```
>>> print(d)
{}
```

7.3.4. Приклади розв'язування задач

Приклад. Написати програму для знаходження відстані між двома точками заданими своїми координатами. Для збереження даних точки скористатися словником.

```
p1={'x':0, 'y':0}
p2={'x':0, 'y':0}
print('введіть координати 1-ї точки: ')
p1['x']=float(input('x:'))
p1['y']=float(input('y:'))
print('введіть координати 2-ї точки: ')
p2['x']=float(input('x:'))
p2['y']=float(input('y:'))
import math
lenP=math.sqrt((p1['x']-p2['x'])**2+(p1['y']-
          p2['y'])**2)
print('відстань між точками ({}; {}) та ({}; {}) =
          {:.3}'.format(p1['x'], p1['y'], p2['x'],
          p2['y'],lenP))
```

7.4. Рядкові величини

З точки зору комп'ютера, текст – це набір символів, кожен з яких має свій код. Аналогічно і в мовах програмування, найменшою текстовою одиницею є символ. Символ – це або буква, або цифра, або інший спеціальний символ (символ пропуску, символ нового рядка і т.д.). А уже весь текст подається у вигляді рядка, який є послідовністю символів. Ми вже зустрічалися з рядками, а саме з рядковими літералами.

Як і більшість мов програмування, мова Python часто використовується для опрацювання текстів: пошуку в тексті, заміни окремих

частин тексту і т.д. Для роботи з текстом у мові Python передбачений спеціальний рядковий тип даних `str`.

Рядковий тип або просто рядки в Python – це послідовність символів, яка використовується для зберігання і подання текстових даних. Користуючись рядковим типом, можна працювати з усім, що може бути подано в текстовому вигляді.

У мові Python символами рядка можуть бути будь-які символи, що підтримуються стандартом Unicode. У Python 3 немає ASCII-рядків, якщо необхідно отримати рядок строго в кодуванні ASCII, необхідно скористатися відповідним методом перекодування.

Варто відмітити, що в мові Python немає типу для символа рядка. Кожен символ рядка є також рядком. Також рядковий тип є незмінюваний.

7.4.1. Рядкові літерали та їх задання

В мові Python існує декілька способів задання рядкових літералів. Наприклад, можна розмістити текст в одинарних або подвійних лапках, і такий запис буде сприйматися, як текстовий літерал:

```
>>> print('Привіт')
Привіт
>>> print("Привіт")
Привіт
```

Наявність двох таких варіантів забезпечує створення рядків, які будуть містити одинарні чи подвійні лапки всередині тексту і однозначно визначати кінці рядка.

```
>>> print("Ім'я")
Ім'я
>>> print('Національний університет "Чернігівський
     колегіум".')
Національний університет "Чернігівський колегіум".
```

Іншим методом додавання в рядок одинарних чи подвійних лапок є так зване екранування символу. Для екранування символу перед ним записується символ зворотнього слешу (бекслеш, backslash) «\».

```
>>> print('Ім\'я')
Ім'я
```

```
>>> print("Національний університет \"Чернігівський
     колегіум\".")
Національний університет "Чернігівський колегіум".
```

Рядкові літерали можна присвоювати змінним, які матимуть рядковий тип, і в подальшому використовувати їх:

```
>>> s='Привіт світ!'
>>> print(s)
Привіт світ!
```

Багаторядкові текстові блоки (Багаторядкові літерали)

При роботі з текстовими даними інколи виникає необхідність в заданні багаторядкового текстового блоку. Такий текстовий блок може використовуватися як багаторядковий коментар до програми чи анотація (синтаксис, на якому будуються підказки в мові Python) до функції.

