, 19],
["Dadvani Semi", 552, 20]

დავალება 9

შექმნილ csv ფაილს დაამატეთ ერთი ახალი ჩანაწერი.

დავალება 10

წაიკითხეთ და გამოიტანეთ შექმნილი csv ფაილის შემადგენლობა.

დავალება 11

მოცემული ობიექტების ნაკრები ჩაწერეთ ფაილში. შემდეგ წაიკითხეთ ფაილიდან და გამოიტანეთ კონსოლზე:

```
students = [
  ["კავსაძე", 550, "ბაკალავრი"],
  ["არეშიძე", 551, "მაგისტრანტი"],
  ["რეკვავა", 552, "ბაკალავრი"]]
```

ლაბორატორიული სამუშაო 10
თემა: მოდულები

დასამუშავებელი საკითხები:

- მოდულის შექმნა;
- პაკეტის შექმნა;
- ძირითადი ჩაშენებული მოდულები;
- math მოდულის ზოგიერთი ფუნქციები.

მოდულის შექმნა

მოდულის შექმნა simplemath.py სახელით, რომელიც შეიცავს ფუნქციებს მარტივი არითმეტიკული მოქმედებების შესასრულებლად:

```
def add(a, b):
  return a + b

def sub(a, b):
  return a - b

def mul(a, b):
  return a * b

def div(a, b):
  return a / b
```

simplemath.py მოდულის ყველა ფუნქციის გამოძახება:

```
import simplemath
```

simplemath.py მოდულის ყველა ფუნქციის გამოძახება და შედეგის გამოტანა:

```
print("\"import\" sample")
```

```
print(simplemath.add(1, 2))
print(simplemath.sub(1, 2))
print(simplemath.mul(1, 2))
print(simplemath.div(1, 2))
```

მიიღება შედეგი:

```
3
-1
2
0.5
```

მოდულის ფსევდონიმის შექმნა:

```
import simplemath as s
```

მოდულის ყველა ფუნქციის გამოძახება ფსევდონიმის მეშვეობით:

```
print(s.add(1, 2))
print(s.sub(1, 2))
print(s.mul(1, 2))
print(s.div(1, 2))
```

მოდულიდან მხოლოდ ერთი, მაგალითად შეკრების, ფუნქციის გამოძახება:

```
from simplemath import add

print(add(1, 2))
```

მოცემულ შემთხვევაში სხვა ფუნქციის გამოძახება გამოიტანს შეცდომას, მაგალითად:

```
print(sub(1, 2))
NameError: name 'sub' is not defined
```

პაკეტის შექმნა

შექმენით mathpack საქაღალდე და მასში გადაიტანეთ simplemath.py მოდული. ამ შემთხვევაში simplemath.py მოდულის გამოძახება მიიღებს სახეს:

```
import mathpack. simplemath
print("\"import\" sample")
print(mathpack.simplemath.add(1, 2))
print(mathpack.simplemath.sub(1, 2))
print(mathpack.simplemath.mul(1, 2))
print(mathpack.simplemath.div(1, 2))
```

simplemath მოდულის იმპორტი შეიძლება მოდიფიცირდეს შემდეგი სახით:

```
from mathpack import simplemath
```

ასეთ შემთხვევაში, დანარჩენი კოდში არაფერი აღარ შეიცვლება:

```
from mathpack import simplemath

print("\"import\" sample")
print(simplemath.add(1, 2))
print(simplemath.sub(1, 2))
print(simplemath.mul(1, 2))
print(simplemath.div(1, 2))
```

შექმენით კიდევ ერთი მოდული: advmath.py და შეინახეთ mathpack საქაღალდეში:

```
def sqrt(value):
   return value**0.5

def pow(value, magn):
```

```
    return value**magn
```

advmath.py მოდულის გამოძახება:

```
import mathpack.advmath
```

მოცემული მოდულიდან ფუნქცის გამოძახება:

```
print(mathpack. advmath. sqrt(9))
```

ან:

```
from mathpack import advmath
print(advmath.sqrt(9))
```

ძირითადი ჩაშენებული მოდულები

```
import random
number = random.random()  # მნიშვნელობა 0.0-დან 1.0-მდე
print(number)

number = random.random() * 100  # მნიშვნელობა 0.0-დან 100.0-მდე
print(number)
```

```
import random
 number = random.randint(20, 35)  # მნიშვნელობა 20-დან 35-მდე
        print(number)
```

```
import random
number = random.randrange(10)  # მნიშვნელობა [0 – 10]
print(number)

number = random.randrange(2, 10) # მნიშვნელობა 2, 3, 4, 5, 6, 7, 8, 9, 10 დიაპაზონში
print(number)
```

```
number = random.randrange(2, 10, 2)  # მნიშვნელობა 2, 4, 6, 8, 10 დიაპაზონში
print(number)
```

სიებთან სამუშაოდ random მოდულში განსაზღვრულია ორი ფუნქცია: shuffle() ფუნქცია გადააადგილებს სიას შემთხვევითი სახით, ხოლო choice() ფუნქცია აბრუნებს სიიდან ერთ შემთხვევით ელემენტს:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
random.shuffle(numbers)
print(numbers) # [2, 6, 1, 7, 3, 8, 5, 4]
random_number = random.choice(numbers)
print(random_number) #7
```

math მოდულის ზოგიერთი ფუნქციები

```
import math

 # 2 -ის  3  ხარისხი
n1 = math.pow(2, 3)
print(n1)  # 8
 # იგივე ოპერაცია შეიძლება ასეც შესრულდეს
n2 = 2**3
print(n2)
 # ფესვის ამოღება
print(math.sqrt(9))  # 3
 # უახლოესი უდიდესი მთელი რიცხვი
print(math.ceil(4.56))  # 5
 # უახლოესი უმცირესი მთელი რიცხვი
print(math.floor(4.56))  # 4
```

```
 # რადიანიდან გრადუსში გადაყვანა
print(math.degrees(3.14159))  # 180
 # გრადუსიდან რადიანში გადაყვანა
print(math.radians(180))   # 3.1415.....
# კოსინუსი
print(math.cos(math.radians(60)))  # 0.5
# სინუსი
print(math.sin(math.radians(90)))   # 1.0
# ტანგენსი
print(math.tan(math.radians(0)))    # 0.0
# ლოგარითმი
print(math.log(8,2))    # 3.0
# ათობითი ლოგარითმი
print(math.log10(100))    # 2.0
```

math მოდული, აგრეთვე, წარადგენს მთელ რიგ ჩაშენებულ მუდმივებს, როგორიცაა PI და E:

```
import math
radius = 30
# 30 რადიუსიანი წრის ფართობი
area = math.pi * math.pow(radius, 2)
print(area)

