

ОСНОВИ ПРОГРАМУВАННЯ МОВОЮ Python

Урок 6

Кортежі, Множини, словники

Зміст

1. Кортежі	4
1.1. Колекції незмінних об'єктів	4
1.2. Застосування та особливості кортежу	5
2. Множини	19
2.1. Математичне поняття множин	19
2.2. Тип даних set()	22
2.3. Операції над множинами	25
2.4. frozenset()	31
2.5. Застосування множин	33
3. Словники	36
3.1. Асоціативні масиви	36
3.2. Хеш-таблиці	37
3.3. Створення словника	40
3.4. Методи словника	44

4. Практичні приклади використання	60
4.1. Приклад 1. Пошук у словнику	60
4.2. Приклад 2. Сортювання елементів словника	63
4.3. Приклад 3. Сортювання списку словників	67
4.4. Приклад 4. Фільтри в словнику	69

1. Кортежі

1.1. Колекції незмінних об'єктів

Кортеж (*tuple*) — це ще один тип Python-колекцій, який є «незмінним списком», тобто є впорядкованим і незмінним набором довільних даних.

Коли є сенс використовувати кортежі і чому нам для цих завдань не підходять списки? Кортежі використовуються для зберігання даних, список значень яких не має змінюватися в програмі, тобто з метою захистити дані від зміни (явної чи випадкової) у коді програми.

Наприклад, наш код має використовувати дані про типи користувача додатка. Якщо ця інформація зберігатиметься у списку, наприклад, так:

```
userTypes = ["admin", "student", "teacher"],
```

тоді рядок коду, подібний до цього:

```
userTypes[0]="user"
```

може видалити не тільки інформацію про тип користувача, а й, можливо, порушити логіку роботи тих функцій програми, які пов'язані з рівнем доступу «**admin**».

Крім цього, кортежі в Python займають менший обсяг пам'яті при зберіганні того ж набору значень. І саме використання кортежів у програмі, замість списків, також підвищить продуктивність додатка. До кортежу можна застосовувати багато функцій списків, крім функцій зміни даних.

1.2. Застосування та особливості кортежу

Почнемо із створення кортежів. Можна створити порожній кортеж (без елементів) або кортеж зі значеннями. Для створення порожнього кортежу можна скористатися одним із способів:

- використовуючи оператор `()`: `myEmptyTuple = ()`;
- або використовуючи функцію-конструктор `tuple()`: `myEmptyTuple = tuple()`;

```
myEmptyTuple1=()  
myEmptyTuple2= tuple()
```

Виведемо вміст і тип змінних `myEmptyTuple1` та `myEmptyTuple2`:

```
print(myEmptyTuple1)  
print(type(myEmptyTuple1))  
  
print(myEmptyTuple1)  
print(type(myEmptyTuple2))
```

Результат:

```
()  
<class 'tuple'>  
()  
<class 'tuple'>
```

Рисунок 1

Для того, щоб створити кортеж з потрібним вмістом, можна прописати його майбутні елементи через кому, як при створенні списку, але замість квадратних дужок слід використовувати круглі. Також можна створити кортеж

з одного або декількох елементів, просто прописавши їх після оператора привласнення через кому.

Примітка: у випадку з одним елементом, кома також обов'язкова після його значення.

```
myTuple1=('element1', 12, 35.6, False)
myTuple2=('item1')
userTypes='admin','student','teacher'
userName='Jane',
```

```
print("myTuple1:", myTuple1,"type: ", type(myTuple1))
print("myTuple2:", myTuple2,"type: ", type(myTuple2))
print("userTypes:", userTypes,"type: ", type(userTypes))
print("userName:", userName,"type: ", type(userName))
```

Результат:

```
myTuple1: ('element1', 12, 35.6, False) type: <class 'tuple'>
myTuple2: item1 type: <class 'str'>
userTypes: ('admin', 'student', 'teacher') type: <class 'tuple'>
userName: ('Jane',) type: <class 'tuple'>
```

Рисунок 2

Як бачимо, кортежі можуть містити дані будь-яких типів. Також можливе дублювання значень всередині кортежу:

```
itemTuple=('item1','item2','item1','item3')
print(itemTuple) #('item1', 'item2', 'item1', 'item3')
```

Якщо ми створюватимемо кортеж з кількома елементами, використовуючи функцію-конструктор `tuple()`, тому потрібно використовувати подвійні круглі дужки:

```
namesTuple = tuple(('Alex', 'Helen'))
print(namesTuple) #('Alex', 'Helen')
```

Елементи кортежу індексуються так само, як елементи списку: перший елемент має індекс [0], другий елемент має індекс [1] і т. д.

Від'ємне індексування також можливе.

```
userTypes=('admin','student','teacher','moderator')

print("1st user:", userTypes[0])
print("last user:", userTypes[len(userTypes)-1])
print("last user once again:", userTypes[-1])
```

Результат:

```
1st user: admin
last user: moderator
last user once again: moderator
```

Рисунок 3

Зрізи, з особливостями яких ми вже познайомилися працюючи зі списками, також можна застосовувати і для кортежів:

```
print("1st two users:", userTypes[:2])
print("2nd and 3rd users:", userTypes[1:3])
```

Результат:

```
1st two users: ('admin', 'student')
2nd and 3rd users: ('student', 'teacher')
```

Рисунок 4

Після створення кортежу ми не можемо змінити його значення, додати елементи в кортеж або видалити з нього.

Подібна спроба:

```
userTypes=('admin','student','teacher','moderator')
userTypes[0]='user'
```

Викличе таку помилку:

```
'tuple' object does not support item assignment
```

Рисунок 5

Видалення окремих елементів із кортежу неможливе. Однак, ми можемо за необхідності видалити весь кортеж. Для цього просто скористайтесь оператором [del](#):

```
del tupleName
```

В Python нам також дозволено витягувати значення кортежів назад у змінні. Такий підхід називається «розпакування»:

```
userTypes=('admin','student','teacher','moderator')
user1, user2, user3, user4 =userTypes

print(user1)    #admin
print(user2)    #student
print(user3)    #teacher
print(user4)    #moderator
```

З використанням цього підходу кількість змінних перед знаком присвоєння має збігатися з кількістю значень у кортежі.

Якщо кількість змінних, які знаходяться лівіше від знака присвоєння, менше кількості значень, тоді можна додати символ `*` до імені змінної (останньої), і тоді значення, що залишилися, будуть зібрані у вигляді списку і присвоєні цій змінній:

```
userTypes=('admin','student','teacher','moderator')
user1, *users =userTypes

print(user1) #admin
print(users) #['student', 'teacher', 'moderator']
```

Якщо символ `*` доданий до імені не останньої змінної, то інтерпретатор Python буде присвоювати змінній значення кортежу доти, доки кількість значень, що залишилися, не співпадає з кількістю змінних, що залишилися після неї.

```
userTypes=('admin','student','teacher','moderator')
irstUser, *users, lastUser =userTypes

print(firstUser) #admin
print(users) #['student', 'teacher']
print(lastUser) #moderator
```

Ми можемо обробляти елементи кортежу в циклі, як ми це робили зі списками:

```
userTypes=('admin','student','teacher','moderator')

for user in userTypes:
    print(user)
```

Результат:

```
admin  
student  
teacher  
moderator
```

Рисунок 6

Ми також можемо перебирати елементи кортежу, звертаючись до їх порядкового номера (індексу), використовуючи функції `range()` та `len()` для організації циклу.

```
userTypes=('admin','student','teacher','moderator')  
for i in range(len(userTypes)):  
    print(userTypes[i])
```

Результат буде аналогічним до попереднього.

Для того, щоб з'єднати два або більше кортежів, можна використовувати оператор `+`:

```
userTypes1=('admin','student','teacher','moderator')  
userTypes2=('user1','user2')  
allUsers=userTypes1+userTypes2  
for user in allUsers:  
    print(user)
```

Результат:

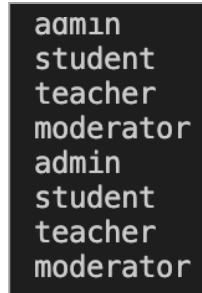
```
admin  
student  
teacher  
moderator  
user1  
user2
```

Рисунок 7

Якщо ви хочете продублювати вміст кортежу потрібну кількість разів, використовуйте оператор `*`:

```
userTypes=('admin','student','teacher','moderator')
for user in userTypes*2:
    print(user)
```

Результат:



```
admin
student
teacher
moderator
admin
student
teacher
moderator
```

Рисунок 8

В Python є два вбудовані методи, які ви можете використовувати для роботи з кортежами.

Метод `count()` повертає кількість разів, коли вказане значення-аргумент з'являється у кортежі:

```
userTypes=('admin','student','teacher','moderator',
           'admin')
print(userTypes.count('admin')) #2
```

Метод `index()` знаходить перше входження вказаного значення-аргументу і повертає його позицію:

```
userTypes=('admin','student','teacher','moderator',
           'admin')
print(userTypes.index('admin')) #0
```

Ми можемо перевірити, чи існує елемент у кортежі чи ні, використовуючи оператор `in`:

```
userTypes=('admin','student','teacher','moderator',
          'admin')

if 'student' in userTypes:
    print('student is correct login' )
```

Розглянемо приклад використання кортежів. Припустимо, що нам необхідно зберігати та обробляти інформацію про користувачів. Причому особиста інформація (прізвище, ім'я, рік народження, стать) вводиться один раз при реєстрації і далі не має змінюватися і доповнятися (тобто нові відомості про користувача, наприклад, адресу в особисту інформацію додавати не можна).