Для того, щоб деякий текст, записаний в декілька рядків, вважався єдиним текстовим літералом, його необхідно взяти в три одинарні чи три подвійні лапки (''' або """). Окрім того в середині такого рядка можна використовувати одинарні чи подвійні лапки (головне, щоб не було трьох лапок підряд).

```
>>> '''Це дуже довгий
текст, записаний
в декілька рядків'''
'Це дуже довгий\nтекст, записаний\nв декілька рядків'
>>> print('''Це дуже довгий
текст, записаний
в декілька рядків''')
Це дуже довгий
текст, записаний
в декілька рядків
```

7.4.2. Задання рядків

Для задання порожнього рядку можна скористатися однією з наступних команд:

```
>>> s=''
>>> s=str()
```

Використання рядкових літералів

Як зазначалося раніше, рядки можна створювати, використовуючи рядкові літерали:

```
>>> s1='Привіт'
>>> s2="Привіт"
>>> c='''Це дуже довгий
текст, записаний
в декілька рядків'''
```

Введення рядків

Аналогічно до введення числових даних, введення рядкових даних відбувається з використанням функції input().

```
>>> s=input('Введіть рядок: ')
Введіть рядок: Привіт
>>> print(s)
Привіт
```

Приведення до рядкового типу

Будь який стандартний об’єкт мови Python можна привести до рядкового типу, який йому відповідає, або отримати для нього «неформальне» рядкове подання. Для цього необхідно скористатися функцією `str(object='')`, передавши в якості параметра об’єкт, значення якого необхідно привести до рядкового типу.

```
>>> s=str(5)
>>> print(s)
'5'
>>> s=str(True)
>>> print(s)
'True'
```

Об’єднання рядків

Нові рядки можуть бути створені методом конкатенації (об’єднання) декількох рядків. Для цього використовується перевизначена операція додавання («+»), яка використовується як операція конкатенації рядків.

```
>>> s1='Привіт '
>>> s2='світ!'
>>> s=s1+s2
>>> print(s)
Привіт світ!
```

Незважаючи на незмінюваність рядкового типу, рядок може бути розширеним, тобто до рядка можна додати деякий інший, використовуючи операцію конкатенації.

```
>>> s='Привіт'
>>> id(s)
10388768
>>> s=s+' Світ!'
>>> print(s)
Привіт Світ!
>>> id(s)
10388768
```

Враховуючи строгу типізацію мови Python і неможливість проводити операції у виразах з даними різних несумісних типів (рядок та число є несумісними типами), виконати об’єднання змінних рядкового та числового типів без додаткових перетворень не є можливим. Проте до проведення такого об’єднання можна привести числове значення до рядкового, а вже потім провести операцію конкатенації для двох рядкових значень.

```
>>> s1='Вавілон '
>>> n=15
>>> s2=str(n)
>>> print(s1+s2)
Вавілон 15
```

Багаторазове повторення рядка або дублювання рядка

Аналогічно до перевизначеної операції («+»), в мові Python перевизначена і операція множення «*». Якщо виконати операцію «*» рядка

s на ціле число n, то в результаті буде отриманий рядок, що складається з n повторень рядка s:

```
>>> s="СПАМ" * 5
>>> print(s)
СПАМСПАМСПАМСПАМСПАМ
```

7.4.3. Доступ до символів рядку. Зрізи

Кожен символ рядка має свій індекс (порядковий номер). Доступ до символів рядка відбувається за їх індексами. Як і в багатьох інших мовах програмування, нумерація символів рядка починається з нуля. Для того, щоб звернутися до певного символу рядка, необхідно вказати ім'я рядкової змінної та в квадратних дужках індекс необхідного символу (рядок[індекс]).

```
>>> s='Привіт!'
>>> s[0]
'П'
```

Індекси можуть бути від'ємними, в такому випадку нумерація буде відбуватися з кінця (кількість символів рядка + від'ємний індекс).

```
>>> s[-1]
'!'
```

Якщо ж розглянути всі індекси для рядка 'Привіт!' , отримаємо:

Рядок s	П	р	и	в	і	т	!
Індекс	s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]
Індекс	s[-7]	s[-6]	s[-5]	s[-4]	s[-3]	s[-2]	s[-1]

Якщо ж вказати індекс, який виходить за межі рядка (наприклад, s[8] або s[-9]), то генерується виняток: IndexError: string index out of range.

