Також неуважність при описі рекурсивної функції може привести до нескінченості рекурсії, коли ланцюжок викликів функцій ніколи не завершується і триває, поки не скінчиться вільна пам'ять в комп'ютері. Тому необхідністю для працездатності рекурсивних функцій є наявність правильно оформленої умови закінчення рекурсивних викликів (наприклад, перевірка значення параметра, що змінюється).

8.6. Приклади розв'язування задач

Приклад. Написати програму, за якою буде обчислюватися площа трикутника, заданого довжинами його сторін.

```
import math
def geron(a,b,c):
    p=(a+b+c)/2
    return math.sqrt(p*(p-a)*(p-b)*(p-c))
print('Введіть довжини сторін трикутника:')
a=float(input())
b=float(input())
c=float(input())
if a+b>c and a+c>b and b+c>a:
    s=geron(a,b,c)
    print('Площа трикутника =', s)
else:
    print('Трикутник з даними сторонами не існує.')
```

Приклад. Обчислити значення виразу

f(x)=x2+x+3+ex2+x+3+sin2(x2+x+3) x=1,2,…,10.

```
import math
def y(x):
    return x*x+x+3
for i in range(1,11):
    f=y(i)+ math.pow(math.e,y(i)) +
     math.pow(math.sin(y(i)),2)
    print('f({})={}'.format(i,f))
```

Приклад. Знайти довжину вектора, заданого своїми координатами у n–вимірному просторі.

```
import math
def vLength(v, n):
    s=0
    for i in range(n):
        s+=v[i]**2;
    return math.sqrt(s)
n=int(input('Введіть кількість елементів списку = '))
a=[]
for i in range(1,n+1):
    x=int(input('a[{}]='.format(i)))
    a.append(x)
print('Довжина вектора a=',vLength(a,n));
```

Приклад. Дано a,b,c – довжини сторін деякого трикутника. Знайти медіани трикутника.

Довжина медіани, проведеної до сторони a, дорівнює

$$m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$$

```
import math
def median(x,y,z):
    return 0.5*math.sqrt(2*x*x+2*y*y-z*z)
print('Введіть довжини сторін трикутника:')
a=float(input())
b=float(input())
c=float(input())
if a+b>c and a+c>b and b+c>a:
    m1=median(a,b,c)
    m2=median(a,c,b)
    m3=median(c,b,a)
    print('Медіана 1 =', m1)
    print('Медіана 2 =', m2)
    print('Медіана 3 =', m3)
else:
    print('Трикутник з даними сторонами не існує.')
```

Приклад. Дано цілочисельну матрицю випадкових цілих чисел розміром m×n (m і n задається користувачем). Знайти мінімальний елемент матриці. Описати функції генерування матриці, виведення матриці та функцію пошуку мінімального елемента.

```
import random
def gener_matr(n,m):
    a=[]
    for i in range(n):
        b=[]
        for j in range(m):
            x=random.randint(10,99)
            b.append(x)
        a.append(b)
    return a
def print_matr(a):
    print('Згенерована матриця:')
    for row in a:
        for elem in row:
            print(elem, end=' ')
        print()
def min_matr(a):
    min_col=list(map(min, a))
    min_el=min(min_col)
    return min_el

n=int(input('Введіть кількість рядків матриці: '))
m=int(input('Введіть кількість стовпців матриці: '))
a=gener_matr(n,m)
print_matr(a)
print('Мінімальний елемент: ',min_matr(a))
```

9. Файли

Досить часто при створенні програм виникає необхідність у використанні даних, які записані на зовнішніх носіях (дисках) і оформлені у

вигляді файлів даних. Файл може бути джерелом інформації – тоді відбувається читання з файла, або приймачем – тоді відбувається запис даних у файл [7].

Для роботи з файлом необхідно пов'язати його логічне позначення (файлову змінну) з фізичним файлом (файл, який зберігається на диску).

9.1. Відкриття та закриття файлу

Для відкриття файлу використовується функція **`open()`**, за якою повертається файловий об'єкт. Виклик функції матиме вигляд:

```
file=open()
```

В результаті змінна `file` буде містити файловий об'єкт, який пов'язаний з відповідним файлом на диску (ім'я якого та режими відкриття вказуються параметрами функції). Для виконання операцій з цим файлом необхідно буде викликати методи для змінної `file`.

Повний формат цієї функції має наступний вигляд

`open(fname, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)`

`fname` - цілочисельний файловий дескриптор або абсолютний чи відносний шлях до файлу, який потрібно відкрити.