Також є й інформація про інтереси (перелік «хобі»), які можуть змінюватись, перелік мов програмування, які знає користувач і список яких також може змінюватися.

Таким чином зручно оформити інформацію користувача у вигляді списку, елементами якого будуть: кортеж (для зберігання незмінної особистої інформації), список інтересів і список мов програмування.

Спочатку створимо функцію, яка запитує користувача про особисту інформацію і повертає її до кортежу:

```
def askPersonalInfo():
    while True:
        firstName=input("Input your first name:")
        lastName=input("Input your last name:")
        yearBirth=input("Input your year of birth:")
        gender=input("Input your gender (M,F):")
```

```

if firstName==" " or lastName==" " or
   yearBirth==" " or gender==" " or
   gender not in ('F','M'):
    print("Wrong data!")
else:
    return firstName, lastName, yearBirth,
           gender

```

```

personalInfo=askPersonalInfo()
print(personalInfo)

```

Ми також використовували кортеж ('F','M'), щоб перевірити, чи користувач ввів допустиме значення для інформації «стать».

Функція повертає чотири значення, які основна програма запише у кортеж `personalInfo`.

Перевіримо, як наша функція обробляє хибні ситуації:

```

Your first name:anna
Your last name:smith
Your year of birth:1990
Your gender (M,F):a
Wrong data!
Your first name:
Your last name:a
Your year of birth:1990
Your gender (M,F):F
Wrong data!
Your first name:anna
Your last name:smith
Your year of birth:1990
Your gender (M,F):F
('anna', 'smith', '1990', 'F')

```

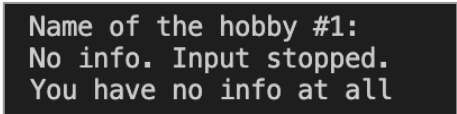
Рисунок 9

Йдемо далі: створимо функцію для введення інформації про інтереси користувача і мови програмування, які він знає.

Процес запиту хобі триватиме доти, доки користувач не введе назву замість порожнього рядка (ознака закінчення введення). Організуємо цю логіку за допомогою нескінченного циклу і його зупинку оператором `break` за вказаною умовою.

```
def askHobby():
    hobbyInd=1
    hobbyList=[]
    while True:
        hobbyName=input("Name of the hobby #{}:".
                        format(hobbyInd))
        if hobbyName=="":
            print("No info. Input stopped.")
            break
        else:
            hobbyList.append(hobbyName)
            hobbyInd+=1
    if len(hobbyList)>0:
        print("You have {} hobbies.".
              format(hobbyInd-1))
    else:
        print("You have no hobbies at all")
    return hobbyList
```

Проведемо її тестування.



```
Name of the hobby #1:
No info. Input stopped.
You have no info at all
```

Рисунок 10

```
Name of the hobby #1:cooking
Name of the hobby #2:drawing
Name of the hobby #3:
No info. Input stopped.
You have 2 hobbies.
```

Рисунок 11

Тепер зробимо логіку більш універсальною, щоб її можна було використовувати і для формування списку мов програмування, які знає користувач: для цього частина рядка запиту («*hobby*» «*programming languages*») передаватимемо як параметр нашої функції. Також зробимо імена її локальних змінних більш універсальними.

```
def askAdditionalInfo(queryStr):
    infoInd=1
    infoList=[]

    while True:
        infoName=input("Name of the {} #{}:".
                        format(queryStr,infoInd))
        if infoName=="":
            print("No info. Input stopped.")
            break
        else:
            infoList.append(infoName)
            infoInd+=1

    if len(infoList)>0:
        if queryStr=='hobby':
            print("You have {} hobbies.".
                  format(infoInd-1))
        elif queryStr=='programming languages':
            print("You know {} programming languages.".
                  format(infoInd-1))
```

```

else:
    print("You have no info at all")

return infoList

```

Весь код нашої програми:

```

def askPersonalInfo():
    while True:
        firstName=input("Your first name:")
        lastName=input("Your last name:")
        yearBirth=input("Your year of birth:")
        gender=input("Your gender (M,F):")
        if firstName=="" or lastName=="" or
            yearBirth=="" or gender=="" or
            gender not in ('F','M'):
            print("Wrong data!")
        else:
            return firstName, lastName,
                yearBirth, gender
def askAdditionalInfo(queryStr):
    infoInd=1
    infoList=[]
    while True:
        infoName=input("Name of the {} #{}:".
            format(queryStr,infoInd))
        if infoName=="":
            print("No info. Input stopped.")
            break
        else:
            infoList.append(infoName)
            infoInd+=1
    if len(infoList)>0:
        if queryStr=='hobby':
            print("You have {} hobbies.".
                format(infoInd-1))

```



```

        elif queryStr=='programming languages':
            print("You know {} programming languages.".
                  format(infoInd-1))

        else:
            print("You have no info at all")
            return infoList

userInfo=[]
userInfo.append(askPersonalInfo())
userInfo.append(askAdditionalInfo('hobby'))
userInfo.append(askAdditionalInfo('programming languages'))
print(userInfo)

```

Результат:

```

Input your first name:Anna
Input your last name:Smith
Input your year of birth:2000
Input your gender (M,F):F
Name of the hobby #1:cooking
Name of the hobby #2:drawing
Name of the hobby #3:
No info. Input stopped.
You have 2 hobbies.
Name of the programming languages #1:JS
Name of the programming languages #2:Python
Name of the programming languages #3:C++
Name of the programming languages #4:
No info. Input stopped.
You know 3 programming languages.
[('Anna', 'Smith', '2000', 'F'), ['cooking', 'drawing'],
 ['JS', 'Python', 'C++']]

```

Рисунок 12

Оскільки кортежі дуже схожі на списки, вони обидва використовуються у схожих ситуаціях. Проте кортеж має певні переваги перед списком:

- зазвичай ми використовуємо кортежі для різнорідних (різних) типів даних та списки для однорідних (схожих) типів даних;

- оскільки кортежі незмінні, ітерація по кортежу виконується швидше, ніж у списку (отже, є невеликий приріст продуктивності під час використання кортежів);
- кортежі, що містять незмінні елементи, можна використовувати як ключ для словника (з якими ми познайомимося трохи пізніше), але зі списками таке неможливо;
- якщо у вас є дані, які не змінюються (не мають змінюватися, наприклад, перелік типів користувачів, з якими працює додаток), їх реалізація у вигляді кортежу гарантує, що вони залишаться захищеними від запису.

2. Множини

2.1. Математичне поняття множин

У повсякденному житті нам постійно доводиться мати справу з різними наборами (сукупностями) об'єктів, тому поняття числа (кількості) і поняття множини є дуже поширеним. Множина є одним із фундаментальних понять не тільки математики, а й у будь-якій предметній галузі.

Множина — це довільна сукупність (набір, колекція) будь-яких об'єктів. Такі об'єкти, що входять до складу множини, називаються елементами множини. Найчастіше розуміються як одне ціле, тому що мають загальні характеристики, ознаки, належать до однієї обставини або критерію, підпорядковуються одному правилу тощо.

Поняття множини абстрактне. Якщо ми розглядаємо певний набір як множину, то ми не робимо акцент на зв'язках (стосунках) між ними, а зберігаємо їх індивідуальні характеристики. Наприклад, множина з трьох чашок і множина із трьох символів — це абсолютно різні множини. При цьому, якщо ми виставимо чашки в ряд або складемо одна на одну, то це буде та сама множина елементів (чашок).

Якщо певний об'єкт x є елементом множини A , то x належить A (пишеться $x \in A$). Зазвичай множини позначаються великими латинськими літерами A, B, C (за необхідності з підрядковими індексами: A_1, A_2 і т. д.), а елементи множини записуються у фігурних дужках, наприклад:

- $A = \{a, b, c, \dots, z\}$ — множина символів латинського алфавіту;

- $S1 = \{\text{Bob, Joe, ..Kate}\}$ — множина студентів першої групи.

Множини у наших прикладах є кінцевими, тобто складаються із закінченого числа елементів. Також існує порожня множина — множина, що не містить жодного елемента.

Множина B є підмножиною множини A , якщо кожен елемент множини B належить множині A . Інакше кажучи, множина B міститься у множині A . Наприклад, є множина студентів першої групи $S1 = \{\text{Bob, Joe, ...Kate}\}$, тоді множина студентів цієї групи, які мають середній бал вищий за 75 — $S_{\text{High}} = \{\text{Bob, Kate}\}$ — це підмножина множини $S1$.

Або є множина студентів університету $S_u = \{st_1, \dots st_n\}$. Тоді множина S_1 є підмножиною множини S_u .

2.1.1. Операції над множинами

Якщо ми маємо декілька множин, то ми можемо виконувати над ними різні операції. Схематично операції над множинами зображують у вигляді кіл Ейлера.

Кола Ейлера — це загальноприйнята геометрична схема для візуалізації логічних зв'язків між об'єктами (явищами, поняттями) предметної галузі. Кожній множині відповідає коло, за допомогою якого ми показуємо, що усі точки всередині кола належать цій множині, а решта — ні.

Почнемо з операції перетину множин (*intersection*). Перетином множин A та B називається така множина, кожен елемент якої належить обом множинам (і безлічі A , і безлічі B), тобто це загальні елементи множин A та B .

