ILE3-010 스트림

Stream

데이터의 흐름을 정의

python의 `IOBase class`를 이용하여 정의
io --- 스트림 작업을 위한 핵심 도구 — 파이썬 설명서 주석판

종류

1. **텍스트 I/O (Text I/O)**
   문자열 데이터에 대한 Stream, 인코딩 및 줄바꿈 문자에 대한 인식을 수행 함
   파일에 대한 읽고 쓰기를 위한 `open()` 함수가 대표적
2. **바이너리 I/O (Binary I/O)**
   바이너리 데이터에 대한 Stream
3. **원시 I/O (Raw I/O)**
   저수준의 I/O Stream, 잘 사용하지 않음

Stream이 지니는 함수

1. **read()**
   Binary 데이터 읽기
2. **write()**
   Binary 데이터 쓰기
3. **close()**
   해당 Stream을 종료
4. **seek()**
   지정한 위치로 Cursor를 이동
5. **tell()**
   현재 Cursor의 위치 확인

Stream의 사용

이미 파일에 대한 읽기/쓰기 시 활용

StringIO 예제

Memory Stream

Memory에 있는 문자열을 File처럼 사용하는 방법

```
import io

raw = '가나다라마\nabcdefg\n(❁´◡`❁)\n'

f = io.StringIO(raw)

line = f.readline()
while line is not None:
    print(line)
```

```
    f.seek(0)

    line = f.readline()

```

BinaryIO 예제

```
00000000  01 00 00 00 02 00 00 00 14 28 E2 9D 81 C2 B4 E2  .........(â..Â´â
00000010  97 A1 60 E2 9D 81 29                             —¡`â..)
```

Read

```
f = open('test.bin', 'rb')

one_raw = f.read(4)
two_raw = f.read(4)
txt_raw = f.read()

one = int.from_bytes(one_raw, byteorder='little')
two = int.from_bytes(two_raw, byteorder='little')
txt = str(txt_raw, encoding='utf-8')

print(one)
print(two)
print(txt)

f.close()

```

```
1
2
(❁´◡`❁)

```

Write

```
f = open('test.bin', 'wb')

one = (1).to_bytes(4, byteorder='little')
two = (2).to_bytes(4, byteorder='little')

f.write(one)
f.write(two)
f.write(b'\x14')
f.write(('(❁´◡`❁)').encode(encoding='utf-8'))

f.close()

```

```
00000000  01 00 00 00 02 00 00 00 14 28 E2 9D 81 C2 B4 E2  .........(â..Â´â
00000010  97 A1 60 E2 9D 81 29                             —¡`â..)
```

Memory Stream 사용

Memory에 있는 Binary 데이터를 File처럼 활용

```

```

```
raw = b'\x01\x00\x00\x00\x02\x00\x00\x00\x28\xE2\x9D\x81\xC2\xB4\xE2\x97\xA1\x60\xE2\x9D\x81\x29'
f = io.BytesIO(raw)

one_raw = f.read(4)
two_raw = f.read(4)
txt_raw = f.read()

one = int.from_bytes(one_raw, byteorder='little')
two = int.from_bytes(two_raw, byteorder='little')
txt = str(txt_raw, encoding='utf-8')

print(one)
print(two)
print(txt)

f.close()

```

과제

바이너리 데이터의 분해 및 Zip 파일 압축 해제

1. 아래의 입력 데이터 형태를 참조하여 압축 전/후 크기를 화면에 출력
2. 압축 파일을 해제하여 파일 저장

입력 데이터 형태

1. **0~3byte (4byte)**
   압축 후 크기 (int)
2. **4~7byte (4byte)**
   압축 전 크기 (int)
3. **이후 데이터**
   ZipFile

DBF 파일 읽기

DBF 파일을 읽어서 화면에 출력하는 프로그램 제작

Layout of file header in dBase level 5

Byte	Contents	Meaning
0	1 byte	Valid dBASE for DOS file; bits 0–2 indicate version number, bit 3 indicates the presence of a dBASE for DOS memo file, bits 4–6 indicate the presence of a SQL table, bit 7 indicates the presence of any memo file (either dBASE m PLUS or dBASE for DOS)
1–3	3 bytes	Date of last update; formatted as YYMMDD (with YY being the number of years since 1900)
4–7	32-bit number	Number of records in the database file
8–9	16-bit number	Number of bytes in the header
10–11	16-bit number	Number of bytes in the record
12–13	2 bytes	Reserved; fill with 0
14	1 byte	Flag indicating incomplete transaction[note 1]
15	1 byte	Encryption flag[note 2]
16–27	12 bytes	Reserved for dBASE for DOS in a multi-user environment
28	1 byte	Production .mdx file flag; 1 if there is a production .mdx file, 0 if not
29	1 byte	Language driver ID
30–31	2 bytes	Reserved; fill with 0
32–*n* [note 3][note 4]	32 bytes each	array of field descriptors (see below for layout of descriptors)
n + 1	1 byte	0x0D as the field descriptor array terminator

