Тема 7. Модуль math, модуль fractions, модуль cmath, модуль struct, модуль shelve (*.ini)

I. Модуль math

Вбудований модуль math у Python надає набір функцій для виконання математичних, тригонометричних та логарифмічних операцій. Деякі з основних функцій модуля:

- **pow(num, power)**: зведення числа num у ступінь power
- **sqrt(num)**: квадратний корінь числа num
- **ceil(num)**: округлення числа до найближчого найбільшого цілого
- **floor(num)**: округлення числа до найближчого найменшого цілого
- **factorial(num)**: факторіал числа
- **degrees(rad)**: переведення з радіан у градуси
- **radians(grad)**: переведення з градусів у радіани
- **cos(rad**): косинус кута в радіанах.
- **sin(rad)**: синус кута в радіанах
- **tan(rad)**: тангенс кута в радіанах
- **acos(rad)**: арккосинус кута в радіанах
- **asin(rad)**: арксинус кута в радіанах
- **atan(rad)**: арктангенс кута в радіанах
- **log(n, base)**: логарифм числа n на основі base
- **log10(n)**: десятковий логарифм числа n
-

Приклад застосування деяких функцій:

```
Import math

# Зведення числа 2 у ступінь 3
n1 = math.pow(2, 3)
print(n1) # 8

ту ж саму операцію можна виконати так
n2 = 2**3
print(n2)

# Зведення в квадрат
print(math.sqrt(9)) # 3

# найближче найбільше ціле число
print(math.ceil(4.56)) # 5

найближче найменше ціле число
```

```
print(math.floor(4.56)) # 4

# Переведення з радіан в градуси
print(math.degrees(3.14159)) # 180

# Переведення з градусів в радіани
print(math.radians(180)) # 3.1415.....
# косинус
print(math.cos(math.radians(60))) # 0.5
# Синус
print(math.sin(math.radians(90))) # 1.0
# тангенс
print(math.tan(math.radians(0))) # 0.0

print(math.log(8,2)) # 3.0
print(math.log10(100)) # 2.0
```

Також модуль math надає ряд вбудованих констант, такі як, PI та E:

```
import math
radius = 30
площа кола з радіусом 30
area = math.pi * math.pow (radius, 2)
print(area)

# натуральний логарифм числа 10
number = math.log(10, math.e)
print(number)
```

II. Модуль ractions - раціональні числа (звичайні дроби)

Модуль fractions дозволяє виконувати арифметичні дії над раціональними числами, що у деяких ситуаціях буває надзвичайно зручно.

Створення звичайних дробів

Способів створення екземплярів раціональних чисел досить багато тому ми розділимо їх на групи.

Найпростіший спосіб створити звичайний дріб – вказати чисельник (numerator) та знаменник (denominator):

```
>>> з fractions import Fraction
>>>
>>> Fraction() # за замовчуванням numerator=0,
denominator=1
Fraction(0, 1)
>>>
```

```
>>> Fraction(numerator=1, denominator=2) # рівносильно
Fraction(1, 2)
Fraction(1, 2)
>>>
>>> Fraction(1, 2)
Fraction(1, 2)
```

Якщо вказані чисельник і знаменник мають спільні дільники, перед створенням раціонального числа вони скорочені:

```
>>> Fraction(8, 16), Fraction(15, 30)
(Fraction(1, 2), Fraction(1, 2))
```

Як чисельник і (або) знаменник можуть бути зазначені інші дроби:

```
>>> Fraction(3, Fraction(1, 2))
Fraction(6, 1)
>>>
>>> Fraction(Fraction(1, 2), 3)
Fraction(1, 6)
>>>
>>> Fraction(Fraction(1, 2), Fraction(3, 7))
Fraction(7, 6)
```

Ціле і речове число, так само можна перетворити на звичайний дріб:

```
>>> Fraction(10)
Fraction(10, 1)
>>>
>>> Fraction(11.11)
Fraction(781796747813847, 70368744177664)
>>>
>>> Fraction(1.25)
Fraction(5, 4)
>>>
>>> Fraction(2e-10)
Fraction(7737125245533627, 38685626227668133590597632)
```

А ось комплексні числа призведуть до помилки:

```
>>> Fraction(1j, 1 + 2j)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: both arguments повинні бути Rational instances
```

Але уявною і дійсною частиною комплексного числа можуть бути вказані раціональні числа:

```
>>> complex(Fraction(3, 5), Fraction(4, 7))
(0.6+0.5714285714285714j)
```