Операція перетину множин — це логічне I (елемент належить одночасно кільком множинам, одночасне виконання кількох умов — це логічне I) (рис. 13):

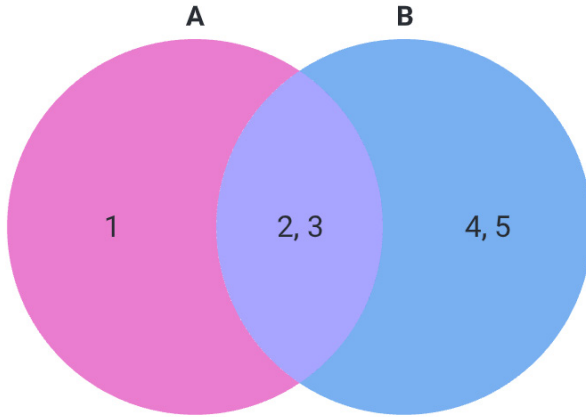


Рисунок 13

Позначається, як $A \cap B$. Наприклад, $A = \{1, 2, 3\}$, а $B = \{2, 3, 4, 5\}$. Тоді $A \cap B = \{2, 3\}$. Якщо у множин немає однакових елементів, їх перетин порожній. Об'єднанням (*union*) множин A та B називається множина, кожен елемент якої належить або множині A , або множині B . Операція об'єднання множин — це логічне АБО (рис. 14).

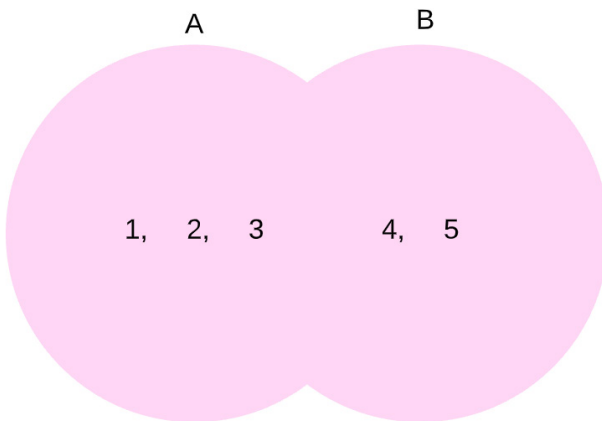


Рисунок 14

Розглянемо з прикладу: $A = \{1, 2, 3\}$, а $B = \{3, 4, 5\}$

$A \cup B = \{1, 2, 3, 4, 5\}$, тобто це усі елементи множини A та B без повторень (тобто елемент 3 у перетині з'являтиметься лише один раз).

Різницею (*difference*) множин A та B називають таку множину, кожен елемент якої належить множині A і не належить множині B :

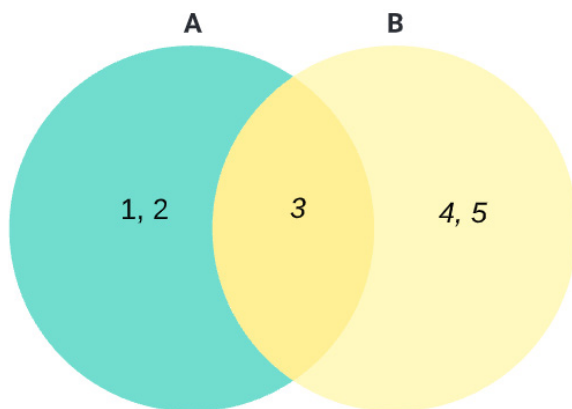


Рисунок 15

Розглянемо з прикладу: $A = \{1, 2, 3\}$, а $B = \{3, 4, 5\}$. $A \setminus B = \{1, 2\}$, тобто це усі елементи множини A , яких немає в множині B .

2.2. Тип даних `set()`

Множина (*set*) в Python — це колекція не повторюваних елементів, яка є невпорядкованою (неіндексованою).

На відміну від кортежів, в множині (після її визначення) можна додавати (видаляти) елементи. Також доступні операції перебору елементів множини та математичні операції над множинами: об'єднання, перетину, різниці.

І, звичайно, ми можемо перевірити, чи належить певний елемент деякій множині.

Елементи, що входять до складу множини, можуть бути будь-якого незмінного типу даних (числові, рядки, кортежі, множини). Однак елементи змінюваних типів даних (наприклад, списки) не можуть увійти до множини.

За аналогією з математикою, в Python множини записуються всередині фігурних дужок.

Множина створюється шляхом поміщення усіх елементів у фігурні дужки {}, розділяючи їх комою:

```
mySet1 = {1, 2, 3}
mySet2= {'Joe', 'Bob', 'Kate'}
mySet3= {23, 'Bob', False, (45.6, 12)}

print("mySet1 - set of numbers:", mySet1)
print("mySet2 - set of strings:", mySet2)
print("mySet3 - of mixed datatypes:", mySet3)
```

Результат:

```
mySet1 - set of numbers: {1, 2, 3}
mySet2 - set of strings: {'Kate', 'Joe', 'Bob'}
mySet3 - of mixed datatypes: {False, (45.6, 12), 'Bob', 23}
```

Рисунок 16

або за допомогою вбудованої функції-конструктора `set()`, але тоді потрібно використовувати подвійні круглі дужки:

```
mySet4=set((10,20,30))
print("mySet4:", mySet4) #{10, 20, 30}
```

При створенні множини, повторювані елементи будуть автоматично видалятися:

```
uniqueUserName= {'Joe', 'Bob', 'Kate', 'Bob'}
print(uniqueUserName) #{'Kate', 'Joe', 'Bob'}
```

Тому множини зручно використовувати для видалення дублікатів (наприклад, зі списку):

```
allUserName= ['Joe', 'Bob', 'Kate', 'Bob']
uniqueUserName=set(allUserName)
print(uniqueUserName) #{'Kate', 'Joe', 'Bob'}
```

При спробі створення множини з елементом змінного типу, отримуємо помилку:

```
passwordsSet={'111', ['222', '333']}
```

unhashable type: 'list'

Рисунок 17

Створити порожню множину за допомогою порожніх фігурних дужок. Порожні фігурні дужки {} створюють порожній словник Python. Для того, щоб створити множину без будь-яких елементів, потрібно скористатися функцією set() без аргументів:

```
myEmptySet=set()
print(myEmptySet) #set()
```

Як ми вже зазначали вище, елементи набору не впорядковані.

Невпорядкований набір означає, що елементи у ньому не мають певного порядку. Перевіримо цю особливість з прикладу:

```
mySetA = {1, 2, 3}
mySetB = {3, 2, 1}
print(mySetA==mySetA) #True
```

Результат **True** говорить про те, що **mySetA** та **mySetB** — це та сама множина. Так і є: набір елементів той самий, а їх порядок у наборі значення не має.

2.3. Операції над множинами

Оскільки множини невідсортовані, індексація не має сенсу. Таким чином, ми не можемо отримати доступ до елемента множини або змінити його за допомогою індексації або зрізів.

Проте, ми можемо пройтися елементами множини, використовуючи цикл **for**:

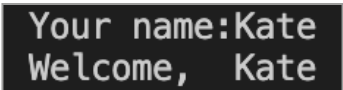
```
userNames= {'Joe', 'Bob', 'Kate'}
for name in userNames:
    print(name)
```

або перевірити, чи є вказане значення у множині, використовуючи ключове слово **in**:

```
userNames= {'Joe', 'Bob', 'Kate'}
name=input("Your name:")

if name in userNames:
    print("Welcome, ", name)
```

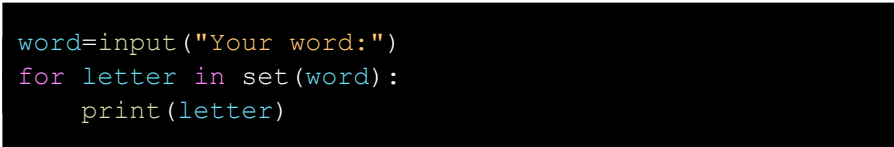
Результат:



```
Your name:Kate
Welcome, Kate
```

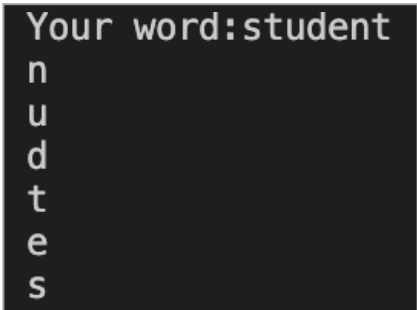
Рисунок 18

А тепер виведемо унікальні символи введеного користувачем слова, використовуючи множини:



```
word=input("Your word:")
for letter in set(word):
    print(letter)
```

Результат:



```
Your word:student
n
u
d
t
e
s
```

Рисунок 19

Після створення множини ви не можете змінювати її елементи, але можете додавати нові елементи або видаляти їх з множини.

Ми можемо додати один елемент у безліч за допомогою методу `add()` і декілька елементів за допомогою методу `update()`. Метод `update()` може приймати як аргумент кортежі, списки, рядки або інші множини. В усіх випадках дублікати видаляються.

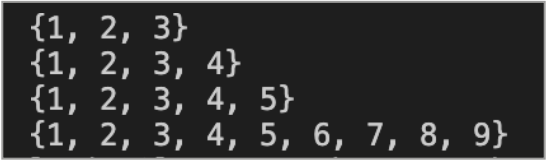
```
mySet = {1, 2, 3}
print(mySet)

mySet.add(4)
print(mySet)

mySet.update([3, 4, 5])
print(mySet)

mySet.update([5, 6, 7], {8, 9})
print(mySet)
```

Результат:



```
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Рисунок 20

Ми також можемо видалити конкретний елемент з множини за допомогою методів `discard()` та `remove()`. Єдина різниця між ними полягає в тому, що метод `discard()` залишає набір без змін, якщо елемент, що видаляється, відсутній. А от метод `remove()` у такому разі викличе помилку.

