Технически Университет – София,
Факултет „Компютърни системи и технологии“

Упражнение 4 – Създаване на приложения с използване на сложни данни: добавяне, изтриване и промяна на списък, търсене в списък, множество, работа с речник

Списък

Списъкът е номерирана съвкупност от елементи, може да съдържа елементи от всякакъв тип. При списъка може да достъпите даден елемент по индекс, като номерацията на елементите започва от 0(нула). Списъкът се отнася към изменяемите типове данни , т.е. може да промените елементите от списъка.

Пример :

```
s = [1, 3, 5, 7]
print(s[1])
s[1] = 11
print(s)
```

На първия ред от програмата създаваме списък с 4 елемента , на следващия ред принтираме стойността на елемента с индекс 1, след което присвояваме нова стойност на същия елемент и той вече има стойност 11.

Как създаваме списък?

1. Може да създадем списък използвайки функцията **list( )**, на която подаваме елементите на списъка, ако не подадем нищо тогава се създава празен списък.

   ```
   s = list()
   s1 = list('Python')
   print(s1)
   ```

2. Може да подадем всички елементи на списъка в квадратни скоби , като елементите могат да са от различен тип

```
s = ['P', 'Y', 'T', 'H', 'O', 'N', 3]
```

3. Може да формираме списък елемент по елемент като използваме метод **append( )**

```
s = []
s.append(5)
s.append(10)
s.append(20)
print(s)
```

4. Може да създаваме списък и като използваме генератор на последователности (list comprehension). Шаблонът на генератора изглежда така :

   Израз **for** *променлива* **in** *диапазон* **if** *условие*

   Първо се указва израз , определящ стойността на елемента . Този израз зависи от някаква променлива. След ключовата дума for се указва името на тази променлива, ключовата дума in и итерируема последователност. Променливата последователно приема стойности от тази последователност. За всяка стойност на променливата се изчислява елементът, който се включва в последователността , формирана от генератора

```
Пример: A=list( k for k in range(1,21) if k%3!=0)
        print(A)
```

Резултатния списък е [1,2,4,5,7,8,10,11,13,14,16,17,19,20]

Операции със списък

−*достъпване на елемент* : използват се **[]**като в тях се посочва индексът на елемента, който искаме да достъпим. Ако се опитате да достъпите несъществуващ елемент ще получите съобщение за грешка
IndexError: list index out of range . Индексът може да бъде и отрицателна стойност , тогава броенето е от края на списъка.

- *оператор за присвояване*: **=** използва се за присвояване на стойност на единичен елемент от списъка или за присвояване на стойност на целия списък

```
s = [2, 4, 6, 8]
```

```
s[0] = 7
print(s[0])
```

-*операция извличане на сечение* -връща посочен фрагмент от списъка

[< начало >:< край >:< стъпка >]

И трите параметъра са незадължителни. Ако липсва първия параметър се използва стойност по подразбиране 0(нула). Ако липсва втория – се връща целия фрагмент до края. Ако третия параметър не е зададен –за стъпка се приема за 1.

Пример:

```
s = [1, 2, 3, 4, 5, 6]
print(s[::-1])          #извеждаме елементите в обратен ред
print(s[:-1])           # принтираме без последния елемент
print(s[1:])            #принтираме без първия елемент
print(s[0:2])            #първите два елемента
print(s[-1:])         #последният елемент
```

-*Конкатенация на списъци* : използва се оператор **+**

```
s = [1, 2, 3, 4, 5, 6]
s1 = [9, 8]
s2 = s +s1
print(s2)
```

-*многомерни списъци*- всеки един елемент от списъка може да съдържа друг списък, кортеж или речник

```
s = [[1, 2, 3, 4], ['a', 'b', 'c'], [10, 20]]
print(s[0][3])  #достъпваме на първия списък 4-тия елемент
print(s[2][0])  #достъпваме на третия списък 1-вия елемент
```

-*обхождане на елементите на списък* – използваме цикъл **for**

```
s = [1, 2, 3, 4, 5, 6]
for i in s:
   print(i, end=' ')
```