 # 10-ის ნატურალური ლოგარითმი
number = math.log(10, math.e)
print(number)
```

დავალება 1

შექმენით მოდული, რომელიც შეიცავს სამ ფუნქცია: ორი რიცხვის საშუალო არითმეტიკულის პოვნა, ორი რიცხვიდან უდიდესის პოვნა და ორი რიცხვიდან უმცირესის პოვნა.

დავალება 2

დავალება 1-ში შექმნილი მოდულიდან გამოიძახეთ ყველა ფუნქცია.

დავალება 3

დავალება 1-ში შექმნილი მოდულიდან გამოიძახეთ ცალკეული ფუნქციები.

დავალება 4

შექმენით საქაღალდე, სადაც გადაიტანეთ დავალება 1-ში შექმნილი ფუნქცია. იმავე საქაღალდეში შეინახეთ ახალი მოდული, რომელიც შედგება ფუნქციებისგან: ორ ცვლადს შორის მნიშვნელობების გაცვლა, უარყოფითი რიცხვის მოდულის გამოთვლა.

დავალება 5

დავალება 4-ში შექმნილი მოდულიდან გამოიძახეთ ყველა ფუნქცია.

დავალება 6

დავალება 4-ში შექმნილი მოდულიდან გამოიძახეთ ცალკეული ფუნქციები.

ლაბორატორიული სამუშაო 11
თემა: გამონაკლისების დამუშავება

დასამუშავებელი საკითხები:

- try – except ბლოკი;
- try/except ბლოკის ფუნქციებთან გამოყენება;
- რამდენიმე გამონაკლისის გამოვლენა;
- finally ოპერატორის გამოყენება;
- try/except ოპერატორთან ერთად else გამოყენება;
- try/except ფაილებთან მუშაობის დროს;
- raise ოპერატორი.

try – except ბლოკი

განვიხილოთ ელემენტარული პრობლემა: 0-ზე გაყოფა, რომლის დროსაც წარმოიშვება გამონაკლისი:

```
1/0
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
ZeroDivisionError: integer division or modulo by zero
```

განვათავსოთ ეს გამოსახულება try – except ბლოკში:

```
try:
  1 / 0
except ZeroDivisionError:
  print("ნულზე გაყოფა შეუძლებელია!")
```

შედეგი:

```
ნულზე გაყოფა შეუძლებელია!
```

შეცდომის დაფიქსირების კიდევ ერთი ხერხი:

```
try:
    1 / 0
except:
    print("ნულზე გაყოფა შეუძლებელია!")
```

აქ არ ჩანს რომელ შეცდომაზეა ლაპარაკი. ამიტომ მოუხერხებელია.

try ბლოკი აგენერირებს გამონაკლისს, რადგან x არ არის განსაზღვრული:

```
try:
  print(x)
except:
  print("წარმოიშვა გამონაკლისი")
```

შედეგი:

```
წარმოიშვა გამონაკლისი
```

შევქმნათ ლექსიკონი სამი ელემენტისგან. ამის შემდეგ შევეცადოთ წვდომა გავუხსნათ გასაღებს, რომელიც არ არის ლექსიკონში. რადგან გასაღები არ არის ლექსიკონში, აღიძვრება KeyError, რასაც გამოვავლენთ პროგრამის კოდში:

```
my_dict = {"a":1, "b":2, "c":3}
try:
    value = my_dict["d"]
except KeyError:
    print("ეს გასაღები არ არსებობს!")
```

შედეგი:

```
ეს გასაღები არ არსებობს!
```

მაგალითში მოცემულია სია, რომლის სიგრძე შედგება 5 ობიექტისგან. შევეცადოთ მივმართოთ მეშვიდე ელემენტს:

```
my_list = [1, 2, 3, 4, 5]
try:
    my_list[6]
except IndexError:
    print("ეს ინდექსი არ არის სიაში!")
```

შედეგი:

```
ეს ინდექსი არ არის სიაში!
```

try/except ბლოკის ფუნქციებთან გამოყენება

მაგალითში გამონაკლისი არ ამუშავდება:

```
def divide(x, y):
    try:
        result = x // y
        print("პასუხი:", result)
    except ZeroDivisionError:
        print("ყურადღება! ნულზე გაყოფაა ")
```

გადავცეთ პარამეტრები, რომლის დროსაც except არ იმუშავებს:

```
divide(3, 2)
```

მაგალითში არის გამონაკლისი, ამიტომ except იმუშავებს:

```
def divide(x, y):
    try:
        result = x // y
        print("პასუხი:", result)
    except ZeroDivisionError:
```

```
    print("ყურადღება! ნულზე გაყოფაა")
```

გადავცეთ პარამეტრები, რომლის დროსაც except იმუშავებს

```
divide(3, 0)
```

შედეგი:

```
ყურადღება! ნულზე გაყოფაა
```

რამდენიმე გამონაკლისის გამოვლენა

თავდაპირველად ვახდენთ წვდომას არარსებულ გასაღებზე. try/except დახმარებით ვამოწმებთ კოდს KeyError შეცდომის არსებობაზე, რომელიც იმყოფება გამონაკლისის მეორე ოპერატორში. კოდის ბოლოში არის ცარიელი გამონაკლისი, რისი გამოყენებაც არ არის რეკომენდებული. მაგალითში მთელი ბლოკი გამოყენებულია ერთადერთი შეცდომის გამოსავლენად:

```
my_dict = {"a":1, "b":2, "c":3}
try:
    value = my_dict["d"]
except IndexError:
    print("ეს ინდექსი არ არსებობს!")
except KeyError:
    print("ეს გასაღები არ არის ლექსიკონში!")
except:
    print("მოხდა სხვა შეცდომა!")
```

შედეგი:

```
ეს გასაღები არ არის ლექსიკონში!
```

განვიხილოთ რამდენიმე გამონაკლისის გამოვლენის მეორე ხერხი:

```
my_dict = {"a":1, "b":2, "c":3}
try:
   value = my_dict["d"]
except (IndexError, KeyError):
   print("წარმოიშვა IndexError ან KeyError!")
```

შედეგი:

```
წარმოიშვა IndexError ან KeyError!
```

თუმცა აქ გაურკვეველია კერძოდ რომელი შეცდომა წარმოიშვა.

finally ოპერატორის გამოყენება

finally ბლოკი შესრულდება იმის მიუხედავად, გამოსცემს თუ არა try ბლოკი შეცდომას:

```
try:
  print(x)
except:
  print("რაღაც შეცდომაა")
finally:
  print("'try except' არის დასრულებული")
```

შედეგი:

```
რაღაც შეცდომაა
'try except' არის დასრულებული
```

მოცემულ სიაში გასაღები არ არის ნაპოვნი, რის გამოც წარმოიშვება KeyError გამონაკლისი:

```
my_dict = {"a":1, "b":2, "c":3}
try:
   value = my_dict["d"]
except KeyError:
```

```
    print("წარმოიშვა KeyError!")
finally:
    print("finally ოპერატორი შესრულებულია!")
```

შედეგი:

```
წარმოიშვა KeyError!
finally ოპერატორი შესრულებულია!
```

try/except ოპერატორთან ერთად else გამოყენება

else იმუშავებს მხოლოდ მაშინ, როდესაც კოდში არცერთი შეცდომა არ არის:

```
my_dict = {"a":1, "b":2, "c":3}
try:
    value = my_dict["a"]
except KeyError:
    print("წარმოიშვა KeyError შეცდომა!")
else:
    print("შეცდომა არ წარმოიშვა!")
```

შედეგი:

```
შეცდომა არ წარმოიშვა!
```

შევქმნათ ლექსიკონი სამი ელემენტისგან და try/except ოპერატორში გავხსნათ წვდომა არსებულ გასაღებზე. KeyError შეცდომა არ აღიძვრება, ამიტომ ამუშავდება else. კოდს დავამატოთ finally ოპერატორი:

```
my_dict = {"a":1, "b":2, "c":3}
try:
    value = my_dict["a"]
except KeyError:
```

```
    print("წარმოიშვა KeyError შეცდომა!")
else:
    print("შეცდომა არ წარმოიშვა!")
finally:
    print("finally ოპერატორი შესრულდა!")
```

შედეგი:

```
შეცდომა არ წარმოიშვა!
finally ოპერატორი შესრულდა!
```

try/except ფაილებთან მუშაობის დროს

შევეცადოთ გავხსნათ და ჩავწეროთ ფაილში, რომელიც არ არის მისაწვდომი ჩასაწერად:

```
try:
  f = open("demofile.txt")
  f.write("ტექნიკური უნივერსიტეტი")
except:
  print("შეცდომაა ფაილის ჩაწერაში")
```

შედეგი:

```
შეცდომაა ფაილის ჩაწერაში
```

მოცემულ შემთხვევაში ფაილთან მუშაობა ჩაშენებულია try ბლოკში. თუ აღიძვრება რაიმე გამონაკლისი, გამოვა „შეცდომაა“, ნებისმიერ შემთხვევაში finally ბლოკში ფაილი დაიხურება, თუ იგი არსებობს:

```
try:
    somefile = open("hello.txt", "w")  #გაიხსნა ფაილი ჩასაწერად
    somefile.write("hello world") #ჩაიწერა ფაილში
except:
     print("შეცდომაა")
finally:
       somefile.close() #დაიხურა ფაილი
```

raise ოპერატორი

ზოგჯერ საჭიროა ხელით დაგენერირდეს ესათუის გამონაკლისი. ამისათვის გამოიყენება raise ოპერატორი:

```
try:
    number1=int(input("შეიტანეთ პირველი რიცხვი:"))
    number2 = int(input("შეიტანეთ მეორე რიცხვი: "))
    if number2 == 0:
            raise Exception("მეორე რიცხვი არ უნდა იყოს 0")
            print("ორი რიცხვის გაყოფის შედეგი:",
    number1/number2)
except ValueError:
            print("შეტანილია არაკორექტული მონაცემები")
except Exception as e:
            print(e)
print("პროგრამის დასრულება")
```

შეიტანეთ ორი რიცხვი, გამონაკლისი არ ამუშავდება და მიიღება შედეგი:

```
შეიტანეთ პირველი რიცხვი:4
შეიტანეთ მეორე რიცხვი: 2
ორი რიცხვის გაყოფის შედეგი: 2.0
პროგრამის დასრულება
```

შეიტანეთ მნიშვნელში ნული, ამუშავდება raise ოპერატორი:

```
შეიტანეთ პირველი რიცხვი:3
შეიტანეთ მეორე რიცხვი: 0
მეორე რიცხვი არ უნდა იყოს 0
პროგრამის დასრულება
```

შეიტანეთ არაკორექტული მონაცემები, ამუშავდება ValueError:

```
შეიტანეთ პირველი რიცხვი:2
შეიტანეთ მეორე რიცხვი: a
შეტანილია არაკორექტული მონაცემები
პროგრამის დასრულება
```

დავალება 1

მოცემულია პროგრამის კოდი:

```
a = float(input("შეიტანეთ რიცხვი"))
print(100/a)
```

გამოიყენეთ კოდში try-except ბლოკი, რათა შემთხვევით შეტანილი სიმბოლოების (არა რიცხვის) შემთხვევაში გამოვიდეს შეტყობინება: „ეს რიცხვი არ არის“

დავალება 2

მოცემულია პროგრამის კოდი:

```
a = float(input("შეიტანეთ რიცხვი"))
print(100/a)
```

გამოიყენეთ კოდში try-except ბლოკი, რათა შეტანილი 0-ის შემთხვევაში გამოვიდეს შეტყობინება: „ნულზე გაყოფა დაუშვებელია“

დავალება 3

დავალება1-სა და დავალება 2-ის მიხედვით შექმენით ერთიანი კოდი, სადაც ერთი try ბლოკი შეიცავს ორივე except ბლოკს.

დავალება 4

დავალება 3-ში except ბლოკს დაამატეთ else ბლოკი, რომელიც ამუშავდება მაშინ, თუ პროგრამა შესრულდება უშეცდომოდ.

დავალება 5

დავალება 4-ში კოდს დაამატეთ finally ბლოკი, რომელიც გამოიტანს: „ავმუშავდები ყველა შემთხვევაში“

ლაბორატორიული სამუშაო 12
თემა: ობიექტზე ორიენტირებული დაპროგრამება (1 ნაწილი)

კლასები, ობიექტები, ინკაფსულაცია

დასამუშავებელი საკითხები:

- კლასის და ობიექტის შექმნა;
- კლასის კონსტრუქტორი, დესტრუქტორი, მეთოდები;
- სტატიკური ცვლადის გამოყენება კლასის მეთოდში;
- სტატიკური მეთოდი;
- ინკაფსულაცია, კერძო წვდომის ატრიბუტები.

კლასის და ობიექტის შექმნა

ცარიელი კლასის შექმნა

```
class Parrot:
   pass
```

ობიექტის შექმნა:

```
obj = Parrot()
```

კლასის კონსტრუქტორი, მეთოდები

შევქმნათ კლასი კონსტრუქტორით და მეთოდებით:

```
class Employee:
  empCount = 0
  def __init__(self, name, salary):
    self.name = name
    self.salary = salary
    Employee.empCount += 1
   def displayCount(self):
   print ("Total Employee %d" % Employee.empCount)
```

```
  def displayEmployee(self):
     print ("Name : ", self.name,  ", Salary: ", self.salary)
```

ობიექტების შექმნა:

```
emp1 = Employee("Zara", 2000)
emp2 = Employee("Manni", 5000)
```

მეთოდებზე წვდომა:

```
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

მთლიანი პროგრამა:

```
class Employee:
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
   def displayCount(self):
     print ("Total Employee %d" % Employee.empCount)
   def displayEmployee(self):
      print ("Name : ", self.name,  ", Salary: ", self.salary)
emp1 = Employee("Zara", 2000)
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

შედეგი:

```
Name :  Zara , Salary:  2000
Name :  Manni , Salary:  5000
Total Employee 2
```

კლასების ატრიბუტები შეიძლება დაემატოს, წაიშალოს ან შეიცვალოს ნებისმიერ დროს:

```
emp1.age = 7  # 'age' ატრიბუტის დამატება.
emp1.age = 8  # 'age' ატრიბუტის შეცვლა
del emp1.age  # 'age' ატრიბუტის წაშლა
```

ატრიბუტებზე წვდომისთვის ჩვეულებრივი ოპერატორების გამოყენების ნაცვლად შეიძლება შემდეგი ფუნქციების გამოყენება:

```
hasattr(emp1, 'age')    # აბრუნებს true-ს თუ  'age' ატრიბუტი არის
getattr(emp1, 'age')    # აბრუნებს 'age' ატრიბუტის მნიშვნელობას
setattr(emp1, 'age', 8) # 'age' ატრიბუტზე 8-ის მინიჭება
delattr(empl, 'age')    # 'age' ატრიბუტის წაშლა
```

დესტრუქტორი

__del __ () დესტრუქტორია, იგი ბეჭდავს კლასის ეგზემპლარის სახელს, რომელიც უნდა განადგურდეს:

```
class Point:
  def __init__( self, x=0, y=0):
    self.x = x
    self.y = y
  def __del__(self):
    class_name = self.__class__.__name__
```

```
    print (class_name, "destroyed")
pt1 = Point()
pt2 = pt1
pt3 = pt1
print ( id(pt1), id(pt2), id(pt3)) # ბეჭდავს ობიექტების id-ებს
del pt1
del pt2
del pt3
```

შედეგი:

```
5024848 5024848 5024848
Point destroyed
```

მოცემულია კლასი BankAccount, ვქმნით ორ ობიექტს და ვიძახებთ მეთოდებს:

```
class BankAccount:
   def __init__(self):
      self.balance = 0
   def withdraw(self, amount):
      self.balance -= amount
     return self.balance
   def deposit(self, amount):
      self.balance += amount
      return self.balance
 a = BankAccount()
 b = BankAccount()
print( a.deposit(100))  #100
print (b.deposit(50)) #50
print (b.withdraw(10)) #40
```

```
print (a.withdraw(10)) #90
```

სტატიკური ცვლადის გამოყენება კლასის მეთოდში

```
class A():
       def __init__(self):
         self.cnt = 0
      def f(self):
         self.cnt += 1
         return self.cnt
 a = A()
```

შედეგი:

```
 print (" Class A testing")
 print (a.f())
 print (a.f())
print (a.f())
```

შედეგი:

```
1
2
3
```

სტატიკური მეთოდი

სტატიკურ მეთოდის გამოძახება ობიექტის შექმნის გარეშე:

```
class MyClass(object):
    @staticmethod
    def the_static_method(x):
        print (x)
MyClass.the_static_method(2) # outputs 2
```

ინკაფსულაცია, კერძო წვდომის ატრიბუტები

მაგალითში გამოყენებულია საერთო წვდომის count მონაცემი, რომელსაც ობიექტი მიმართავს პირდაპირი წვდომით:

```
class B:
   count = 0
   def __init__(self):
      B.count+=1
a = B()
print(a.count) #1
d=a.count+5
print(d) # 6
```

count მონაცემზე საერთო წვდომა შევცვალოთ კერძო წვდომით, ობიექტის პირდაპირი მიმართვა მოგვცემს შეცდომას:

```
class B:
   __count = 0
   def __init__(self):
      B.count+=1
a = B()
d=a.count+5
print(d)
```

მაგალითში გამოყენებულია საჯარო წვდომის მონაცემები:

```
class Person:
   def __init__(self, name, age):
      self.name = name    # ვაყენებთ სახელს
      self.age = age      # ვაყენებთ ასაკს
```

```
    def display_info(self):
        print("სახელი:", self.name, "\tასაკი:", self.age)
tom = Person("თომა", 23)
tom.display_info()
tom.name="დელფინი"  #ვცვლით name ატრიბუტს
tom.age = -19               # ვცვლით age ატრიბუტს
tom.display_info()
```

შედეგი:

```
სახელი: თომა      ასაკი: 23
სახელი: დელფინი           ასაკი: -19
```

მაგალითი კერძო წვდომის მონაცემებით:

```
class Person:
    def __init__(self, name, age):
        self.__name = name  # ვაყენებთ სახელს
        self.__age = age      # ვაყენებთ ასაკს
    def set_age(self, age):
        if age in range(1, 100):
            self.__age = age
        else:
            print("დაუშვებელი ასაკი")
    def get_age(self):
        return self.__age
    def get_name(self):
        return self.__name
    def display_info(self):
        print("სახელი:",self.__name, "\tასაკი:", self.__age)
tom = Person("თომა", 23)
tom.display_info()
```