```
mySet = {1, 2, 3, 4, 5}
print(mySet)

mySet.discard(4)
print(mySet)

mySet.remove(5)
print(mySet)
```

```
mySet.discard(10)
print(mySet)
```

Результат:

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

Рисунок 21

Як бачимо, спроба видалити елемент 10 (якого немає у множині) за допомогою методу `discard()` не змінила нашу множину.

А от якщо ми спробуємо повторити це використовуючи метод `remove()` — отримаємо помилку:

```
mySet.remove(10)
```

KeyError ✕

Рисунок 22

Ми також можемо видалити усі елементи з множини, використовуючи метод `clear()`.

Python надає нам набір вбудованих методів, які виконують розглянуті нами операції перетину, об'єднання та різниці множин. Ми можемо виконувати зазначені операції над множинами за допомогою спеціальних методів або операторів `&`, `|`, `-`. Розглянемо їх докладніше. Перетин виконується за допомогою оператора `&`. Те саме можна зробити за допомогою методу `intersection()`.

Об'єднання множин доступне нам через оператор `|` або метод `union()`.

Різниця множин виконується оператором `-`. Те саме можна зробити методом `different()`.

```
studGroup1={'Hanna','Joe','Kate'}
studGroup2={'Bob','Joe','Jane','Kate','Jack'}
print("studGroup1:",studGroup1)
print("studGroup2:",studGroup2)

print("Intersection of sets:")
print(studGroup1&studGroup2) #
print(studGroup1.intersection(studGroup2))

print("Union of sets:")
print(studGroup1|studGroup2) #
print(studGroup1.union(studGroup2))

print("Difference of two sets:")
print(studGroup2-studGroup1) #
print(studGroup2.difference(studGroup1))
```

Результат:

```
studGroup1: {'Kate', 'Hanna', 'Joe'}
studGroup2: {'Bob', 'Joe', 'Kate', 'Jane', 'Jack'}
Intersection of sets:
{'Kate', 'Joe'}
{'Kate', 'Joe'}
Union of sets:
{'Bob', 'Joe', 'Jack', 'Kate', 'Jane', 'Hanna'}
{'Bob', 'Joe', 'Jack', 'Kate', 'Jane', 'Hanna'}
Difference of two sets:
{'Jack', 'Bob', 'Jane'}
{'Jack', 'Bob', 'Jane'}
```

Рисунок 23

Також під час роботи з множинами доступні вже відомі нам вбудовані функції, такі, як: `enumerate()`, `len()`, `max()`, `min()`, `sorted()`, `sum()` і т. д., які корисні під час виконання різних завдань.

Примітка: функція `enumerate()` повертає перелічений об'єкт, який містить індекс і значення для усіх елементів набору у вигляді пари (кортежу).

Розглянемо їх застосування з множинами у прикладах:

```
studSet={'Bob', 'Joe', 'Jane', 'Kate', 'Jack'}

print("We have {} students in our group.".
      format(len(studSet)))
for ind,item in enumerate(studSet):
    print(ind,item)
```

Результат:

```
We have 5 students in our group.
0 Jack
1 Joe
2 Jane
3 Bob
4 Kate
```

Рисунок 24

```
scores={1,2,3,4,5,6,7,8,9,10,11,12}

print("Min score is", min(scores))
print("Max score is", max(scores))
print("Sum of scores:", sum(scores))
```

Результат:

```
Min score is 1
Max score is 12
Sum of scores: 78
```

Рисунок 25

2.4. frozenset()

Frozenset — це додатковий клас у Python, який має усі характеристики множини (*set*), але його вміст змінювати не можна. Також, як ми називаємо кортежі незмінюваними списками, ми можемо називати **frozenset** незмінюваними множинами.

Цей тип даних підтримує розглянуті підходи для об'єднання, перетину і знаходження різниці для множин.

```
frozenA = frozenset(['Hanna', 'Joe', 'Kate'])
frozenB =
frozenset(['Bob', 'Joe', 'Jane', 'Kate', 'Jack'])

print("frozenA:", frozenA)
print("frozenB:", frozenB)

print("Intersection of frozensets:")
print(frozenA & frozenB)
print(frozenA.intersection(frozenB))

print("Union of frozensets:")
print(frozenA | frozenB)
print(frozenA.union(frozenB))

print("Difference of two frozensets:")
print(frozenB - frozenA)
print(frozenB.difference(frozenA))
```

Результат:

```
frozenA: frozenset({'Joe', 'Kate', 'Hanna'})
frozenB: frozenset({'Jack', 'Bob', 'Kate', 'Joe', 'Jane'})
Intersection of frozensets:
frozenset({'Kate', 'Joe'})
frozenset({'Kate', 'Joe'})
Union of frozensets:
frozenset({'Jack', 'Hanna', 'Bob', 'Kate', 'Joe', 'Jane'})
frozenset({'Jack', 'Hanna', 'Bob', 'Kate', 'Joe', 'Jane'})
Difference of two frozensets:
frozenset({'Bob', 'Jane', 'Jack'})
frozenset({'Bob', 'Jane', 'Jack'})
```

Рисунок 26

Однак у зв'язку з незмінністю **frozenset** не підтримує методи для додавання елементів до множини та видалення з нього.

```
frozenA.add('user')
```

'frozenset' object has no attribute 'add'

Рисунок 27

```
frozenB.remove('Bob')
```

'frozenset' object has no attribute 'remove'

Рисунок 28

Через можливість зміни свого вмісту, звичайні множини не можна використовувати як ключі словників, з якими ми познайомимося трохи пізніше. А от **frozenset** підходять для цього.

2.5. Застосування множин

Як згадувалося вище, використання множин дозволяє легко вирішити проблему видалення дублікатів (наприклад, зі списку.)

Припустимо, що в нас є загальний список категорій піц (наприклад, з декількох піцерій), який містить повторювані елементи. І нам треба створити список, який містить лише унікальні назви піци. За допомогою множини це завдання легко вирішується у кілька рядків коду:

```
allPizzaTypes=['Veggie', 'Pepperoni', 'Meat',  
               'Margherita', 'Meat', 'BBQ Chicken',  
               'Hawaiian', 'Veggie']  
  
uniquePizzaTypes=list(set(allPizzaTypes))  
  
print(uniquePizzaTypes)  
['BBQ Chicken', 'Veggie', 'Meat', 'Margherita',  
 'Hawaiian', 'Pepperoni']
```

Тепер розглянемо наступну ситуацію: ми маємо логіни клієнтів двох додатків, які зберігаються окремо (наприклад, у двох списках). Нам потрібно визначити:

- які клієнти користуються обома додатками;
- які клієнти користуються тільки першим (тільки другим) додатком;
- загальний перелік логінів клієнтів.

Перше завдання — це перетин двох множин, друге — знаходження різниці між першою і другою множинами (і навпаки), і третє завдання — це об'єднання множин.

```

usersApp1=['user134', 'admin56', 'superBob',
           'student', 'spider34']
usersApp2=['fifa56', 'user134', 'studGood', 'admin56',
           'spider34']

print("App1+App2 users:")
print(set(usersApp1)&set(usersApp2))

print("App1 users only:")
print(set(usersApp1)-set(usersApp2))

print("App2 users only:")
print(set(usersApp2)-set(usersApp1))

print("All users:")
print(set(usersApp1)|set(usersApp2))

```

Результат:

```

App1+App2 users:
{'admin56', 'spider34', 'user134'}
App1 users only:
{'student', 'superBob'}
App2 users only:
{'studGood', 'fifa56'}
All users:
{'admin56', 'user134', 'superBob_', 'spider34',
 'studGood', 'student', 'fifa56'}

```

Рисунок 29

І ще один приклад: у нас є два слова і потрібно визначити, чи є друге слово анаграмою першого.

Анаграма — це прийом, який полягає у перестановці літер певного слова, що в результаті дає інше слово.

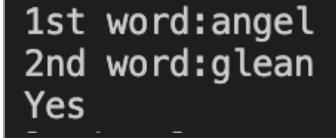
Ми пам'ятаємо, що порядок елементів у множині не має жодного значення. Відповідно, якщо множини складаються з одного набору символів, які знаходяться у різній послідовності, то фактично це та сама безліч.

```
word1=input("1st word:")
word2=input("2nd word:")

if set(word1)==set(word2):
    print("Yes")
else:
    print("No")
```

Результат:

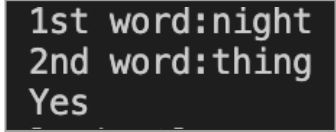
Кейс 1.



```
1st word:angel
2nd word:glean
Yes
```

Рисунок 30

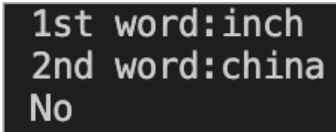
Кейс 2.



```
1st word:night
2nd word:thing
Yes
```

Рисунок 31

Кейс 3.



