§9 ФАЙЛИ

Файли

Файл (англ. file – папка) – це іменований блок інформації (послідовність байтів), який зберігається на носії інформації.

Файл є найменшою одиницею збереження інформації на носії. Файл має такі ознаки:

- фіксоване ім'я (назва файла) – послідовність символів, що однозначно характеризує файл;
- певне логічне зображення (що визначається типом інформації, що міститься у файлі) і відповідні йому операції читання/запису;
- розмір файла (характеризується розміром інформації, що в ньому міститься).

Роботу з файлами можна розділити на три основних етапи:

- Відкриття файлу;
- Робота з інформацією, що стосується файлу (читання/запис);
- Закриття файлу.

При роботі з файлами розрізняються **двійкові** (або бінарні) і **текстові** файли. Двійкові файли відкриваються як послідовність байтів. При цьому відповідальність за коректну робота з даними повністю покладається на програму, що використовує цей файл. Текстові файли відкриваються як послідовність рядків (символів), що міститься у файлі. При цьому фізичний рядок у файлі відповідає рядковому літералу у програмі. Створювати, змінювати та опрацьовувати двійкові файли можна як правило лише за допомогою спеціальної програми. З текстовими файлами можна працювати за допомогою будь-якого текстового редактора.

Відкриття файлу

Відкриття файлу здійснюється інструкцією

```
f = open(file_name, mode)
```

де `f` – ім'я файлової змінної, `file_name` – ім'я файла, що відкривається, `mode` – режим роботи з файлом – це рядковий літерал, що будується за допомогою значень наведених у таблиці.

Таблиця 9.1. Режими роботи з файлом

Режим	Опис
`"r"`	Відкриття для читання. Якщо файла з таким ім'ям не існує, то інструкція породжує помилку (типове значення).
`"w"`	Відкриття для запису. Якщо файл з таким ім'ям вже існує, то цей файл буде перезаписаний новим.
`"x"`	Відкриття для запису. Якщо файл з таким ім'ям вже існує, то інструкція породжує помилку.
`"a"`	Відкриття для додавання даних у кінець файлу.
`"+"`	Відкриття на читання та запис.
`"b"`	Відкриття у двійковому режимі
`"t"`	Відкриття у текстовому режимі (типове значення).

Значення режиму `mode` отримується об'єднанням значень наведених вище. Наприклад, `"rb"` – відкриття існуючого двійкового файлу для читання. `"r"` і `"t"` – це типові значення, їх можна опускати при формуванні режиму `mode`. Типове значення параметра `mode` – це `"rt"`.

Інструкція `open` ототожнює вміст файла `file_name` зі змінною `f`. Подальша робота з файлом відбувається лише через цю змінну. Крім цього інструкція `open` блокує файл для змін іншими програмами, щоб уникнути конфліктних ситуацій.

Закриття файлу

Як було сказано вище, робота з файлом відбувається через файлову змінну, що зберігає свої дані у оперативній пам'яті. Для того, щоб зберегти результат роботи з файлом на носій інформації, файл необхідно закрити. Крім цього операція закриття файлу повідомляє операційній системі, що файл розблокований і може використовуватися (для зміни) іншими програмами.

Для того, щоб закрити файл використовується інструкція

```
f.close()
```

де `f` – ім'я файлової змінної, пов’язаної з файлом.

Операції з файлами

Операція відкриття файлу породжує спеціальний об'єкт (що належить файловій змінній), що називається **маркером**. Маркер вказує на поточну позицію у файлі, тобто місце з якого відбувається читання чи запис даних.

Під час відкриття файлу у режимі `"a"` (додавання) маркер встановлюється у позицію після останнього запису. Для інших режимів маркер встановлюється

на найперший запис. Будь-яка операція читання/запису змінює поточну позицію маркера.

Нехай `f` – файлова змінна, яка асоційована з файлом відкритим у відповідному режимі. Тоді

Таблиця 9.2. Операції з файлами

Операція	Опис
`f.read()`	Повертає весь вміст файла.
`f.read(n)`	Читає `n` байтів з поточного положення маркера.
`f.readline()`	Читає та повертає поточний рядок з текстового файлу.
`f.readlines()`	Повертає список всіх рядків текстового файлу.
`f.write(s)`	Записує значення змінної `s` у файл.
`print(s, file = f)`	Записує значення змінної `s` у файл.
`f.writelines(lineslist)`	Записує список рядків `lineslist` у текстовий файл.
`f.tell()`	Повертає значення поточної позиції маркера у файлі.
`f.seek(n)`	Встановлює маркер у позицію `n` у файлі.

