даних, за необхідності можна перетворити його в список, внести необхідні зміни і знову повернути в кортеж.

```
>>> b=(1,2,3)
>>> b=list(b)
>>> b[1]=5
>>> b=tuple(b)
>>> print(b)
(1, 5, 3)
```

Методи кортежів

Враховуючи незмінюваність кортежів, вони мають лише методи: `tuple.index(x[, start[, end]])` та `tuple.count(x)` призначення яких аналогічне до призначенням однойменних методів списків.

7.3. Словники

Звичайні списки являють собою набір пронумерованих елементів, отже, для звернення до деякого елемента списку необхідно вказати його номер (індекс). Номер елемента в списку однозначно ідентифікує сам елемент. Проте досить часто ідентифікувати дані лише за числовими індексами не завжди зручно.

Словник (dict) – це структура даних, призначена для зберігання довільних об'єктів з доступом за довільним ключем. Дані в словнику зберігаються в форматі `ключ=значення`. Ключі в межах словника мають бути унікальними, тобто двох однакових ключів в словнику бути не може. Ключ повинен мати незмінюваний тип даних: ціле або дійсне число, рядкок, кортеж. Враховуючи зазначену структуру, словник інколи ще називають асоціативним масивом.

Словник записується, як перелік пар ключ : значення, розділених комою та взятих в фігурні дужки: `{'A1':2, 'A2':3}`.

Словник є змінюваним типом даних: в нього можна додавати нові елементи з довільними ключами і вилучати вже існуючі елементи. При цьому розмір використовуваної пам’яті пропорційний розміру словника. Доступ до елементів словника виконується хоча і повільніше, ніж до звичайного списку, але в цілому достатньо швидко.

7.3.1. Створення словників

Для задання порожнього словника можна скористатися однією з наступних команд:

```
>>> a={}
>>> b=dict()
```

Задання словника з наперед заданим набором елементів:

```
>>> a={'A1':2, 'A2':3}
>>> b=dict(id1=4, id2=8)
>>> print(b)
{'id1': 4, 'id2': 8}
```

Варто відмітити, що у випадку створення словника, в якого ключем має виступати кортеж чи ключ має починатися з цифри, можливе використання лише першої команди:

```
>>> a={1: 'A1', (1,2,3):'A2'}
>>> a=dict(1: 'A1', 2:'A2')
SyntaxError: invalid syntax
>>> a=dict((1,2,3): 'A1', (4,6):'A2')
SyntaxError: invalid syntax
```

7.3.2. Виконання дій над елементами словника

В цьому пункті буде розглянуте виконання дій над словниками та іхніми елементами без застосування методів словників, які подані в наступному пункті.

Доступ до значень словника

Враховуючи те, що елементом словника є пара ключ=значення, то, як такого, доступу до елемента словник не має. В словнику передбачена можливість доступу до значення елемента словника за його ключем.

Для того, щоб звернутися до значення елемента словника, необхідно вказати ім'я змінної словника та в квадратних дужках ключ необхідного елемента (`ім'я_словника[ключ]`).

```
>>> e={'A1':2, 'A2':3}
>>> print(e['A2'])
3
```

При спробі доступу за неіснуючим у словнику ключем виникає виняток KeyError.

```
>>> print(e['A3'])
Traceback (most recent call last):
  File "<pyshell#99>", line 1, in <module>
    print(e['A3'])
KeyError: 'A3'
```

Можна перевірити приналежність деякого ключа до словника, використовуючи оператор in (`ключ in ім'я_словника`).

```
>>> e={'A1':2, 'A2':3}
>>> 'A1' in e
True
>>> 'A3' in e
False
```

Також можна перевірити наявність певного значення в словнику, проте така перевірка буде відбуватися за результатом методу `values()`.

```
>>> e={'A1':2, 'A2':3}
>>> 2 in e.values()
True
```

За необхідності можна організувати перебір ключів усіх елементів словника.

```
d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
for k in d:
    print(k, d[k])
```