Так само будь-який десятковий дріб модуля Decimal може бути перетворений на звичайний дріб:

```
>>> з import decimal Decimal
>>> Fraction(Decimal('7.7'))
Fraction(77, 10)
```

Будь-який рядок, який може бути перетворений на будь-яке допустиме число, також може бути використаний для отримання звичайних дробів:

```
>>> Fraction('111')
Fraction(111, 1)
>>>
>>> Fraction('1.11')
Fraction(111, 100)
>>>
>>> Fraction('1.11e+10')
Fraction(11100000000, 1)
>>>
>>> Fraction('-17/63')
Fraction(-17, 63)
>>>
>>> Fraction('-0.115')
Fraction(-23, 200)
>>>
>>> Fraction('-.115')
Fraction(-23, 200)
>>>
>>> Fraction('1.11 \t\n')
Fraction(111, 100)
>>> Fraction('\t\n 1.11 \t\n')
Fraction(111, 100)
>>> Fraction('\t\n 1.11')
Fraction(111, 100)
>>>
>>> Fraction('\t\n -13/37')
Fraction(-13, 37)
```

Математичні операції над раціональними числами

Усі арифметичні оператори підтримують обчислення з раціональними числами:

```
>>> x = Fraction(2, 5)
>>> y = Fraction(3, 7)
```

```
>>>
>>> x + y, y - x
(Fraction(29, 35), Fraction(1, 35))
>>>
>>> x*y, x/y
(Fraction(6, 35), Fraction(14, 15))
>>>
>>> x**0.5, 2**0.5/5**0.5
(0.6324555320336759, 0.6324555320336759)
>>>
>>> x**y, (x**3)**(1/7)
(0.6752339686501552, 0.6752339686501552)
>>>
>>> y//x
1
>>> x%y
Fraction(2, 5)
```

Однак арифметичні оператори не здатні до дій над типами 'Fraction' та 'decimal.Decimal' в одному виразі:

```
>>> x + 1.1
1.5
>>>
>>> x + Decimal('1.1') # призведе до помилки
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'Fraction'
and 'decimal.Decimal'
```

Функції модуля math здатні приймати раціональні числа як аргументи, так як останні, по суті, можуть бути просто перетворені до речовинних чисел:

```
>>> import math
>>>
>>> x = Fraction(4, 11)
>>>
>>> math.exp(x), math.exp(4/11)
(1.438551009577678, 1.438551009577678)
```

Fraction - атрибути та методи

`x.numerator`

повертає чисельник дробу:

```
>>> з fractions import Fraction
>>>
```

```
>>> x = Fraction(3, 8)
>>> x
Fraction(3, 8)
>>>
>>> x.numerator
3
```

x.denominator

повертає знаменник дробу:

```
>>> x.denominator
8
```

Fraction.from_float(flt)

приймає flt – число типу float і повертає звичайний дріб ставлення чисельника до знаменника якого максимально наближається до значення flt:

```
>>> з fractions import Fraction
>>>
>>> Fraction.from_float(0.5)
Fraction(1, 2)
>>>
>>> Fraction.from_float(0.6)
Fraction(5404319552844595, 9007199254740992)
```

Як можна помітити, через зберігання чисел з плаваючою точкою в двійковому поданні Fraction.from_float(0.6) зовсім не одно Fraction(6, 10):

```
>>> Fraction.from_float(0.6) == Fraction(6, 10)
False
```

Fraction.from_decimal(dec)

створює звичайний дріб, який є точним уявленням десяткового дробу зазначеного в dec, де dec – це екземпляр класу decimal.

```
>>> з import decimal Decimal
>>>
>>> Fraction.from_decimal(Decimal('0.7'))
Fraction(7, 10)
>>>
>>> Fraction.from_decimal(Decimal('0.77'))
Fraction(77, 100)
>>>
>>> Fraction.from_decimal(Decimal('0.125'))
Fraction(1, 8)
```

Fraction.limit_denominator(max_denominator=1000000)

повертає найближче раціональне уявлення вказаного числа, знаменник якого не перевищує значення max_denominator.

```
>>> import math
>>> math.pi
3.141592653589793
>>>
>>> Fraction(math.pi).limit_denominator(10)
Fraction(22, 7)
>>>
>>> Fraction(math.pi).limit_denominator(1000)
Fraction(355, 113)
>>>
>>> Fraction(math.pi).limit_denominator(100000)
Fraction(312689, 99532)
```