За обхождане на списък може да използваме и функция ***range( )***

range(< начало>,<край >, < стъпка>)

```
s = [1, 2, 3, 4, 5, 6]
for i in range(len(s)):
    print(s[i], end=' ')
```

-търсене на елемент от списък- може да разберем дали един елемент е в списък с оператор **in** , но този метод работи само за едномерни списъци, за търсене на елемент в многомерни списъци трябва да се използва обхождане на елементите на списъка .

```
s = [2, 4, 6, 8]
print(4 in s)  #True
```

Метод **index( )** връща индекса на елемента в списъка , ако елемента не е в списъка ,методът връща грешка: *ValueError*
Метод **count( )** връща броят на елементите с определена стойност (подава се като параметър на функцията)
Функции **min( )** и **max( )** ни дават възможност да намерим елемента с най-малка и съответно с най- голяма стойност в списъка

```
s = [2, 4, 6, 8, 2]
print(s.count(2))   # 2
print(s.index(3))  #ValueError: 3 is not in list
print(min(s))   # 2
print(max(s))   # 8
```

-добавяне и премахване на елемент от списък : използват се методи (append, insert, pop, remove, del)
append( )- добавя елемент в края на списъка
За да добавим елемент в края на списъка можем да използваме и оператора **+=**
Метод **insert( )** поставя елемент в посочената позиция :
insert(<индекс > , < обект>)
pop( ) премахва елемент с посочения индекс и го връща, ако не е зададен индекс, изтрива последния елемент
С помощта на метод **remove( )** премахваме елемент с определена стойност
Оператор del( ) премахва елемента с дадения индекс и не връща нищо

```
s = [2, 4, 6, 8, 2]
s.append(22)       # добавя в края на списъка 22
print(s)
s += [44, 88]      # добавя в края 44,88
print(s)
```

```
s.insert(0, 90)   # вмъква в началото 90
print(s)
s.pop(0)          # изтрива първия елемент на списъка
print(s)
s.remove(2)       # изтрива елемента със стойност 2
print(s)

del s[4]          # премахва елемента с индекс 4
print(s)
```

-разбъркване на елементи и избиране на случаен елемент

Функция **shuffle( )** от модула **random** служи за разбъркване на списък

Можем да изберем случаен елемент от списък с помощта на функция **choice( )**

Метод **reverse( )** обръща подредеността на елементите на списъка, работи със самия списък , а не с негово копие

```
import random
s = [2, 4, 6, 8, 2]
random.shuffle(s)
print(s)          #[4, 8, 6, 2, 2]
print(random.choice(s))   # 8
s.reverse()
print(s)          #[2, 4, 6, 8, 2]
```

-сортиране на списък използва се метод **sort()** , методът променя списъка без да връща нищо

sort(< key=None> , <reverse=False>)- и двата параметъра не са задължителни. За сортиране в обратен ред трябва параметър **reverse =True.** Ако трябва да сортираме елементите по регистъра на символите ,тогава задаваме **key=str.lower**

Пример :

```
s = [45, 10, 55, 5, 35]
s.sort()
print(s)             #[5, 10, 35, 45, 55]
s.sort(reverse=True)
print(s)             #[55, 45, 35, 10, 5]

s1 = ['s', 'T', 'a', 'E', 'f']
s1.sort(key=str.lower)
print(s1)             # ['a', 'E', 'f', 's', 'T']
```

```
s1 = ['s', 'T', 'a', 'E', 'f']
s1.sort()                  # ['E', 'T', 'a', 'f', 's']
print(s1)                 # ['E', 'T', 'a', 'f', 's']
```

Кортежи – tupple-наредени n-торки

Кортежите са последователности , подобно на списъците ,но за разлика от списъците кортежите са неизменяеми. Кортежите могат да съдържат елементи от различен тип.

Как се създават кортежи?

```
k = (7, 5, 3)  #създаване на кортеж от 3 елемента
print(k)
```

Създаваме кортеж като изброим отделните елементи, разделени един от друг със запетая.