Зміна значень словника

Враховуючи, що словник є змінюваним типом даних, за необхідності можна змінювати значення його елементів. Для зміни значення елемента словника необхідно вказати ім'я змінної словника та в квадратних дужках ключ та виконати присвоєння нового значення (`ім'я_словника[ключ]=нове_значення`).

```
>>> e={'A1':2, 'A2':3}
>>> e['A1']=5
>>> print(e)
{'A1': 5, 'A2': 3}
```

Додавання елементів до словника

Щоб додати елемент до словника, потрібно вказати ім'я змінної словника та в квадратних дужках новий ключ та виконати присвоєння нового значення (`ім'я_словника[новий_ключ]=значення`).

```
>>> e={'A1':2, 'A2':3}
>>> e['A3']=4
>>> print(e)
{'A1': 2, 'A2': 3, 'A3': 4}
```

Вилучення елемента зі словника

Для видалення елемента зі словника можна скористатися командою `del (del ім'я_словника[ключ])`.

```
>>> e={'A1':2, 'A2':3}
>>> del e['A1']
>>> print(e)
{'A2': 3}
```

7.3.3. Методи словників

`dict.fromkeys(iterable [, value])`. Створює новий словник, ключами якого будуть елементи з `iterable` і однаковим для всіх значенням `value`.

```
>>> d.fromkeys(['a', 'b', 'c'],12)
{ 'a': 12, 'b': 12, 'c': 12}}
```

`dict.update([other])`. Доповнює словник `dict` парами (ключ=значення) з словника `other`, якщо ключ вже присутній в словнику, то його значення оновлюється.

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> s={'A1':13, 'A5':15}
>>> d.update(s)
>>> print(d)
{'A1': 13, 'A2': 3, 'A3': 5, 'A4': 7, 'A5': 15}
```

`dict.copy()`. Повертає копію словника `dict`.

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> e=d.copy()
```

```
>>> print(e)
{'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
```

dict.get(key[, default]). Повертає значення зі словника dict за ключем key. У випадку відсутності елемента з ключем key повертається значення default.

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> d.get('A2', 10)
3
>>> d.get('A8', 10)
10
```

dict.setdefault(key[, default]). Повертає значення зі словника dict за ключем key. У випадку відсутності елемента з ключем key повертається значення default, і до словника додається елемент з ключем key і значенням default.

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> d.setdefault('A1', 13)
1
>>> d.setdefault('A5', 13)
13
>>> print(d)
{'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7, 'A5': 13}
```

dict.keys(). Повертає ключі елементів словника dict у вигляді об'єкту перегляду словника, що забезпечують динамічний перегляд записів словника.

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> d.keys()
dict_keys(['A1', 'A2', 'A3', 'A4'])
```

dict.values(). Повертає значення елементів словника dict у вигляді об'єкту перегляду словника, що забезпечують динамічний перегляд записів словника.

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> d.values()
dict_values([1, 3, 5, 7])
```

dict.items(). Повертає ключі та значення елементів словника dict у вигляді об'єкту перегляду словника, що забезпечують динамічний перегляд записів словника. Елементи словника подаються в вигляді кортежів (ключ, значення).

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5}
>>> d.items()
dict_items([('A1', 1), ('A2', 3), ('A3', 5)])
```

dict.pop(key[, default]). Вилучає зі словника dict елемент з ключем key та повертає його значення, як результат виконання функції. У випадку відсутності елемента з ключем key повертається значення default. Якщо default не вказаний, і елемент з ключем key відсутній, то виникає виняток KeyError.

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> d.pop('A2')
3
>>> print(d)
{ 'A1': 1, 'A3': 5, 'A4': 7}
```

dict.popitem(). Вилучає і повертає пару (ключ, значення) зі словника dict, як результат виконання функції. Пари повертаються в порядку LIFO (last-in first-out). Якщо словник порожній, то виникає виняток KeyError.

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> print(d)
{'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> d.popitem()
('A4', 7)
>>> print(d)
{'A1': 1, 'A2': 3, 'A3': 5}
```

dict.clear(). Очищує словник dict (вилучає всі елементи зі словнику).

```
>>> d={'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> print(d)
{'A1': 1, 'A2': 3, 'A3': 5, 'A4': 7}
>>> d.clear()
```

```
>>> 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