Field descriptor array [edit]

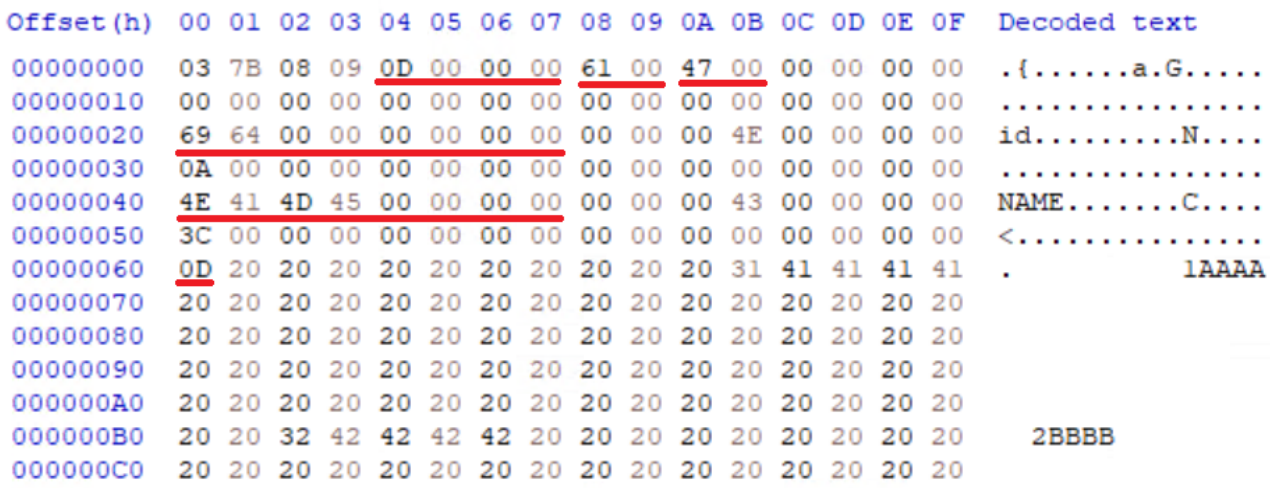
Layout of field descriptors in dBase level 5 (used inside the file header)

Byte	Contents	Meaning
0–10	11 bytes	Field name in ASCII (zero-filled)
11	1 byte	Field type. Allowed values: `C`, `D`, `F`, `L`, `M`, or `N` (see next table for meanings)
12–15	4 bytes	Reserved
16	1 byte	Field length in binary (maximum 254 (0xFE)).
17	1 byte	Field decimal count in binary
18–19	2 bytes	Work area ID
20	1 byte	Example
21–30	10 bytes	Reserved
31	1 byte	Production MDX field flag; 1 if field has an index tag in the production MDX file, 0 if not

Database records [edit]

Each record begins with a 1-byte "deletion" flag. The byte's value is a space (0x20), if the record is active, or an asterisk (0x2A), if the record is deleted. Fields are packed into records without field separators or record terminators.

All field data is ASCII. Depending on the field's type, the application imposes further restrictions:



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	03	7B	08	09	0D	00	00	00	61	00	47	00	00	00	00	00	.{.....a.G.....
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000020	69	64	00	00	00	00	00	00	00	00	00	4E	00	00	00	00	id.....N....
00000030	0A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000040	4E	41	4D	45	00	00	00	00	00	00	00	43	00	00	00	00	NAME.....C....
00000050	3C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	<.....
00000060	0D	20	20	20	20	20	20	20	20	20	20	31	41	41	41	41	. 1AAAA
00000070	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000080	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000090	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
000000A0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
000000B0	20	20	32	42	42	42	42	20	20	20	20	20	20	20	20	20	2BBBB
000000C0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	

출력 예제

```
필드 정보
- id
- NAME

Row 1
- 1
- AAAA

Row 2
- 2
- BBBB
....
```