Даний метод дозволяє отримувати як раціональні наближення, так і відновлювати раціональні значення за їх неточним поданням у вигляді чисел з плаваючою точкою:

```
>>> math.cos(math.pi/3) # точне значення 0.5
0.5000000000000001
>>>
>>> Fraction(math.cos(math.pi/3)).limit_denominator()
Fraction(1, 2)
>>>
>>>
>>> 2**(1/3)
1.2599210498948732
>>>
>>> Fraction(2**(1/3)).limit_denominator()
Fraction(1054215, 836731)
>>>
>>> 1054215/836731
1.2599210498953666
```

x.__floor__()

повертає значення типу int, яке є найбільшим серед усіх int <= x:

```
>>> Fraction(234, 47).__floor__()
4
```

Та й з огляду на те, що всі функції модуля math можуть обробляти прості дроби, то скористатися даним методом можна і так:

```
>>> math.floor(Fraction(234, 47))
4
```

x.__ceil__()

повертає значення типу int, яке є найменшим серед усіх int >= x:

```
>>> Fraction(234, 47).__ceil__()
```

```
5
>>>
>>> import math
>>>
>>> math.ceil(Fraction(234, 47))
5
```

x.__round__(ndigits=None)

Округлює до найближчого парного числа.

Якщо ndigits=None, округлення виконується до найближчого парного цілого числа:

```
>>> Fraction('1/2').__round__()
0
>>>
>>> Fraction('3/2').__round__()
2
```

Якщо ndigits не дорівнює None, то округлення виконується до найближчого числа кратного дробу Fraction(1, 10**ndigits), при цьому якщо остання цифра 5 то округлення буде виконано до найближчої парної цифри:

```
>>> Fraction('7/3').__round__(2)
Fraction(233, 100)
>>>
>>> Fraction('7/3').__round__(3)
Fraction(2333, 1000)
>>>
>>>
>>> Fraction('555/1000').__round__(1)
Fraction(3, 5)
>>>
>>> Fraction('555/1000').__round__(2)
Fraction(14, 25)
```

Якщо ndigits не дорівнює None, але менше 0, то округлення виконується до розряду на який вказує abs(ndigits), при цьому якщо остання цифра 5 то округлення буде виконано до найближчої парної цифри:

```
>>> Fraction('-5555555/10').__round__(-1)
Fraction(-555560, 1)
>>> Fraction('-5555555/10').__round__(-3)
Fraction(-556000, 1)
>>>
>>> Fraction('77777/10').__round__(-3)
Fraction(8000, 1)
```

fractions.gcd(a, b)

повертає найбільший загальний дільник чисел a та b.

Повертає НОД(a, b). Знак результату залежить від знака b, якщо b не дорівнює 0, інакше знак результату дорівнює знаку a. Повертає 0, тільки якщо a та b обидва рівні 0:

```
>>> fractions.gcd(135, 15)
15
>>> fractions.gcd(135, -15)
-15
>>> fractions.gcd(-135, 15)
15
>>> fractions.gcd(-135, 0)
-135
>>> fractions.gcd(0, 0)
0
```

З версії 3.5. вважається застарілим, і краще використовувати команду math.gcd().

Приклад:

Обчислити $2:\frac{3}{5}+\frac{3}{5}:2+1\frac{1}{2}:6+6:1\frac{1}{2}$

```
від fractions import Fraction as dr
rez=dr(2,1)/dr(3,5)+dr(3,10)+dr(3,2)/6+6/dr(3,2)
print(rez, '=',end="")
a = int (rez.numerator / rez.denominator)
b = rez - a
print(a,"+", b, sep="")

>>> 473/60 = 7+53/60
```

III. Модуль cmath

Модуль cmath – надає функції до роботи з комплексними числами.

cmath.phase(x) – повертає фазу комплексного числа (її ще називають аргументом). Еквівалентно math.atan2(x.imag, x.real). Результат лежить у проміжку [-π, π].

Отримати модуль комплексного числа можна за допомогою вбудованої функції abs().

cmath.polar(x) – перетворення до полярних координат. Повертає пару (r, phi).

cmath.rect(r, phi) – перетворення з полярних координат.

cmath.exp(x) - ex.

cmath.log(x[, base]) - логарифм x на основі base. Якщо base не вказано, повертається натуральний логарифм.

cmath.log10(x) – десятковий логарифм.

cmath.sqrt(x) - квадратний корінь із x.
cmath.acos(x) – арккосинус x.
cmath.asin(x) – арксинус x.
cmath.atan(x) – арктангенс x.
cmath.cos(x) – косинус x.
cmath.sin(x) – синус x.
cmath.tan(x) – тангенс x.
cmath.acosh(x) – гіперболічний арккосинус x.
cmath.asinh(x) – гіперболічний арксинус x.
cmath.atanh(x) – гіперболічний арктангенс x.
cmath.cosh(x) – гіперболічний косинус x.
cmath.sinh(x) – гіперболічний синус x.
cmath.tanh(x) – гіперболічний тангенс x.
cmath.isfinite(x) - True, якщо дійсна та уявна частини кінцеві.
cmath.isinf(x) - True, якщо дійсна, або уявна частина нескінченна.
cmath.isnan(x) - True, якщо або дійсна, або уявна частина NaN.
cmath.pi- π.
cmath.e- e.