```
tom.__age = 43          # age ატრიბუტი არ შეიცვლება
tom.display_info()      # სახელი: თომა  ასაკი: 23
tom.set_age(-486)       # დაუშვებელი ასაკი
tom.set_age(25)
tom.display_info()      # სახელი: თომა  ასაკი: 25
```

დავალება 1

შექმენით კლასი თანამშრომელი, საჯარო ტიპის მონაცემებით: გვარი, სახელი, ასაკი, მუშაობის სტაჟი, ხელფასი. გამოიყენეთ კონსტრუქტორი ობიექტის ინიციალიზებისთვის. შექმენით მეთოდი აღნიშნული მონაცემების გამოსატანად. გამოაცხადეთ მოცემული კლასის ერთი ობიექტი, მეთოდის მეშვეობით გამოიტანეთ მონაცემები თანამშრომლის შესახებ. შემდგომ შეცვალეთ თანამშრომლის მუშაობის სტაჟი და ხელფასი და ხელახლა გამოიტანეთ თანამშრომლის შესახებ მონაცემები. შედეგების ანალიზიდან გამომდინარე გააკეთეთ დასკვნები.

დავალება 2

შექმენით კლასი სტუდენტი, კერძო ტიპის მონაცემებით: გვარი, სახელი და შეფასება სამ საგანში. გამოიყენეთ კონსტრუქტორი ობიექტის ინიციალიზებისთვის. შექმენით მეთოდი აღნიშნული მონაცემების გამოსატანად. გამოაცხადეთ მოცემული კლასის ერთი ობიექტი, მეთოდის მეშვეობით გამოიტანეთ მონაცემები სტუდენტის შესახებ. შემდგომ შეცვალეთ სტუდენტის ქულები სამივე საგანში და ხელახლა გამოიტანეთ სტუდენტის მონაცემები. შედეგების ანალიზიდან გამომდინარე გააკეთეთ დასკვნები. პროგრამას დაამატეთ ფუნქცია, რომელიც გამოითვლის და გამოიტანს სტუდენტის საშუალო ქულას.

ლაბორატორიული სამუშაო 13

თემა: ობიექტზე ორიენტირებული დაპროგრამება (2 ნაწილი)

მემკვიდრეობითობა, პოლიმორფიზმი, აბსტრაქტული მეთოდები

დასამუშავებელი საკითხები:

- მარტივი მემკვიდრეობითობა;
- მემკვიდრე კლასში super() ფუნქციის გამოყენება;
- მრავლობითი მემკვიდრეობითობა;
- super () ფუნქციის გამოყენება მრავლობითი მემკვიდრეობითობის დროს;
- mro() მეთოდი;
- პოლიმორფიზმი;
- პოლიმორფიზმი მემკვიდრეობითობაში;
- აბსტრაქტული მეთოდი.

მარტივი მემკვიდრეობითობა

შევქმნათ საბაზო კლასი Person:

```
class Person:
    name = ""
    age = 0
 def __init__(self, person_name, person_age):  #კონსტრუქტორი
        self.name = person_name
        self.age = person_age
    def show_name(self):   # მეთოდი
        print(self.name)
    def show_age(self):  #მეთოდი
```

```
        print(self.age)
```

შევქმნათ Person კლასის მემკვიდრე კლასი - Student:

```
class Student(Person):
    studentId = ""
    def __init__(self, student_name, student_age, student_id):
        Person.__init__(self, student_name, student_age)
        self.studentId = student_id
    def get_id(self):
        return self.studentId
```

ვაცხადებთ Person და Student კლასების ობიექტებს და ვიძახებთ მეთოდებს:

```
person1 = Person("Richard", 23)
person1.show_age()
student1 = Student("Max", 22, "102")
print(student1.get_id())
student1.show_name()
```

შედეგი:

```
23
102
Max
```

მოცემულ მაგალითში მშობელი კლასის ფუნქცია გამოვიძახეთ, როგორც:

```
Person.__init__(self, student_name, student_age)
```

ეს შეიძლება შევცვალოთ სუპერფუნქციის გამოძახებით:

```
super().__init__(student_name, student_age)
```

მემკვიდრე კლასში super() ფუნქციის გამოყენება

შევქმნათ საბაზო კლასი: Animal

```
class Animal:
    # კონსტრუქტორი
   def __init__(self, name):
     # Animal კლასს აქვს 1 ატრიბუტი: 'name'.
       self.name= name
   def showInfo(self):
       print ("I'm "+ self.name)
   def move(self):
       print ("moving ...")
```

შევქმნათ Animal კლასის მემკვიდრე Cat კლასი, რომელსაც აქგრეთვე აქვს საბაზო კლასის ატრიბუტები:

```
class Cat (Animal):
    def __init__(self, name, age, height):
       # გამოიძახებს Animal მშობელი კლასის კონსტრუქტორს,
       #რათა name - მშობელი კლასის ატრიბუტს მიენიჭოს მნიშვნელობა
        super().__init__(name)
        self.age = age
       self.height = height
   # ხელახლა განსაზღვროთ (override) მშობელი კლასის მეთოდი იგივე სახელით
     def showInfo(self):
       print ("I'm " + self.name)
```

```
        print (" age " + str(self.age))
        print (" height " + str(self.height))
```

შევქმნათ Cat კლასის ობიექტი და გამოვიძახოთ მეთოდები:

```
tom = Cat("Tom", 3, 20)
tom.move()
tom.showInfo()
```

შედეგი:

```
moving ...
I'm Tom
 age 3
 height 20
```

მრავლობითი მემკვიდრეობითობა

შევქმნათ Horse და Donkey კლასები, მათგან მივიღოთ მემკვიდრე კლასი - Mule:

```
class Horse:
    maxHeight = 200; # სანტიმეტრი
    def __init__(self, name, horsehair):
        self.name = name
        self.horsehair = horsehair
    def run(self):
        print ("Horse run")
    def showName(self):
        print ("Name: (House's method): ", self.name)
    def showInfo(self):
        print ("Horse Info")
class Donkey:
    def __init__(self, name, weight):
```

```
        self.name = name
        self.weight = weight
    def run(self):
        print ("Donkey run")
    def showName(self):
        print ("Name: (Donkey's method): ", self.name)
    def showInfo(self):
        print ("Donkey Info")
# Mule კლასი მემკვიდრეობითობას იღებს Horse და Donkey კლასებიდან
class Mule(Horse, Donkey):
    def __init__(self, name, hair, weight):
        Horse.__init__(self, name, hair)
        Donkey.__init__(self, name, weight)
    def run(self):
        print ("Mule run")
    def showInfo(self):
        print ("-- Call Mule.showInfo: --")
        Horse.showInfo(self)
        Donkey.showInfo(self)
# ---- Test ----------------------------------
# 'maxHeight' ცვლადი მემკვიდრეობით მიღებულია Horse კლასიდან
print ("Max height ", Mule.maxHeight)
mule = Mule("Mule", 20, 1000)
mule.run()
mule.showName()
mule.showInfo()
```

პროგრამის შედეგი:

```
 RESTART: C:/Users/Guliko/AppData/Local
dr.py
Max height  200
Mule run
Name: (House's method):  Mule
-- Call Mule.showInfo: --
Horse Info
Donkey Info
>>>
```

super () ფუნქციის გამოყენება მრავლობითი მემკვიდრეობითობის დროს

```
class A:
    def __init__(self):
        print('ინიციალიზდება: class A')
    def sub_method(self, b):
        print('დაიბეჭდება class A-დან:', b)
class B(A):
    def __init__(self):
        print('ინიციალიზდება: class B')
        super().__init__()
    def sub_method(self, b):
        print('დაიბეჭდება class B-დან:', b)
        super().sub_method(b + 1)
class C(B):
    def __init__(self):
        print('ინიციალიზდება: class C')
        super().__init__()
  def sub_method(self, b):
```