Може да използваме и функция **tuple( )** , която преобразува подадената като аргумент последователност в кортеж .

```
tup = tuple([10, 20, 30]) #преобразуване списък в кортеж
print(tup)

tup1 = tuple('python')  #преобразуване на низ в кортеж
print(tup1)                  ('p', 'y', 't', 'h', 'o', 'n')

tup2 = tuple()           #празен кортеж
print(tup2)
```

Кортежите поддържат операциите : обръщане към елемент по индекс, конкатенация, проверка за наличие, повторение, получаване на сечение.

```
k = (7, 5, 3, 6, 1)
print(k[0])       #достъп до елемент по индекс
print(k[2:3])     # сечение
print(7 in k)     # проверка за наличие на елемент
print(k * 3)      #повторение
tup = k + (2, 4)  #конкатенация
print(tup)
```

Кортежите поддържат следните методи:

index(<стойност >, < начало>, < край>)- връща индекса на елемента

count(< стойност>)- връща броя срещания на елемента с тази стойност

```
tup = (7, 5, 3, 6, 1)
tup.index(1)
print(tup.index(1))   # 4
```

```
tup.count(5)
print(tup.count(5))  # 1
```

Обхождане на елементите на кортеж

```
for i in tup:

    print(i, end=' ')
```

Към кортеж може да приложим функция **len( )** , която връща броя на елементите в кортежа.
В заключение можем да кажем :
Кортежите работят по-бързо от списъците, неизменяемостта на кортежите позволява да ги използваме като константи, потребяват по-малко памет .

Речник

Речникът е набор от обекти достъпът до които се реализира посредством ключ. Речниците могат да съдържат данни от различен тип и могат да се влагат . Елементите на речниците са в произволен порядък, затова се използва ключ за достъп до елементите , а не индекс, защото реда на добавяне е без значение.
Поради тази причина речниците не поддържат операциите: конкатенация,повторение, сечение.

Създаване на речник

Чрез използване на функция **dict( )**

dict(< ключ1> =<стойност1 >,..., <ключ n > =<стойност n >)

dict(< списък с кортежи от по два елемента- ключ и стойност >)

```
d = dict()
d1 = dict(name='Ivan', last_name='Petrov')
d3 = dict([ ('name', 'Polina'), ('l_name', 'Koleva')])
print(d1)
print(d3)
```

Речник може да създадем и като го запълваме елемент по елемент

```
d = { }
d['name'] = 'Petyr' d['l_name'] = 'Kolev'

print(d)
```

Може също във фигурните скоби да зададете всички елементи на речника

```
d = {'name': 'Ivan', 'last_name': 'Ivanov'}
print(d)
```

Важно ! При създаване на речник в променливата не се пази самия речник, а препратка към него, затова когато искаме да копираме речник трябва да използваме метода **copy( )** , а не оператор за присвояване.

Операции с речници

-достъп до елемент- осъществява се по ключ. При опит да се достъпи елемент с несъществуващ ключ се генерира изключение : KeyError: 'fname'

```
d = {'name': 'Ivan', 'last_name': 'Ivanov'}
print(d['fname'])  # KeyError: 'fname'
```

-можем да проверим дали даден ключ съществува с оператор **in**

```
b = 'fname' in d
print(b)  # False
```

-броят на ключовете в речника получаваме с функция **len( )**

```
d = {'name': 'Ivan', 'last_name': 'Ivanov'}
len(d)
print(len(d)) # 2
```

-добавяне на елементи в речника става по следния начин:

```
d['s_name'] = 'Petrov'
print(d)        # {'name': 'Ivan', 'last_name': 'Ivanov', 's_name': 'Petrov'}
```

Ако ключа не съществува в речника ще бъде добавен, но ако вече го има в речника тогава ще му бъде присвоена новата стойност.