IV. Модуль struct -упаковка даних у бінарний файл

У деяких додатках доводиться мати справу з упакованими двійковими даними, які створюються, наприклад, програмами на мові C. Для їх збереження та відновлення можна скористатися модулем struct із стандартної бібліотеки Python.

Приклад запису у файл та читання з файлу:

```
import struct

myfile = open("data.bin", "wb")
# упаковуємо дані
bytes = struct.pack(b'>i4sh', 7, b'spam', 8)
myfile.write(bytes)
myfile.close()

myfile = open("data.bin", "rb")
data = myfile.read()
# Вилучаємо дані
values = struct.unpack(b'>i4sh', data)
print(values)
```

Методи:

struct.unpack(format, data)

Розпаковує дані зі структури

Приклади:

```
два цілі 4х байтові числа прямого порядку
struct.unpack('>LL', b'x00x00x00x9ax00x00x00x8d')
# (154, 141)

два цілі 4х байтові числа прямого порядку
struct.unpack('>2L', b'x00x00x00x9ax00x00x00x8d')
# (154, 141)

два цілі 4х байтові числа прямого порядку, пропустивши 1
і 2 байти по краях
struct.unpack('>1x2L2x',
     b'x00x00x00x00x9ax00x00x00x8dx00x00')
# (154, 141)
```

struct.pack(format, data)

Пакує дані до структури

```
struct.pack('>L', 154)
# b'x00x00x00x9a'

struct.pack('>L', 141)
# b'x00x00x00x8d'
```

Специфікатори формату

>- Прямий порядок
< -зворотний порядок
x- пропустити один байт
b -знаковий один байт
B -беззнаковий байт
h -знакове коротке ціле число, 2 байти
H -беззнакове коротке ціле число, 2 байти
i -знакове ціле число, 4 байти
I -беззнакове ціле число, 4 байти
l -знакове довге ціле число, 4 байти
L -беззнакове довге ціле число, 4 байти
Q -беззнакове дуже довге ціле число, 8 байт
f -число з плаваючою точкою, 4 байти
d -число з плаваючою точкою подвійної точності, 8 байт
p -лічильник та символи, 1 + count байт
s -символи, count символів

Приклад запису та читання речових даних у бінаному файлі

```
importsys
importstruct
myfile = open("data.bin", "wb")
# упаковуємо дані
fori in range(100):
    x1=10.0*float(i+0)
    x2=10.0*float(i+1)
    x3=10.0*float(i+2)
    x4=10.0*float(i+3)
    x5=10.0*float(i+4)
    print(x1, x2, x3, x4, x5)
    b=struct.pack(b'>fffff', x1,x2,x3,x4,x5)
    myfile.write(b)
myfile.close()
#sys.exit(0)
myfile = open("data.bin", "rb")
while(True):
    data = myfile.read(4*5)
    if data==b'':
        break
    # Вилучаємо дані
    values = struct.unpack(b'>fffff', data)
    y1=values[0]
    y2=values[1]
    y3=values[2]
    y4=values[3]
    y5=values[4]
    print(y1, y2, y3, y4, y5)
myfile.close()
```

Приклад запису файлу на FORTRAN та читання на PYTHONречових даних у бінаному файлі

Формування файлу:

```
! компілятор ming gfortran
  implicit none
  integer :: j, i
  real(4), DIMENSION(5) :: buf
  character(40) :: fname='d:\float5.bin'
  open(unit=50, file=trim(fname), status='UNKNOWN',
form='UNFORMATTED',err=100)
```

```
!
! запис у файлі містить у "початку та "в кінці" запису
! "дискриптор" Чотири байти - кількість байт
! у запису крім дискрипторів
!
  do i=1,50
     buf(1)=float(i)
     buf(2)=float(i+1)
     buf(3)=float(i+2)
     buf(4)=float(i+3)
     buf(5)=float(i+4)
     write(50) (buf(j), j=1,5)
     write(*, '( 5(g12.5,1x) )' ) (buf(j), j=1,5)
  enddo
  close(50)
  stop 0
100 write(*,*) '* error open file ', trim(fname), '*'
    stop 16
    end
```