```
1st word:inch
2nd word:china
No
```

Рисунок 32

3. СЛОВНИКИ

3.1. Асоціативні масиви

Більшість вивчених нами структур, які є колекціями, засновані на ідеї масивів, тобто впорядкованих наборів елементів.

Упорядкованість — це властивість колекції (набору), коли кожен її елемент характеризується не тільки своїм значенням, але й індексом (порядковим номером елемента в колекції). Таким чином, під час роботи зі списками або кортежами ми використовуємо індексацію для доступу до певного елемента. При цьому індексом є лише ціле значення. Однак, використовувати число для ідентифікації значення (деякої інформації про об'єкт) не завжди зручно і може суперечити реальній ситуації в предметній області.

Наприклад, для ідентифікації рейсу літака використовується літерно-цифровий код. Або якщо за логіном користувача (які унікальні), потрібно дізнатися пароль?

Уявімо, що ми можемо використовувати лише індекси у цих ситуаціях. Тоді для вирішення завдання нам знадобляться вже два списки. Будемо зберігати в одному списку логіни, знаходити потрібний логін і отримувати його індекс. Далі за цим індексом потрібно отримати пароль з іншого списку. Це, звичайно, можливо, але дуже незручно та заплутано.

Набагато зручніше було б скористатися зв'язкою «логін» — «пароль», де «логін» — ідентифікатор. Анотація структури даних, яка, як індекс, може використовувати довільний тип даних (а не тільки цілі числа), називається **асоціативним масивом**. Доступ до елемента у цьому випадку здійснюється через ключ.

Така асоціація є зручною та однозначно відображає багато реальних зв'язків у предметних областях. Наприклад, нам потрібно дізнатися середній бал студента *Joe Smith*, ми не знаємо (і не повинні знайти) його порядковий номер у групі студентів. Ми отримуємо потрібну інформацію (середній бал) через його прізвище. І тут прізвище було б ключем асоціативного масиву, а середній бал — значенням.

3.2. Хеш-таблиці

Для представлення асоціативних масивів у пам'яті, Python використовує хеш-таблиці. Термін «хеш-таблиця» говорить про реалізацію як конкретний спосіб організації ваших даних у пам'яті.

Хеш-таблиці — це спосіб організації даних типу «ключ-значення» у пам'яті, у вигляді спеціальної структури, де індекс (позиція елемента в структурі) є результатом роботи хеш-функції.

Для отримання індексу в хеш-таблиці, ключі перетворюються як раз за допомогою функції хешування, а сам процес такого перетворення називається хешуванням.

Фактично, хеш-функція — це функція, яка отримує рядок (ключ) і повертає число (індекс).

Абстрактно, цей процес можна відтворити так:

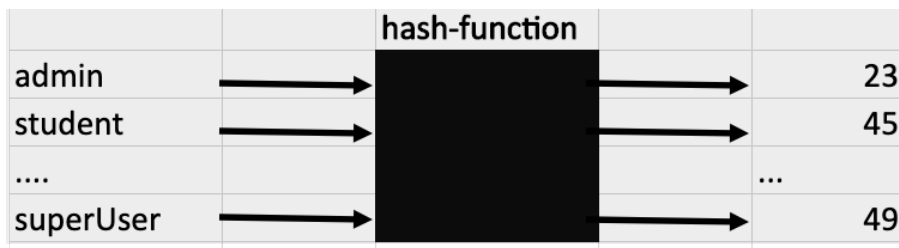


Рисунок 33

До хеш-функції пред'являються наступні вимоги:

- послідовність: припустимо, що ми передали хеш-функції рядок «**admin**» і отримали 23, і щоразу, коли ми передаватимемо їй рядок «**admin**», отримуватимемо лише 23;
- різним ключам мають відповідати різні індекси (у кожного вхідного слова має бути своє число).

Як це працює? Припустимо, що усі значення середніх балів студентів зберігатимуться в масиві.

Ми подаємо на вхід хеш-функції рядок «*Joe Smith*». Хеш-функція видає значення «2». Зберігаємо середній бал цього студента в елементі масиву з індексом 2.

При такому підході до організації даних, операції пошуку та вставки елемента в набір виконуються швидше, оскільки значення ключів перетворюються на індекси масиву, в якому зберігаються лише дані. Схематично, це можна відтворити так, як на рисунку 34.

А тепер припустимо, що нам потрібно дізнатися про середній бал студента *Bob Dylan*. Шукати в масиві нічого не доведеться, ми просто передаємо ключ «*Bob Dylan*» нашій хеш-функції, отримаємо індекс 4 і за цим індексом витягнемо з масиву середній бал цього студента — 91.

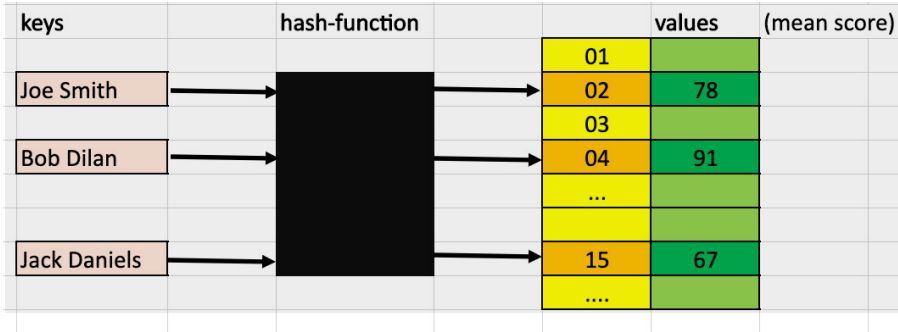


Рисунок 34

Хеш-функція завжди знає розмір масиву (N) і повертатиме індекси в його межах (тобто від 0 до $N-1$).

Окрема проблема, з якою ми можемо зіткнутися під час створення хеш-таблиці — це колізії (зіткнення). Ця ситуація може виникнути, коли функція хешування видасть той самий індекс для різних ключів. Для уникнення зіткнень або зменшення їх кількості потрібна підходяща функція хешування.

«Підходящою» вважається хеш-функція, для якої виконуються наступні умови:

- при подачі на вхід функції одного і того ж ключа, ми завжди будемо отримувати одне й теж хеш-значення (індекс);
- не може існувати двох різних ключів, котрим функція поверне одне й теж хеш-значення;
- навіть невелика зміна (відмінність) ключа дасть настільки відмінне хеш-значення, що між попереднім і новим хеш-значеннями виявити зв'язок (залежність) не вийде;
- швидкість обчислення хеш-значення приблизно однакова для будь-якого значення ключа.

У Python більша частина незмінних типів даних (наприклад, кортежі) хешована, тобто мають хеш-значення. Для виконання операції хешування (у тому числі й неявного, наприклад, під час створення словників, з якими ми познайомимося згодом) інтерпретатором Python, використовується вбудована функція `hash()`. Ця функція задовольняє усім переліченим вище вимогам, тому виконувати підбір та реалізацію відповідної хеш-функції Python-розробнику немає потреби.

3.3. Створення словника

В Python реалізація хеш-таблиць представлена у вигляді типу даних (колекцій) «Словник» (*Dictionary*).

Словник — це колекція елементів, кожен із яких являє собою пару «ключ-значення». Вивчені раніше списки на практиці більше підходять для зберігання однорідних даних. Наприклад, список цін на 10 товарів, які є чисельними значеннями (цілими або речовими), зручно зберігати у списку та обробляти за номером товару. А от різного роду інформацію, наприклад, про характеристики якогось об'єкта предметної області зручніше обробляти, посилаючись на назви цих характеристик.

Припустимо, що у нас є така інформація про клієнта: ім'я, прізвище, вік, логін, дата реєстрації в системі. Звичайно, можна зберігати ці дані у списку:

```
['Joe', 'Smith', 25, 'admin25', '10.01.2022']
```

Однак, при такій організації даних для отримання, наприклад, віку, потрібно чітко знати, що це третій елемент у списку.

Набагато зручніше обробляти ці дані, представивши їх у вигляді:

```
firstName: 'Joe',  
lastName: 'Smith',  
age: 25,  
login: 'admin25',  
signUpDate: '10.01.2022'
```

Ми можемо звертатися до даних за ключем, а не за порядковим номером колекції. Саме таку організацію даних і забезпечує словник.

При цьому ключі в словнику мають бути унікальними, оскільки вони оброблятимуться функцією хешування для отримання унікальних індексів у хеш-таблицях. Тому в словниках не може бути двох елементів з однаковим ключем.

Елементи словника не впорядковані, тобто порядок елементів даних у словнику не фіксований. Тому для доступу до елементів використовується не їх порядковий номер у наборі (індекс), а ключ.

В Python ключем може бути об'єкт будь-якого незмінного типу даних: рядок, булева змінна, число (ціле або речове), кортеж. Також ключем у словнику може бути елемент типу [frozenset](#).

Значення елемента словника може бути будь-якого типу даних і мати необмежений рівень вкладеності.

Словники можна змінювати: додавати нові пари «ключ-значення», оновлювати існуючі значення в парах (отримавши доступ за ключем), видаляти елементи словника.

Примітка: змінювати значення ключів в існуючих елементах (парах) словника не можна.

Подібно до розглянутих раніше колекцій, словники також можна створювати або порожніми, або з деяким набором елементів.

Для того, щоб створити словник з даними, потрібно набір пар у форматі «ключ: значення» помістити у фігурні дужки `{}`. Всередині `{}` пари поділяються комою.

Формат словника:

```
D = {  
    <key1>: <value1>,  
    <key>2: <value2>,  
    .  
    .  
    .  
    <keyM>: <valueM>  
}
```

Наприклад:

```
myDict1= {'key1': 1, 'key2': 20.5, 'key3': True}  
myDict2= {1: 'student', 2: 'admin'}  
  
print(myDict1)  #{'key1': 1, 'key2': 20.5, 'key3': True}  
print(myDict2)  #{1: 'student', 2: 'admin'}
```

Створимо словник, який містить характеристики книги:

```
bookDict = {'author': 'Eric Matthes',  
            'title': 'Python Crash Course',  
            'price': 14.43,  
            'reading age': '12 years and up',  
            'language': 'English'  
}
```

Також для створення словника можна використати вбудовану функцію `dict()`. Як аргументи, їй можна передати будь-яку послідовність (колекцію), яка містить набори з двох елементів (тобто пари). Тоді перший елемент набору стане ключем у парі, а другий — значенням.

Розглянемо на прикладах варіанти створення словника з використанням функції `dict()`:

1. Зі списку кортежів:

```
myDict3 = dict(("a", 111), ("b", 222))
```

2. Зі списку списків:

```
myDict4 = dict(["a", 111], ["b", 222])
```

3. Зі списку двосимвольних рядків:

```
myDict5 = dict(['qw', 'er', 'ty'])
print(myDict5)  #{'q': 'w', 'e': 'r', 't': 'y'}
```

4. Із двох списків (списку ключів та списку значень) за допомогою функції `zip()`:

```
keyList=['a','b']
valueList=[111,222]
myDict6=dict(zip(keyList,valueList))
print(myDict6)  #{'a': 111, 'b': 222}
```

Також можна скористатися функцією `dict()` і у таким спосіб:

```
myDict7=dict(firstName='Joe', lastName='Smith')
print(myDict7)  #{'firstName': 'Joe', 'lastName': 'Smith'}
```

Якщо наші ключі є простими рядками (що складаються з одного слова), їх можна вказати як аргументи функції `dict()`.

Якщо нам потрібно створити порожній словник, то це також можна зробити двома способами:

```
myEmptyDict1={}
print(myEmptyDict1)    #{}

myEmptyDict2=dict()
print(myEmptyDict2)    #{}

```

Для визначення кількості елементів (пар) у словнику використовуйте вбудовану функцію `len()`:

```
bookDict={'author':'Eric Matthes',
          'title':'Python Crash Course',
          'price':14.43,
          'reading age':'12 years and up',
          'language':'English'
        }
print(len(bookDict))    #5

```

3.4. Методи словника

Як нам відомо, найпоширеніша операція при роботі з колекціями — це отримання значення елемента колекції. Так само й при роботі зі словниками: жодне завдання не обходиться без отримання значення елемента словника.

Словники є невпорядкованими наборами, тому отримання елемента за індексом неможливе. Звернення до елементів словника відбувається за допомогою ключа,

який вказується у квадратних дужках (коли, наприклад, під час роботи зі списками ми вказували індекс):

```
ім'яСловника[ім'яКлюча]
```

```
myDict1= {'key1': 1, 'key2': 20.5, 'key3': True}
print(myDict1['key1']) #1

bookDict={'author':'Eric Matthes',
          'title':'Python Crash Course',
          'price':14.43,
          'reading age':'12 years and up',
          'language':'English'
        }

print(bookDict['author']) #Eric Matthes
```

Якщо в такій операції ми вкажемо ім'я ключа, якого немає у словнику, то буде викликано виняток [KeyError](#).

```
print(bookDict['pages'])
```



Рисунок 35

Для отримання елемента за ключем також існує метод [get\(\)](#), з яким ми познайомимось трохи пізніше.

Для запобігання вказаній вище ситуації (звернення до неіснуючого ключа), перед операцією вилучення елемента по ключу, можна перевірити, чи існує такий ключ у словнику.

Якщо нам потрібно перевірити, чи є певний (потрібний нам) ключ у словнику, можна скористатися оператором [in](#).

Якщо ключ знайдено у списку ключів словника, ми отримаємо значення **True**, інакше — **False**.

```
bookDict={'author':'Eric Matthes',
          'title':'Python Crash Course',
          'price':14.43,
          'reading age':'12 years and up',
          'language':'English'
        }
```

```
infoType=input("What info do you want to know about
               the book?")

if infoType in bookDict:
    print(bookDict[infoType])
else:
    print("Sorry!")
```

Результат:

```
What info do you want to know about the book?page
Sorry!
_
```

Рисунок 36

```
What info do you want to know about the book?title
Python Crash Course
_
```

Рисунок 37

Оскільки словники є типами даних, що змінюються, то ми можемо змінювати і додавати елементи по ключу.

Для додавання нового елемента (пари «ключ-значення») потрібно просто звернутися до елемента за новим

ключем і задати йому нове значення за допомогою операції присвоєння:

```
bookDict = {'author': 'Eric Matthes',
            'title': 'Python Crash Course',
            'price': 14.43,
            'reading age': '12 years and up',
            'language': 'English'
            }
```

```
for dictKey, dicVal in bookDict.items():
    print("{}:{}".format(dictKey, dicVal))

bookDict['pagesN']=350
for dictKey, dicVal in bookDict.items():
    print("{}:{}".format(dictKey, dicVal))
```

Результат:

```
author:Eric Matthes
title:Python Crash Course
price:14.43
reading age:12 years and up
language:English
```

Рисунок 38

```
author:Eric Matthes
title:Python Crash Course
price:14.43
reading age:12 years and up
language:English
pagesN:350
```

Рисунок 39

Також можна додати пару, використовуючи змінні для ключа та значення елемента:

```
newInfo=input("What info about the book do you want
              to add?")
if newInfo!="":
    newInfoValue=input("Input value for the key
                       '{}':".format(newInfo))
    if newInfoValue!="":
        bookDict[newInfo]=newInfoValue
    else:
        print("No value for the key '{}':".
              format(newInfo))
else:
    print("No key!")
```

Якщо ми проведемо описану операцію для вже існуючого у словнику ключа, його попереднє значення зміниться на нове:

```
bookDict={'author':'Eric Matthes',
          'title':'Python Crash Course',
          'price':14.43,
          'reading age':'12 years and up',
          'language':'English'
          }
```

```
for dictKey, dicVal in bookDict.items():
    print("{}:{}".format(dictKey,dicVal))

bookDict['price']=12

for dictKey, dicVal in bookDict.items():
    print("{}:{}".format(dictKey,dicVal))
```


Результат:

```
author:Eric Matthes
title:Python Crash Course
price:14.43
reading age:12 years and up
language:English
```

Рисунок 40

```
author:Eric Matthes
title:Python Crash Course
price:12
reading age:12 years and up
language:English
```

Рисунок 41

Інтерпретатор Python «контролює» той момент, коли ключ має бути у словнику унікальним. Тому, при спробі додати до словника новий елемент із існуючим ключем відбувається оновлення значення ключа (як у нашому прикладі).

Більшість операцій для роботи зі словниками організовано у вигляді відповідних методів словників. Розглянемо найбільш популярні та корисні з них.

Метод словника `Python.get()` — зручний спосіб отримання значення за ключем зі словника без попередньої перевірки на існування ключа (і без виникнення виключення `KeyError`)

Загальний синтаксис:

```
dictName.get(<keyname>[, <defaultValue>])
```

Метод шукає у словнику `dictName` ключ `keyname` і повертає відповідне значення, якщо воно знайдено, то повертається значення `defaultValue`. Якщо параметр `defaultValue` не заданий при виклику методу, то повертається значення `None`:

```
bookDict={'author':'Eric Matthes',
          'title':'Python Crash Course',
          'price':14.43,
          'reading age':'12 years and up',
          'language':'English'}
```

```
print(bookDict.get('author')) #Eric Matthes
print(bookDict.get('page'))   #None
print(bookDict.get('page',0)) #0
```

Для оновлення словника, Python надає метод `update()`:

```
dictName.update(<iterObj>)
```

Метод `update()` оновить словник елементами, які є у складі аргумента-об'єкта. Аргумент має бути словником або об'єктом, що ітерується, з парами «ключ: значення» (наприклад, списком кортежів або списком списків).

Якщо `iterObj` є словником, то буде виконано об'єднання записів з `iterObj` із записами в `dictName` за наступним алгоритмом (для кожного ключа в `iterObj`):

- якщо ключ відсутній у `dictName`, до `dictName` додається відповідна пара “ключ-значення” з `iterObj`;
- якщо ключ вже є у `dictName`, відповідне значення в `dictName` для цього ключа оновлюється до значення з `iterObj`.

```
bookDict={'author':'Eric Matthes',
          'title':'Python Crash Course',
          'price':14.43,
          'reading age':'12 years and up',
          'language':'English'
        }
print(bookDict)
```

```
bookDict.update([('pages', 600), ('discount', True)])
print(bookDict)

bookDict.update([('pages', 700), ('online', False)])
print(bookDict)
```

Результат:

```
{'author': 'Eric Matthes', 'title': 'Python Crash Course',
 'price ': 'English'}
{'author': 'Eric Matthes', 'title': 'Python Crash Course',
 'price ': 'English', 'pages': 600, 'discount': True}
{'author': 'Eric Matthes', 'title': 'Python Crash Course',
 'price ': 'English', 'pages': 700, 'discount': True, 'online': False}
```

Рисунок 42

Такий метод дуже зручний, коли потрібно скопіювати усі елементи з одного словника до іншого.