Можна перевірити приналежність деякого символу (підрядка) до рядка, використовуючи оператор in (підрядок in рядкок).

```
>>> s='Привіт!'
>>> 'р' in s
True
```

```
>>> 'a' in s
False
```

Зрізи (slice)

Досить часто, необхідно отримати не один якийсь символ (за індексом), а деякий набір символів за певними простими правилами. Наприклад: перші 5 символів, останні 3 символи, кожен другий символ. В таких завданнях замість перебору в циклі набагато зручніше використовувати так званий зріз (slice, slicing). Зріз – отримання з даного рядка набору з його символів. Зріз рядка також є рядком. Отримання одного символу рядка є найпростішим варіантом зрізу.

Слід пам'ятати, що беручи символ за індексом або зрізом, ми ніяк не змінюємо початковий рядок, ми просто скопіювали його частину для подальшого використання.

Задати зріз можна одним з двох варіантів:

`str[start: stop]`.

`str[start: stop: step]`.

Для рядку `str` береться зріз від символу з індексом `start`, до символу з індексом `stop` (не включаючи його), з кроком `step` (тобто будуть взяті символи з індексами `start`, `start + step`, `start + 2 * step` і т. д.). Також при записі зрізу деякі, а можливо і всі параметри можуть бути опущені (знаки двокрапки в записі все рівно залишаються). У випадку відсутності деяких параметрів їх значення встановлюється за замовчуванням, а саме: start = 0, stop = кількості символів рядку, step = 1.

```
>>> S='Привіт!'
>>> S[2:5]
'иві'
```

Якщо опустити другий параметр (залишивши двокрапку), то зріз береться до кінця рядка. Наприклад, щоб отримати зріз без перших двох символів, можна записати `S[2:]`. Якщо опустити перший параметр, то отримаємо зріз, який містить вказану кількість символів, що йдуть від початку рядка.

```
>>> S[2:]
'ивіт!'
```

```
>>> S[:3]
'При'
```

При заданні значення третього параметра, рівному 2, в зріз потрапить кожний другий символ рядку.

```
>>> S[::2]
'Пиі!'
>>> S[1::2]
'рвт'
```

Можна бачити, що взяття зрізу схоже на створення діапазону (`range()`).

Якщо значення параметру stop буде перевищувати кількість символів в рядку, воно буде проігнороване.

```
>>> S[2:50]
'ивіт!'
```

У випадку, якщо параметри start і stop мають від'ємні значення, то нумерація відбувається з кінця (кількість символів рядка + від'ємний індекс). Наприклад, `S[1: -1]` - це рядок без першого і останнього символу (зріз починається символом з індексом 1 і закінчується символом з індексом -1, не включаючи його).

```
>>> S[1:-1]
'ривіт'
>>> S[:-4]
'При'
>>> S[-4:]
'віт!'
```

Якщо параметр step має від'ємне значення, то зріз береться зправа наліво.

```
>>> S[::-1]
'!тівирП'
>>> S[-2::-1]
'тівирП'
>>> S[1:4:-1]
''
```

В останньому прикладі був отриманий порожній рядок, так як `start < stop`, а `step` від'ємний.

7.4.4. Виконання дій над рядками та їхніми елементами

Визначення довжини рядка

Як зазначалося раніше, рядки складаються з окремих символів. Для того щоб дізнатися кількість символів у рядку (довжину рядка), необхідно скористатися функцією `len(рядок)`.

```
>>> len('Привіт!')
7
```

Зміна елементів рядка

Як було зазначено раніше, рядковий тип є незмінюваним, тобто зміна значення символу рядка чи його вилучення є неможливими.

```
>>> s='Привіт!'
>>> s[0]='п'
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    s[0]='п'
TypeError: 'str' object does not support item
assignment
```

За необхідності можна записати оператори для зміни чи вилучення окремих символів рядка, проте в результаті ми отримаємо новий рядок.

```
>>> s='Hello'
>>> s=s[:1]+'a'+s[2:]
>>> print(s)
Hallo
>>> s=s[:1]+s[3:]
>>> print(s)
Hlo
```

Отримання коду символу та символу за кодом

Як було зазначено раніше, кожен символ текстового рядка має свій код. Інколи виникає необхідність отримання цього коду для певного символу,

або зворотна операція – отримання символу за його кодом. При виконанні таких дій варто пам'ятати, що символи в мові Python є символами Unicode.