Читання файлу:

```
import sys
import struct
myfile = open(r"d:\float5.bin", "rb")
##kbyteB=myfile.read(4) # дискриптор запису
nzap=0
while(True):
    kbyteB=myfile.read(4) # дискриптор запису -
                          # кількість байт у цьому записі
    if kbyteB==b'':
        break
    nzap+=1
    kbyte = struct.unpack(b'<L', kbyteB)[0]
    print('запис № ',nzap,' містить ',kbyte,' байт')
    data = myfile.read(4*5) #читання запису
    if data==b'':
        break
    # Вилучаємо дані
    values = struct.unpack(b'<fffff', data)
    y1=values[0]
    y2=values[1]
    y3=values[2]
    y4=values[3]
    y5=values[4]
    print(y1, y2, y3, y4, y5) #values)
```

```
    kbyteB=myfile.read(4) # дискриптор кінця запису - # кількість байт у цьому записі
myfile.close()
```

V. Файли CSV

Програмісти часто стикаються із завданням обробки великих обсягів структурованих даних. Python має вбудовану бібліотеку CSV, за допомогою якої програміст може працювати із спеціальними CSV файлами. Це своєрідні електронні таблиці.

Кожен рядок у файлі csv представляє окремий запис або рядок, який складається з окремих стовпців, розділених комами. Саме тому формат і називається Comma Separated Values. Але хоча формат CSV - це формат текстових файлів, Python для спрощення роботи з ним надає спеціальний вбудований модуль CSV

Що таке файли CSV

CSV - це особливий вид файлу, який дозволяє структурувати великі обсяги даних.

По суті він є звичайним текстовим файлом, однак кожен новий елемент відокремлений від попереднього комою або іншим роздільником. Зазвичай кожен запис починається з нового рядка. Дані CSV можна легко експортувати до електронних таблиць або баз даних. Програміст може розширювати файл CSV, додаючи нові рядки.

Приклад CSV файлу, де як роздільник використовується кома:

```
Ім'я,Професія,Рік народження
Віктор, Токар, 1995
Сергій, Зварювальник, 1983
```

Як видно з прикладу, у першому рядку зазвичай вказується, яка інформація буде знаходитись у кожному стовпці. Крім того, після останнього елемента рядка кома не ставиться, інтерпретатор визначає кінець рядка за символом перенесення.

Замість коми можна використовувати будь-який інший роздільник, тому при читанні файлу CSV потрібно заздалегідь знати, який символ використовується.

Важливо пам'ятати, що CSV - це звичайний текстовий файл, який не підтримує символи в кодуваннях, які відрізняються від ASCII або Unicode.

Бібліотека CSV

Це основна бібліотека для роботи з файлами CSV в Python.

Бібліотека csv є вбудованою, тому її не потрібно завантажувати, достатньо використовувати звичайний імпорт:

```
import csv
```

Читання з файлів (парсинг)

Для того, щоб прочитати дані з файлу, програміст повинен створити об'єкт reader:

```
reader_object = csv.reader(file, delimiter = ",")
```

readerмає метод __next__(), тобто є об'єктом, що ітерується, тому читання з файлу відбувається наступним чином:

```
import csv
with open("classmates.csv", encoding='utf-8') as r_file:
    # Створюємо об'єкт reader, вказуємо символ-розділювач ","
    file_reader = csv.reader(r_file, delimiter = ",")
    # Лічильник для підрахунку кількості рядків та виведення заголовків стовпців
    count = 0
    # Зчитування даних із CSV файлу
    for row in file_reader:
        if count == 0:
            # Виведення рядка, що містить заголовки для стовпців
            print(f'Файл містить стовпці: {", ".join(row)}')
        else:
            # Виведення рядків
            print(f'{row[0]} - {row[1]} і він народився в {row[2]} році.')

        count += 1
    print(f'Всього у файлі {count} рядків.')
```

Припустимо, що ми маємо CSV файл, який містить таку інформацію:

```
Ім'я,Успішність,Рік народження
Саша, відмінник, 200
Маша,хорошистка,1999
Петя, трієчник, 2000
```

Тоді, якщо відкрити цей файл у нашій програмі, будуть отримані наступні результати:

```
Файл містить стовпці: Ім'я, Успішність, Рік народження
    Сашко - відмінник і він народився 200 року.
    Маша - хорошистка і він народився 1999 року.
    Петро - трієчник і він народився 2000 року.
Усього у файлі 4 рядків.
```

Використання конструкції with…as дозволяє програмісту бути впевненим, що файл буде закритий, навіть якщо під час виконання коду станеться якась помилка.

