Действия с файлами

Общение с файлами идет не напрямую, а через ОС, общепринятая последовательность операций с файлом следующая:

1. Открыть файл в различных режимах для чтения или записи, в бинарном или текстовом режиме.
2. Поработать с информацией из файла, используя, в том числе, операции чтения/записи.
3. Закрыть файл.

После успешного завершения python-программы все файлы закрываются автоматически. Но важно все равно закрывать файл, как только он перестает быть вам нужным. Это поможет избежать конфликтов совместного доступа или риска получить неконсистентный (испорченный) файл, если программа завершится аварийно.

Из всех параметров при указании файла основные – следующие:

- file – имя открываемого файла. Оно может быть задано с использованием и относительных и абсолютных путей
- mode – режим открытия. r (или rt) – чтение в текстовом режиме, rb – чтение в бинарном режиме, w – запись, wb – запись в бинарном режиме. По умолчанию файл открывается в режиме r
- encoding – если работа идет в текстовом режиме, Python должен получить имя кодировки, чтобы корректно работать с данными. Независимо от кодировки файла, в результате чтения будет возвращаться стандартная юникод-строка Python

Открытие файла

Полное описание параметров в документации (здесь ссылка!) по функции open.

В путях до файла используются прямые слеши (/).

Теперь откроем файл в бинарном режиме и прочитаем первые 20 байт:

```
f =open("files/bibl.txt", mode="rb")
f.read(20)
```

Перед строкой стоит модификатор b. Он говорит о том, что перед нами поток байт. Поток байт в языке Python представляется классом bytes. Если вы открываете файл для чтения в бинарном режиме, результат метода read() имеет тип bytes.

```
data=open("files/bibl.txt", mode="rb").read()
print(type(data))
print(data[19])
```

Объект, который возвращает нам функция open, ассоциирован (связан) с открытым файлом и содержит следующие поля и методы:

['__class__', '__del__', '__delattr__', '__dict__', '__dir__', '__doc__', '__enter__', '__eq__', '__exit__', '__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__lt__', '__ne__', '__new__', '__next__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_checkClosed', '_checkReadable', '_checkSeekable', '_checkWritable', '_dealloc_warn', '_finalizing', 'close', 'closed', 'detach', 'fileno', 'flush', 'isatty', 'mode', 'name', 'peek' 'raw', 'read', 'read1', 'readable', 'readinto', 'readinto1', 'readline', 'readlines', 'seek', 'seekable', 'tell', 'truncate', 'writable', 'write', 'writelines']

Пример:

```
f =open("files/bibl.txt", mode="rb")
print(f.name)
print(f.readable())
print(f.tell())
print(f.writable())
print(f.seekable())
print(f.mode)
```

Теперь откроем файл в текстовом режиме и проверим работу некоторых методов:

```
f =open("files/bibl.txt", encoding="utf-8")
print(f.read(100))
print(f.tell())
print(f.seek(1245))
print(f.read(100))
print(f.tell())
```

- Если мы открываем файл в текстовом режиме, чтение происходит посимвольно (символ может занимать физически 1, 2, 4 и даже 6 байт в некоторых случаях в зависимости от кодировки). За одну операцию можно прочитать различное количество символов
- Если мы используем бинарный режим, чтение осуществляется побайтно и за одну операцию можно прочитать сразу несколько байт
- Метод tell возвращает позицию в байтах от начала файла, а метод seek изменяет ее (перематывает) на заданную позицию. Использование seek с текстовыми файлами затруднено из-за несоответствия номера байта и номера символа.

Запись в файл

Для записи в файл также есть два режима: ‘w’ (если файл существовал, его содержимое будет потеряно) и ‘a’ – запись идет в конец файла.

Один из способов записать информацию в файл – метод write. Если мы хотим сделать запись в середину файла, должны сначала спозиционироваться на место предполагаемой записи (метод seek), а уже потом записывать (метод write). Метод write возвращает количество записанных символов.

```
f =open("files/bibl.txt",'w')
print(f.write('123\n456'))
print(f.seek(3))
print(f.write('34352'))
f.close()
f =open("files/bibl.txt",'r')
print(f.read())
f.close()
```

Второй способ записи в файл – стандартная функция print. Для этого применяется именованный параметр file.

Например:

```
From math import sin
f =open("files/didgital.txt", 'w')
for i in range(10):
print("%0.2f"% sin(i), file=f)
f.close()
```

Закрытие файлов

Операционная система контролирует доступ к файлам. Если какая-то программа открыла файл для записи, все попытки любых других программ изменить содержание файла заблокируются для сохранения целостности.

Поэтому, после того как работа с файлом закончена, файл необходимо закрыть методом close, что мы делали практически во всех примерах.

```
f.close()
```

После завершения программы все файлы, которая она использовала в своей работе, автоматически закроются.

Для того чтобы файл закрывался автоматически даже в случае ошибок во время выполнения других операций, в языке Python есть блок with – он дает закрыть файл после выхода из блока.

Его синтаксис такой:

```
withopen('files/bibl.txt','rt')as f:
read_data=f.read()
print(read_data[:100])
print(f.closed)
```

Дополнительно ознакомиться с информацией о работе с файлами можно по ссылкам:

https://pythonworld.ru/tipy-dannyx-v-python/fajly-rabota-s-fajlami.html
https://www.easycoding.org/2017/01/23/schityvaem-chislovye-dannye-iz-fajla-na-python.html