-премахване на елемент от речник- използва се оператора **del**

```
del d['s_name']
print(d)          #   {'name': 'ivan', 'last_name': 'ivanov'}
```

-обхождане на елементите на речник – използва се цикъл **for**

```
 d = {"name": "Ivan", "last_name": "Ivanov"}

for key in d.keys():              # .keys() не е задължително
   print("({0} => {1})".format(key, d[key]), end=' ')
```

-методи **keys( )** и **values( )**

keys( ) -връща обекта dict_keys, който съдържа всички ключове на речника.
values( )-връща обекта dict_values, който съдържа всички стойности в речника.
dict_keys и dict_values поддържат итерация и операции с множества. -
сортиране на речник – тъй като речника е неподредена структура от данни при обхождане на речника неговите ключове се извеждат в произволен ред ,

но можем да сортираме речника по неговите ключове, като първо трябва да получим списък с всички ключове в речника и след това да използваме метода **sort( ).**

```
d = {"name": "Ivan", "last_name": "Ivanov"}
keys = list(d.keys())
keys.sort()
for key in keys:
    print("{0} => {1} ".format(key, d[key]), end=' ')
     #   last_name => Ivanov  name => Ivan
    print(f"{key} => {d[key]}", end=' ')
```

Множество

Множеството е неподредена съвкупност от уникални елементи. Създаваме множество с функция **set( )**. Тази функция може да преобразува в множество други типове данни- низове, кортежи, списъци, като в новополученото множество ще останат само уникални елементи.

```
s = set([1, 2, 3, 1])      #list
print(s)
s2 = set("hello")          #string
```

Операции с множества

-обхождане на множество- използва се цикъл

```
s2 = set("hello")
for i in s2:
   print(i, end=' ')     # l h e o
```

-функция **len( )** връща броя елементи в множеството

```
print(len(s2))      #4
```

-обединение на множества : оператор **|**, в новосъздаденото множество попадат само уникални елементи

```
s = set([1, 2, 3])
s1 = set([4, 2, 6])
s3 = s | s1
print(s3)           # {1, 2, 3, 4, 6}
```

-разлика на множества: оператор **– (минус) резултатът е остават само тези елементи от първото множество , които ги няма във второто множество**

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

```
s1 = set([2, 4, 6])
s2 = s - s1     # {1, 3}
s3 = s1 - s     # {6 }
```

-пресичане на множества – оператор **&** -резултатът са елементите , които се съдържат и в двете множества

```
s = set([1, 2, 3, 4])
s1 = set([2, 4, 6])
s4 = s & s1        # {2, 4}
```

-оператор **^** -връща елементите на двете множества, изключвайки еднаквите

```
s = set([1, 2, 3, 4])
s1 = set([2, 4, 6])
s5 = s ^ s1       # {1, 3, 6}
```

-други оператори

In – проверява дали даден елемент е от множеството

```
3 in s  # True
```

Оператор == проверява дали две множества са равни

Оператор **s1 <= s** проверява влизат ли всички елементи от множеството **s1** в множеството **s**

Методи на множествата

add(< елемент>) – добавя елемент

remove(<елемент >) – премахва елемент, ако елемента го няма в множеството **ще върне грешка**

discard(< елемент>) – премахва посочения елемент, ако елемента го няма **не връща грешка**

pop( ) - премахва произволен елемент и го връща

clear( ) – изчиства множеството

```
s1 = set([2, 4, 6])
s1.add(8)       # {8, 2, 4, 6}
s1.remove(2)  # {8, 4, 6}
print(s1)
s1.remove(2)  # KeyError: 2
```

Низове

Низът е подредена последователност от символи. Низовете се ограждат в двойни кавички или апострофи

```
print("Hello")
print('Hello')
```

```
print('Hello, " World " ')          # тук външните апострофи ограждат низа, а вътрешните кавички се извеждат като обикновени символи
```

На функция print() могат да бъдат предадени няколко стойности, разделенисъс запетая

```
print("hello","world")
```

Низовете се явяват неизменяеми типове данни ,затова почти всички методи за работа с низове връщат като стойност нов низ, а не изменят съществуващия. Можете да получите даден символ от низа по неговия индекс, но не можете да го измените.

Python поддържа следните типове низове : str, bytes , bytearr. Str е обикновен unicode низ. Типът bytes представлява неизменяема последователност от байтове, всеки елемент от такава последователност може да пази число от 0 до 255, което обозначава кода на символа. Типът bytearray е изменяема последователност от байтове, при него може да изменяте елементите от низа по индекс.