Текстові файли

Файлова змінна для текстового файлу – це колекція його рядків. Отже, текстовий файл можна проходити по рядках, використовуючи цикл **`for`**.

Нехай `f` – файлова змінна відкритого для читання текстового файлу. Тоді інструкція

```
for line in f:
    process_iteration
```

виконує `process_iteration` для всіх рядків файлу `f`.

Розглянемо кілька прикладів, що ілюструють роботу з текстовими файлами.

Приклад 9.1. У текстовому файл містить набір чисел, кожне з яких записано з нового рядка. Потрібно знайти суму цих чисел.

Розв'язок. Для розв'язання задачі, у програмі потрібно відкрити файл у текстовому режимі для читання і пройти по його колекції рядків.

Отже програма буде мати вигляд

```
f_name = input("Введіть ім'я файлу ")
f = open(f_name) # Відкриваємо текстовий файл для читання
suma = 0
for line in f:          # Пробігаємо файл f по рядках
    suma += float(line) # Перетворюємо line у дійсне число
print("Сума чисел записана у файлі = {}".format(suma))
f.close()               # Не забуваємо закривати файл!
```

Для перевірки роботи програми створимо текстовий файл з іменем `123.txt` який розмістимо у тій же папці, що і файл з програмою. Заповнимо цей текстовий файл деякими числами, наприклад, таким чином:

```
1
3
12
5
34
```

Запустимо виконання програми. На запит введемо ім'я вищестореного файлу. Тоді результатом виконання цієї програму буде

```
Введіть ім'я файлу 123.txt
Сума чисел записана у файлі = 55.0
```

Вище, у параграфі «Обробка виключень» було розказано про менеджер контексту. Робота з файлами, це один з випадків, коли використання контексту є більш ніж обґрунтованим. Змінимо вищенаведений код програми еквівалентним йому з використанням менеджеру контексту

```
f_name = input("Введіть ім'я файлу ")
suma = 0

with open(f_name) as f:      # Відкриваємо файл для читання за
                             # допомогою менеджеру контексту
    for line in f:           # Пробігаємо файл f по рядках
        suma += float(line) # Підраховуємо суму
# закриття файлу буде здійснено автоматично,
# щойно відбудетьс вихід з контесту

print("Сума чисел записана у файлі = {}".format(suma))
```

Приклад 9.2. Згенерувати текстовий файл, що буде містити послідовність чисел Фібоначчі, що обмежена заданим числом.

Розв'язок.

```
N = int(input("Задайте N "))
f_name = input("Введіть ім'я файлу для запису ")
f = open(f_name, "w")  # Створюємо текстовий файл для запису
F1 = F2 = 1
print(F2, file=f)      # Записуємо F2 у файл f
while F1 <= N:
    F1, F2 = F1 + F2, F1
    print(F2, file=f)  # Записуємо F2 у файл f
f.close()  # Щоб зберегти зміни у файлі
           # не забудьмо закрити файл!
```

Приклад 9.3. Дано текстовий файл, що містить послідовність слів. Крім цього задано пару слів `w1` та `w2`. Замінити у файлі усі входження слова `w1`, словом `w2`.

Розв'язок. Обробку файлу здійснимо у кілька кроки. Спочатку відкриємо файл та зчитаємо з нього дані у список рядків (після чого обов'язково закриємо файл). Далі, у отриманому списку рядків здійснимо заміну всіх входжень слова `w1` словом `w2`. І нарешті, на останньому кроці перезапишемо початковий файл оновленими даними.

```
f_name = input("Введіть ім'я файлу ")
w1 = input("Задайте w1 ")
w2 = input("Задайте w2 ")

f = open(f_name) # Відкриваємо текстовий для читання
f_lines = f.readlines()  # Зчитуємо рядки файла f у f_lines
f.close()        # Закриваємо файл, щоб перезаписати його

f = open(f_name, "w") # Перезаписуємо файл
for s in f_lines:     # Пробігаємо список усіх рядків
    s1 = s.replace(w1, w2) # Будуємо рядок s1 по рядку s у
                           # якому слова w1 замінено на w2
    print(s1, file = f, end = "")  # Записуємо s1 у файл f
f.close()
```