Зверніть увагу, що при відкритті потрібно вказати правильне кодування, в якому зберігаються дані. В даному випадку encoding = 'utf-8'. Якщо не вказувати, то використовуватиметься кодування за замовчуванням. Для Windows це CP1251.

Бібліотека CSV дозволяє працювати з файлами, як із словниками, для цього потрібно створити об'єкт DictReader. Звертатися до елементів можна на ім'я стовпців, а не за допомогою індексів. Щоб вихідна програма робила аналогічний висновок, її слід змінити так:

```
import csv
with open("classmates.csv", encoding='utf-8') as r_file:
    #Створюємо об'єкт DictReader, вказуємо символ-розділювач ","
    file_reader = csv.DictReader(r_file, delimiter = ",")
    # Лічильник для підрахунку кількості рядків та виведення заголовків стовпців

    count = 0
    # Зчитування даних із CSV файлу
    for row in file_reader:
        if count == 0:
            # Виведення рядка, що містить заголовки для стовпців
            print(f'Файл містить стовпці: {", ".join(row)}')
        # Виведення рядків
        print(f' {row["Ім'я"]} - {row["Успішність"]}', end='')
        print(f' і він народився в {row["Рік народження"]} року.')
        count += 1
    print(f'Всього у файлі {count + 1} рядків.')
```

Звертатися до елементів за назвою стовпця зручніше, крім того, це спрощує розуміння коду.

Зверніть увагу, що цикл for при першій ітерації буде записаний в row не шапка таблиці, а перший її рядок. Тому за виведення кількості рядків змінну count збільшили на 1.

Додаткові параметри об'єкта DictReader

DictReaderмає параметри:

- `dialect`— Набір параметрів форматування інформації. Докладніше про них нижче.
- `line_num`— Встановлює кількість рядків, які можна прочитати.
- `fieldnames`— Визначає заголовки для стовпців, якщо не визначити атрибут, в нього запишуться елементи з першого прочитаного рядка файлу. Заголовки потрібні для того, щоб легко було зрозуміти, яка інформація міститься або повинна міститись у стовпці.

Наприклад, якби в classmates.csv не було першого рядка із заголовками, то можна було б його відкрити наступним чином:

```
fieldnames = ['Ім'я', 'Успішність', 'Рік народження']
file_reader = csv.DictReader(r_file, fieldnames = fieldnames)
```

Також можна використовувати метод __next__() для отримання наступного рядка. Цей метод робить об'єкт reader ітерованим. Тобто він викликається за кожної ітерації і повертає наступний рядок. Цей метод і використовується при кожній ітерації в циклі для отримання чергового рядка.

Запис до файлів

Для запису інформації в CSV файл необхідно створити об'єкт writer:

```
file_writer = csv.writer(w_file, delimiter = "\t")
```

Для запису файл даних використовується метод writerow(), який має наступний синтаксис:

```
writerow(["Ім'я", "Прізвище", "По батькові"])
```

Код програми для запису в CSV файл виглядає так:

```
import csv
with open("classmates.csv",mode="w",encoding='utf-8') as w_file:
        file_writer = csv.writer(w_file, delimiter = ",", lineterminator="\r")

        file_writer.writerow(["Ім'я", "Клас", "Вік"])
        file_writer.writerow(["Женя", "3", "10"]))
        file_writer.writerow(["Саша", "5", "12"]))
        file_writer.writerow(["Маша", "11", "18"])
```

Зверніть увагу, що при записі використовувався lineterminator="r". Це роздільник між рядками таблиці, за умовчанням він \r\n.

Після виконання програми у файлі CSV буде наступний текст:

```
Ім'я,Клас,Вік
Женя,3,10
Саша, 5,12
Маша, 11,18
```

Як параметр метод writerow() приймає список, елементи якого будуть записані в рядок через символ-розділювач.

Запис у файл також можна здійснити за допомогою об'єкта DictWriter.

Важливо пам'ятати, що він потребує явної вказівки параметра fieldnames. Як аргумент методу writerow використовується словник.

Код програми виглядає так:

```
import csv
with open("classmates.csv", mode="w", encoding='utf-8') as w_file:

        names = ["Ім'я", "Вік"]
        file_writer = csv.DictWriter(w_file, delimiter = ",",
```

```
                                        lineterminator="\r",
                                            fieldnames=names)
    file_writer.writeheader()
    file_writer.writerow({"Ім'я": "Саша", "Вік": "6"})
    file_writer.writerow({"Ім'я": "Маша", "Вік": "15"})
    file_writer.writerow({"Ім'я": "Вова", "Вік": "14"})
```

Виведення у файл буде наступним:

```
Ім'я,Вік
Саша,6
Маша,15
Вова,14
```

Додаткові параметри DictWriter

Об'єкт writer також має атрибут dialect, який визначає, як форматуватимуться дані під час запису файл, про нього буде описано нижче.