```
        print('დაიბეჭდება  class C-დან:', b)
        super().sub_method(b + 1)
if __name__ == '__main__':
    c = C()
    c.sub_method(1)
```

შედეგი:

```
ინიციალიზდება: class C
ინიციალიზდება: class B
ინიციალიზდება: class A
დაიბეჭდება class C-დან: 1
დაიბეჭდება class B-დან: 2
დაიბეჭდება class A-დან: 3
```

mro() მეთოდი

mro() მეთოდის გამოყენებით დავათვალიეროთ მშობელი კლასების სია:

```
class X: pass
class Y: pass
class Z: pass
class A(X,Y): pass
class B(Y,Z): pass
class M(B,A,Z): pass
print(M.mro())
```

პროგრამის შედეგი:

```
 RESTART: C:/Users/Guliko/AppData/Local/P
[<class '__main__.M'>, <class '__main__.B
n__.X'>, <class '__main__.Y'>, <class '__
>>>
```

პოლიმორფიზმი

პოლიმორფიზმის რეალიზება ფუნქციით:

მაგალითში შექმნილია ორი კლასი - English და French. ორივე კლასს აქვს greeting() მეთოდი. ორივე ქმნის სხვადასხვა მისალმებას. შექმნილია ორი სხვადასხვა ობიექტი მოცემული კლასებიდან და გამოძახებულია ამ ორი ობიექტის მოქმედება ერთ intro ფუნქციაში:

```
class English:
  def greeting(self):
     print ("Hello")
class French:
  def greeting(self):
     print ("Bonjour")
def intro(language):
   language.greeting()
flora  = English()
alase = French()
intro(flora)
intro(alase)
```

შედეგი:

```
 RESTART: C:/Users/Guliko/AppData/L
Hello
Bonjour
>>>
```

პოლიმორფიზმი მემკვიდრეობითობაში

sparrow და ostrich წარმოებულ კლასებში განსაზღვრულია flight ფუნქცია, ისევე როგორც საბაზო კლასში, მხოლოდ მათ გამოაქვთ განსხვავებული ტექსტები:

```
class Bird:
  def intro(self):
    print("არის ფრინველის  მრავალი სახეობა.")
  def flight(self):
    print("უმეტესობა ფრინველი დაფრინავს, თუმცა არა
ყველა.")
class sparrow(Bird):
  def flight(self):
    print("ბეღურას შეუძლია ფრენა")
class ostrich(Bird):
  def flight(self):
    print("სირაქლემა არ დაფრინავს")
obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
obj_spr.flight()

obj_ost.intro()
obj_ost.flight()
```

შედეგი:

```
არის ფრინველის  მრავალი სახეობა.
უმეტესობა ფრინველი დაფრინავს, თუმცა არა ყველა.
არის ფრინველის  მრავალი სახეობა.
ბეღურას შეუძლია ფრენა
არის ფრინველის  მრავალი სახეობა.
სირაქლემა არ დაფრინავს
```

აბსტრაქტული მეთოდი

შევქმნათ კლასი, რომელიც შეიცავს აბსტრაქტულ მეთოდს, ხოლო მემკვიდრე კლასებში აბსტრაქტული მეთოდი ხელახლა განვსაზღვროთ:

```
class Class1(object):
    def test(self, x):   # აბსტრაქტული მეთოდი
        pass
class Class2(Class1):      #მემკვიდრეობს აბსტრაქტულ მეთოდს
    def test(self, x):     # ახლიდან განვსაზღვროთ მეთოდი
        print (x)
class Class3(Class1):      #მემკვიდრეობს აბსტრაქტულ მეთოდს
    def test(self, x):     # ახლიდან განვსაზღვროთ მეთოდი
        print (x*x)
c2 = Class2()
c2.test(30)  # გამოიტანს 30
c3 = Class3()
c3.test(30)  # გამოიტანს 900
```

დავალება 1

შექმენით საბაზო კლასი მართკუთხედი, მონაცემებით: სიგრძე, სიგანე. შექმენით მეთოდი, რომელიც გამოითვლის და დაბეჭდავს მართკუთხედის ფართობს. შექმენით მართკუთხედი კლასის მემკვიდრე კლასი - მართკუთხა პარალელეპიპედი, მონაცემებით: სიგრძე, სიგანე, სიმაღლე. სიგრძე და სიგანე კონსტრუქტორმა უნდა გადმოიტანოს საბაზო კლასიდან. შემქნით მეთოდი, რომელიც გამოითვლის და გამოიტანს მართკუთხა პარალელეპიპედის ფართობს. შექმენით ორივე კლასის თითო ობიექტი და გამოიძახეთ შესაბამისი მეთოდები.

დავალება 1

შექმენით ორი საბაზო კლასი. აქედან ერთი კლასი - ბაკალავრი, მონაცემებით: გვარი, სახელი, კურსი, ხოლო მეორე კლასი - მაგისტრანტი, მონაცემებით: გვარი, სახელი, კურსი, სამაგისტრო თემის დასახელება. ასევე, საბაზო კლასებში შექმენით მეთოდები შესაბამისი მონაცემების გამოსატანად. ამ ორი საბაზო კლასიდან შექმენით წარმოებული კლასი დოქტორანტი, მონაცემებით: გვარი, სახელი, კურსი, სამაგისტრო თემის დასახელება და გამოცემული შრომების რაოდენობა. ასევე, შექმენით მეთოდი მონაცემების გამოსატანად. შექმენით ცალკეული კლასის თითო ობიექტი და გამოიძახეთ შესაბამისი მეთოდები.

ლაბორატორიული სამუშაო 14
თემა: თარიღსა და დროსთან მუშაობა

დასამუშავებელი საკითხები:

- თარიღის მიღება სხვადასხვა რაოდენობის პარამეტრით;
- თარიღის კონვერტირება სტრიქონად;
- თარიღის გამოტანა მითითებული ფორმატით;
- ოპერაციები თარიღებზე;
- თარიღების შედარება;
- დროის გამოტანა სხვადასხვა რაოდენობის პარამეტრით;
- კოდის შესრულების დაყოვნება განსაზღვრული დროით.

თარიღის მიღება სამი პარამეტრით: წელი, თვე, რიცხვი:

```
import datetime
a = datetime.datetime(2019, 9, 9)
print(a)
```

```
== RESTART: C:/Users/Guliko/AppData
2019-09-09 00:00:00
```

დღევანდელი თარიღის მიღება:

```
print(datetime.date.today())
```

```
== RESTART: C:/Users/Guliko/
2019-09-09
```

datetime.datetime გამოყენების რამდენიმე ვარიანტი:

```
import datetime
a = datetime.datetime(2019, 9, 9)
print(a)
b = datetime.datetime(2019, 9, 9, 20, 10)
```

```
print(b)
d = datetime.datetime(2019, 9, 9, 20, 10, 15)
print(d.year)
print(d.second)
print(d.hour)
```

```
== RESTART: C:/Users/Guliko/I
2019-09-09 00:00:00
2019-09-09 20:10:00
2019
15
20
```

დღევანდელი თარიღის კონვერტირება სტრიქონად:

```
import datetime
a = datetime.datetime.today().strftime("%Y%m%d")
print(a)
```

```
== RESTART: C:/Users/Guliko/I
20190909
```

თარიღის გამოტანა მითითებული ფორმატით:

```
import datetime
today = datetime.datetime.today()
print( today.strftime("%m/%d/%Y") )
```

```
== RESTART: C:/Users/Guliko/
09/09/2019
```

თარიღის გამოტანა განსხვავებული ფორმატით:

```
import datetime
today = datetime.datetime.today()
print( today.strftime("%Y-%m-%d-%H.%M.%S") )
```

```
== RESTART: C:/Users/Guliko/
2019-09-09-09.07.15
```

თარიღებს შორის სხვაობის გამოთვლა:

```
import datetime
now = datetime.datetime.now()
then = datetime.datetime(2019, 8, 21)
delta = now - then
print(delta.days)
print(delta.seconds)
```

```
== RESTART: C:/Users/Guliko/
19
33267
```

თარიღის მიღება, რომელიც იქნება 7 დღის შემდეგ:

```
from datetime import timedelta, datetime
now = datetime.now()
print(now)
sewen_days = timedelta(7)
in_sewen_days = now + sewen_days
print(in_sewen_days)
```

```
== RESTART: C:/Users/Guliko/
2019-09-09 09:22:33.916458
2019-09-16 09:22:33.916458
```

თარიღების შედარება. კოდში თარიღის შეცვლით მიიღეთ სხვადასხვა შედეგი:

```
from datetime import datetime
now = datetime.now()
deadline = datetime(2019, 1, 1)
```

```
if now > deadline:
  print("გამოცდის ჩაბარების ვადა გავიდა")
elif now.day == deadline.day and now.month == deadline.month
and now.year == deadline.year:
  print("გამოცდის ჩაბარების ვადა დღეს მთავრდება")
else:
 period = deadline - now
 print("დარჩა {} დღე".format(period.days))
print("ეხლა არის:", now)
```

```
== RESTART: C:/Users/Guliko/AppData/L
გამოცდის ჩაბარების ვადა გავიდა
ეხლა არის: 2019-09-09 09:33:08.902777
```

დროის გამოტანა სხვადასხვა რაოდენობის პარამეტრით:

```
from datetime import time
current_time = time()
print(current_time)
current_time = time(12, 15)
print(current_time)
current_time = time(12, 15, 35)
print(current_time)
```

```
== RESTART: C:/Users/Guliko/
00:00:00
12:15:00
12:15:35
```

კოდის შესრულების დაყოვნება, მოცემულ შემთხვევაში დააყოვნებს 2 წამით:

```
import time
for x in range(5):
```

```
time.sleep(2)
print("Slept for 2 seconds")
```

```
== RESTART: C:/Users/Guliko/A
Slept for 2 seconds
Slept for 2 seconds
Slept for 2 seconds
Slept for 2 seconds
Slept for 2 seconds
```

დავალება 1

მიიღეთ თარიღი დღევანდელი მონაცემების: წელი, თვე, რიცხვის მითითებით.

დავალება 2

გამოიტანეთ დღევანდელი თარიღი. გამოიტანეთ დრო სხვადასხვა როდენობის პარამეტრით.

დავალება 3

დღევანდელი თარიღს გაუკეთეთ კონვერტირება სტრიქონის სახით.

დავალება 4

დღევანდელი თარიღი გამოიტანეთ ისეთი სახით, რომ თვეს, რიცხვსა და წელს შორის იყოს „ტირე“.

დავალება 5

განსაზღვრეთ სხვაობა დღევანდელ თარიღსა და ამავე წლის 1 აგვისტოს შორის.

დავალება 6

განსაზღვრეთ თარიღი, რომელიც მიიღება 3 დღის შემდეგ.

დავალება 7

შეიტანეთ ნებისმიერი რიცხვი, რომლის გამოტანა დაყოვნდეს 3 წამით.

ლაბორატორიული სამუშაო 15

თემა: გრაფიკული ინტერფეისის შექმნა

დასამუშავებელი საკითხები:

- მარტივი ფანჯრის შექმნა;
- ფანჯარაზე ღილაკის დამატება;
- ღილაკის ატრიბუტების განსაზღვრა;
- ღილაკზე დაჭერის დამუშავება;
- Label - ტექსტური ჭდე;
- შეტანის ველის შექმნა;
- Checkbutton შექმნა;
- Radiobutton შექმნა;
- Listbox შექმნა;
- მენიუს შექმნა;
- მენიუში ქვემენიუს დამატება.

მარტივი ფანჯრის შექმნა

```
from tkinter import *
root = Tk()
root.title("გარაფიკული პროგრამა Python-ზე")
root.geometry("400x300")
root.mainloop()
```

ფანჯარაზე ღილაკის დამატება

```
from tkinter import *
root = Tk()
root.title("GUI - Python")
root.geometry("300x250")
btn = Button(text="Hello")
```

```
btn.pack()
root.mainloop()
```

ღილაკის ატრიბუტების განსაზღვრა

```
from tkinter import *
root = Tk()
root.title("GUI -Python")
root.geometry("300x250")
btn = Button(text="Hello",     # ღილაკის ტექსტი
   background="#555",# ღილაკის ფონის ფერი
   foreground="#ccc",# ტექსტის ფერი
     padx="20",  #შეწევა საზღვრიდან ჰორიზონტალურად
     pady="8",   # შეწევა საზღვრიდან  ვერტიკალურად
     font="16" )       # შრიფტის სიმაღლე
btn.pack()
root.mainloop()
```

ღილაკზე დაჭერის დამუშავება

```
from tkinter import *
clicks = 0
def click_button():
   global clicks
   clicks += 1
   root.title("Clicks {}".format(clicks))
root = Tk()
root.title("GUI - Python")
root.geometry("300x250")
btn=Button(text="Click Me", background="#555",
 foreground="#ccc",
```

```
            padx="20",pady="8",font="16",
command=click_button)
    btn.pack()
    root.mainloop()
```

Label - ტექსტური ჭდე