Для тестування програми створимо файл `1.txt` з таким вмістом

```
1 112 change 33 44
2 11 223 change 44
3 11 22 334 44
4 change 22 33 445
```

```
5 11 252 33 44
6 11 change 353 44
```

де слово `change` планується замінити іншим словом, наприклад, `replaced`. Запустимо програму та введемо дані таким чином:

```
Введіть ім'я файлу 1.txt
Задайте w1 change
Задайте w2 replaced
```

Відкривши файл `1.txt`, переконаємося, що всі входження слова `change` були замінені словом `replaced`:

```
1 112 replaced 33 44
2 11 223 replaced 44
3 11 22 334 44
4 replaced 22 33 445
5 11 252 33 44
6 11 replaced 353 44
```

Бінарні файли і сереалізатори

Обробка бінарних файлів без додаткових засобів є досить не простою задачею, оскільки запис і читання з файлу потрібно проводити у двійковому форматі (як послідовність байтів).

Розглянемо роботу з бінарними файлами за допомогою спеціалізованих бібліотек які дозволяють зберігати та завантажувати складні об'єкти Python, зокрема списки, словники, користувацькі об'єкти тощо.

Модуль pickle

Практично будь-який об'єкт можна зберегти на диску (сереалізувати) у будь-який момент його існування, а пізніше відновити його з диску (десереалізувати). Модуль `pickle` реалізує потужний алгоритм сереалізації та десереалізації об'єктів Python.

Перш ніж користуватися можливостями модуля, його необхідно підключити інструкцією

```
import pickle
```

Відкриття та закриття файлу здійснюється стандартним чином. При відкритті файлу інструкцію `open` для читання чи запису, значення `mode` має бути `'wb'` або `'rb'` відповідно.

Щоб сереалізувати об'єкт `obj` у відкритому для запису файлі `f` треба викликати інструкцію

```
pickle.dump(obj, f)
```

Для десереалізації даних з відкритого для читання файлу `f` треба скористатися інструкцією

```
obj = pickle.load(f)
```

При цьому дані будуть записані у змінну `obj`.

Ніколи не завантажуйте з допомогою модуля `pickle` дані з ненадійних джерел. Це може призвести до незворотних наслідків з даними на вашому комп'ютері.

Приклад 9.4. Нехай задано деякий об'єкт з даними (наприклад, словник). Потрібно сереалізувати цей об'єкт у файл з допомогою модуля `pickle`. Відновити дані з файлу і вивести на екран.

Розв'язок.

```
import pickle
# Дані записані у вигляді словника
data = {
    'a': [1, 2.0, 3, 4 + 6j],
    'b': ("String", "Hello"),
    'c': {None, True, False}
}

f_name = input("Введіть ім'я файлу ")
f = open(f_name, "wb")     # Відкриваємо файл для читання
pickle.dump(data, f)       # Сереалізація даних у файл
f.close()                  # Закриваємо файл

f = open(f_name, "rb")     # Відкриваємо файл для читання
data_new = pickle.load(f) # Десереалізуємо дані у data_new
f.close()
print(data_new)
```

Запустимо програму на виконання. На запит програми введемо ім'я файлу `data.pickle`.

```
Введіть ім'я файлу data.pickle
```

```
{'b': ('String', 'Hello'), 'c': {False, True, None}, 'a': [1,
2.0, 3, (4+6j)]}
```

Звернемо увагу на те, що у папці з програмою з'явився файл `data.pickle`.

Модуль shelve

Модуль `shelve` використовують для зберігання у файлах та відновлення з них даних, що мають вигляд словника. Під час роботи з модулем `shelve`, Python створює окрім основного декілька додаткових службових файлів, що використовуються для організації доступу до даних.

Перш ніж користуватися можливостями модуля `shelve`, його необхідно підключити інструкцією

```
import shelve
```

Модуль `shelve` має власну команду для відкриття файлів.

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

де `filename` – ім'я файлу у якому містяться дані. При цьому змінна `d` одночасно є і файловою змінною і словником, що містить дані.