Для отримання коду символу можна скористатися функцією `ord(c)`, за якою повертається ціле число, яке відповідає коду цього символу.

```
>>> ord('a')
97
>>> ord('€')
8364
>>> ord('½')
189
```

Зворотною до функції `ord()` є функція `chr(n)`, за якою для цілого числа `n` (від 0 до 1114111=0x10FFFF) повертається символ (рядок, що є єдиним символом), для якого `n` є його кодом.

```
>>> chr(97)
'a'
>>> chr(8364)
'€'
>>> chr(189)
'½'
```

7.4.5. escape-послідовності

В попередньому пункті при виведенні рядкової змінної (без використання функції Print), яка містила багаторядковий текст, в місцях, де мали бути переходи на новий рядок, можна було бачити пару символів «\n». Ця пара символів починається з зворотного слеша і називається escape-послідовністю або керувальною послідовністю. Така послідовність в мові Python не єдина.

Використовуючи escape-послідовності, в рядки можна вставляти символи, які важко ввести з клавіатури. Серед найбільш вживаних escape-послідовностей можна виокремити:

\f – перехід на нову сторінку;

\n – перехід на новий рядок;

\r – повернення каретки (лише в Windows);

\t – горизонтальна табуляція;

```
>>> print('Це речення \nв два рядки')
Це речення
в два рядки
```

Отже, за функцією `print()` було розпізнано escape-послідовність і виконано перехід на новий рядок.

Проте інколи є необхідність, щоб символ зворотнього слеша не сприймався, як початок escape-послідовності. Наприклад, це є важливим при заданні шляху до файлу в операційній системі Windows. Для такого відключення аналізу escape-послідовностей необхідно перед відкриваючою лапкою вставити символ r (у будь-якому регістрі),

```
>>> S = r'C:\newt.txt'
>>> print(S)
C:\newt.txt
```

Проте рядки з відключеним аналізом escape-послідовностей не можуть закінчуватися символом зворотнього слеша. Але це можна обійти:

```
>>> S = r'\n\n' + '\\'
>>> print(S)
\n\n\
>>> S = r'\n\n\\'[:-1]
>>> print(S)
\n\n\
```

7.4.6. Методи рядків

Для роботи з рядками в Python передбачена велика кількість вбудованих методів.

При роботі з методами рядків необхідно пам'ятати, що рядковий тип в Python є незмінюваним типом, тому всі рядкові методи повертають новий рядок, не змінюючи даного. Для того, щоб змінити сам рядок, необхідно викликати для нього потрібний метод і присвоїти його результат (рядкове значення) тій же змінній, в якій зберігається поточний рядок: `str=str.метод()`.

str.find(substr [, start [,end]]). Повертає найменший індекс, за яким знаходиться початок підрядка `substr` в зрізі `str[start:end]` (необов'язкові параметри `start` та `end` інтерпретуються як нотації зрізу). Тобто знаходиться перше входження підрядка в рядку. Значення, що повертається, є індексом рядка `str`. Якщо підрядок не знайдено, то повертається значення -1.

```
>>> 'habrahabr'.find('r')          # 3
>>> 'habrahabr'.find('r',4)        # 8
>>> 'habrahabr'.find('abr')        # 1
>>> 'habrahabr'.find('Abr')        # -1
```

str.rfind(substr [, start [,end]]). Повертає найбільший індекс, за яким знаходиться початок підрядка `substr` в зрізі `str[start:end]` (необов'язкові параметри `start` та `end` інтерпретуються як нотації зрізу). Тобто знаходиться останнє входження підрядка в рядку. Значення, що повертається, є індексом рядка `str`. Якщо підрядок не знайдено, то повертається значення -1.

```
>>> 'habrahabr'.rfind('abr')       # 6
>>> 'habrahabr'.rfind('abr',7)     # -1
```

str.index(substr [, start [,end]]). Повертає найменший індекс, за яким знаходиться початок підрядка `substr` в зрізі `str[start:end]` (необов'язкові параметри `start` та `end` інтерпретуються як нотації зрізу). Тобто знаходиться перше входження підрядка в рядку. Значення, що повертається, є індексом рядка `str`. Якщо підрядок не знайдено, то виникає виняток `ValueError`.