```
studGr1={'Joe':75,'Bob':92}
studGr2={'Kate':62,'Joe':90, 'Jack':84}

print(studGr1) #{'Joe': 75, 'Bob': 92}
studGr1.update(studGr2)
print(studGr1) #{'Joe': 90, 'Bob': 92, 'Kate': 62,
                  'Jack': 84}
```

Видалити елемент із словника можна за допомогою вбудованої функції `del()`:

```
studGr2={'Kate':62,'Joe':90, 'Jack':84}
del studGr2['Jack']
print(studGr2) #{'Kate': 62, 'Joe': 90}
```

Для видалення усіх елементів із словника потрібно скористатися методом `clear()`:

```
studGr2.clear()
print(studGr2) #{} 
```

Для того, щоб отримати усі ключі словника (у вигляді списку), будемо використовувати метод `keys()`:

```
bookDict={'author':'Eric Matthes',
          'title':'Python Crash Course',
          'price':14.43,
          'reading age':'12 years and up',
          'language':'English'
          }

print(bookDict.keys())
#dict_keys(['author', 'title', 'price', 'reading age',
            'language'])
```

```
keyList=list(bookDict.keys())
print(keyList) #['author', 'title', 'price',
                'reading age', 'language']
```

А для отримання усіх значень словника (у вигляді списку), будемо використовувати `values()`:

```
bookDict = {'author': 'Eric Matthes',
            'title': 'Python Crash Course',
            'price': 14.43,
            'reading age': '12 years and up',
            'language': 'English'
            }
```

```
valuesList = list(bookDict.values())
print(valuesList)
#['Eric Matthes', 'Python Crash Course', 14.43,
  '12 years and up', 'English']
```

Метод `items()` повертає список кортежів, які містять пари «ключ-значення», із словника. Перший елемент у кожному кортежі — це ключ, а другий — це значення ключа:

```
bookDict = {'author': 'Eric Matthes',
            'title': 'Python Crash Course',
            'price': 14.43,
            'reading age': '12 years and up',
            'language': 'English'
            }
```

```
dictItems=list(bookDict.items())
print(dictItems)
```

Результат:

```
[('author', 'Eric Matthes'), ('title', 'Python Crash Course'),
 ('price', 14.43), ('reading age', '12 years and up'),
 ('language', 'English')]
```

Рисунок 43

Метод `pop()` використовується для видалення ключа із словника. Результат роботи — видалення елемента та повернення значення за ключем:

```
dictName.pop(<keyname>[, <defaultValue>])
```

Якщо ключа `keyname` у словнику `dictName` немає, то повернеться значення `defaultValue`.

```
bookDict={'author':'Eric Matthes',
          'title':'Python Crash Course',
          'price':14.43,
          'reading age':'12 years and up',
          'language':'English'
        }

print(bookDict)
```

```
delItem=bookDict.pop("price")
print("item {} was deleted".format(delItem))
print(bookDict)

delItem=bookDict.pop("discount","None")
print(delItem)
```

Результат:

```
{'author': 'Eric Matthes', 'title': 'Python Crash Course',
 'price': 14.43, 'reading age': '12 years and up',
 'language': 'English'} item 14.43 was deleted
{'author': 'Eric Matthes', 'title': 'Python Crash Course',
 'reading age': '12 years and up', 'language': 'English'}
None
```

Рисунок 44

Якщо під час виклику методу `pop()` не було встановлено значення параметра `defaultValue`, і вказаного ключа теж немає, буде викликано виняток `KeyError`.

```
delItem=bookDict.pop("discount")
```

KeyError ✕

Рисунок 45

Для перебору елементів словника традиційно використовується цикл `for`. При цьому, на кожній ітерації циклу, ми отримуємо ключ словника.

Виведемо усі ключі у словнику (кожен з нового рядка, у стовпчик):

```
bookDict={'author':'Eric Matthes',
          'title':'Python Crash Course',
          'price':14.43,
          'reading age':'12 years and up',
          'language':'English'
        }

for dictKey in bookDict:
    print("{}:{}".format(dictKey, bookDict[dictKey]))
```

Результат:

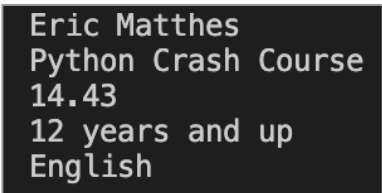
```
author
title
price
reading age
language
_
```

Рисунок 46

Маючи доступ до ключів, ми можемо трохи змінити код і вивести усі значення елементів словника:

```
for dictKey in bookDict:  
    print(bookDict[dictKey])
```

Результат:



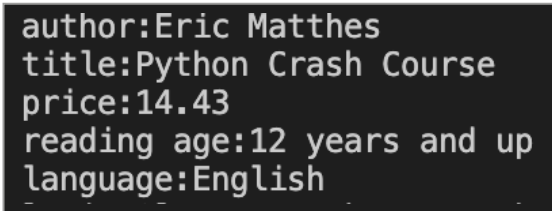
```
Eric Matthes  
Python Crash Course  
14.43  
12 years and up  
English
```

Рисунок 47

Або так, для зручнішого представлення інформації:

```
for dictKey in bookDict:  
    print("{}:{}".format(dictKey, bookDict[dictKey]))
```

Результат:



```
author:Eric Matthes  
title:Python Crash Course  
price:14.43  
reading age:12 years and up  
language:English
```

Рисунок 48

Як зазначалося раніше, ключі словника є незмінним набором, тобто ми можемо додати або видалити повністю пару «ключ-значення», змінити значення за ключем, але не сам ключ.

Таким чином, якщо до попереднього прикладу, всередині циклу додати рядок коду, що змінює змінну `dictKey`, то виконається лише зміна значення цієї змінної, а не поточного значення ключа з `bookDict`, який у цій ітерації перебував у змінній `dictKey`.

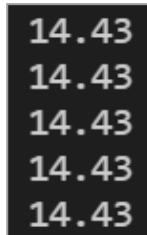
```
for dictKey in bookDict:  
    dictKey='name'  
    print(bookDict[dictKey])
```

У наведеному прикладі ми взагалі отримаємо виняток `KeyError`, оскільки ключа `name` у словнику `bookDict` немає.

Якщо ж ми задамо змінній `dictKey` значення існуючого ключа (наприклад, `price`),

```
for dictKey in bookDict:  
    dictKey='price'  
    print(bookDict[dictKey])
```

то ми продублюємо виведення значення за цим ключем стільки разів, скільки виконуватиметься цикл (а саме стільки разів, скільки ключів є на момент запуску циклу у словнику `bookDict`, тому що ми використовуємо оператор `in`).



```
14.43  
14.43  
14.43  
14.43  
14.43
```

Рисунок 49

Перебір усіх ключів та значень словника можливий у циклі з використанням методу `items()`:

```
bookDict={'author':'Eric Matthes',  
          'title':'Python Crash Course',  
          'price':14.43,  
          'reading age':'12 years and up',  
          'language':'English'  
}
```

```
for dictKey, dicVal in bookDict.items():  
    print("{}:{}".format(dictKey,dicVal))
```

Результат:

```
author:Eric Matthes  
title:Python Crash Course  
price:14.43  
reading age:12 years and up  
language:English
```

Рисунок 50

Ми не можемо зробити копію словника простим присвоєнням (як і в ситуації зі списками):

```
myDict2=myDict2
```

Тому що в цьому випадку `myDict2` буде тільки поси-
ланням на `myDict1`, а зміни, проведені в `myDict1`, будуть
автоматично проведені і в `myDict2`.

Для створення копії необхідно використовувати ме-
тод `copy()`.

```
myDict1= {'key1': 1, 'key2': 20.5, 'key3': True}
myDict2=myDict1.copy()

myDict1['key1']=111
print(myDict1['key1']) #111
print(myDict2['key1']) #1
```

4. Практичні приклади використання

Тепер, коли ми познайомилися з особливостями усіх колекцій у Python (списки, кортежі, множини та словники), розглянемо кілька прикладів їх використання, що найчастіше зустрічаються на практиці.

4.1. Приклад 1: пошук у словнику

Завдання пошуку інформації зустрічається у кожному додатку. Тому давайте розглянемо, як шукати потрібний елемент у словнику.

Ми знаємо, що звернення до елементів словника відбувається через ключ. Тому необхідно буде пройти по елементах списку (словника) перевіряючи потрібну пару «ключ-значення» на відповідність критерію пошуку.

Припустимо, що у нас є список користувачів, і по кожному з них відома така інформація: ім'я, вік, логін. Представимо цей набір даних у вигляді списку словників, де кожен словник зберігає вказану інформацію про користувача.

```
users = [  
    {'name': 'Hanna', 'age': 10, 'login': 'user56'},  
    {'name': 'Mark', 'age': 15, 'login': 'usER111'},  
    {'name': 'Jane', 'age': 17, 'login': 'superGirl'},  
    {'name': 'Jack', 'age': 7, 'login': 'userJack'}]
```

Спочатку спитаємо користувача, за яким полем будемо проводити пошук і дізнаємося також про критерій пошуку

(необхідне користувачеві значення ключа). І далі, як ми вже визначилися, перебиратимемо у циклі всі елементи списку (словники), перевіряючи потрібний нам ключ на потрібне значення.

Як тільки зустрінемо потрібний елемент, одразу зупинимо пошук. Для цього введемо змінну `isElementFound`, якщо після закінчення пошуку її значення все ще залишається `False`, отже пошук виявився невдалим — потрібного нам елемента (користувача з потрібними характеристиками) в наборі немає.

```
keyName=input("Input info type:")
keyValue=input("Input info value:")
isElementFound=False

for user in users:
    if user.get(keyName)==keyValue:
        print("Element is found!")
        for key,val in user.items():
            print("{}:{}".format(key,val))
        isElementFound=True
        break
if isElementFound==False:
    print("Element is not found!")
```

Результат на рисунках 51-52:

```
Input info type:login
Input info value:usER111
Element is found!
name:Mark
age:15
login:usER111
```

Рисунок 51

```
Input info type:name
Input info value:Joe
Element is not found!
```

Рисунок 52

Однак, якщо ми спробуємо здійснити такий пошук за полем «age», то результат пошуку завжди буде невдалим:

```
Input info type:age
Input info value:10
Element is not found!
```

Рисунок 53

Причина в тому, що поле «age» містить числове значення, а результатом виконання рядка коду:

```
keyValue=input("Input info value:")
```

буде рядок. Нам для цього випадку (пошук за віком) потрібно перетворити змінну `keyValue` на число (наприклад, за допомогою тернарного оператора) перед її подальшим використанням у процесі пошуку:

```
keyValue=keyValue if keyName!='age' else int(keyValue)
```

Результат:

```
Input info type:age
Input info value:15
Element is found!
name:Mark
age:15
login:usER111
```

Рисунок 54

4.2. Приклад 2. Сортування елементів словника

Припустимо, що в нас є словник, який містить дані про фільм.