`mode` - рядковий параметр за яким вказується режим відкриття файлу. Файл може бути відкритий в таких режимах: `'r'` - відкрити файл для читання (є значенням за замовчуванням); `'w'` - відкрити файл для запису. Якщо файлу не існує, то створюється новий файл, якщо файл існує, то вміст файлу вилучається; `'x'` - відкрити файл для запису. Якщо файлу не існує, то створюється новий, якщо файл існує, то виникає виняток `FileExistsError`; `'a'` - відкрити файл для до запису, який відбувається в кінець файлу; `'+'` – відкрити файл для читання та для запису, який відбувається в кінець файлу (має суттєве значення лише в поєднанні з `'r'`, тобто `'r+'`, проте може використовується в поєднанні і з іншими режимами)

Для будь-яких з перерахованих вище режимів додатково може бути вказаний символ `'b'` або `'t'` (є значенням за замовчуванням), за яким буде розрізнятися двійковий та текстовий ввід/вивід (наприклад, `'rb'`, `'wt'` і т. д.). Файл, відкритий в двійковому режимі, повертає вміст у вигляді об'єктів

байт без будь-якого декодування В текстовому режимі – вміст файлу повертається як `str`.

`buffering` – параметр, за яким вказується політика буферизації при роботі з файлом. Може набувати таких значень: `0` - використовується лише для двійкового режиму і відключає буферизацію; `1` - використовується лише для текстового режиму і встановлює буферизацію рядка; `ціле додатне число` - визначає розмір буферу в байтах..

`encoding` – параметр, за яким вказується кодування файлу (використовується лише в текстовому режимі). Кодування за замовчуванням залежить від платформи. Поточне кодування можна отримати за методом `locale.getpreferredencoding()`.

`errors` – параметр, за яким вказується метод опрацювання помилок при кодуванні та декодуванні файлу (використовується лише в текстовому режимі). Можливі значення: `None, 'strict', 'ignore', 'replace', 'surrogateescape', 'xmlcharrefreplace', 'backslashreplace', 'namereplace'` та інші.

`newline` – параметр, за яким вказується опрацювання вказівника закінчення рядка (використовується лише для текстових файлів). Можливі значення: `None, '', '\n', '\r', '\r\n'`. *Читання з файлу*: при значенні `newline=None` після читання вказівником закінчення рядка буде встановлений `'\n'`; при інших значеннях параметра — вказівник закінчення рядка залишиться без змін. *Запис до файлу*: при значенні `newline=None` в кінці кожного рядка буде встановлений вказівник закінчення рядка за замовчуванням системи (`os.linesep`); якщо `newline=''` або `newline='\n',` то вказівником закінчення рядка буде `'\n'`; інші значення параметра явно вказують вказівник закінчення рядка.

`closefd` – параметр, за яким вказується необхідність закриття файлового дескриптора (використовується лише для файлів, вказаних своїм дескриптором). При значенні параметра `closefd=False` файловий дескриптор залишиться відкритий навіть після явного закриття файлу, інакше файловий дескриптор закривається разом з закриттям файлу.

`opener` – параметр, за яким може бути вказаний користувацький метод відкриття файлу.

Проте на практиці найчастіше використовуються параметри `fname` та `mode`.

Після роботи з файлом його обов'язково необхідно закрити, для цього передбачений метод `close()`.

9.2. Атрибути файлового об'єкта

У файлового об'єкта є ряд атрибутів, які можна отримати навіть після закриття файлу.

`file.closed` – містить `True`, якщо файл закритий і `False`, якщо файл відкритий.

`file.mode` – містить режим доступу до файлу

`file.name` – містить ім'я файлу

```
>>> f=open("test.txt", "r")
>>> print("file.closed: " + str(f.closed))
file.closed: False
>>> print("file.mode: " + f.mode)
file.mode: r
>>> print("file.name: " + f.name)
file.name: test.txt
>>> f.close()
>>> print("file.closed: " + str (f.closed))
file.closed: True
```

Також для файлового об'єкта передбачені два методи, за якими визначається можливість запису даних до файлу (**`file.readable()`**) чи можливість зчитування даних з файлу (**`file.writable()`**). Результатом цих методів є значення логічного типу.