str.rindex(substr [, start [,end]]). Повертає найбільший індекс, за яким знаходиться початок підрядка `substr` в зрізі `str[start:end]` (необов'язкові параметри `start` та `end` інтерпретуються як нотації зрізу). Тобто знаходиться останнє входження підрядка в рядку. Значення, що повертається, є індексом рядка `str`. Якщо підрядок не знайдено, то виникає виняток `ValueError`.

str.startswith(prefix[, start[, end]]). Повертає `True`, якщо зріз `str[start:end]` (необов'язкові параметри `start` та `end` інтерпретуються як нотації зрізу) починається з префіксу `prefix` (`prefix`

може бути кортежем, в такому випадку як префікс перевіряються всі елементи кортежу), інакше – `False`.

```
>>> 'habrahabr'.startswith('ha')        # True
>>> 'habrahabr'.startswith('ha',1)      # False
>>> 'habrahabr'.startswith('ab',1)       # True
>>> 'habrahabr'.startswith(('ha','bla')) # True
>>> 'blablacar'.startswith(('ha','bla')) # True
```

`str.endswith(suffix[, start[, end]])`. Повертає `True`, якщо зріз `str[start:end]` (необов'язкові параметри `start` та `end` інтерпретуються як нотації зрізу) закінчується на суфікс `suffix` (`suffix` може бути кортежем, в такому випадку як суфікс перевіряються всі елементи кортежу), інакше – `False`.

```
>>> 'habrahabr'.endswith('abr')         # True
>>> 'habrahabr'.endswith('abr',1,4)     # True
>>> 'habrahabr'.endswith('abr',1,8)     # False
>>> 'habrahabr'.endswith(('abr','car')) # True
>>> 'blablacar'.endswith(('abr','car')) # True
```

`str.count(substr [, start [,end]])`. Повертає кількість входжень підрядка `sub` в зріз `str[start:end]` (необов'язкові параметри `start` та `end` інтерпретуються як нотації зрізу) без самоперетинів.

```
>>> 'habrahabr'.count('ha')       # 2
>>> 'hahahahahah'.count('hah')    # 3
>>> 'hahahahahah'.count('ha')     # 5
>>> 'hahahahahah'.count('ha',1,6) # 2
```

`str.isalpha()`. Повертає `True`, якщо рядок є непорожнім і складається лише з алфавітних символів, інакше – `False`. Алфавітні символи – це символи, які належать до категорії Юнікоду "Letter" ("Lm", "Lt", "Lu", "Ll", "Lo").

```
>>> 'Habr'.isalpha()        # True
>>> ''.isalpha()            # False
>>> 'Habr1'.isalpha()       # False
>>> 'Habr%'.isalpha()       # False
```

str.isdecimal(). Повертає `True`, якщо рядок є непорожнім і складається лише з десяткових цифр (десяткових символів), інакше – `False`. Десяткові символи – це символи, які належать до категорії Юнікоду "Number, Decimal Digit" ("Nd").

```
>>> '123'.isdecimal()       # True
>>> ''.isdecimal()          # False
>>> '1.1'.isdecimal()       # False
```

str.isdigit(). Повертає `True`, якщо рядок є непорожнім і складається лише з цифр, інакше – `False`. Цифри – це символи, які належать до категорій Юнікоду "Number, Decimal Digit" та "Number, Other" ("Nd", "No"). Наприклад, до них належать цифри надрядкового знаку.

```
>>> '²'.isdigit()      # True
```

str.isnumeric(). Повертає `True`, якщо рядок є непорожнім і складається лише з числових символів, інакше – `False`. Числові символи – це символи, які належать до категорії Юнікоду "Number" ("Nd", "No", "Nl").

```
>>> 'VII'.isnumeric()       # True
```

str.isalnum(). Повертає `True`, якщо рядок є непорожнім і складається лише з літеро-числових символів, інакше – `False`. Літеро-числові символи – це символи, які належать до категорій Юнікоду "Letter" та "Number" ("Lm", "Lt", "Lu", "Ll", "Lo", "Nd", "No", "Nl").

```
>>> 'Habr1'.isalnum()       # True
```