```
    from tkinter import *
    root = Tk()
    root.title("GUI - Python")
    root.geometry("300x250")
    label1 = Label(text="მუხრან მაჭავარიანი", fg="#eee",
        bg="#333")
    label1.pack()
    poetry = "ქართული მარტო ენაა?!\nქართული
ქართველთ რწმენაა!\nღმერთია!\nბედისწერაა!\nზღვა
როა!\nიმოდენაა!"
    label2 = Label(text=poetry, justify=LEFT)
    label2.place(relx=.2, rely=.3)
root.mainloop()
```

შეტანის ველის შექმნა

```
    from tkinter import *
    from tkinter import messagebox
    def show_message():
       messagebox.showinfo("GUI Python", message.get())
    root = Tk()
    root.title("GUI - Python")
    root.geometry("300x250")
    message = StringVar()
    message_entry = Entry(textvariable=message)
```

```
message_entry.place(relx=.5, rely=.1, anchor="c")
message_button = Button(text="Click Me",
      command=show_message)
message_button.place(relx=.5, rely=.5, anchor="c")
 root.mainloop()
```

შევქმნათ უფრო რთული მაგალითი შეტანის ფორმით:

```
from tkinter import *
from tkinter import messagebox
def display_full_name():
   messagebox.showinfo("GUI Python", name.get() + " " +
   surname.get())
root = Tk()
root.title("GUI - Python")
name = StringVar()
surname = StringVar()
name_label = Label(text="შეიტანეთ სახელი:")
surname_label = Label(text="შეიტანეთ გვარი:")
name_label.grid(row=0, column=0, sticky="w")
surname_label.grid(row=1, column=0, sticky="w")
name_entry = Entry(textvariable=name)
surname_entry = Entry(textvariable=surname)
name_entry.grid(row=0,column=1, padx=5, pady=5)
surname_entry.grid(row=1,column=1, padx=5, pady=5)
message_button = Button(text="Click Me",
     command=display_full_name)
```

```
message_button.grid(row=2,column=1, padx=5, pady=5,
    sticky="e")
root.mainloop()
```

Checkbutton შექმნა

```
from tkinter import *
root = Tk()
root.title("GUI - Python")
root.geometry("300x250")
ismarried = IntVar()
ismarried_checkbutton = Checkbutton(text="დაოჯახებული",
    variable=ismarried)
ismarried_checkbutton.pack()
ismarried_label = Label(textvariable=ismarried)
ismarried_label.place(relx=.5, rely=.5, anchor="c")
root.mainloop()
```

Radiobutton შექმნა

```
from tkinter import *
root = Tk()
root.title("GUI - Python")
root.geometry("300x250")
header = Label(text="აირჩიეთ კურსი",
        padx=15, pady=10)
header.grid(row=0, column=0, sticky=W)
lang = IntVar()
python_checkbutton = Radiobutton(text="Python",
        value=1, variable=lang, padx=15, pady=10)
python_checkbutton.grid(row=1, column=0, sticky=W)
```

```
javascript_checkbutton = Radiobutton(text="JavaScript",
        value=2, variable=lang, padx=15, pady=10)
javascript_checkbutton.grid(row=2, column=0, sticky=W)
selection = Label(textvariable=lang, padx=15, pady=10)
selection.grid(row=3, column=0, sticky=W)
root.mainloop()
```

Listbox შექმნა

```
from tkinter import *
languages = ["Python", "JavaScript", "C#", "Java"]
root = Tk()
root.title("GUI - Python")
root.geometry("300x280")
languages_listbox = Listbox()
for language in languages:
    languages_listbox.insert(END, language)
languages_listbox.pack()
root.mainloop()
```

მენიუს შექმნა

```
from tkinter import *
root = Tk()
root.title("GUI - Python")
root.geometry("300x250")
main_menu = Menu()
main_menu.add_cascade(label="File")
main_menu.add_cascade(label="Edit")
main_menu.add_cascade(label="View")
```

```
root.config(menu=main_menu)
root.mainloop()
```

მენიუში ქვემენიუს დამატება

```
from tkinter import *
root = Tk()
root.title("GUI - Python")
root.geometry("300x250")
main_menu = Menu()
file_menu = Menu()
file_menu.add_command(label="New")
file_menu.add_command(label="Save")
file_menu.add_command(label="Open")
file_menu.add_separator()
file_menu.add_command(label="Exit")
main_menu.add_cascade(label="File", menu=file_menu)
main_menu.add_cascade(label="Edit")
main_menu.add_cascade(label="View")
root.config(menu=main_menu)
root.mainloop()
```

დავალება 1

შექმენით ფორმა, რომელზეც განათავსეთ ოთხი შეტანის ველი და ერთი ღილაკი. შეტანის ველებში შეიტანეთ სტუდენტის: გვარი, სახელი, ასაკი. ღილაკზე დაწკაპებით სამივე მონაცემი გამოვიდეს მეოთხე შეტანის ველში.

დავალება 2

ფორმაზე დაამატეთ სამი Radiobutton, რომელზეც დაწკაპებით აირჩიეთ დაქორწინებულია თუ არა სტუდენტი.

დავალება 3

ფორმაზე დაამატეთ სამი Checkbutton, სადაც ჩამოთვალეთ არჩევითი საგნები, საიდანაც სტუდენტმა უნდა აირჩიოს მინიმუმ ორი საგანი.

დავალება 4

ფორმაზე დაამატეთ ტექსტური ჭდე, სადაც სტუდენტი შეიტანს თავის CV-ს.

დავალება 5

ფორმაზე გამოიტანეთ მენიუ: ბაკალავრიატი, მაგისტ-რატურა, დოქტორანტურა.

დავალება 6

მე-5 დავალებაში შექმნილ მენიუს დამატეთ ქვემენიუ ისე, რომ მიიღოთ შემდეგი სტრუქტურა: ბაკალავრიატი->1 კურსი, 2 კურსი, 3 კურსი, 4 კურსი; მაგისტრატურა->1 კურსი, 2 კურსი; დოქტორანტურა->1 კურსი, 2 კურსი, 3 კურსი.

ლიტერატურა

1. გულნარა ჯანელიძე, Python დაპროგრამების ენა, თბილისი, სტუ, 2018წ, ISBN 978-9941-8-0603-2, 302 გვ;
2. Charles R. Severance, Python for Everybody, Exploring Data Using Python 3, 2016, 247 pp;
3. Andrew Johansen, Python The Ultimate Beginner's Guide, 2016, 79pp;
4. Dave Kuhlman, A Python Book: Beginning Python, Advanced Python, and Python Exercises, 2013, 278pp;