Крім того, writer має методи:

- **writerows(rows)**- Записує всі елементи рядків.
- **writeheader()**-Виводить заголовки для стовпців. Заголовки повинні бути передані об'єкту writer у вигляді списку, як атрибут fieldnames. writeheader був використаний у попередньому прикладі.

Розглянемо застосування writerows:

```
file_writer.writerows([{"Ім'я": "Саша", "Вік": "6"},
                       {"Ім'я": "Маша", "Вік": "15"},
                      {"Ім'я": "Вова", "Вік": "14"}])
```

Діалекти

Щоб не вказувати формат вхідних і вихідних даних, певні параметри форматування згруповані в діалекти (dialect).

При створенні об'єкта reader або writer програміст може вказати потрібний діалект, крім того, деякі параметри діалекту можна перевизначити вручну, також вказавши їх при створенні об'єкта.

Для створення діалекту використовується команда:

```
register_dialect("ім'я", delimiter = "\t", ...)
```

Клас Dialect дозволяє визначити наступні атрибути форматування:

Атрибут	Значення
`delimiter`	Встановлює символ, за допомогою якого елементи розділяються у файлі. За замовчуванням використовується кома.
`doublequote`	Якщо True, то символ quotechar подвоюється, якщо False, то до символу qutechar додається ecsapechar як префікс.
`escapechar`	Рядок з одного символу, який використовується для екранування символу-розділювача.

Атрибут	Значення
lineterminator	Визначає роздільник для рядків, за замовчуванням використовується \r\n
quotechar	Визначає символ, який використовується для оточення символу-розділювача. За замовчуванням використовуються подвійні лапки, тобто quotechar = '".
quoting	Визначає символ, який використовується для екранування роздільного символу (якщо не використовуються лапки).
skipinitialspace	Якщо встановити значення цього параметра в True, всі прогалини після символу-розділювача будуть ігноруватися.
strict	Якщо встановити в True, то при неправильному введенні CSV збуджуватиметься виняток Error.

Приклад використання:

```
import csv
csv.register_dialect('my_dialect', delimiter=':', lineterminator="\r")

with open("classmates.csv", mode="w", encoding='utf-8') as w_file:
    file_writer = csv.writer(w_file, 'my_dialect')
    file_writer.writerow(["Ім'я", "Клас", "Вік"])
    file_writer.writerow(["Женя", "3", "10"]))
    file_writer.writerow(["Саша", "5", "12"]))
    file_writer.writerow(["Маша", "11", "18"])
```

В результаті отримаємо:

```
Ім'я:Клас:Вік
Женя:3:10
Саша:5:12
Маша:11:18
```

приклад

```
import csv
FILENAME = "users.csv"
users = [
    [1, "математика", 75],
    [2, "програмування", 85],
    [3, "мат аналіЗ", 34]
```

```
    ]

    with open(FILENAME, "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerows(users)

    with open(FILENAME, "a", newline="") as file:
        user = [4, "веб прог", 80]
        writer = csv.writer(file)
        writer.writerow(user)

    d={}
    with open(FILENAME, "r", newline="") as file:
        reader = csv.reader(file)
        for row in reader:
            print(row[0], "-", row[1], "-", row[2])
            d[row[1]]=int(row[2])
    print("d=",d)

    #сформуємо список D1 зі словника та d
    # та його сортуємо за ключом словника key=lambda x:x[0]
    # якщо хочемо за значенням, пишіть key=lambda x:x[1]
    # якщо хочемо у зворотному порядку, то пишіть key=lambda x:-
x[1]
    D1=sorted(d.items(),key=lambda x:-x[1]) # за ключами

    print("D1=",D1) # друк списку

    # відсортований список у словник D2 та його друк
    D2={D1[i][0] : D1[i][1] for i in range(len(D1)) }
    #друк таблицею D2
    for kk, vv in D2.items():
        print(vv,'\t',kk)

    ll=list(D2.items())
    kkl = len (ll)-1
    print("maximum score of", ll[0][1], "in the subject", ll[0][0])
    print("minimum score of", ll[kkl][1], "in the subject",
ll[kkl][0])

    user1=[]
    nn=0
    for kk, vv in D2.items():
        user1.append([nn,kk,vv])
```

```
        nn+=1

    namerow=["№пп", "предмет", "оцінка"]
    with open(FILENAME, "w", newline="") as file: #encoding='utf-8'
        writer=csv.writer(file, delimiter=",", lineterminator="\n")
        writer.writerow(namerow)
        writer.writerows(user1)
```