str.islower(). Повертає `True`, якщо рядок містить принаймні одну літеру, і всі літери записані в нижньому регістрі, інакше – `False`.

```
>>> 'habra habr'.islower()        # True
>>> 'habra habr 1'.islower()      # True
>>> 'habra Habr'.islower()        # False
>>> '1'.islower()                 # False
```

str.isupper(). Повертає `True`, якщо рядок містить принаймні одну літеру, і всі літери записані в верхньому регістрі, інакше – `False`.

```
>>> 'HABRAHABR'.isupper()         # True
>>> 'HABRAHABR 1'.isupper()       # True
>>> 'HaBRAHABR 1'.isupper()       # False
>>> '1'.isupper()                 # False
```

str.istitle(). Повертає `True`, якщо рядок містить принаймні одну літеру, і літери, записані в верхньому регістрі, не йдуть безпосередньо після літер в нижньому чи верхньому регістрі, а перед групою літер в нижньому регістрі завжди стоїть літера в верхньому регістрі, інакше – `False`.

```
>>> 'Habra Habr'.istitle()        # True
>>> '1Habra %Habr'.istitle()      # True
>>> 'Habra habr'.istitle()        # False
>>> 'hAbra Habr'.istitle()        # False
```

str.isspace().

Повертає `True`, якщо рядок є непорожнім і складається лише з символів пропуску (whitespace), інакше – `False`. Whitespace символи – це символи, які в базі даних Unicode визначені як «Other» або «Separator». Наприклад, до таких символів належать: пробіл, перехід на нову сторінку (\f), перехід на новий рядок (\n), переведення каретки (\r), горизонтальна та вертикальна табуляції (\t та \v).

```
'  '.isspace()              # True
'\n\t'.isspace()            # True
''.isspace()                # False
'Habr!'.isspace()           # False
```

str.isprintable().

Повертає `True`, якщо рядок містить лише символи, що друкуються (можуть бути виведені при друці), інакше – `False`. До символів, що друкуються, не входять символи пропуску (whitespace) окрім пробілу.

```
''.isprintable()            # True
'Habra habr!'.isprintable() # True
'\n\t'.isprintable()        # False
'Habra\nhabr'.isprintable() # False
```

str.upper(). Повертає копію рядка, в якому всі літери, записані в нижньому регістрі, будуть приведені до верхнього регістру.

```
>>> 'Habra Habr 5%2=1'.upper()
'HABRA HABR 5%2=1'
```

str.lower(). Повертає копію рядка, в якому всі літери, записані в верхньому регістрі, будуть приведені до нижнього регістру.

```
>>> 'Habra Habr 5%2=1'.lower()
'habra habr 5%2=1'
```

str.swapcase(). Повертає копію рядка, в якому всі літери, записані в верхньому регістрі, будуть приведені до нижнього регістру, а нижньому – до верхнього.

```
>>> 'Habra Habr 5%2=1'.swapcase()
'hABRA hABR 5%2=1'
```

str.title(). Повертає копію рядка, в якому перша літера кожного слова буде приведена до верхнього регістру, а всі інші – до нижнього. Першою літерою слова вважається літера, перед якою не міститься інші літери.

```
>>> 'hAbra Habr 5%2=1'.title()
'Habra Habr 5%2=1'
```

str.capitalize(). Повертає копію рядка, в якому перший символ, якщо він є літерою, буде приведений до верхнього регістру, а всі інші літери до нижнього.

```
>>> 'hAbra Habr 5%2=1'.capitalize()
'Habra habr 5%2=1'
>>> '5%2=1 hAbra Habr'.capitalize()
'5%2=1 habra habr'
```

str.replace(old, new[, count]). Повертає копію рядка, в якому всі входження підрядка `old` будуть замінені на новий підрядок `new`. Якщо задано парметри `count`, то буде виконано не більше чим `count` замін.

```
>>> 'habrahabr'.replace('a','A')
'hAbrAhAbr'
>>> 'habrahabr'.replace('a','A',2)
'hAbrAhabr'
```