9.3. Читання з файлу

Для зручності будемо вважати, що на диску вже є текстовий файл `test.txt`, який має наступну структуру:

```
1 2 3 4 5
Work with file
```

Для зчитування даних з файлу передбачено кілька методів.

file.read(size). Зчитує з файлу file size символів. Якщо параметр size не заданий чи його значення є від'ємним, то зчитується весь файл. Якщо досягнуто кінець файлу, то за методом file.read() буде повернуто порожній рядок ".

```
>>> f=open("test.txt", "r")
>>> f.read()
'1 2 3 4 5\nWork with file\n'
>>> f.close()
>>> f=open("test.txt", "r")
>>> f.read(5)
'1 2 3'
>>> f.close()
```

За методом **file.readline()** зчитується один рядок з файлу file. Символ закінчення рядка "\n" залишається в кінці рядка і відсутній лише в останньому рядку, якщо файл не закінчується порожнім рядком. Це робить однозначним значення, отримане за методом readline(); якщо отримано порожній рядок, то було досягнуто кінець файлу, порожній же рядок подається символом '\ n'.

```
>>> f=open("test.txt", "r")
>>> f.readline()
'1 2 3 4 5\n'
>>> f.close()
```

Порядкове читання з файлу можна виконати, використовуючи оператор for, тобто можна організувати перебір всіх елементів з файлу.

```
>>> f=open("test.txt", "r")
>>> for line in f: print(line, end='')
1 2 3 4 5
Work with file
>>> f.close()
```

За необхідності прочитати всі рядки в список, можна скористатися методами: **list(file)** або **file.readlines()**

```
>>> f=open("test.txt", "r")
>>> a=list(f)
>>> a
```

```
['1 2 3 4 5\n', 'Work with file']
>>> f.close()
>>> f=open("test.txt", "r")
>>> b=f.readlines()
>>> b
['1 2 3 4 5\n', 'Work with file']
>>> f.close()
```

9.4. Запис у файл

Для запису даних у файл використовується метод **`file.write(string)`**.

При вдалому запису рядка до файлу за методом повертається кількість записаних символів.

```
>>> f=open("test.txt", "a")
>>> f.write("Test string")
11
>>> f.close()
```

Щоб записати щось відмінне від рядка, воно повинно бути приведене до рядка:

```
>>> f=open("test.txt", "a")
>>> f.write(str(2019))
4
>>> f.close()
```

Варто відмітити, що за методом `file.write(string)` відбувається запис відповідних рядкових даних у файл, і ніякого додавання вказівника кінця рядка не відбувається. Тобто всі дані заносяться послідовно, без розбиття на рядки, якщо це явно не вказується.

Якщо поглянути на вміст файлу `"test.txt"`, то можна побачити наступне:

```
1 2 3 4 5
Work with file2019
```

Запис даних до файлу можна також організувати з використанням методу `print()`. Як зазначалося раніше, цей метод має необов'язковий

параметр `file`, який за замовчуванням рівний `sys.stdout` (стандартний пристрій виведення - екран).

Якщо ж вказати значенням для цього параметра файлову змінну, то за методом `print()` буде відбуватися виведення даних не на екран, а в файл (запис даних до файлу).

```
>>> f=open("test.txt", "a")
>>> print("Test string1",  file=f)
>>> print("Test string2",  file=f)
>>> f.close()
```

На відміну від методу `file.write(string)`, за методом `print()` крім запису власне рядкових даних відбувається і дописування вказівника кінця рядка.

Якщо поглянути на вміст файлу `"test.txt"`, то можна побачити наступне:

```
1 2 3 4 5
Work with file2019Test string1
Test string2
```

9.5 Додаткові методи роботи з файлами

`file.tell()`. Повертає ціле число, яке відповідає позиції файлового покажчика («умовного курсору») в файлі. Визначається в байтах (символах).

Наприклад, якщо прочитати перші пять символів, то файловий покажчик буде встановлений на позиції 5.

```
>>> f=open("test.txt", "r")
>>> f.read(5)
'1 2 3'
>>> f.tell()
5
>>> f.close()
```

`file.seek(offset, from_what=0).` Встановлює файловий покажчик в нову позицію, яка обраховується додаванням зміщення `offset` до точки відліку заданої параметром `from_what`. При `from_what=0` –

точкою відліку є початок файлу; при from_what=1 – точкою відліку є поточна позиція файлового покажчика; при from_what=2 - точкою відліку є кінець файлу. Значення параметра from_what, відмінне від 0, використовується лише для бінарних файлів.

```
>>> f=open("test.txt", "rb")
>>> f.seek(4)
>>> f.read(1)
b'3'
>>> f.seek(1, 1)
>>> f.read(1)
b'4'
>>> f.close()
```

9.6. Використання менеджера контексту

Для деяких об'єктів визначені стандартні завершальні дії, які повинні бути виконанні після завершення роботи з цим об'єктом. Причому такі завершальні дії повинні виконуватися незалежно від того, чи пройшли операції з об'єктом успішно, чи виникла помилка. Звичайно, можна скористатися блоком опрацювання виняткових ситуацій, проте в мові Python передбачений менеджер контексту with as.