```
filmDict={'originalTitle':'Forever',
          'creator':'Matthew Miller',
          'rate':8.3,
          'description':'A 200-year-old man worksin
            the New York City Morgue trying to find
            a key to unlock the curse of his
            immortality.',
          'years':[2014,2015]
        }

for key,value in filmDict.items():
    print("{}:{}".format(key,value))
```

Результат:

```
originallitle: Forever
creator:Matthew Miller
rate:8.3
description:A 200-year-old man worksin the New York City Morgue
trying to find a key to unlock the curse of his immorta lity.
years:[2014, 2015]
```

Рисунок 55

Однак нам потрібно вивести цю інформацію так, щоб ключі словника (характеристики фільму) були в алфавітному порядку.

Раніше ми вже працювали із вбудованою функцією `sorted()`, яка працює з об'єктами, що ітеруються, тому підійде й для роботи із словником.

Хоча під час ітерування словника ми отримуємо лише ключі. Отже, щоб одержати усі елементи (пар «ключ-значення») словника, потрібно скористатися методом `items()`.

Метод `items()`, як ми вже знаємо, видасть нам список кортежів `[(key1, value1), (key2, value2), ...]`. Знову ж таки, елемент списку — це кортеж, комплексний тип даних (містить декілька значень), тому потрібно вказати, за яким значенням кортежу проводитиметься сортування. Тут слід згадати, що функція `sorted()`, як і метод списків `sort()`, має параметр `key`.

Цей параметр використовується для завдання функції, яка викликатиметься для кожного елемента списку перед виконанням порівнянь.

Нам потрібно, щоб ця функція просто витягувала перший елемент кортежу (ключ словника). Тому, немає сенсу створювати окрему функцію, зручніше застосувати `lambda`-функцію.

```
sortedTuples = sorted(filmDict.items(),
                      key=lambda x: x[0])
print(sortedTuples)
```

```
for element in sortedTuples:
    print(element)
```

Функція `sorted()` не змінює переданий до неї об'єкт, а повертає новий. На даному етапі ми отримали список кортежів, відсортованих за першим елементом:


```
( 'creator', 'Matthew Miller')
( 'description', 'A 200-year-old man worksin the New York
  City Morgue trying to find a key to unlock the curse of his
  immortality.')
( 'originalTitle', 'Forever')
( 'rate', 8.3)
( 'years', [2014, 2015])
```

Рисунок 56

Тепер зберемо їх назад у новий словник:

```
filmDictSorted=dict(sortedTuples)
for key,value in filmDictSorted.items():
    print("{}:{}".format(key,value))
```

Результат:

```
creator:Matthew Miller
description:A 200-year-old man worksin the New York City Morgue
trying to find a key to unlock the curse of his immorta lity.
originalTitle:Forever rate:8.3
years:[2014, 2015]
```

Рисунок 57

Є й інший спосіб розв'язання цього завдання. Ми можемо створити відсортований список ключів. І далі, використовуючи відсортовану послідовність ключів у цьому списку, виведемо інформацію зі словника.

```
keyList=list(filmDict.keys())
print(keyList)
sortedKeys=sorted(keyList)
```

```
print(sortedKeys)

for key in sortedKeys:
    print("{}:{}".format(key, filmDict[key]))
```

Результат:

```
['originalTitle', 'creator1', 'rate1', 'description1', 'years']
['creator', 'description', 'originalTitle', 'rate', 'years']
creator:Matthew Miller
description:A 200-year-old man worksin the New York City Morgue
trying to find a key to unlock the curse of his immorta lity.
originalTitle:Forever
rate:8.3
years:[2014, 2015]
```

Рисунок 58

Використовуючи підхід, реалізований у першому способі, ми зможемо відсортувати словник за значеннями.

Розглянемо на прикладі словника, що містить інформацію про книгу.

```
bookDict = {'author': 'Matthes',
            'title': 'Python',
            'reading age': '12',
            'language': 'English'
            }
```

```
print("Unsorted book info:")
for key,value in bookDict.items():
    print("{}:{}".format(key,value))
```

Єдине, що нам потрібно виправити — змінити в нашій lambda-функції індекс з нульового на перший, оскільки

значення елемента словника — це другий елемент у кортежі. Таким чином ми змінимо ключ сортування.

```
print("Book info sorted by element value:")
bookDictSorted=dict(sorted(bookDict.items(),
                           key=lambda x: x[1]))
for key,value in bookDictSorted.items():
    print("{}:{}".format(key,value))
```

Результат:

```
Unsorted book info:
author:Matthes
title:Python
reading age:12
language:English
Book info sorted by element value:
reading age:12
language:English
author:Matthes
title:Python
```

Рисунок 59

4.3. Приклад 3. Сортування списку словників

Розглянемо складнішу, але й більш реальну ситуацію для сортування: наш список інформації користувача. Припустимо, що цей набір потрібно відсортувати за ім'ям користувача.

У цьому завданні нам підійде рішення, що використовується у Прикладі 2: застосування вбудованої функції `sorted()` і завдання ключа сортування за допомогою `lambda`-функції.

Єдиний нюанс — це те, що звертатися до поля словника потрібно через ім'я ключа, а не за допомогою індексу, як у Прикладі 2:

```
users = [  
    {'name': 'Hanna', 'age': 10, 'login': 'user56'},  
    {'name': 'Mark', 'age': 15, 'login': 'usER111'},  
    {'name': 'Jane', 'age': 17, 'login': 'superGirl'},  
    {'name': 'Jack', 'age': 7, 'login': 'userJack'}  
]  
  
sortedUsersbyName=sorted(users, key=lambda x: x['name'])  
  
print("Users list sorted by name:")  
  
for user in sortedUsersbyName:  
    for key,value in user.items():  
        print("{}:{}".format(key,value))
```

Результат:

```
Users list sorted by name:  
name:Hanna  
age:10  
login:user56  
name:Jack  
age:7  
login:userJack  
name:Jane  
age:17  
login:superGirl  
name:Mark  
age:15  
login:usER111
```

Рисунок 60

4.4. Приклад 4. Фільтри в словнику

Завдання пошуку елементів з потрібними нам характеристиками — одне з найпоширеніших, і найчастіше його називають фільтрацією.

Фільтр — це набір критеріїв (у найпростішому випадку маємо справу з одним критерієм), яким мають відповідати елементи певної колекції.

Продовжимо роботу з нашим списком користувачів:

```
users = [
    {'name': 'Hanna', 'age': 10, 'login': 'user56'},
    {'name': 'Mark', 'age': 15, 'login': 'usER111'},
    {'name': 'Jane', 'age': 17, 'login': 'superGirl'},
    {'name': 'Jack', 'age': 7, 'login': 'userJack'}
]
```

Під час роботи зі списками ми вже використовували вбудовану функцію `filter()`, яка витягує ті елементи зі списку, для яких функція, зазначена як перший аргумент, повертає значення `True`. Припустимо, що нам потрібно вивести тих користувачів, які старше 12 років. Очевидно, що фільтр буде: `users['age'] > 12`. Також зручно використовувати тут `lambda`-функцію, яка повертатиме `True`, якщо фільтр на елементі «спрацював».

Пам'ятаємо, що функція `filter()` повертає об'єкт,

```
<filter object at 0x10b8c5490>
```

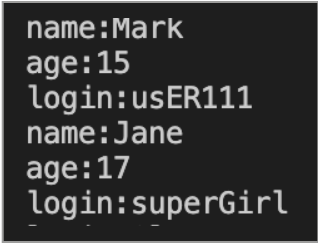
Рисунок 61

тому нам також буде необхідно виконати його перетворення в список:

```
users12=list(filter(lambda user: user['age'] >12, users))

for user in users12:
    for key,val in user.items():
        print("{}:{}".format(key,val))
```

Результат:



```
name:Mark
age:15
login:usER111
name:Jane
age:17
login:superGirl
```

Рисунок 62



Урок 6

Кортежі, Множини, словники

© STEP IT Academy, www.itstep.org

© Анна Егошина

Усі права на фото-, аудіо- і відеотвори, що охороняються авторським правом і фрагменти яких використані в матеріалі, належать їх законним власникам. Фрагменти творів використовуються в ілюстративних цілях в обсязі, виправданому поставленим завданням, у рамках учбового процесу і в учбових цілях, відповідно до законодавства про вільне використання твору без згоди його автора (або іншої особи, яка має авторське право на цей твір). Обсяг і спосіб цитованих творів відповідає прийнятим нормам, не завдає збитку нормальному використанню об'єктів авторського права і не обмежує законні інтереси автора і правовласників. Цитовані фрагменти творів на момент використання не можуть бути замінені альтернативними аналогами, що не охороняються авторським правом, і відповідають критеріям добросовісного використання і чесного використання.

Усі права захищені. Повне або часткове копіювання матеріалів заборонено. Узгодження використання творів або їх фрагментів здійснюється з авторами і правовласниками. Погоджене використання матеріалів можливе тільки якщо вказано джерело.

Відповідальність за несанкціоноване копіювання і комерційне використання матеріалів визначається чинним законодавством.