str.lstrip([chars]). Повертає копію рядка з вилученими початковими символами, вказаними в рядку `chars`. Якщо параметр `chars` відсутній або `None`, то вилучаються пропуски.

```
>>> '   habrahabr   '.lstrip()
'habrahabr   '
>>> 'www.habr.com'.lstrip('cmowa.')
'habr.com'
```

str.rstrip([chars]). Повертає копію рядка з вилученими кінцевими символами, вказаними в рядку `chars`. Якщо параметр `chars` відсутній або `None`, то вилучаються пропуски.

```
>>> '   habrahabr   '.rstrip()
'   habrahabr'
>>> 'www.habr.com'.rstrip('cmowa.')
'www.habr'
```

str.strip([chars]). Повертає копію рядка з вилученими початковими та кінцевими символами, вказаними в рядку `chars`. Якщо параметр `chars` відсутній або `None`, то вилучаються пропуски.

```
>>> '   habrahabr   '.strip()
'habrahabr'
>>> 'www.habr.com'.strip('cmowa.')
'habr'
```

str.expandtabs(tabsize=8). Повертає копію рядка, в якому всі символи табуляції (\t) замінюються декількома пропусками, в залежності від стовпця табулювання і заданого розміру табуляції tabsize.

```
>>> '\t1\t10\t100'.expandtabs()
'        1       10      100'
>>> '\t1234567\t10'.expandtabs()
'        1234567 10'
>>> '\t12345678\t10'.expandtabs()
'        12345678        10'
>>> '\t1\t10\t100'.expandtabs(4)
'    1   10  100'
```

str.split(sep=None, maxsplit=-1). Повертає список слів, які отримуються розбиттям рядка за роздільником рядком `sep`. Якщо параметр `sep=None`, то роздільником буде виступати пропуск, і результуючий список не буде містити порожніх елементів. Якщо задано параметр `maxsplit`, то буде виконано не більше чим `maxsplit` розбиттів (результуючий список буде мати не більше `maxsplit+1` елемент).

```
>>> 'habrahabr'.split('a')
['h', 'br', 'h', 'br']
```

```
>>> 'haabrahabr'.split('a')
['h', '', 'br', 'h', 'br']
>>> 'habrahabr'.split('ab')
['h', 'rah', 'r']
>>> 'habrahabr'.split('a',2)
['h', 'br', 'habr']
```

str.join(iterable). Повертає рядок, який є результатом конкатенації всіх рядків з iterable. Під час конкатенації між рядковими елементами iterable буде розміщений рядок str. Якщо iterable містить принаймні одне нерядкове значення, то генерується виняток TypeError.

```
>>> '..'.join(['1', '2'])
'1..2'
>>> '..'.join('hello')
'h..e..l..l..o'
```

str.partition(sep). Повертає кортеж з трьома рядковими значеннями, які є частинами рядку str. Першим елементом є частина рядка str, що міститься до першого входження роздільника sep. Другим елементом є сам роздільник sep. Третім елементом є частина рядка str, що міститься після роздільника sep. Якщо роздільника sep в рядку str не знайдено, то першим елементом кортежу буде сам рядок str, а другий та третій елементи будуть порожніми рядками.

```
>>> 'habrahabr'.partition('ra')
('hab', 'ra', 'habr')
>>> 'habrahabr'.partition('a')
('h', 'a', 'brahabr')
>>> 'habrahabr'.partition('car')
('habrahabr', '', '')
```

str.rpartition(sep). Повертає кортеж з трьома рядковими значеннями, які є частинами рядку str. Першим елементом є частина рядка str, що міститься до останнього входження роздільника sep. Другим елементом є сам роздільник sep. Третім елементом є частина рядка str, що міститься після роздільника sep. Якщо роздільника sep в рядку str не

знайдено, то першим елементом кортежу буде сам рядок `str`, а другий та третій елементи будуть порожніми рядками.

```
>>> 'habrahabr'.rpartition('a')
('habrah', 'a', 'br')
```

`str.ljust(width, fillchar=" ")`. Повертає копію рядка `str`, доповненого справа символами `fillchar` до довжини `width`. Якщо довжина рядка більша або рівна `width`, повертає оригінальний рядок.

```
>>> 'habr'.ljust(7)
'habr   '
>>> 'habr'.ljust(7,'_')
'habr___'
```