Ако е необходимо да присвоите на дадена променлива многоредов текст използвайте тройни кавички или тройни апострофи

```
S=""" hello ,
      *****
      World!!! """
```

Действия с низове- обръщане към елемент по индекс, сечение, конкатенация, проверка за наличие , повторение

```
S="123"
print(S[0]) # 1
s1="Hello"
s1[:]  #фрагмент от позиция 0 до края на низа
s1[:-1:] #изрязваме последния символ на низа 'Hell'
s1[1::2] #започва от втория символ до края,стъпка 2 'el'
s1[1::] #изрязваме първия символ 'ello'
```

конкатенация на низове +

```
print("Hello"+"world")
```

проверка за наличие- можем да провери дали подниз е наличен в даден низ с оператор **in ,** връща True ако е наличен и False ако не е наличен

```
"hell" in "Hello"  # False
"hell" in "hello"  # True
```

Повторение * повтаря даден низ определен брой пъти

```
print("f" * 5)  # fffff
```

Функции за работа с низове

len() -връща дължината (броя символи) на низа

s = "Здравей"
print(len(s)) # 6

str([обект])- преобразува един обект в низ, ако не е посочен параметър връща празен низ, използва се от функция **`print`** и от някои други функции за извеждане на обекти

print(str(123)) # "123"

repr(<обект>) – връща низова репрезентация на обекта. Използва се в IDLE за извеждане на обекти

s = "Hello World"
print(repr(s)) # 'Hello World'

Методи за работа с низове

capitalize() – прави главна първата буква в низа

join(< последователност>) – преобразува последователност в низ, елементите се разделят от указания разделител . Методът се използва по следния начин :
<низ >=<разделител>.join(<последователност>)

lower() – привежда всички символи на низа в долен регистър

split([<разделител>[,< лимит>]]) – разделя низа на поднизове на базата на посочения разделител, ако такъв не е зададен за разделител се използва интервал

strip([символи]) – изтрива посочените символи в началото и в края на низа, ако няма посочени тогава се премахват символите за празно пространство .

swapcase() -обръщане на регистъра, горния регистър се заменя с долен и обратно

title() – прави главна първата буква на всяка дума

upper()- привежда всички символи в горен регистър

find() -търсене на подниз в низ ,връща номера на позицията от която започва подниза в низа, или връща -1 ако подниза не е намерен

<низ>.find(<подниз>[,<начало>[,<край>]])

Index()- подобен на метод find() , но връща изключение ValueError ако поднизът не може да бъде намерен.

count() -методът връща броя на съвпаденията на подниза в низа, връща 0 ако

няма съвпадение, синтаксисът му е следния:

<низ>.count(<подниз>[,<начало>[,<край>]])

replace()- методът осъществява замяна на всички съвпадения на подниз в низа с указан подниз

<низ> . replace(<заменяем подниз> , <нов подниз> [, <лимит>])

Параметър лимит не е задължителен, той ограничава броя на замените.

Задачи:

1. Напишете програма, в която потребителя въвежда цяло число, а програмата формира два кортежа, състоящи се от цифрите , влизащи в това число. Единият с цифрите на числото в прав ред и втори , в който цифрите са в обратен ред.

2. Напишете програма, в която се създава числов списък. Той се запълва със случайни числа. След това между всяка двойка елементи от този списък се вмъква нов , равен на сумата от стойностите на съседните елементи.

3. Напишете програма , в която потребителя въвежда текст и на негова база се създава речник. За ключове на речника служат символите от текста, а стойността на елементите се определя от броя на съответните символи в текста.
   Примерен вход : SSWTWWTAAA
   Речникът ще се състои от 4 елемента :
   A:3 S:2 T:2 W:3

4. Напишете програма, в която потребителя въвежда чяло число. От него се създава списък състоящ се от числата от 1 до това число . Въз основа на този списък се създава речник, в който елементите на списъка служат за ключове на елементите на речника , а стойностите на тези елементи в речника са елементите на списъка но в обратен ред.
   Пример : ако сме въвели числото 4 , създава се списъка [1,2,3,4] и на негова основа се създава речник с 4 елемента : {1:4, 2:3, 3:2, 4:1}