У кінці роботи потрібно закрити файл командою.

```
d.close()
```

Приклад 9.5. Нехай задано деякий словник. Потрібно зберегти цей словник у файл з допомогою модуля `shelve`. Відновити дані з файлу і вивести на екран.

Розв'язок.

```
import shelve
# Дані записані у вигляді словника
data = {
    'a': [1, 2.0, 3, 4 + 6j],
    'b': ("String", "Hello"),
    'c': {None, True, False}
}

f_name = input("Введіть ім'я файлу ")
# Створюємо/перезаписуємо файл
```

```
d = shelve.open(f_name)
for k, v in data.items():
    d[k] = v
d.close()

# Відкриваємо файл та виводимо його вмісти на екран
d = shelve.open(f_name)
for k, v in d.items():
    print(k, v)
d.close()
```

Запустимо програму на виконання. На запит програми введемо ім'я файлу `data`.

```
Введіть ім'я файлу data
c {False, True, None}
b ('String', 'Hello')
a [1, 2.0, 3, (4+6j)]
```

Звернемо увагу на те, що у папці з програмою з'явився файл `data.dat` – при створенні файлу йому автоматично було додано розширення `dat`.

Робота з файловою системою

Файлова система визначає спосіб організації файлів на носіях даних. У сучасних операційних системах файлова система на логічному рівні є ієрархією каталогів (також використовують терміни тека, папка) у вигляді дерева. Кожен каталог може містити файли або підкаталоги. Кожна програма має поточний каталог. Як правило, типовим значенням поточного каталогу є каталог з якого запускається програма.

Для роботи з файловою системою використовується стандартний модуль `os`. Розглянемо основні функції модуля `os` для роботи з файловою системою.

Таблиця 9.3. Основні операції роботи з файловою системою.

Операція	Опис
`os.listdir(path='.')`	Повертає список файлів та підкаталогів у каталозі `path`.
`os.walk(top)`	Генератор-функція, що повертає послідовність кортежів для всіх підкаталогів дерева, що починається з каталогу `top`. Для кожного підкаталогу (та для самого `top`) повертається кортеж (`dirpath`, `dirnames`, `filenames`), де `dirpath` – каталог, `dirnames` – список

	підкаталогів каталогу `dirpath`, `filenames` – список файлів каталогу `dirpath`.
`os.getcwd()`	Повертає поточний каталог.
`os.chdir(path)`	Робить поточним каталог `path`.
`os.mkdir(path)`	Створює каталог `path`.
`os.rmdir(path)`	Видаляє каталог `path`. Каталог повинен бути порожнім.
`os.remove(path)`	Видаляє файл `path`.
`os.path.exists(path)`	Перевіряє, чи існує каталог (файл) `path`.
`os.path.getmtime(path)`	Повертає дату та час останньої зміни файлу (каталогу) `path`
`os.path.getctime(path)`	Повертає дату та час створення файлу (каталогу) `path`
`os.path.getsize(path)`	Повертає розмір файлу `path`
`os.path.isdir(path)`	Повертає **`True`** якщо `path` є каталогом
`os.path.isfile(path)`	Повертає **`True`** якщо `path` є файлом
`os.path.normpath(path)`	Повертає `path` у «стандартному» вигляді.
`os.path.join(path, *paths)`	Об'єднує декілька частин шляху та імені файлу `paths` у єдиний шлях `path`. При цьому символи-розділювачі у шляху вставляються/змінюються у відповідності до вимог операційної системи.
`os.path.split(path)`	Розбиває шлях на дві частини: каталог та ім'я файлу.
`os.path.splitext(path)`	Розбиває шлях на дві частини: каталог плюс початок імені файлу та розширення імені файлу.

Задачі для самостійної роботи

9.1. У текстовому файлі записана непорожня послідовність дійсних чисел, які розділяються пропусками. Визначити функції для обчислення:

a) суми компонент файла;
b) кількості від'ємних компонент файла;
c) останньої компоненти файла;
d) найбільшого із значень компонент файла;
e) найменшого із значень компонент файла з парними номерами;
f) суми найбільшого і найменшого із значень компонент;
g) різниці першої й останньої компоненти файла;
h) кількості компонент файла, які менші за середнє арифметичне всіх його компонент.

9.2. Дано текстовий файл, компоненти якого є цілими числами. Скласти