Розглянемо можливості використання менеджера контексту при роботі з файлами. Послідовність роботи з файлом складається з таких кроків: відкрити файл, опрацювати (прочитати чи записати) дані файлу, закрити файл. Зрозуміло, що при опрацюванні файлу можуть виникнути нештатні ситуації, які призведуть до завершення програми, окрім того, обов’язково слід потурбуватися про закриття файлу. Використання оператору with as при роботі з файлами буде забезпечувати виконання завершальних дій без явного їх вказування, тобто закриття файлу.

```
with open("test.txt") as f:
    for line in f:
        print(line, end="")
```

Після завершення виконання зазначеного коду файл f завжди закривається, навіть якщо виникла проблема при обробці файлу. Отже, наведений код не потребує виклику оператора закриття файлу.

9.7. Приклади розв'язування задач

Приклад. Написати програму створення текстового файлу з рядків заданих користувачем.

```
fn=input("Введіть ім'я файлу: ")
f=open(fn,'w')
print('Введіть рядки для запису в файл.')
print('Для завершення введіть "end." без лапок.')
s=input()
while s!='end.':
    f.write(s+'\n')
    s=input()
f.close()
```

Приклад. Дано текстовий файл, який містить дані записані окремими рядками. Вивести на екран всі рядки файлу згрупувавши спочатку рядки парної довжини, а потім рядки непарної довжии.

```
fn=input("Введіть ім'я файлу: ")
with open(fn,'r') as f:
    print('Рядки парної довжини:')
    for i in f:
        if (len(i)-1)%2==0: print(i,end='')
    print('Рядки не парної довжини:')
    f.seek(0)
    for i in f:
        if (len(i)-1)%2==1: print(i,end='')
```

Приклад. Дано текстовий файл кожен рядок якого містить координату точок в декартовій системі координат. Координата кожної точки записана двома дійсними числа через пропуск. Вивести на екран координату точки максимально віддаленої від початку координат та відстань до неї.

```
dist=0
dist_x=0
```

```
dist_y=0
import math as m
fn=input("Введіть ім'я файлу: ")
with open(fn, 'r') as f:
    for i in f:
        x, y=i.split()
        x, y=float(x), float(y)
        tmp=m.sqrt(x**2+y**2)
        if tmp>dist:
            dist=tmp
            dist_x=x
            dist_y=y
print('На максимальній відстані ({:.3}) від початку
     координат знаходиться точка з координатами ({},
     {})'.format(dist,dist_x,dist_y))
```

Приклад. Дано цілочисельну матрицю випадкових цілих чисел розміром n×n (n задаються користувачем). Написати програму для запису даної матрицю до файлу в «природному» вигляді (кожен окремий рядок файлу має містити елементи відповідного рядка матриці записані через пропуск).

```
import random as rnd
n=int(input('Вкажіть розмірність матриці:'))
m=[]
for i in range(n):
    a=[]
    for j in range(n):
        x=rnd.randint(1,9)
        a.append(x)
    m.append(a)
fn=input("Введіть ім'я файлу: ")
with open(fn,'w') as f:
    for i in m:
        for j in i:
            print(str(j), sep='', end=' ',file=f)
```

```
            print('',file=f)
```

Приклад. Дано текстовий файл який містить дані про матрицю записану в «природному» вигляді (кожен окремий рядок файлу має містити елементи відповідного рядка матриці записані через пропуск). Вичитати матрицю з файлу та знайти мінімальний елемент матриці.

```
fn=input("Введіть ім'я файлу: ")
m=[]
with open(fn,'r') as f:
    a=list(f)
    for i in a:
        b=list(map(int, i.split()))
        m.append(b)
min_col=list(map(min, m))
min_el=min(min_col)
print(min_el)
```

Вичитати матрицю з файлу можна і наступними операторами.

```
m=[]
with open('4.txt','r') as f:
    m=[list(map(int, i.split())) for i in list(f)]
```