eebb

ЯЗЫК PYTHON

Лекция 12. Файлы

Введение

- Именованный набор данных
- На внешнем носителе
- Чтобы обработать, надо прочитать
- Чтобы сохранить обработанное, надо записать

Текстовые и двоичные

- Примеры двоичных: word-документ (как правило), картинка, pdf
- Пример текстового: py-файл, простой текст
- Бывают текстовые файлы, оформленные по определенным правилам
- Примеры: html, csv, excel

Текстовые и двоичные

- Для сложных форматов есть встроенные модули
- И/или внешние библиотеки
- Бинарные здесь не рассматриваем
- Сфокусируемся на простых текстовых

Подходы к изменению данных

- Можем открыть один файл на запись и чтение
- Можем двигаться по файлу
- Где-то что-то читать
- Где-то что-то записывать
- Поверх старого или добавлять

Подходы к изменению данных

- Можем открыть один файл на чтение
- И измененные данные писать в новый файл
- Или результаты обработки
- Сфокусируемся на этом подходе
- Он проще
- И устойчив к внезапному завершению

Открытие и закрытие файла

- Открывается функцией open
- Два параметра - имя файла и режим
- Оба - строки
- Имя - абсолютное или относительное
- Зависит от операционной системы

Открытие и закрытие файла

- Всегда работаем в каком-то текущем каталоге
- Просто имя файла - это файл в текущем каталоге
- Если в текущем каталоге есть подкаталог subdir
- А в нем файл data.txt
- Его назовем subdir/data.txt

Открытие и закрытие файла

- В Windows официальный разделитель имен в каталоге - '\'
- Но '/' тоже понимается
- И лучше использовать его
- Потому что '\' имеет специальный смысл в строке
- И получаем универсальную форму пути

Открытие и закрытие файла

- Можно попасть в родительский каталог
- Через специальное имя '..'
- И в его родительский
- А потом спуститься в другой подкаталог
- Или подниматься до корня

Открытие и закрытие файла

- Абсолютные пути начинаются с имени диска на Windows
- Или '/' на MacOS
- Второй параметр - строка из нескольких букв
- Одна - 't' или 'b' (текстовый или двоичный)
- Другая - 'r' или 'w' (чтение или запись)

Пример

```
1 f = open('data1.txt', 'rt')
2 print(f)
3 f = open('data2.txt', 'wt')
4 print(f)
```

Открытие и закрытие файла

- Файл надо закрывать
- Иначе можно получить неприятности
- Документация утверждает, что может закрыться "автоматически"
- Это не очень корректное утверждение

Открытие и закрытие файла

- Базовый способ закрыть файл - метод close
- Но можно получить исключение
- И не дойти но вызова
- Лучше использовать with ... as

Пример

```
1 f = open('data1.txt', 'rt')
2 f.close()
3 print(f)
4
5 f = open('data2.txt', 'wt')
6 print(f)
7 f.close()
```

Пример

```
1 with open('data1.txt', 'rt') as f:
2     print(f)
3
4 with open('data2.txt', 'wt') as f:
5     f.write('hello\n')
```

Запись в файл

- Метод write и функция print
- Метод write не добавляет перевод строки
- В функции print используем именованный параметр file
- Можно писать в стандартный выход через write
- Используя sys.stdout

Пример

```
1 import sys
2
3 with open('data2.txt', 'wt') as f:
4     f.write('Hello, world\n')
5
6 with open('data3.txt', 'wt') as f:
7     print('Hello, world', file=f)
8
9 sys.stdout.write('Hello, world\n')
```

Особенности национальных алфавитов

- С записью национальных алфавитов возможны сложности
- Или с последующим чтением
- Каждый символ представлен в виде числового кода
- Есть разные стандарты, сопоставляющие символу число
- Называются кодировками
- Исторически первый - ASCII

Особенности национальных аофавитов

- ASCII использует 1 байт, то есть 256 символов
- Задает символы для кодов 0 - 127
- Вторую половину диапазона использовали национальные кодировки
- В каждой стране работали с латиницей и с национальным алфавитом
- Работать с несколькими национальными было сложно

Unicode

- Была придумана универсальная кодировка, Unicode
- Два байта на символ, 65535 кодовых позиций
- Коды 0 - 127 совпадают с ASCII
- Остальные распределены по национальным алфавитам

Новая сложность

- Строки в памяти представлены в Unicode
- Строка 'hello' занимает 10 байтов
- Строка 'вася' - 8 байтов
- Можем узнать код каждого символа
- Функция ord

Пример

```
1 for c in 'hello, world':
2     print(c, ord(c))
3
4 for c in 'однажды в студеную зимнюю пору':
5     print(c, ord(c))
```

Новая сложность

- Теоретически, в файл можно писать Unicode-коды
- А при чтении из преобразовывать в символы
- Есть даже обратная ord фуккция - chr
- Но так мало кто делает

Новая сложность

- Можно записать 'hello' в файл
- Посмотреть на размер файла
- И увидеть, что он - 5, а не 10
- На это есть две причины

Новая сложность

- Историческая: огромное количество программ написано для ASCII
- Их сложно взять и переписать
- Можно записать файл в Unicode
- Но его не прочитают текстовый редактор, компилятор, браузер и т.п.