`str.rjust(width, fillchar=" ")`. Повертає копію рядка `str` ,доповненого зліва символами `fillchar` до довжини `width`. Якщо довжина рядка більша або рівна `width`, повертає оригінальний рядок.

```
>>> 'habr'.rjust(7)
'   habr'
>>> 'habr'.rjust(7,'_')
'___habr'
```

`str.center(width[, fillchar])`. Повертає копію рядка `str`, доповненого зліва та справа символами `fillchar` до довжини `width`, таким чином, щоб рядок був вирівняний за центром результуючого рядка. Якщо довжина рядка більша або рівна `width`, повертає оригінальний рядок.

```
>>> 'habr'.center(8,'_')
'__habr__'
>>> 'habr'.center(7,'_')
'__habr_'
```

`str.zfill(width)`. Повертає копію рядка `str`, доповненого символами «0» до довжини `width`. Якщо рядок починається зі знаку («+» або «-»), то доповнюючи символи вставляються після знаку. Якщо довжина рядка більша або рівна `width`, повертає оригінальний рядок.

```
>>> '9'.zfill(4)
'0009'
>>> '-9'.zfill(4)
```

```
'-009'
>>> '+9'.zfill(4)
'+009'
```

str.format(*args, **kwargs). Повертає копію рядка str, відформатованого відповідним чином. Детальні відомості щодо цього методу викладено в додатку З

7.4.7. Приклади розв'язування задач

Приклад. Написати програму за якою буде визначатися, чи є задане користувачем слово паліндромом (паліндром – слово, що читається однаково зліва направо і навпаки, наприклад, "шалаш").

```
s=input('Введіть рядок: ')
if s==s[::-1]:
    print('Паліндром')
else:
    print('Не паліндром')
```

Приклад. Дано рядок, який складається із слів розділених пропусками. Написати програму за якою буде обраховуватися кількість слів в рядку.

Якщо припустити, що пропуски містяться лише між словами їх там строго по одному, то розвязок задачі може мати вигляд:

```
s=input('Введіть рядок: ')
n=s.count(' ')+1
print(n)
```

Якщо ж пропусків між словами може бути скільки завгодно, то розвязок задачі може мати вигляд:

```
s=input('Введіть рядок: ')
a=list(s.split(' '))
b=[]
for i in a:
    if i!='' :
        b=b+[i]
print(len(b))
```

Останій цикл можна переписати в вигляді генератору спсику:

```
b =[i for i in a if i!='']
```

Приклад. Дано рядок. Написати програму за якою буде відбуватися поділ рядка на дві рівні частини (якщо довжина рядка непарна то перша частина має бути на 1 символ меншою) та виведення їх окремими рядками. При розв'язання задачі не використовувати інфструкцію if.

```
s=input('Введіть рядок: ')
print(s[:len(s)//2])
print(s[len(s)//2:])
```

Приклад. Дано рядок, який складається із слів розділених пропусками та розділовими знаками (,.!?). Написати програму за якою буде виведено всі слова окремими рядками.

```
s=input('Введіть рядок: ')
a=list(s.split(' '))
for i in a:
    if i!='':
        print(i.strip(',.!?'))
```

Останій цикл можна переписати в вигляді генератору спсику:

```
[print(i.strip(',.!?')) for i in a if i!='']
```

Приклад. Дано рядок, який складається із слів розділених пропусками та розділовими знаками (,.!?). Написати програму за якою буде виведено всі слова парної довжини окремими рядками.

```
s=input('Введіть рядок: ')
a=list(s.split(' '))
a=[i.strip(',.!?') for i in a if i!='']
[print(i) for i in a if len(i)%2==0]
```

Приклад. Дано рядок, який складається із слів розділених пропусками та розділовими знаками (,.!?). Написати програму за якою буде виведено всі слова що не містять повторюваних символів.

```
s=input('Введіть рядок: ')
a=list(s.split(' '))
a=[i.strip(',.!?') for i in a if i!='']
for i in a:
    flag=True
    for j in i:
```