VI. Модуль shelve (*.ini)

Для роботи з бінарними файлами Python може застосовуватися ще один модуль - shelve. Він зберігає об'єкти у файл із певним ключем. Потім за цим ключем може витягти збережений об'єкт з файлу. Процес роботи з даними через модуль shelve нагадує роботу зі словниками, які також використовують ключі для збереження та вилучення об'єктів.

Для відкриття файлу модуль shelve використовує функцію open():

```
open(шлях_до_файлу[, flag="c"[, protocol=None[,
writeback=False]]])
```

Де параметр flag може приймати значення:

- **c**: файл відкривається для читання та запису (значення за промовчанням). Якщо файл не існує, він створюється.
- **r**: файл відкривається лише для читання.
- **w**: файл відкривається для запису
- **n**: файл відкривається для запису Якщо файл не існує, він створюється. Якщо він існує, то він перезаписується

Для закриття підключення до файлу викликається метод close():

```
importshelve
d = shelve.open(filename)
d.close()
```

Або можна відкривати файл за допомогою оператора with.

Збережемо та рахуємо у файл кілька об'єктів:

```
importshelve

FILENAME = "states2"
withshelve.open(FILENAME) as states:
    states["London"] = "Great Britain"
    states["Paris"] = "France"
    states["Berlin"] = "Німеччина"
    states["Madrid"] = "Spain"

withshelve.open(FILENAME) as states:
    print(states["London"])
```

```
    print(states["Madrid"])
```

Запис даних передбачає встановлення значення для певного ключа:

```
states["London"] = "Great Britain"
```

А читання з файлу еквівалентне отриманню значення за ключом:

```
print(states["London"])
```

Як ключі використовуються рядкові значення.

При читанні даних, якщо запитуваний ключ відсутній, то генерується виняток. У цьому випадку перед отриманням ми можемо перевіряти наявність ключа за допомогою оператора in:

```
withshelve.open(FILENAME) as states:
    key = "Brussels"
    if key in states:
        print(states[key])
```

Можна також використовувати метод get(). Перший параметр методу - ключ, яким слід отримати значення, а другий - значення за промовчанням, яке повертається, якщо ключ не знайдено.

```
withshelve.open(FILENAME) as states:
    state = states.get("Brussels", "Undefined")
    print(state)
```

Використовуючи цикл for, можна перебрати всі значення файлу:

```
withshelve.open(FILENAME) as states:
    for key in states:
        print(key," - ", states[key])
```

Метод keys() повертає всі ключі з файлу, а метод values() - всі значення:

```
withshelve.open(FILENAME) as states:

    for city in states.keys():
        print(city, end=" ") # London Paris Berlin Madrid
    print()
    for country in states.values():
        print(country, end=" ") # Great Britain # France
Germany Spain
```

Ще один метод items() повертає набір кортежів. Кожен кортеж містить ключ та значення.

```
with shelve.open(FILENAME) as states:

    for state in states.items():
        print(state)
```

Консольний висновок:

```
("London", "Great Britain")
("Paris", "France")
```

```
("Berlin", "Німеччина")
("Madrid", "Spain")
```

Оновлення даних

Для зміни даних достатньо присвоїти по ключу нове значення, а для додавання даних визначити новий ключ:

```
importshelve

FILENAME = "states2"
withshelve.open(FILENAME) as states:
    states["London"] = "Great Britain"
    states["Paris"] = "France"
    states["Berlin"] = "Німеччина"
    states["Madrid"] = "Spain"

withshelve.open(FILENAME) as states:

    states["London"] = "United Kingdom"
    states["Brussels"] = "Belgium"
    for key in states:
        print(key, "-", states[key])
```

Видалення даних

Для видалення з одночасним отриманням можна використовувати функцію pop(), в яку передається ключ елемента та значення за промовчанням, якщо ключ не знайдено:

```
withshelve.open(FILENAME) as states:

    state = states.pop("London", "NotFound")
    print(state)
```

Також для видалення може застосовуватись оператор del:

```
withshelve.open(FILENAME) as states:

    del states["Madrid"] # видаляємо об'єкт # з ключем
Madrid
```

Для видалення всіх елементів можна використовувати метод clear():

```
with shelve.open(FILENAME) as states:

    states.clear()
```