Новая сложность

- Техническая: Unicode увеличивает размер файлов
- В два раза
- Это обидно
- Потому что в реальности нечасто используются много алфавитов в одном тексте

Новая сложность

- Решение: придумать схему преобразования двухбайтового кода
- В последовательность байтов
- Чтобы у каждого кода была своя последовательность
- По последовательности восстанавливался двухбайтовый код
- Более частные символы представлялись более короткими последовательностями
- Кода 0 - 127 превращались в однобайтовую последовательность без изменений

Новая сложность

- Такая схема преобразования называется encoding
- encoding и кодировка - это разное
- Есть самый популярный encoding с описанными свойствами
- Он называется utf-8
- Обычно он используется по умолчанию

Новая сложность

- Существуют другие
- Есть другие универсальные
- Которые кодируют все Unicode
- Но с другими приоритетами

Новая сложность

- А есть те, которые соотвествуют старым схемам
- ASCII + национальная половина
- Например, cp1251
- В ней представлены тексты, набранные в старых русифицированных Windows

Новая сложность

- Есть параметр encoding
- В open и в write
- Часто его можно не указывать
- И по умолчанию будет использован utf-8
- Можно указать другую, в виде строкового имени
- Python поддерживает много, очень гибок в именах

Примерная схема решения проблем

- Если не хочется вникать в детали, попробуйте указать utf-8 явно
- Проверьте, точно ли вы пишете питоновскую строку
- Варианта "не пишется" с utf-8 быть не должно
- А с другим encoding-ом - может
- Например, если писать китайские символы в cp1251

Примерная схема решения проблем

- Если вы пишете, а кто-то не может прочитать
- Скорее всего ваш encoding не совпадает с ожиданием читателя
- Надо настроить у себя или у читателя
- Если "не читается", скорее всего ваш encoding не совпадает читаемым файлом
- Надо разобраться и добиться соответствия

Чтение из файла

- Метод read читает файл полностью
- Удобно для маленьких файлов
- Опасно для потенциально больших
- Можно потратить много памяти или даже упасть

Пример

```
1 with open('data1.txt', 'rt') as f:
2     s = f.read()
3     print(s[::2])
```

Чтение из файла

- Метод readline читает строку
- Относительно безопасен
- В предположении, что есть переводы строк
- С разумной частотой

Пример

```
1 with open('data1.txt', 'rt') as f:
2     s = f.readline()
3     print(s)
4     s = f.readline()
5     print(s[::2])
```

Чтение из файла

- Файл, открытый на чтение - это коллекция
- Точнее - ленивый генератор
- Тоже "исчерпывается" по мере чтения
- Но можно открыть заново
- Или использовать метод seek

Пример

```
1 def stripped(f):
2     return (s.rstrip() for s in f)
3
4 with open('data1.txt', 'rt') as f:
5     for s in stripped(f):
6         print(s)
```

Форматирование

- Формально - независимая от ввода-вывода тема
- Но часто используется в связке
- Хотим выводим данные в определенном текстовом формате
- Есть текстовая основа
- В нее надо что-то вставить

Мотивирующий пример

- Печатаем фрагменты пособия по математике
- Нужно печатать простые уравнения
- Типа $3x + 10 = 5$
- Определяем функцию

Первый вариант

```
1 def equation(a, b, c):
2     return str(a) + 'x + ' + str(b) + ' = ' + str(c)
3
4 print(equation(2, 3, 4))
```

Альтернатива

- Напишем строчку, в которой будут спецсимволы
- Обозначающие что-то, что можно подставить
- И рядышком передадим конкретные значения для подстановки
- Посмотрим на примере

Первый вариант

```
1 def equation(a, b, c):
2     return '%dx + %d = %d' % (a, b, c)
3
4 print(equation(2, 3, 4))
```

Детали

- Буква после процента обозначает тип данных
- d - число
- Между процентом и обозначением типа могут идти уточнения формата
- Например, `%-10d` - используем 10 позиций, подравниваем влево
- Удобно для таблиц

Другие варианты

- Метод format
- Позволяет использовать именованные позиции
- Удобно для повторяющихся вставок
- 'Price is %{price:.2f}'.format(price = 49)

Другие варианты

- Форматная строка
- Более компактная запись
- Выражение помещается в точку использования
- f'Price is {price:.2f}'

Когда что использовать

- При прочих равных - форматная строка
- Два других - при наличии обстоятельств, делающих их удобными
- Если уже есть готовый кортеж, можно рассмотреть вариант %
- Если значения повторяются или есть готовый кортеж, можно рассмотреть вариант format